Basic about Mesh

Data structure, io, show

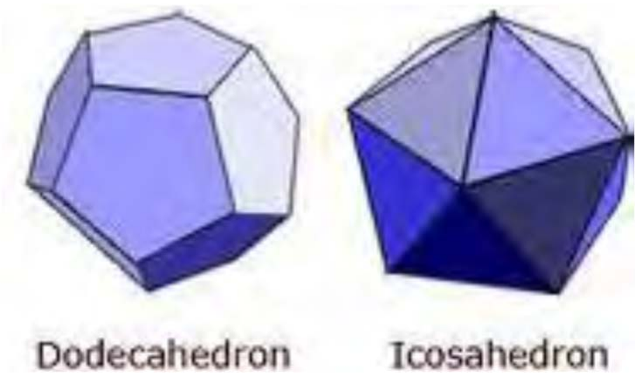
Jjcao 2013-5-24

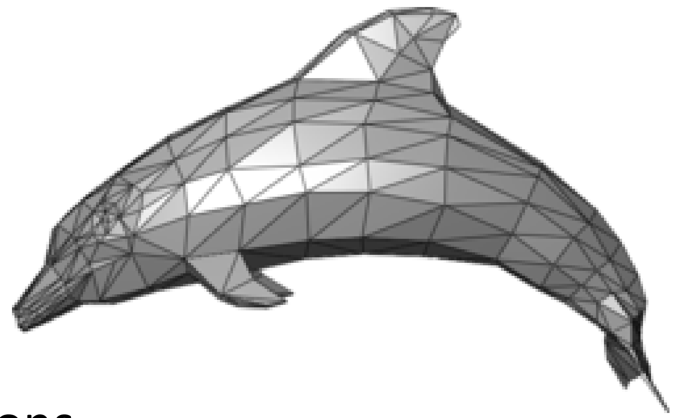
Music is dynamic, while score is static;
Movement is dynamic, while law is static.

Context

- Data structure of mesh
  - Indexed face set
  - Half-edge
- Input & output of mesh
- Display a mesh

Definition

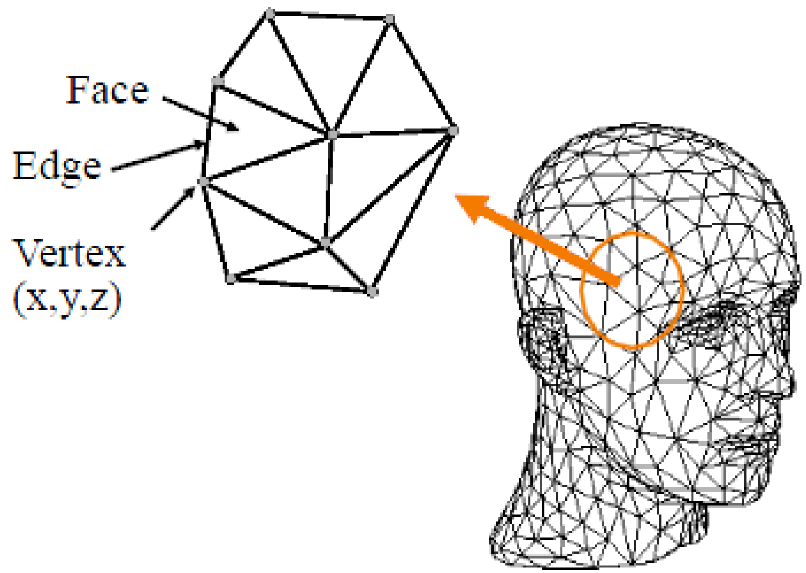




- A **polygon mesh** or **unstructured grid**
  - is a **collection** of **vertices, edges** & **faces**
  - that defines the **shape** of a **polyhedral object**
  - in 3D computer graphics & solid modeling.

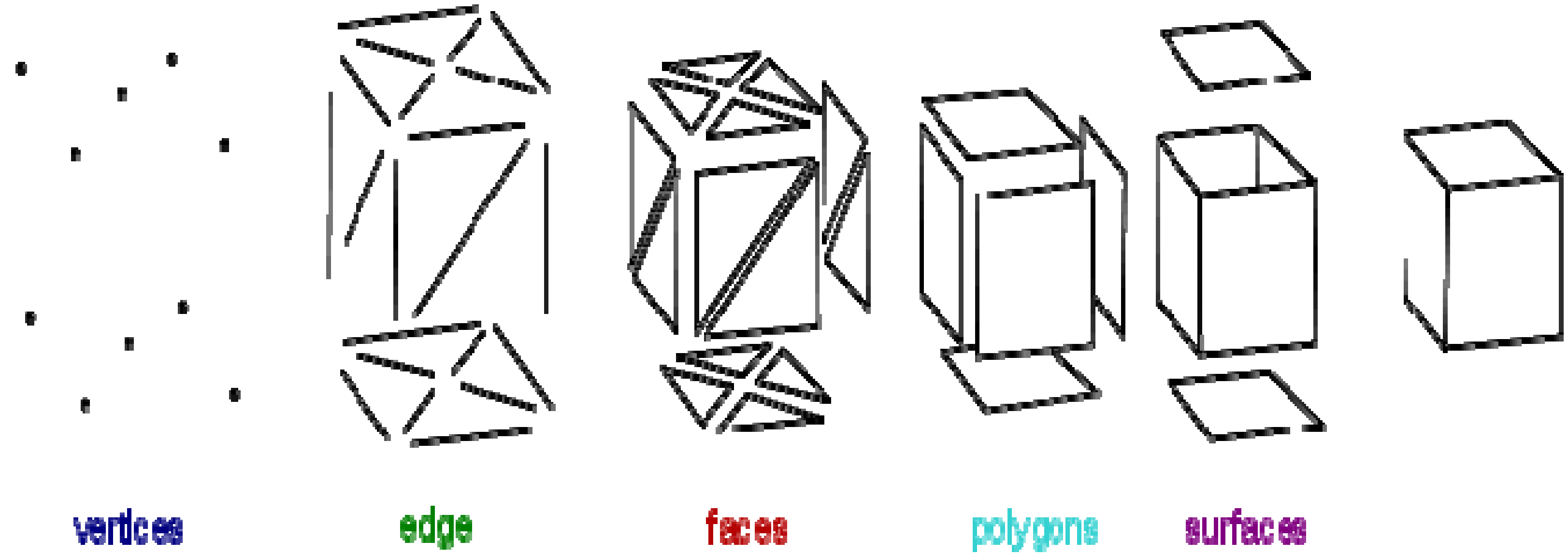
- Faces
  - triangles, quadrilaterals or other simple convex polygons
  - since this simplifies rendering,
  - but may also be composed of more general concave polygons, or polygons with holes.

Set of polygons representing a 2D surface embedded in 3D



Zorin & Schroeder

M={V,E,F}, composed of
- topology (connectivity, genus) &
- geometry information



What is a Mesh?

- A Mesh is a pair *(P,K)*, where *P* is a set of point positions $P = \{p_i \in R^3 \mid 1 \leq i \leq n\}$ and *K* is an abstract simplicial complex which contains all topological information.
- *K* is a set of subsets of $\{1, \ldots, N\}$:
  - Vertices $v = \{i\} \in V$
  - Edges $e = \{i, j\} \in E$
  - Faces $f = \{i_1, i_2, \ldots, i_{n_f}\} \in F$
- $K = V \cup E \cup F$

What is a Mesh?

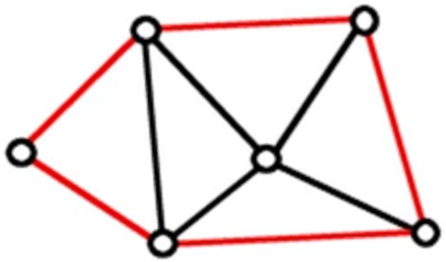
- Each edge must belong to at least one face, i.e.

$$e = \{j, k\} \in E \text{ iff } \exists\, f = \{i_1, \cdots, j, k, \cdots, i_{n_f}\} \in F$$

- Each vertex must belong to at least one edge, i.e.

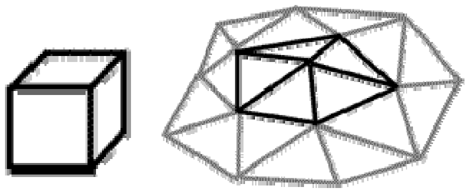
$$v = \{j\} \in V \text{ iff } \exists\, e = \{i, j\} \in E$$

- An edge is a boundary edge if it only belongs to one face

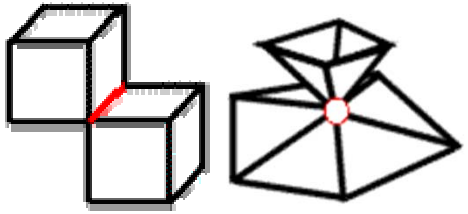


What is a Mesh?

- A mesh is a manifold if
  - Every edge is adjacent to one (boundary) or two faces
  - For every vertex, its adjacent polygons form a disk (internal vertex) or a half-disk (boundary vertex)



Manifold



Non-manifold

- A mesh is a polyhedron if
  - It is a manifold mesh and it is closed (no boundary)
  - Every vertex belongs to a cyclically ordered set of faces (local shape is a disk)

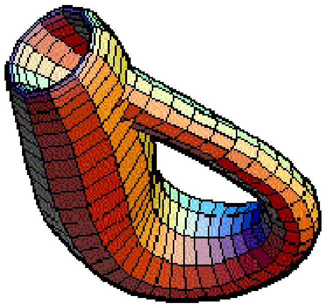
Orientation of Faces

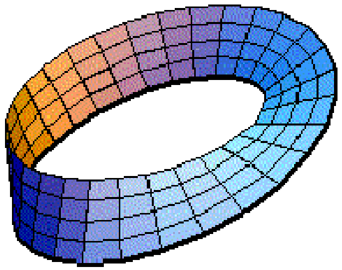
- Each face can be assigned an orientation by defining the ordering of its vertices
- Orientation can be clockwise or counter-clockwise.

- The orientation determines the normal direction of face. Usually counterclockwise order is the “front” side.

Orientation of Faces

- A mesh is well oriented (orientable) if all faces can be oriented consistently (all CCW or all CW) such that each edge has two opposite orientations for its two adjacent faces
- Not every mesh can be well oriented.
  e.g. Klein bottle, Möbius strip





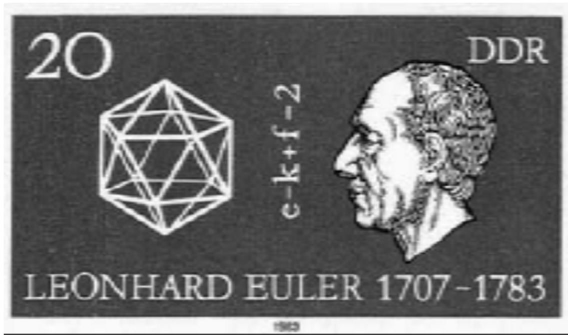
non-orientable surfaces

Euler Formula

- The relation between the number of vertices, edges, and faces.

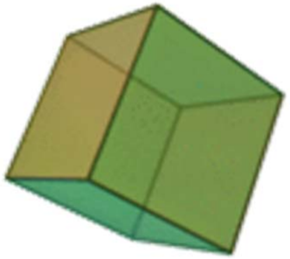
$$V - E + F = 2$$

  - where
    - V : number of vertices
    - E : number of edges
    - F : number of faces



Euler Formula

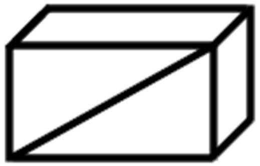
- Tetrahedron
  - V = 4
  - E = 6
  - F = 4
  - 4 - 6 + 4 = 2



- Cube
  - V = 8
  - E = 12
  - F = 6
  - 8 -12 + 6 = 2

- Octahedron
  - V = 6
  - E = 12
  - F = 8
  - 6 -12 + 8 = 2

V = 8
E = 12
F = 6
8 - 12 + 6 = 2



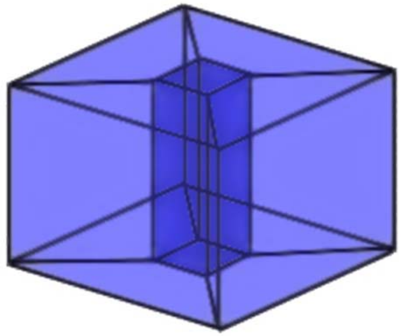
V = 8
E = 12 + 1 = 13
F = 6 + 1 = 7
8 - 13 + 7 = 2

Euler Formula

- More general rule

$$V - E + F = 2(C - G) - B$$

- where
  - V : number of vertices
  - E : number of edges
  - F : number of faces
  - C : number of connected components
  - G : number of genus (holes, handles)
  - B : number of boundaries



V = 16
E = 32
F = 16
C = 1
G = 1
B = 0
16 – 32 + 16 = 2 (1 - 1) - 0

Definition

- A mesh is a piecewise linear approximation of a 2D smooth **surface/manifold** in 3D, and it is a C^0 surface.

- **Theorem** Given a smooth surface S and a given error $\varepsilon > 0$, there exists a piecewise linear surface (mesh) M, such that $|\mathrm{M} - S| < \varepsilon$ for all points of M.

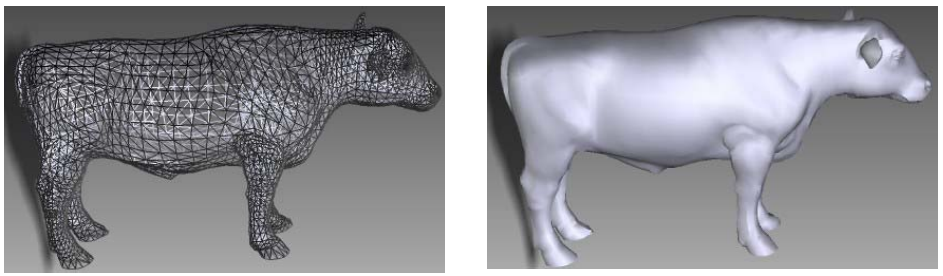
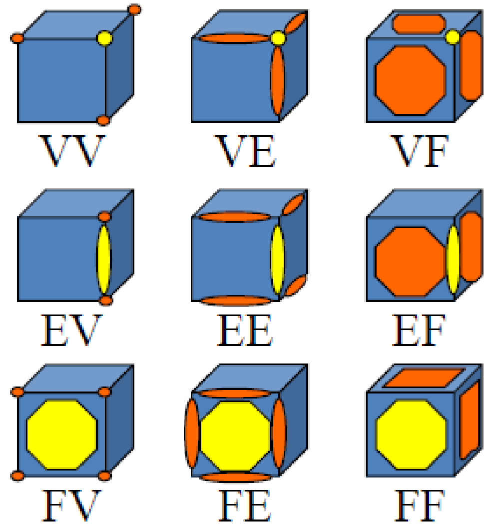


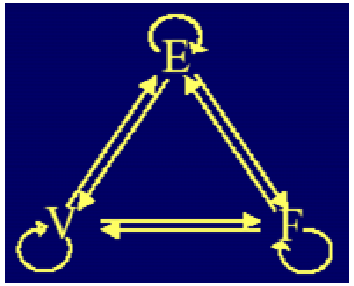
Figure 1.1. The mesh surface, (b) is the rendering of (a).

Data Structure

Neighborhood relations [Weiler 1985]

1. Vertex – Vertex VV
2. Vertex – Edge VE
3. Vertex – Face VF
4. Edge – Vertex EV
5. Edge – Edge EE
6. Edge – Face EF
7. Face – Vertex FV
8. Face – Edge FE
9. Face – Face FF





Knowing some types of relation, we can discover other (but not necessary all) topological information
e.g. if in addition to VV, VE and VF, we know neighboring vertices of a face, we can discover all neighboring edges of the face

Adjacency Relationships

Definition 3 (Vertex 1-ring). *The vertex 1-ring of a vertex $i \in V$ is*

$$V_i \stackrel{\text{def.}}{=} \{j \in V \setminus (i, j) \in E\} \subset V.$$

The s-ring is defined by induction as

$$\forall\, s > 1, \quad V_i^{(s)} = \left\{ j \in V \setminus (k, j) \in E \quad \textit{and} \quad k \in V_i^{(s-1)} \right\}.$$

Definition 4 (Face 1-ring). *The face 1-ring of a vertex $i \in V$ is*

$$F_i \stackrel{\text{def.}}{=} \{(i, j, k) \in F \setminus i, j \in V\} \subset F.$$

Mesh Representations

- Representations
  - Face-vertex meshes
    - Problem: different topological structure for triangles and quadrangles
  - Winged-edge meshes
    - Problem: traveling the neighborhood requires one case distinction
  - Half-edge meshes
  - Quad-edge meshes, Corner-tables, Vertex-vertex meshes, …
  - LR (*Laced Ring*): more compact than halfedge [siggraph2011: compact connectivity representation for triangle meshes]
    - Suited for processing meshes with fixed connectivity

Mesh Representations

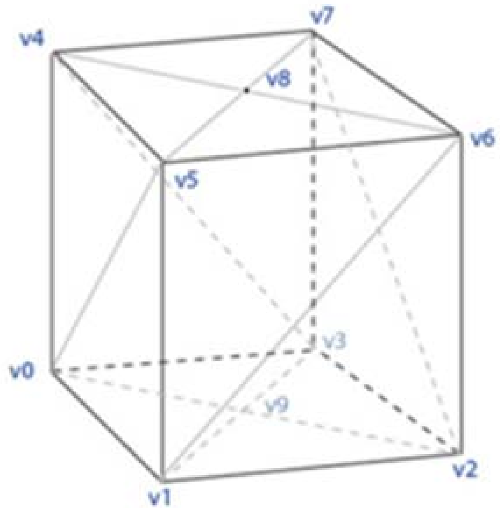
- Choice
  - Each of the representations above have particular advantages & drawbacks
  - Choice is governed by
    - Application,
    - Performance required,
    - Size of the data,
    - and Operations to be performed.
- Example
  - it is easier to deal with triangles than general polygons, especially in computational geometry.
  - For certain operations it is necessary to have a fast access to topological information such as edges or neighboring faces; this requires more complex structures such as half-edge representation.
  - For hardware rendering, compact, simple structures are needed; thus the corner-table (triangle fan) is commonly incorporated into low-level rendering APIs such as DirectX and OpenGL.

Vertex-vertex Meshes

- a set of vertices connected to other vertices
  - simplest representation, benefit from small storage space & efficient morphing of shape
  - not widely used since the face and edge information is implicit.
  - operations on edges and faces are not easily accomplished.

Vertex List

v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3



Face-vertex meshes

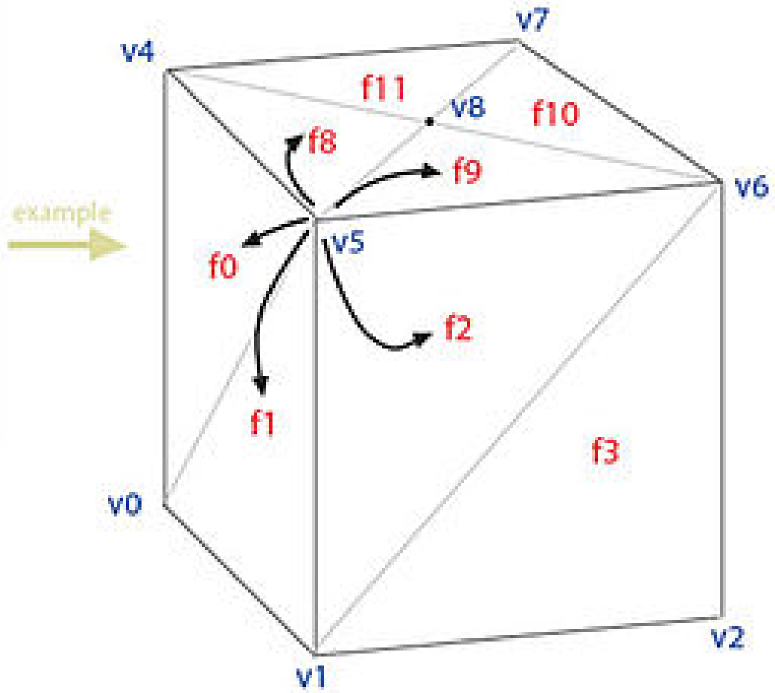
1. a set of faces and a set of vertices.
2. **most widely used, being the input typically accepted by modern graphics hardware.**
3. **One-to-one correspondence with OBJ**

Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15



Face-vertex meshes

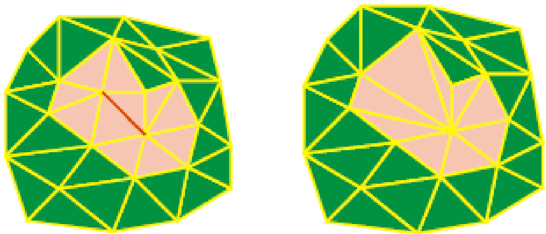
1. locating neighboring faces and vertices is constant time
2. a search is still needed to find all the faces surrounding a given face.
3. Other dynamic operations, such as splitting or merging a face, are also difficult with face-vertex meshes.

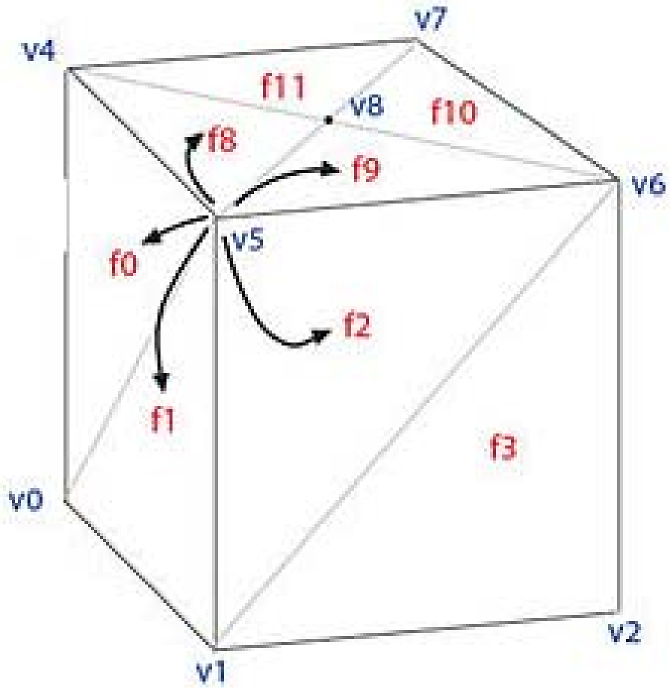
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15



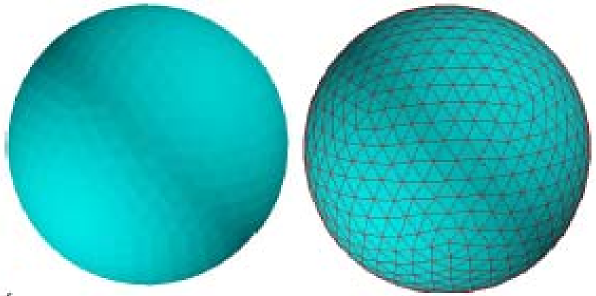


Indexed Face Set

- 3D file formats:
  - set of vertices in $\mathbb{R}^3$
  - polygons reference into vertex set
    - implicitly define edges
  - e.g.

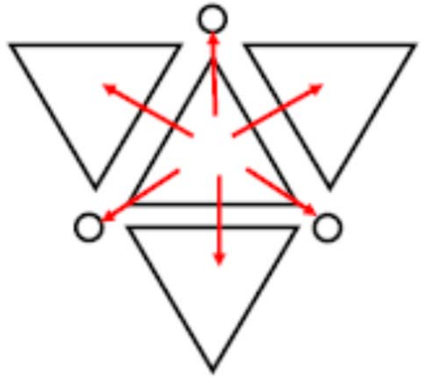
```
vertex 0 0 0
vertex 0 1 0
....
polyon 3 2 1 3
polyon 3 5 6 8
...
```

- perform transformations only on vertices



```
#VRML V1.0 ascii
#
Separator {
        Material {
                ambientColor 0.2 0.2 0.2
                diffuseColor 1.0 1.0 1.0
        }
        Coordinate3 {
                point [
                        4.455030 -1.193380 1.930940,
                        4.581220 -1.506290 1.320410,
                        4.219560 -1.875190 1.918070,
                        3.535530 1.858740 -3.007500,
                        3.793260 1.185430 -3.034130,
                        4.045080 1.545080 -2.500000,
                        3.510230 3.468900 0.803110,
                        3.556410 3.514540 0.000000,
                        3.919220 3.078210 0.405431,
                        ....
                        ....
        IndexedFaceSet {
                coordIndex [
                        0, 1, 2, -1,
                        3, 4, 5, -1,
                        6, 7, 8, -1,
                        9, 10, 11, -1,
                        12, 13, 14, -1,
                        15, 16, 17, -1,
                        18, 19, 20, -1,
                        21, 22, 23, -1,
                        ....
                        ....
```

Face-based mesh representation in C++



```
struct Face{
  Face* face[3]; //pointers to neighbors
  Vertex* vertex[3];
}
struct Vertex{
  Face* face; //pointer to a triangle adjacent to the vertex
  doulbe coordinate[3];
}
```

Face-based mesh representation in Matlab

Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vert

v0	0,0,0
v1	1,0,0
v2	1,1,0
v3	0,1,0
v4	0,0,1
v5	1,0,1
v6	1,1,1
v7	0,1,1
v8	.5,.5,0
v9	.5,.5,1

compute_vertex_ring (basic version)

- compute the 1 ring of each vertex in a triangulation.

Too slow for matlab!

```
function vring = compute_vertex_ring(face)
nverts = max(max(face));
ring{nverts} = [];

for i=1:nfaces
   for k=1:3
      ring{face(i,k)} = [ring{face(i,k)} face(i, mod((k+1),3)) face(i,
mod((k+2),3))];
   end
end

for i=1:nverts
   ring{i} = unique(ring{i});
end
```

Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Face array to Adjacency Matrix

```
function A = triangulation2adjacency(faces)
% the matlab operation should be faster! Than self implemented c++, operated
by element-wise.

A = sparse([faces(:,1); faces(:,1); faces(:,2); faces(:,2); faces(:,3); faces(:,3)], ...
           [faces(:,2); faces(:,3); faces(:,1); faces(:,3); faces(:,1); faces(:,2)], ...
           1.0);
% avoid double links
A = double(A>0);
```

```
% Test It!
verts = [0 0 0; 1 0 0; 1 1 0; 0 1 0];
faces = [1 2 3; 3 1 4];
plot_mesh(verts, faces);

OA = [0 1 1 1;
      1 0 1 0;
      1 1 0 1;
      1 0 1 0];
A = triangulation2adjacency(faces);
assert( sum(sum(A-OA)) == 0);
```

compute_vertex_ring (advanced version)

```
function vring = compute_vertex_ring(face)

nverts = max(max(face));

A = triangulation2adjacency(face);
[i,j,s] = find(sparse(A));

vring{nverts} = [];
for m = 1:length(i)
   vring{i(m)}(end+1) = j(m);
end
```

compute_vertex_face_ring

- compute the faces adjacent to each vertex
  - Direct method

```
function ring = compute_vertex_face_ring(face)
nverts = max(face(:)); ring{nverts} = [];
for i=1:nverts
   idx = find(face(:)==i);
   ring{i}=mod(idx,nface)
end
```

  - Indirect method

```
function ring = compute_vertex_face_ring(face)
nverts = max(face(:)); ring{nverts} = [];
for i=1:nfaces
   for k=1:3
      ring{face(i,k)}(end+1) = i;
   end
end
```

Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

compute_connected_verts_region

Ncut on a connected graph may lead to a cluster may contain multi non-connected patches. We wish to locate them, so the following problem is needed to be solved:

- Region growth from a seed vertex
  - The growth stops when the region is isolated by verts not in the cluster.
  - Input: **A** (Adjacency matrix of a connected graph), $\mathrm{C} = C_1 \cup \cdots \cup C_k$ (Multi-clusters, each cluster contains several connected regions. C(1)==2 means that vertex 1 belongs to cluster 2)
  - Output: Multi-clusters $\mathrm{C} = C_1 \cup \cdots \cup C_m$, each cluster contains a connected region

verts_patch

compute_connected_verts_region

Region growth from a seed vertex

The growth stops when the region is isolated by verts not in the cluster.

Input: **A** (Adjacency matrix of a connected graph), $\mathrm{C} = C_1 \cup \cdots \cup C_k$ (Multi-clusters, each cluster contains several connected regions. C(1)==2 means that vertex 1 belongs to cluster 2)

Output: Multi-cluster $\mathrm{C} = C_1 \cup \cdots \cup C_m$, each cluster contains a connected region

```
function [verts_patch] = compute_cluster(A, verts_patch)
npatch = max(verts_patch);
for i=1:npatch
   fid = find(verts_patch==i);
   B = A(fid,fid);
   [S C] = graphconncomp(B);
   for j=2:S
      tmp = fid(C==j);
      verts_patch (tmp) = max(verts_patch) + 1;
   end
end
```

Questions of mesh rep.?

- How to solve the following questions efficiently?
  - compute_edge_face_ring
    - compute faces adjacent to each edge
  - compute_face_ring
    - compute the 1 ring faces of each face in a triangulation
  - Whether a given vertex in on the boundary
  - How to traverse from one vertex to another vertex?
  - …

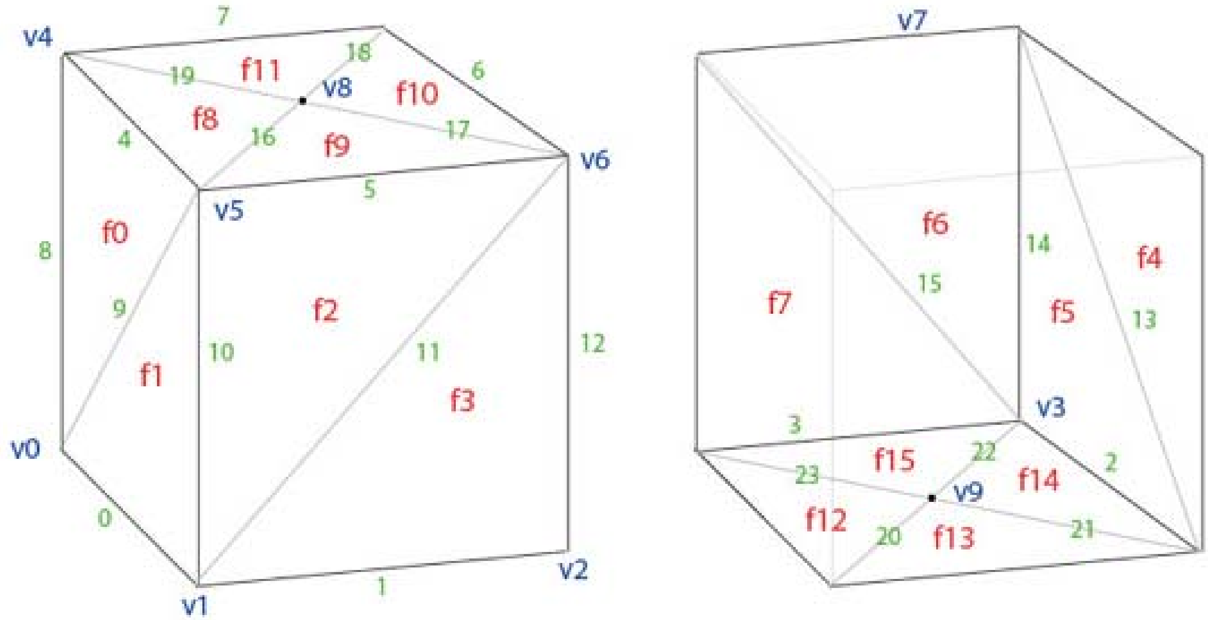
Transversal operations

- Most operations are slow for the connectivity info is not explicit.
- Need a more efficient representation

collect adjacent \ iterate over	V	E	F
V	quadratic	quadratic	linear
E	quadratic	quadratic	linear
F	quadratic	quadratic	linear

Winged-edge meshes

- explicitly represent the vertices, faces, and edges of a mesh.
- greatest flexibility in dynamically changing the mesh
- large storage requirements and increased complexity due to maintaining many indices



Winged-edge meshes

Face List

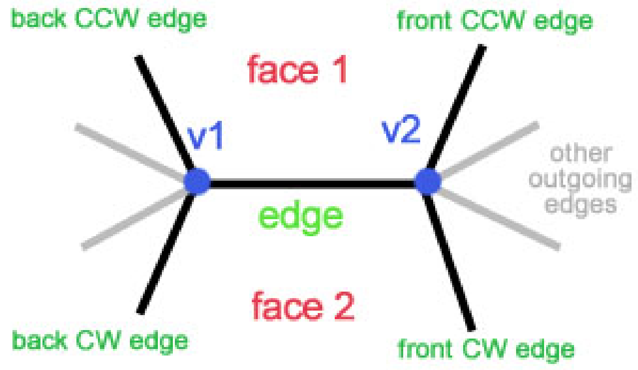
f0	4 8 9
f1	0 10 9
f2	5 10 11
f3	1 12 11
f4	6 12 13
f5	2 14 13
f6	7 14 15
f7	3 8 15
f8	4 16 19
f9	5 17 16
f10	6 18 17
f11	7 19 18
f12	0 23 20
f13	1 20 21
f14	2 21 22
f15	3 22 23

Edge List

e0	v0 v1	f1 f12	9 23 10 20
e1	v1 v2	f3 f13	11 20 12 21
e2	v2 v3	f5 f14	13 21 14 22
e3	v3 v0	f7 f15	15 22 8 23
e4	v4 v5	f0 f8	19 8 16 9
e5	v5 v6	f2 f9	16 10 17 11
e6	v6 v7	f4 f10	17 12 18 13
e7	v7 v4	f6 f11	18 14 19 15
e8	v0 v4	f7 f0	3 9 7 4
e9	v0 v5	f0 f1	8 0 4 10
e10	v1 v5	f1 f2	0 11 9 5
e11	v1 v6	f2 f3	10 1 5 12
e12	v2 v6	f3 f4	1 13 11 6
e13	v2 v7	f4 f5	12 2 6 14
e14	v3 v7	f5 f6	2 15 13 7
e15	v3 v4	f6 f7	14 3 7 15
e16	v5 v8	f8 f9	4 5 19 17
e17	v6 v8	f9 f10	5 6 16 18
e18	v7 v8	f10 f11	6 7 17 19
e19	v4 v8	f11 f8	7 4 18 16
e20	v1 v9	f12 f13	0 1 23 21
e21	v2 v9	f13 f14	1 2 20 22
e22	v3 v9	f14 f15	2 3 21 23
e23	v0 v9	f15 f12	3 0 22 20

Vertex List

v0	0,0,0	8 9 0 23 3
v1	1,0,0	10 11 1 20 0
v2	1,1,0	12 13 2 21 1
v3	0,1,0	14 15 3 22 2
v4	0,0,1	8 15 7 19 4
v5	1,0,1	10 9 4 16 5
v6	1,1,1	12 11 5 17 6
v7	0,1,1	14 13 6 18 7
v8	.5,.5,0	16 17 18 19
v9	.5,.5,1	20 21 22 23



Winged Edge Structure

Render dynamic meshes

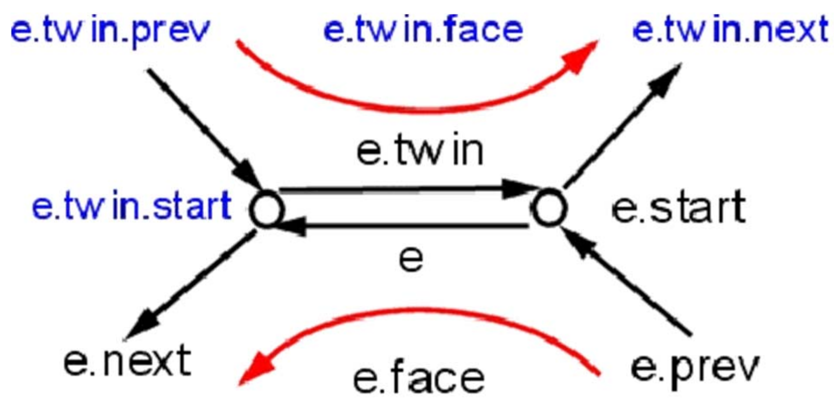
- combines winged-edge meshes and face-vertex meshes
- require slightly less storage space than standard winged-edge meshes,
- and can be directly rendered by graphics hardware since the face list contains an index of vertices.

Operation		Vertex-vertex	Face-vertex	Winged-edge	Render dynamic
V-V	All vertices around vertex	Explicit	V → f1, f2, f3, ... → v1, v2, v3, ...	V → e1, e2, e3, ... → v1, v2, v3, ...	V → e1, e2, e3, ... → v1, v2, v3, ...
E-F	All edges of a face	F(a, b, c) → {a, b}, {b, c}, {a, c}	F → {a, b}, {b, c}, {a, c}	Explicit	Explicit
V-F	All vertices of a face	F(a, b, c) → {a, b, c}	Explicit	F → e1, e2, e3 → a, b, c	Explicit
F-V	All faces around a vertex	Pair search	Explicit	V → e1, e2, e3 → f1, f2, f3, ...	Explicit
E-V	All edges around a vertex	V → {v, v1}, {v, v2}, {v, v3}, ...	V → f1, f2, f3, ... → v1, v2, v3, ...	Explicit	Explicit
F-E	Both faces of an edge	List compare	List compare	Explicit	Explicit
V-E	Both vertices of an edge	E(a, b) → {a, b}	E(a, b) → {a, b}	Explicit	Explicit
Flook	Find face with given vertices	F(a, b, c) → {a, b, c}	Set intersection of v1, v2, v3	Set intersection of v1, v2, v3	Set intersection of v1, v2, v3
Storage size		V*avg(V, V)	3F + V*avg(F, V)	3F + 8E + V*avg(E, V)	6F + 4E + V*avg(E, V)
		Example with 10 vertices, 16 faces, 24 edges:			
		10 * 5 = 50	3*16 + 10*5 = 98	3*16 + 8*24 + 10*5 = 290	6*16 + 4*24 + 10*5 = 242

Figure 6: summary of mesh representation operations

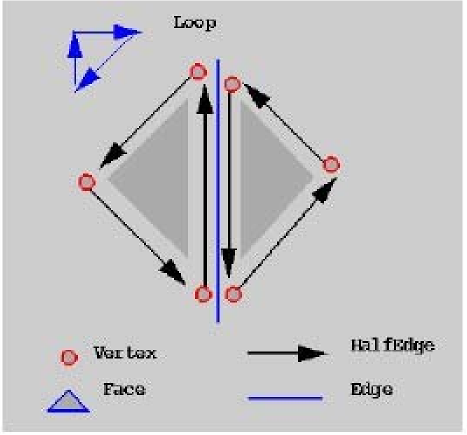
Half-Edge Data Structure

- Each edge is divided into two half-edges
- Each half-edge has 5 references:
  - The face on left side (assume counter-clockwise order)
  - Previous and next half-edge in counterclockwise order
  - The "twin" edge
  - The starting vertex
- Each face has a pointer to one of its edges
- Each vertex has a pointer to a half edge that has this vertex as the start vertex



Half-Edge Data Structure (cont.)

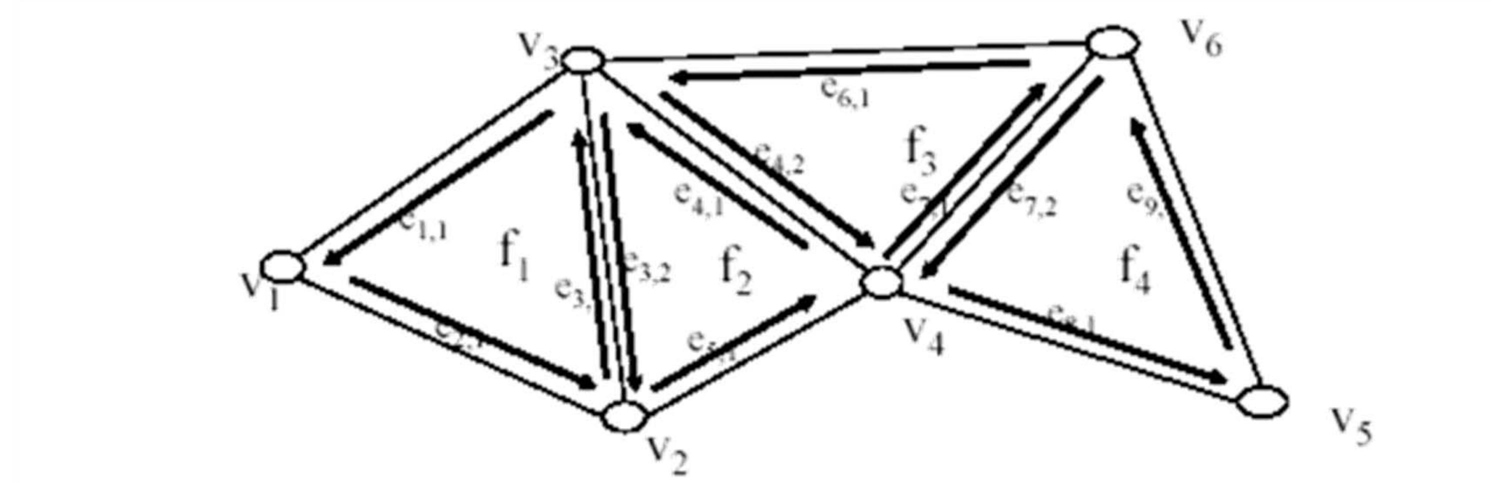
- For each **edge**:
  - it has 2 half-edges (the boundary edge has 1)
  - they are called twins to each other
- For each **half-edge**:
  - bounds 1 face and 1 edge → a face pointer, an edge pointer, respectively
  - has one origin, and one target vertex → a vertex pointer (for the target)
  - <u>To be able to walk around a face:</u>
  - 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.

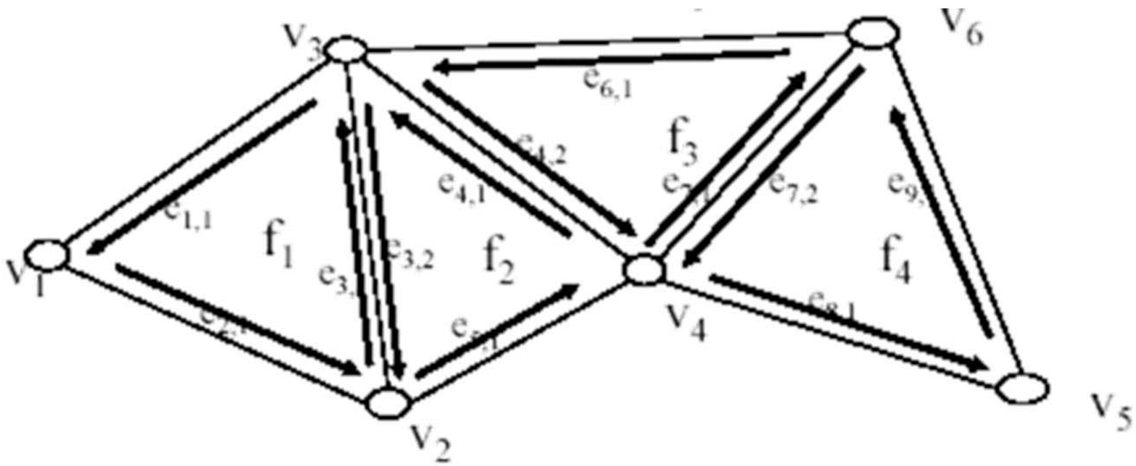
Linear Storage! Constant Local Traversal!

Half-edge data structure: example



half-edge	origin	twin	incident face	next	prev
$e_{3,1}$	v_2	$e_{3,2}$	f_1	$e_{1,1}$	$e_{3,1}$
$e_{3,2}$	v_3	$e_{3,1}$	f_2	$e_{5,1}$	$e_{4,1}$
$e_{4,1}$	v_4	$e_{4,2}$	f_2	$e_{3,2}$	$e_{5,1}$
$e_{4,2}$	v_3	$e_{4,1}$	f_3	$e_{7,1}$	$e_{6,1}$

Half-Edge Data Structure



vertex	coordinate	IncidentEdge
v_1	(x_1,y_1,z_1)	$e_{2,1}$
v_2	(x_2,y_2,z_2)	$e_{5,1}$
v_3	(x_3,y_3,z_3)	$e_{1,1}$
v_4	(x_4,y_4,z_4)	$e_{7,1}$
v_5	(x_5,y_5,z_5)	$e_{9,1}$
v_6	(x_6,y_6,z_6)	$e_{7,2}$

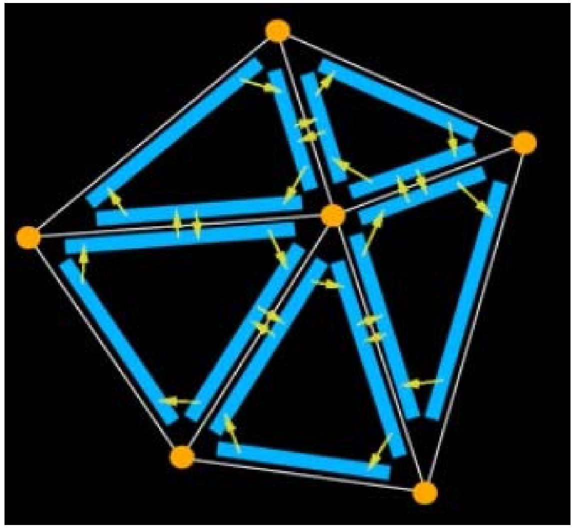
face	edge
f_1	$e_{1,1}$
f_2	$e_{5,1}$
f_3	$e_{4,2}$
f_4	$e_{8,1}$

Half-edge data structure

(What?)A common way to represent triangular (polyhedral) mesh.
3D analogy: half-face data structure for tetrahedral mesh

(Why?) Effective for maintaining incidence info of vertices:

- Efficient local traversal
- Low spatial cost
- Supporting dynamic local updates/manipulations (edge collapse, vertex split, etc.)



(Who?)

- CGAL, OpenMesh (OpenSG), MCGL (for matlab)
- A free library from Xin li.
- A free surface library from Xianfeng Gu.
- Denis Zorin uses it in implementing Subdivision.

Mesh operations

Traversals over all elements of certain type

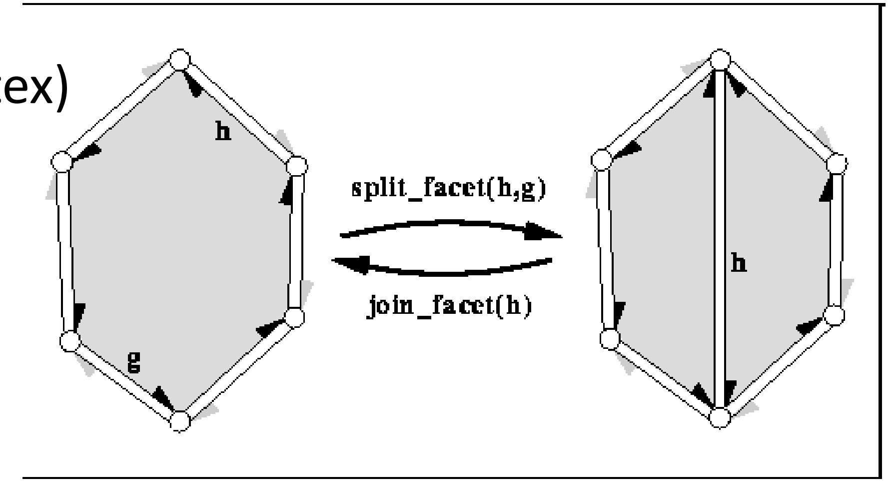
Navigate adjacent elements (e.g. one-ring of a vertex)

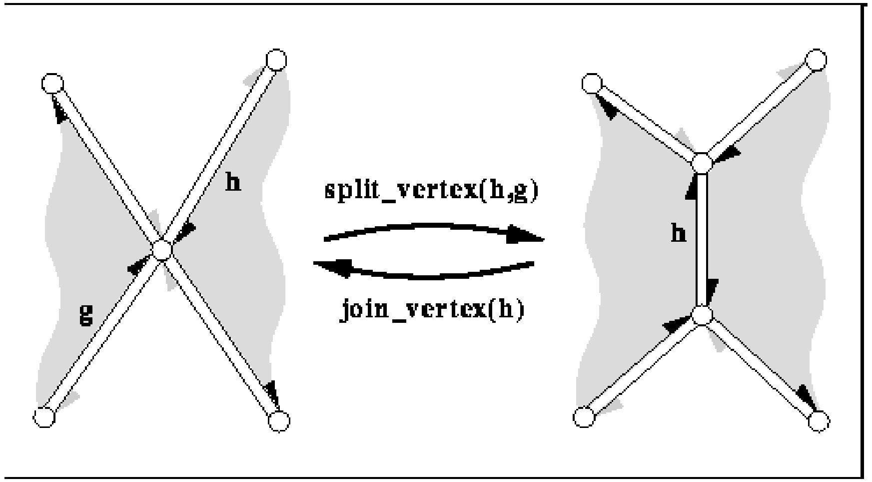
Refinement

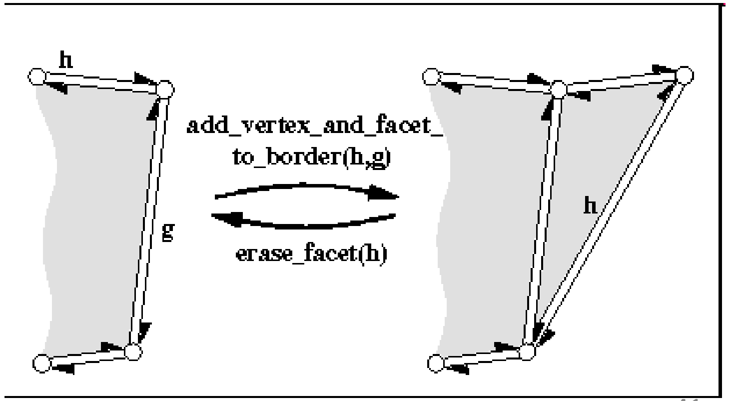
Edge flips

Face addition/deletion

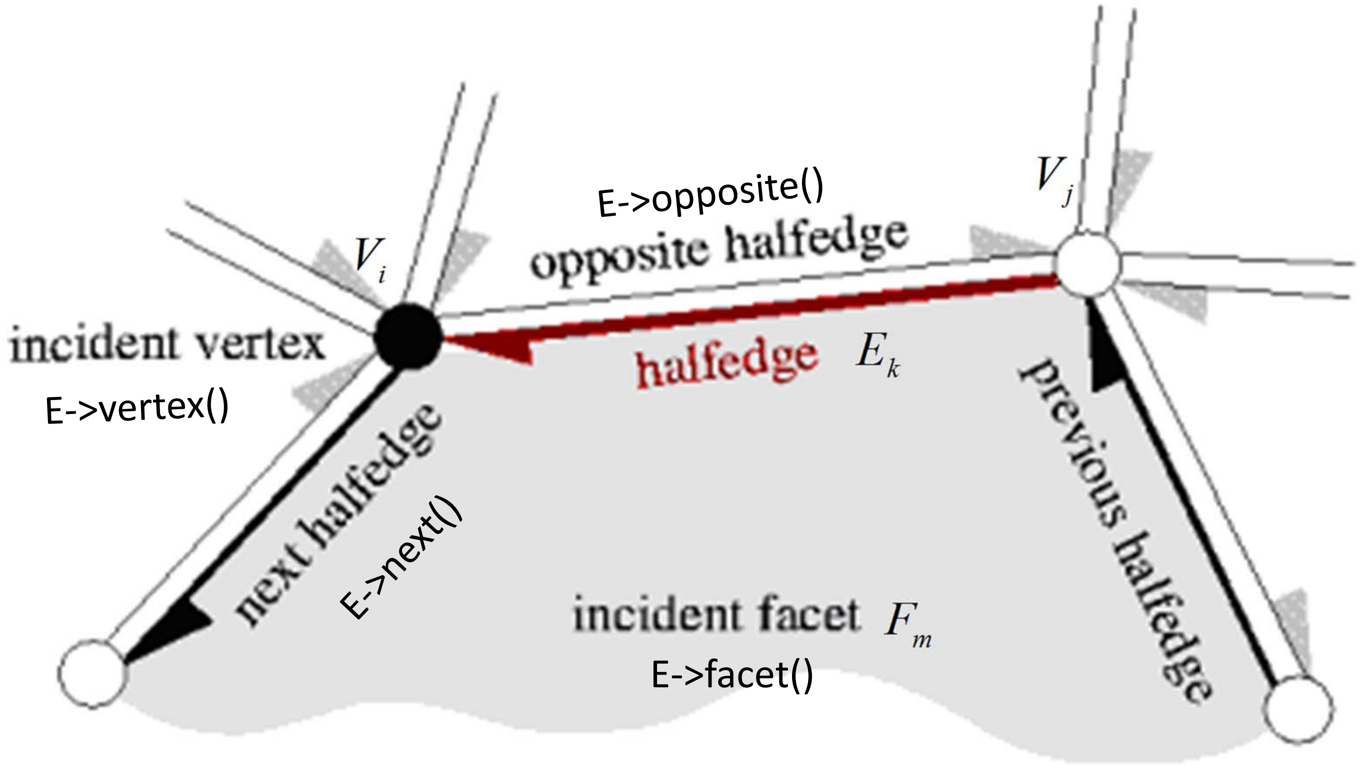
Face merge





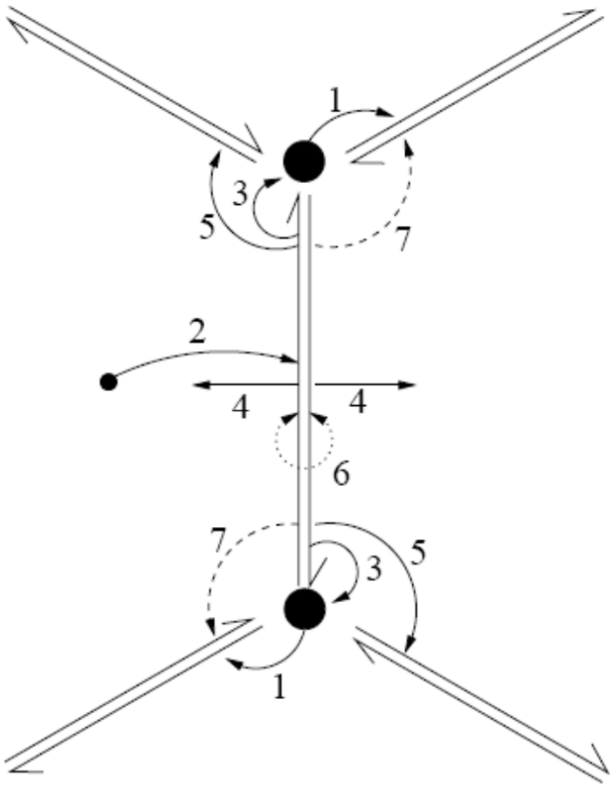


How HDS can -- CGAL



Halfedge_around_vertex_const_circulator cir = V->vertex_begin(), cir_end =cir;
CGAL_For_all(cir, cir_end) { if (cir->opposite()->vertex() == source) …;}

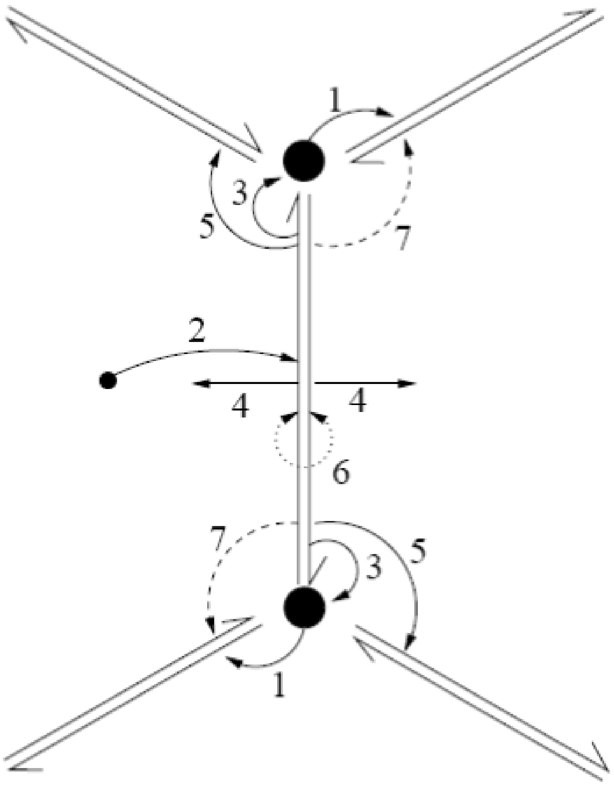
How HDS can -- OpenSG

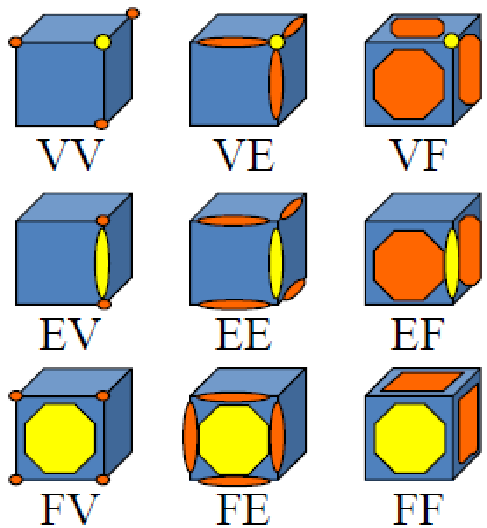


1. Vertex $\mapsto$ one outgoing halfedge,
2. Face $\mapsto$ one halfedge,
3. Halfedge $\mapsto$ target vertex,
4. Halfedge $\mapsto$ its face,
5. Halfedge $\mapsto$ next halfedge,
6. Halfedge $\mapsto$ opposite halfedge (implicit),
7. Halfedge $\mapsto$ previous halfedge (optional).

All basic queries take constant O(1) time!

How HDS can -- OpenSG





All basic queries take constant O(1) time!

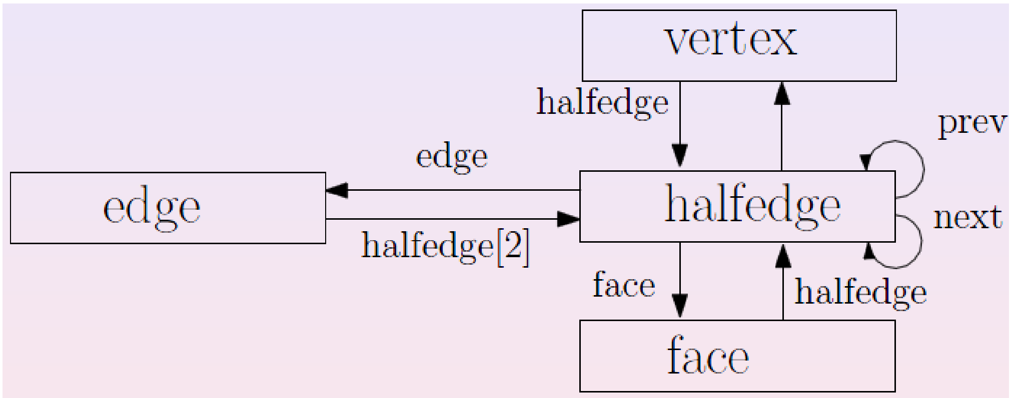
Attributes

- Each object stores attributes (traits) which defines other structures on the mesh:
  - metric structure: edge length
  - angle structure: halfedge
  - curvature : vertex
  - conformal factor: vertex
  - Laplace-Beltrami operator: edge
  - Ricci flow edge weight; edge
  - holomorphic 1-form: halfedge

HE of David Gu

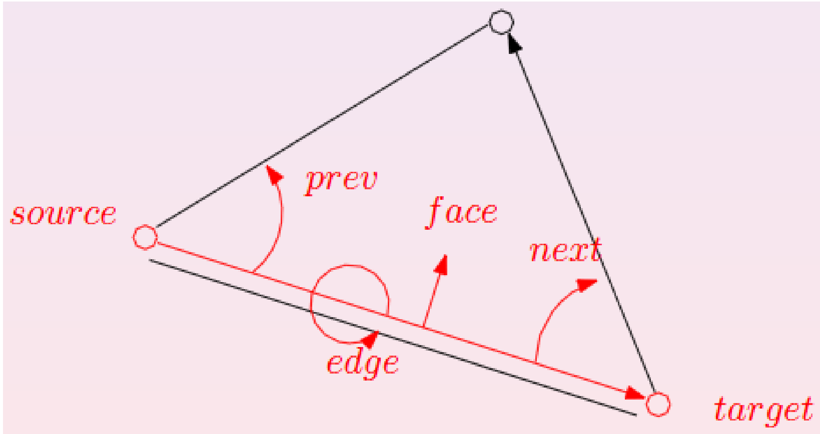
Halfedge data structure

- Fundamental classes
  - Vertex
  - Halfedge, oriented edge
  - Edge, non-oriented edge
  - Face, oriented
- All objects are linked together through pointers, such that
  - The local Eucler operation can be easily performed
  - The memory cost is minimized



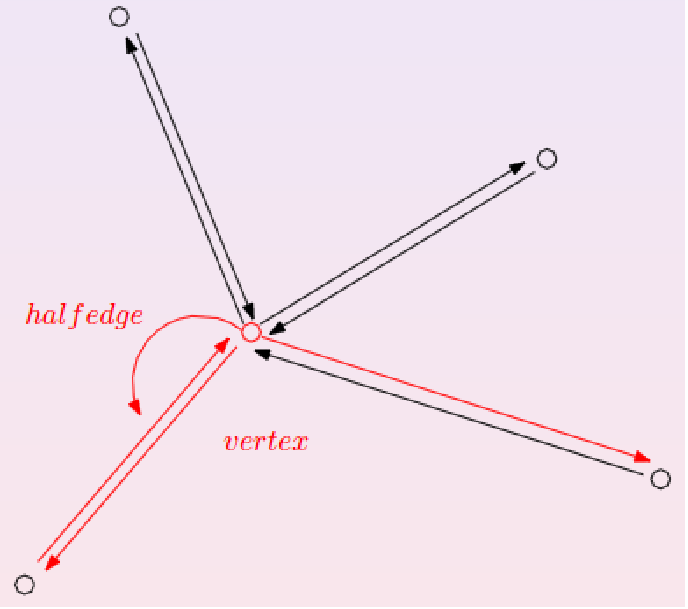
Halfedge class

- Pointers
  - Halfedge pointers: prev, next halfedge;
  - Vertex pointers: target vertex, source vertex;
  - Edge pointer: the adjacent edge;
  - face pointer: the face it belongs to;



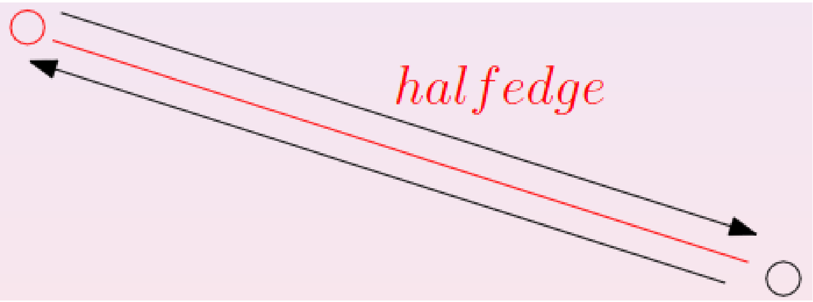
Vertex class

- Pointers
  - Halfedge pointers: the first in halfedge



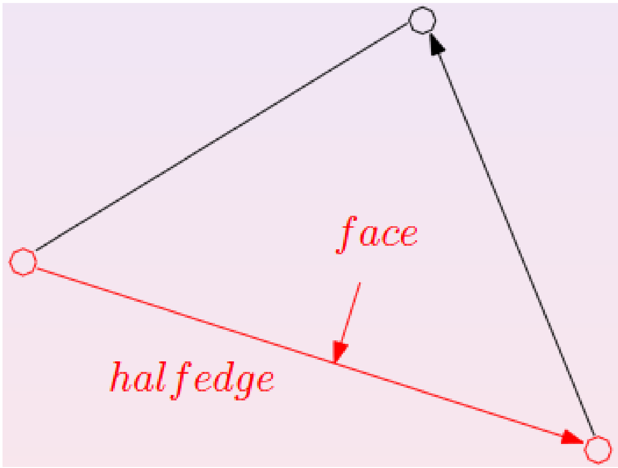
Edge class

- Pointers
  - Halfedge pointers: to the adjacent two halfedges.
  - if the edge is on the boundary, then the second halfedge pointer is null.



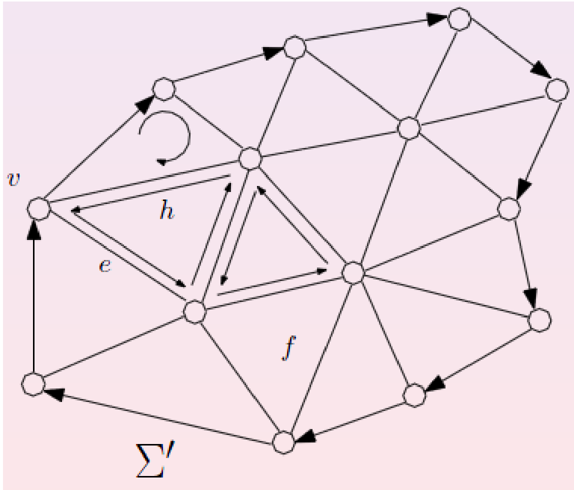
Face class

- Pointers
  - Halfedge pointers: to the first halfedge.



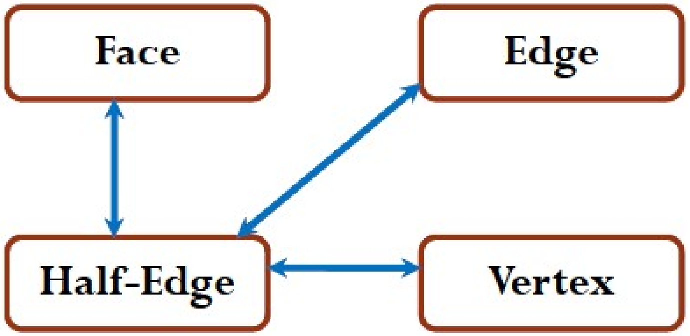
Mesh class

- Data members
  - A list of vertices;
  - A list of halfedges;
  - A list of edges;
  - A list of faces;



Vertex List:	**Half-Edge List:**	**Face List:**
Vertices V	Edges E	Faces F
x_{11}, y_{11}, z_{11}; e_1	v_1; f_1; o_1, n_1, p_1	e_1
...	...	...
x_{V1}, y_{V1}, z_{V1}; e_V	v_E; f_E; o_E, n_E, p_E	e_F

Relationship between primitives



HDS Design requirements 1

Represent each of the mesh items **explicitly**

- in order to be able to attach additional attributes and functionality to them.
- If not, then **cumbersome** and **error-prone**!

- Traditional
- Object-based (Encapsulation**)**
- Object-oriented (Inheritance and Polymorphism)
- Generic Programming (Template, compile-time vs. run-time, virtual functions lead to a certain overhead in space and time)
- Aspect-oriented Programming
- Design pattern (multiple inheritances vs. template, etc.) [能力从继承来，从组合来，…]

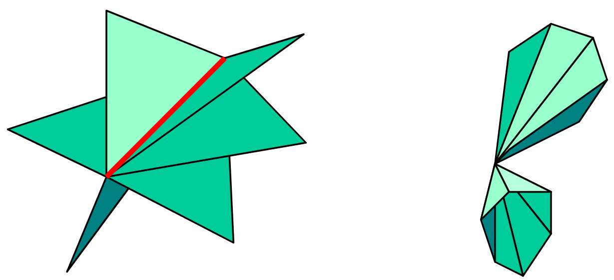
HDS Design requirements 2

General polygonal facet type

- Triangle + quadrangle + …

non-manifold degeneracies

To provide this flexibility, meshes can be custom-tailored for the needs of specific algorithms by parameterizing them by so-called **traits** class.



HDS Interface-specify a mesh

- **Item**

  Face (polygonal or triangle), edge, vertex

- **Kernel (HDS)**

  The mesh kernel is responsible for storing the mesh items internally.

- **Traits**: Traits allow enhancing mesh items by arbitrary functionality.
  - These **user-defined classes** are added to the corresponding mesh items in terms of **inheritance**.
  - The traits class also selects the **coordinate type** and the **scalar type** of the mesh, e.g. 2D-, 3D- vectors and float, double arithmetic.

HDS Interface-specify a mesh 1

```
#include <ACG/Mesh/TriMesh_ArrayKernelT.hh>

struct MyTraits : public DefaultTraits
{
  template <class Base> class VertexT : public Base
  {
  public:
    const Vec3f& cog() const { return cog_; }
    void   set_cog(const Vec3f& cog) { cog_ = cog; }
  private:
    Vec3f cog_;
  };
};

typedef TriMesh_ArrayKernelT<MyTraits> MyMesh;
```

HDS Interface - visit mesh items

Handle type: indices or pointers

Iterators: enumerate all mesh items of one type. (STL, begin(), end())

Circulators: access one-ring neighborhood

Circulator – Halfedge_vertex_circulator

```
template < class It, class Ctg>
class I_HalfedgeDS_vertex_circ : public It {…
Self& operator++() {
    *((Iterator*)this) = (*this)->next()->opposite();
    return *this;   }
…}

I_HalfedgeDS_vertex_circ< Halfedge_handle, circulator_category>
                              Halfedge_around_vertex_circulator;
Halfedge_const_handle get_halfedge( Vertex_const_handle source, target) {
  Halfedge_around_vertex_circulator cir=target->vertex_begin(), cir_end = cir;
  CGAL_For_all(cir, cir_end)
    if (cir->opposite()->vertex() == source)
      return cir;
}
```

Implementation – highly customizable and efficiency

The low-level implementation of the HDS is **encapsulated** into the **mesh kernel**:

- **Store and access** mesh items

  Through handles, iterators, circulators

- Keep the connectivity info **consistent**

- **Higher-level Functionality**

  topological operators, etc.

Implementation – highly customizable and efficiency

- Trouble? How to design algorithms operating on all of these mesh types?

  All these custom-tailored meshes will be different C++ types.

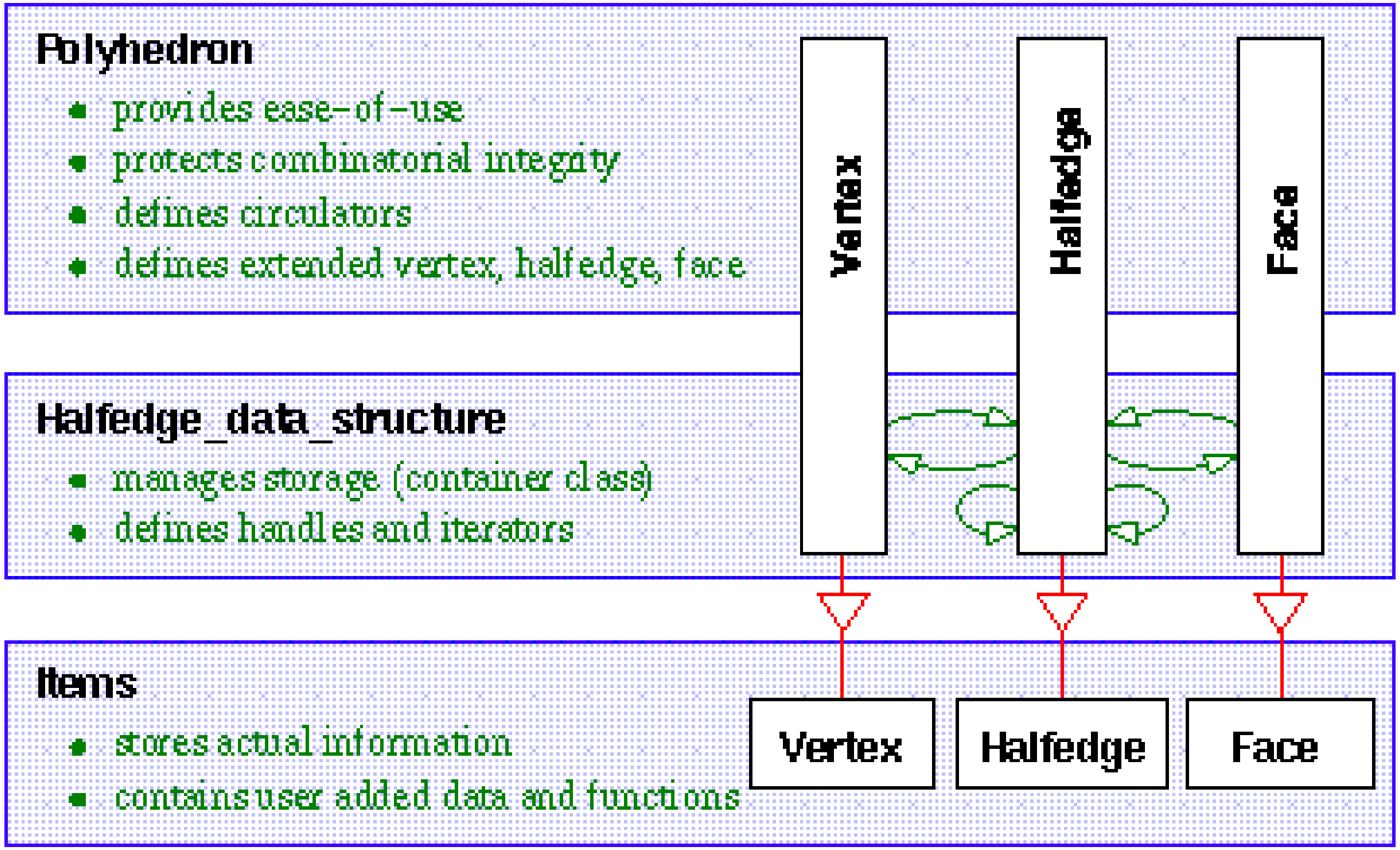
- virtual base class vs. generic programming methods

  Runtime vs. compile time

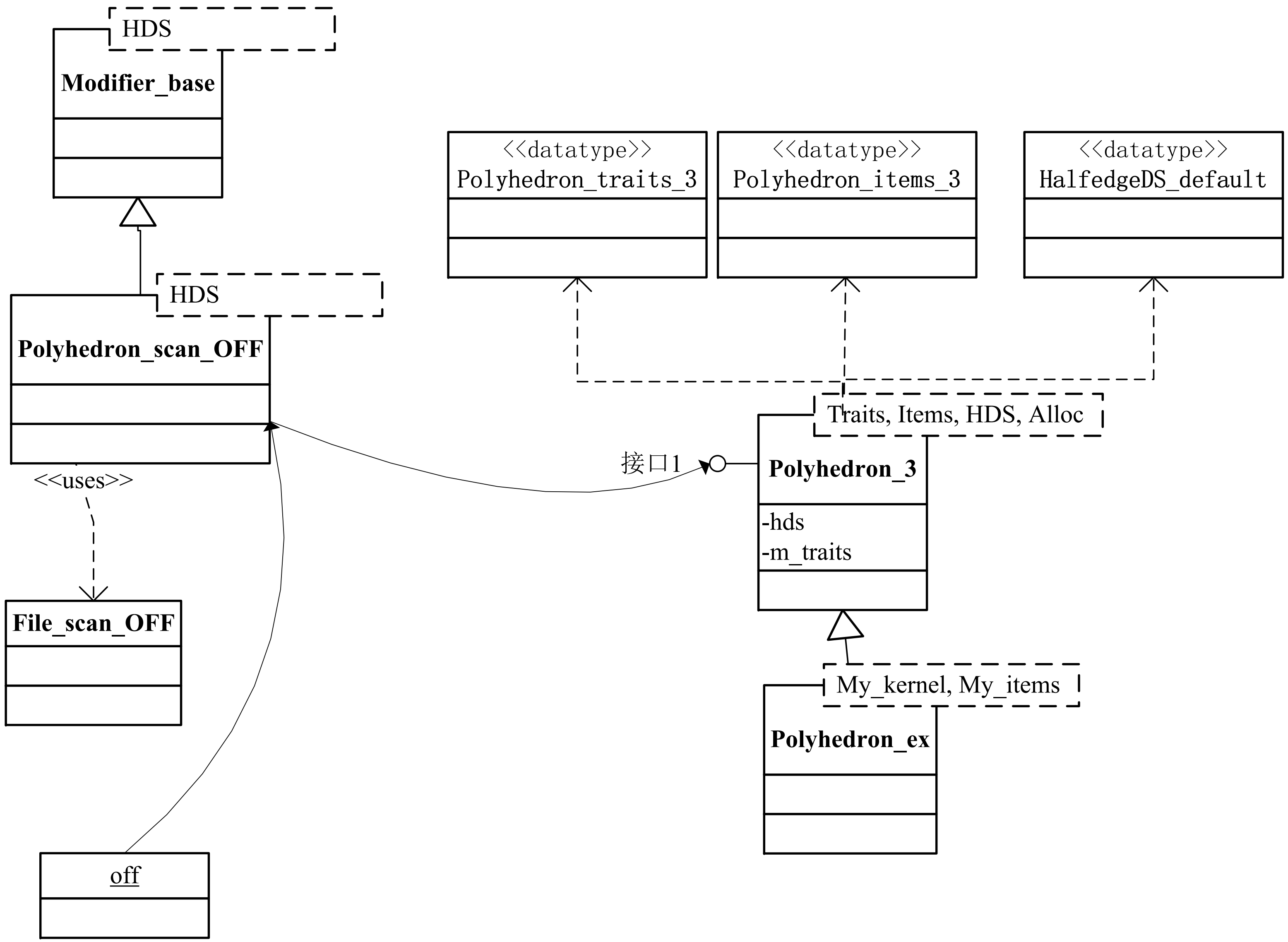
```
struct Tag_true  {};
  typedef typename Vertex::Base              VBase;
  typedef typename         VBase::Support_prev_halfedge Support_prev_halfedge;     //Tag_true for supporting

  find_prev( Halfedge_const_handle h, Tag_true) // no if else
  find_prev( Halfedge_const_handle h, Tag_false) //
```

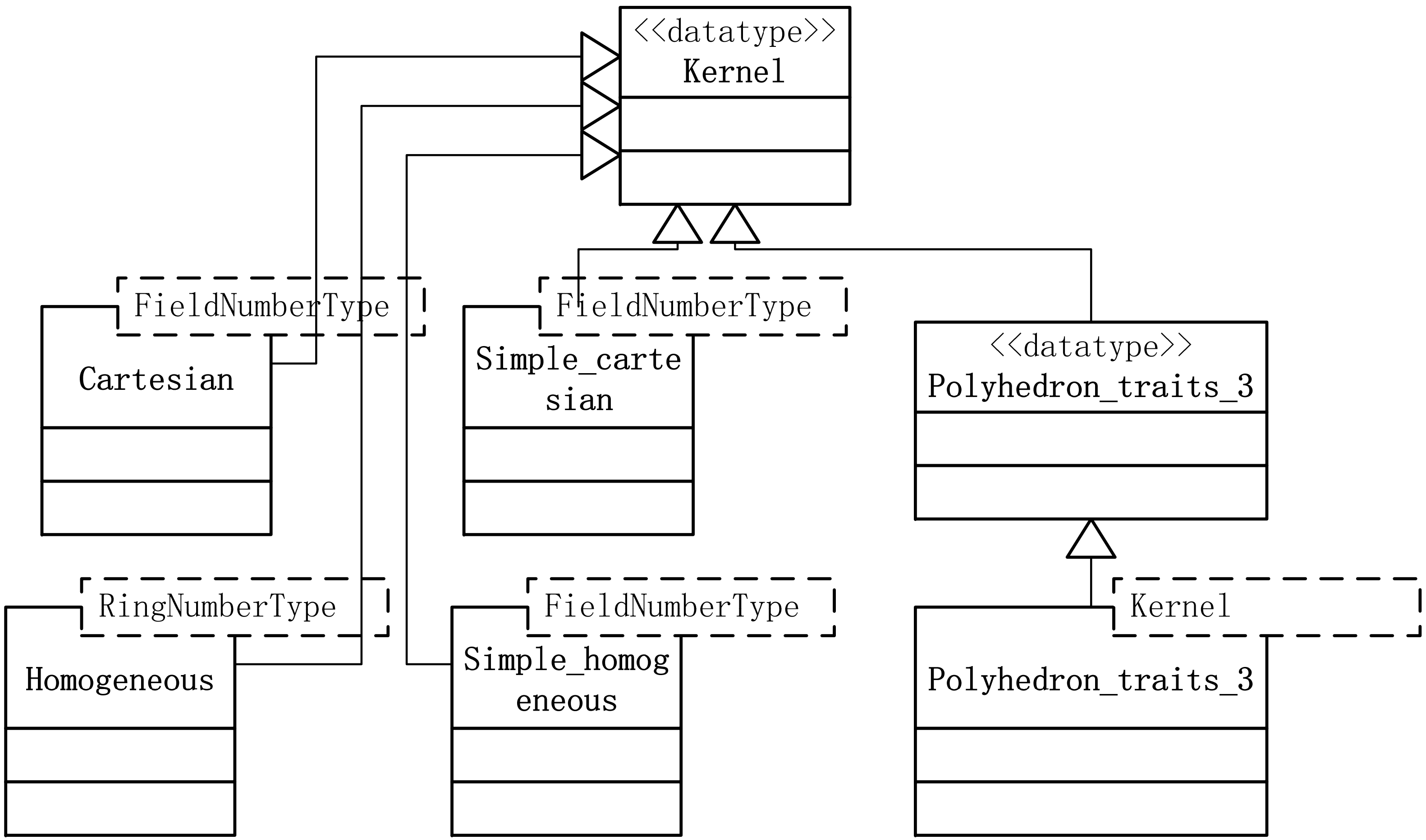
- As attributes are only allocated when actually needed, no **memory** is wasted for unused attributes.
- **template forward declarations**

Half-edge in CGAL



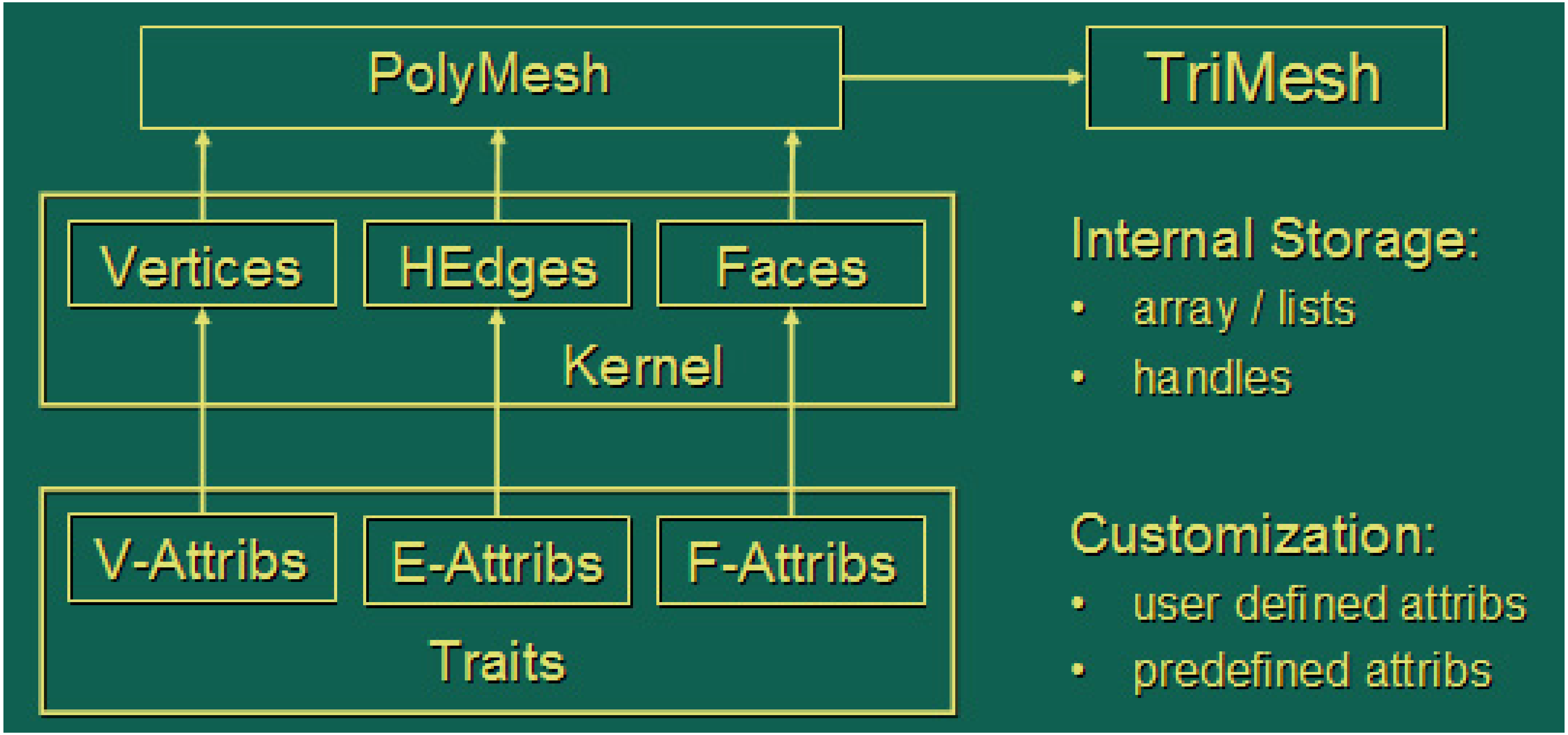
No support for **specialized mesh kernels** such as **quad-tree**, which is necessary for efficient subdivision implementation. **non-manifold**


HDS
Modifier_base
HDS
Polyhedron_scan_OFF
<<uses>>
File_scan_OFF
off
<<datatype>>
Polyhedron_traits_3
<<datatype>>
Polyhedron_items_3
<<datatype>>
HalfedgeDS_default
Traits, Items, HDS, Alloc
接口1
Polyhedron_3
-hds
-m_traits
My_kernel, My_items
Polyhedron_ex



Polyhedron_traits_3 is just a subset concept of concept Kernel. So any 3d kernel can be used in Polyhedron<Polyhedron_traits_3> as a traits directly, for example, Polyhedron<Cartesian<double> >.

Half-edge in OpenMesh

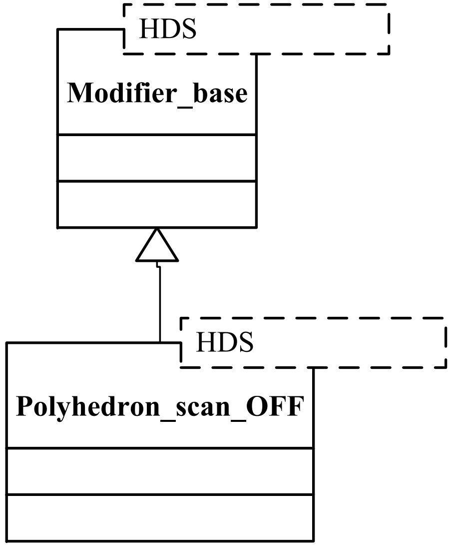


Create from scratch

```
#include <CGAL/Simple_cartesian.h>
#include <CGAL/Polyhedron_3.h>

typedef CGAL::Simple_cartesian<double>     Kernel;
typedef CGAL::Polyhedron_3<Kernel>         Polyhedron;
typedef Polyhedron::Halfedge_handle        Halfedge_handle;

int main() {
   //Point_3 p( 0.0, 0.0, 0.0); Point_3 q( 1.0, 0.0, 0.0);
   //Point_3 r( 0.0, 1.0, 0.0);  Point_3 s( 0.0, 0.0, 1.0);
   Polyhedron P;
   P.make_tetrahedron();// P.make_tetrahedron(p, q, r, s)
   if ( P.is_tetrahedron(h))
      return 0;
   return 1;
}
```

IO -- Create from off

```
std::ifstream stream(input_filename);
Polyhedron mesh;
stream >> mesh;
mesh.compute_type();//pure triangle or quad
mesh.compute_normals();
```




```
Build_Mesh<Polyhedron, Polyhedron::HDS> bm(stream);
Polyhedron *mesh = new Polyhedron();
mesh->delegate(bm);
```

Refer: D:\CGAL-4.2\examples\Polyhedron_IO

Examples: mesh_io

Examples -- Create from off -- Detail

```
template <class Polyhedron, class HDS>
class Build_Mesh : public CGAL::Modifier_base<HDS> {//delegate
    typedef CGAL::Polyhedron_incremental_builder_3<HDS>      builder;
    void operator()( HDS& hds) {
        builder B( hds, true);
        int n_h = int( (vhs_.size() + f_indexs_.size() - 2 + 12) * 2.1);

        B.begin_surface( vhs_.size(), f_indexs_.size(), n_h);
        add_vertices(B);    add_facets(B);
        B.end_surface();
    }
    void add_vertices(builder &B){
            …
        for ( int i = 0; i < (int)vhs_.size(); ++i) {
          Vertex_handle ovh = vhs_[i];
           Point p = ovh->point();
           Vertex_handle vh = B.add_vertex( p);
        }
    }
}
```

```
void add_facets(builder &B){
…
        B.begin_facet();
        B.add_vertex_to_facet( fv_ind[0]);
        B.add_vertex_to_facet( fv_ind[1]);
        B.add_vertex_to_facet( fv_ind[2]);
        B.end_facet();
    }
```

Examples -- Visit mesh elements

访问所有半边

```
for(Halfedge_iterator pHe = P.halfedges_begin();pHe!= P. halfedges _end(); ++ pHe)
{
Halfedge_handle eh = pHe;
}
```

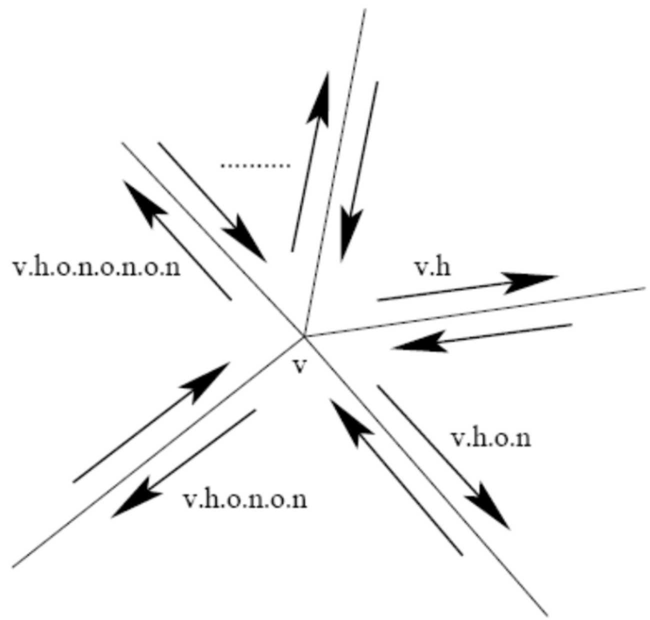
访问所有面

```
for(Facet_iterator pFacet = P.facets_begin(); pFacet != P.facets_end(); ++pFacet)
{
   Facet_handle fh = pFacet;
   Halfedge_facet_circulator j = fh->facet_begin();
   std::cout << "Begin a surface" << std::endl;
   do {
         std::cout << '(' << j->vertex()->point() << ");"; //输出每一个点
      } while ( ++j != fh->facet_begin());
   std::cout << std::endl << "End a surface" << std::endl;
}
```

Examples -- Navigate 1-ring of a Vertex

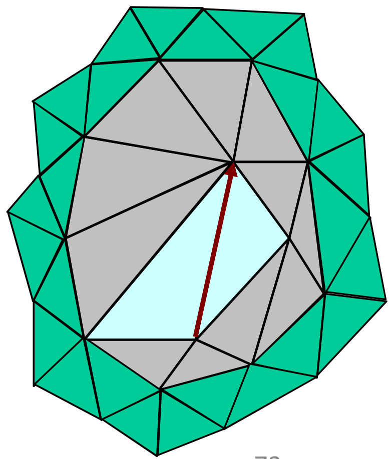
```
// Vertex_iterator可以付值给Vertex_handle
Vertex_handle pVertex；
Halfedge_vertex_circulator  pHalfEdge  =  pVertex->vertex_begin();
Halfedge_vertex_circulator       end  =  pHalfEdge;

CGAL_For_all(pHalfEdge,end)
{
  //取每个半边对应的边的两个顶点
  Point_3 p1 = pHalfEdge->vertex()->point();
  //这里用到了取反边的操作。
  Poing_3 p2 = pHalfEdge->opposite()->vertex()->point();
  Facet_handle f_h = pHalfEdge->facet()
}
```

How to navigate 1-ring vertices of a vertex?

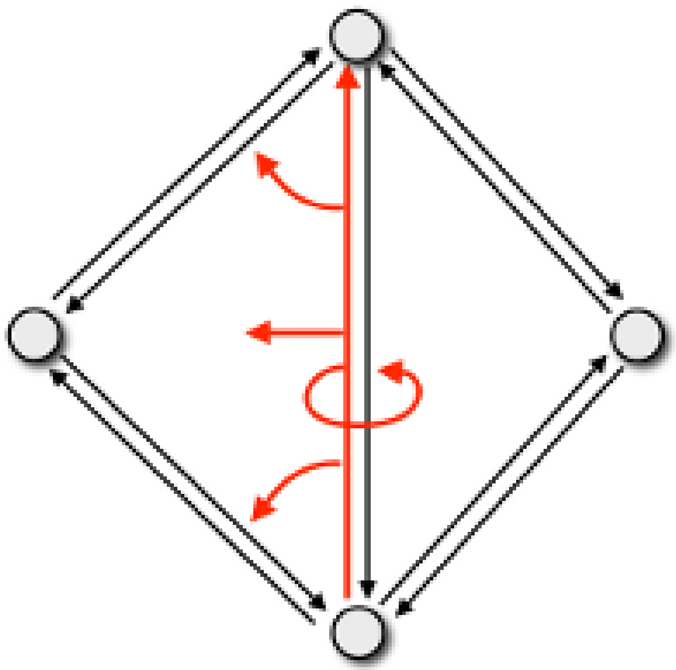


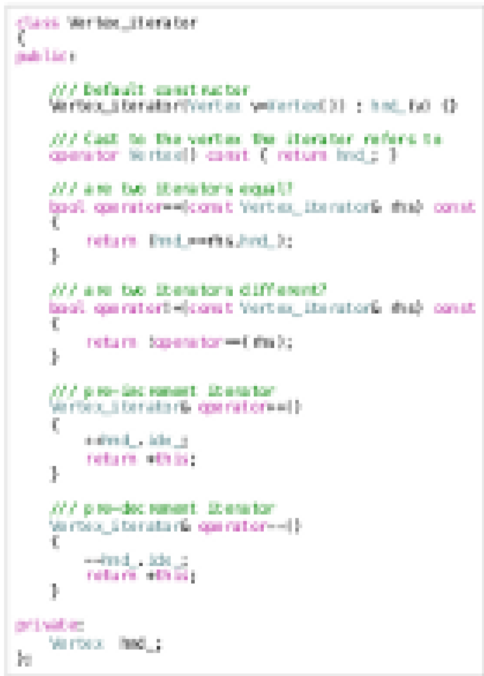
Examples -- Navigate 0,1,2-ring of a Facet

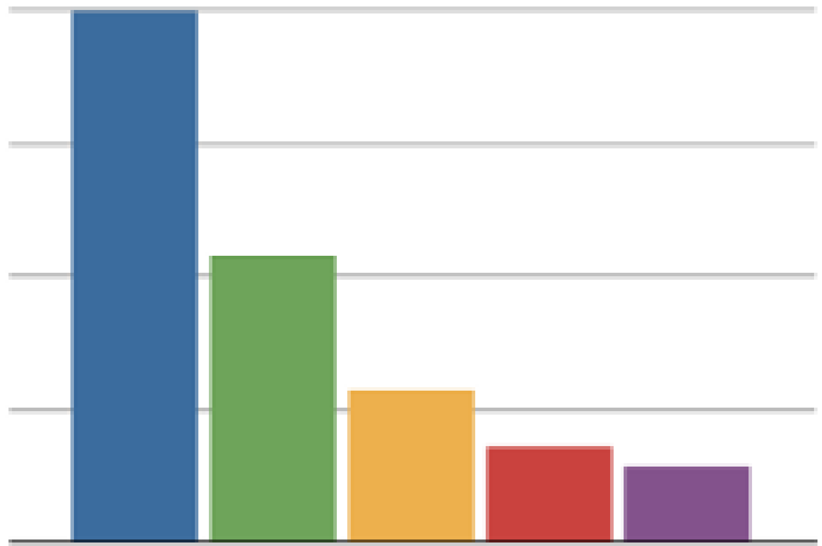
```
Halfedge_facet_circulator j = fh ->facet_begin();
std::vector<Face_handle> hh_set;
do {//访问fh的个顶点
    Vertex_handle pVertex = j->vertex();
    Halfedge_vertex_circulator  pHalfEdge  =  pVertex->vertex_begin();
    Halfedge_vertex_circulator      end  =  pHalfEdge;
    //通过访问pVertex的每个邻半边，达到访问pVertex的一环邻面的目的。这里必然有重复推进
    CGAL_For_all(pHalfEdge,end)
    {
            Face_handle f_h = pHalfEdge->facet();
            std::vector<Face_handle>::iterator itv = std::find(hh_set.begin(), hh_set.end(), f_h);
            if ( hh_set.end() ==  itv)//!没找到
            {
                    hh_set.push_back(fh);
            }
    }
} while ( ++j != fh->facet_begin());
```



Design, Implementation, and Evaluation of the Surface_mesh Data Structure







Discussion

- Say a word

Resources

- *https://github.com/jjcao/jjcao_code.git*
- SourceTree

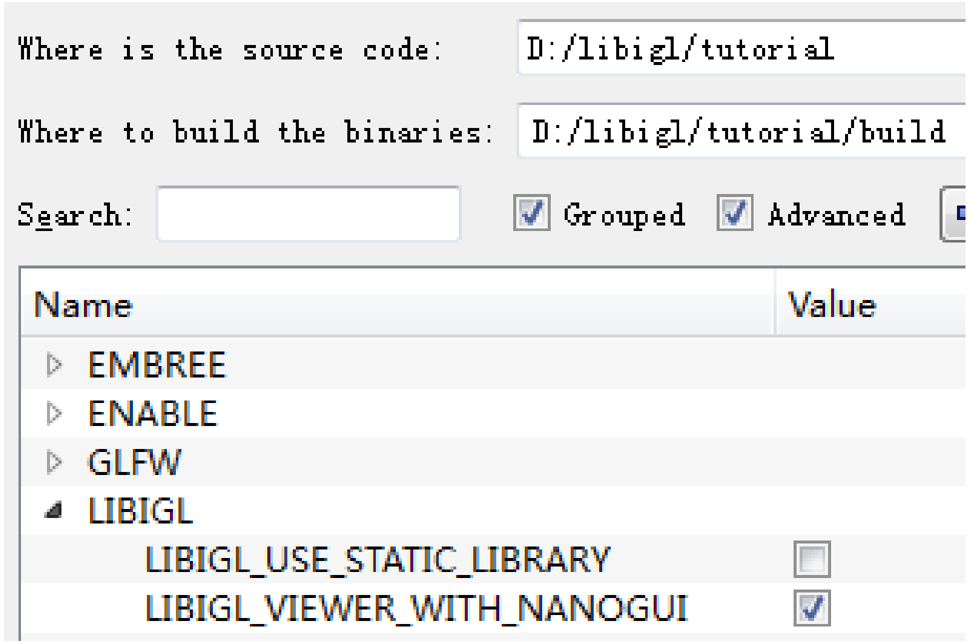
- Gabriel Peyre's numerical tour!

- Wiki
- OFF file format specification

- Andrew Nealen: CS 523: Computer Graphics : Shape Modeling
- Xianfeng Gu, lecture_8_halfedge_data_structure

Environment – c++

- Visual studio 2015 community
- CMAKE
- Python 3
- CGAL
  - Boost
  - Qt
  - libQGLViewer (cool example for picking)
- Eigen
- Libigl (use cmakegui to generate vc solution: Visual Studio 14 2015 Win64,)
  - CoMISo
  - Nanogui (build it first, then cmake libigl)
  - Embree (for picking) (copy bin and lib to D:\libigl\external\embree, then cmake again)



Environment - Matlab

- Matlab 2015b
- jjcao_code: https://github.com/jjcao/jjcao_code.git

Lab

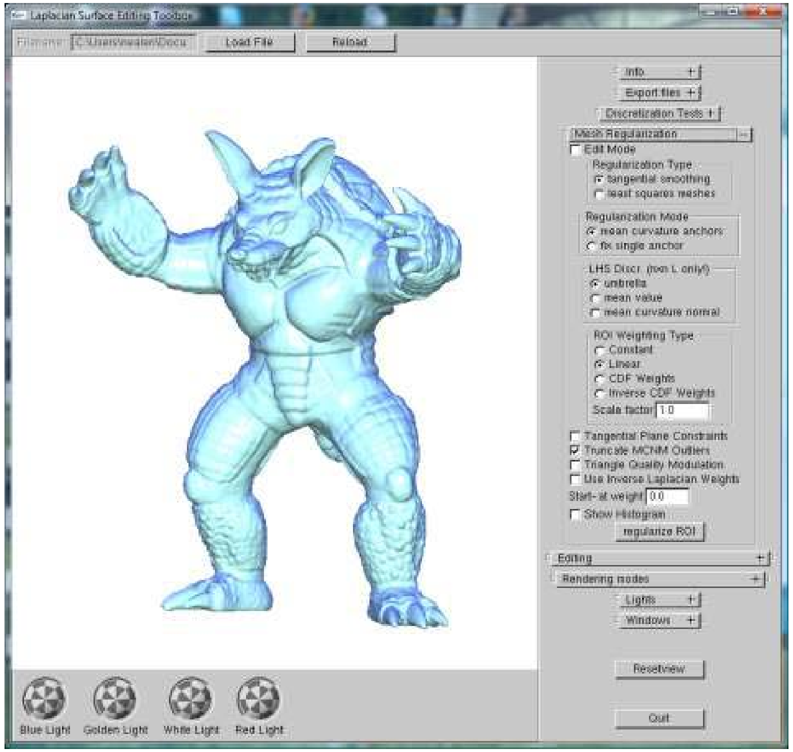
- Lab1
  - **Chapter 1 of libigl tutorial** or jjcao_code\toolbox\jjcao_plot\eg_trisurf.m
- Lab2 [optional]
  - See User manual of Halfedge Data Structures of CGAL
  - run the examples or jjcao_code\toolbox\jjcao_mesh\datastructure\test_to_halfedge.m

The end

Old assignment

Assignment 1: Mesh processing “Hello World”

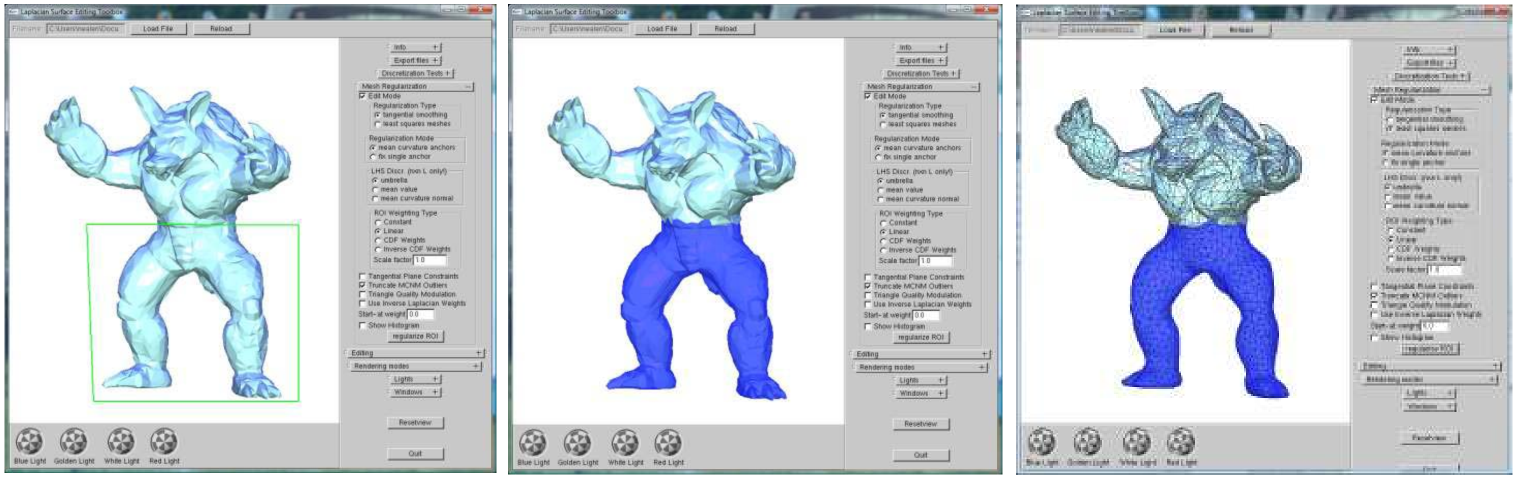
- Goals: learn basic mesh data structure programming + rendering (flat/gouraud shaded, wireframe) + basic GUI programming
- by **MATLAB** or **VC**



You can ask the help from school senior!

Assignment 2: selection + operation tools

- Goals: implement image-space selection tools and perform local operations (smoothing, etc.) on selected region
- VC



Final Project

- Implementation/extension of a space or surface based editing tool
  - makes use of assignments 1 + 2
  - Your own suggestion, with instructor approval
- Includes written project report & presentation
  - Latex style files will be provided?
  - Power Point examples will be provided?