Half-Edge Structure for Triangle Meshes

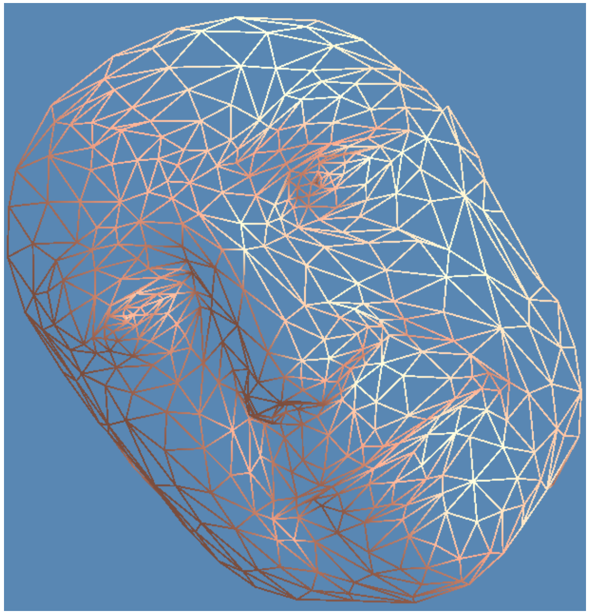
Half-Edge Data Structure

- (What?) A common way to represent triangular mesh for geometric processing
  - we focus on triangle-mesh here (it works for general polygonal mesh).
  - 3D analogy: half-face data structure for tetrahedral mesh
- (Why?) Effective for maintaining incidence information of vertices
  - Efficient local traversal
  - Low spatial cost
  - Supporting dynamic local updates/manipulations (edge collapse, vertex split, etc.)

Questions of mesh rep.?

- Remember when we store a triangle mesh by
  →A vertex table (geometry) + A facet table (connectivity)
- Enough to preserve all the information, but how will you use this representation to solve the following questions and how efficient your algorithm can be?

▪Whether a given vertex is on the boundary?
▪What are the 1-ring neighboring vertices of a vertex?
▪How to traverse from one vertex to another vertex?
▪…
▪…



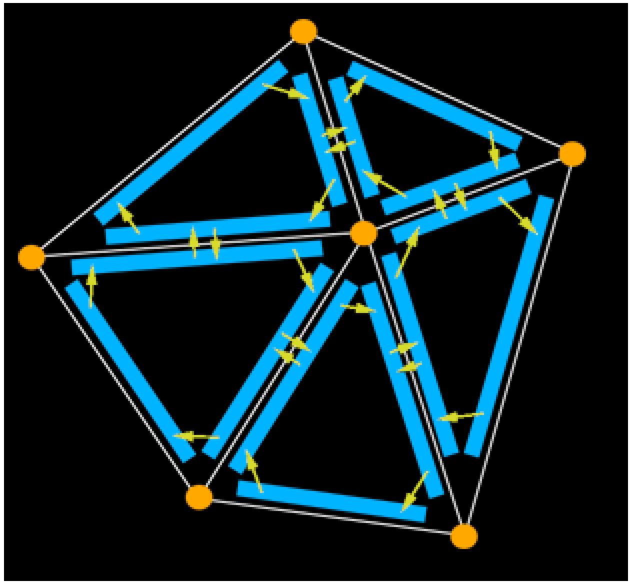
We need to answer these questions when we manipulate meshes
(e.g. computing surface normal, detecting how curved a region is…)

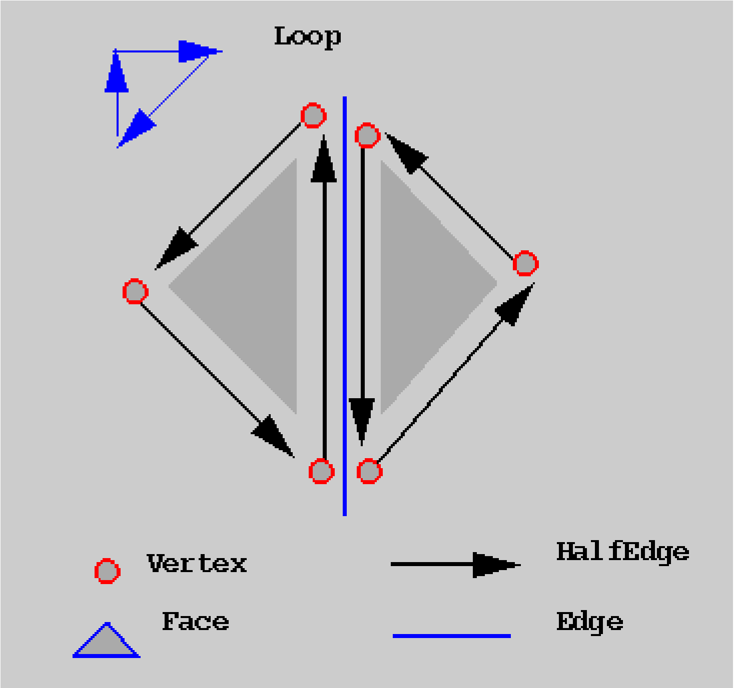
- seems difficult by just looking at those two tables
- Need a more efficient representation

Half-Edge Data Structure

Looking at a triangle mesh:

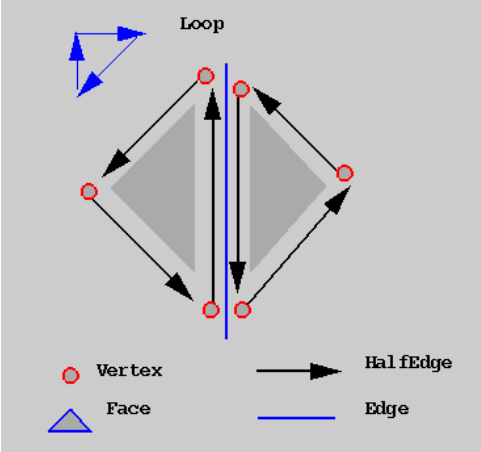
- ❑2 vertices share an edge, 2 faces share an edge
- ❑each face has 3 vertices and 3 edges…
- ➤We can store all incidence information and build a big network
  - ❑But a vertex can have many neighboring vertices, edges, and faces
- ➤Storing "half-edges" is simply enough
- ❑Each edge has 2 half-edges (a boundary edge only has 1)





Half-Edge Data Structure (cont.)

- For each **edge**:
  - it has 2 half-edges (the boundary edge has 1)
  - they are called *twins* to each other
- For each **half-edge**:
  - bounds 1 face and 1 edge → a face pointer, an edge pointer, respectively
  - has one origin, and one target vertex → a vertex pointer (for the target)
  - ➢ To be able to walk around a face:
  - it has a pointer to the next half-edge
  - also a pointer to the previous half-edge
- For each **face**:
  - 3 half-edges belong to it
  - To simply access all its incident elements → Only need a pointer to a (random) half-edge
- For each **vertex**
  - A pointer to an arbitrary half-edge that has it as the target
  - Record its 3D coordinates (its geometric location)



Note the directions of those half-edges bounding a face.

Linear Storage! Constant time Local Traversal!

Half-Edge Data Structure (example)

→ Containers to store primitives:

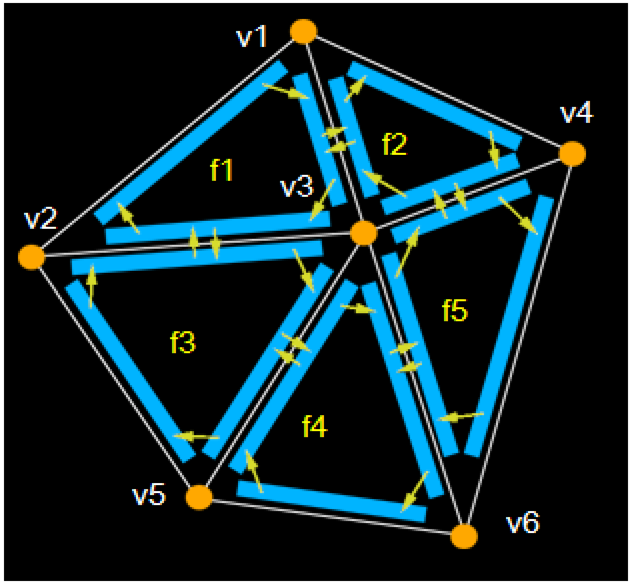
The Vertex Container	v1 … v6
~~The Half Edge Container~~	~~[v1,v2], [v2, v3], [v3, v1], [v1, v3], [v3, v4], …~~
~~The Edge Container~~	~~[v1,v3], [v1,v2], [v2,v3], [v1,v4], [v3,v4], …~~
The Face Container	f1[v1,v2,v3] … f5[v4,v3,v6]

Need only one

Remember the Half-Edge direction: [v1, v2] or [v2, v1] around each face?

Should be consistent:

e.g. CCW in our configuration (right hand rule)



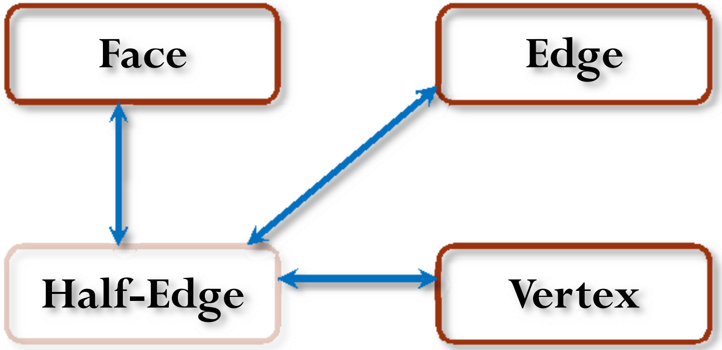
Note: the container could be array, list, binary search tree…
(it depends, but usually List is good enough!)

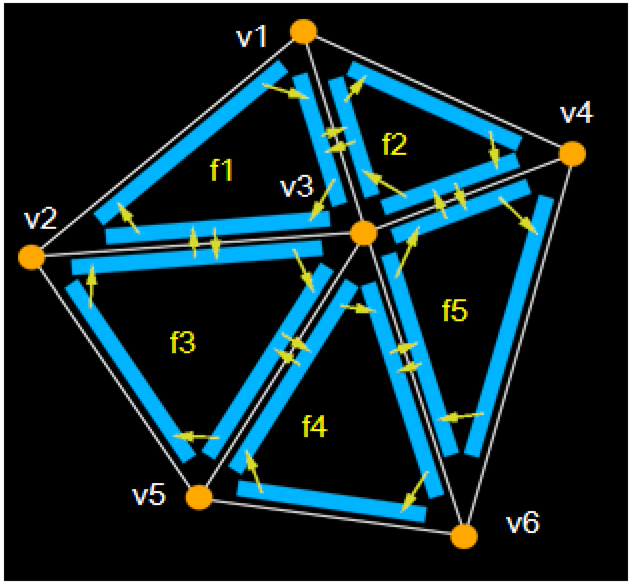
Half-Edge Data Structure (example)

→ Containers to store primitives:

The Vertex Container	v1 … v6
~~The Half Edge Container~~	~~[v1,v2], [v2, v3], [v3, v1], [v1, v3], [v3, v4], …~~
The Edge Container	[v1,v3], [v1,v2], [v2,v3], [v1,v4], [v3,v4], …
The Face Container	f1[v1,v2,v3] … f5[v4,v3,v6]

→ Relationship between primitives:

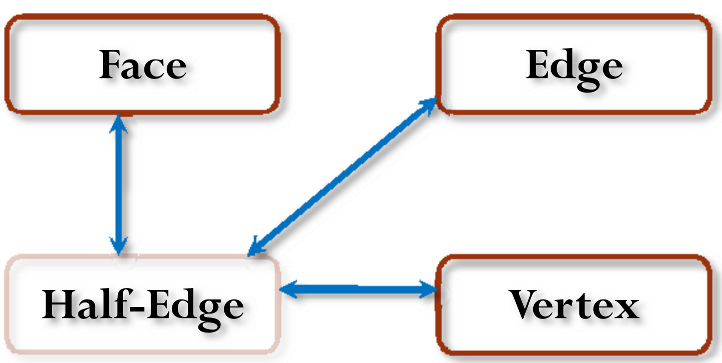


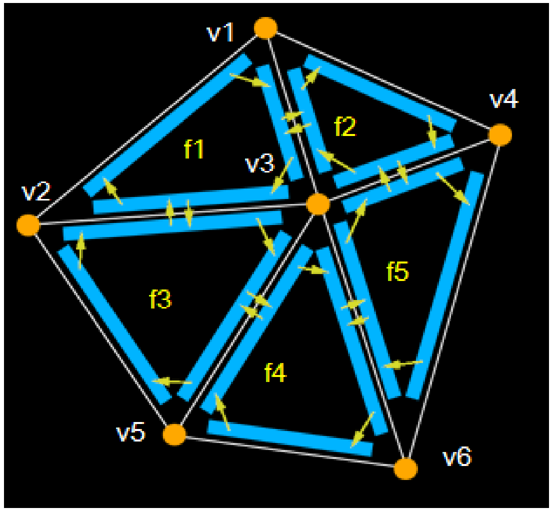


Using Half-Edge Data Structure

Examples:

1. How to check whether a vertex/edge/face is on the boundary?
   - ✓ Simply check whether an edge has one half-edge
2. How to find the one-ring neighboring vertices of a vertex v?
   - ✓ Get any half-edge targeting v, iteratively get "next()", then "twin()"
3. How to travel along the boundary?
   - ✓ ...get a boundary vertex and its most CLW outwards halfedge, iteratively do next(), twin(), next()...
4. Some other operations such as subdivision/simplification...?





Using Half-Edge Data Structure

Examples:

1. How to check whether a vertex/edge/face is on the boundary?

<u>Example codes to print all boundary edges of a given mesh "cmesh"</u>

```
Mesh * cmesh;
…
For  (MeshEdgeIterator eit(cmesh); !eit.end(); ++eit){
    Edge * e = *eit;
    if (e→he(1)==NULL)
     {    //this is a boundary edge, output it
        Halfedge * he = *e→he(0);
        std::cout << "[" << he→source()→id() << " ," << 
                         he→target()→id() << "]" << std::endl;
     }
}
```

Using Half-Edge Data Structure

Examples:

2. How to find the one-ring neighboring vertices of a vertex v?

Example codes to print one-ring vertex of a given vertex "cv"

(Method 1: Try to traverse using the half-edge data structure)

```
Vertex * cv;
…
Halfedge * he0 = cv→he();
Halfedge * he = he0;
Do {
   Vertex * v = he→source();
   std::cout << v→id() << std::endl;
   he = he→he_next();
   he = he→he_twin();
}while (he!=he0);
```

Using Half-Edge Data Structure

Examples:

2. How to find the one-ring neighboring vertices of a vertex v?

Example codes to print one-ring vertex of a given vertex "cv"

(Method 2: Using the "iterator" class, when you have the mesh library)

```
Vertex * cv;
…
For  (VertexVertexIterator vvit(cv); !vvit.end(); ++vvit){
     Vertex * v = * vvit;
     std::cout << v→id() << std::endl;
}
```

Using Half-Edge Data Structure

Examples: 3. How to travel along the boundary?

<u>Example codes to traverse the boundary (given a boundary halfedge, go on and collect all following halfedges in the same loop)</u>

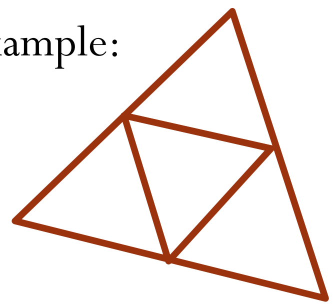
```
Halfedge * he0;   //suppose it is a boundary half-edge
…
Halfedge * he = he0;
Do {
   std::cout << he << std::endl;
   Halfedge * he1= he;
   Do {
            he1 = he1→he_next();
            he = he1;
            he1 = he1→he_twin();
    } while (he1 != NULL)
}while (he!=he0);
```

Using Half-Edge Data Structure

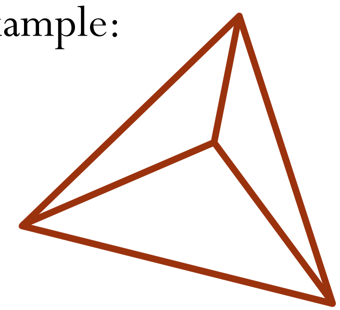
Examples:

4. Subdivision:

One example:



Another example:



…

<u>to split a face (type 2)</u>

Face * f0;   //the face we want to split

…

-Create a new vertex nv ← the mass center of f0

-Create three new edges, six new halfedges

-Update half-edges, forming three cycles

-Create three new faces, link edges, halfedges accordingly

-Delete the original face

Using Half-Edge Data Structure

Examples:

4. Simplification:

In one week

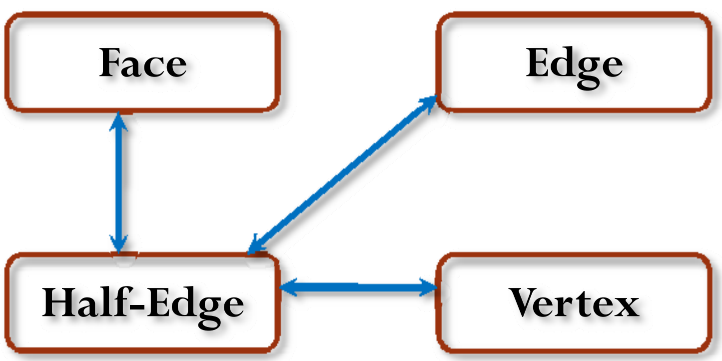
Resources for "Half-Edge" Data Structure

To get better understanding about it, you can

1) Download and read codes from: http://www.ece.lsu.edu/xinli/teaching/MeshLib.zip
2) In Computational Geometry, it is well known as "doubly-connected edge list" structure (extendible to general polygonal mesh)
   Comp. Geom. book: "Computational Geometry Algorithms and Applications", by M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Springer-Verlag.

To better understand the codes:

- Go through the "copyTo()" method in the class "Mesh", see what points need to be filled in for each element;
- Go through the "read()" method in the class "Mesh", see how we build up half-edge structure from the vertex table + face table;
- Go through "iterators.h", see what iterator you can use to help you traverse around



Some other issues

for people don't know how to program using half-edge data structure and OpenGL, but want to work on 3D shapes and meshes first:

- Store meshes with 2 tables, use viewers/programs written by others as a black box to visualize or even edit the model...
- Before we can design a fully robust/powerful GUI for editing and visualization (which we may keep doing through the semester), here are something for us to play with triangle meshes and 3D shapes:
  - Some mesh data (.m format) can be downloaded at: http://www.ece.lsu.edu/xinli/teaching/meshdata1.zip
  - A small viewer "G3dOGL.exe" (for .m format mesh) can be downloaded at: http://www.ece.lsu.edu/xinli/Tools/G3dOGL.exe
  - Many 3D triangle mesh models online (but in different formats):
    - Stanford 3D Scanning Repository: http://graphics.stanford.edu/data/3Dscanrep/
    - Aim@Shape Repository http://shapes.aim-at-shape.net/index.php