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.

Last time: overview of geometry

- **Many types of geometry in nature**
- **Demand sophisticated representations**
- **Two major categories:**
  - **IMPLICIT - “tests” if a point is in shape**
  - **EXPLICIT - directly “lists” points**
- **Lots of representations for both**

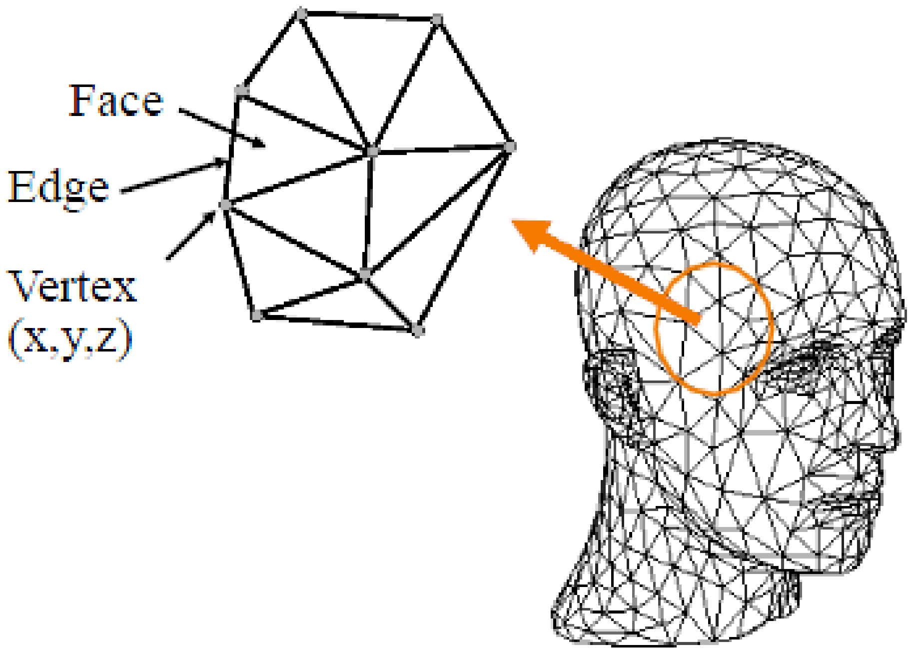
What is a Mesh?

What is a Mesh?

- A Mesh is a pair *(P,K)*, where *P* is a set of point positions $P = \{ p_i \in R^3 \mid 1 \le i \le n \}$ and *K* is an abstract simplicial complex which contains all topological information.
- *K* is a set of subsets of $\{1, \ldots, N\}$
  - Vertices
  - Edges
  - Faces

$$v = \{i\} \in V$$
$$e = \{i, j\} \in E$$
$$f = \{i_1, i_2, \ldots, i_{n_f}\} \in F$$

$$K = V \cup E \cup F$$

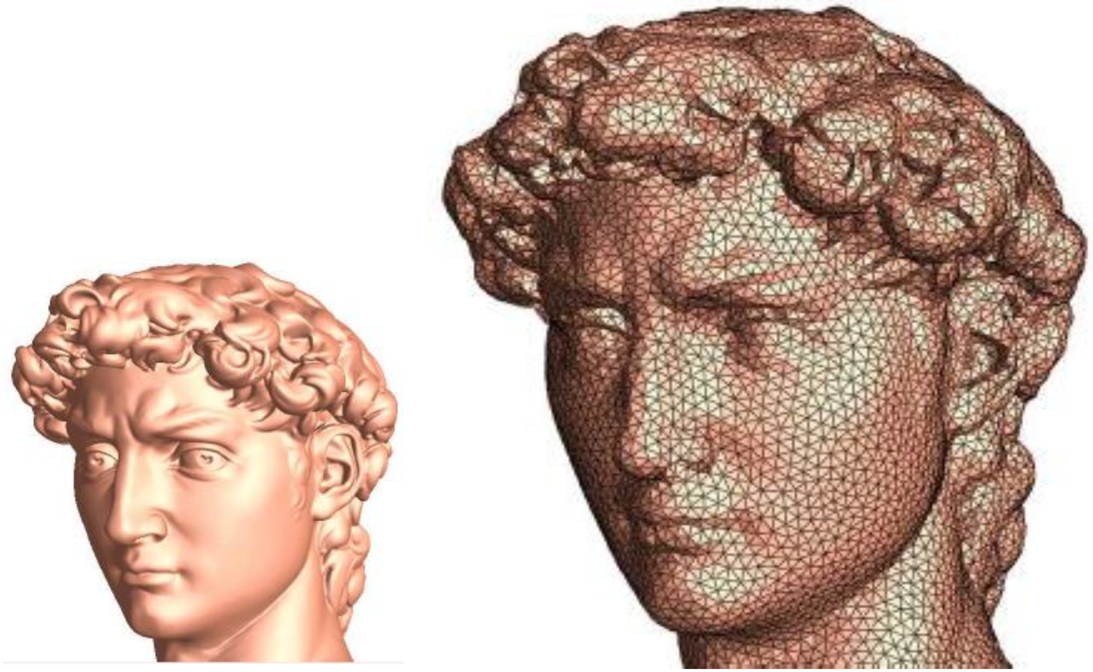


- A **Graph** is a pair G=(V,E)
  - Degree or valence of a vertex

Zorin & Schroeder

Polygonal Meshes

- Topology
  - Simplicial Complex, Combinatorics
  - connectivity of the vertices
- Geometry
  - Conformal Structure – Corner angles (and other variant definitions)
  - Riemannian metrics – Edge lengths
  - Embedding – Vertex coordinates



What is a Mesh?

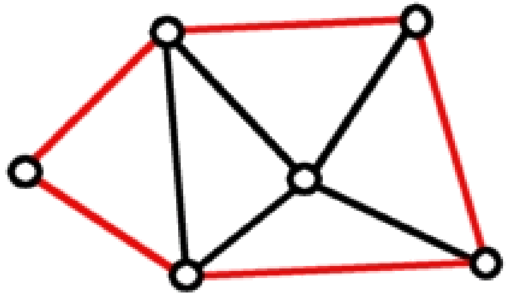
- Each edge must belong to at least one face, i.e.

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

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

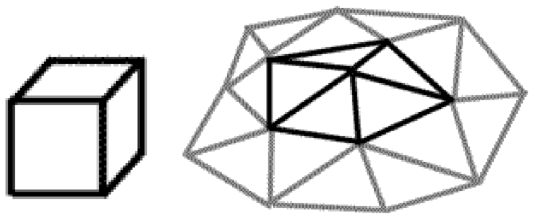
$$v = \{j\} \in V \text{ iff } \exists\, e = \{i, j\} \in E$$

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

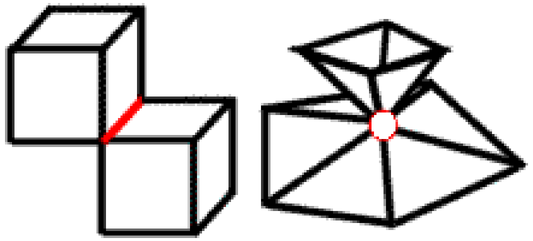


What is a Mesh?

- 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

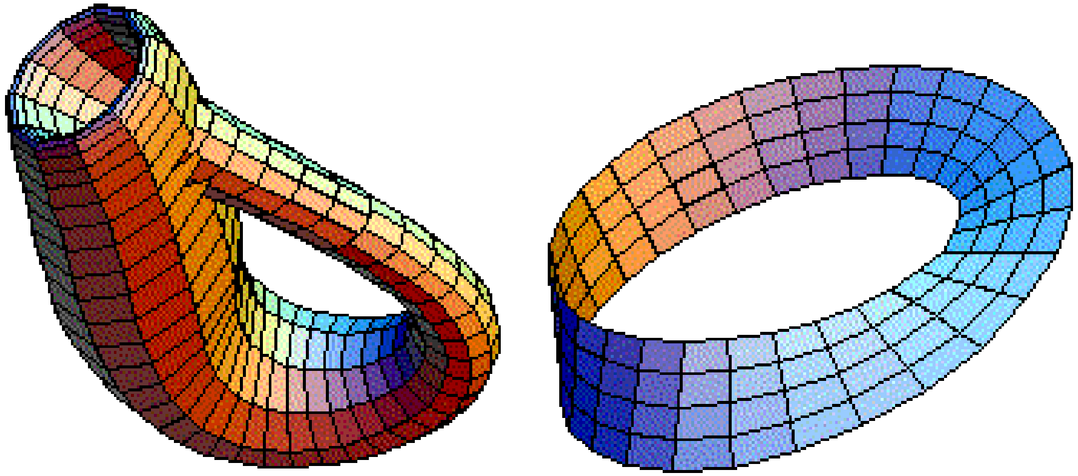
- A mesh is a polyhedron if
  - It is a manifold mesh and it is closed (no boundary)
  - Every vertex belongs to a cyclically ordered set of faces (local shape is a disk)

Where Meshes Come From

- Model manually
  - Write out all polygons
  - Write some code to generate them
  - Interactive editing: move vertices in space
- Acquisition from real objects
  - 3D scanners, vision systems
  - Generate set of points on the surface
  - Need to convert to polygons

Orientation of Faces

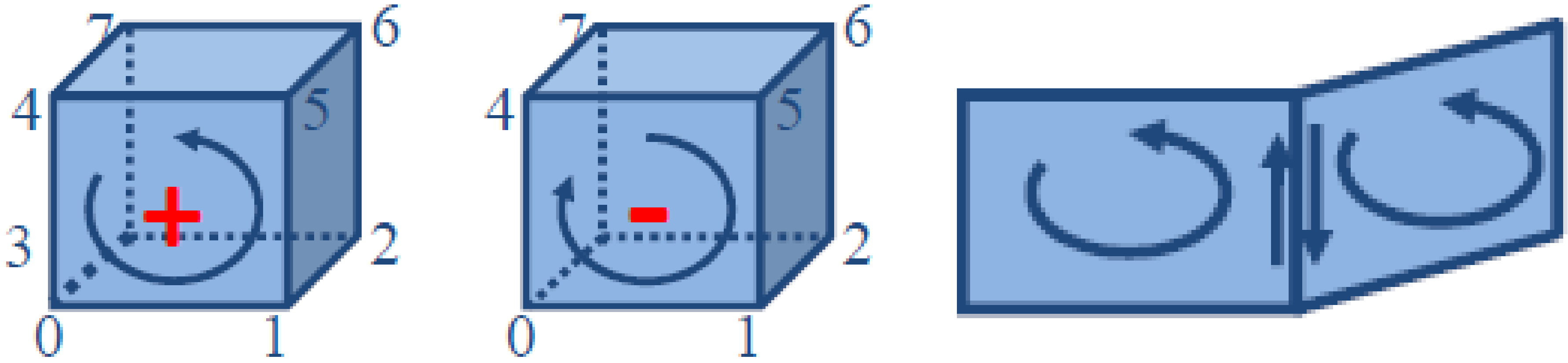
- A mesh is well oriented (orientable) if all faces can be oriented consistently (all CCW or all CW) such that each edge has two opposite orientations for its two adjacent faces
- Not every mesh can be well oriented.
  e.g. Klein bottle, Möbius strip



non-orientable surfaces

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.



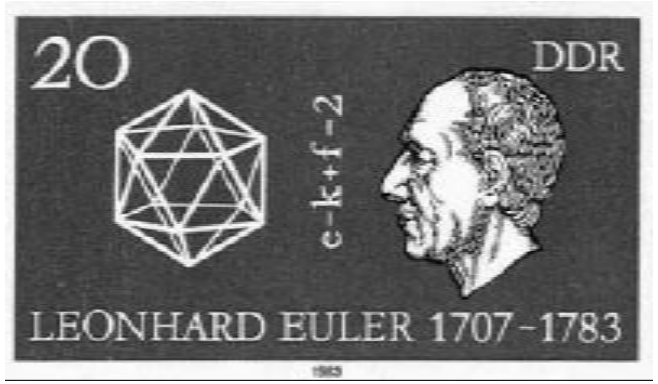
- Two neighboring facets are **equally oriented**, if the edge directions of the shared edge (induced by the face orientation) are opposing
- A polygonal mesh is **orientable**, if the incident faces to every edge can be equally oriented.

Euler-Poincaré Formula

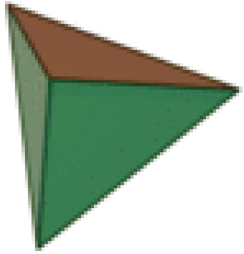
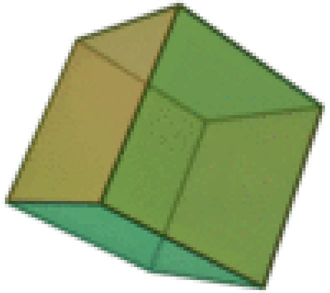
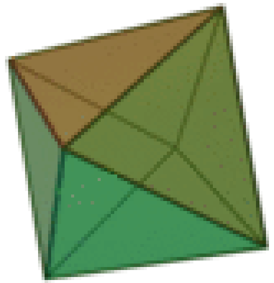
- The relation between the number of vertices, edges, and faces.

$$V - E + F = 2$$

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



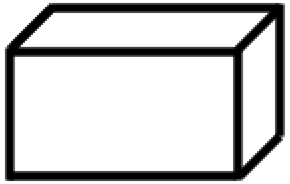
Euler Formula

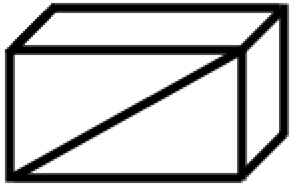
- Tetrahedron
  - V = 4
  - E = 6
  - F = 4
  - 4 - 6 + 4 = 2

- 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

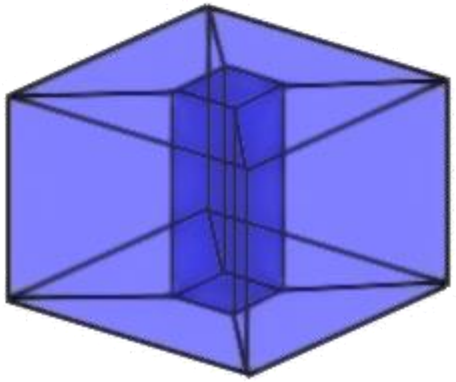


V = 8
E = 12 + 1 = 13
F = 6 + 1 = 7
8 - 13 + 7 = 2

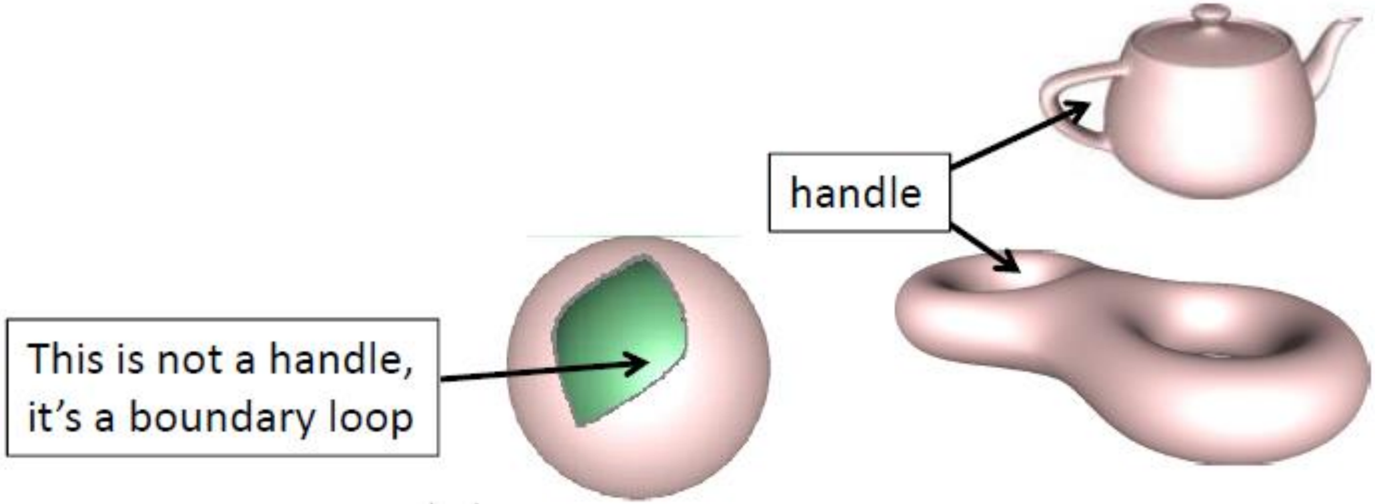
Euler-Poincaré Formula

- More general rule **Euler characteristic** χ**= 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



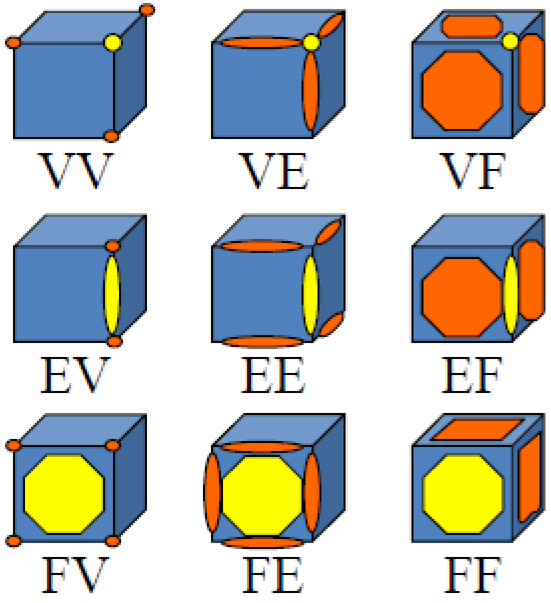
V = 16
E = 32
F = 16
C = 1
G = 1
B = 0
16 – 32 + 16 = 2 (1 - 1) - 0

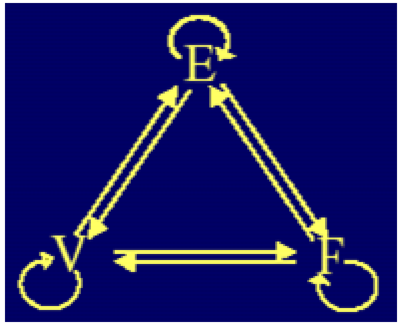


Data Structure

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

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

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

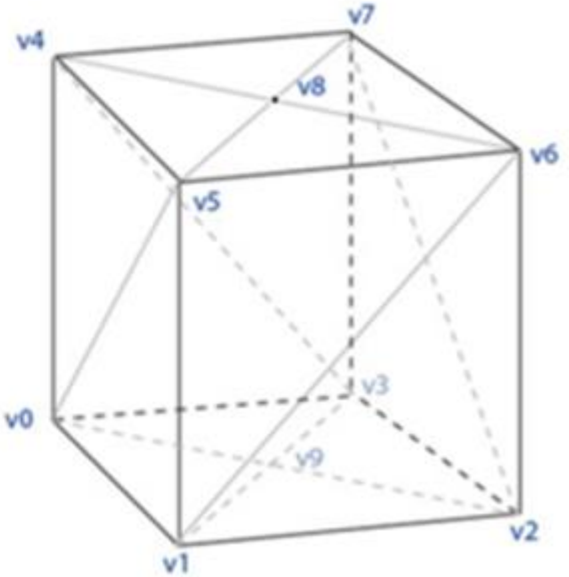
- Choice
  - Each of the representations above have particular advantages & drawbacks
  - Choice is governed by
    - Application,
    - Performance required,
    - Size of the data,
    - and Operations to be performed.
- Example
  - it is easier to deal with triangles than general polygons, especially 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 Set (STL)

- **Face:**
  - 3 vertex positions

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}

9*4 = 36 B/f (single precision)
72 B/v (Euler Poincaré)

No explicit connectivity

Shared Vertex (OBJ,OFF)

- **Indexed Face List:**
  - Vertex: position
  - Face: Vertex Indices

Vertices
$x_1 \; y_1 \; z_1$
...
$x_V \; y_V \; z_V$

Triangles
$i_{11} \; i_{12} \; i_{13}$
...
...
...
...
$i_{F1} \; i_{F2} \; i_{F3}$

12 B/v + 12 B/f = 36B/v

No explicit adjacency info

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**

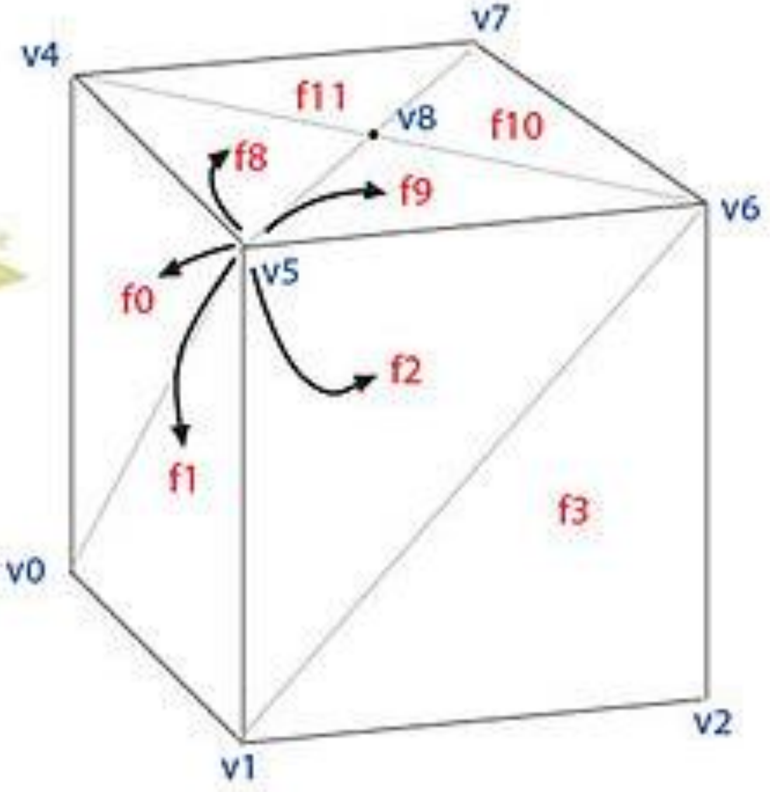
Face List

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

example



Face-vertex meshes

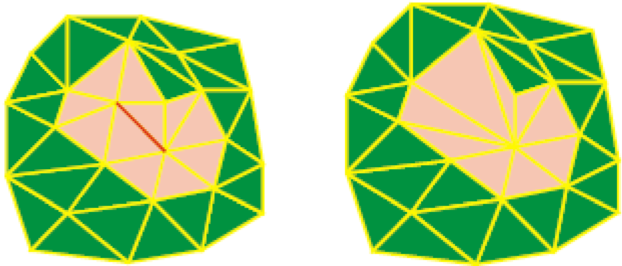
1. locating neighboring faces and vertices is constant time
2. a search is still needed to find all the faces surrounding a given face.
3. Other dynamic operations, such as splitting or merging a face, are also difficult with face-vertex meshes.

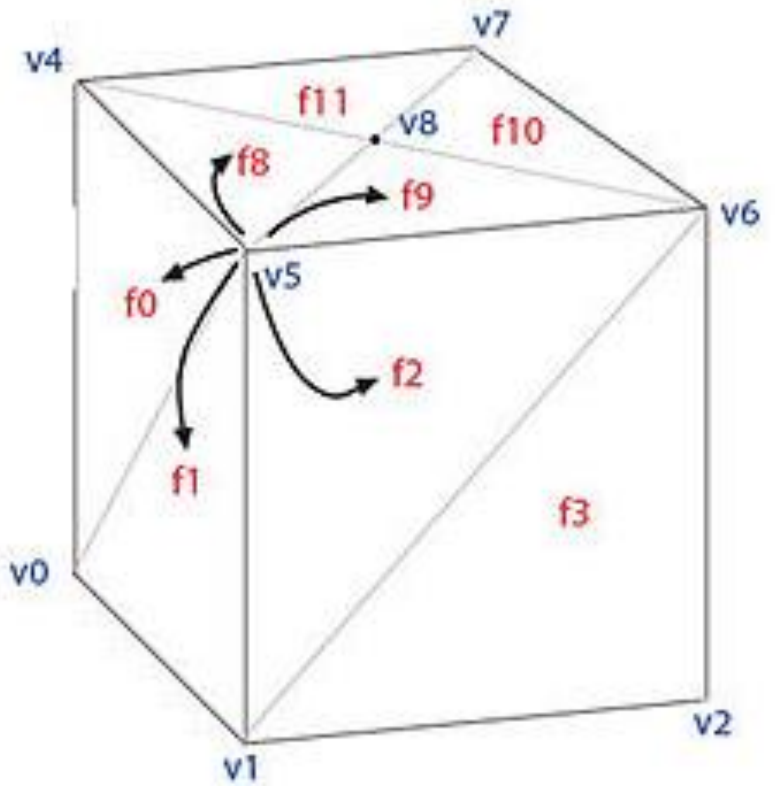
Face List

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





Transversal operations

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

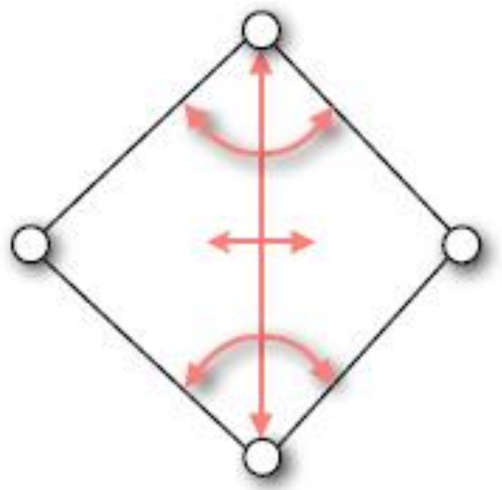
collect adjacent \ iterate over	V	E	F
V	quadratic	quadratic	linear
E	quadratic	quadratic	linear
F	quadratic	quadratic	linear

Edges always have the same topological structure

Efficient handling of polygons with variable valence

(Winged) **Edge-Based Connectivity**

- **Vertex:**
  - position
  - 1 edge
- **Edge:**
  - 2 vertices
  - 2 faces
  - 4 edges
- **Face:**
  - 1 edge

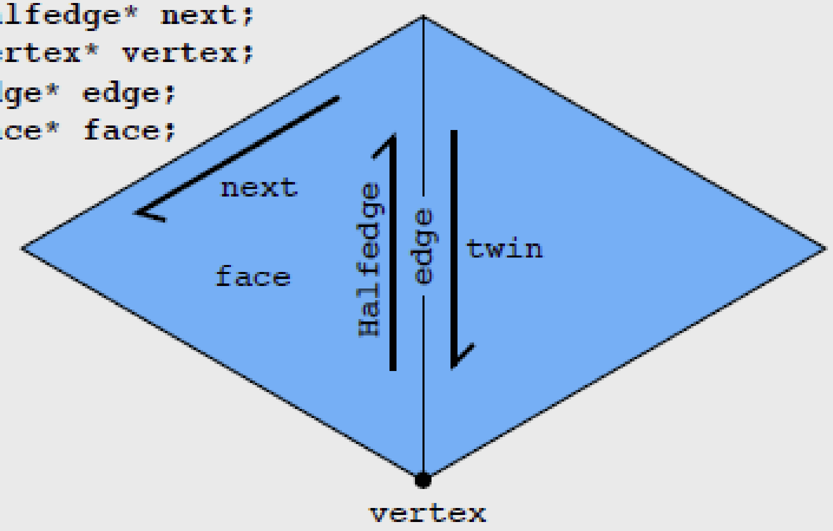


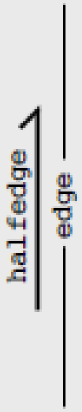
120 B/v

Edges have no orientation: special case handling for neighbors

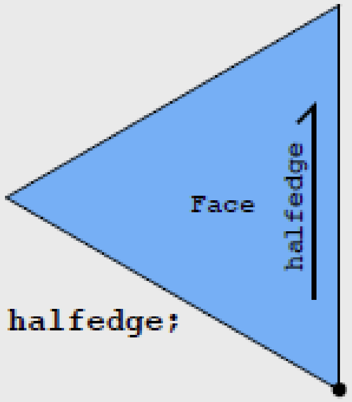
Halfedge-Based Connectivity

```
struct Halfedge
{
   Halfedge* twin;
   Halfedge* next;
   Vertex* vertex;
   Edge* edge;
   Face* face;
};
```

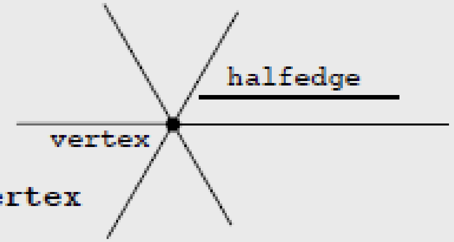




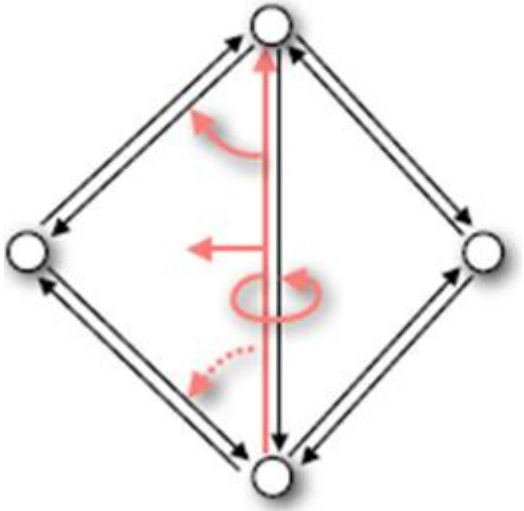

```
struct Edge
{
   Halfedge* halfedge;
};
```




```
struct Face
{
   Halfedge* halfedge;
};
```



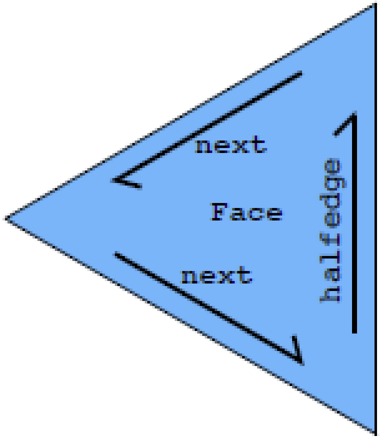

```
struct Vertex
{
   Halfedge* halfedge;
};
```



Halfedge makes mesh traversal easy

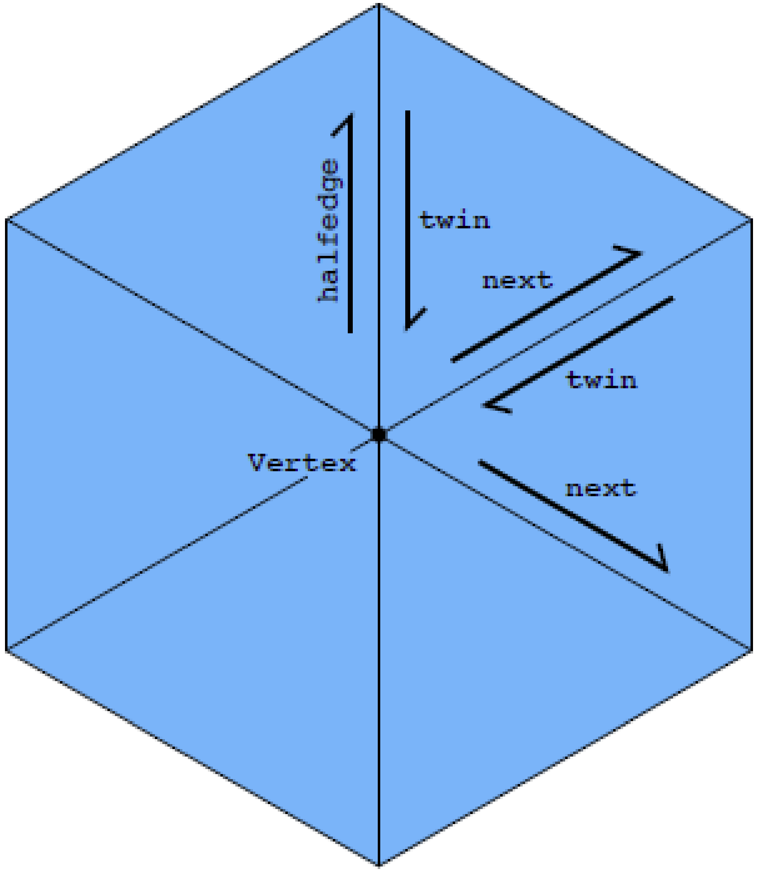
- **Use "twin" and "next" pointers to move around mesh**
- **Use "vertex", "edge", and "face" pointers to grab element**
- **Example: visit all vertices of a face:**

```
Halfedge* h = f->halfedge;
do {
   h = h->next;
}
while( h != f->halfedge );
```



- **Example: visit all neighbors of a vertex:**

```
Halfedge* h = v->halfedge;
do {
   h = h->twin->next;
}
while( h != v->halfedge );
```



Mesh operations

Traversals over all elements of certain type

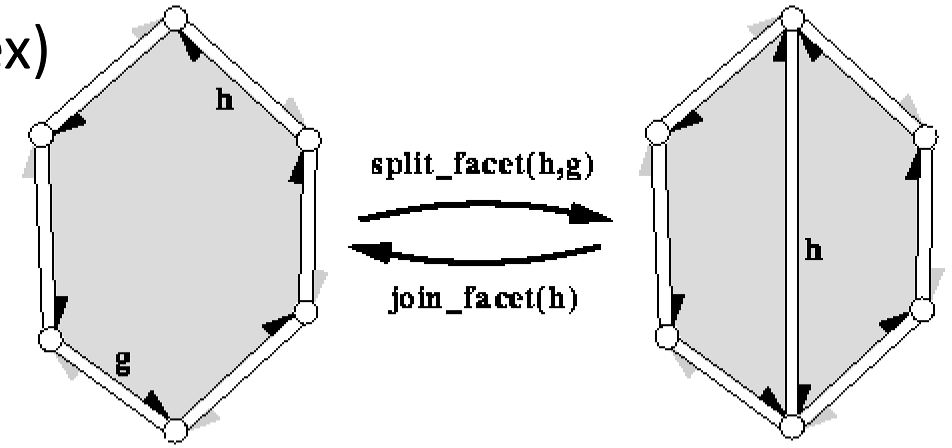
Navigate adjacent elements (e.g. one-ring of a vertex)

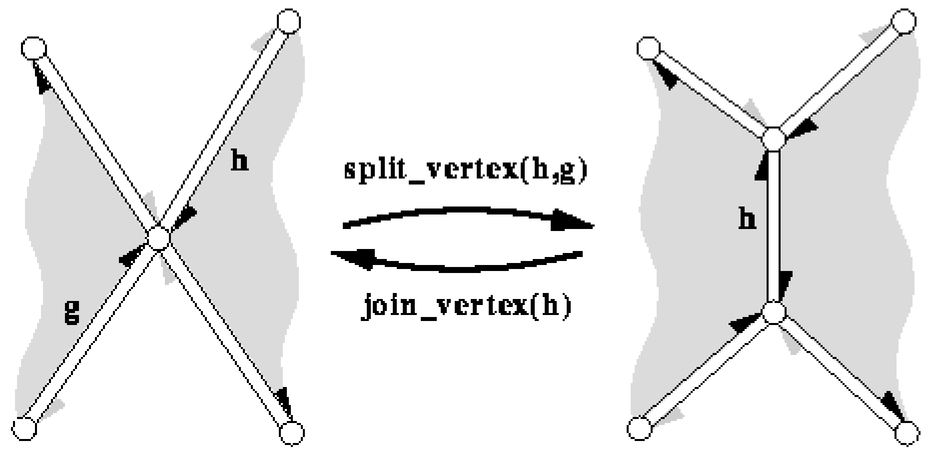
Refinement

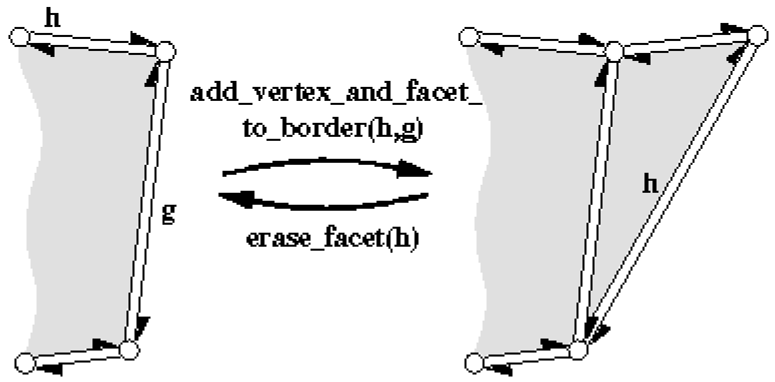
Edge flips

Face addition/deletion

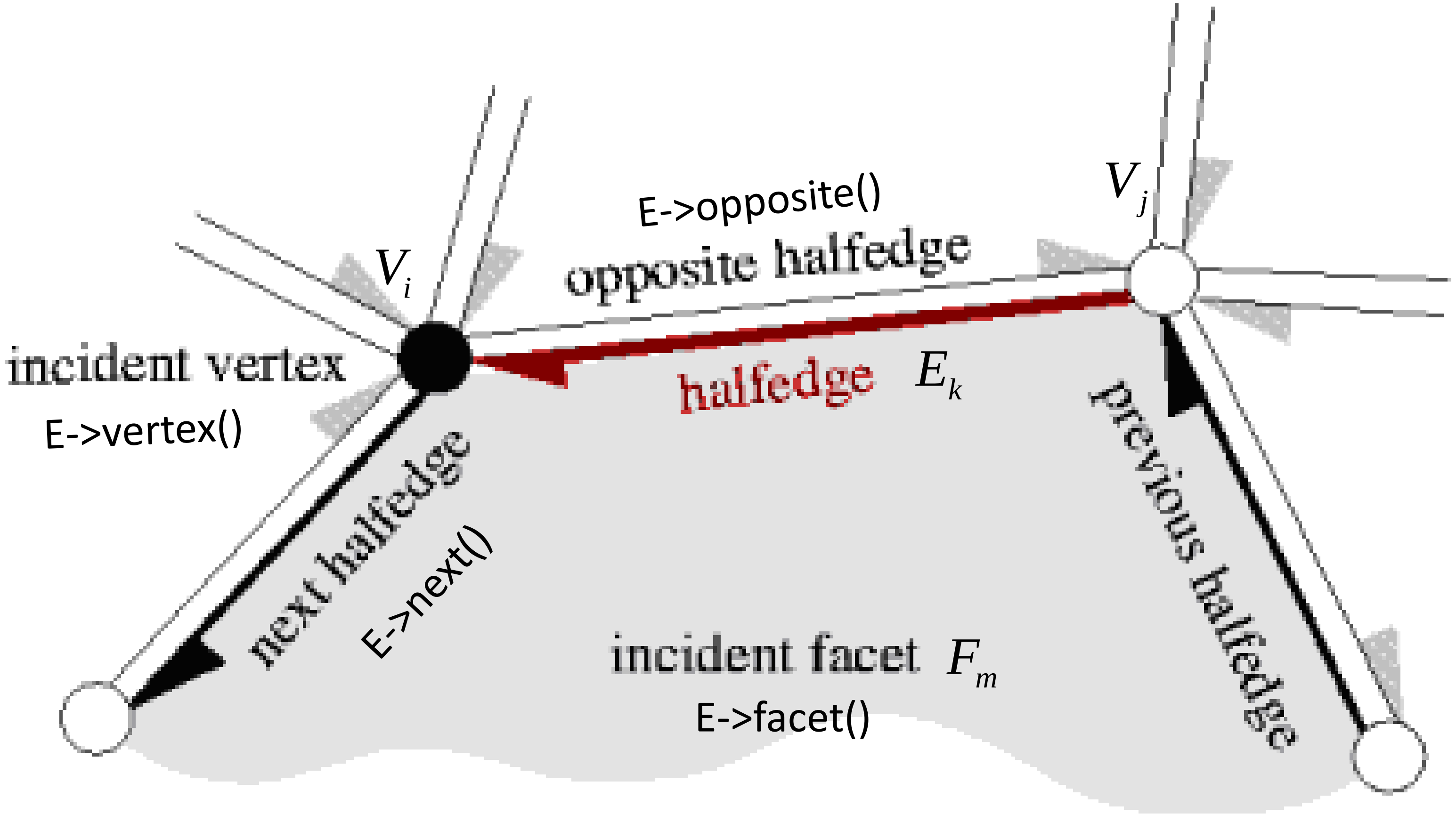
Face merge





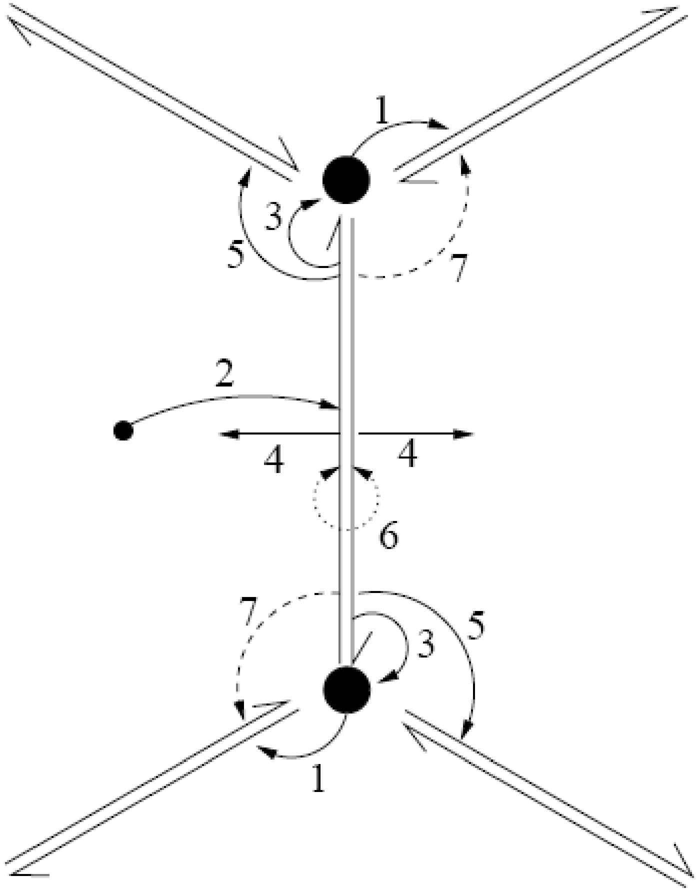


How HDS can -- CGAL



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

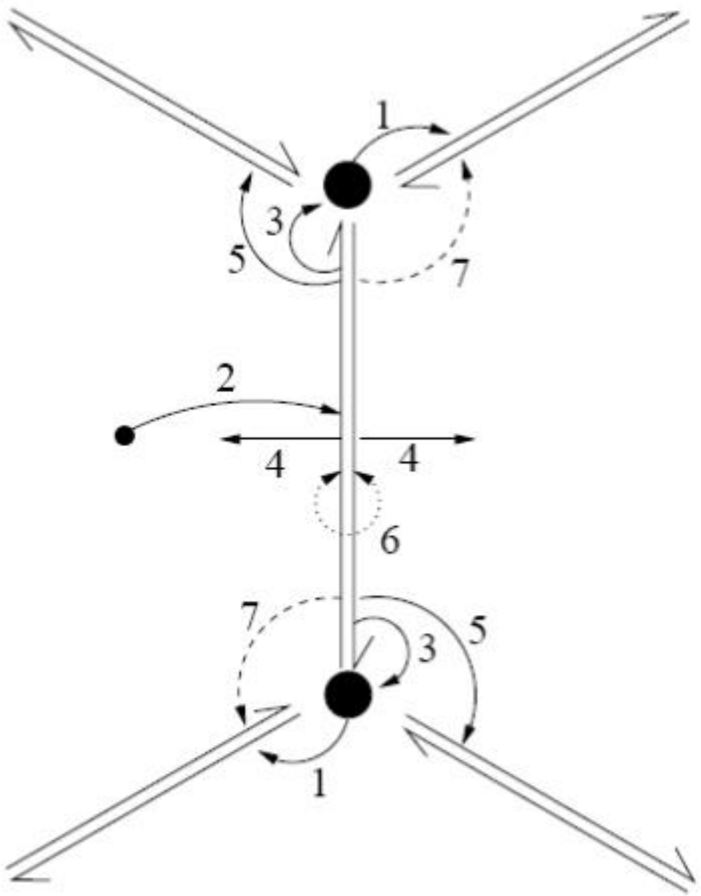
How HDS can -- OpenSG

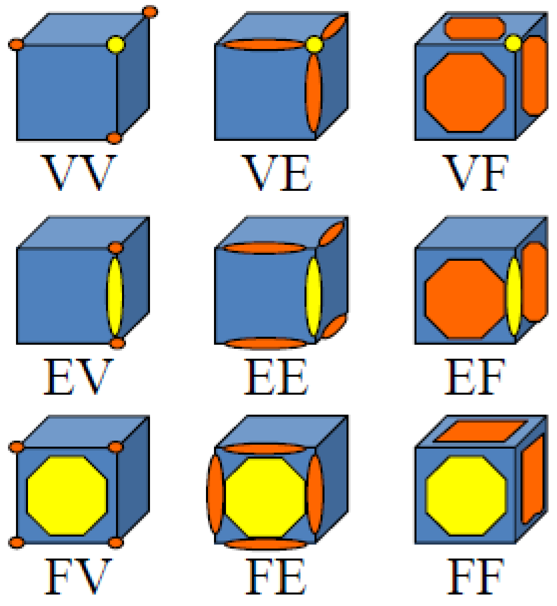


1. Vertex $\mapsto$ one outgoing halfedge,
2. Face $\mapsto$ one halfedge,
3. Halfedge $\mapsto$ target vertex,
4. Halfedge $\mapsto$ its face,
5. Halfedge $\mapsto$ next halfedge,
6. Halfedge $\mapsto$ opposite halfedge (implicit),
7. Halfedge $\mapsto$ previous halfedge (optional).

All basic queries take constant O(1) time!

How HDS can -- OpenSG





All basic queries take constant O(1) time!

Attributes

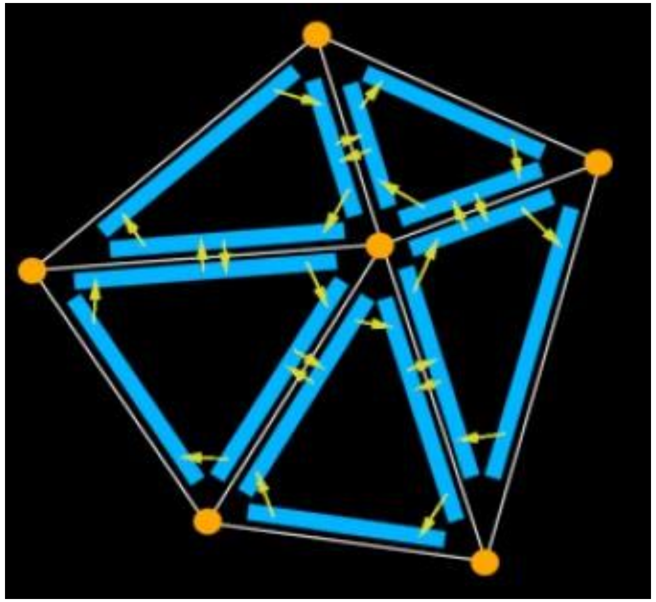
- Each object stores attributes (traits) which defines other structures on the mesh:
  - metric structure: edge length
  - angle structure: halfedge
  - curvature : vertex
  - conformal factor: vertex
  - Laplace-Beltrami operator: edge
  - Ricci flow edge weight; edge
  - holomorphic 1-form: halfedge

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:

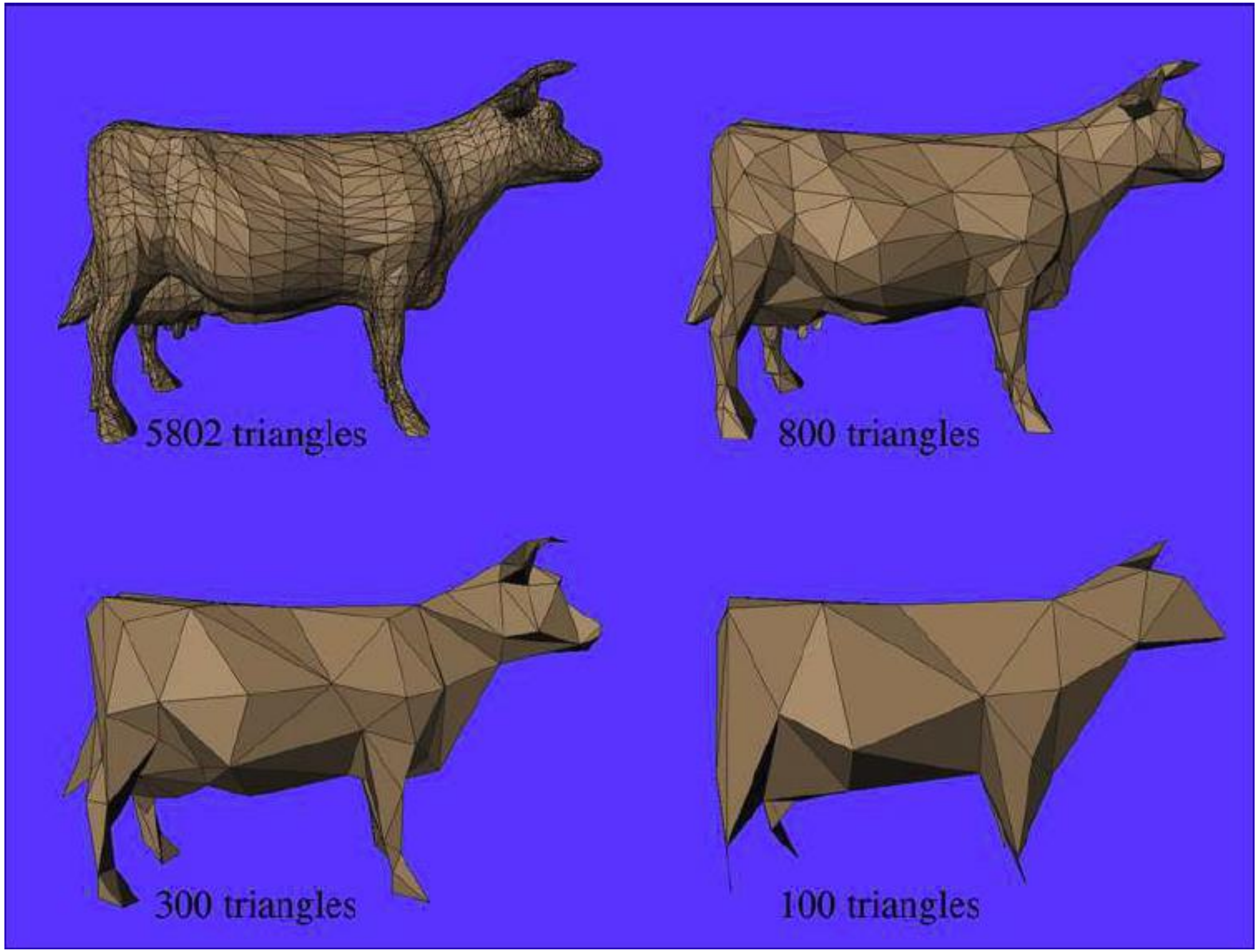
- Efficient local traversal
- Low spatial cost
- Supporting dynamic local updates/manipulations (edge collapse, vertex split, etc.)



(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.

How Many Polygons to Use?



Polygon Models in OpenGL

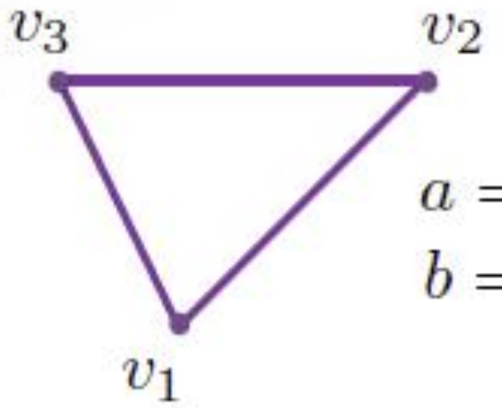
- for faceted shading

```
glNormal3fv(n);
glBegin(GL_POLYGONS);
    glVertex3fv(vert1);
    glVertex3fv(vert2);
    glVertex3fv(vert3);
glEnd();
```

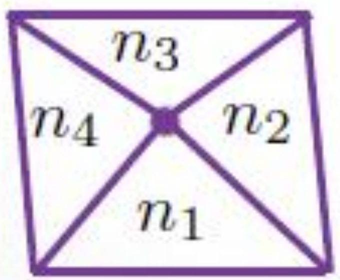
- for smooth shading

```
glBegin(GL_POLYGONS);
    glNormal3fv(normal1);
    glVertex3fv(vert1);
    glNormal3fv(normal2);
    glVertex3fv(vert2);
    glNormal3fv(normal3);
    glVertex3fv(vert3);
glEnd();
```

Normals



$$a = v_2 - v_1$$
$$b = v_3 - v_1$$



- Triangle defines unique plane
  - can easily compute normal

$$n = \frac{a \times b}{\|a \times b\|}$$

  - depends on vertex orientation!
  - clockwise order gives

$$n' = -n$$

- Vertex normals less well defined
  - can average face normals
  - works for smooth surfaces
  - but not at sharp corners (think of a cube)

Why Level of Detail?

- Different models for near and far objects
- Different models for rendering and collision detection
- Compression of data recorded from the real world

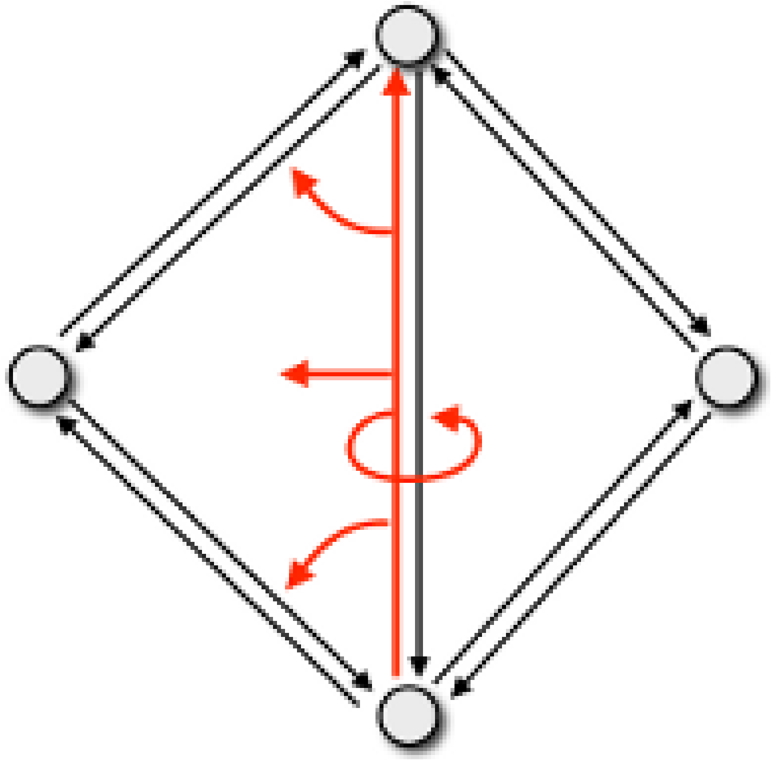
- We need automatic algorithms for reducing the polygon count without
  - losing key features
  - getting artifacts in the silhouette
  - popping

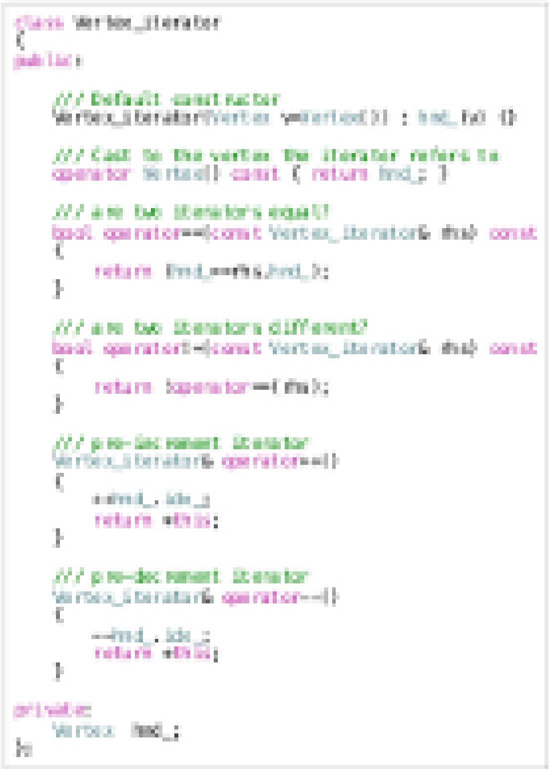
Problems with Triangular Meshes?

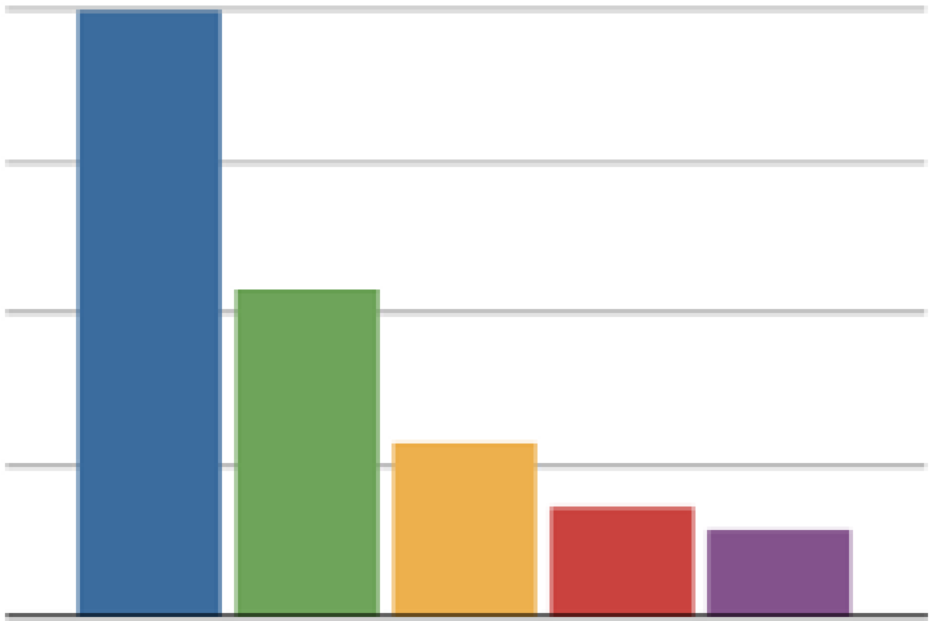
- Need a lot of polygons to represent smooth shapes
- Need a lot of polygons to represent detailed shapes

- Hard to edit
- Need to move individual vertices
- Intersection test? Inside/outside test?

Design, Implementation, and Evaluation of theSurface_mesh Data Structure







[Design, Implementation and Evaluation of the Surface_mesh Data Structure](), *IMR 2011*. has code

Discussion

- Say a word

Resources

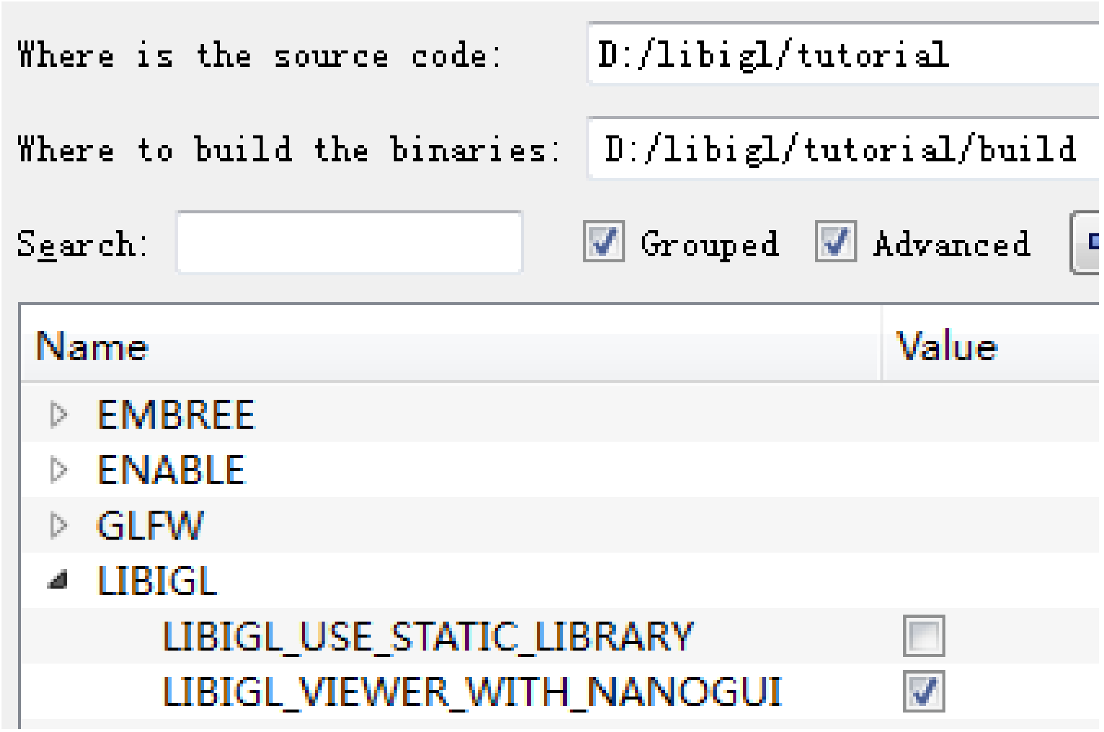
- *https://github.com/jjcao/jjcao_code.git*
- SourceTree

- 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

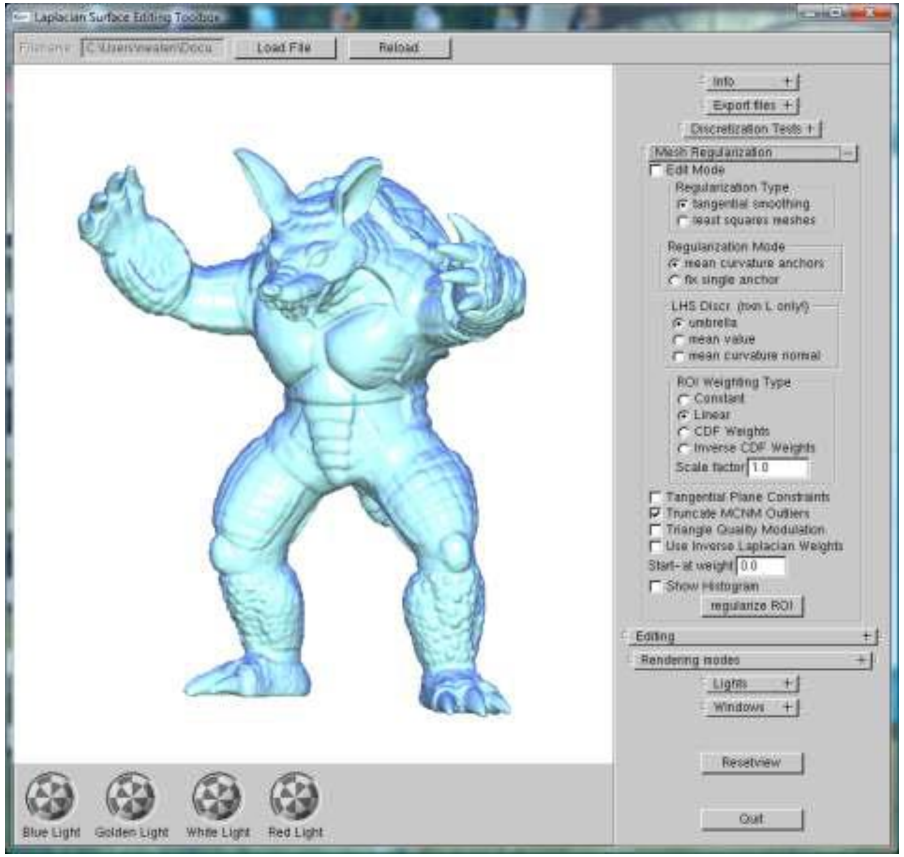
- Lab1
  - **Chapter 1 of libigl tutorial** or jjcao_code\toolbox\jjcao_plot\eg_trisurf.m
- Lab2 [optional]
  - See User manual of Halfedge Data Structures of CGAL
  - run the examples or jjcao_code\toolbox\jjcao_mesh\datastructure\test_to_halfedge.m

The end

Old assignment

Assignment 1: Mesh processing “Hello World”

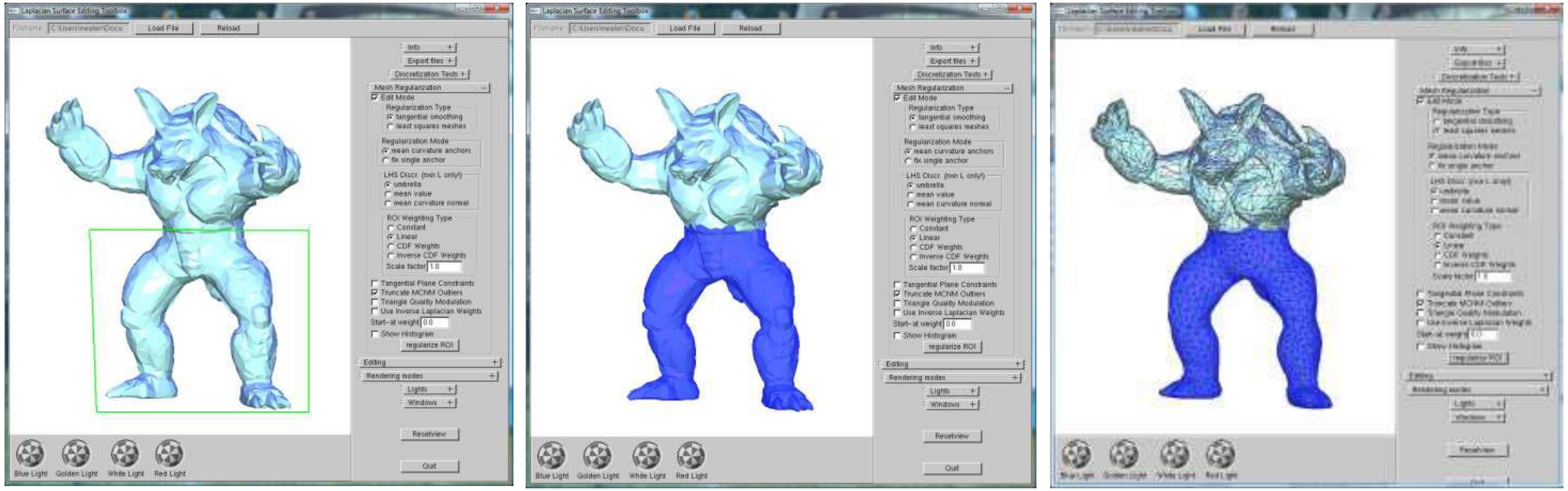
- Goals: learn basic mesh data structure programming + rendering (flat/gouraud shaded, wireframe) + basic GUI programming
- by **MATLAB** or **VC**



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?