Lecture 2
Half-Edge Structure for Triangle Meshes

Xin (Shane) Li (xinli@lsu.edu)

Course Email: ee4700fall2009@gmail.com

Lectures: Tu Th 12:10pm – 1:30pm
2150 Patrick Taylor Hall

Office Hours: Tu Th 9:30am – 11:00am
313 Electrical Engineering Building
Louisiana State University

http://www.ece.lsu.edu/xinli/teaching/EE4700Fall2009.htm

Last class

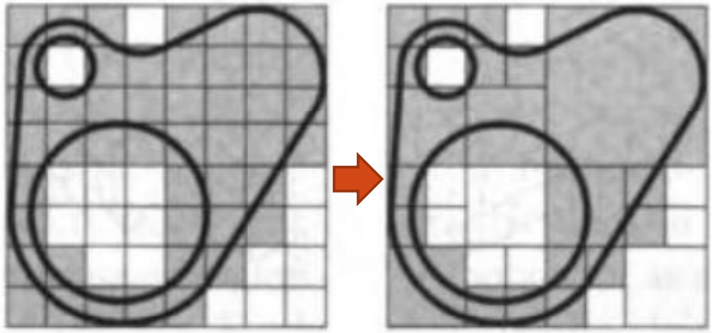
- What is Computer Graphics?
- Various applications of computer graphics techniques
  : movies, games, scientific research, engineering design, virtual reality…
- CG pipeline:
  - Data acquisition → modeling/processing → output/rendering
- Shape representation methods overview (part)
  - Polygonal meshes
  - CSG method
  - Implicit representation

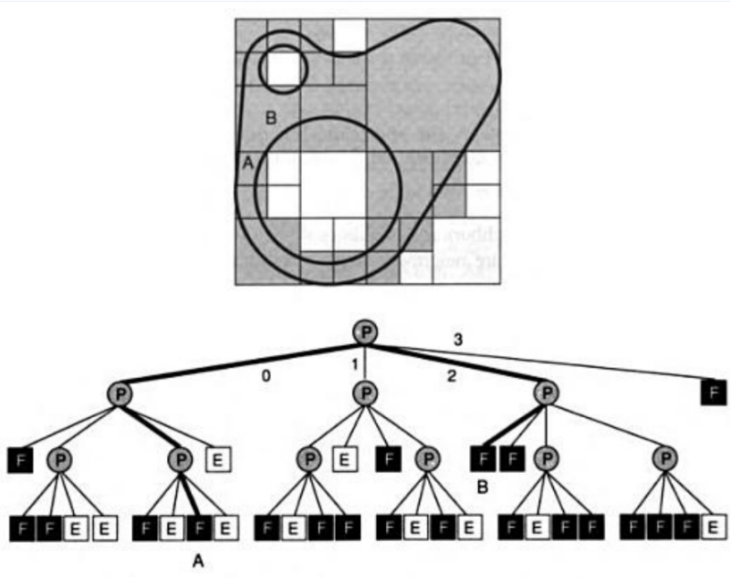
Today

- Outline
  - Shape representation methods overview (cont.)
    - Spatial partitioning method → quad-tree (2D), oct-tree (3D)
      ←will revisit it when we do animation and collision detection
    - Spline representation (in a few weeks)
    - Skeleton representation
      ← will revisit it when we do shape comparison, animation,…
    - Generalized cylinders representation
  - Focus on Half-Edge Data Structure
    - Half-edge data structure
    - Triangle mesh as a general representation scheme before you can program everything

Quadtree Rep.

- A hierarchical structure based on divide-and-conquer subdivision for 2D shapes
  - A quadtree → hierarchically represent a shape in the plane
  - Each cell may be full, partially full, or empty (depending on how much of the cell intersects the shape)
  - A partially full cell is recursively subdivided into sub-cells
  - Continue the subdivision until
    - all quadrants are homogeneous (either full or empty), or
    - a predetermined cutoff depth is reached

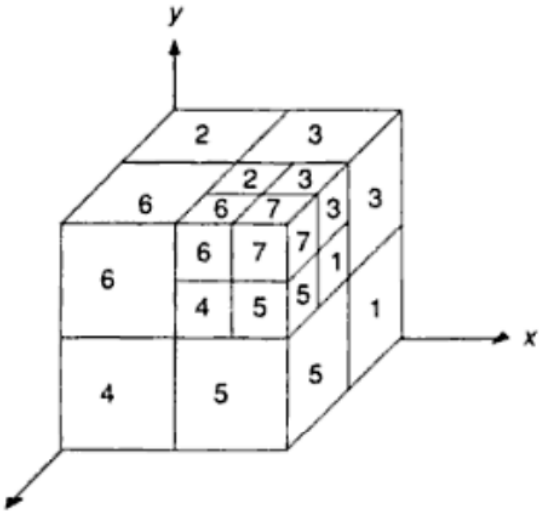




J. Warnock, "A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures", Technical Report, Univ. of Utah, 1969.

Octree Rep.

- Similar to the quadtree, but in 3D
  - Each cell → 8 children
- Much research on efficiently storing and processing quadtrees and octrees
  - e.g. Boolean operations; Neighbor finding…

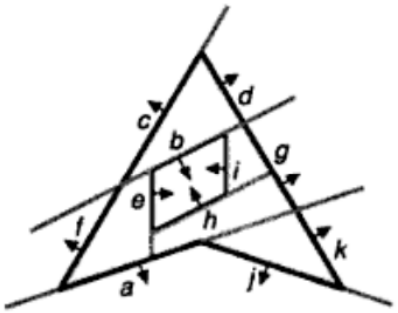


Only horizontal/vertical cutting?

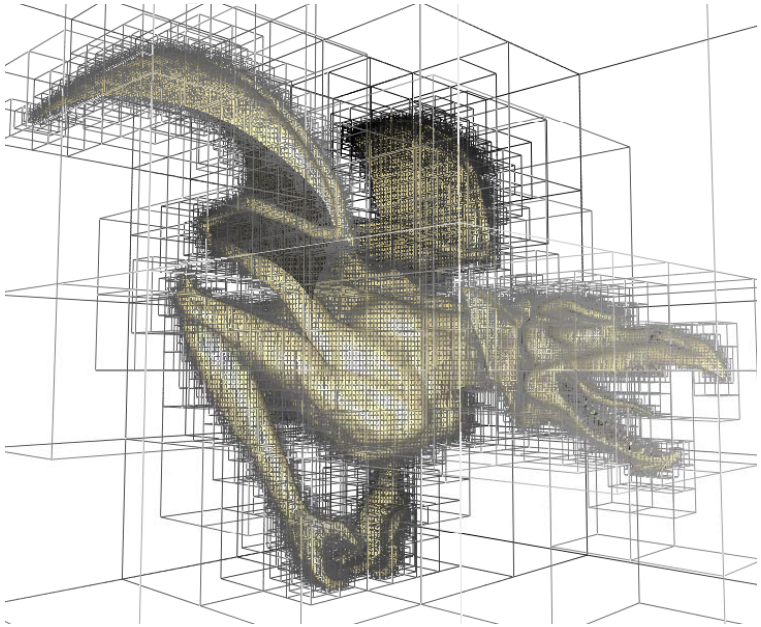
- one variant methods

→Binary space-partitioning tree

divide the space into pairs of subspaces by an arbitrary plane

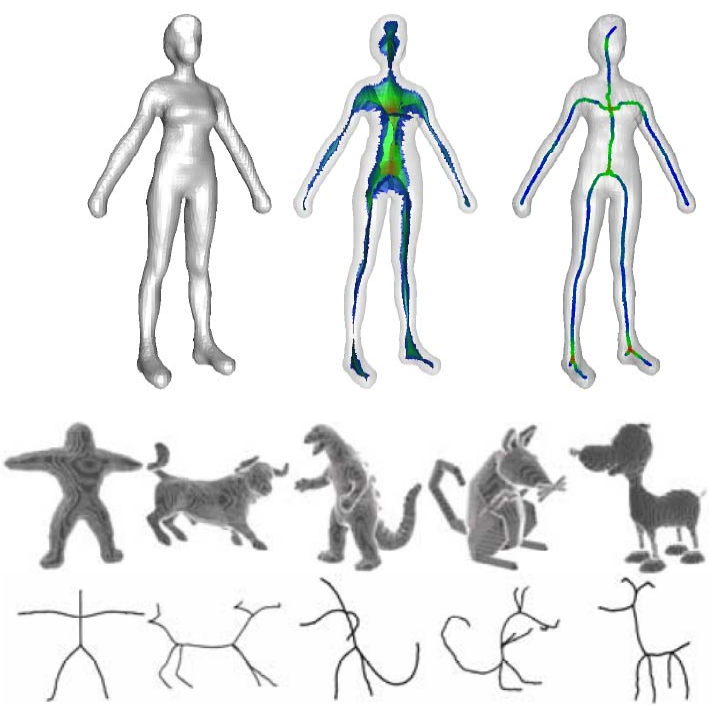


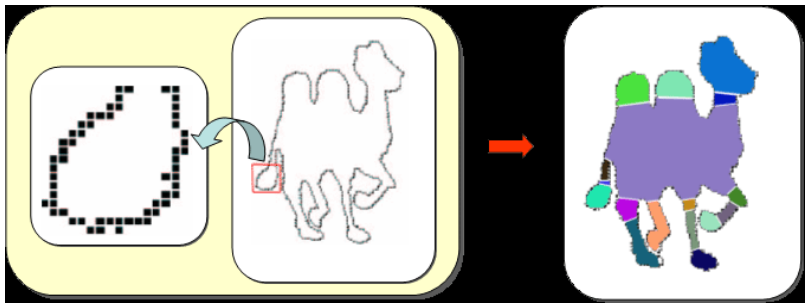
A 2D BSP tree



Skeleton Rep.

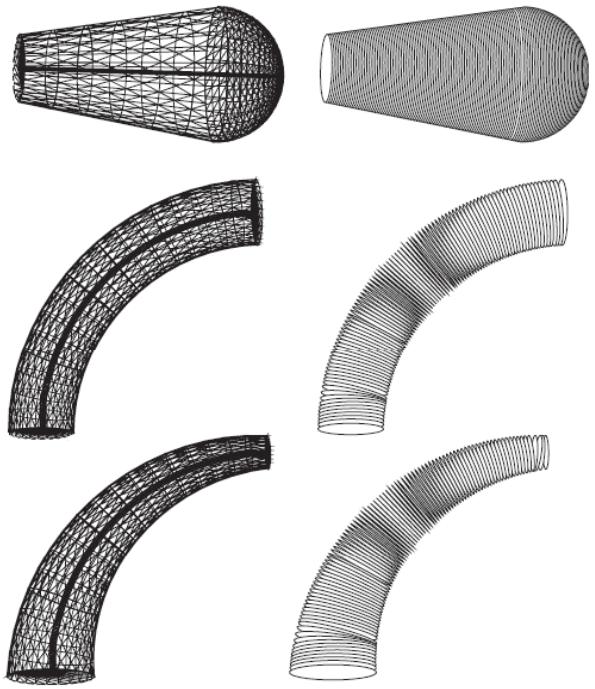
- Skeleton → a thin 1D representation of 2D/3D Objects
- Skeleton Rep. of a Shape = a (hierarchical) set of bones + attached skins
- Widely used in animation, matching, object recognition…



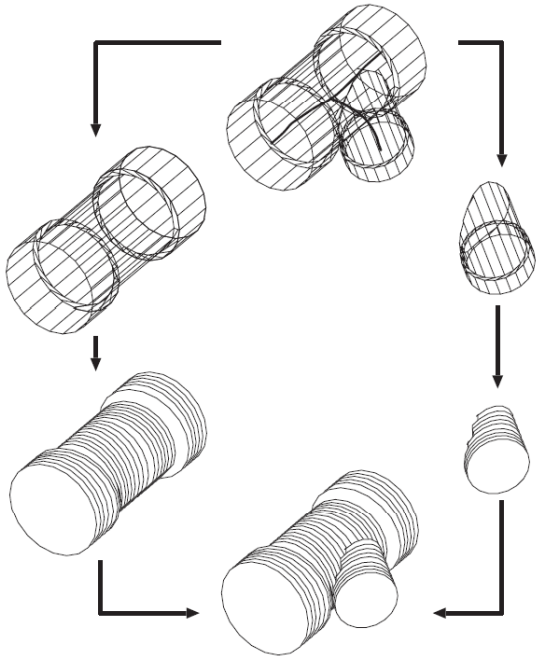


Generalized Cylinder Rep.

- A representation good for modeling articulated objects
- A shape = {axis, a cross-section curve, a scaling function}



- Good for symmetric shapes with few local details and with clear skeletal structure
- Widely used in vision community for shape recognition, and shape recover



GC axis generation

↓

Cross-Sections Generation

Half-Edge Data Structure

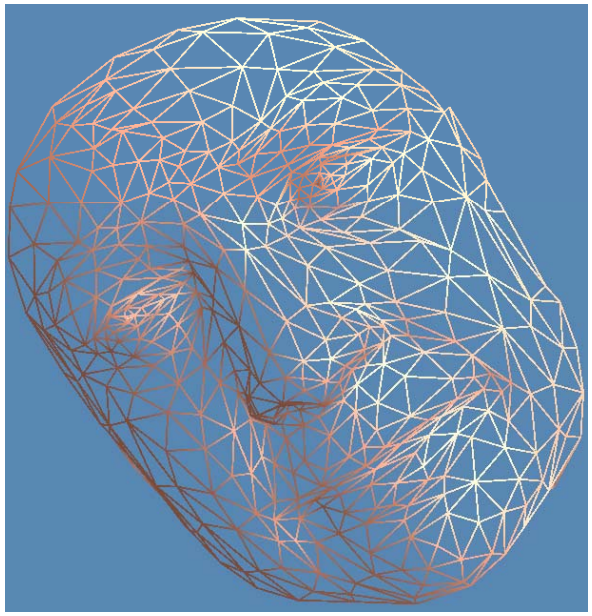
- (What?) A common way to represent triangular mesh for geometric processing
  - we focus on triangle-mesh here (it works for general polygonal mesh).
  - 3D analogy: half-face data structure for tetrahedral mesh
- (Why?) Effective for maintaining incidence information of vertices
  - Efficient local 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…)

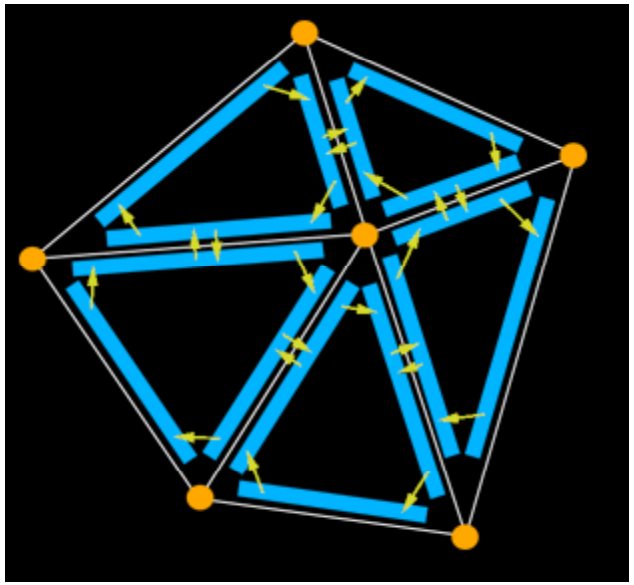


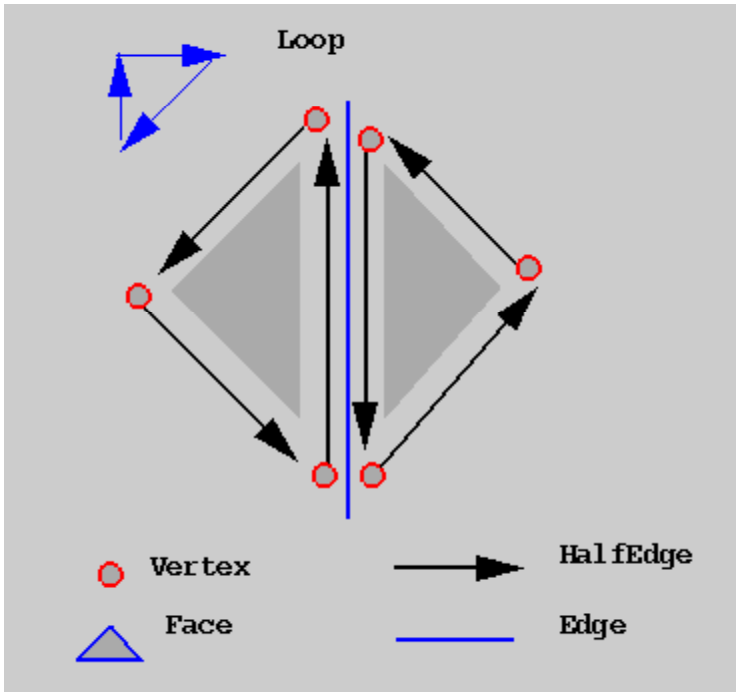
- But very difficult by just looking at those two tables
- Need a more efficient representation

Half-Edge Data Structure

Looking at a triangle mesh:

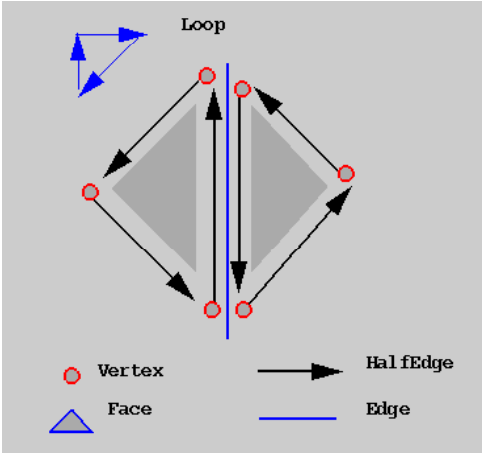
- 2 vertices share an edge, 2 faces share an edge
- each face has 3 vertices and 3 edges…
- We can store all incidence information and build a big network
  - But a vertex can have many neighboring vertices, edges, and faces
- Storing "half-edges" is simply enough
- Each edge has 2 half-edges (the boundary 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**:
  - To simply access all its incident elements → Only need a pointer to any 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)



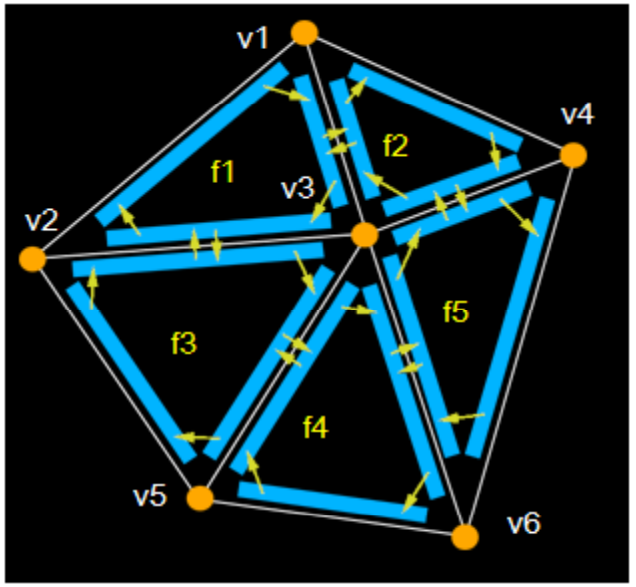
Note the directions of those half-edges bounding a face.

Linear Storage! Constant 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]



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…

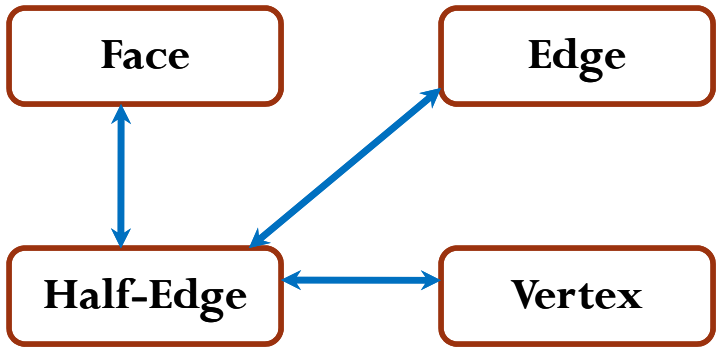
(it depends, but usually List is good enough!)

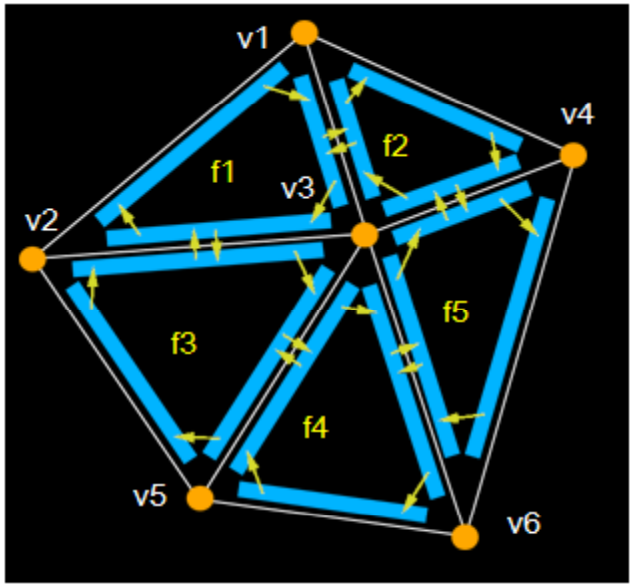
Half-Edge Data Structure (example)

→ 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:

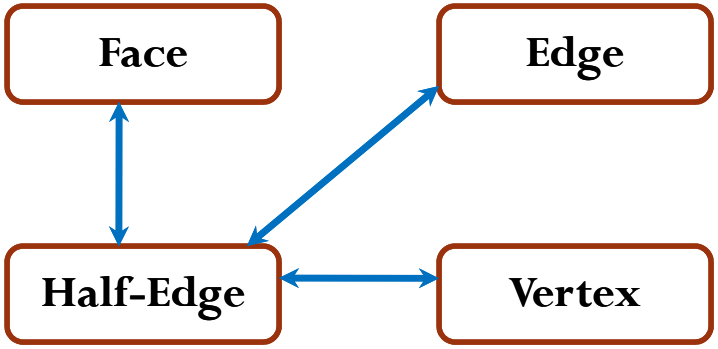


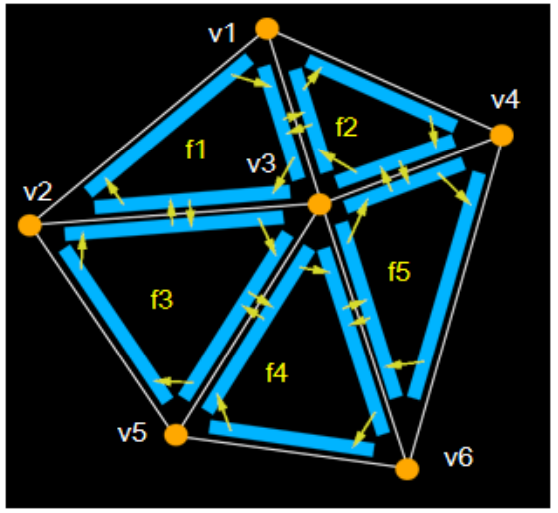


Using Half-Edge Data Structure

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…?





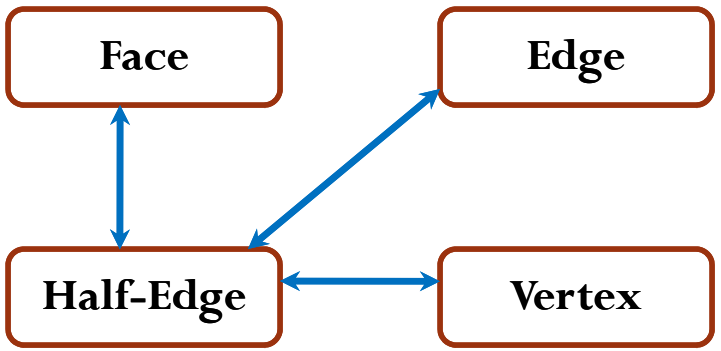
Resources for "Half-Edge" Data Structure

To get better understanding about it, you can

1) Download and read codes from:
   http://www.ece.lsu.edu/xinli/teaching/MeshLib_Simple.zip
2) Google Half-edge data structure for tutorials
3) 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.

Hint for 1):

- 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

Next class:

How to write a simple OpenGL program to render a triangle mesh?

(take your time to read half-edge data structure, not using it here yet)

On the other hand, if you hate data structure and programming…
for people don't use OpenGL, but work on 3D shapes and meshes:

- Store meshes with 2 tables, use some viewers/programs written by others as a black box to visualize even edit the model, only manipulate, compute over, and analyze the ".m" file…
- Before we can design a fully robust/powerful GUI and visualization system (which we may keep doing through the semester), here are something for us to firstly play a little bit 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 format):
    - Stanford 3D Scanning Repository: http://graphics.stanford.edu/data/3Dscanrep/
    - Aim@Shape Repository http://shapes.aim-at-shape.net/index.php
      - Part of hw 1: convert meshes with other formats to ".m" format, (will be explained later), so that they are compatible to our language…