Lecture 2
Triangle Mesh and its Data Structure

Overview

- The data structure to represent surfaces → efficiency and memory consumption of the geometric modeling algorithms
- We will
  - Give a brief overview of the various data structures for mesh representations in the literature
  - Elaborate the half-edge data structure, which is commonly used in modeling/processing 3D data

Surfaces defined by triangle meshes

- A surface (2-manifold, two-dimensional manifold): a continuous topological space with infinitely many points, where each point has a local neighborhood homeomorphic to a 2-dimensional Euclidean space E^2
- A triangle mesh is its approximation:
  - We use a finite number of vertices and triangles
  - Simply a collection of these triangles without any mathematical structure
  - But it defines a piecewise linear representation with quadratic approximation

Piecewise Linear Representation by Barycentric Parameterization

- Every triangle T is determined by its three vertices $\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3$, we denote it as $T=[\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3]$
- Any point $\mathbf{p}$ in the interior of T can be represented uniquely using a barycentric combination of the corner points:

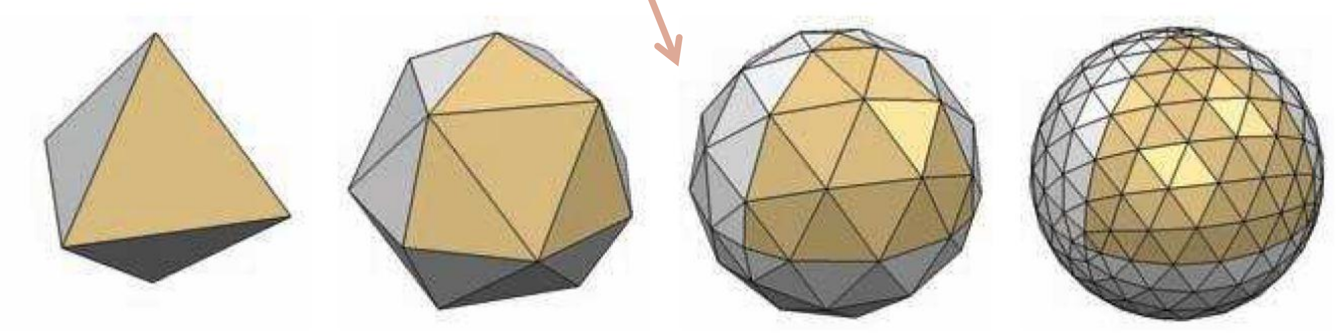
$$\mathbf{p}= a\mathbf{v}_1+b\mathbf{v}_2+c\mathbf{v}_3 ,$$

with $a+b+c=1,\ a,b,c >= 0$.

Therefore, based on this per-triangle mapping, a 2D parameterization can be defined f: $\boldsymbol{R}^2 \rightarrow \boldsymbol{R}^3$, to represent the entire continuous surface approximated by this triangle mesh (details and algorithms will be discussed later, on how to actually construct this 2D layout → parameterization)

Smooth Surfaces can be Well Approximated

- A sufficiently smooth surface is approximated by a triangle mesh (piecewise linear function)
  - approximation error $O(h^2)$, with h = maximum edge length
  - Error Reduced by a factor of $\frac{1}{4}$: if we evenly split all edges
  - A simple subdivision scheme for this: cutting each triangle into 4
    - Face number: $F \rightarrow 4F$
    - Approximation error : inversely proportional to F
  - The actual approximation error depends on the 2nd order Taylor expansion, i.e., on the curvature of the underlying smooth surface
  - But roughly: a sufficient approximation can be obtained using moderate mesh complexity (you may want to adaptively adjust vertex density according to surface curvature, will be discussed later in meshing sessions)



Geometric and Topological Components of a Triangle Mesh

- A triangle mesh has two components:
  - Geometric components: vertex table → positions of points
  - Topological components: face table → the graph encoding the connectivity
- What if you fix the topology (face table), and change the positions of vertices?
  - → a continuously deforming surface
- What if you only have the positions of sampled points, but don't know the connectivity?
  - → could be complicated… different connectivity indicates different shapes

Face-based Data Structure

- A simplest way to represent a surface mesh
  - Storing a set of faces represented by their vertex positions

- also called **"triangle soup"**
- used in the stereolithography (**STL**) format
- if using x (e.g. 32) bits to represent a vertex coordinate
  - Each triangle needs 3*3*x/8 = 36 bytes
- No connectivity info stored
- Inefficient for many geometric computing: e.g. traversing local adjacency information
- Vertex positions replicated as many times as the degree of the vertices

Triangles		
x_{11} y_{11} z_{11}	x_{12} y_{12} z_{12}	x_{13} y_{13} z_{13}
x_{21} y_{21} z_{21}	x_{22} y_{22} z_{22}	x_{23} y_{23} z_{23}
...	...	...
...	...	...
...	...	...
x_{F1} y_{F1} z_{F1}	x_{F2} y_{F2} z_{F2}	x_{F3} y_{F3} z_{F3}

Face-based Data Structure (2)

- An improved face-based data structure:
  - To prevent the redundancy by indexed face set
  - Stores an array of vertices
  - Stores faces as sets of indices into this array
- Simple and efficient in storage
- Widely used in many formats such as **OFF**, **OBJ**, **VRML**, as well as our .**M** files
- if using x (e.g. 32) bits to represent a vertex coordinate and face indices
  - Each vertex requires 3*x/8 = 12 bytes
  - Each triangle needs 3*x/8 = 12 bytes
  - (Roughly F=2V, by Euler formula)
  - So on average: 18 bytes / triangle
  - Only a half storage space
- No connectivity info stored
- <u>Inefficient</u> for many geometric computing: e.g. traversing local adjacency information

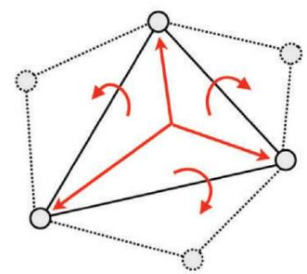
Vertices
x_1 y_1 z_1
. . .
x_V y_V z_V

Triangles
i_{11} i_{12} i_{13}
. . .
. . .
. . .
. . .
i_{F1} i_{F2} i_{F3}

What Does Geometric Computing Need?

- Access to individual elements (vertices edges, and faces): enumeration of all elements
- Local traversal, e.g.:
  - What are the edges in a given face;
  - What are the vertices in a given face or edge;
  - What are incident faces of a given edge;
  - What are incident faces or edges of a given vertex; ...
- Can you develop efficient algorithms to do these using the previous face-based data structure?

➢ An improved **face-based data structure** for efficient local traversal:
  - For each face: store references to its 3 vertices + references to its neighboring triangles
  - For each vertex: store a reference to its neighboring triangle + 3 coordinates
  - Used in **CGAL** for representing 2D Triangulation, 32 bytes / triangle (google **CGAL**)
  - Enumerating the one-ring of a vertex is not easy
  - Not easily extendable to general/mixed polygonal meshes

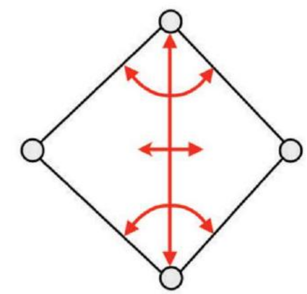


Vertex	
`Point`	`position`
`FaceRef`	`face`

Face	
`VertexRef`	`vertex[3]`
`FaceRef`	`neighbor[3]`

Edge-based Data Structure

- A more generally used data structure, since the connectivity is a graph, directly relates to the mesh edges
- Many well known methods: winged-edge [Baumgart 72], quad-edge [Guibas and Stolfi 85], and variants [O'Rourke 94]
- An example: Winged-edge structure
  - Each edge stores references to its endpoint vertices + two incident faces + next and previous edge within the left and right faces
  - Each vertex stores a reference to one of its incident edges
  - Each face stores a reference to one of its incident edges
  - 60 bytes / triangle
- Still not easy to traversing the one-ring (e.g. to traverse the one-ring of a vertex v, how do you know if it is the first or second vertex of an edge?)



Vertex	
Point	position
EdgeRef	edge

Face	
EdgeRef	edge

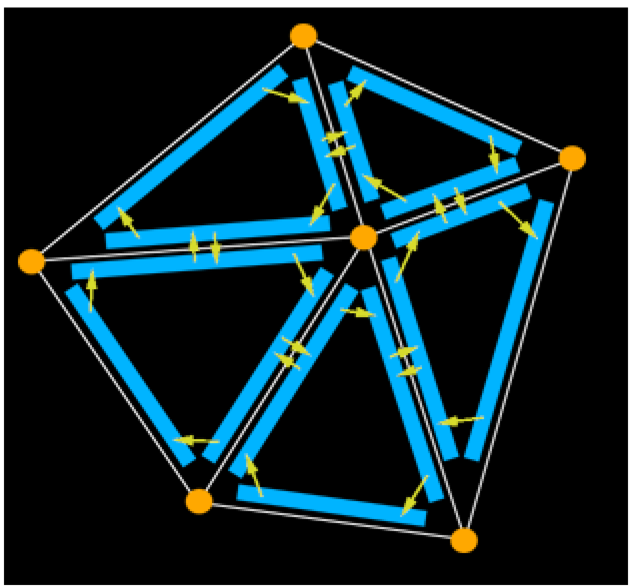
Edge	
VertexRef	vertex[2]
FaceRef	face[2]
EdgeRef	next[2]
EdgeRef	prev[2]

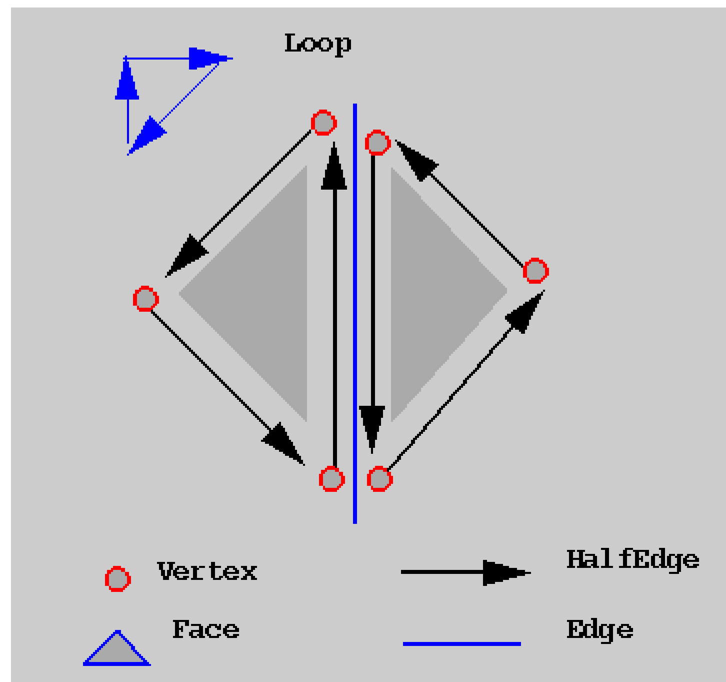
Half-Edge Data Structure

- (What?) A common way to represent triangular mesh for geometric processing
  - We first focus on triangle-mesh, (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
  - Relatively low spatial cost
  - Supporting dynamic local updates/manipulations (edge collapse, vertex split, etc.)
- (Resources?) Codes are provided on the course website. After the class, please go through them carefully, we will work on it during the whole semester.

Half-Edge Data Structure (cont.)

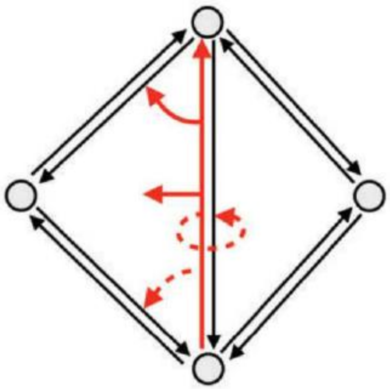
- 2 vertices share an edge, 2 faces share an edge
- Each face has 3 vertices,
- → To store all adjacency information on half-edges
  - Each edge has 2 half-edges (the boundary edge has 1)





Half-Edge Data Structure (cont.)

- Halfedges are oriented consistently in counterclockwise order around each face
- Each halfedge designates a unique corner on each face (can be used to store texture coordinates, later in texture mapping)



- For each halfedge, we store:
  - the vertex it points to (its target);
  - its adjacent face (the face this halfedge locates);
  - the next halfedge of the face;
  - the previous halfedge in the face;
  - its twin halfedge;
- For each vertex: store one of its incident incoming halfedges
- For each face: store one of its halfedges
- For each edge: store its two halfedges

- # of halfedges H is about 6 times of V :
  - → 72 bytes /triangle

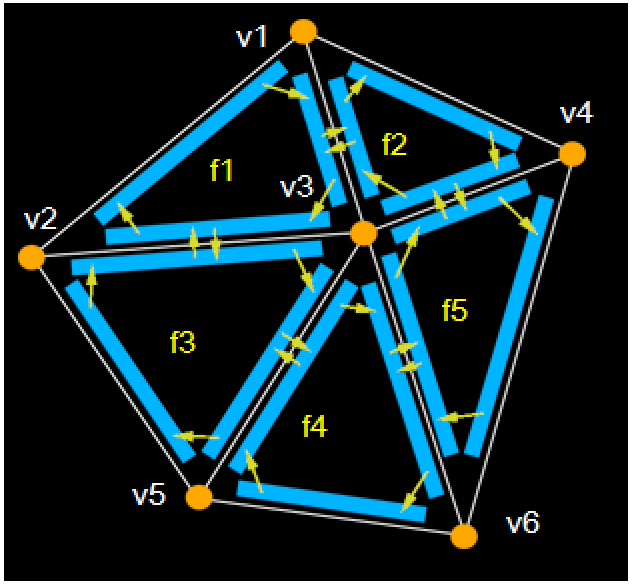
Half-Edge Data Structure (example)

1). Containers 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]

Half-Edge: [v1, v2] or [v2, v1] ?

Should be consistent: *e.g. CCW in our configuration*



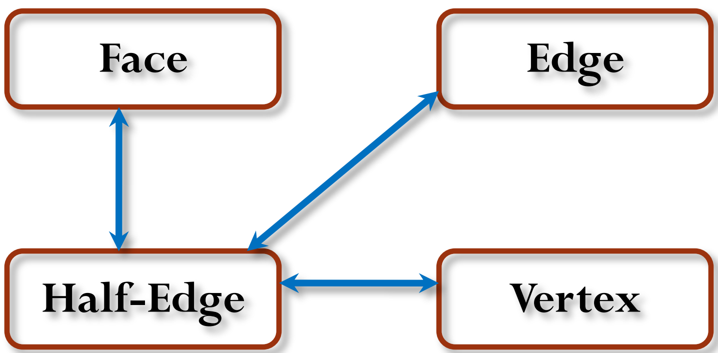
Note*: the container could be array, list, binary search tree…
(it depends, our sample codes used list)

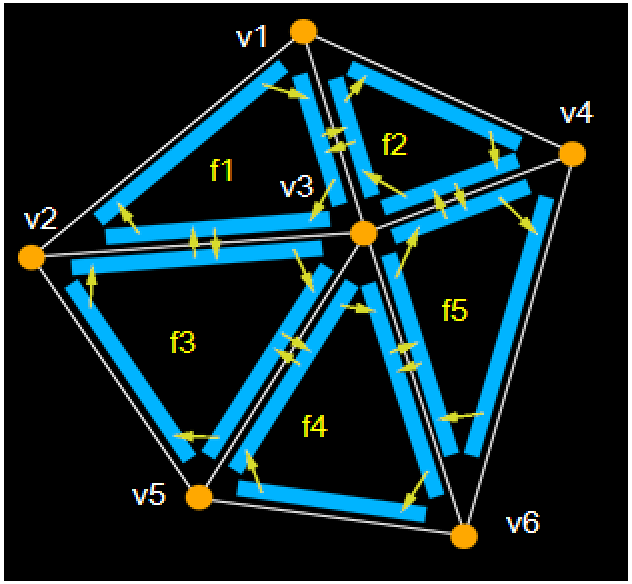
Half-Edge Data Structure (example)

1). Containers 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]

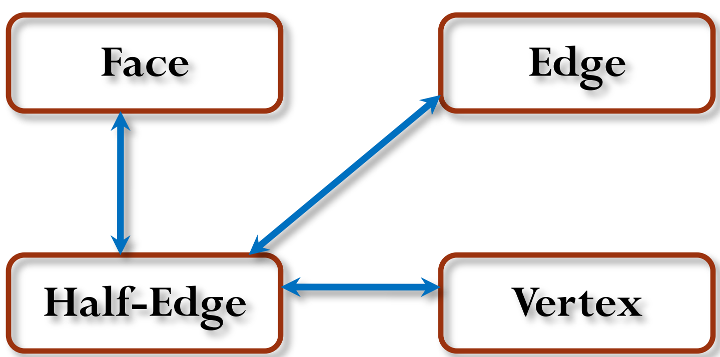
2). Relationship between primitives:

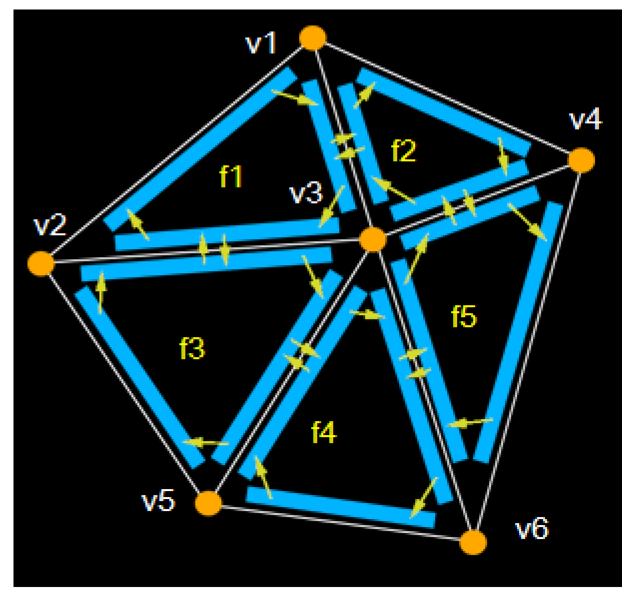




Using Half-Edge Data Structure

1. How to check whether a vertex/edge/face is on the boundary?
2. How to track the boundary?
3. How to find your one-ring neighbor?
4. How to do subdivision/simplification…?

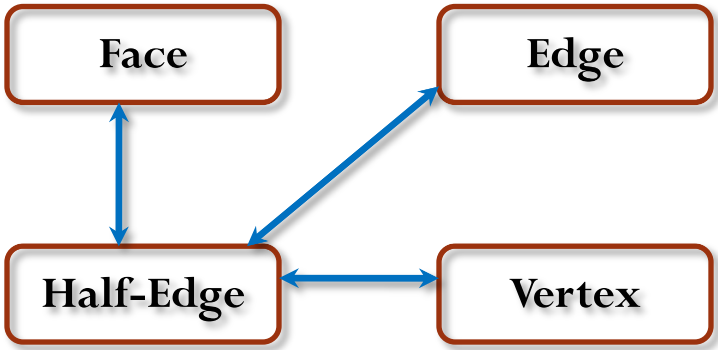


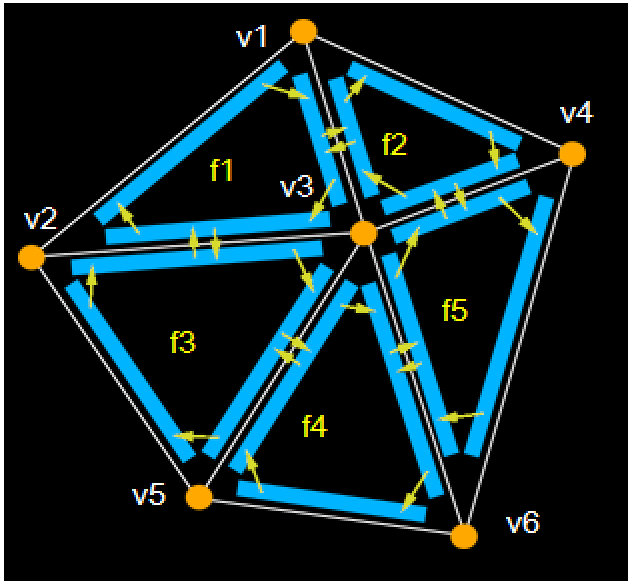


Half-Edge Data Structure (cont.)

Warm-up Assignment:
Compile and run the "meshlib" codes; use it to load a mesh

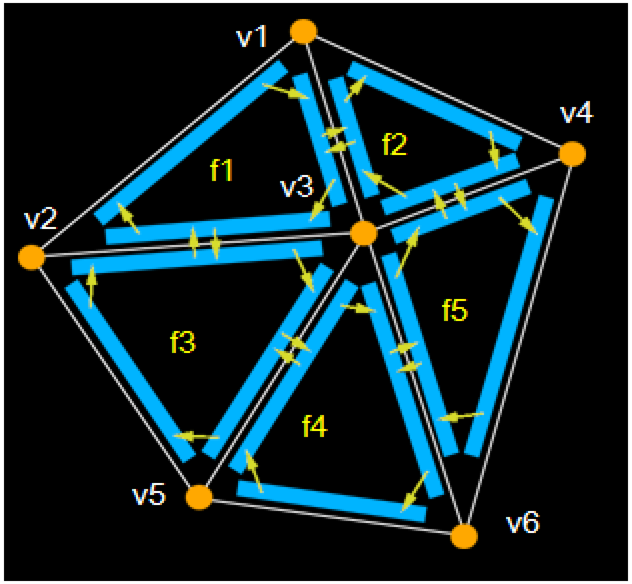
1) Go through "iterators.h" and "mesh.h", to see how you can traverse global/local elements.
2) Go through "read()" method, to see how this structure is built up.





Half-Edge Data Structure (cont.)

Questions about Half-Edge Data Structure, or the assignment?



Some 3D Models in Polygonal Meshes

- Before we can design a fully robust/powerful GUI and visualization system (which you may keep doing through the semester), here are 3D shapes for you to play a little bit with :
- 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 (you can drag your downloaded ".m" file into it directly)

- Many 3D shapes/data online (but in various formats):
- Stanford 3D Scanning Repository: http://graphics.stanford.edu/data/3Dscanrep/
- Aim@Shape Repository: http://shapes.aim-at-shape.net/index.php
- Google 3D warehouse