Lecture #14

Modeling

Computer Graphics

Winter Term 2016/17

Marc Stamminger / Roberto Grosso

Content

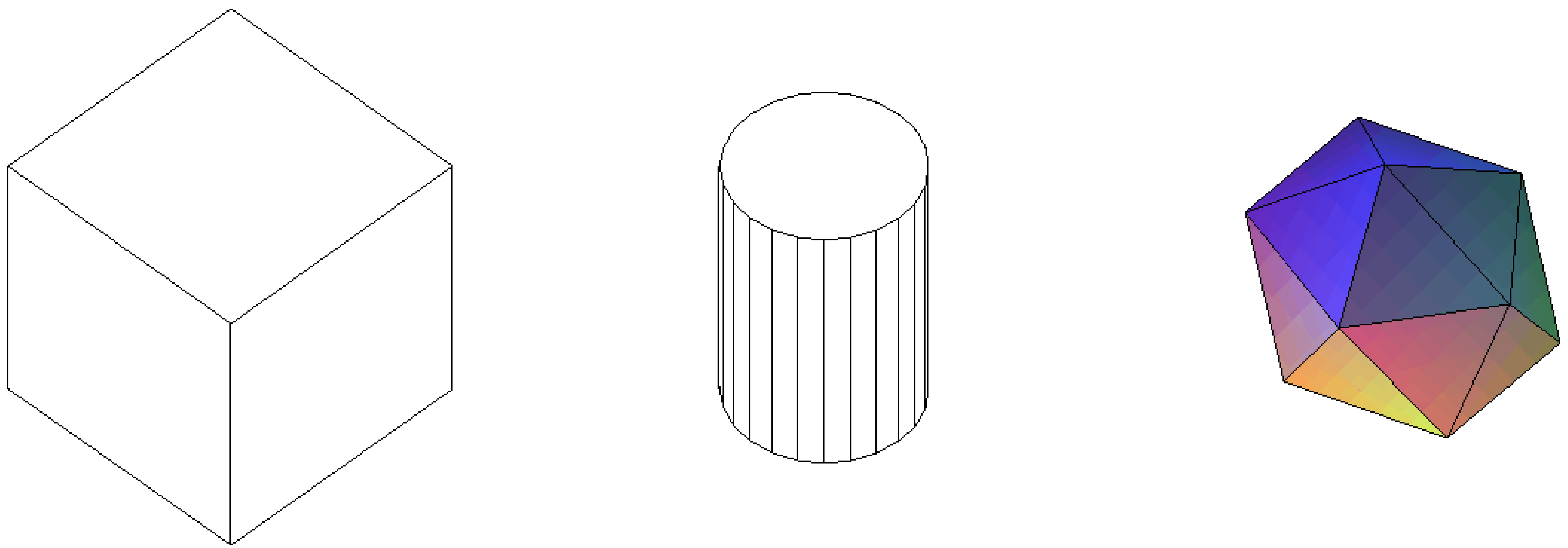
- Up to now:
  → scene objects are triangle meshes, represented as **indexed face sets**
  → scenes composited of multiple objects using scene graphs (lecture #10)

- More on modeling single objects
  - **Polygon meshes → today**
  - Parametric surfaces, Subdivision surfaces → tomorrow

- Other modeling paradigms than triangle meshes:
  - Constructive solid geometry → ray tracing
  - Implicit modeling, signed distance fields → ray tracing

Single Objects

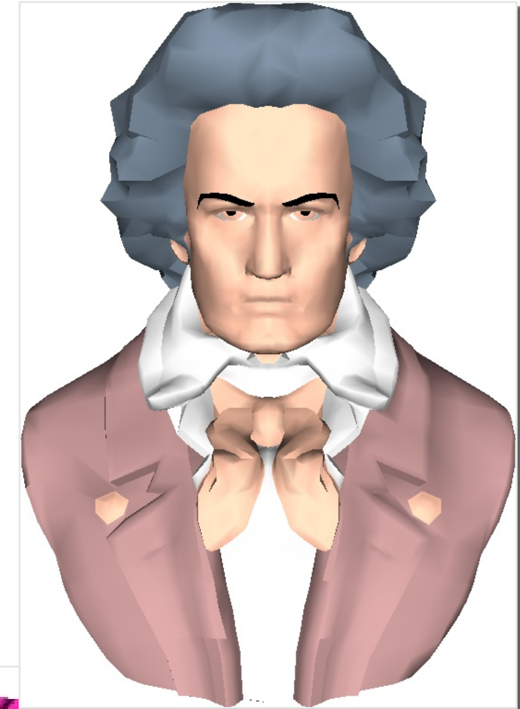
- Simple objects

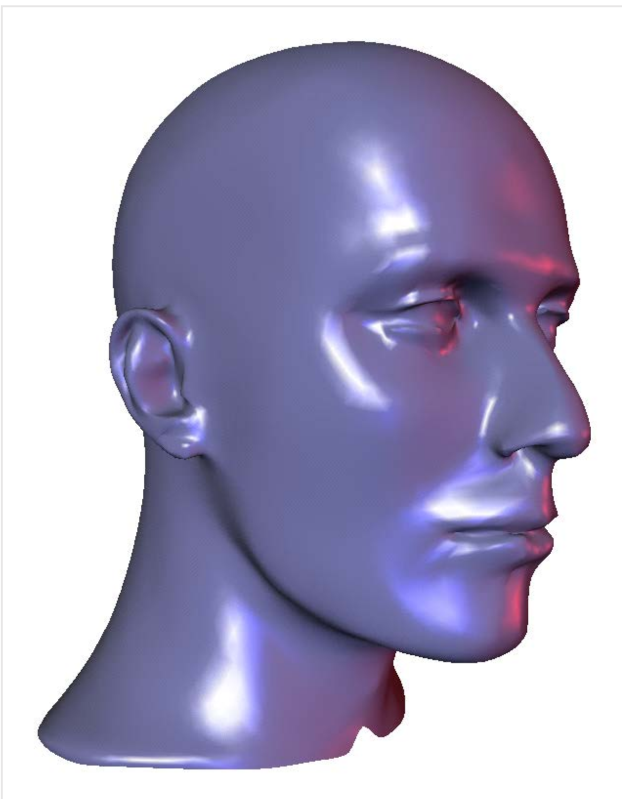


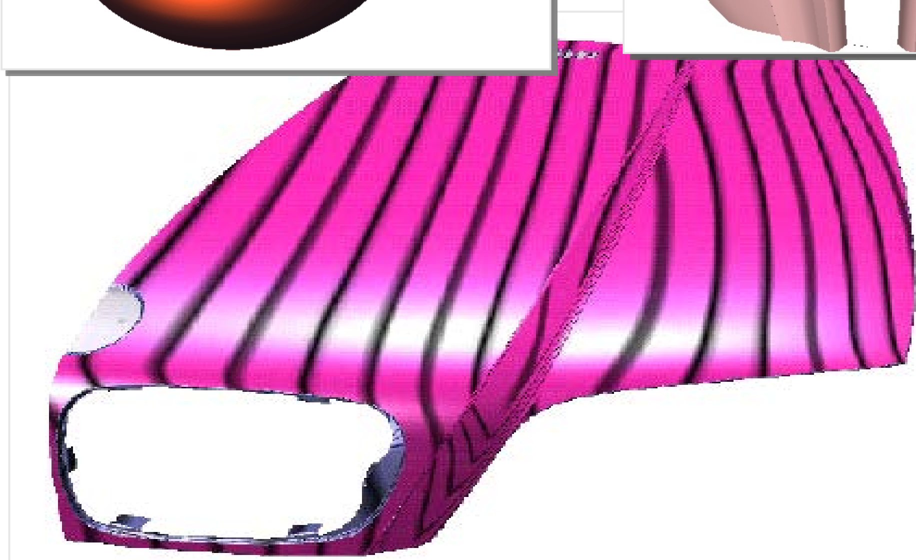
- polygon mesh to represent the object‘s boundary
- mesh only approximates boundary
- object‘s inside filled
  - → only look at it from outside, backface culling possible
- triangle mesh closed: **„watertight“**

Single Objects

- More complex geometries

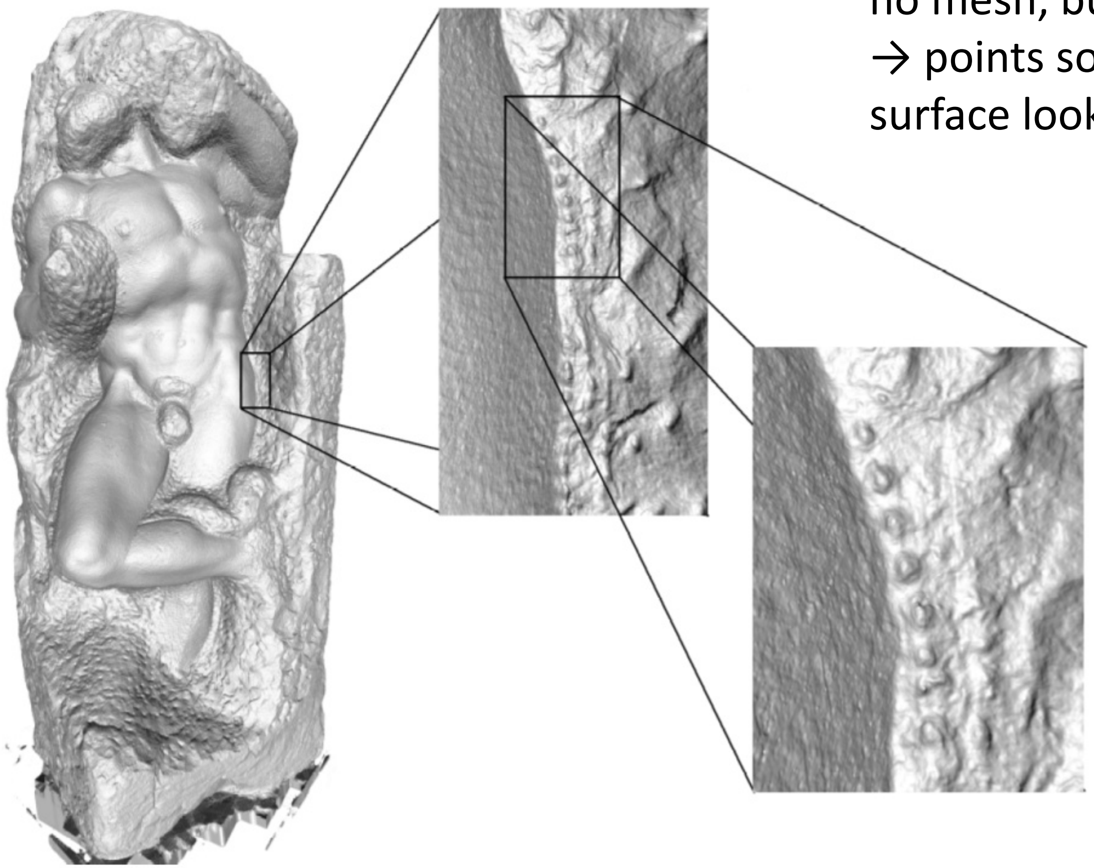






Single Objects

- Very large discrete surfaces



no mesh, but a **point cloud**
→ points so dense, that surface looks closed

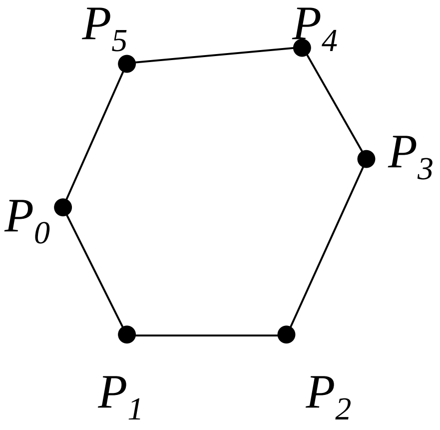
Michelangelo “Awakening” 381 million points

Polygon / Triangle meshes

- Polygons are most important in modeling for real-time graphics
  - Everything can be turned into polygons (almost everything)
  - We know how to render polygons quickly
  - Many operations are easy to do with polygons

- Polygon are specified by an ordered set of vertices (ideally points on a common plane)

$$P_0, P_1, P_2, \dots, P_{n-1}$$

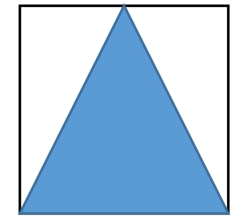


- Most often used:
  triangles $\rightarrow$ triangle meshes
  quadrilaterals $\rightarrow$ quad meshes

Remember from Lecture #4: Triangle meshes

- One single triangle:

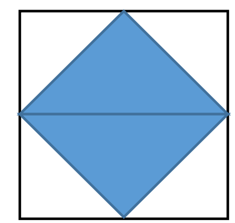
```
var v = [-1,-1, 1,-1, 0,1];
```



→ OpenGL coordinates go from -1 to 1 ! (for now)

- Two triangles:

```
var v = [-1,0, 1,0, 0,1, -1,0, 1,0, 0,-1];
```



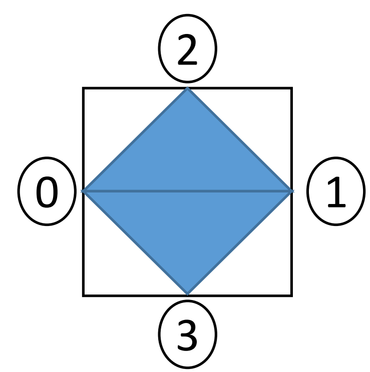
→ inefficient, because two vertices are used twice

- Indexed Face Set data structure:

```
var v = [-1,0, 1,0, 0,1, 0,-1];
var i = [0,1,2, 0,3,1];
```

vertex coordinates

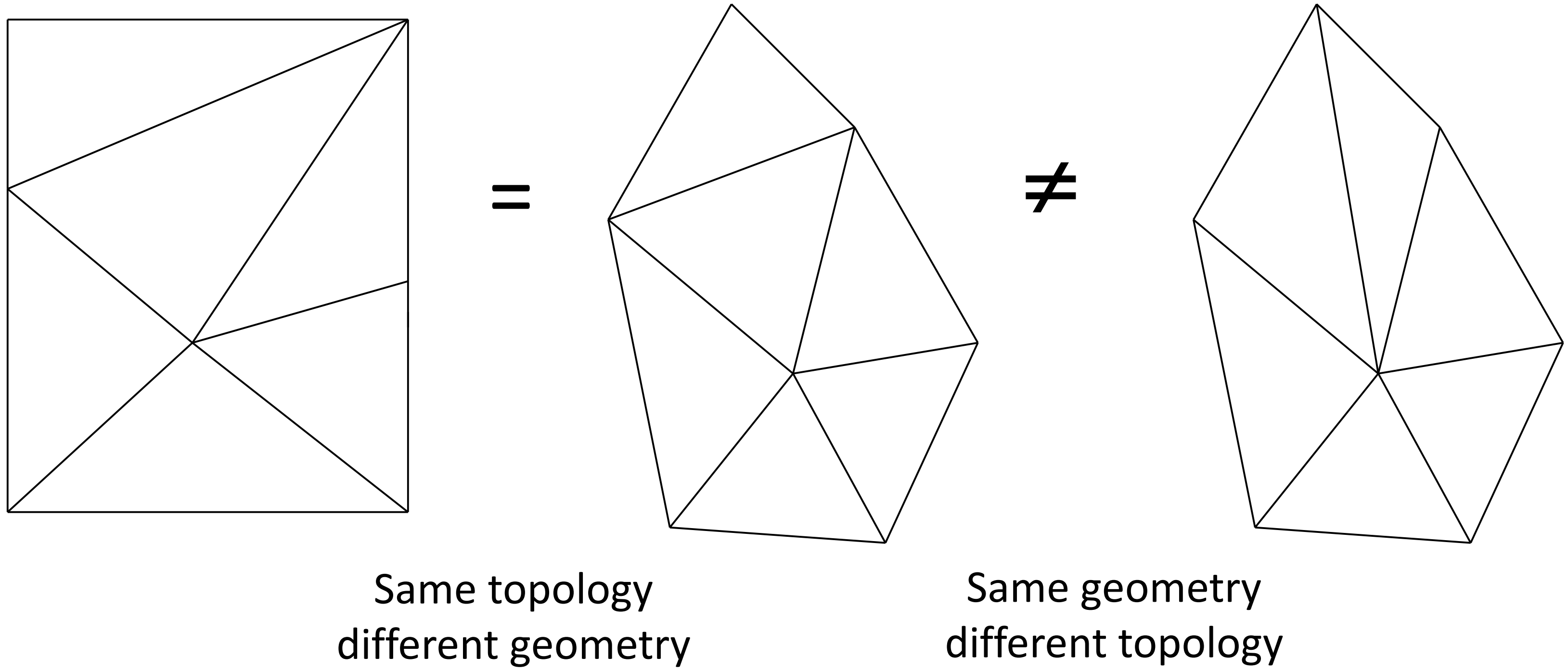
vertex indices per triangle



- → Needed for large scenes with many triangles (millions)

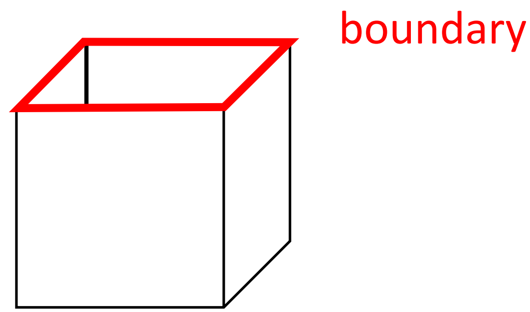
Polygon / Triangle meshes

- Distinguish between topology and geometry of a mesh.
  - Geometry: position of vertices (x,y,z-coordinates) (= vertex array)
  - Topology: neighborhood / connectivity relation (= index array)

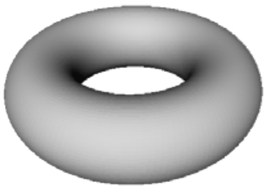


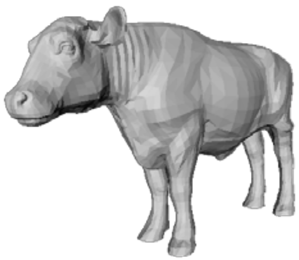
Polygon / Triangle meshes

- Topology:
  - with boundary / open surface

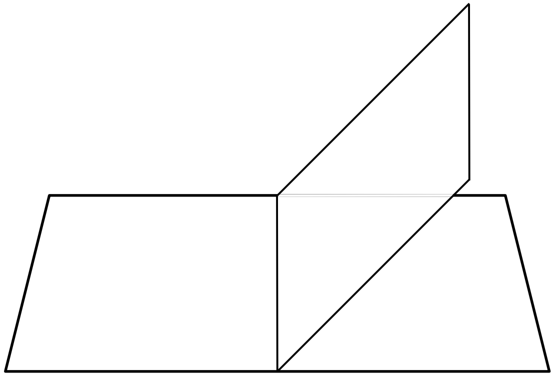


  - closed surface





  - Manifold / non-manifold



Three faces sharing an edge

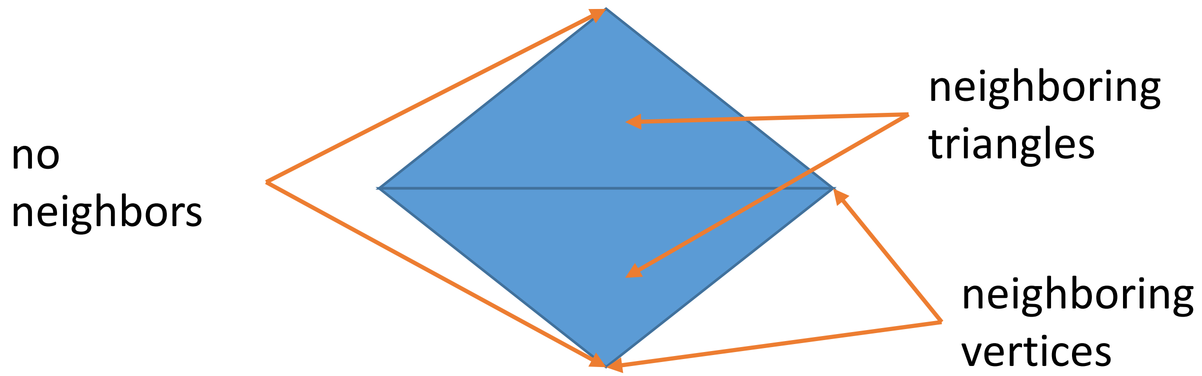
Triangle / Polygon meshes

- In the following, we will mainly consider **triangle meshes**
  → every polygon mesh can be transformed to triangle mesh
  → most general

- Today, we examine the **topology** of meshes
  - Open / closed meshes (watertight)
    - Can we see the inside of an object?
    - → Backface culling possible
  - Manifold meshes
    - Important for algorithms on mesh
  - Orientable meshes
    - can we define inside / outside
    - equals to: can we consistently assign normals?

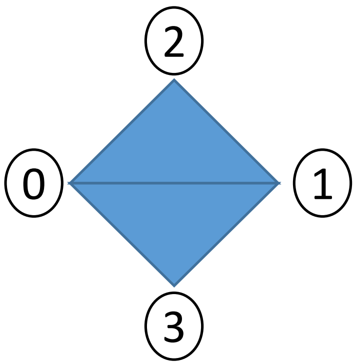
Triangle / Polygon meshes

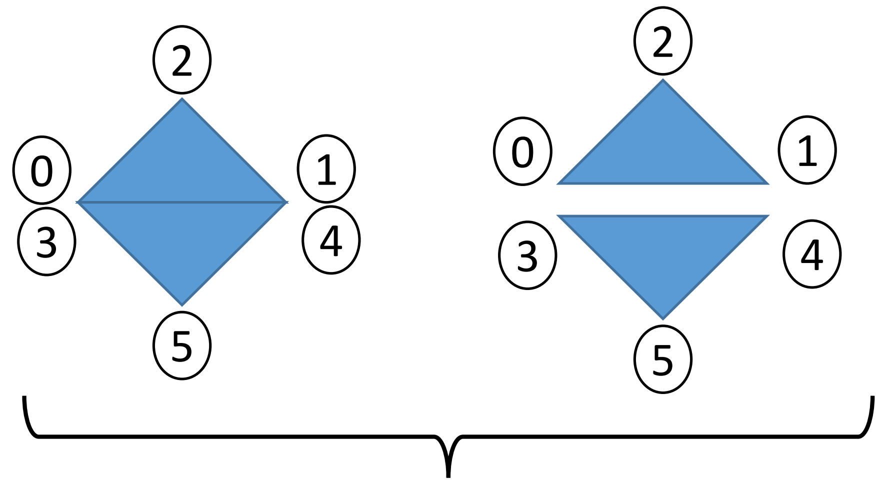
- Topology:
  - mesh consists of a **faces**, each face connects **vertices**
  - in a triangle mesh, the faces are **triangles** of three vertices
  - each triangle (face) defines three (n) **edges**, each edge connects two vertices
  - an edge can appear in multiple triangles (faces) (usually one or two)
- Adjacency / Neighborhood:
  - vertex and triangle are adjacent if vertex is one of the three triangle's vertices
  - edge and triangle are adjacent if edge is one of the three triangle's edges
  - triangles are neighbors if they share one edge
  - vertices are neighbors if they are connected by an edge



Triangle meshes

- Important: we only look at topology, i.e. at indices, not at the vertex position





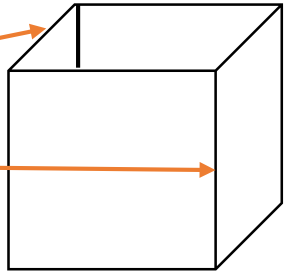
- index buffer:
  [0,1,2, 0,3,1]

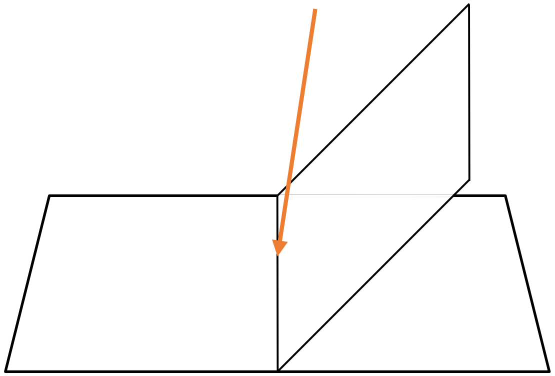
[0,1,2, 3,4,5]
→ topologically equivalent

- essentially the same mesh, but on the right the neighborhood is defined via the vertex positions (geometry), not via the index buffer
- we only look at the index buffer → **topology**

Triangle / Polygon meshes

- Open or closed ? = Does mesh have a boundary ?
- Look at edges of mesh
  - one adjacent face→ boundary
  - two adjacent face → inner edge
  - more than two adjacent faces → ???





Triangle / Polygon meshes

- Euler-Formula for general polyhedra:

$$V - E + F = 2(1 - G) - B$$

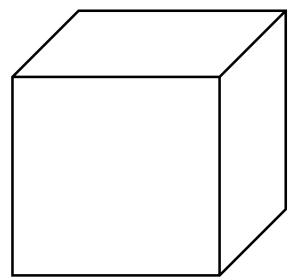
  where
  - V: number of vertices
  - E: number of edges
  - F: number of faces
  - G: **Genus** of the object (see later)
  - B: number of borders (see later)

Triangle / Polygon meshes

- Euler-Formula for General Polyhedra:

$$V - E + F = 2(1 - G) - B$$

- Example: cube
  - $V = 8$
  - $E = 12$
  - $F = 6$
  - $B = 0$
  - $G = 0$
  - $V - E + F = 2,\ 2(1 - G) - B = 2$ → okay

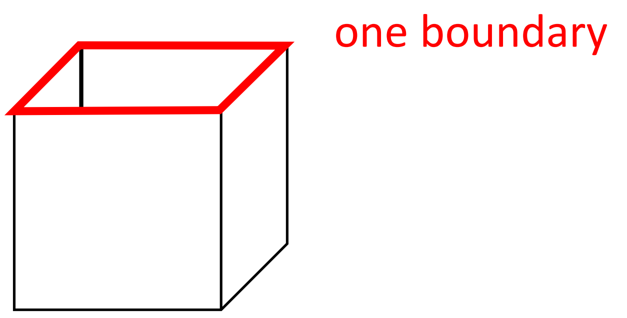


Triangle / Polygon meshes

- Euler-Formula for General Polyhedra:

$$V - E + F = 2(1 - G) - B$$

- Example: cube with one open face
  - $V = 8$
  - $E = 12$
  - $F = 5$
  - $B = 1$ (see figure)
  - $G = 0$
  - $V - E + F = 1, 2(1 - G) - B = 1$ → okay

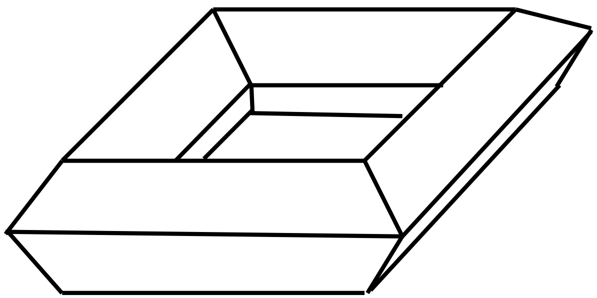


Triangle / Polygon meshes

- Euler-Formula for General Polyhedra:

$$V - E + F = 2(1 - G) - B$$

- Example: square ring
  - $V = 16$
  - $E = 32$
  - $F = 16$
  - $B = 0$
  - $G = 1$ (!)
  - $V - E + F = 0, 2(1 - G) - B = 0$ → okay



- Genus: number of “holes”
  - cube: genus = 0
  - ring / torus: genus = 1
  - “Breze” on the right: genus = 3

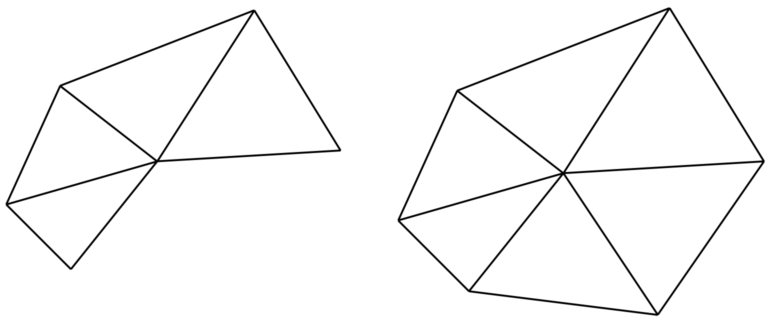
Triangle / Polygon meshes

- Not every possible index buffer forms a regular mesh
  - e.g. three faces adjacent to one edge
  - mesh should form a compact, connected surface
  - such a “regular” mesh is called **“manifold”**

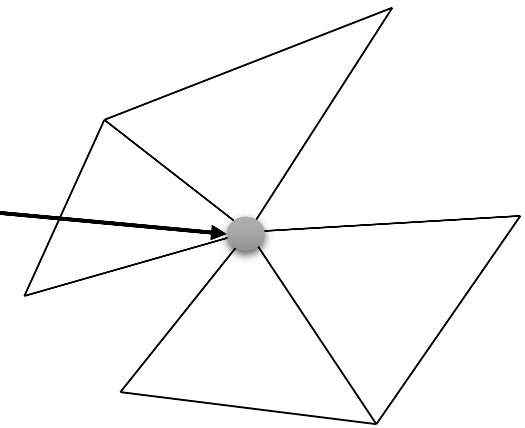
- A triangle mesh is called **manifold** if
  - the intersection of two triangles is either
    - empty, or
    - a common vertex, or
    - a common edge
  - edges have either
    - one adjacent triangle: border edge
    - two adjacent triangles: interior edge
  - For a border vertex the adjacent triangles form an open fan
  - For an inner vertex the adjacent triangles form a closed fan

Triangle mesh

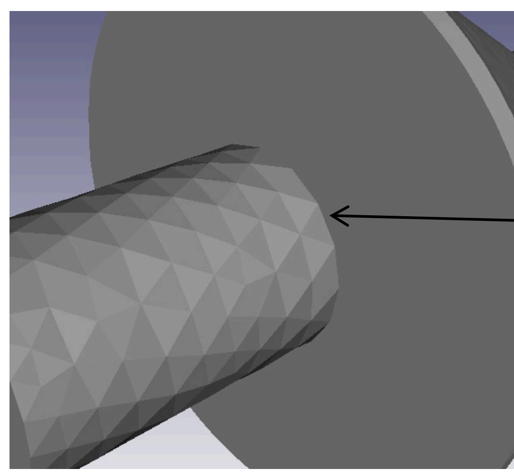
- open and closed fan around a vertex → non-manifold



non-manifold
vertex



- real-world example



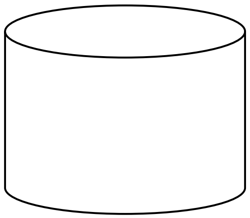
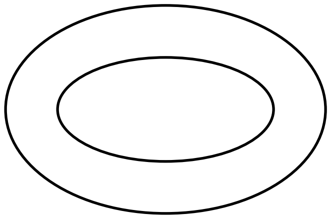
attached
→ three faces per edge

Surfaces

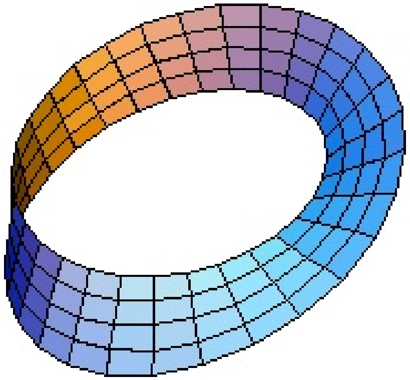
- For triangle meshes, being a manifold means
  - An edge has one or two neighbors
  - one can iterate over the triangles adjacent to a vertex in one iteration
  - … ???

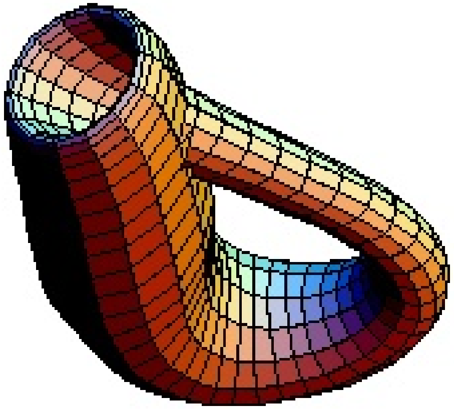
Surfaces

- Orientable surfaces

cylinder  torus 

- Non-orientable surfaces

Möbius strip 



Klein bottle
(see "Felix Klein")

Triangle Mesh Data Structures

- Applications
  - only rendering
    - Enumerate triangles one by one → simple
  - modification of mesh
    - Move vertices (animation) → simple
    - Modify mesh topology → tricky
  - performance for geometry (adjacency) queries
    - Triangles around vertex, neighbor vertices, triangle over edge, …

Simple Triangle Mesh

- Simple data structure

```
struct Vertex {
    float coords[3];
}

struct Triangle {
    struct Vertex verts[3];
}

struct Triangle mesh[n];
```

- multiple copies of the same vertex
- no adjacency information

Triangle strips

- Triangle Strips: Specify sequence of vertices (e.g. by indexing to a vertex list): P0, P1, P2, P3, P4, P5, …, Pn-1
- form triangles from three successive vertices.

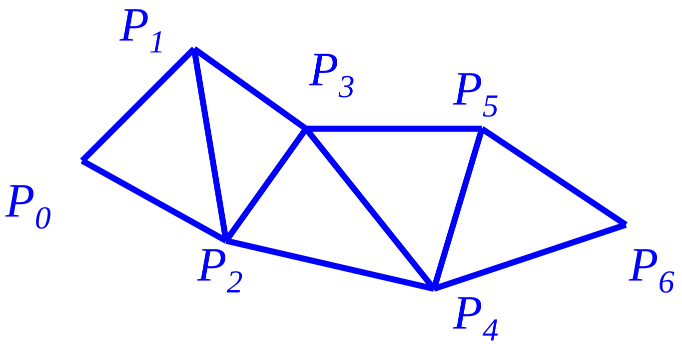
  P0, P1, P2;

  P1, P2, P3;

  P2, P3, P4;

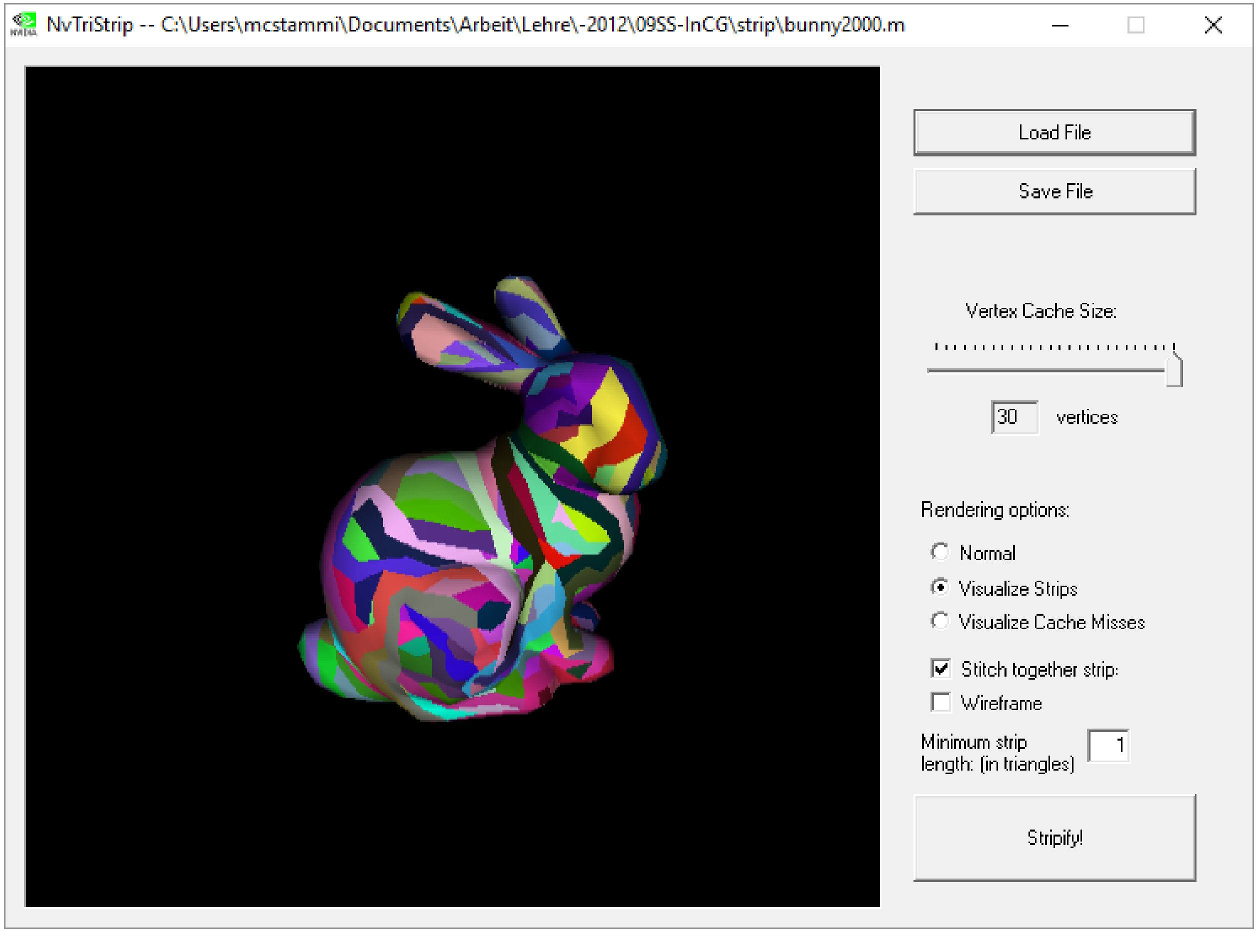
  P3, P4, P5;

  P4, P5, …



Triangle strips

- Stripification: represent triangular mesh as union of triangle strips

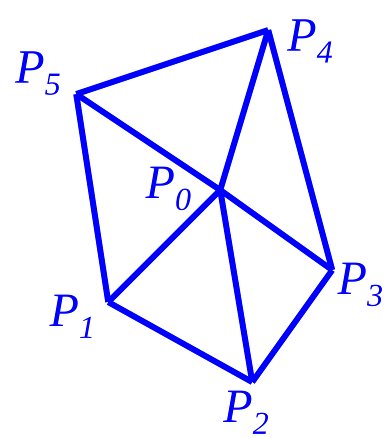


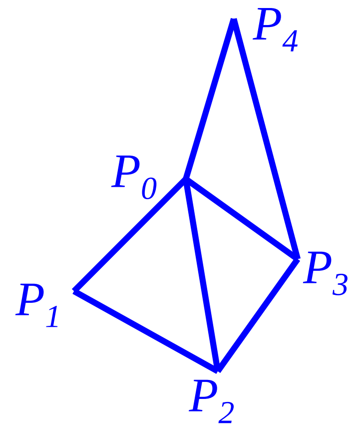
NVTriStrip transform mesh to set of triangle strips

Triangle fans

- Triangle Fan
  - Specify sequence of vertices (e.g. by indexing to a vertex list): P0, P1, P2, P3, P4, P5, …, Pn-1
  - form triangles from first (center) vertex and successive vertices.

  P0, P1, P2;
  P0, P2, P3;
  P0, P3, P4;
  P0, P4, P5;
  P0, P5, P1;





Quad strips

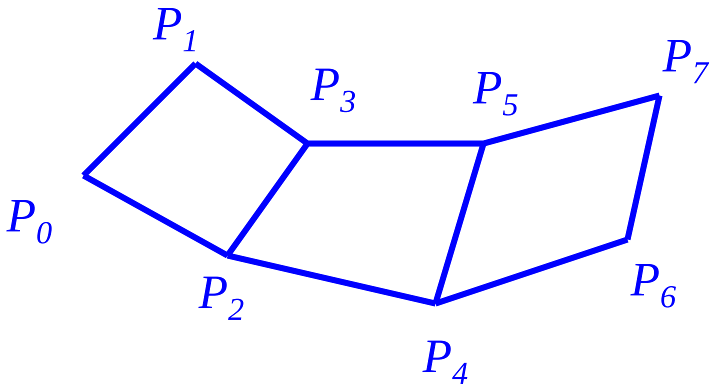
- Quad Strips
  - Specify sequence of vertices (e.g. by indexing to a vertex list
  - P0, P1, P2, P3, P4, P5, …, Pn-1
  - form quads of four successive vertices.

    P0, P1, P3 , P2;

    P2, P3 , P5, P4;

    P4, P5, P7 , P6;

    ……

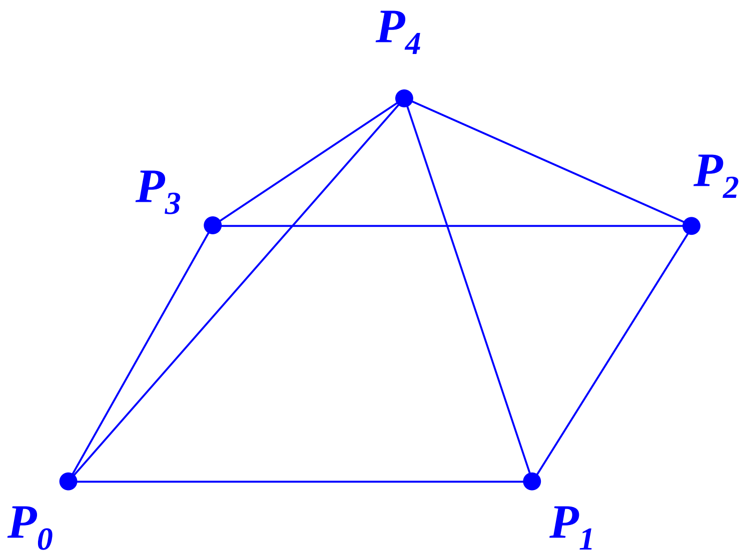


Indexed Face Set

- **Shared Vertex** or **Indexed Face set**
  - widely used
  - compact, simple, efficient
  - used in many file formats
- Two lists:
  - Vertex list (captures geometry)
  - Face list (captures topology)
- Problem: adjacency queries
  - need more sophisticates boundary representations (b-reps), explicit model of vertices, edges and faces with adjacency information

Indexed Face Set

vertex list	*face list*
0 : x_0, y_0, z_0;	0, 1, 4;
1 : x_1, y_1, z_1;	1, 2, 4;
2 : x_2, y_2, z_2;	2, 3, 4;
3 : x_3, y_3, z_3;	3, 0, 4;
4 : x_4, y_4, z_4;	3, 2, 1, 0;



Example: pyramid

Indexed Face Set

- Vertices often have multiple attributes
  - Position
  - Normal (for lighting)
  - Colors
  - Texture coordinates
- Two possibilities
  - Geometry array also contains these values, one index per triangle vertex
  - Separate arrays for each attribute, one index into each array for any triangle vertex

Indexed Face Set

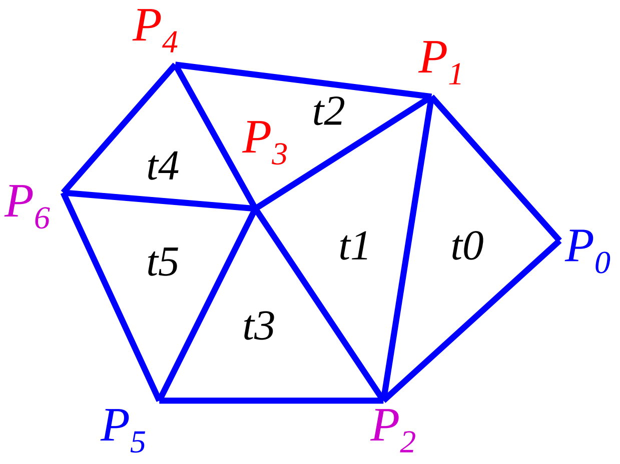
- Navigation in meshes is difficult (e.g. find neighboring triangle)
- Enhance face list by references to neighboring triangles

vertex list

0 : x_0, y_0, z_0;
1 : x_1, y_1, z_1;
2 : x_2, y_2, z_2;
3 : x_3, y_3, z_3;
4 : x_4, y_4, z_4;
5 : x_5, y_5, z_5;
6 : x_6, y_6, z_6;

enhanced face list

0, 1, 2; 1,-1,-1;
1, 3, 2; 3, 0, 2;
1, 4, 3; 4, 1,-1;
3, 5, 2;-1, 1, 5;
4, 6, 3; 5, 2,-1;
6, 5, 3; 3, 4,-1;



File Formats for Polygon Meshes

- Mostly based on shared vertex
  - X3D: ISO standard XML based file format for representing 3D geometric objects. Successor of VRML (VRML 1.0 derived from OpenInventor)
    → X3DOM in current advanced exercise
  - OBJ: file format developed by Wavefront Technologies for representing 3D geometric objects, including normals and texture coordinates
  - …

File Formats for Polygon Meshes

```
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
vn -0.000000 -1.000000 0.000000
vn 0.000000 1.000000 -0.000000
vn 1.000000 0.000000 0.000000
vn -0.000000 -0.000000 1.000000
vn -1.000000 -0.000000 -0.000000
vn 0.000000 0.000000 -1.000000
f 1//1 2//1 3//1 4//1
f 5//2 8//2 7//2 6//2
f 1//3 5//3 6//3 2//3
f 2//4 6//4 7//4 3//4
f 3//5 7//5 8//5 4//5
f 5//6 1//6 4//6 8//6
```

OBJ file format

Vertex positions

Vertex normals

Topology 3//1 means:

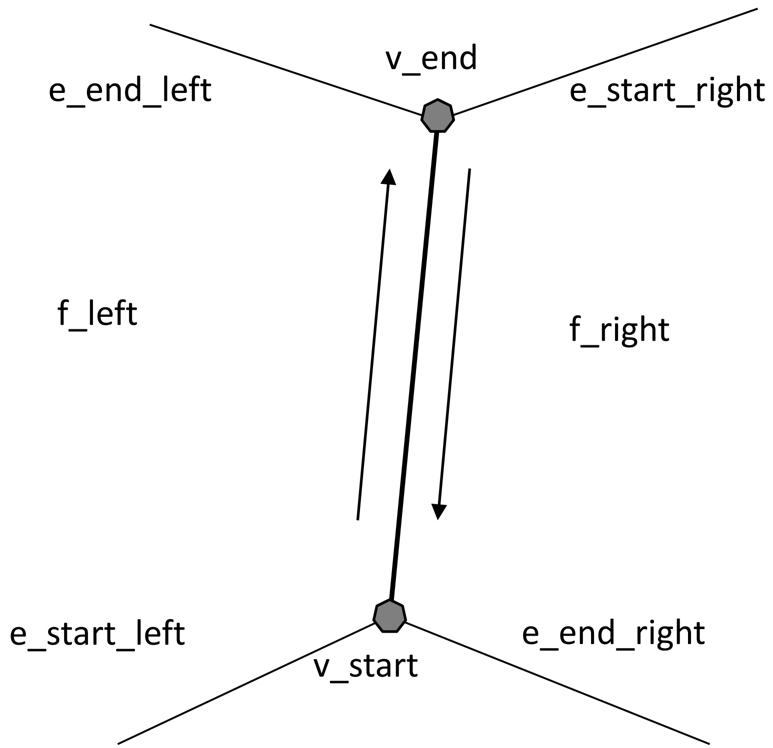
vertex with 3rd position and 1st normal

Winged Edge Data Structure

- probably the oldest b-rep is the winged edge data structure, Baumgart 1975
- store information per vertex, per edge, and per face
- lots of data, but easy to traverse, e.g.:
  - find all edges around a face
  - find all neighbor vertices
  - …

```
Vertex table
•  vertex coordinates
•  incident edge (one of the adjacent edges)
Edge table
•  start and end vertex
•  start and end edge, when traversing left face
•  start and end edge when traversing right face
•  neighboring faces
Face table
•  incident edge (one of the adjacent edges)
```

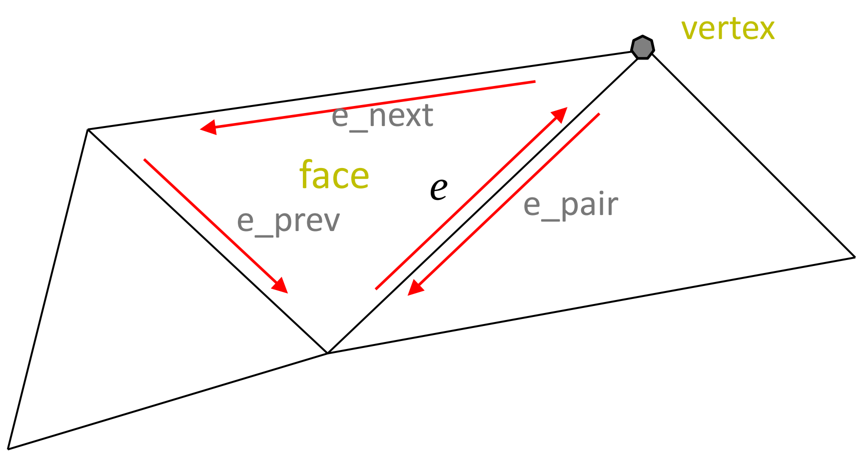
Winged Edge Data Structure



per edge information

Half-Edge Data Structure

- Eastman, 1982
- widely used in geometric computations
- also known as doubly connected edge list
- sophisticated b-rep, allows geometric queries in constant time
- similar to winged edge, but information per edge is split into two half-edges
- half-edge: information for one side / direction of one edge



half edge information

oriented counter-clockwise

Half-Edge Data Structure

- Data Structure: Mesh

```
Edge table
    • end vertex
    • oppositely oriented adjacent edge
    • adjacent face
    • next edge, can be extended including previous edge
Vertex table
    • incident edge
Face table
    • incident edge
```

Half-Edge Data Structure

```
struct HE_edge
{
    HE_vert* vert; // vertex at the end of edge
    HE_edge* pari; // opposite edge
    HE_face* face; // border (adjacent) face
    HE_edge* next; // next half edge around the face
};

struct HE_vert
{
    float x,y,z; // vertex coords
    HE_edge* edge; // half-edge emanating from vertex
};

struct HE_face
{
    HE_edge* edge; // half-edge bordering face
};
```

Max McGuire, Aug 2000

Half-Edge Data Structure

- Queries
  - walk around edges of a given face
  - find adjacent edges of a vertex
  - find adjacent faces to a vertex
  - find adjacent vertex to a face
  - find neighbor faces to a face

rather simple to implement

- Complexity
  - most of the queries are O(1)

Half-Edge Data Structure

- Query: find all edges around a face

```
HE_edge* edge = face->edge;

do {
   // do the job with the edge
   edge = edge->next;
} while (edge != face->edge)
```

- Query: find edges adjacent to a vertex

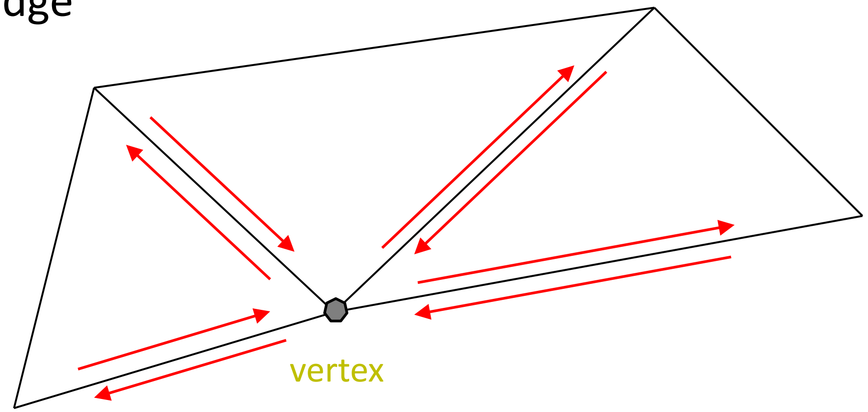
```
HE_edge* edge = vert->edge;

do {
   // do the job with the edge or edge->pair
   edge = edge->pair->next;
} while (edge != vert->edge)
```


Max McGuire, Aug 2000

Half-Edge Data Structure

- If vertex is on boundary, then the edge stored with the vertex should be the boundary edge, so that enumeration of all neighboring edges remains simple
- remember: only one open fan around vertex allowed (otherwise non-manifold)

- open triangle fan
  - vertex points to emanating edge
  - choose edge at the border



Iteration to find all edges adjacent to a vertex

Half-Edge Data Structure

- Restrictions
  - represent only planar maps or in general orientable manifold meshes
  - No non-orientable surfaces (why?)
  - No non-manifold surfaces (why?)
  - generation of data structure is cumbersome (and time consuming)

- Remarks
  - winged-edge data structure: manages non-orientable surfaces
  - indexed face set data structure: manages non-manifold surfaces

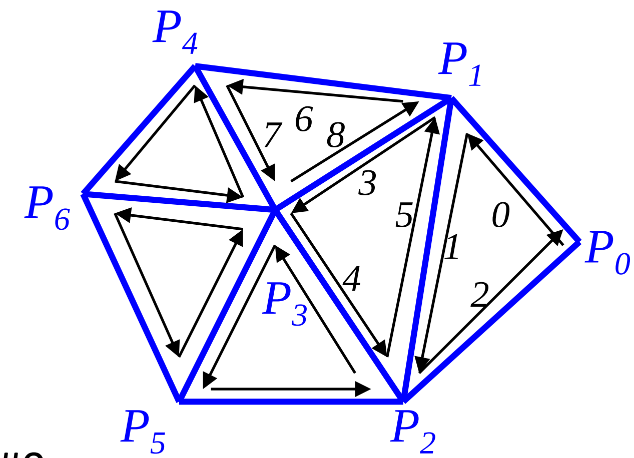
Directed Edges

- Half-edge data structure for triangle meshes
- Vertex list + list of half edges
- always three successive edges form triangle
  → triangle for an edge = edge index / 3
  → next edge = 3*(edge index/3) + (edge index +1) % 3
  → previous edge = 3*(edge index/3) + (edge index +2) % 3
  → …
- If needed: store one outgoing half edge per vertex

- Restrictions
  - Only triangle meshes
  - Only manifold meshes

Directed Edges

- Example



vertex list

0 : x_0, y_0, z_0;
1 : x_1, y_1, z_1;
2 : x_2, y_2, z_2;
3 : x_3, y_3, z_3;
4 : x_4, y_4, z_4;
5 : x_5, y_5, z_5;
6 : x_6, y_6, z_6;

edge list

0 : 0,-1;
1 : 1, 5;
2 : 2,-1;

triangle #0

3 : 1, 8;
4 : 3, x;
5 : 2, 1;

triangle #1

6 : 1,-1;
7 : 4, x;
8 : 3, 3;

…

9 : ….