CSE-170 Computer Graphics

Lecture 25

B-Rep Structures and Euler Formula

Dr. Renato Farias
rfarias2@ucmerced.edu

Euler Formula

Euler Formula

- REMEMBER THIS EULER FORMULA:

$$V - E + F = 2(s - h)$$

s: shells (or components)

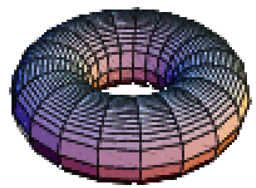
h: genus (or holes)

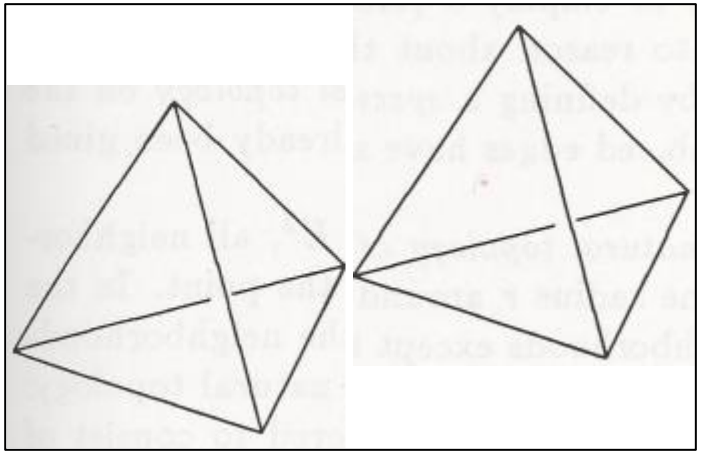
Euler characteristic

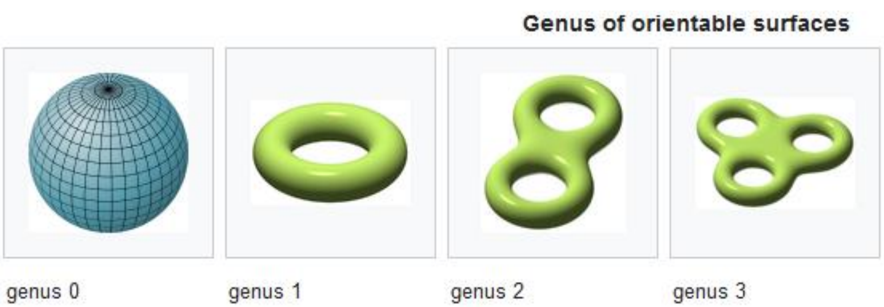
- Consider the cube example:
  - F=6, E=12, V=8
  - 8-12+6 = 2(1-0)
- V-E+F = the Euler characteristic:
  - It is the same for all cubes
    - No matter how many faces (triangles, squares, etc.) are used to describe the boundary of the cube
  - It is a constant for all objects in the same topological class
    - Ex: cubes are in the same topological class as spheres

Examples

- What are the shell and genus numbers for the following models?

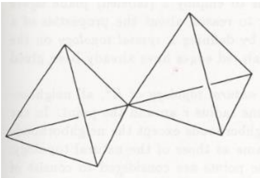






Data Structures for B-Rep

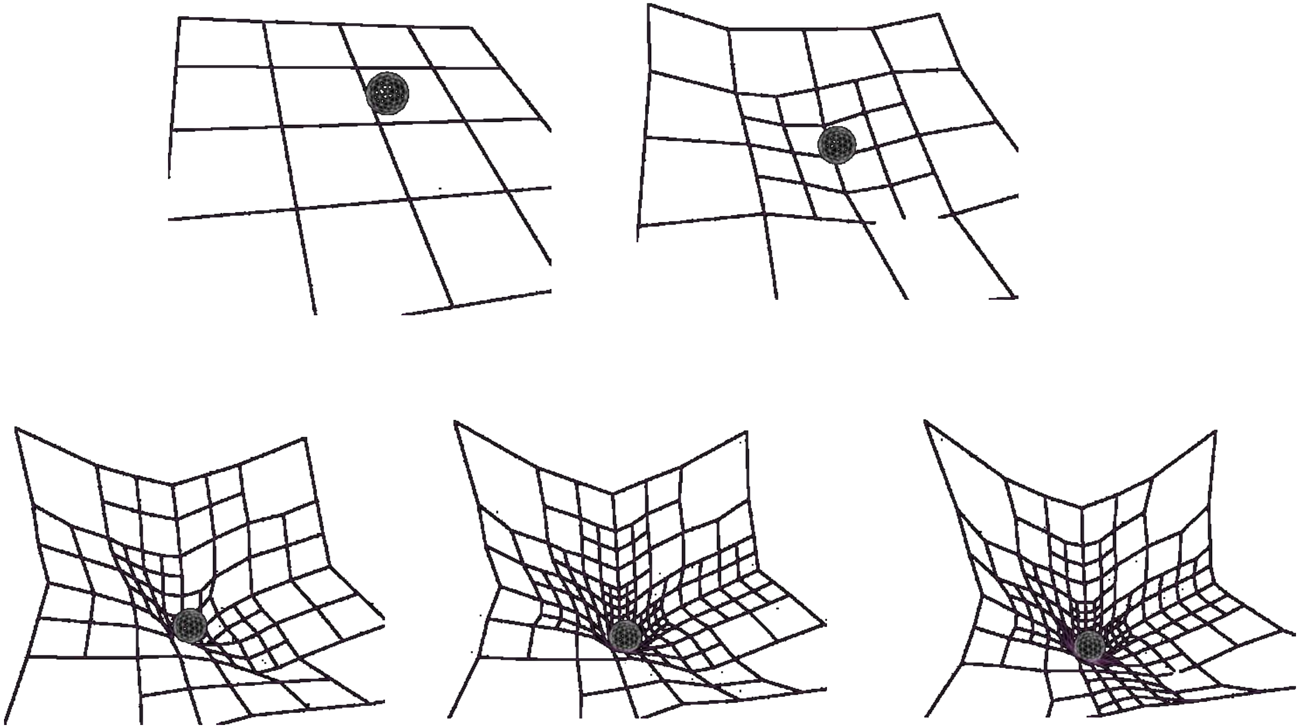
Data Structures

- Face lists
  - Already seen: simple but no adjacency relations maintained
  - No properties are guaranteed
- Edge-based graph models are needed for encoding adjacency relations
  - We want adjacency relations in constant time!
  - Main properties:
    - <u>Every edge is adjacent to only 2 faces</u>
    - <u>Edges around a vertex are ordered</u>
      - Forcing this case to become 2 shells: 

Data Structures with Fast Adjacency Relations

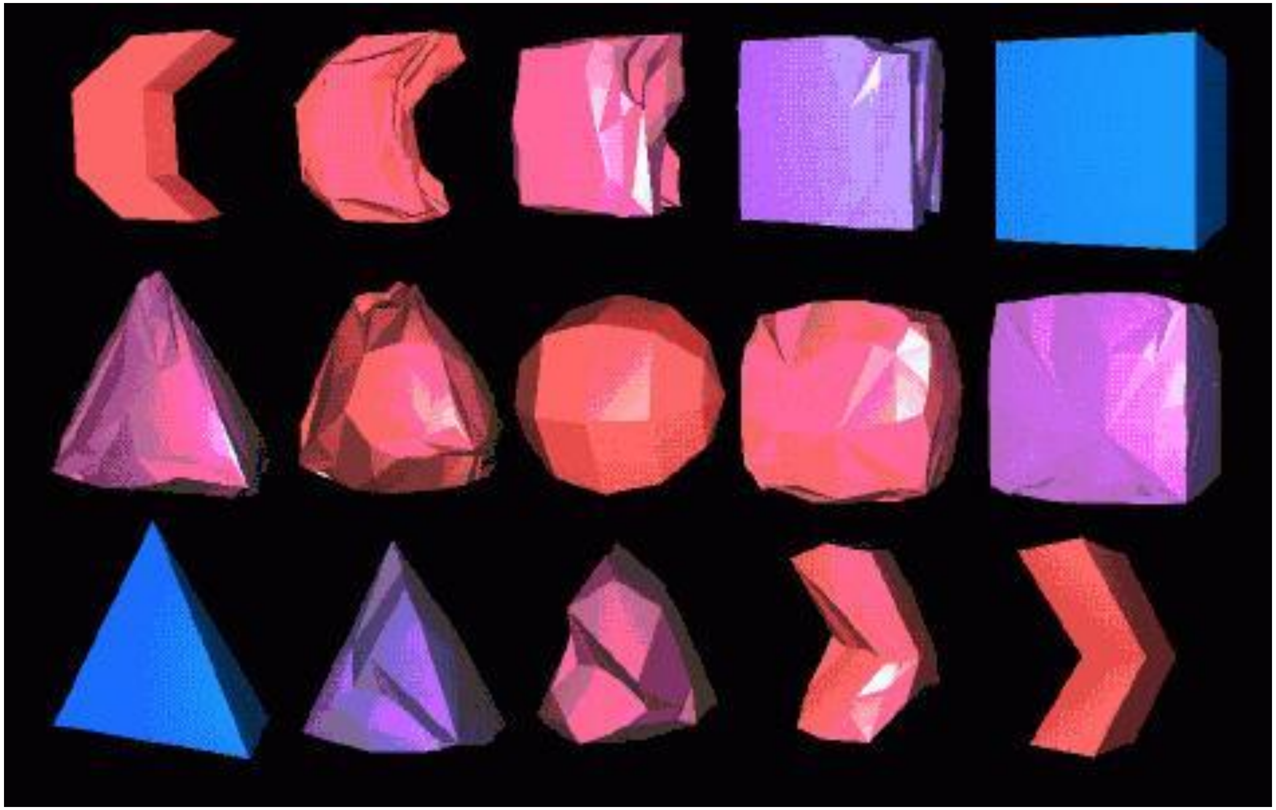
Need for adjacency relations

- Application Example
  – local refinement for deformable surfaces:



Need for adjacency relations

- Application Example
  - mesh interpolation:



Data Structures

- Important to encode:
  - Adjacencies
  - Information per vertex, edge and face
  - Structure similar to a Plane Model graph
- Edge-based models
  - First one proposed: winged-edge
  - Popular ones: half-edge, quad-edge

    (There are variations like the sym-edge - a simplified half-edge)
- Others available
  - Direct-edge, quad-edge, star-vertex, etc.

Quick Overview of Some Data Structures for Meshes

Winged-Edge Structure

Winged-Edge

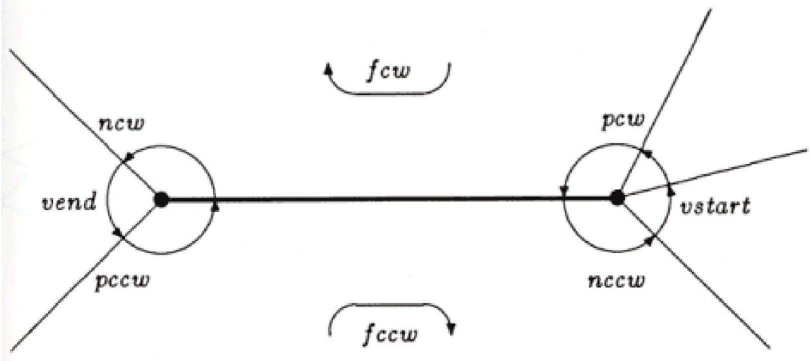
- Winged Edges
  - The Winged-Edge is the first structure of its type
    - While its main concepts remain, most of current data structures are based on pointers (forming linked lists of adjacent elements)

- Notation
  - ccw = counter-clock wise
  - cw = clockwise
  - p = prior
  - n = next

Winged-Edge

- Winged-Edge diagram:

edge	*vstart*	*vend*	*fcw*	*fccw*	*ncw*	*pcw*	*nccw*	*pccw*
e_1	v_1	v_2	f_1	f_2	e_2	e_4	e_5	e_6
e_2	v_2	v_3	f_1	f_3	e_3	e_1	e_6	e_7
e_3	v_3	v_4	f_1	f_4	e_4	e_2	e_7	e_8
e_4	v_4	v_1	f_1	f_5	e_1	e_3	e_8	e_5
e_5	v_1	v_5	f_2	f_5	e_9	e_1	e_4	e_{12}
e_6	v_2	v_6	f_3	f_2	e_{10}	e_2	e_1	e_9
e_7	v_3	v_7	f_4	f_3	e_{11}	e_3	e_2	e_{10}
e_8	v_4	v_8	f_5	f_4	e_{12}	e_4	e_3	e_{11}
e_9	v_5	v_6	f_2	f_6	e_6	e_5	e_{12}	e_{10}
e_{10}	v_6	v_7	f_3	f_6	e_7	e_6	e_9	e_{11}
e_{11}	v_7	v_8	f_4	f_6	e_8	e_7	e_{10}	e_{12}
e_{12}	v_8	v_5	f_5	f_6	e_5	e_8	e_{11}	e_9



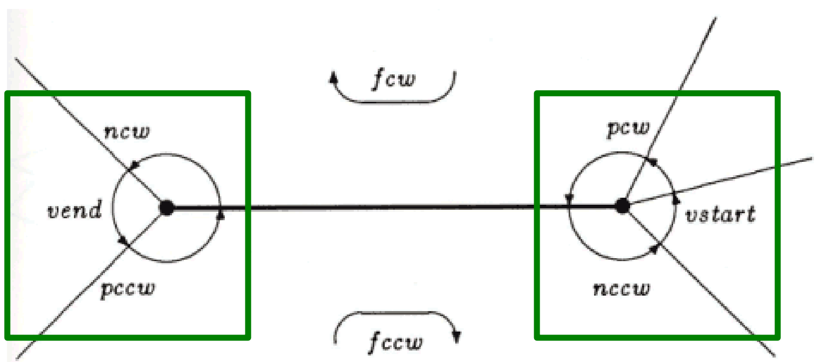
vertex	*first edge*	*coordinates*
v_1	e_1	$x_1\ y_1\ z_1$
v_2	e_2	$x_2\ y_2\ z_2$
v_3	e_3	$x_3\ y_3\ z_3$
v_4	e_4	$x_4\ y_4\ z_4$
v_5	e_9	$x_5\ y_5\ z_5$
v_6	e_{10}	$x_6\ y_6\ z_6$
v_7	e_{11}	$x_7\ y_7\ z_7$
v_8	e_{12}	$x_8\ y_8\ z_8$

face	*first edge*
f_1	e_1
f_2	e_9
f_3	e_6
f_4	e_7
f_5	e_{12}
f_6	e_9

Winged-Edge

- Ex 1: list all edges around vertex v_1 in CCW order:

edge	vstart	vend	fcw	fccw	ncw	pcw	nccw	pccw
e_1	v_1	v_2	f_1	f_2	e_2	e_4	e_5	e_6
e_2	v_2	v_3	f_1	f_3	e_3	e_1	e_6	e_7
e_3	v_3	v_4	f_1	f_4	e_4	e_2	e_7	e_8
e_4	v_4	v_1	f_1	f_5	e_1	e_3	e_8	e_5
e_5	v_1	v_5	f_2	f_5	e_9	e_1	e_4	e_{12}
e_6	v_2	v_6	f_3	f_2	e_{10}	e_2	e_1	e_9
e_7	v_3	v_7	f_4	f_3	e_{11}	e_3	e_2	e_{10}
e_8	v_4	v_8	f_5	f_4	e_{12}	e_4	e_3	e_{11}
e_9	v_5	v_6	f_2	f_6	e_6	e_5	e_{12}	e_{10}
e_{10}	v_6	v_7	f_3	f_6	e_7	e_6	e_9	e_{11}
e_{11}	v_7	v_8	f_4	f_6	e_8	e_7	e_{10}	e_{12}
e_{12}	v_8	v_5	f_5	f_6	e_5	e_8	e_{11}	e_9



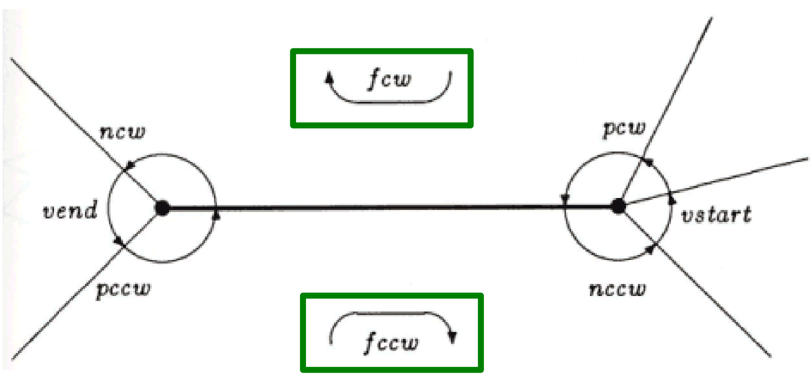
vertex	first edge	coordinates
v_1	e_1	$x_1\ y_1\ z_1$
v_2	e_2	$x_2\ y_2\ z_2$
v_3	e_3	$x_3\ y_3\ z_3$
v_4	e_4	$x_4\ y_4\ z_4$
v_5	e_9	$x_5\ y_5\ z_5$
v_6	e_{10}	$x_6\ y_6\ z_6$
v_7	e_{11}	$x_7\ y_7\ z_7$
v_8	e_{12}	$x_8\ y_8\ z_8$

face	first edge
f_1	e_1
f_2	e_9
f_3	e_6
f_4	e_7
f_5	e_{12}
f_6	e_9

Winged-Edge

- Ex 2: list all edges around face f_5 in CW order:

edge	vstart	vend	fcw	fccw	ncw	pcw	nccw	pccw
e_1	v_1	v_2	f_1	f_2	e_2	e_4	e_5	e_6
e_2	v_2	v_3	f_1	f_3	e_3	e_1	e_6	e_7
e_3	v_3	v_4	f_1	f_4	e_4	e_2	e_7	e_8
e_4	v_4	v_1	f_1	f_5	e_1	e_3	e_8	e_5
e_5	v_1	v_5	f_2	f_5	e_9	e_1	e_4	e_{12}
e_6	v_2	v_6	f_3	f_2	e_{10}	e_2	e_1	e_9
e_7	v_3	v_7	f_4	f_3	e_{11}	e_3	e_2	e_{10}
e_8	v_4	v_8	f_5	f_4	e_{12}	e_4	e_3	e_{11}
e_9	v_5	v_6	f_2	f_6	e_6	e_5	e_{12}	e_{10}
e_{10}	v_6	v_7	f_3	f_6	e_7	e_6	e_9	e_{11}
e_{11}	v_7	v_8	f_4	f_6	e_8	e_7	e_{10}	e_{12}
e_{12}	v_8	v_5	f_5	f_6	e_5	e_8	e_{11}	e_9

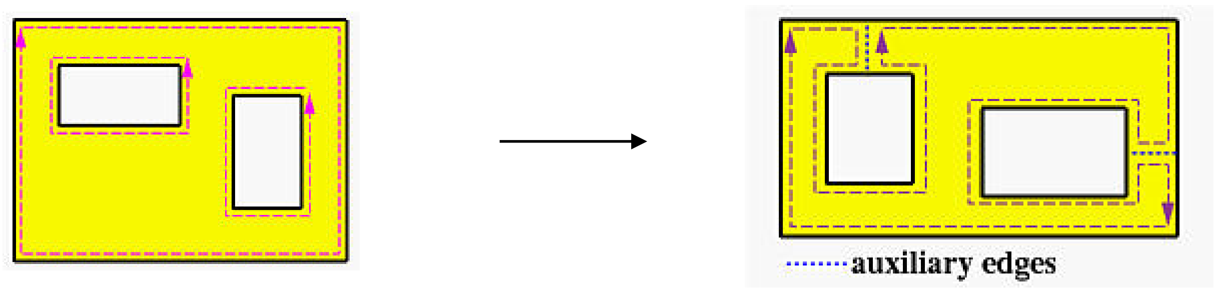


vertex	first edge	coordinates
v_1	e_1	$x_1\ y_1\ z_1$
v_2	e_2	$x_2\ y_2\ z_2$
v_3	e_3	$x_3\ y_3\ z_3$
v_4	e_4	$x_4\ y_4\ z_4$
v_5	e_9	$x_5\ y_5\ z_5$
v_6	e_{10}	$x_6\ y_6\ z_6$
v_7	e_{11}	$x_7\ y_7\ z_7$
v_8	e_{12}	$x_8\ y_8\ z_8$

face	first edge
f_1	e_1
f_2	e_9
f_3	e_6
f_4	e_7
f_5	e_{12}
f_6	e_9

Winged-Edge

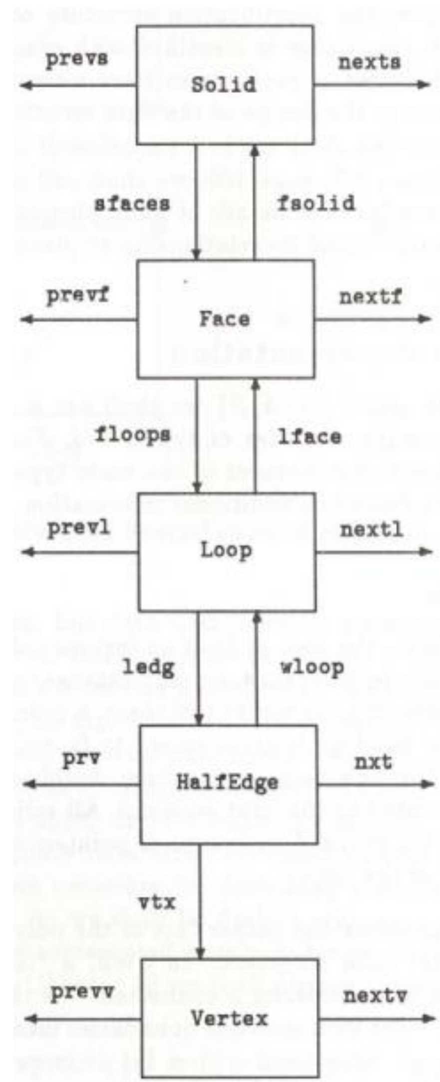
- The "hole" problem
  - If the structure does not explicitly support holes, you would need to connect them to their polygon with auxiliary edges:



Half-Edge Structure

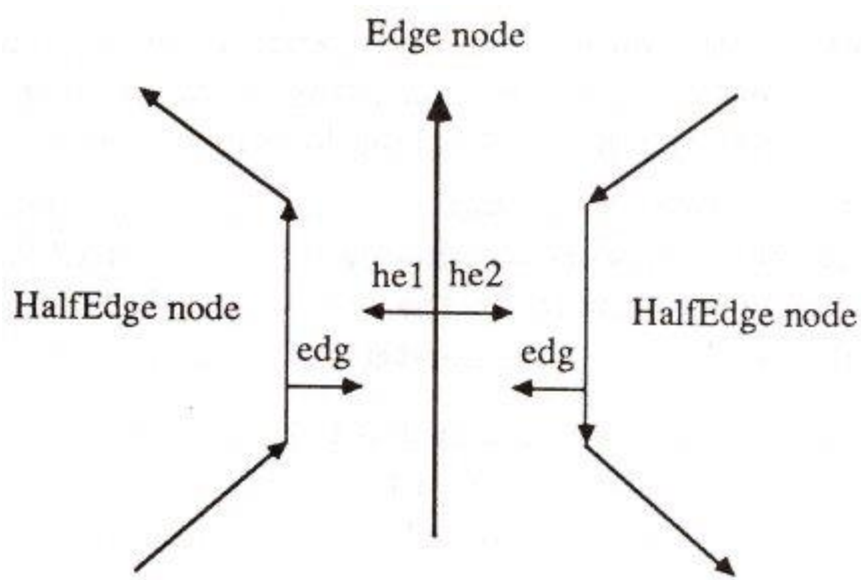
Half-Edge

- Uses the "half-edge" concept
- Based on linked lists of several elements
  - Better for dynamic updates



The Half-Edge

```
struct HalfEdge
   { Edge* edg;
     Vertex* vtx;
     Loop*   loop;
     HalfEdge* next;
     HalfEdge* prev;
   }

struct Loop
   { HalfEdge* he;
     Face*     face;
     Loop*     nextl;
     Loop*     prevl;
   }

struct Edge
   { HalfEdge* he1;
     HalfEdge* he2;
     Edge* nexte;
     Edge* preve;
   }

   etc...
```



For each edge, there are always two halfedges!

Quad-Edges, Sym-Edges, and Others

Other structures

- Quad-edge
  - Similar ideas from half-edge, but simpler
  - Optimized for triangulations
- Easier-to-implement variation: "sym-edge"
  - Simplification of Half-Edge and Quad-Edge
  - Only 2 "adjacency pointers" needed
- Others
  - Direct-edge, etc.
  - Star-vertices

Sym-Edge

- Adjacency relations
  - Every "sym-edge" has only two pointers:
    1. to the next half-edge around the same face (next)
    2. to the next half-edge around the same vertex (rotate)

Sym-Edge

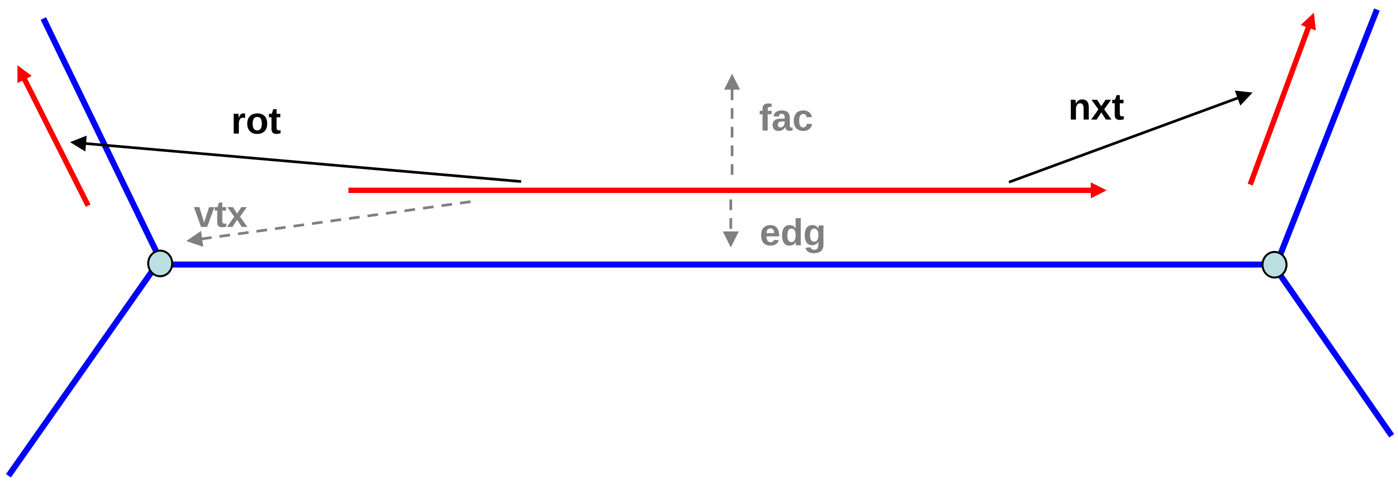
- Simple Implementation:

```
class SymEdge
 { public :
    SymEdge* next;
    SymEdge* rotate;

   public :
    Vertex* vertex;
    Edge*   edge;
    Face*   face;
 };


class Element
 { public :
    Element*  next;    // to form a linked list of elements
    Element*  prior;   // to form a linked list of elements
    SymEdge*  symedge; // to retrieve one symedge adjacent to this element
 };

class Vertex : public Element
 { float x, y, z; // vertex data comes here
 }

class Edge : public Element
 { // edge data comes here
 }

class Face : public Element
 { // face data comes here
 }
```

Sym-Edge

- Scheme:

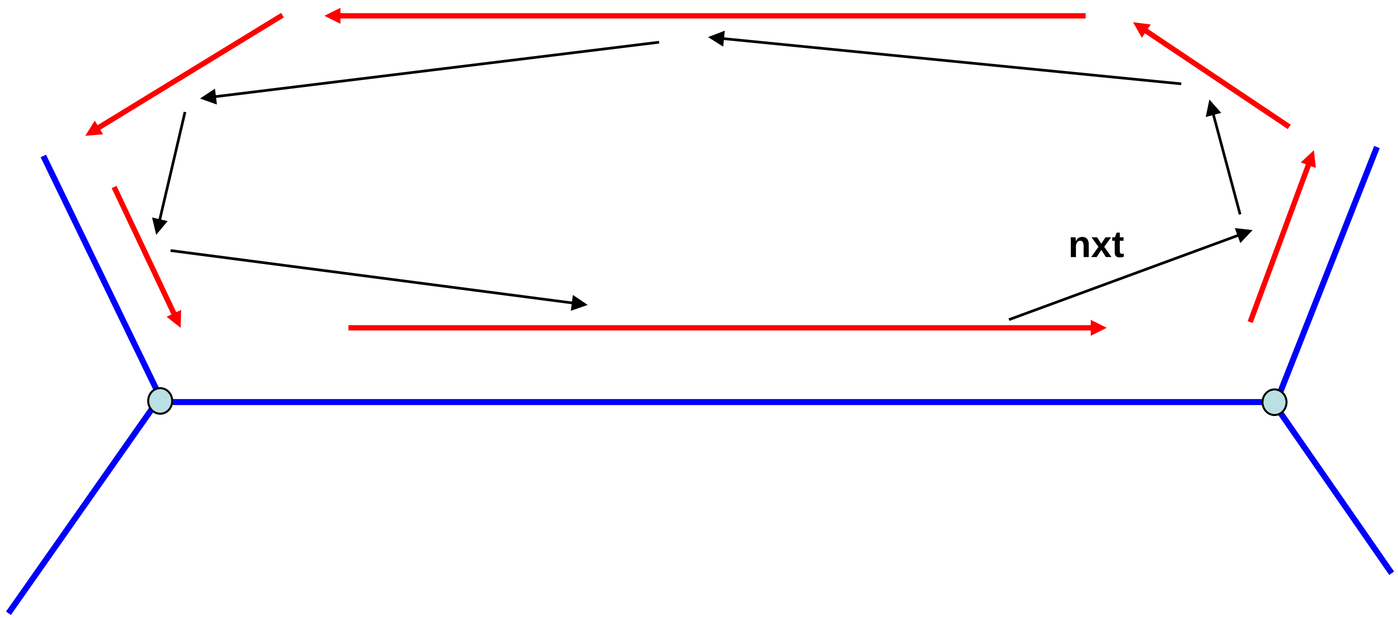
  Adjacency: rot,nxt;   optional: vtx,edg,fac



vtx, edg, fac point to additional structures storing information such as vertex coordinates, materials, etc.

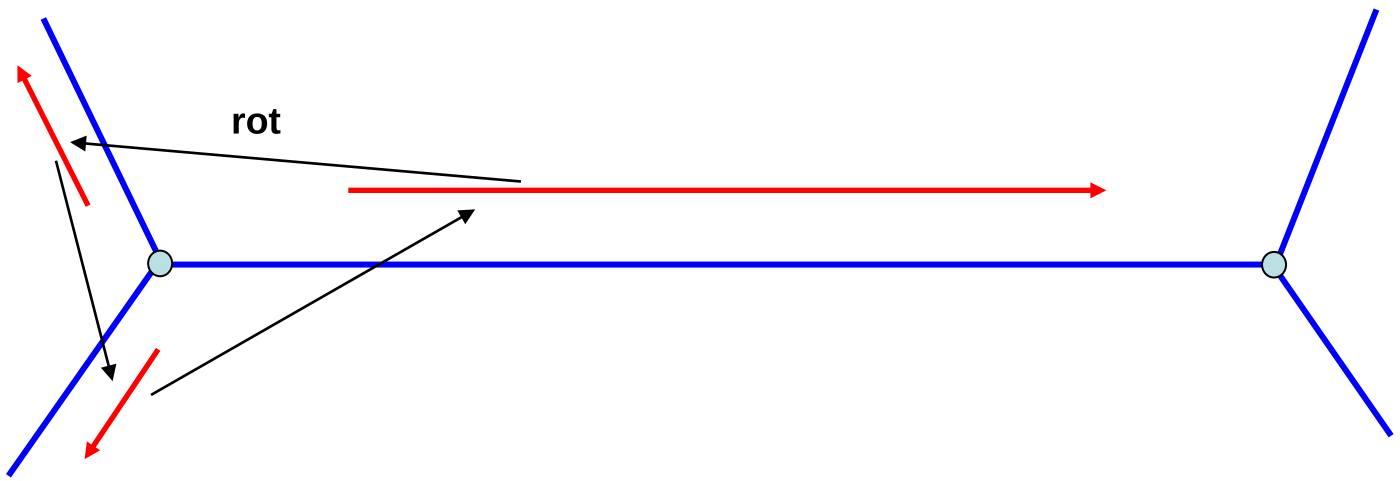
Sym-Edge

- Traversing edges around a face:



Sym-Edge

- Traversing edges around a vertex:



Sym-Edge

- All other relations can be easily found:

  Clock-wise orderings, the symmetric half-edge, etc.