3D Models

Dr. Francesco Banterle,
francesco.banterle@isti.cnr.it
banterle.com/francesco

3D Models

- A 3D model is a computational representation of a real-world object. This is typically:
  - C0
  - Closed (not always!)
  - Discretized

3D Models

- Two main representations:
  - **Boundary representations** (b-rep): a 3D object is represented as a collection of connected surface elements; i.e., the boundary between solid and non-solid
  - **Volume representations**: a 3D object is represented by its interior volume. For example, 3D volumes or volume mesh (FEM)

Our focus is on

boundary representations

Polygonal Meshes

3D Representation: Polygonal Meshes

- Discretize the surface in a set of simple primitives:
  - Many points
  - **Triangles**
  - Quads
  - Polygons
  - Our focus is on:
    - simplicial complexes, e.g., triangle!

Why triangular meshes?

- Two main practical reasons:
  - Data-structures are straightforward
  - Graphics hardware (e.g., a GPU) uses triangles;

Why triangular meshes?

- Two main theoretical reasons:
  - Nice theory, i.e., simplicial complexes
  - Less limiting cases:
    - a triangle is always planar!
    - if we remove a vertex, we get another simplicial!

Simplex

- A *k*-simplex, **σ**, is convex combination of (k+1) points ($\mathbf{p}_i$) that are linear independent in the d-dimensional Euclidian space, R^k:

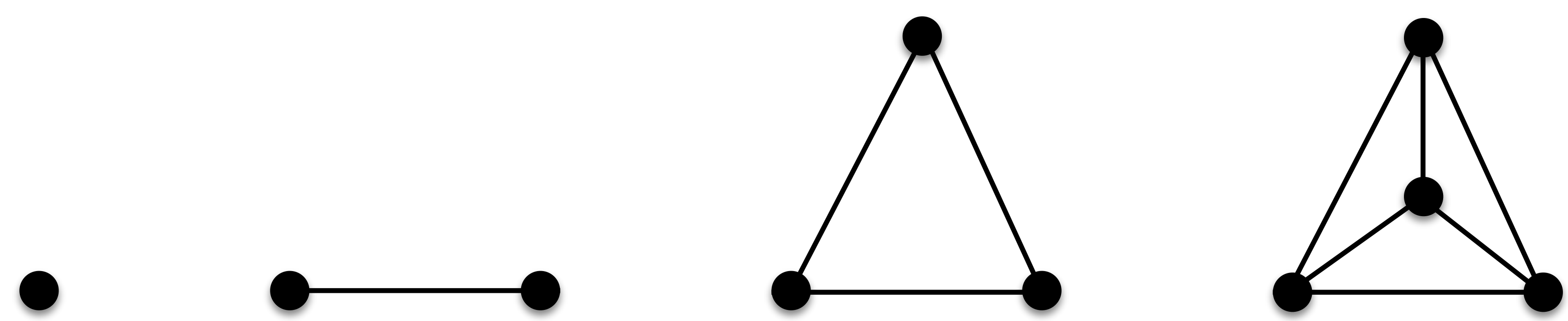
$$\mathbf{x} = \sum_{p_i \in S} \alpha_i \mathbf{p}_i$$

$$\sum_i \alpha_i = 1 \ \wedge \ \alpha_i \geq 0 \ \forall i$$

- A point $\mathbf{p}_i$ is called a vertex

- *k* is the order of the simplex

Simplices Example



k=0 k=1 k=2 k=3

Sub-Simplex

- A sub-simplex **σ'** is called a face of a simplex **σ** if it is a sub-set of vertices of **σ**.

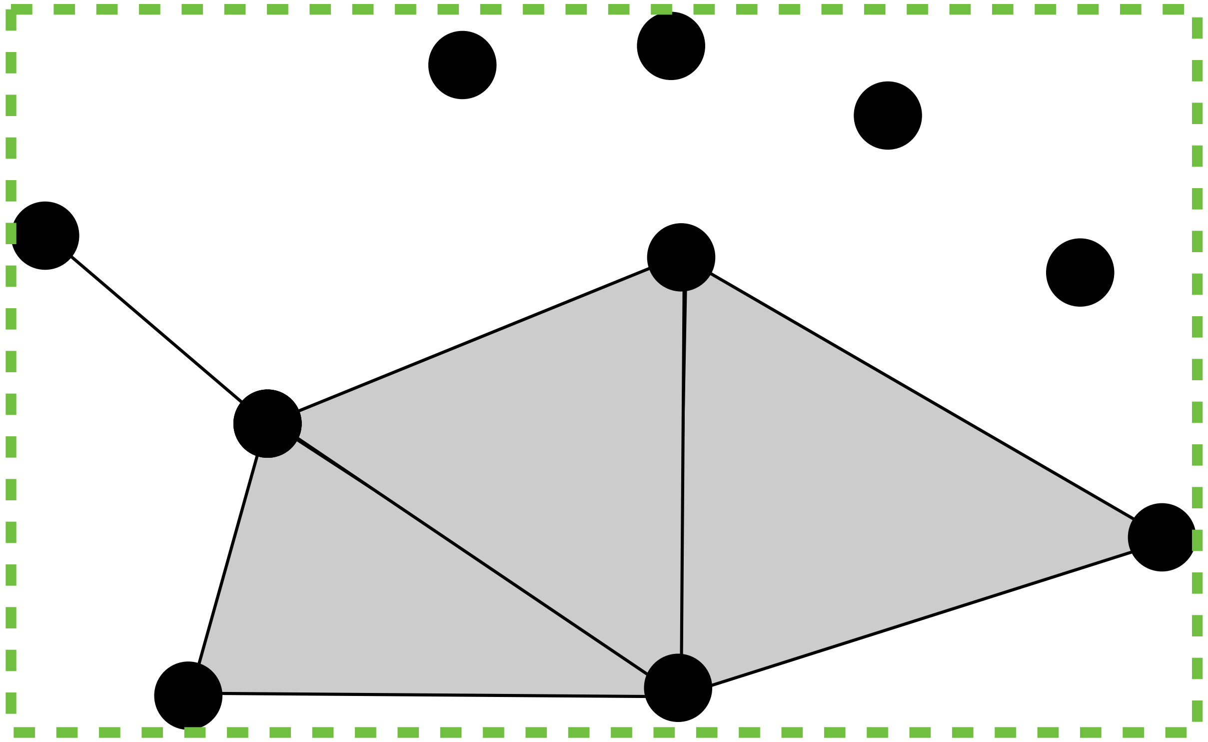
Simplicial Complexes

- A semplicial complex, $\boldsymbol{\Sigma}$, is a finite collection of *K* simplices such that:

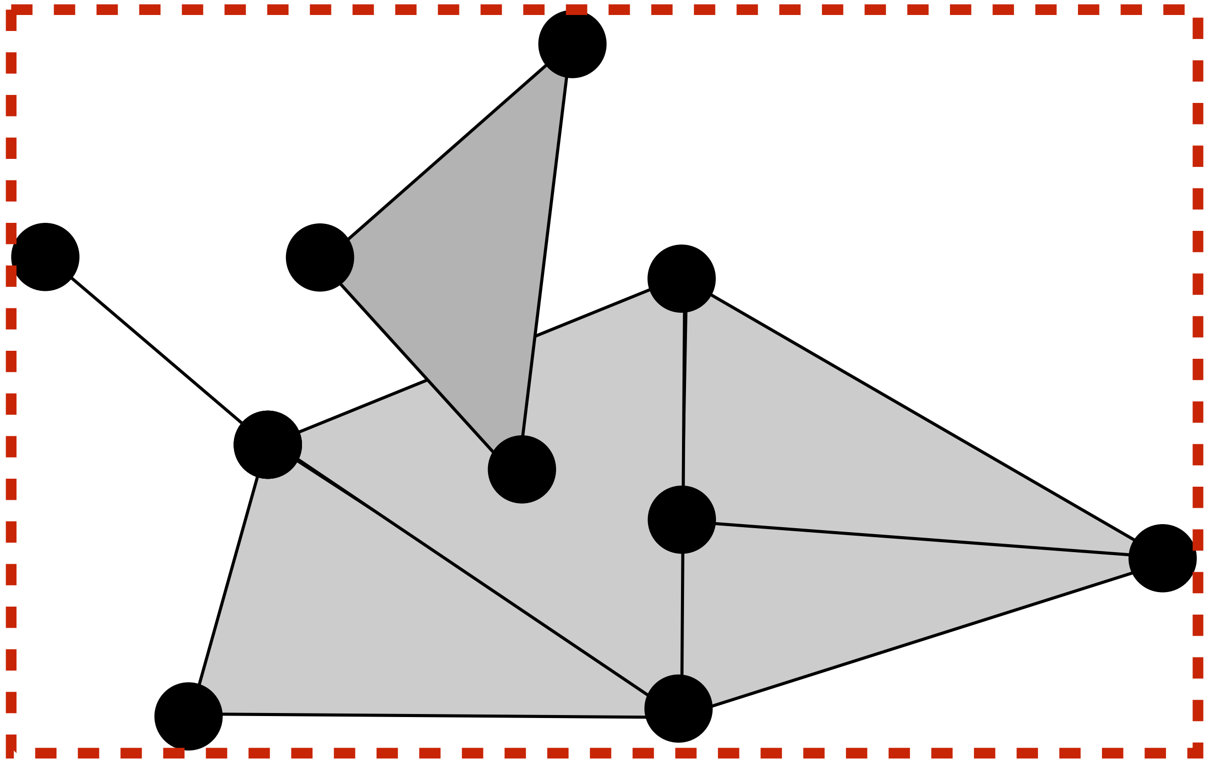
$$(i)\ \sigma_1, \sigma_2 \in \Sigma \rightarrow \sigma_1 \cap \sigma_2 \leq \sigma_1, \sigma_2$$

$$(ii)\ \sigma \in \Sigma \wedge \tau \leq \sigma \rightarrow \tau \in \Sigma$$

Simplicial Complexes Example



OK

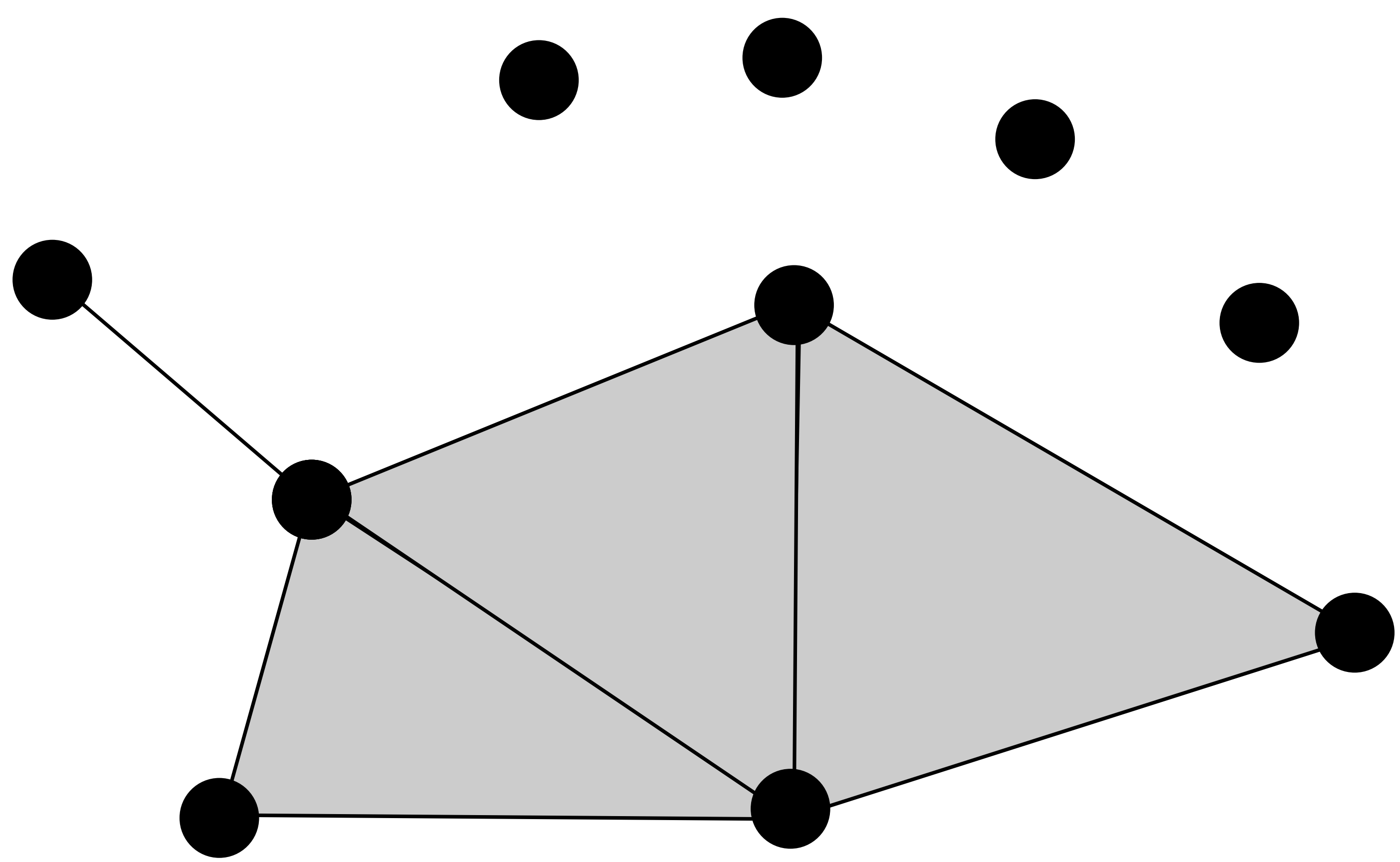


BAD!

Simplicial Complexes

- A simplex, σ, is maximal in a simplicial complex, $\boldsymbol{\Sigma}$, if it does not belong to any other simplex σ_2 of $\boldsymbol{\Sigma}$.
- A *k*-simplicial complex, $\boldsymbol{\Sigma}$, is maximal if all maximal simplices have order k.

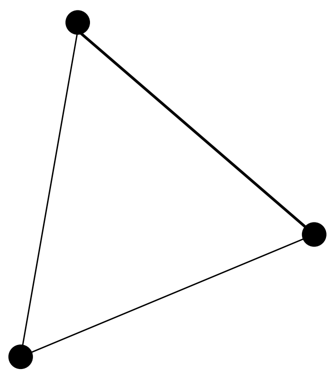
A Non-Maximal Simplicial Complex Example

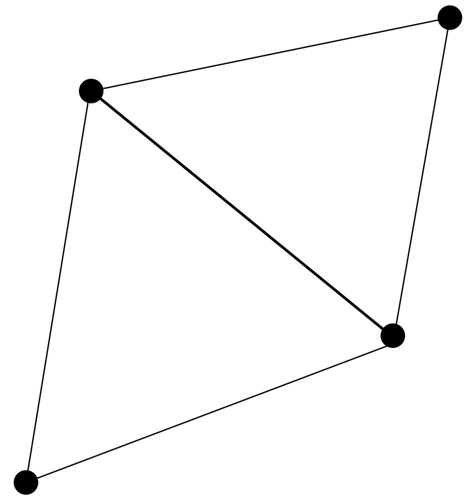


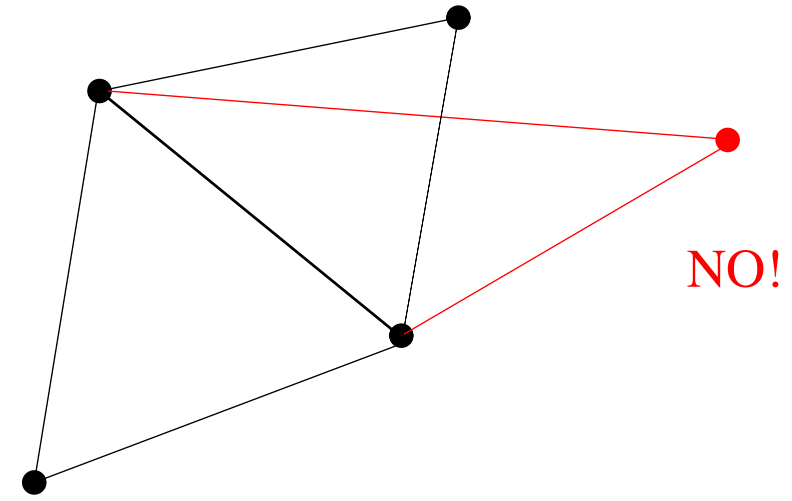
2-Manifold

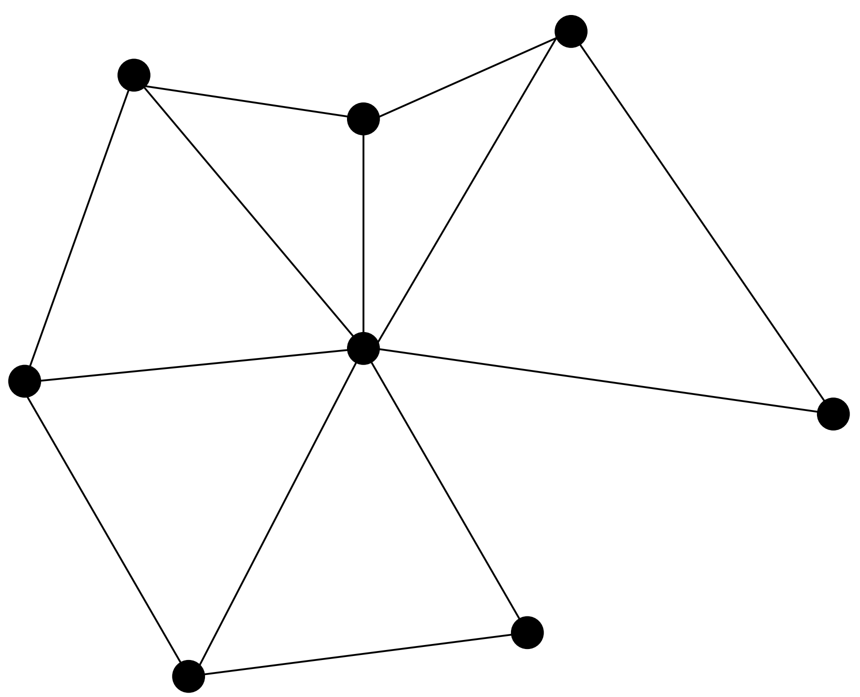
- A surface, *S*, in R^3 such that whose points all have open disks as neighborhoods.
    - This means *S* that looks locally like the plane everywhere.

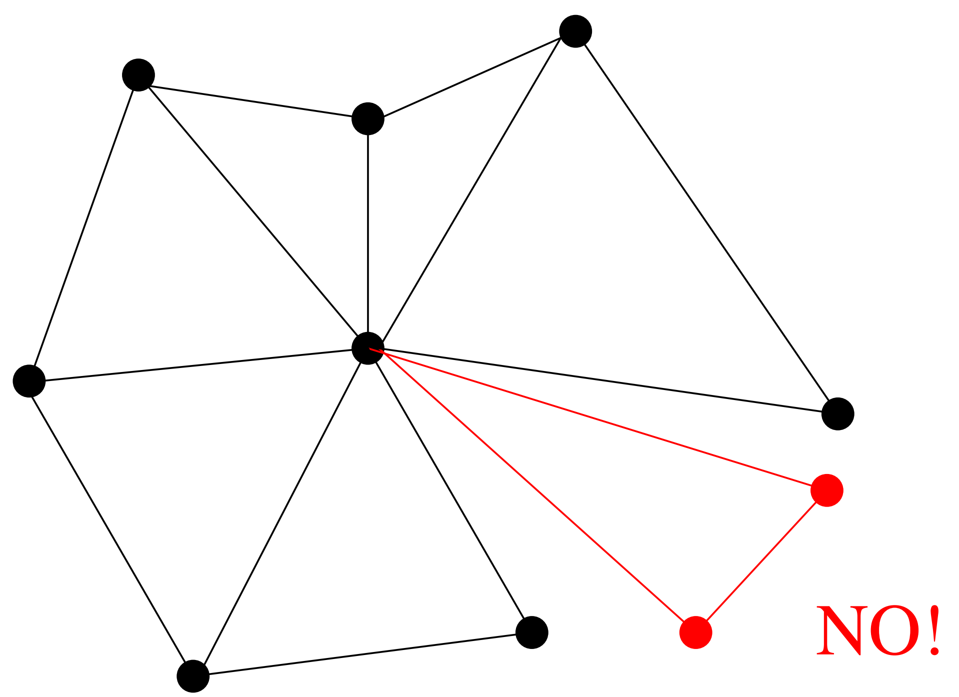
Non-manifold Examples





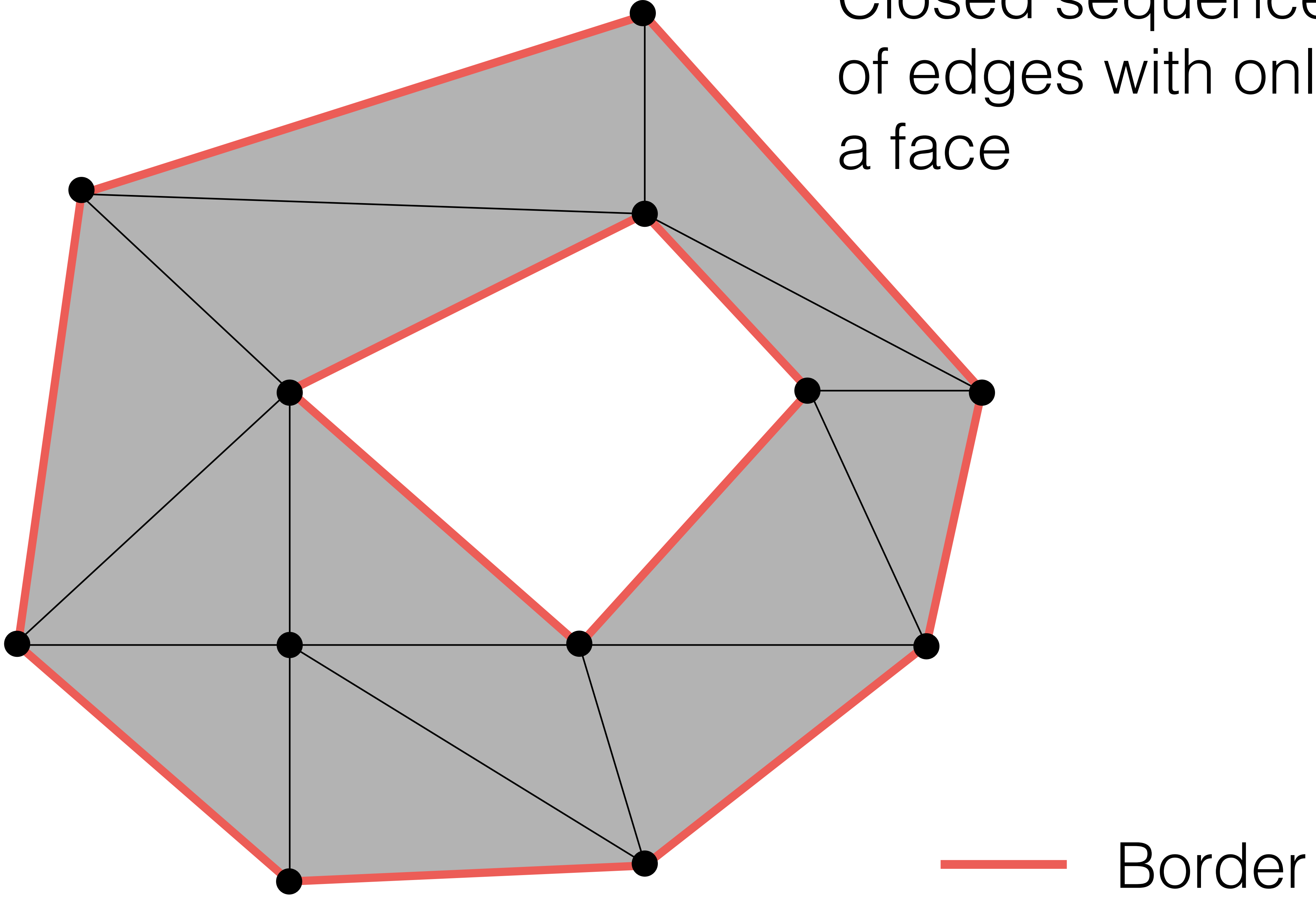






Borders

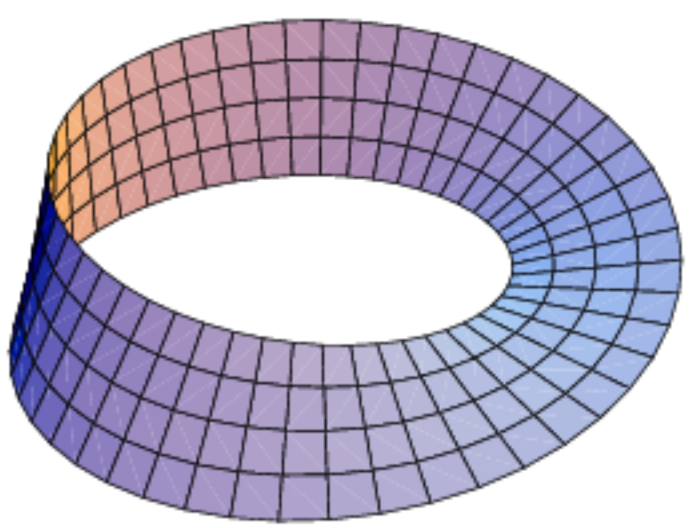
Closed sequence of edges with only a face



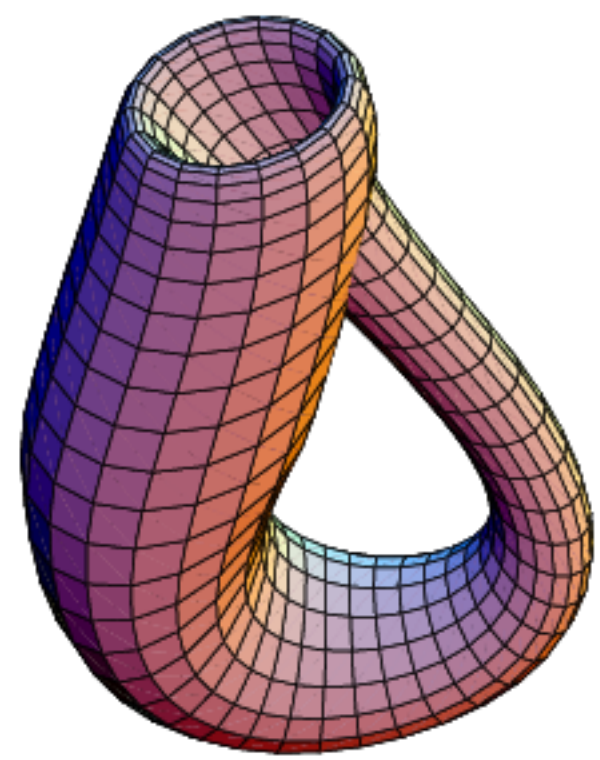
Orientability

- A surface, *S*, is orientable if it is possible to set a coherent normal to each point of the surface

- Note: Möbius strip and Klein bottle and non-manifold surfaces are not orientable:

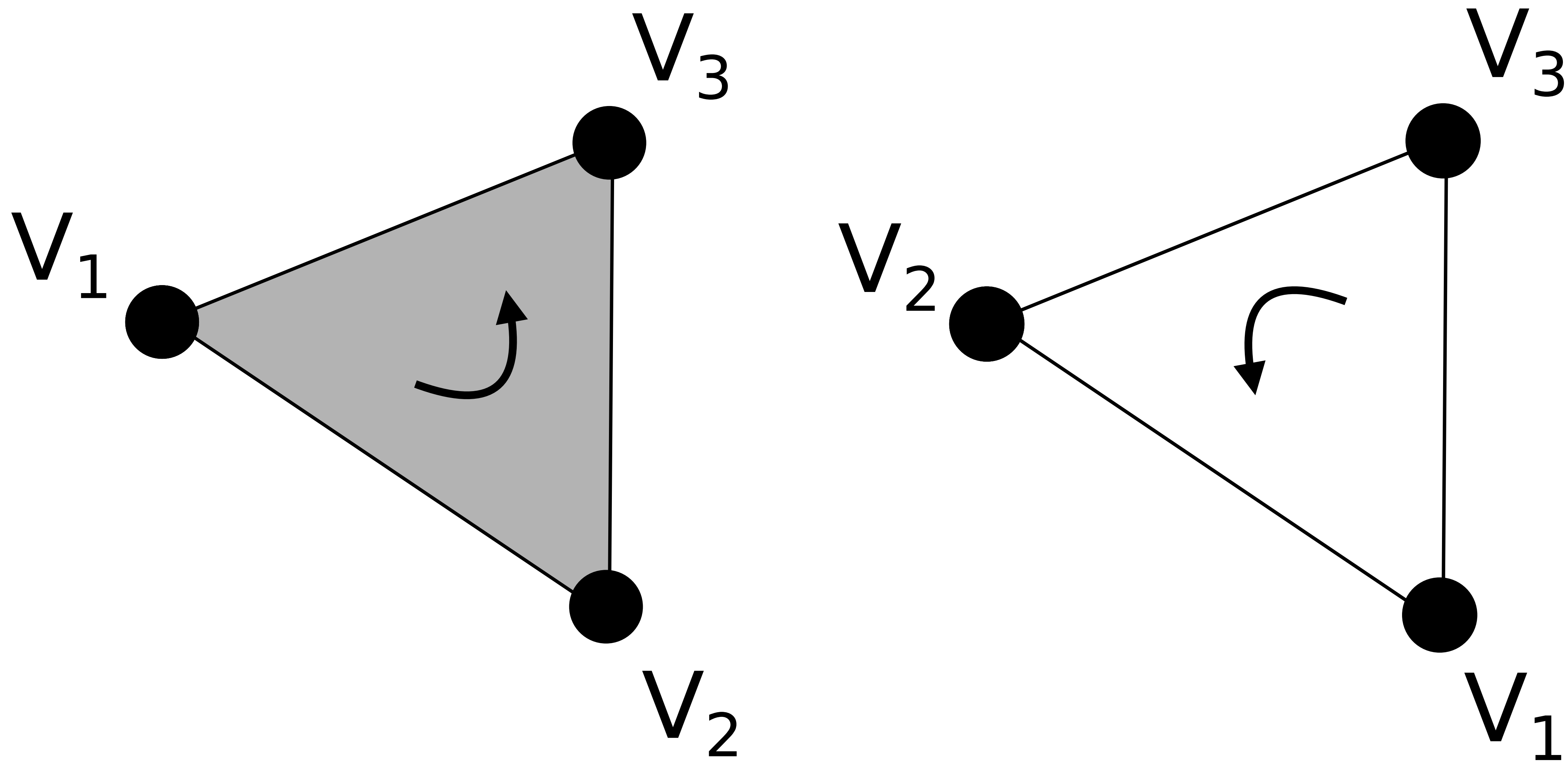


Möbius strip



Klein bottle

Orientability



Front
(counter-clockwise)

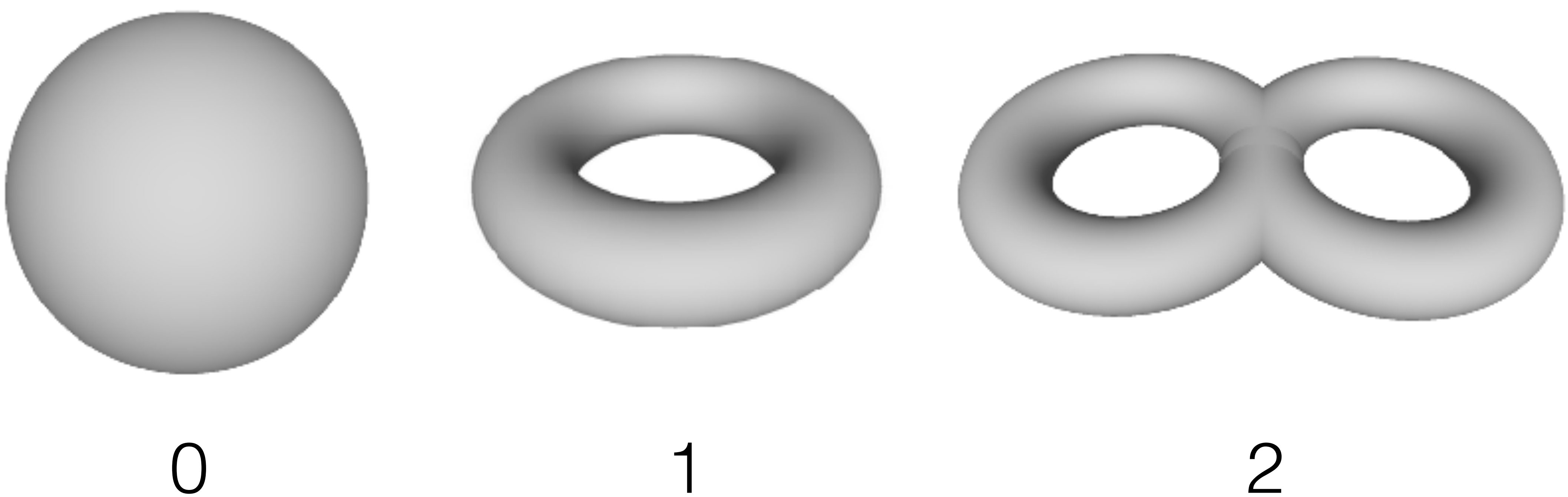
Back
(clockwise)

Mesh

- A mesh is maximal 2-simplicial complexes that is a 2-manifold orientable surface.
  - We can have non 2-manifold meshes
  - We assume that they are maximal

Genus

- The genus is the maximum number of cuttings along non-intersecting closed simple curves without rendering the resultant manifold disconnected



- Genus —> "the number of handles"

Euler Characteristic

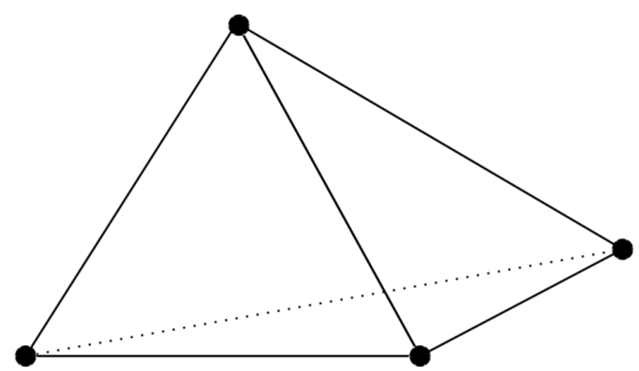
- Given V vertices, E edges, and F faces of a polygonal closed and orientable surface with genus G, we have that:

$$2 - 2G = V - E + F$$
$$\chi = V - E + F$$

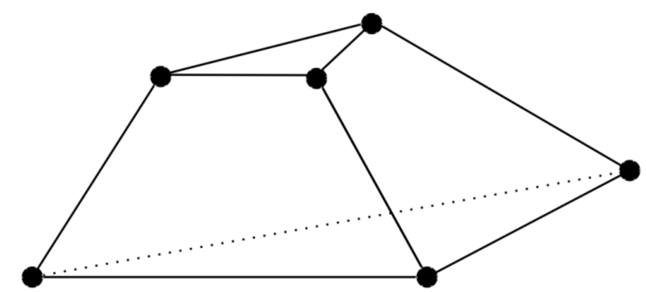
- More in general for a 2-manifold orientable polygonal mesh (with S connected components and B borders):

$$V - L + F = 2(S - G) - B$$

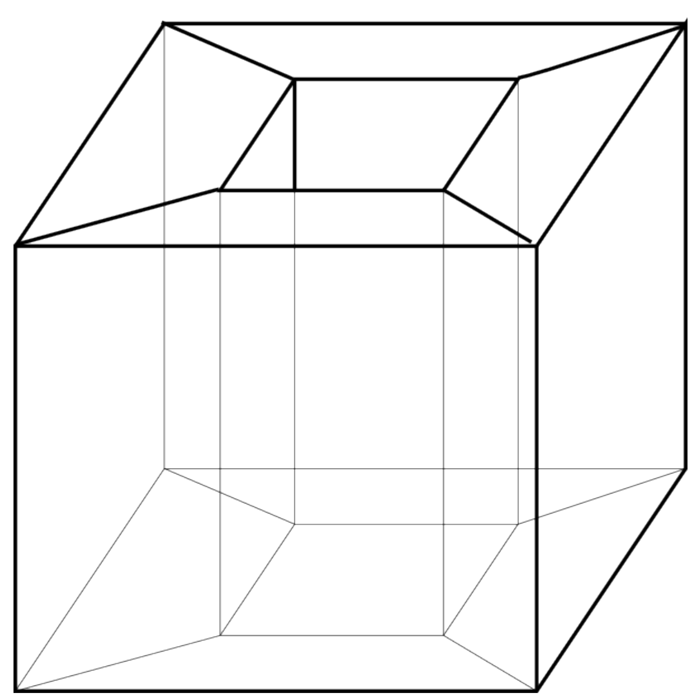
Euler Characteristic Example



$$\chi = V - E + F$$
$$\chi = 4 - 6 + 4 = 2$$



$$\chi = (V+2) - (E+3) + (F+1) =$$
$$\chi = (4+2) - (6+3) + (4+1) = 2$$



$$\chi = V - E + F$$
$$\chi = 16 - 32 + 16 = 0 = \boldsymbol{2 - 2g}$$

Adjacency Relations

- Given two simplices, σ_1 and σ_2, they are incident if σ_1 is a face of σ_2 or vice-versa.
- Two k-simplices are m-adjacent ($k > m$) if a m-simplex exists such that it is a face of both.
- For example:
  - Two triangles sharing an edge are 1-adjacent
  - Two triangle sharing a vertex are 0-adjacent

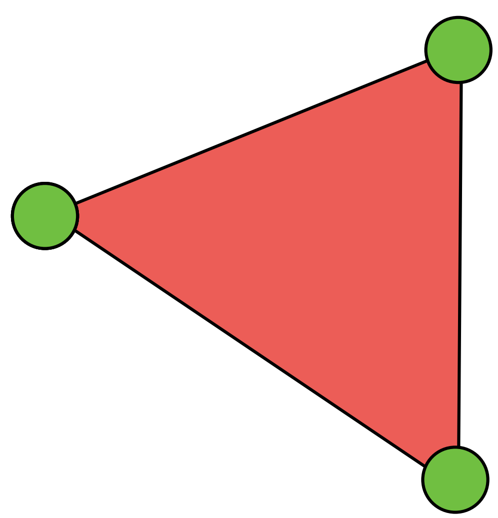
Adjacency Relations

- An adjacency relations is an ordered couple of the following elements:
    - E —> edge
    - F —> Face
    - V —> Vertex
- For example: (E,E), (V,V), (F,F), (E,F), (F,E), (E,V), (V,E), (F,V), (V,F), (E,V), and (V,E).

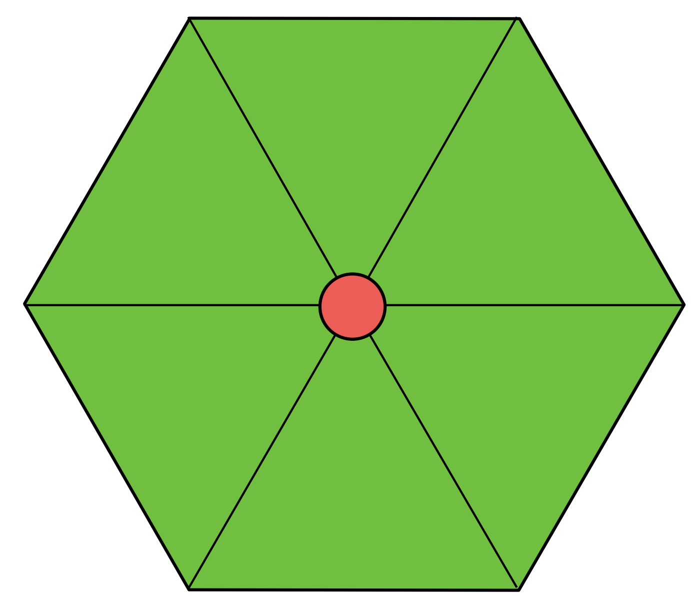
Adjacency Relations Example

- Meaning of some relations:
  - FF —> adjacency between triangles
  - FV —> vertices of a triangle
  - VF —> triangles sharing a vertex

Adjacency Relations Example

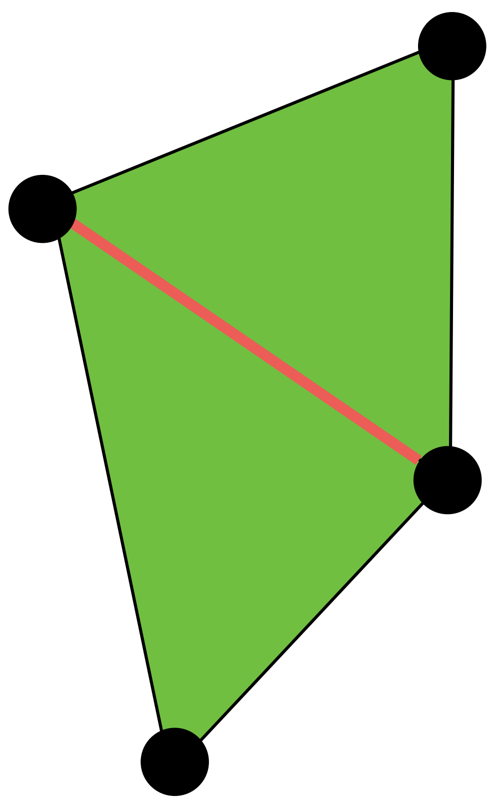


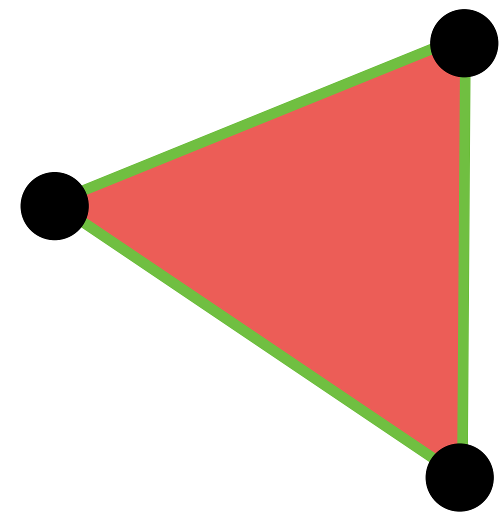
FV



VF

Adjacency Relations Example

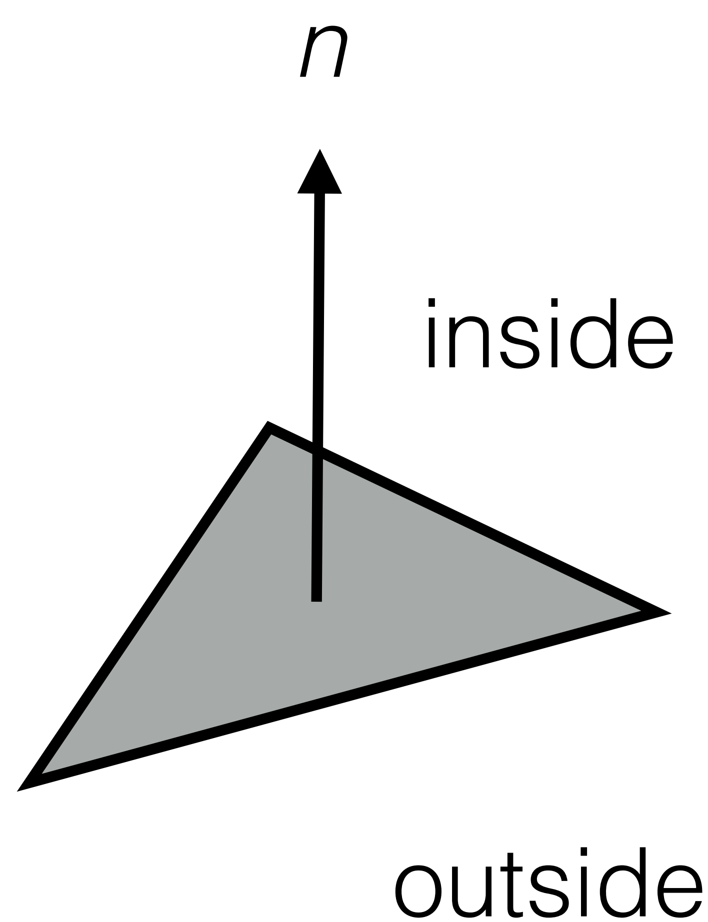




Normals

Normals

- A normal is an important attribute for a vertex:
    - It defines the direction of the object boundary

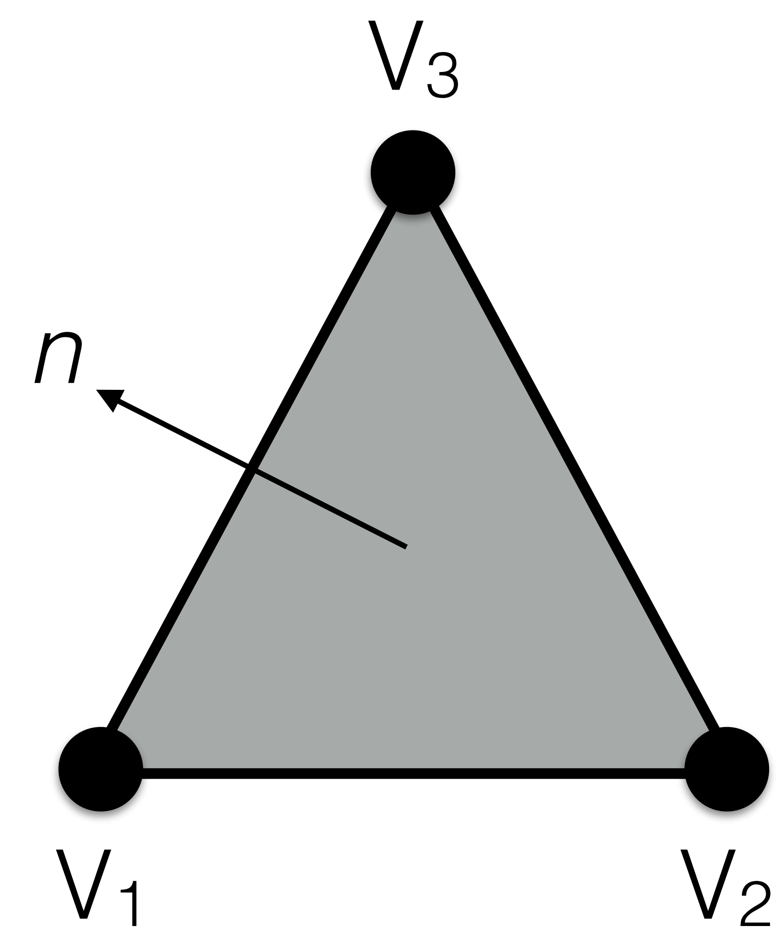


How to compute triangle normal?

- Given a triangle (V_1, V_2, and V_3), its normal (outer-pointing normal):

$$\vec{n} = (V_3 - V_2) \times (V_1 - V_2)$$

$$\vec{n} = \frac{\vec{n}}{\|\vec{n}\|}$$



- This means that vertices order is important! Typically is counter-clockwise

How to compute per vertex normal?

- We compute normals for each triangle
- For each vertex:
  - We compute the sum of normals of all triangles sharing that vertex:

$$\vec{n}_s(V) = \sum_{\{i | V \in T_i\}} \vec{n}_{T_i}$$

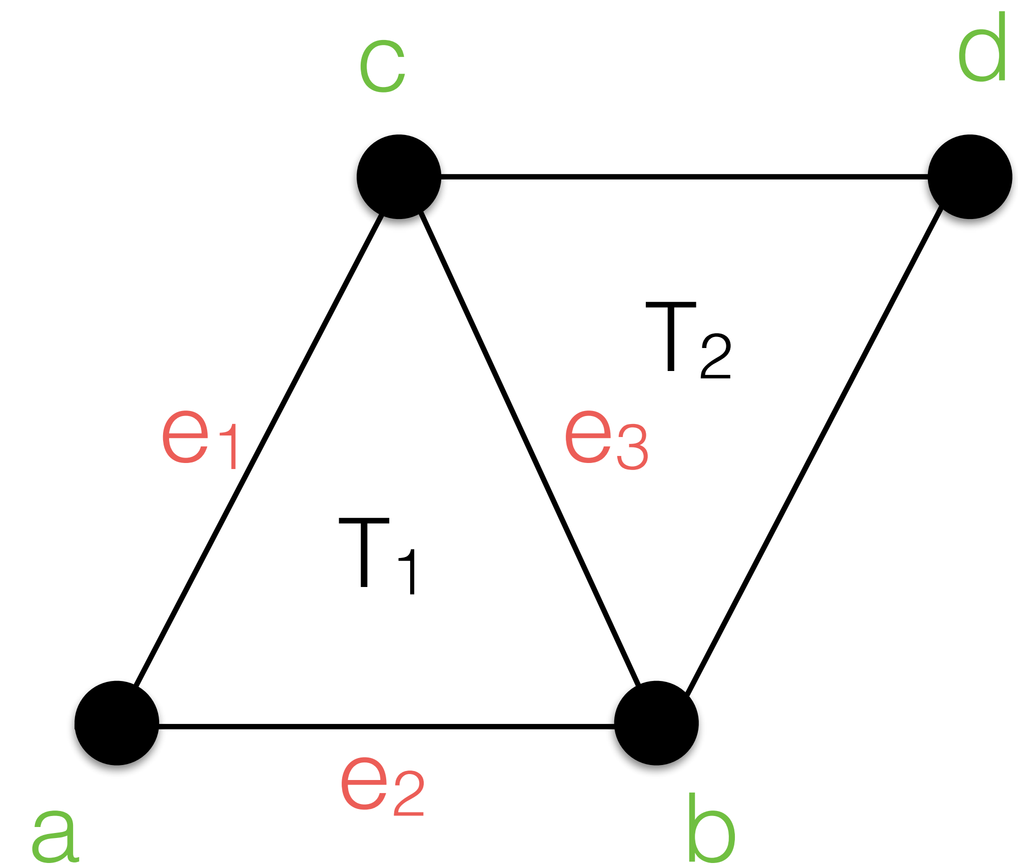
  - We normalize this sum
- Note: per-vertex normals are useful but not correct!

Data Structures for 3D Meshes

List of Triangles

- For each triangle of the 3D model, we store its coordinates.

- For example:



Triangle 1:	(3,-2,5); (2,2,4); (-6,2,4)
Triangle 2:	(2,2,4) ; (0,-1,-2); (9,4,0)
Triangle 3:	(1,2,-2); (3,-2,5); (-6,2,4)
	....
Triangle *n*:	(-8,2,7); (-2,3,9); (1,2,-7)

What's ***very wrong*** with this??

Triangle 1: (3,-2,5); (2,2,4); (-6,2,4)
Triangle 2: (2,2,4) ; (0,-1,-2); (9,4,0)
Triangle 3: (1,2,-2); (3,-2,5); (-6,2,4)
....
Triangle *n*: (-8,2,7); (-2,3,9); (1,2,-7)

What's ***very wrong*** with this??

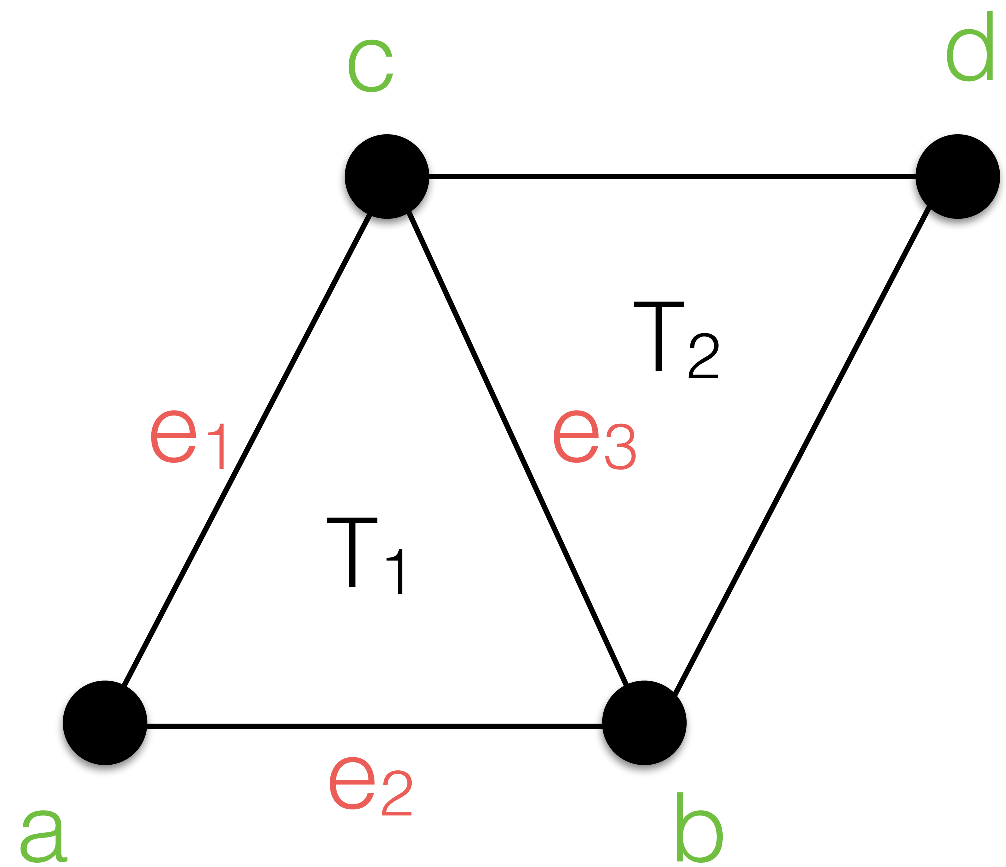
Triangle 1: (**3,-2,5**); (**2,2,4**); (**-6,-2,4**)
Triangle 2: (**2,2,4**) ; (0,-1,-2); (9,4,0)
Triangle 3: (1,2,-2); (**3,-2,5**); (**-6,-2,4**)
....
Triangle *n*: (-8,2,7); (-2,3,9); (1,2,-7)

List of Triangles

- Disadvantages:
  - Wasted disk and memory space:
    - Vertices are duplicated!
    - Memory: |V| * |T|
  - Difficult to manage:
    - if we modify a vertex of a triangle, we will need to find and update its clones!
  - How do we query neighbors?

List of Unique Vertices

- We store vertices in a list
- For each triangle of the 3D model, we store indices to the vertices' list



Vertices:

1. (-1.0, -1.0, -1.0)
2. (-1.0, -1.0, 1.0)
3. (-1.0, 1.0, -1.0)
4. (-1, 1, 1.0)
5. (1.0, -1.0, -1.0)
6. (1.0, -1.0, 1.0)
7. (1.0, 1.0, -1.0)
8. (1.0, 1.0, 1.0)

Faces:

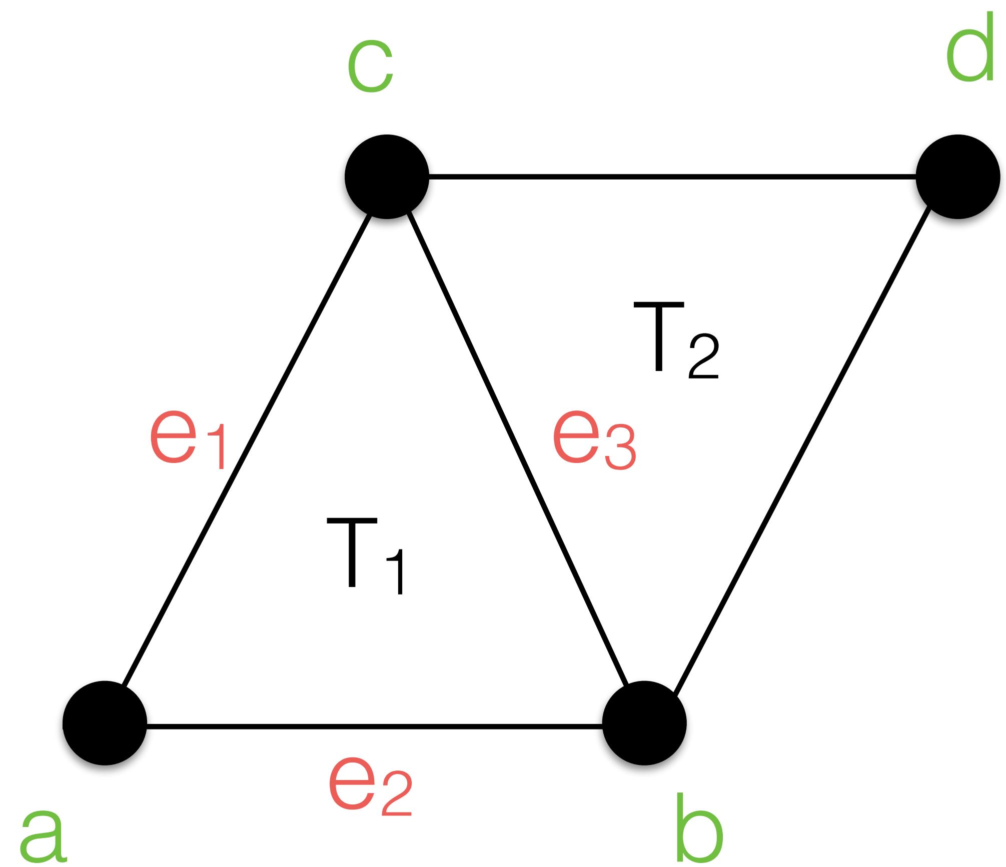
1. 1 2 4
2. 5 7 6
3. 1 5 2
4. 3 4 7
5. 1 7 5

List of Unique Vertices

- Wasted disk and memory space:
  - Common edges between two triangles are stored two times in the list of faces!
  - Memory: $|V| + |T|$
- Better management:
  - Easy to edit a vertex's attribute (e.g., its position)!
- How do we query neighbors?

List of Unique Edges

- We store vertices in a list
- For each edge, we store indices to the vertices' list
- For each triangle of the 3D model, we store indices to edges's list



Vertices:
1. (-1.0, -1.0, -1.0)
2. (-1.0, -1.0, 1.0)
3. (-1.0, 1.0, -1.0)
4. (-1, 1, 1.0)

Edges:
1. 1 2
2. 2 3
3. 4 2
4. 3 4
5. 1 3

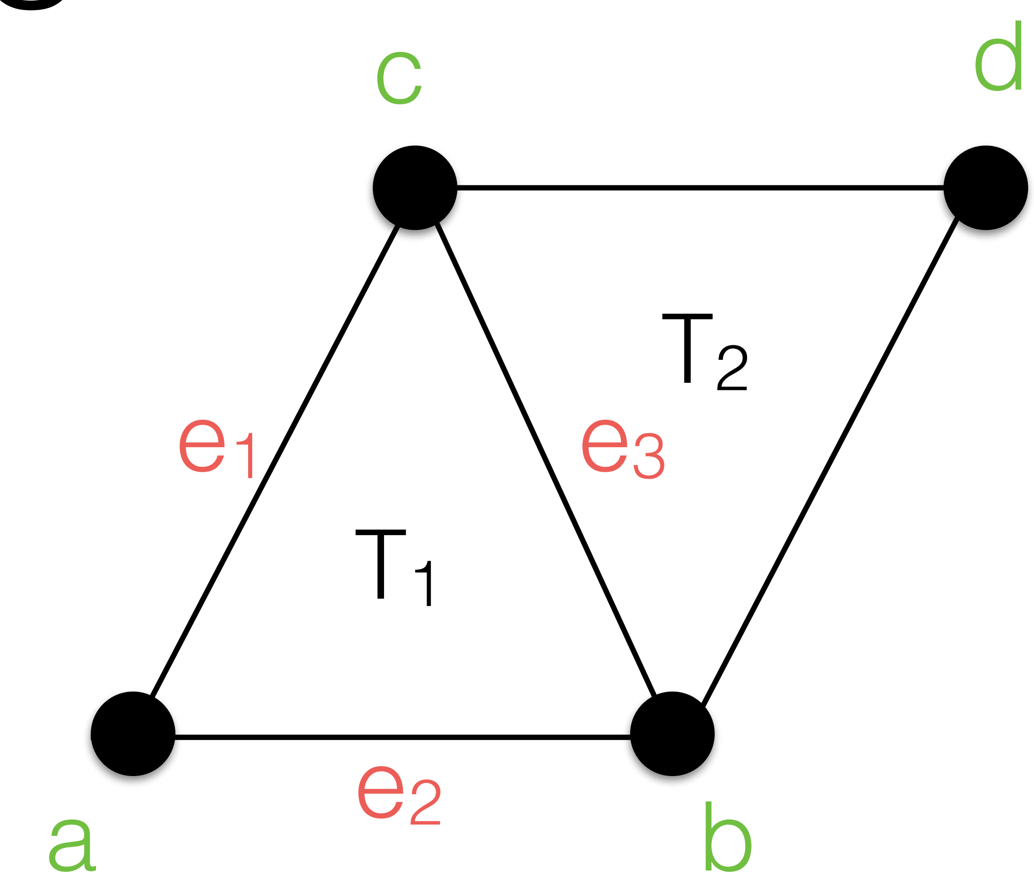
Faces:
1. 1 2 5
2. 2 4 3

List of Unique Edges

- Better management:
  - Easy to edit an edge’s attribute (e.g., its color)!
- We can do some queries, but not all of them!

Extended List of Unique Edges

- We add to an edge the indices of its left and right triangle
- This simplifies edge-face queries!



Vertices:

1. (-1.0, -1.0, -1.0)
2. (-1.0, -1.0, 1.0)
3. (-1.0, 1.0, -1.0)
4. (-1, 1, 1.0)

Edges:	**Faces**:
1. 1 2	1. -1 1
2. 2 3	2. 1 2
3. 4 2	3. -1 2
4. 3 4	4. -1 2
5. 1 3	5. 1 -1

Faces:

1. 1 2 5
2. 2 4 3

File Formats

File Formats

- There are many 3D file formats. The most used, and de-facto standard:
  - STL
  - PLY
  - OBJ
- Standards:
  - COLLADA: https://www.khronos.org/collada/
  - X3D: http://www.web3d.org/x3d/

STL File Format

- Standard Triangle Language (STL) created by 3D Systems
- This format represents only the 3D geometry:
    - No color/texture
    - No other attributes
- The format specifies both ASCII and binary representations

STL File Format

- Data structure: list of triangles
- Vertices are ordered using the right-hand rule
- 3D coordinates must be positive
- No scale metadata; i.e., units are arbitrary

STL File Format

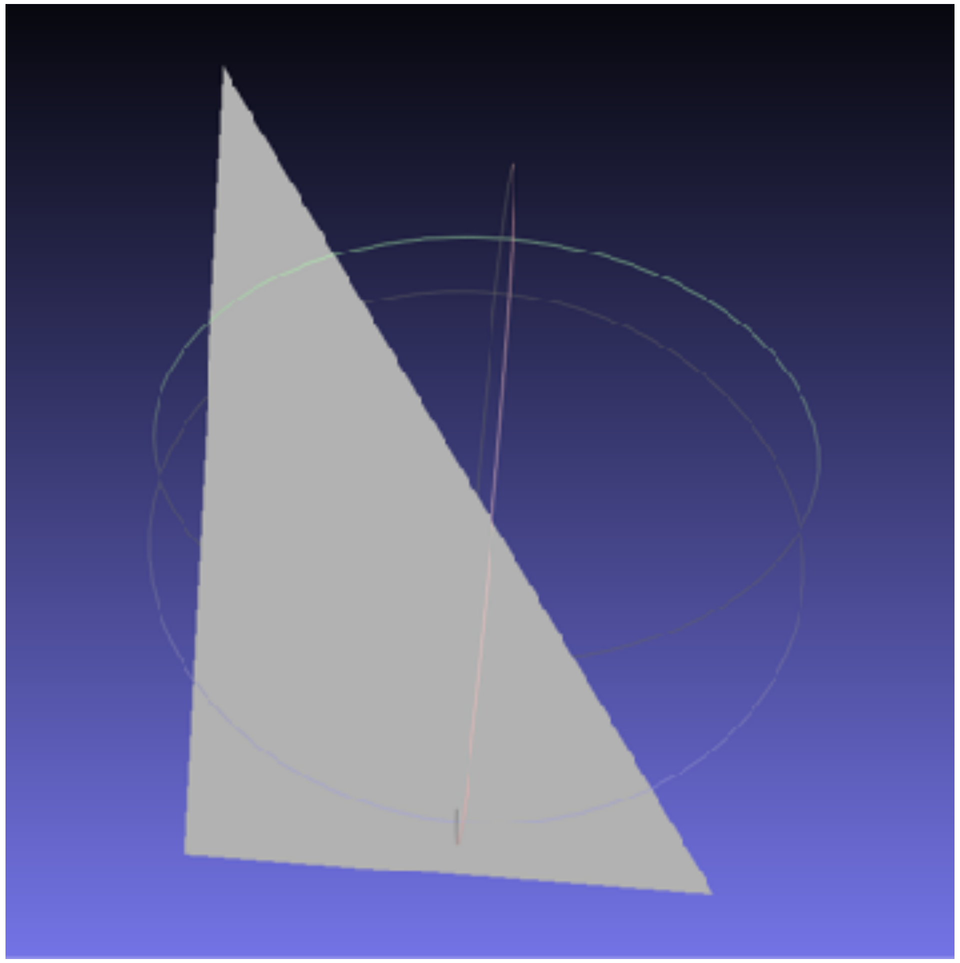
- The file begins as

```
solid name
```

- A face is defined as

```
facet normal nx ny nz
    outer loop
        vertex v1x v1y v1z
        vertex v2x v2y v2z
        vertex v3x v3y v3z
    endloop
endfacet
```

STL File Format: An Example



```
solid triangle
facet normal 0 1 0
        outer loop
                vertex 0.0 0.0 0.0
                vertex 1.0 0.0 0.0
                vertex 0.0 1.0 1.0
        endloop
endfacet
endsolid triangle
```

PLY File Format

- Polygon File Format (PLY) is a popular format created by Stanford University (Greg Turk)
- The format is very flexible:
  - we can add many attributes
  - we can define triangular and polygonal meshes
- The format specifies both ASCII and binary representations

STL File Format

- Data structure: list of unique vertices
- No scale metadata; i.e., units are arbitrary
- The file is divided into two parts:
    - **Header** that specifies vertices and faces
    - **Body** that specifies the concrete data

PLY File Format: Header

- The file begins as

```
ply
format ascii 1.0
```

- Vertex specification is defined as

```
element vertex num_vertices
property float x
property float y
property float z
```

properties can be: char, uchar, short, ushort, int, uint float, double, etc.

PLY File Format: Header

- Faces are defined as

```
element face num_faces
property list uchar int vertex_indices

end_header
```

PLY File Format: Body

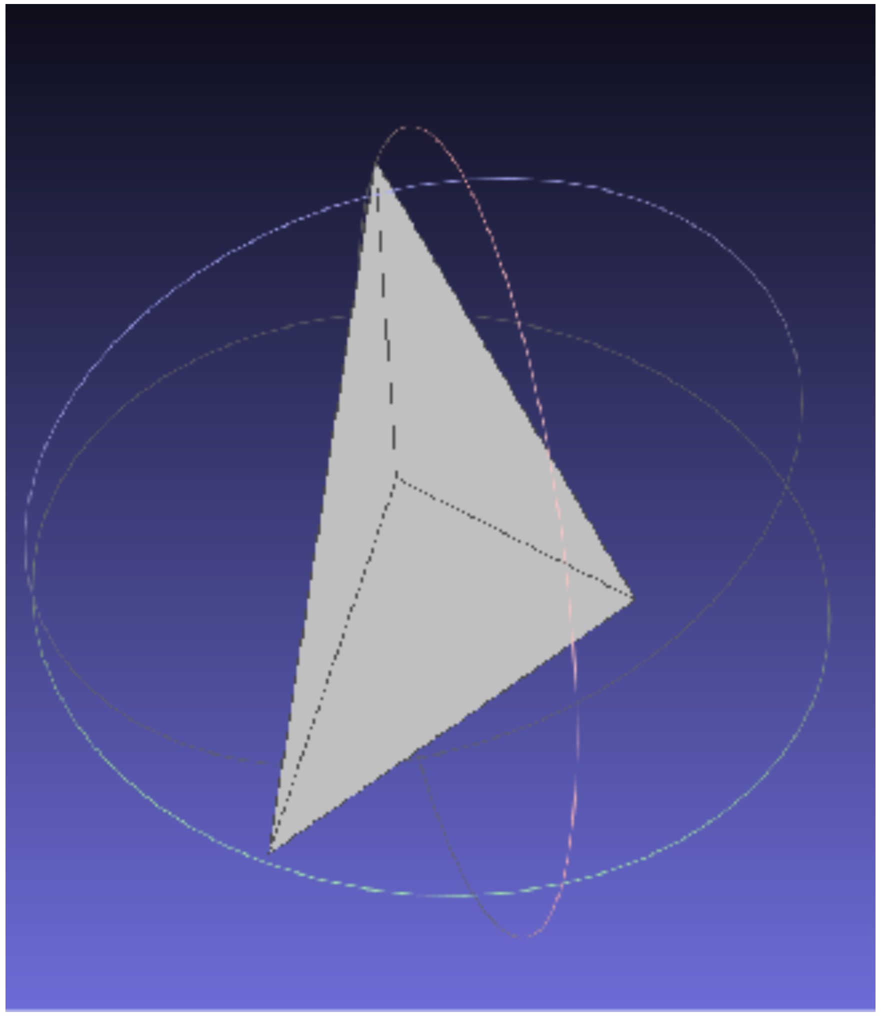
- Each i-th vertex is specified as

```
vix viy viz
```

- Each face is specified as

```
3 index_v1 index_v2 index_v2
```

PLY File Format: An Example



```
ply
format ascii 1.0
element vertex 4
property float x
property float y
property float z
element face 4
property list uchar int vertex_indices
end_header
-0.60 -0.97 0.37
-0.34 0.98 0.76
0.037 0.65 -1.06
0.88 -0.75 -0.25
3 1 3 2
3 0 1 2
3 0 3 1
3 3 0 2
```

Acknowledgements

- Some images and text are based on work by:
  - Dr. Paolo Cignoni:
    - http://vcg.isti.cnr.it/~cignoni/