Advanced Computer Graphics
Boundary Representations for Graphical Models

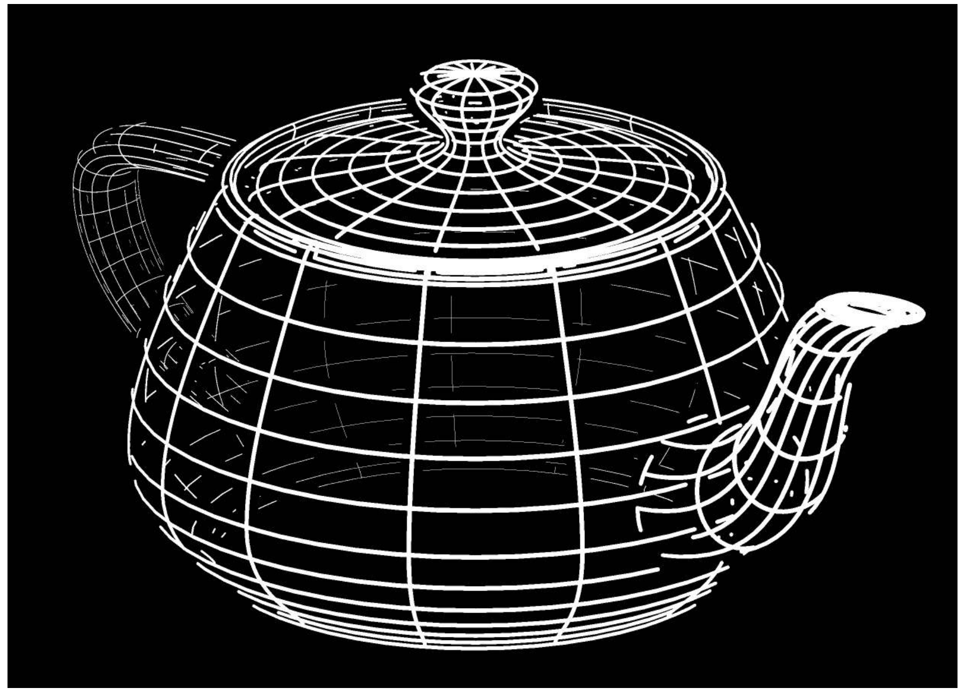
G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

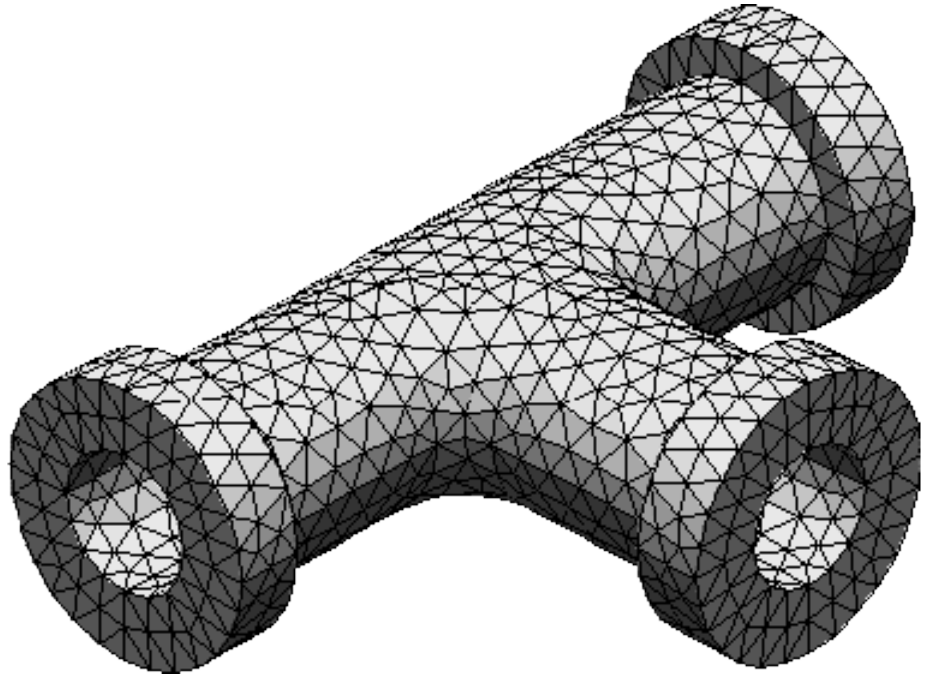
The Problem

- How to store objects in versatile and efficient data structures?

- Definition *Boundary Representation* (*B-Rep*): Objects "consist" of
  1. Triangles, quadrangles, and polygons, i.e., geometry; and
  2. Incidence and adjacency relationships, i.e., connectivity ("topology")

- By contrast, there are also representations that try to model the volume directly, or that consist only of individual points



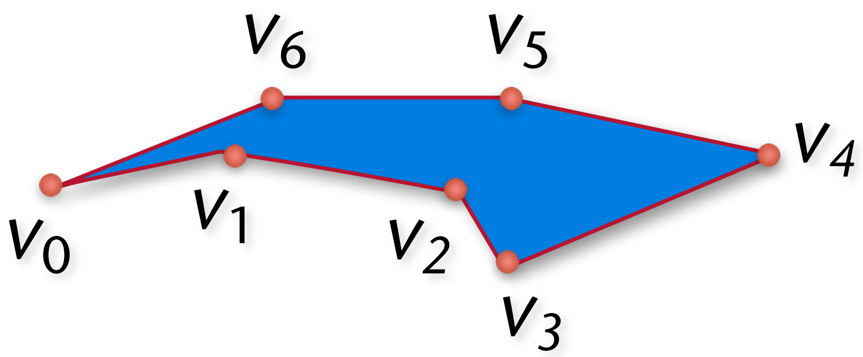


Definitions: Graphs

- A graph is a pair $G=(V, E)$, where $V=\{v_0,v_1,\ldots,v_{n-1}\}$ is a non-empty set of n different nodes (points, vertices) and E is a set of edges (v_i, v_j)
- When V is a (discrete) subset of $\mathbb{R}^d$ with $d \geq 2$, then $G=(V, E)$ is called a geometric graph
- Two edges/nodes are called neighboring or adjacent, iff they share a common node/edge
- If $e=(v_i, v_j)$ is an edge in G, then e and v_i are called incident (dito for e und v_j ; v_i and v_j are called neighboring or adjacent)
- In the following, edges will be *undirected* edges, and consequently we will denote them just by $v_i v_j$
- The degree of a node/vertex := number of incident edges

Polygons

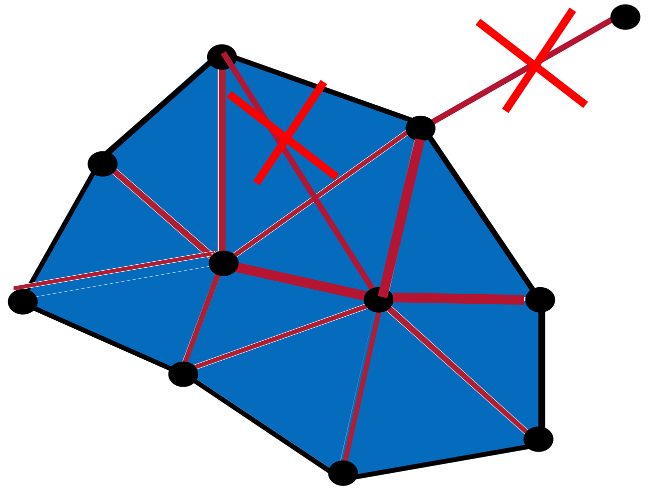
- A polygon is a geometric graph P=(V, E), with nodes $V=\{v_0,v_1,\ldots,v_{n-1}\} \subset \mathbb{R}^d$, $d \geq 2$, and edges $E=\{ (v_0,v_1), \ldots, (v_{n-1}, v_0) \}$, such that all nodes lie in the same plane.
  - Nodes are called vertices (sg. vertex)

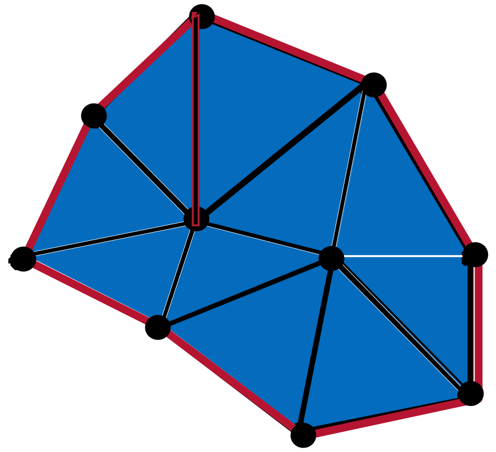


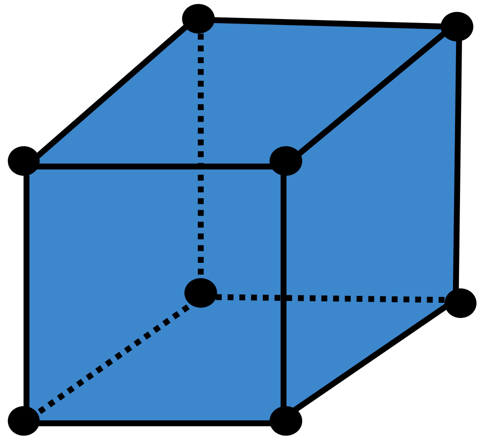
- Almost always we also require it to be simple = the *intersection of every two edges* in E is either empty or a vertex in V, and every vertex is incident to exactly two edges (i.e., the polygon does not have self-intersections).

Mesh (Polygonal Mesh)

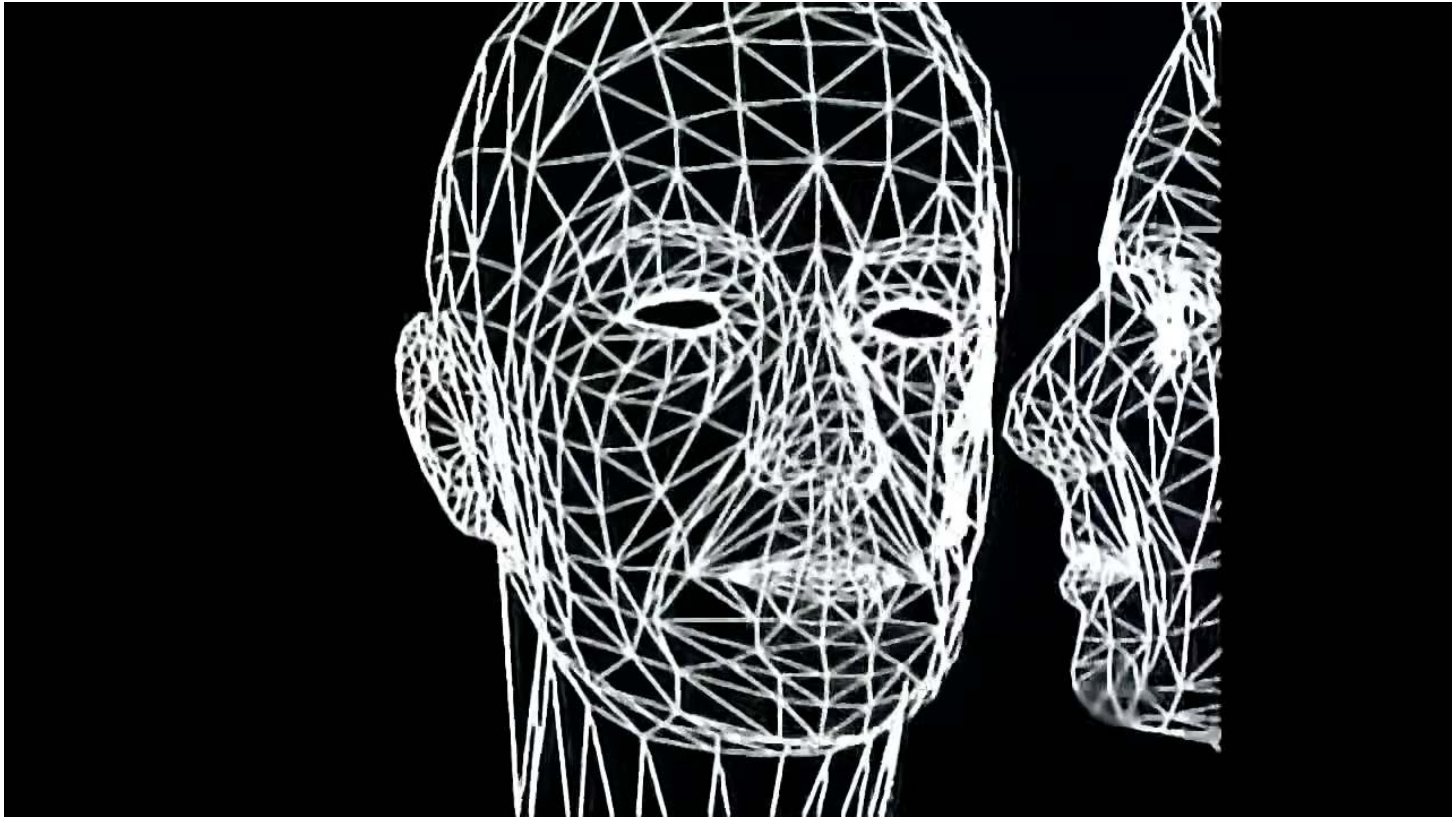
- Let M be a set of simple polygons P_i;
  let $V = \bigcup_i V_i \quad E = \bigcup_i E_i$
- M is called a mesh iff
  - the intersection of two polygons in M is either empty, a point $v \in V$, or an edge $e \in E$; and
  - each edge $e \in E$ belongs to at least one polygon (no dangling edges)
- The set of all edges, belonging to one polygon only, is called the border of the mesh
- A mesh with no border is called a closed mesh
- The set of all points V and edges E of a mesh constitute a graph







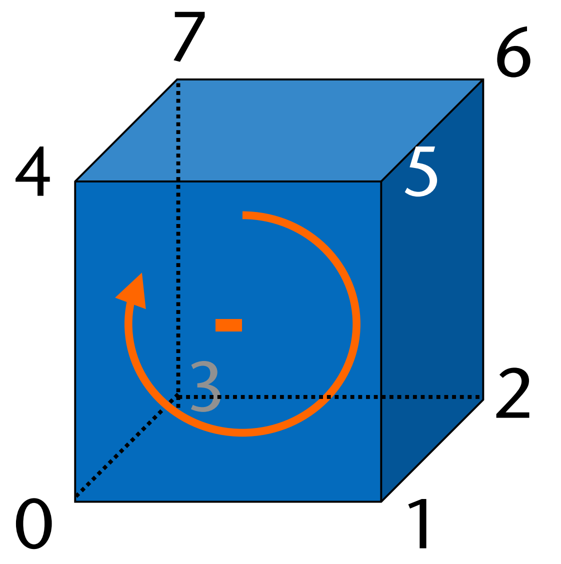
First Explicit Application of a Mesh for a Music Video :-)

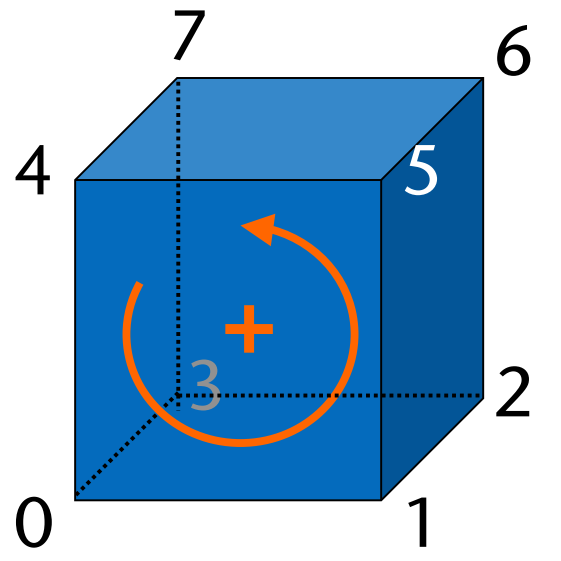


Kraftwerk: *Musique non stop*, 1986. Music video by Rebecca Allen.

Orientation

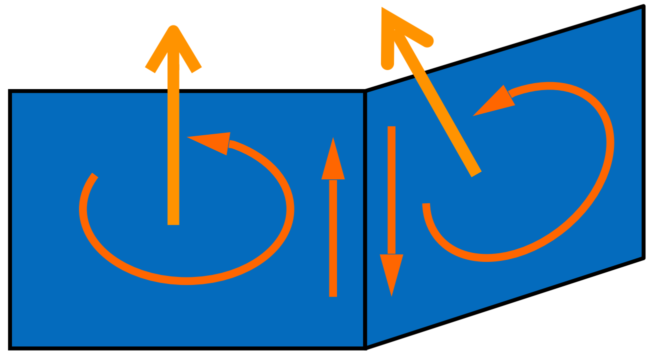
- Each polygon of a mesh can be oriented by the vertex order in which its vertices are listed
    - Positive front face = (0, 1, 5, 4)
    - Negative front face = (0, 4, 5, 1)



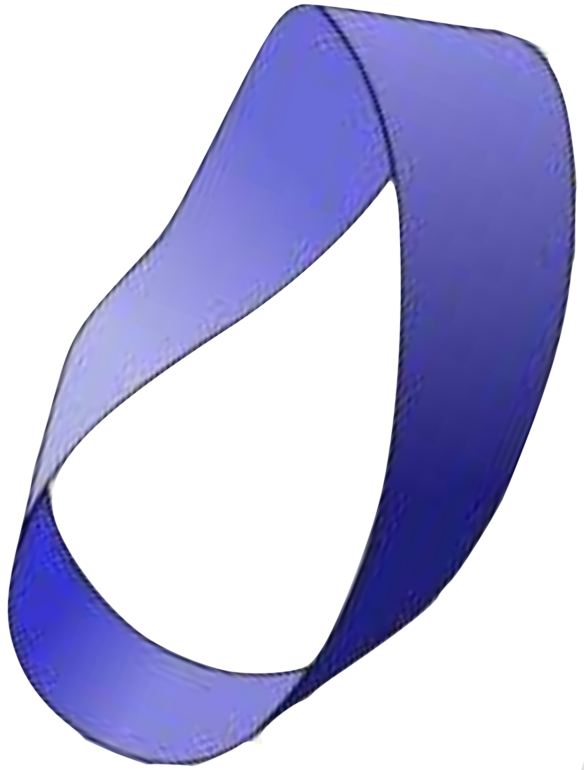


- Two adjacent polygons have the same orientation, if the common edge is traversed in *opposite* directions, when the two polygons are traversed according to their orientation

- The orientation determines the surface normal of a polygon. By convention, it is obtained using the right-hand-rule



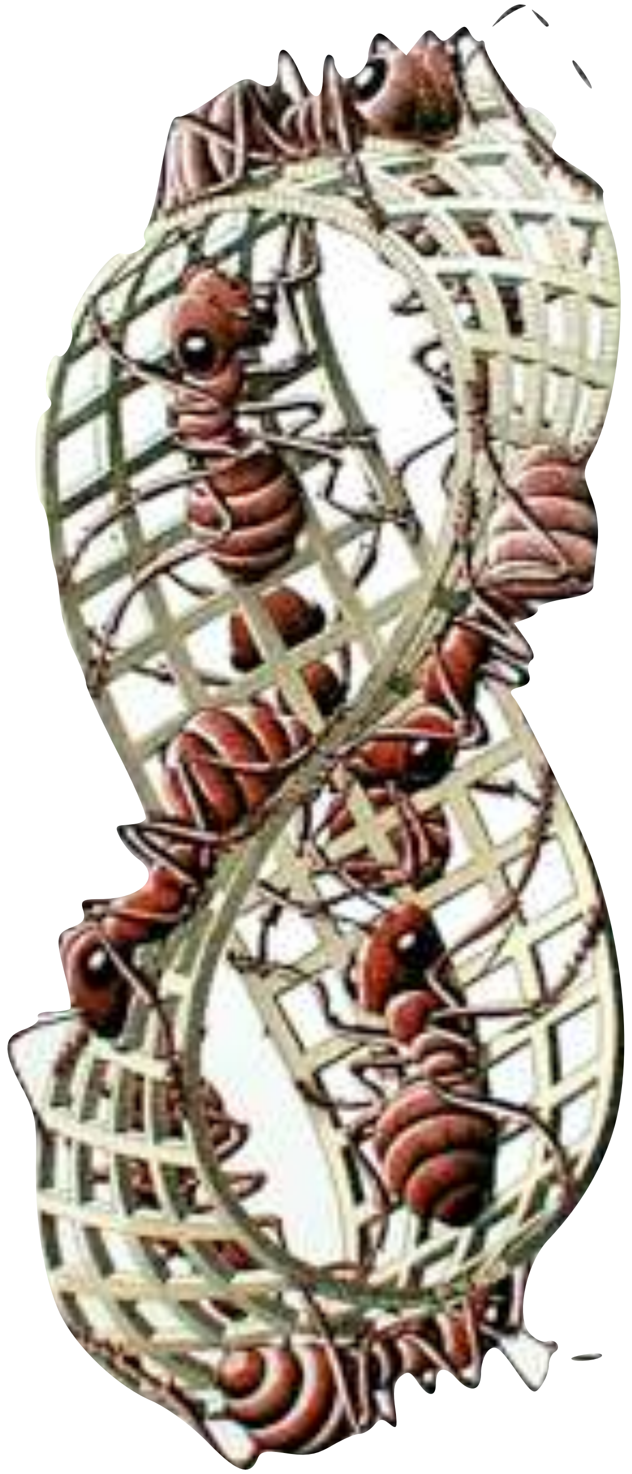
- A mesh is called orientable, if all polygons *can* be oriented such that every two adjacent polygons have the same orientation
  - The mesh is called oriented, if all polygons actually do have the same orientation
- A mesh is called non-orientable, if there are always two adjacent polygons that have opposite orientation, no matter how the orientation of all polygons is chosen



- Theorems (w/o proof):
  - Each *non-orientable* and closed surface that is embedded in *three-dimensional* space must have a self-intersection.

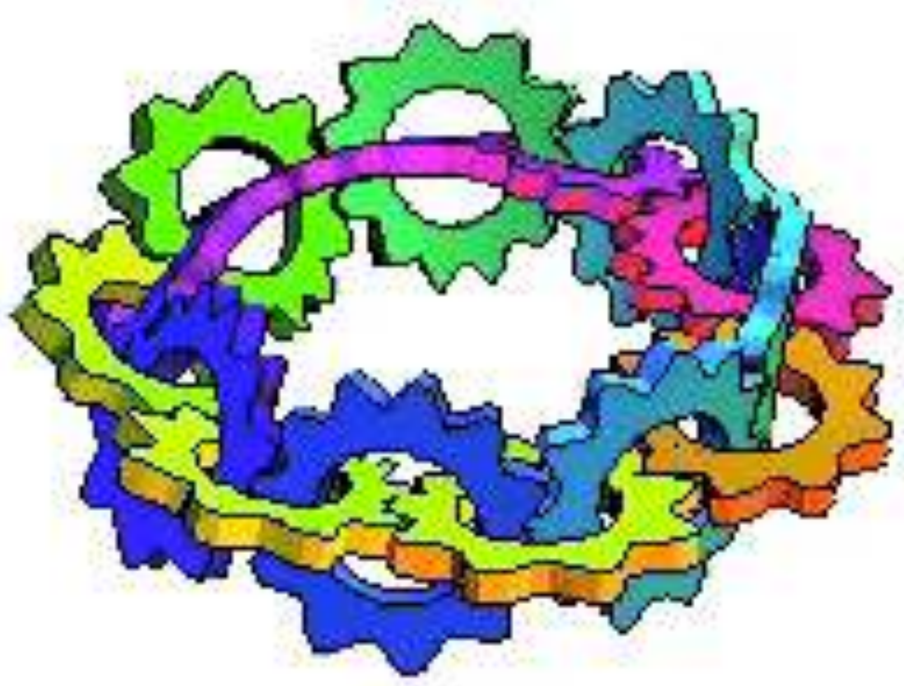
➢ The surface of a polyhedron is always orientable

Digression: the Möbius Strip in the Arts



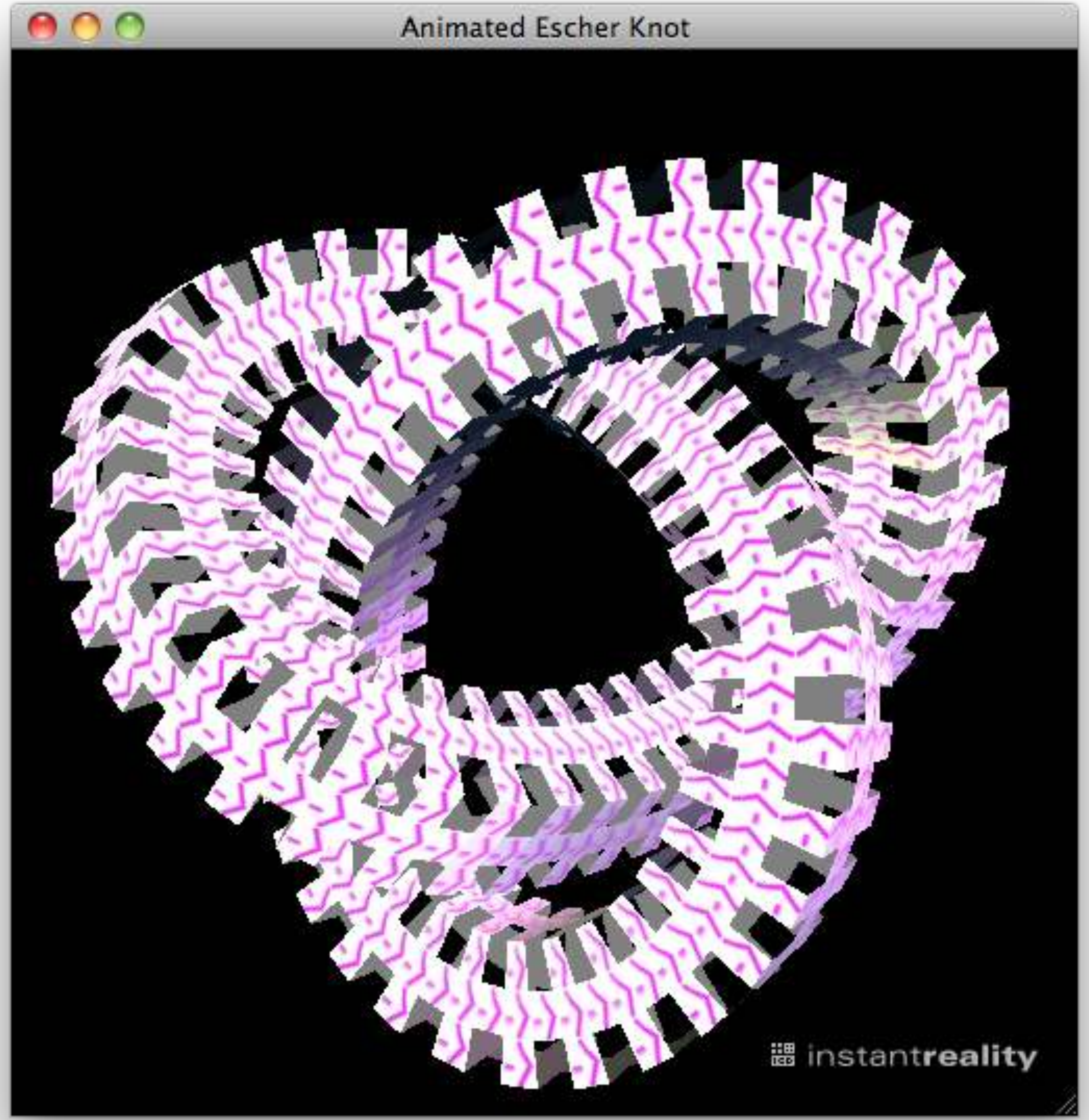
Möbius Strip II, woodcut, 1963

Max Bill



Interlocked Gears,
Michael Trott,
2001

Is the Escher Knot an Orientable Mesh or Not?

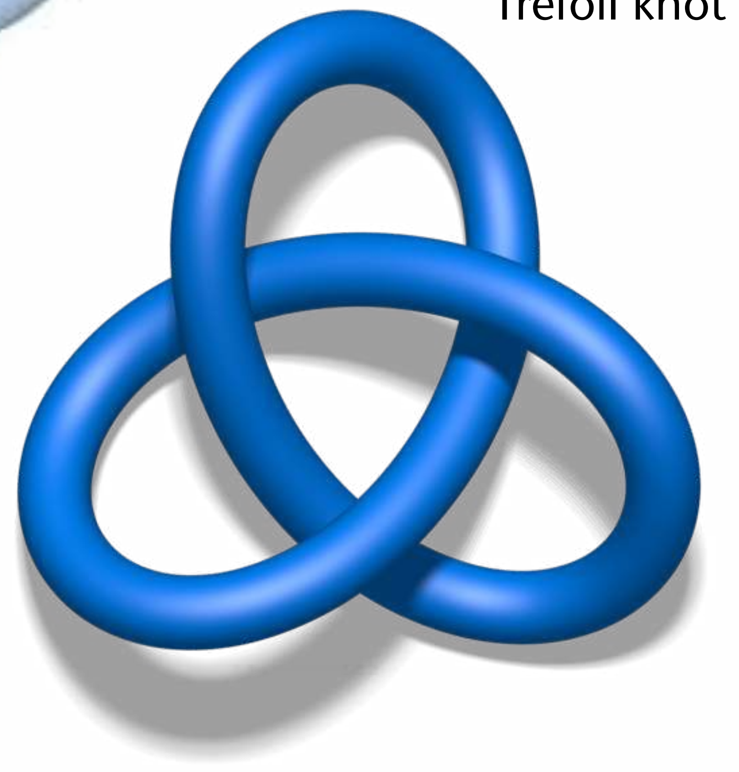


http://homepages.sover.net/~tlongtin

Definition: Homeomorphism

- Homeomorphism = bijective, continuous mapping between two "objects" (e.g. surfaces), the inverse mapping of which must be continuous, too
  - Two objects are called homeomorph iff there is a homeomorphism between the two
- Colloquial illustrations:
  - Squishing, stretching, twisting is allowed
  - Making holes is *not* allowed
  - Cutting is allowed *only*, if the object is glued together afterwards at *exactly* the same place
- Note: don't confuse this with *homomorphism* or *homotopy*!

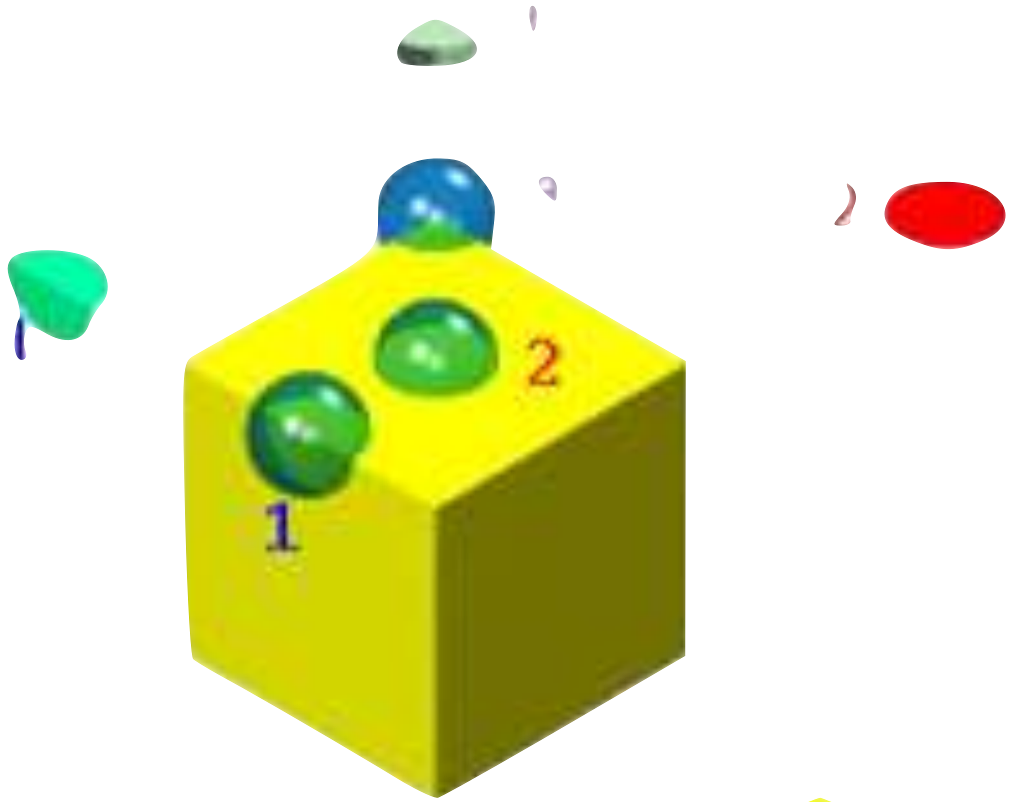
- Homeomorphic objects are also called topologically equivalent
- Examples:
  - Disc and square
  - Cup and torus
  - An object and its mirror object
  - *Trefoil knot* and …. ?
  - The *border* of the Möbius strip and … ?
- All *convex* polyhedra are homeomorphic to a sphere
- Many non-convex ones are topologically equivalent to the sphere, too

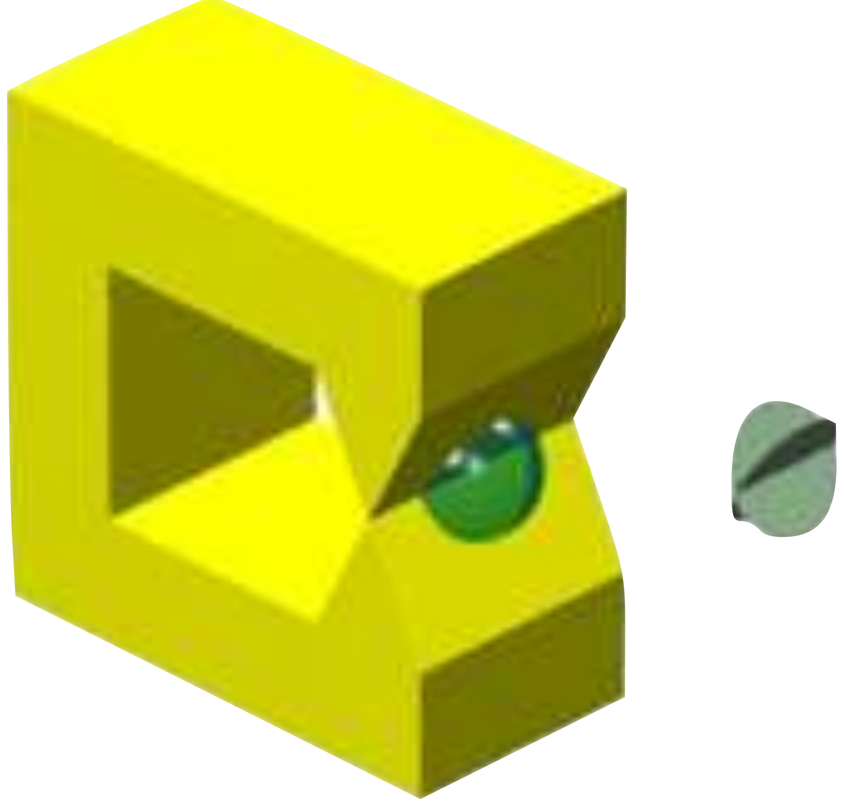


Trefoil knot

Two-Manifolds (Zwei-Mannigfaltigkeiten)

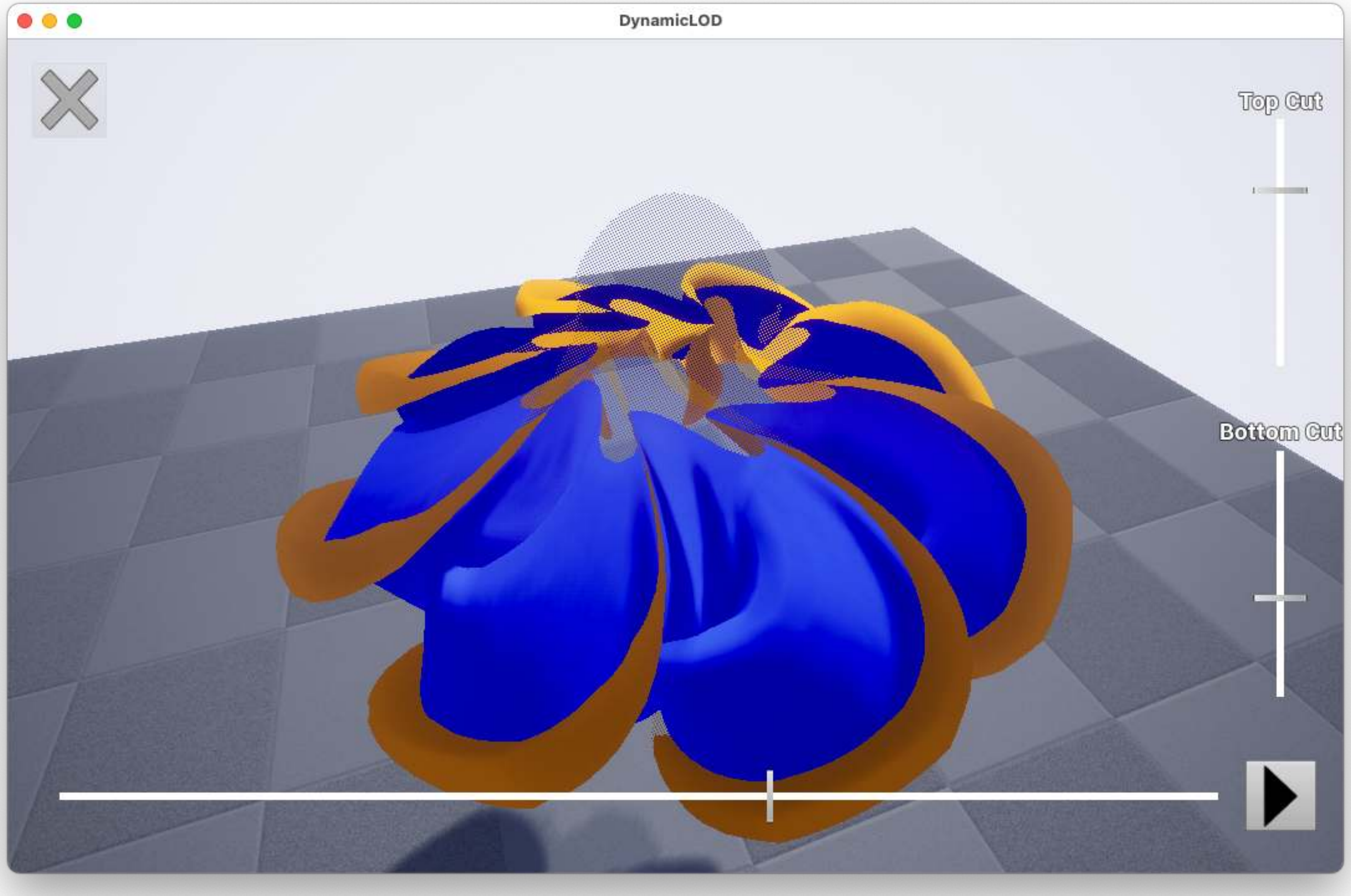
- Definition: a surface is called two-manifold, iff for each point on the surface there is an open ball such that the intersection of the ball and the surface is topologically equivalent to a two-dimensional disc
- Examples:
- Notice: in computer graphics, often the term "manifold" is used when 2-manifold is meant!
- The term "*piecewise linear manifold*" is sometimes used by people, to denote just a mesh ...



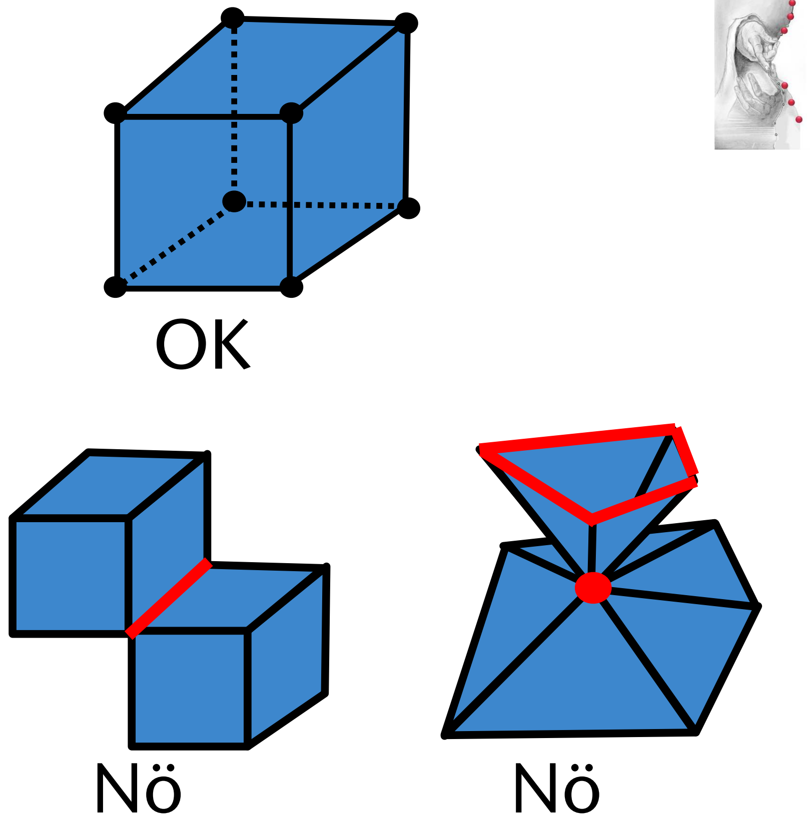


Digression: Sphere Eversion

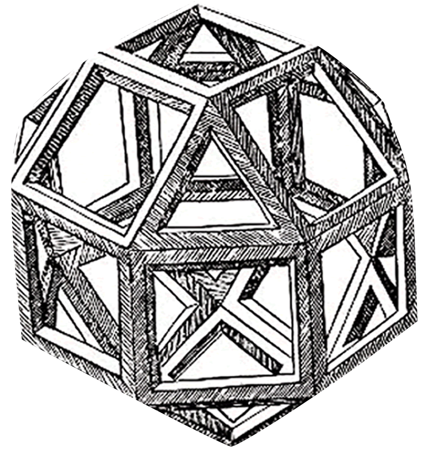
The rules of the game



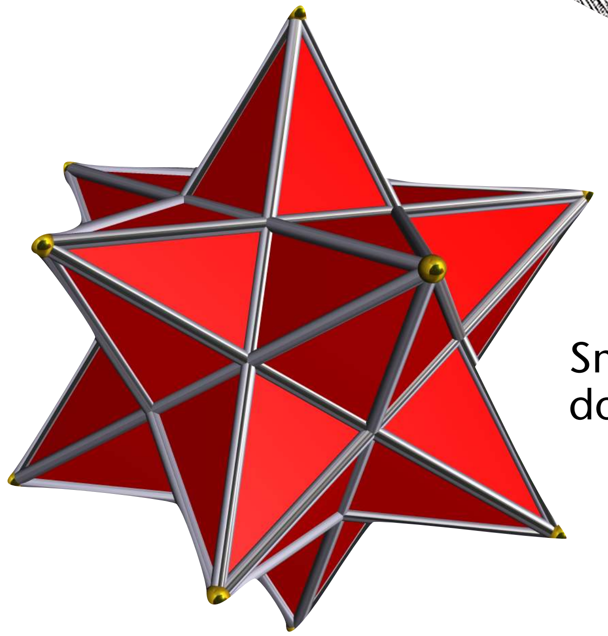
Definition of Polyhedron



- A polyhedron (in CG) is a polygonal mesh in 3D that is
  1. closed,
  2. two-manifold, and
  3. has no self-intersection (sometimes dropped).
- The polygons are also called *facets / faces* (Facetten)
- From the earlier theorem it follows that polyhedra are always orientable
- Jordan Curve Theorem (w/o proof):
  Every topological sphere (e.g. polyhedron) partitions space into three subsets: surface, interior, and (unbounded) exterior.
- Warning: definitions differ depending on context!



Leonardo da Vinci

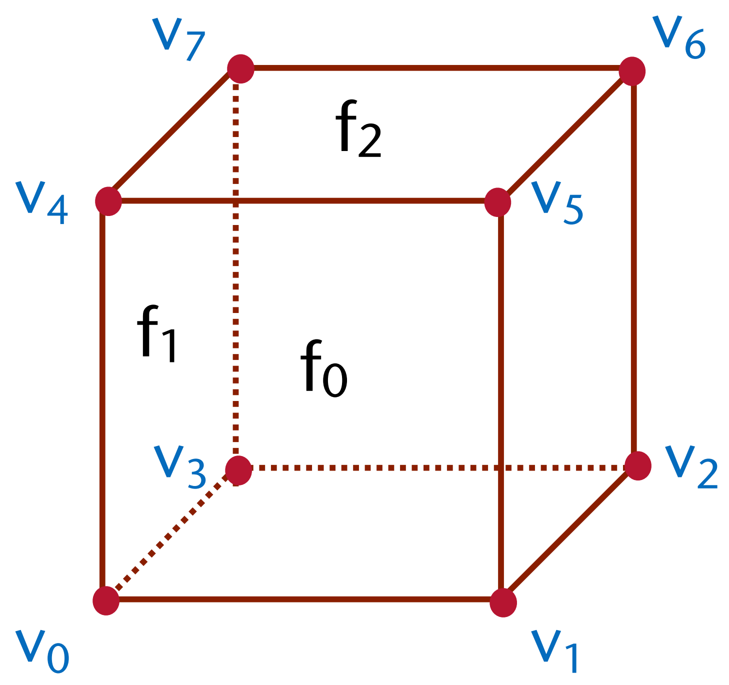


Small stellated dodecahedron

The Most Naive Data Structure for Meshes

- Array of polygons; each polygon = array of vertices
- Example:

face[0] =	*face[1]* =	*face[2]* =	
$x_0\ y_0\ z_0$	$x_0\ y_0\ z_0$	$x_4\ y_4\ z_4$	
$x_1\ y_1\ z_1$	$x_4\ y_4\ z_4$	$x_5\ y_5\ z_5$	. . .
$x_5\ y_5\ z_5$	$x_7\ y_7\ z_7$	$x_6\ y_6\ z_6$	
$x_4\ y_4\ z_4$	$x_3\ y_3\ z_3$	$x_7\ y_7\ z_7$	



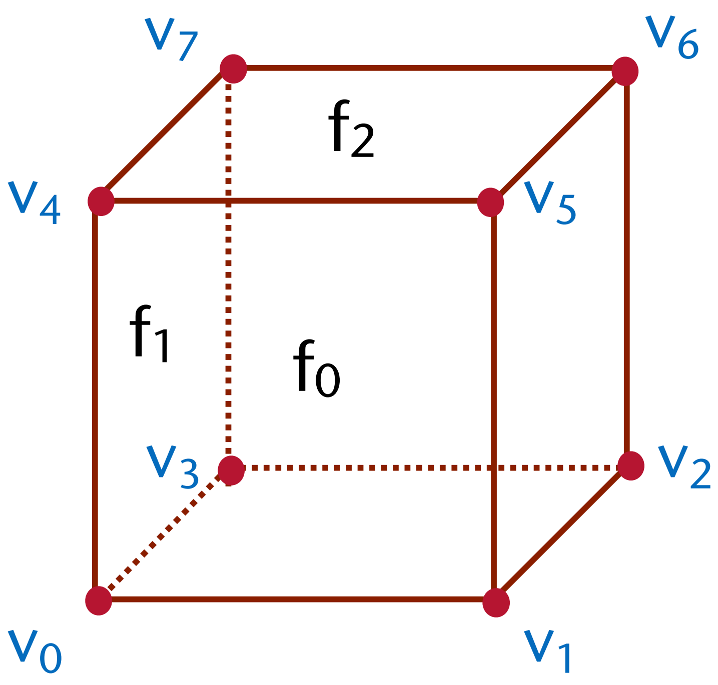
- Problems:
  - Vertices occur several times!
    - Waste of memory, problems with animations, ...
  - How to find all faces, incident to a given vertex?
  - Different array sizes for polygons with different numbers of

The Indexed Face Set

- Idea: use common "vertex pool" (*shared vertices*)
- Example:

vertices =
$x_0\ y_0\ z_0$
$x_1\ y_1\ z_1$
$x_2\ y_2\ z_2$
$x_3\ y_3\ z_3$
. . .

face	*vertex index*
0	0, 1, 5, 4
1	0, 3, 7, 4
2	4, 5, 6, 7
. . .	



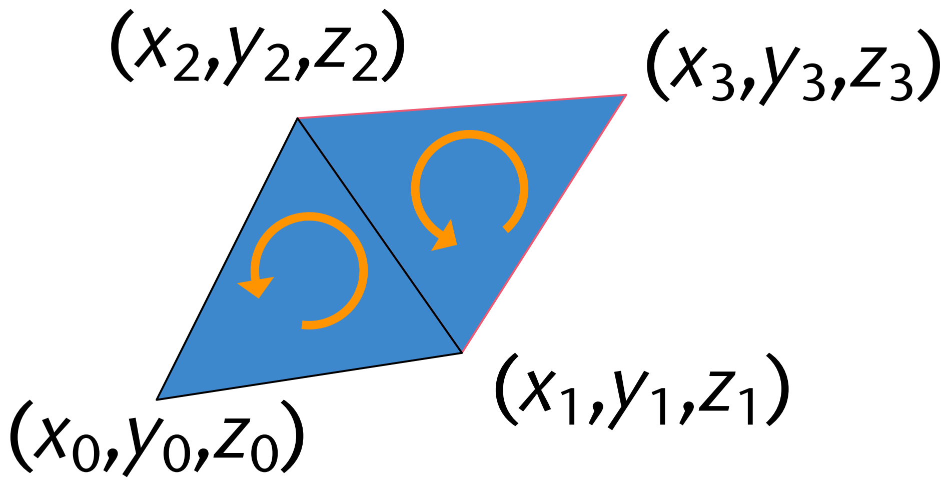
- Advantage: significant memory savings
  - 1 vertex = 1 point + 1 vector (v.-normal) + uv-texture coord. = 32 bytes
  - 1 index = 1 integer = 4 bytes
- Deformable objects / animations are much easier
- Probably the most common data structure

The OBJ File Format

- OBJ = indexed face set + further features
- Line based ASCII format

1. Ordered list of vertices:
   - Introduced by "v" on the line
   - Spatial coordinates x, y, z
   - Index is given by the order in the file
2. Unordered list of polygons:
   - A polygon is introduced by "f"
   - Then, ordered list of vertex indices
   - Length of list = # of edges
   - Orientation is given by order of vertices

- In principle, "v" and "f" can be mixed arbitrarily

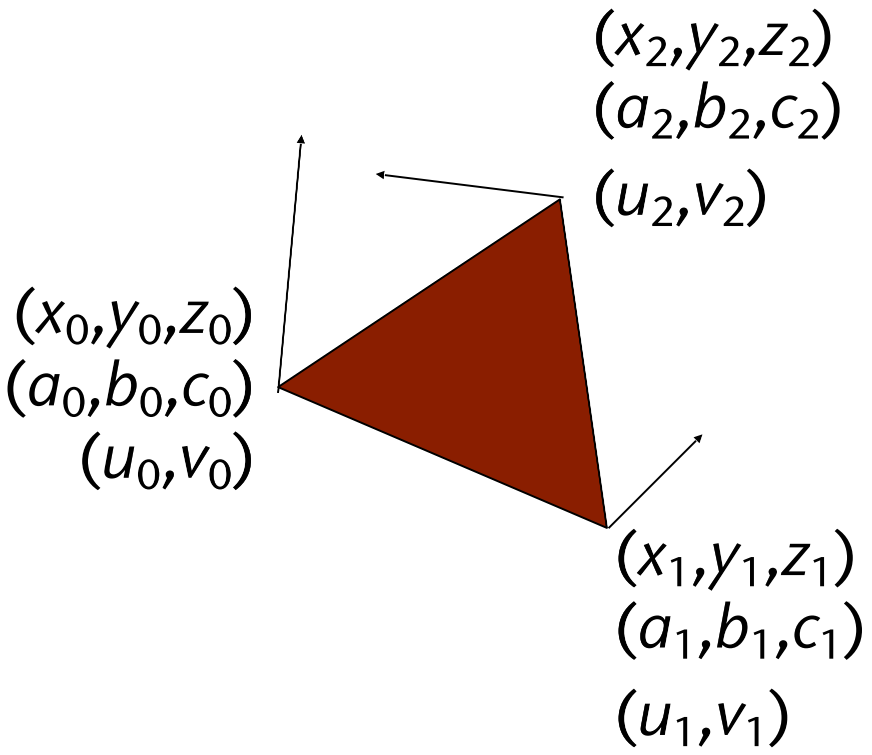


v x_0 y_0 z_0
v x_1 y_1 z_1
v x_2 y_2 z_2
v x_3 y_3 z_3

f 0 1 2
f 1 3 2

More Attributes

- Vertex normals:
  - prefix"vn"
  - contains x, y, z for the normals
  - not necessarily normalized
  - not necessarily in the same order as the vertices
  - indizes similar to vertex indices
- Texture coordinates:
  - prefix "vt"
  - not necessarily in the same order as the vertices
  - Contains u,v texture coordinates
- Polygons:
  - use "/" as delimiter for the indices
  - vertex / normal / texture
  - normal and texture are optional
  - use "//" to omit normals, if only tex coords are given

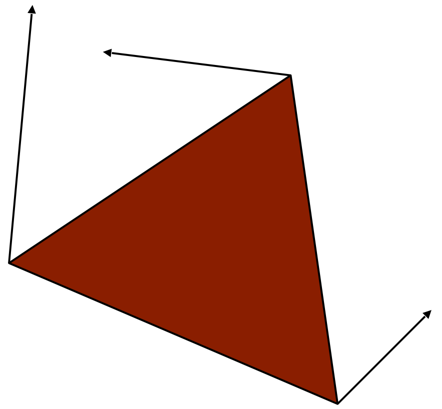


v x_0 y_0 z_0
v x_1 y_1 z_1
v x_2 y_2 z_2

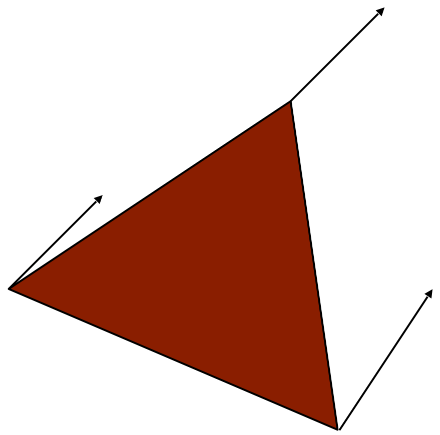
vn a_0 b_0 c_0
vn a_1 b_1 c_1
vn a_2 b_2 c_2

vt u_0 v_0
vt u_1 v_1
vt u_2 v_2

f 0/0/0 ...
f ...



f 0/0/0 1/1/1 2/2/2



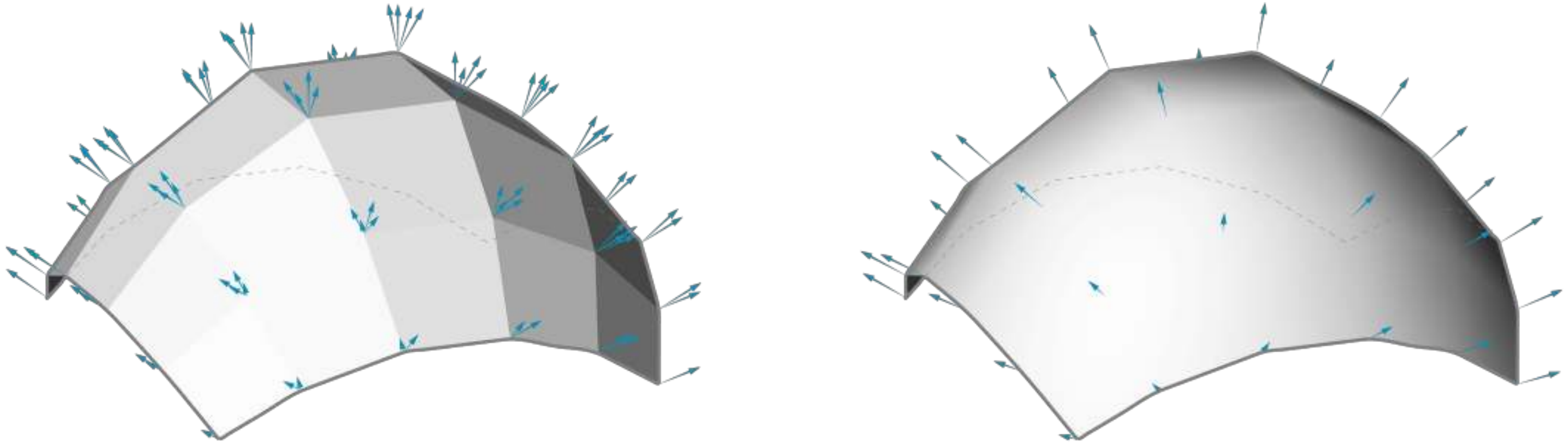
f 0/1/0 1/1/1 2/1/2

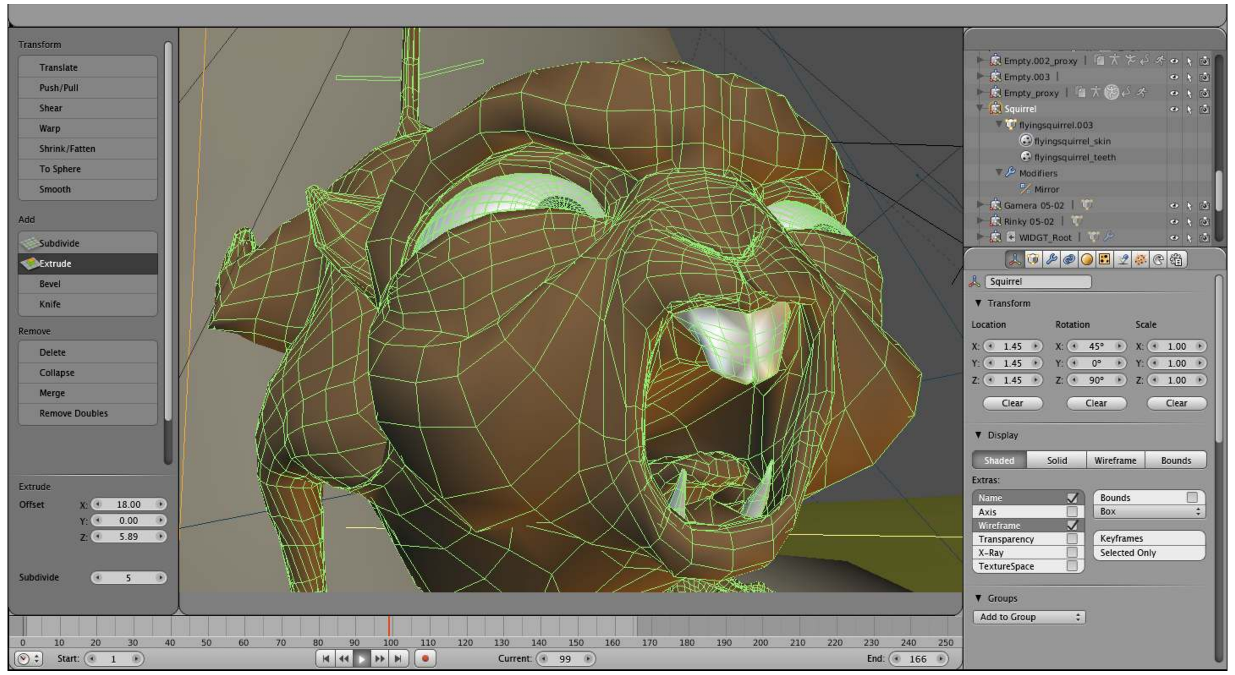
- Problems:
  - Edges are (implicitly) stored *twice*
  - Still no adjacency information (no "topology")

- Consequence:
  - Finding all facets incident to a given vertex takes time $O(n)$, where n = # vertices of the mesh
  - Dito for finding all vertices adjacent to another given vertex (the 1-ring)
  - A complete mesh traversal takes time $O(n^2)$
    - With a mesh traversal you can, for instance, test whether an object is closed
    - Can be depth-first or breadth-first

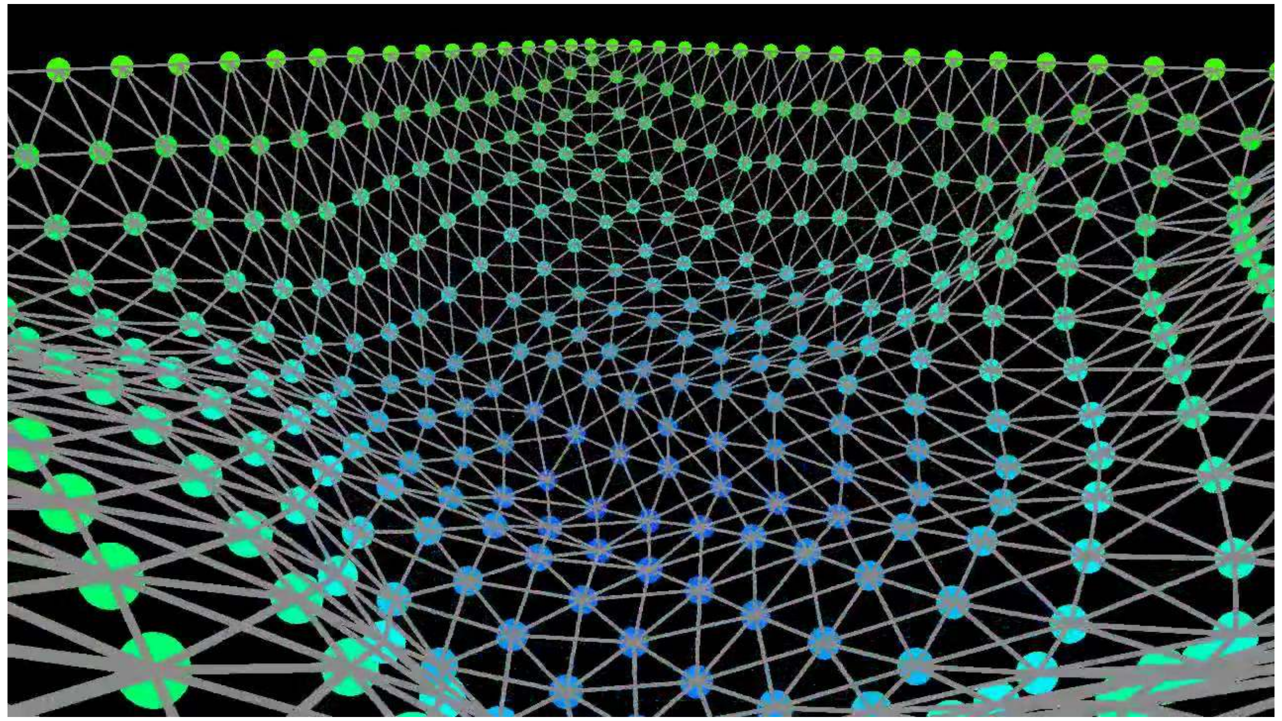
Examples Where Adjacency Information is Needed

- Computing vertex normals





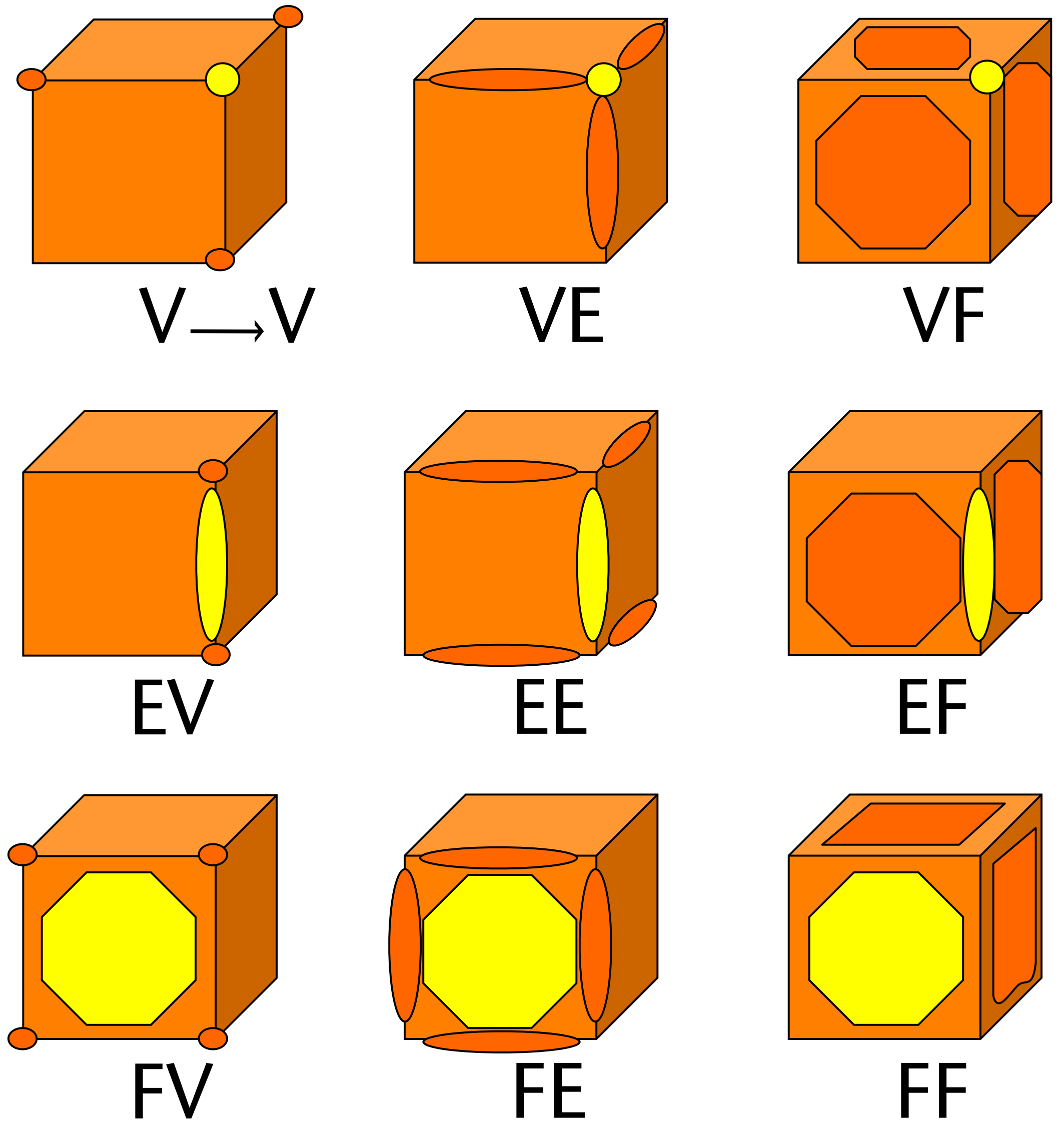
- Editing meshes



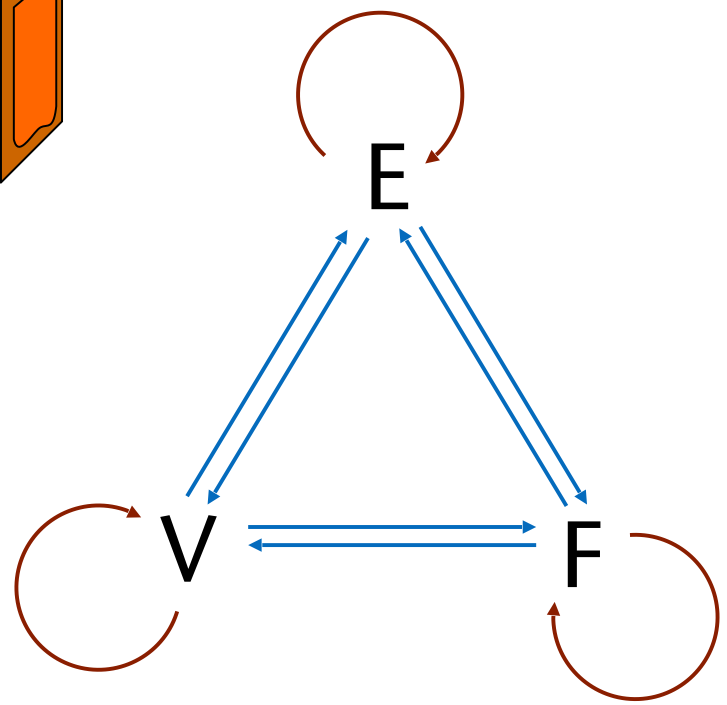
- Simulation, e.g., mass-spring systems

All Possible Connectivity Relationships

	Given	Looking for (neighboring)	Notation
1	Vertex	Vertices	V → V
2	Vertex	Edges	V → E
3	Vertex	Faces	V → F
4	Edge	Vertices	E → V
5	Edge	Edges	E → E
6	Edge	Faces	E → F
7	Face	Vertices	F → V
8	Face	Edges	F → E
9	Face	Faces	F → F



Abstract notation of a data structure with all connectivity relationships: arrows show the incidence/adjacency info in the DS

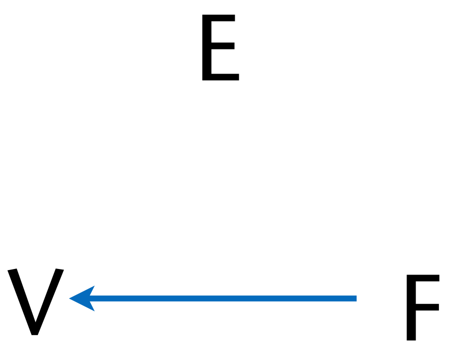


- Example: the Indexed Face Set

vertices
$x_0\ y_0\ z_0$
$x_1\ y_1\ z_1$
$x_2\ y_2\ z_2$
$x_3\ y_3\ z_3$
. . .

face	*vertex index*
0	0, 1, 5, 4
1	0, 3, 7, 4
2	4, 5, 6, 7
. . .	

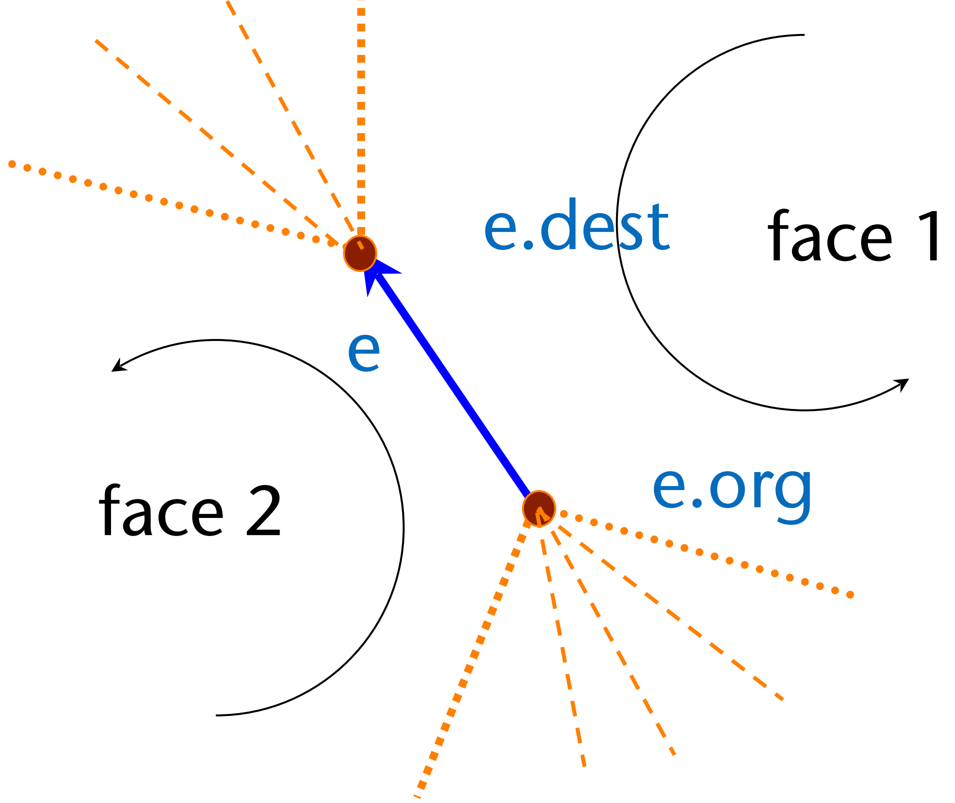
=



- Question: What is the minimal data structure, that can answer all neighboring queries in time O(1)?

The Winged-Edge Data Structure Just FYI

- Idea: edge-based data structure (in contrast to face-based)
- Observations:
  - An edge is defined by exactly two vertices: ***e.org* , *e.dest***
    → yields an orientation of the edge
  - In a closed polyhedron, each edge is incident to exactly 2 facets
  - If it is oriented, then one of these facets has the same orientation as the edge, the other one is opposite



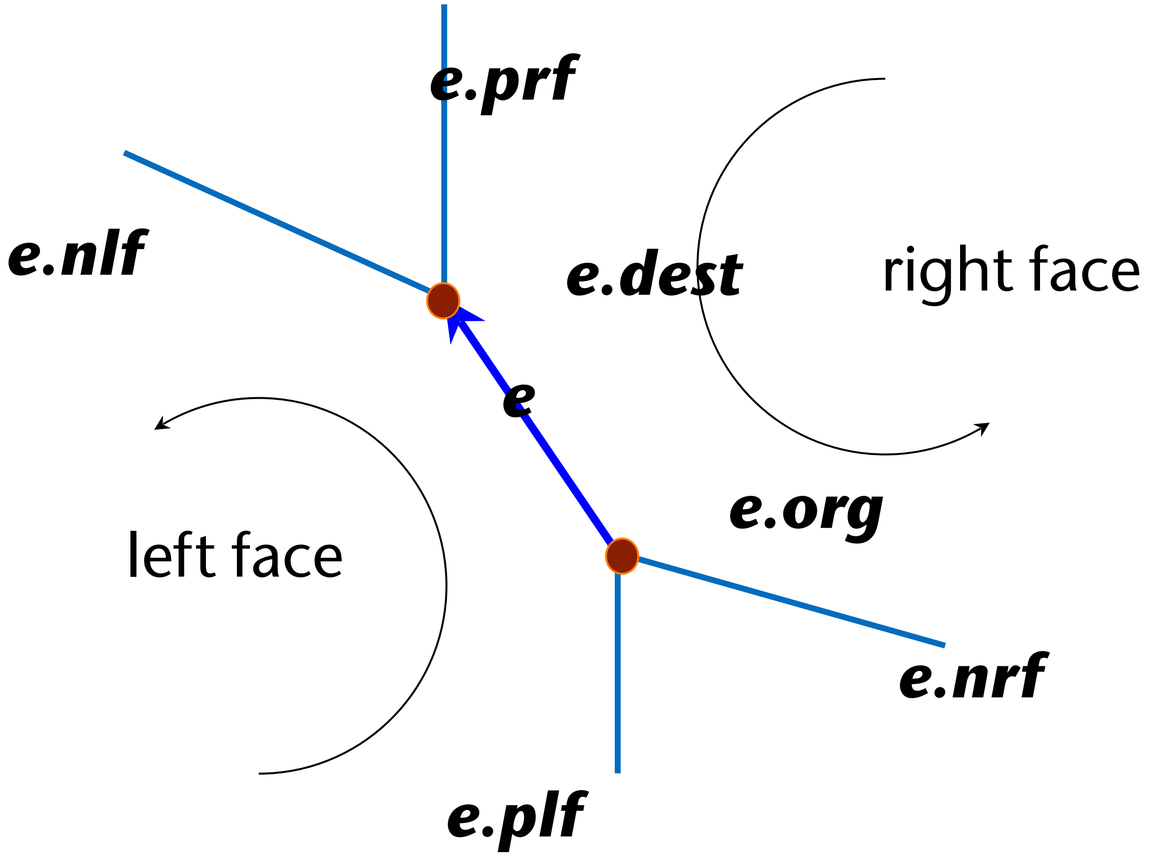
Optional

- Each edge has 4 pointers to 4 adjacent edges:

1. **e.prf** = edge incident to ***e.dest*** and incident to *right face*
   (prf = "previous right face")
2. **e.nrf** = edge incident to ***e.org*** and incident to *right face*
   ("next right face")

3./4. **e.nlf / e.plf** = edge adjacent to ***e*** and incident to *left face* ("next/ previous left face")

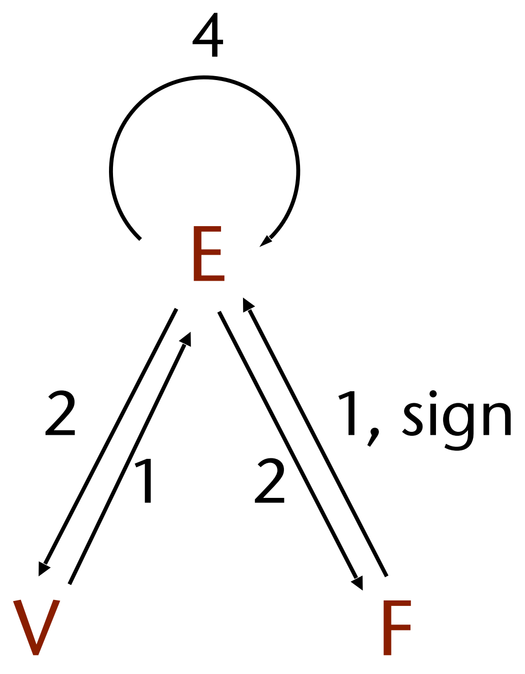
- Observation: if all facets are oriented consistently, then each edge occurs once from org→dest and once from dest→org



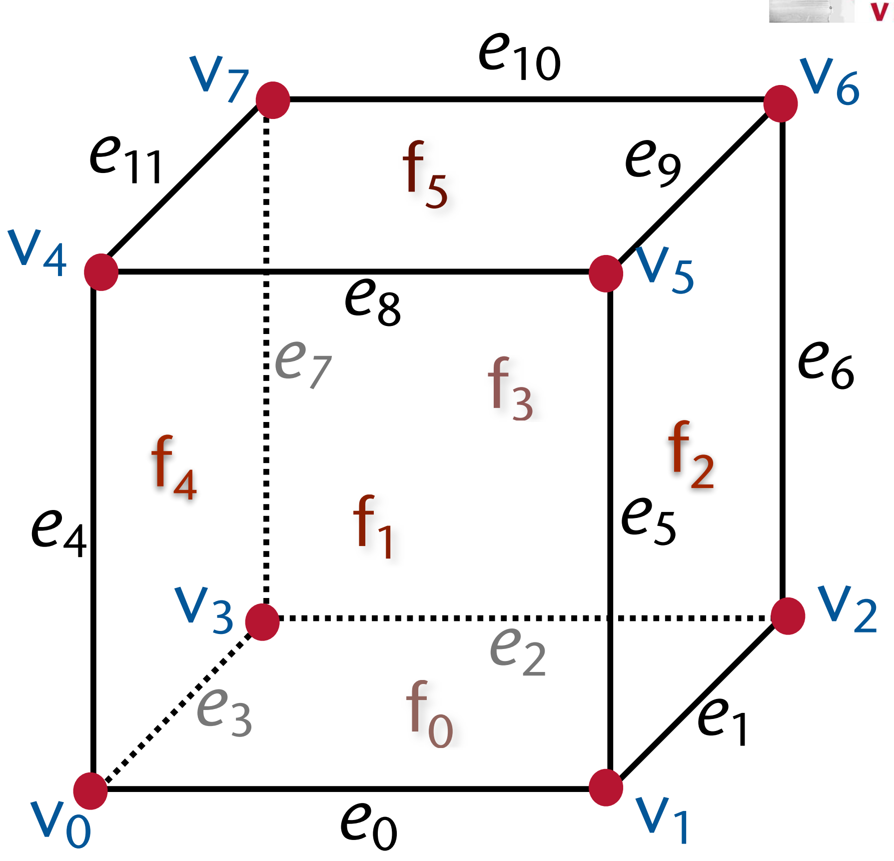
Optional

- In addition:
  - Each edge stores *one* pointer to the left and right facet (***e.lf, e.rf***)
  - Each facet & each vertex stores *one* pointer to an arbitrary edge incident to it

- Abstract representation of the data structure:



Example **Optional**



Edge Table

e	org	dest	ncw	nccw	pcw	pccw	lf	rf
0	v0	v1	e1	e5	e4	e3	f1	f0
1	v1	v2	e2	e6	e5	e0	f2	f0
2	v2	v3	e3	e7	e6	e1	f3	f0
3	v3	v0	e0	e4	e2	e7	f4	f0
4	v0	v4	e8	e11	e0	e3	f4	f1
5	v1	v5	e9	e8	e1	e0	f1	f2
6	v2	v6	e10	e9	e2	e1	f2	f3
7	v3	v7	e11	e10	e3	e2	f3	f4
8	v4	v5	e5	e9	e4	e11	f5	f1
9	v5	v6	e6	e10	e5	e8	f5	f2
10	v6	v7	e7	e11	e9	e6	f5	f3
11	v7	v4	e4	e8	e10	e7	f5	f4

Vertex Table

v	x	y	z	edge
0	0.0	0.0	0.0	0
1	1.0	0.0	0.0	1
2	1.0	1.0	0.0	2
3	0.0	1.0	0.0	3
4	0.0	0.0	1.0	8
5	1.0	0.0	1.0	9
6	1.0	1.0	1.0	10
7	0.0	1.0	1.0	11

Face Table

f	edge	orient
0	e0	-
1	e8	-
2	e5	-
3	e6	-
4	e11	-
5	e8	+

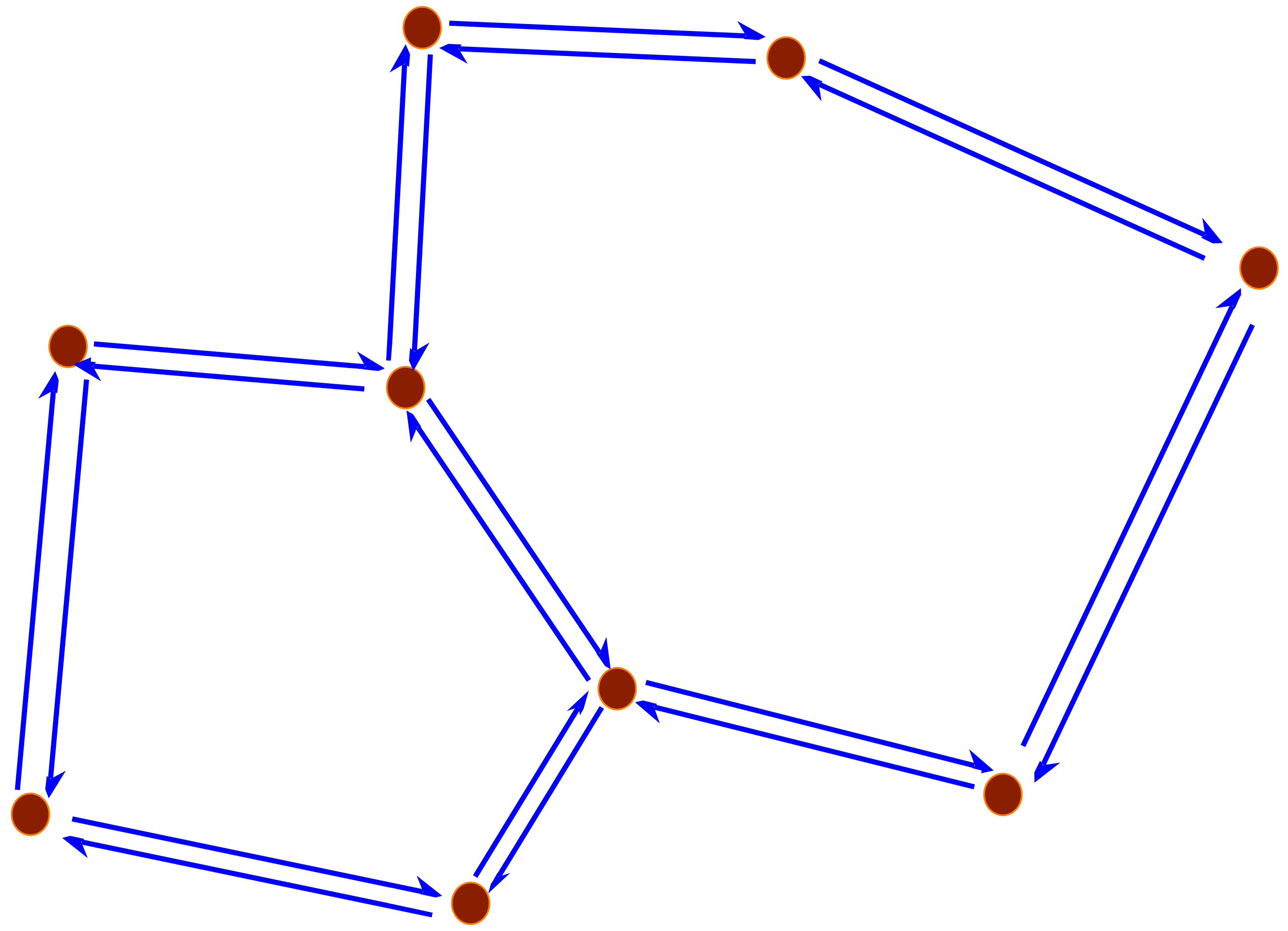
Optional

- All neighborhood/connectivity queries can be answered in time $O(k)$ where (k = size of the output)
  - 3 kinds of queries can be answered directly in $O(1)$,
    and 6 kinds of queries can be answered by a local traversal of the data structures around a facet or a vertex in $O(k)$
- Problem: When following edges, one has to test for each edge *how* it is oriented, in order to determine whether to follow ***n[c]cw*** or ***p[c]cw***!

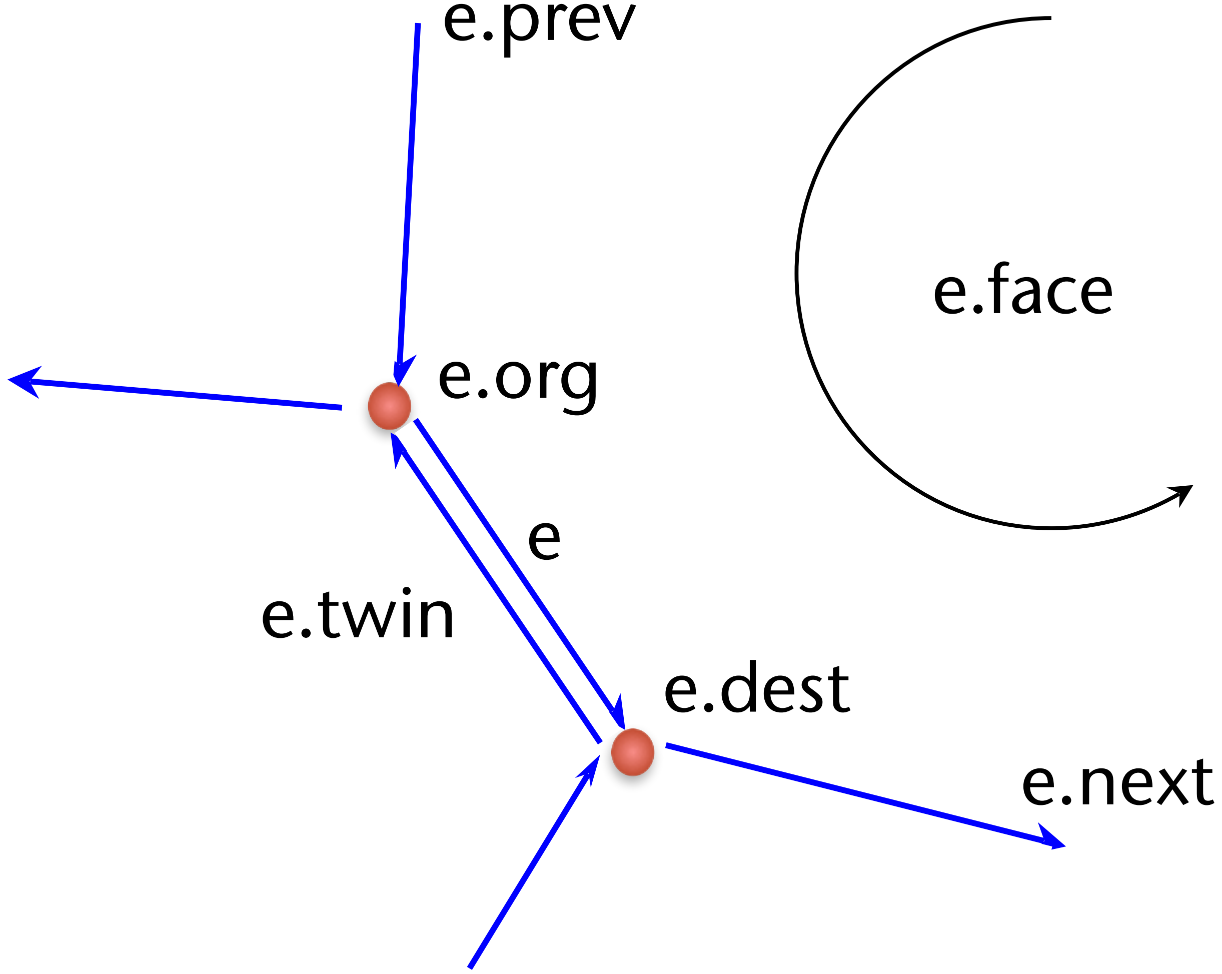
Doubly Connected Edge List

[Preparata & Müller, 1978]

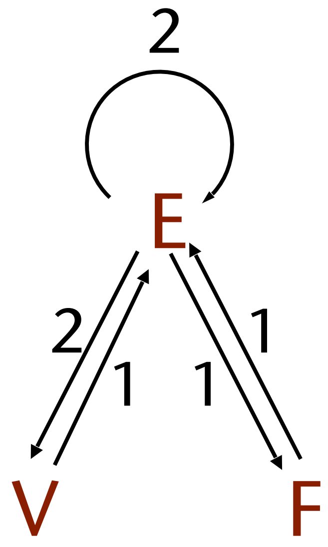
- In computer graphics also known as *"half-edge data structure"*
- Arguably the easiest and most efficient connectivity data structure
- Idea:
  - Edges are the first-class citizens, each edge is split into two half-edges
  - One half-edge represents only one direction and one "side" of the complete edge
  - Main data structure = table of half-edges



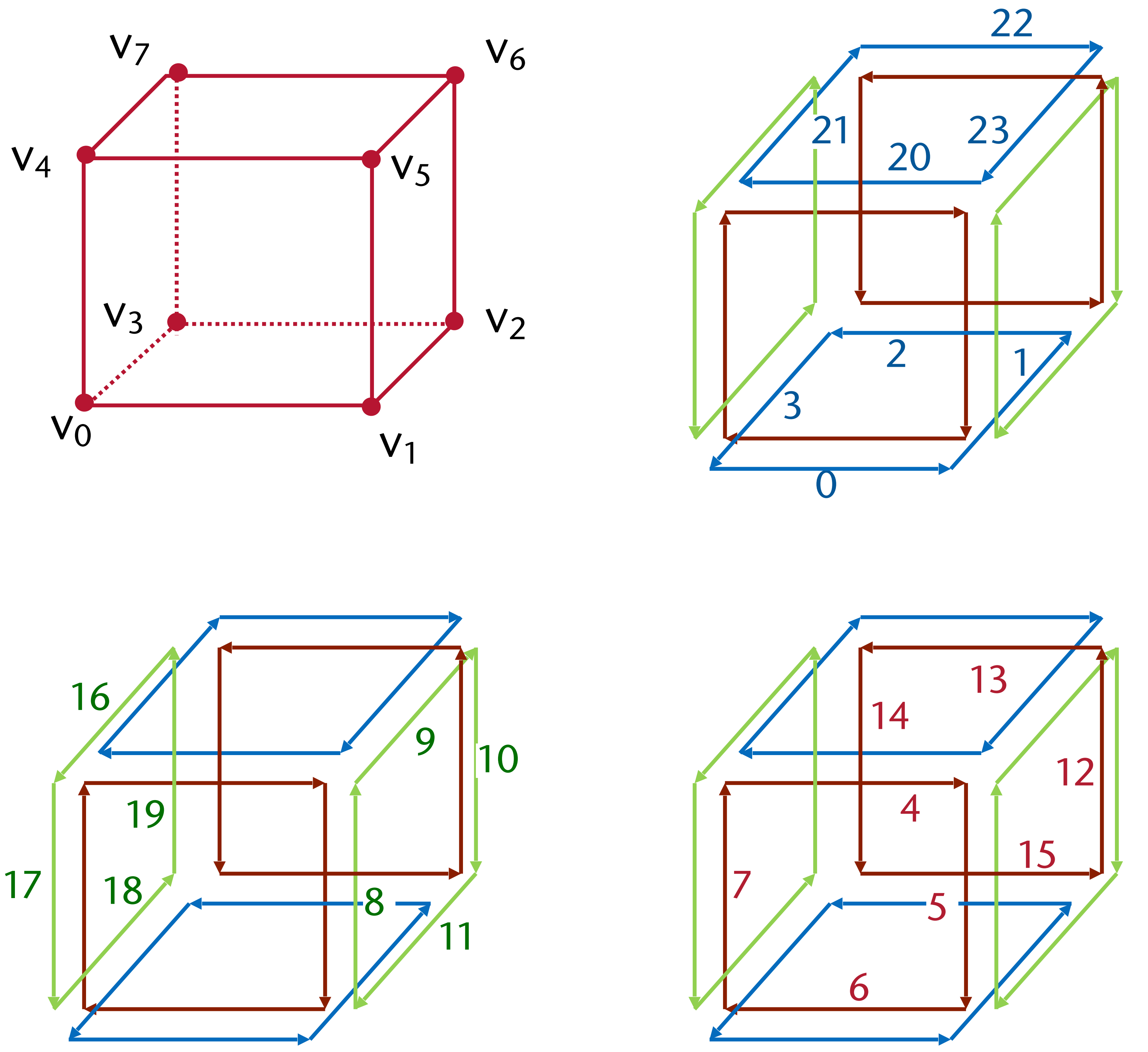
- The pointers stored with each half-edge:
    - Start (**org**) and end vertex (**dest**)
    - Incident **face** (to the left-hand side, when walking along the half-edge)
    - **Next** und **prev**ious edge (in traversal order)
    - **Twin** edge
    - (Originating vertex could be omitted, because e.org = e.twin.dest)



- Abstract notation:



Example (Here in CW Order!)

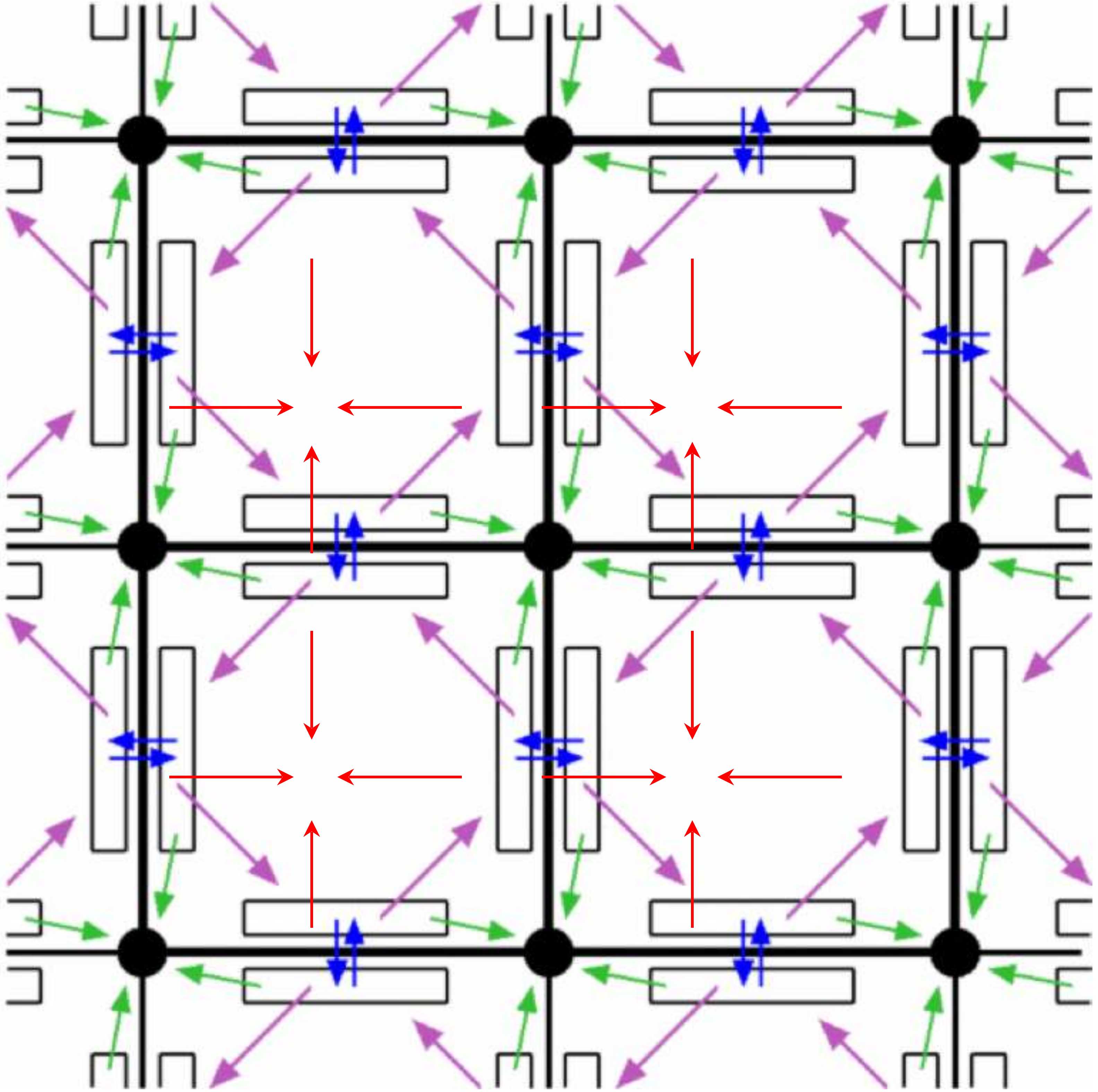


v	x	y	z	e
0	0.0	0.0	0.0	0
1	1.0	0.0	0.0	1
2	1.0	1.0	0.0	2
3	0.0	1.0	0.0	3
4	0.0	0.0	1.0	4
5	1.0	0.0	1.0	9
6	1.0	1.0	1.0	13
7	0.0	1.0	1.0	16

0	e20
1	e4
2	e0
3	e15
4	e16
5	e8

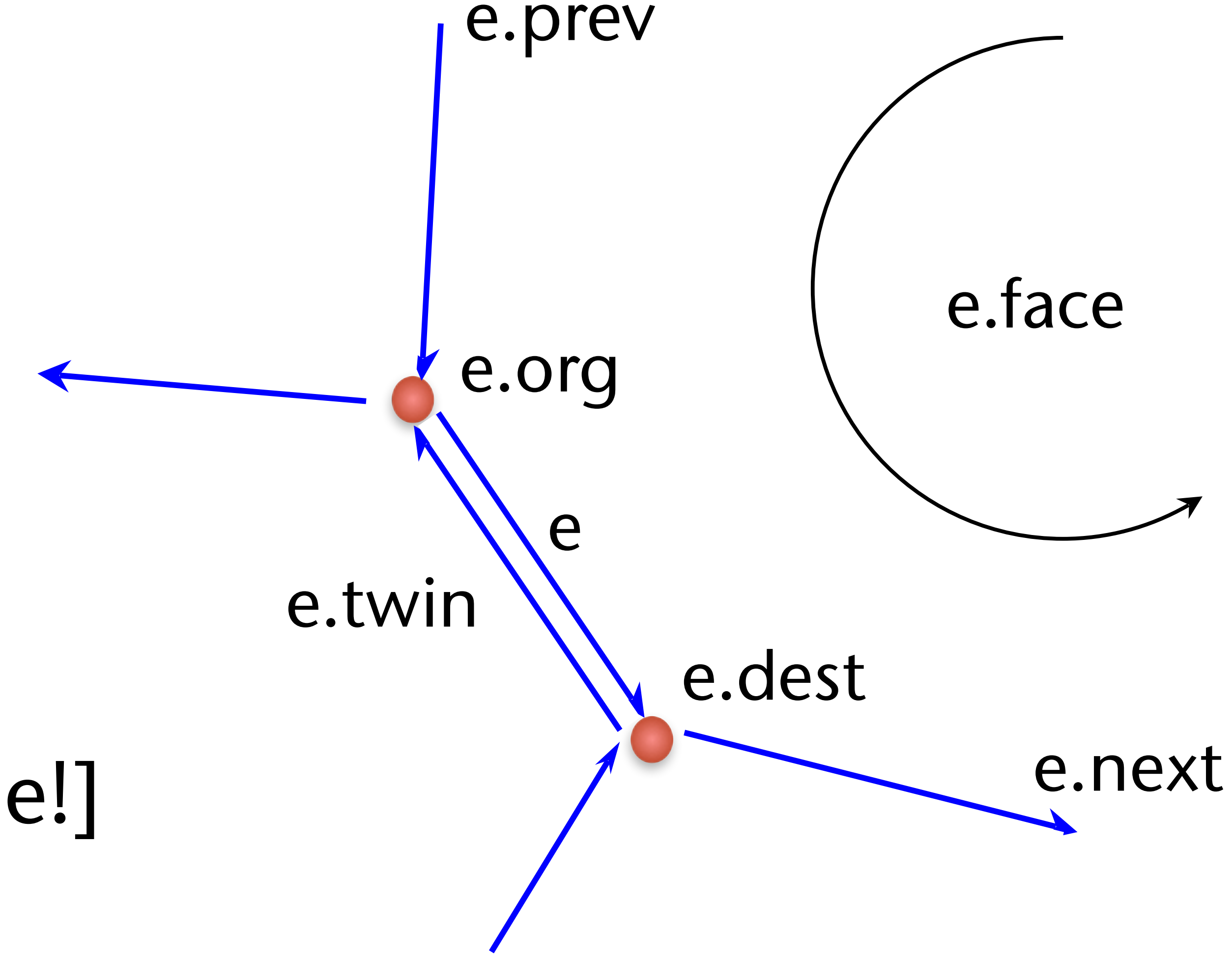
e	org	next	prv	twin
0	0	1	3	6
1	1	2	0	11
2	2	3	1	15
3	3	0	2	18
4	4	5	7	20
5	5	6	4	8
6	1	7	5	0
7	0	4	6	17
8	1	9	11	5
9	5	10	8	23
10	6	11	9	12
11	2	8	10	1
12	2	13	15	10
13	6	14	12	22
14	7	15	13	19
15	3	12	14	2
16	7	17	19	21
17	4	18	16	7
18	0	19	17	3
19	3	16	18	14
20	5	21	23	4
21	4	22	20	16
22	7	23	21	13
23	6	20	22	9

Visualization of the DCEL for a Quad Mesh



Invariants in a DCEL

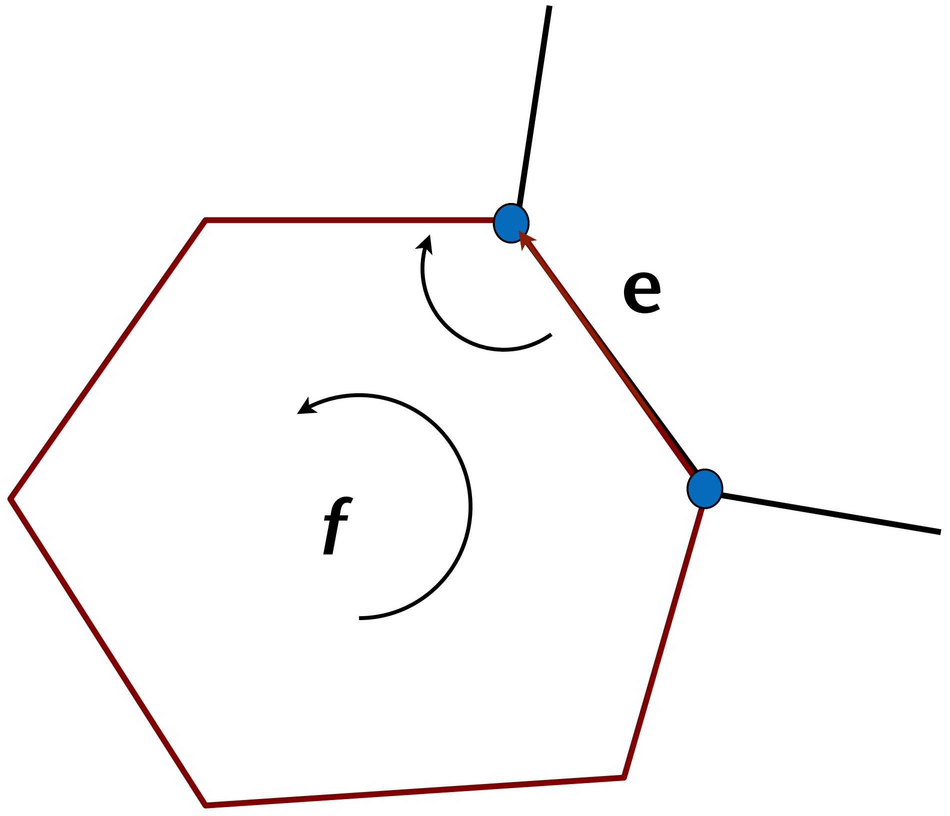
- Here, we will use the "functional notation", i.e., **`twin(e) = e.twin`**
- Invariants (= axioms in an *abstract data type* "DCEL"):
  - **`twin( twin(e) ) = e ,`**
    [if the mesh is closed]
  - **`org( next(e) ) = dest(e)`**
  - **`org(e) = dest( twin(e) )`**
    [if twin(e) is existing]
  - **`org( v.edge ) = v`**
    [provided **v.edge** always points to a leaving edge!]
  - etc. …



Face and Vertex Cycling

- Given: a closed, 2-manifold mesh
- Wanted: all vertices incident to a given face ***f***
- Algorithm:

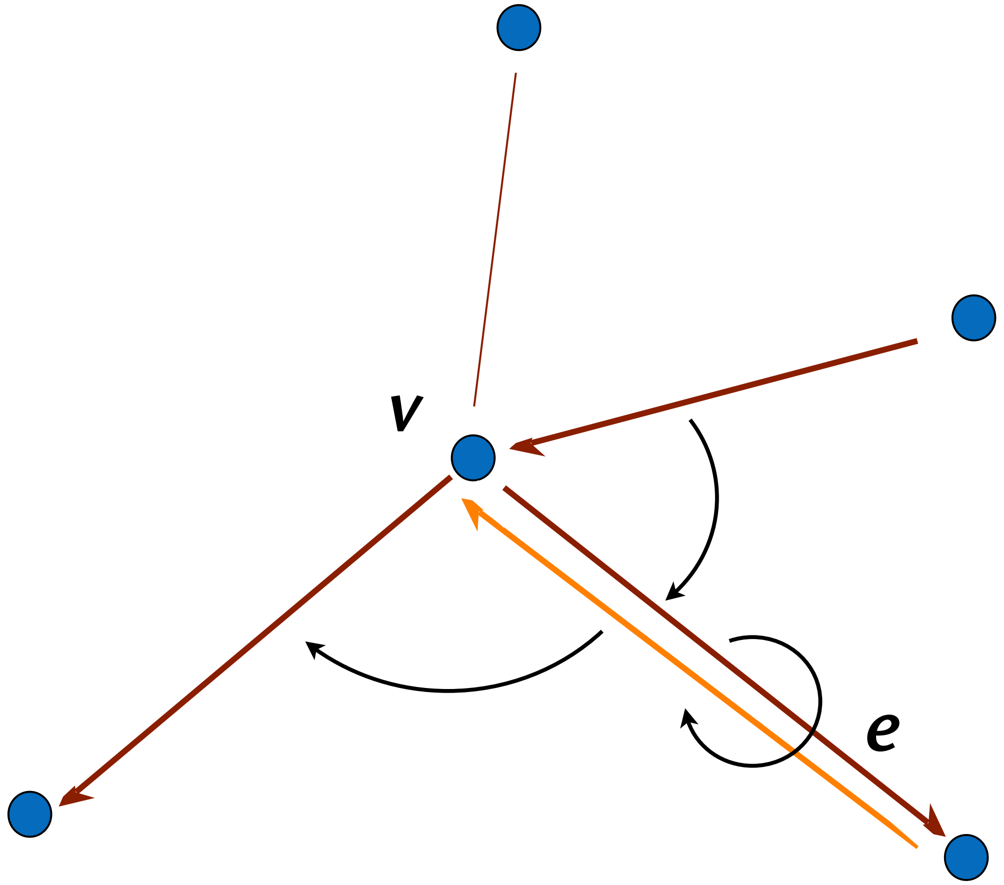
```
e_start = f.edge
e = e_start
repeat
  output e.dest
  e = e.next
until e == e_start
```



- Running time is in $O(k)$, with k = # vertices of ***f***

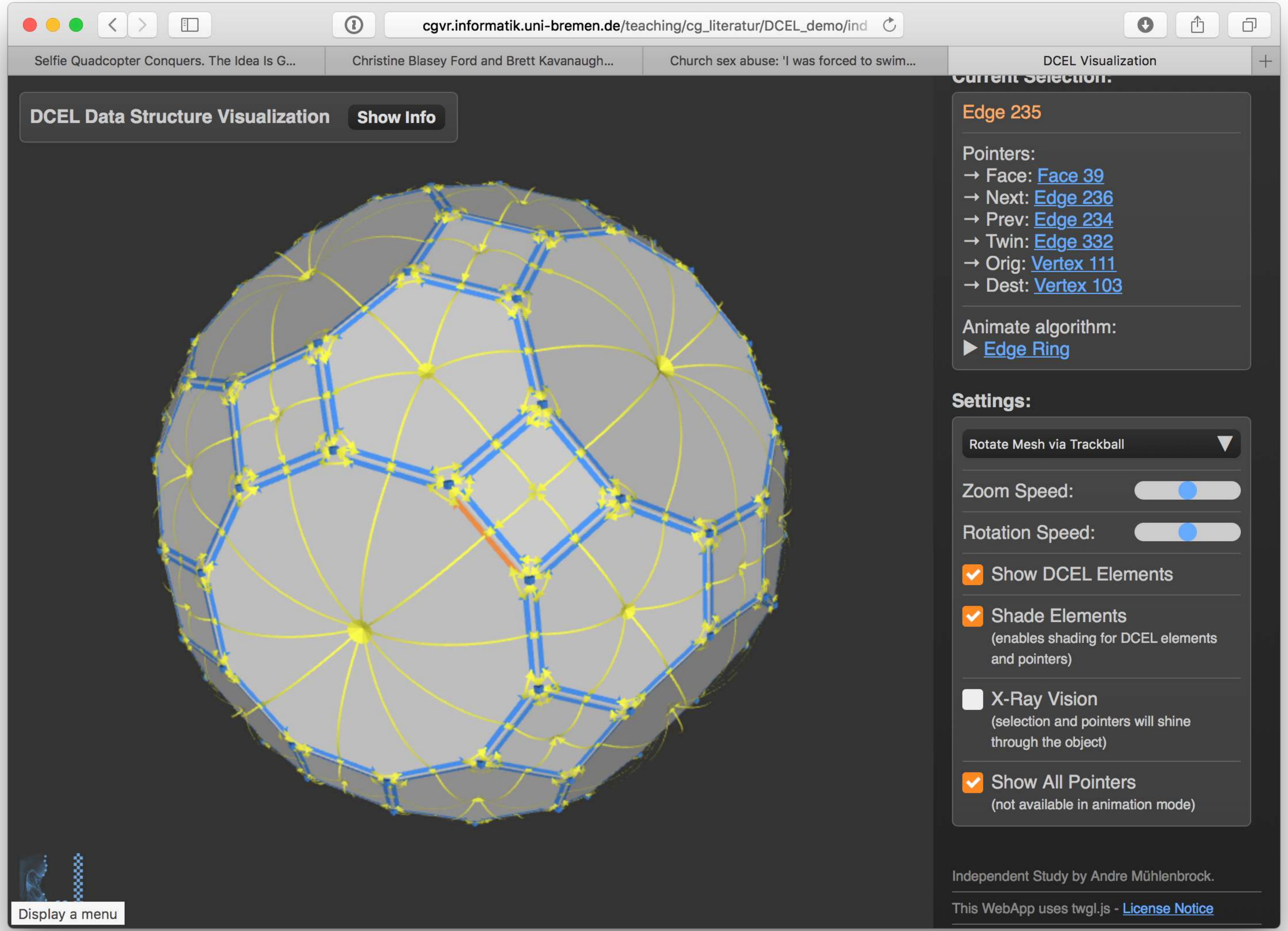
- Task: report all vertices *adjacent* to a given vertex ***v***
- Algorithm (wlog., **v.edge** points to a leaving edge):

```
e_start = v.edge
e = e_start
repeat
  output e.dest
  e = e.twin.next
until e == e_start
```



- Running time is in $O(k)$, where k = # neighbours of ***v***

Demo



DCEL Data Structure Visualization

Edge 235

Pointers:
→ Face: Face 39
→ Next: Edge 236
→ Prev: Edge 234
→ Twin: Edge 332
→ Orig: Vertex 111
→ Dest: Vertex 103

Animate algorithm:
▶ Edge Ring

Settings:

Rotate Mesh via Trackball

Zoom Speed:

Rotation Speed:

☑ Show DCEL Elements

☑ Shade Elements
(enables shading for DCEL elements and pointers)

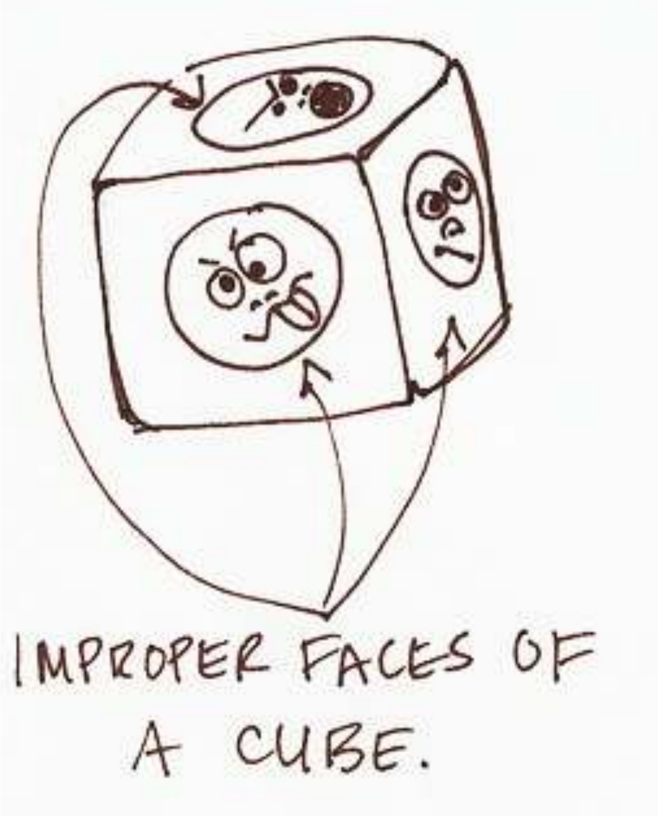
☐ X-Ray Vision
(selection and pointers will shine through the object)

☑ Show All Pointers
(not available in animation mode)

Independent Study by Andre Mühlenbrock.

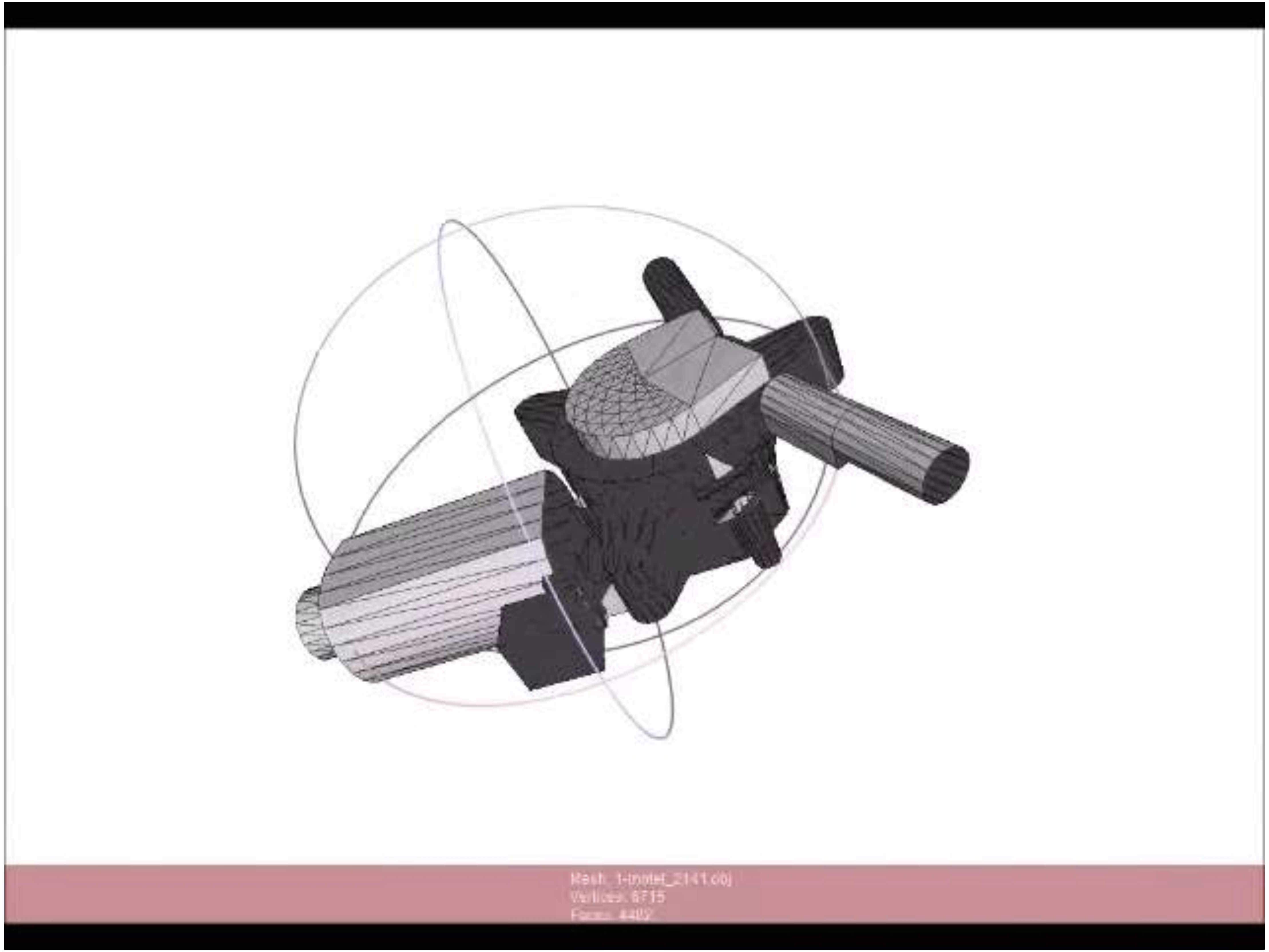
This WebApp uses twgl.js - License Notice

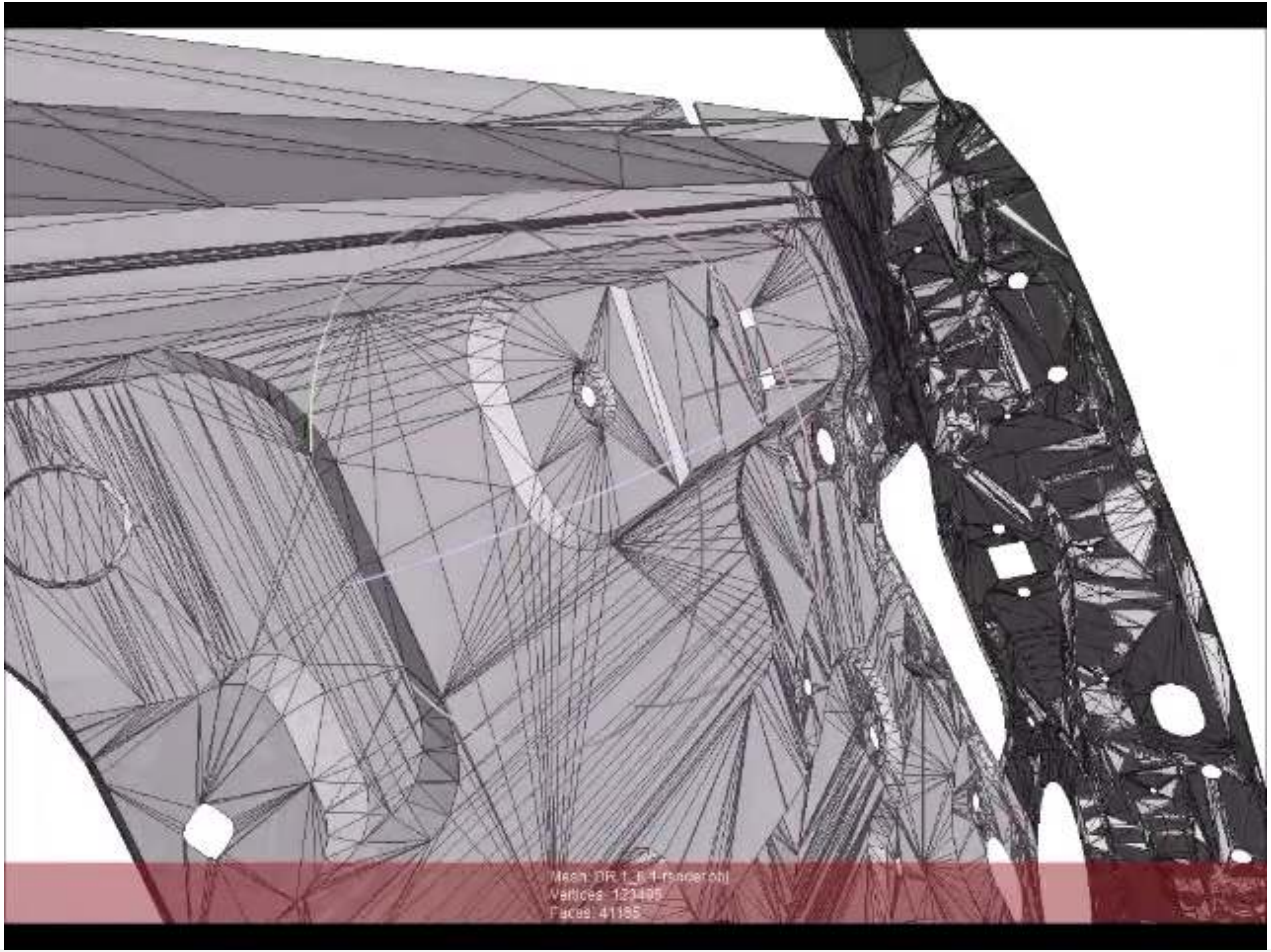
- Terminology: a feature = a vertex, or an edge, or a facet
- Theorem:
  A DCEL over a 2-manifold, oriented mesh supports all incidence and adjacency queries for a given feature in time $O(1)$ or $O(k)$, where k = # neighbors.
- Crucial property (I learned it the hard way):
  the DCEL must be consistent!
  → all faces must be properly oriented!
  → mesh must be orientable



Courtney Gibbons 2007

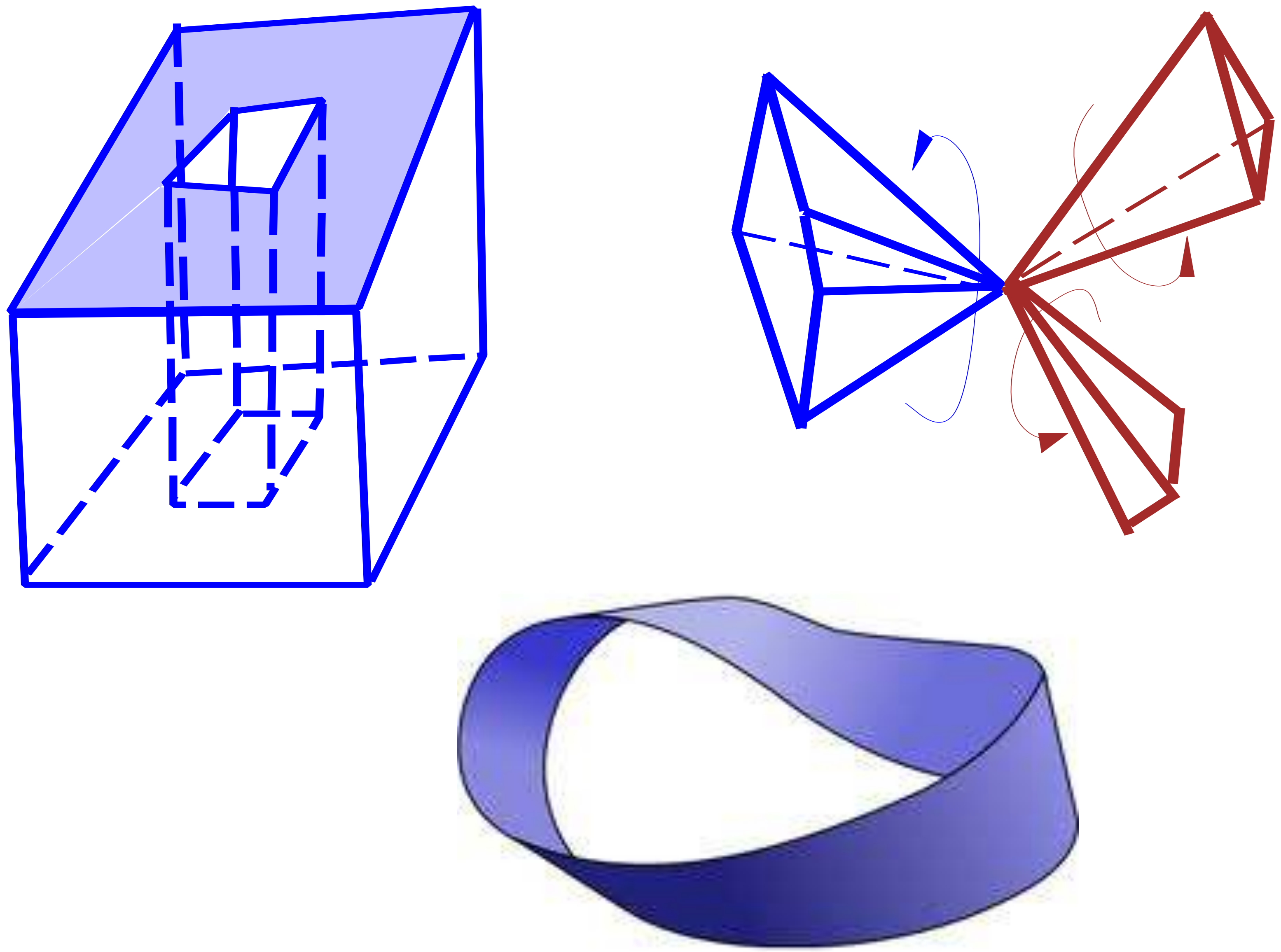
Real-World CAD Output




Faces: 41185

Limitations / Extensions of the DCEL

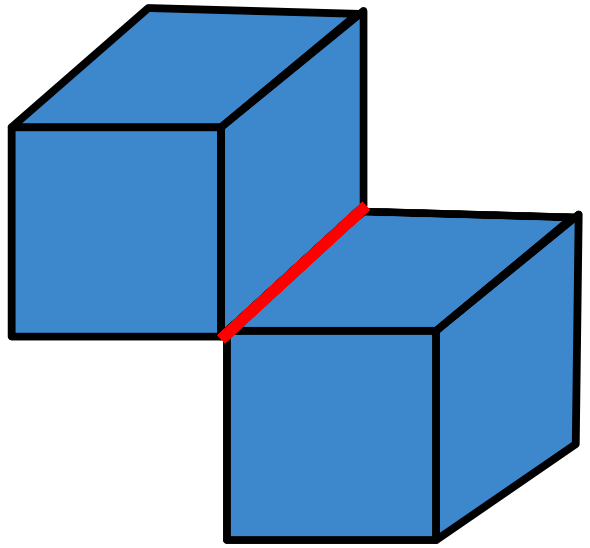
- A DCEL can store only meshes that are …
  1. two-manifold and
  2. oriented, and
  3. the polygons of which do not have "holes"!

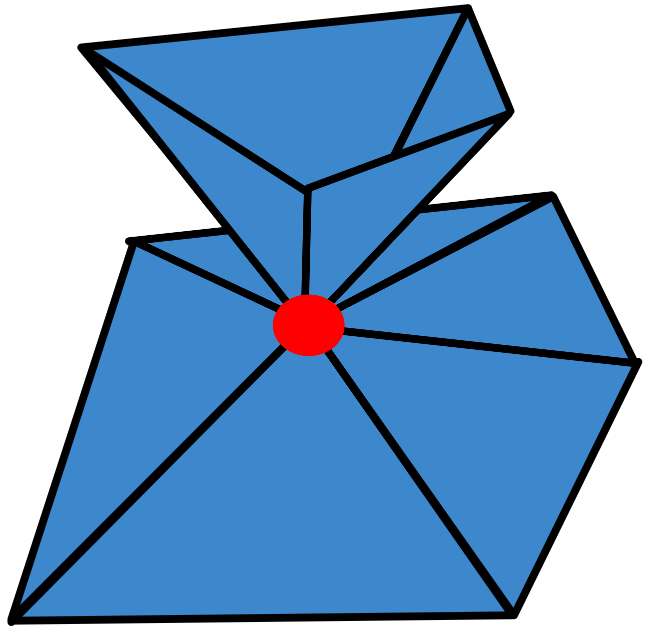


- Extensions: lots of them, e.g. those of Hervé Brönnimann
  - For non-2-manifold vertices, store several pointers to incident edges
  - Dito for facets with holes
  - Yields several cycles of edges for such vertices/faces

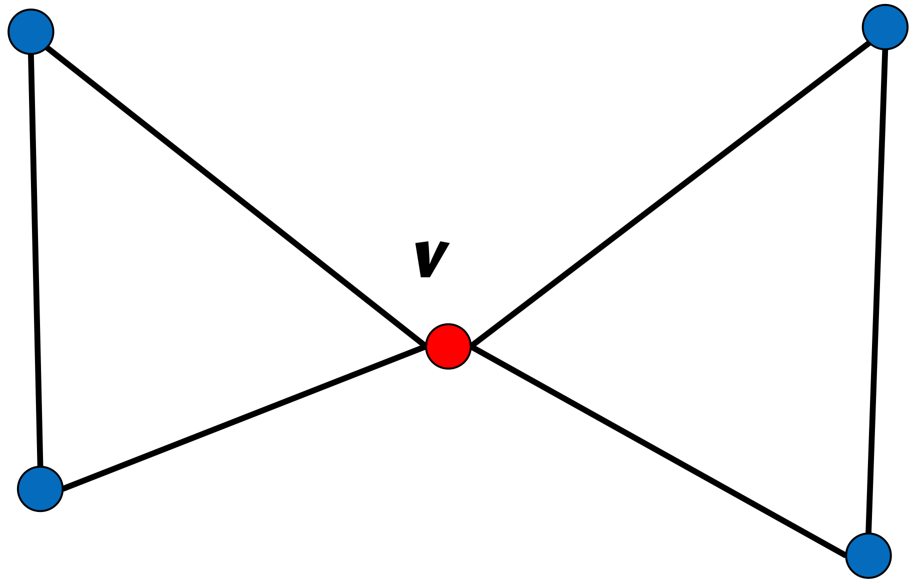
A DCEL Data Structure for Non-2-Manifolds

- Directed Edge DS: extension of half-edge DS for meshes that are not 2-manifold, ideally only at just a few "extraordinary" places
- Idea:
  - Store pointers to edges (e.next, e.prev, v.edge, f.edge) as integer indices into the edge array
  - Interpret negative indices as pointers into "overflow" arrays, e.g.,
    - a list of all edges emanating from a vertex; or
    - the connected component accessible from a vertex / edge





- Why does the conventional DCEL fail for the following example?



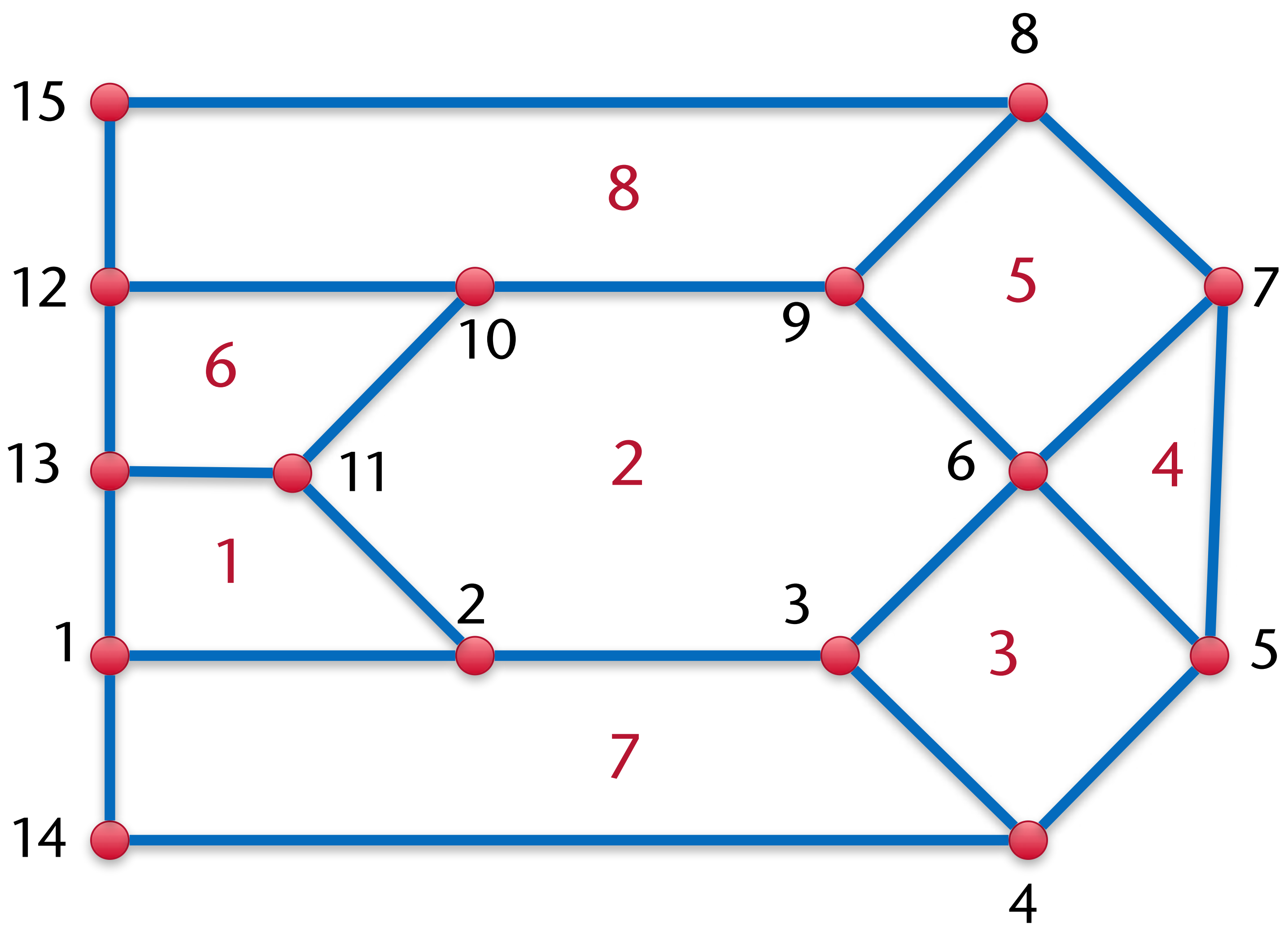
Combinatorial Maps

- Remark: *winged-edge* and *DCEL* data structures are (simple) examples of so-called combinatorial maps
- Other combinatorial maps are:
  - Quad-edge data structure (and *augmented quad-edge*)
  - Many extensions of DCEL
  - Cell-chains, *n*-Gmaps
    (like DCELs that can be extended to *n*-dimensional space)
  - Many more …

Matrix Representation of Meshes

[2017]

- Starting point: indexed face set



Face	Vertex Index					
1	1	2	11	13		
2	2	3	6	9	10	11
3	4	5	6	3		
4	5	7	6			
5	6	7	8	9		
6	11	10	12	13		
7	1	14	4	3	2	
8	9	8	15	12	10	
. . .	. . .					

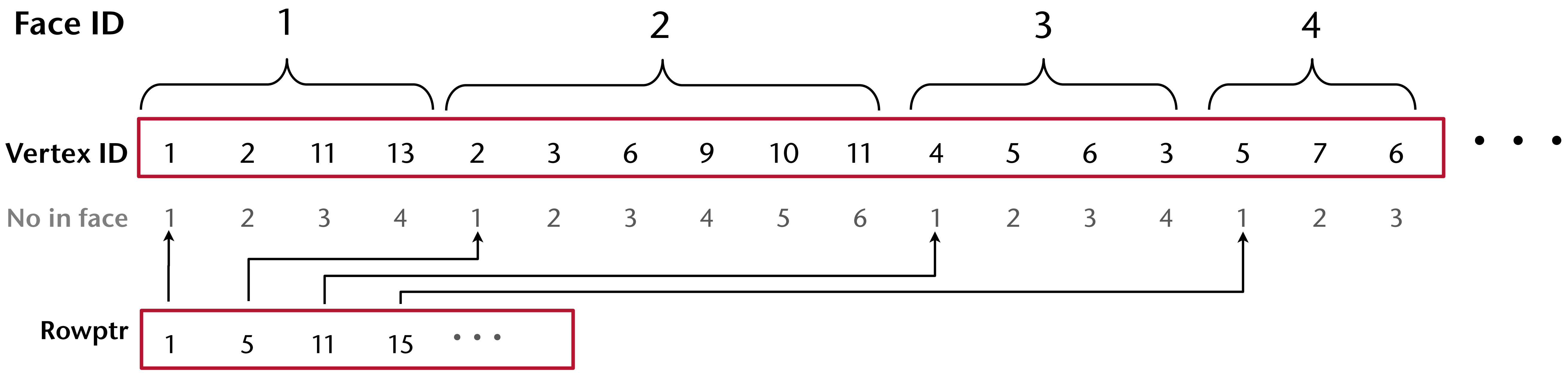
Note: always list vertices in CCW order in each row (= face)!

- Write the face table as a single, linear array (actually, 3 arrays):

Only Vertex-ID and Colptr need to be stored

- This matrix representation is called CSR (Compressed Sparse Row)

Face	Vertex Index					
1	1	2	11	13		
2	2	3	6	9	10	11
3	4	5	6	3		
4	5	7	6			
5	6	7	8	9		
6	11	10	12	13		
7	1	14	4	3	2	
8	9	8	15	12	10	



Face ID	1				2						3				4		
Vertex ID	1	2	11	13	2	3	6	9	10	11	4	5	6	3	5	7	6 • • •
No in face	1	2	3	4	1	2	3	4	5	6	1	2	3	4	1	2	3
Rowptr	1	5	11	15	• • •												

The Expanded, Sparse Mesh Matrix *M*

- The indexed face set set can also be represented by a mesh matrix *M*
  - Vertex ID = row index, Face ID = column index, no. within face = value $M(i,j)$
- Incidentally, the same CSR representation is now a Compact Sparse Column (CSC) representation of the mesh matrix!

Face	Vertex Index				
1	1	2	11	13	
2	2	3	6	9	10 11
3	4	5	6	3	
4	5	7	6		
5	6	7	8	9	
6	11	10	12	13	
7	1	14	4	3	2
8	9	8	15	12	10

Face ID

Vertex ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	1							
2	2	1						
3		2	4					
4			1					
5			2	1				
6		3	3	3	1			
7				2	2			
8					3			
9		4			4			
10		5						
11	3	6						
12								
13								
14	4							
15								

Face ID	1				2						3				4		
Vertex ID	1	2	11	13	2	3	6	9	10	11	4	5	6	3	5	7	6
No in face	1	2	3	4	1	2	3	4	5	6	1	2	3	4	1	2	3
Colptr	1	5	11	15	. . .												

Equivalence of Cyclic Permutations in Mesh Matrix

- All cyclic permutations of the values ("No in face") within one "Face ID"-block in the CSC describe the same mesh with the same topology!
- So: all cyclic permutations (rotations) of the values within one column (i.e., face) in the mesh matrix (for the non-zero rows) represent exactly the same mesh!

Face ID

Vertex ID

	1	2	3	4	5	6	7	8
1	1							
2	2	1						
3		2	1					
4			2					
5			3	1	1			
6		3	4	3	2			
7				2	3			
8					4			
9		4						
10		5						
11	3	6						
12								
13	4							
14								
15								

	1	2	3	4	5	6	7	8
1	4							
2	1	6						
3		1	4					
4			1					
5			2	2	4			
6		2	3	1	1			
7				3	2			
8					3			
9		3						
10		4						
11	2	5						
12								
13	3							
14								
15								

Face ID	1				2						3				4		
Vertex ID	1	2	11	13	2	3	6	9	10	11	4	5	6	3	5	7	6
No in face	1	2	3	4	1	2	3	4	5	6	1	2	3	4	1	2	3

The defining invariant of a mesh matrix *M*

- Let $\mathbf{f}_i = (v_{i_1}, v_{i_2}, \ldots, v_{i_l})$, where v_{i_j} = face indices of face $\mathbf{f}_i$
- Then, for all non-zero entries in *M:*

$$M(\,\mathbf{f}_i(j), i\,) = j$$

	1	2	3	4	5	6	7	8
1	1							
2	2	1						
3		2	1					
4			2					
5			3	1	1			
6		3	4	3	2			
7				2	3			
8					4			
9		4						
10		5						
11	3	6						
12								
13	4							
14								
15								

Face ID	1				2						3				4		
Vertex ID	1	2	11	13	2	3	6	9	10	11	4	5	6	3	5	7	6
No in face	1	2	3	4	1	2	3	4	5	6	1	2	3	4	1	2	3

The Face-Vertex Incidence Matrix $\bar{M}$

- Replace each non-zero entry in *M* by 1

Mesh matrix *M*

	1	2	3	4	5	6	7	8
1	1							
2	2	1						
3		2	1					
4			2					
5			3	1	1			
6		3	4	3	2			
7				2	3			
8					4			
9		4						
10		5						
11	3	6						
12								
13	4							
14								
15								

Face-vertex incidence matrix $\bar{M}$

	1	2	3	4	5	6	7	8
1	1							
2	1	1						
3		1	1					
4			1					
5			1	1	1			
6		1	1	1	1			
7				1	1			
8					1			
9		1						
10		1						
11	1	1						
12								
13	1							
14								
15								

Examples for Mesh Processing Operations Using Matrices

- Given: triangle mesh
- Task: calculate the barycenter for each triangle
- Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be the vertices of the mesh, $\mathbf{v}_i = (x_i, y_i, z_i)$
- "Vector" representation of the mesh's vertices:

$$P = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \ldots & & \\ x_n & y_n & z_n \end{pmatrix}$$

- The incidence matrix allows to compute the vector of barycenters per triangle:

$$B = \frac{1}{3} \bar{M}^{\mathsf{T}} P$$

- Definition: matrix vector multiplication *with replacements*

$$M \odot P \,, \quad \text{with } (a, b, c) \rightarrow (d, e, f)$$

meaning: "perform matrix-vector multiplication, but replace values *a, b, c of M* by *d, e, f*, respectively, before the actual multiplication" (note: uses *M*, not $\bar{M}$!)

- Example: calculate vector of barycenters

$$B = M^{\mathsf{T}} \odot P \,, \quad \text{with } (1, 2, 3) \rightarrow (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$$

- Note: this can be implemented directly in the matrix-vector multiplication code

```
for w = 0, 1, 2:             // x,y,z coords
  for j = 0..m-1:            // m = #faces
    s = 0
    for k = 0..n-1:          // with j-th column
      mkj = m[k][j]
      if ( mkj == 1 )  mkj = 0.3
      if ( mkj == 2 )  mkj = 0.3
      if ( mkj == 3 )  mkj = 0.3
      s += mkj * p[k][w]
    b[j][w] = s
```

Example: Calculate Vertex Normals (Assuming Triangle Mesh)

- Calculate the edge vectors $A, B \in \mathbb{R}^{n\times 3}$ for each triangle:

$$A = M^{\mathsf{T}} \odot P \; , \quad \text{with } (1,2,3) \rightarrow (-1,1,0)$$

$$B = M^{\mathsf{T}} \odot P \; , \quad \text{with } (1,2,3) \rightarrow (-1,0,1)$$

- Calculate "row-wise vector product" between A and B:

$$N_f = A \otimes B$$

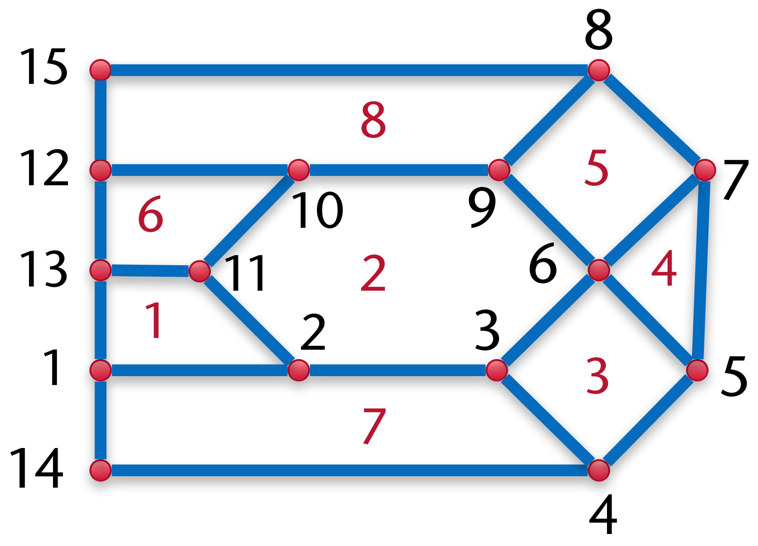
  where N_f = all face normals = "vector" with n rows and 3 columns (x,y,z)

- Calculate un-normalized vertex normals:

$$N_v = M \odot N_f \; , \quad \text{with } (1,2,3) \rightarrow (1,1,1)$$

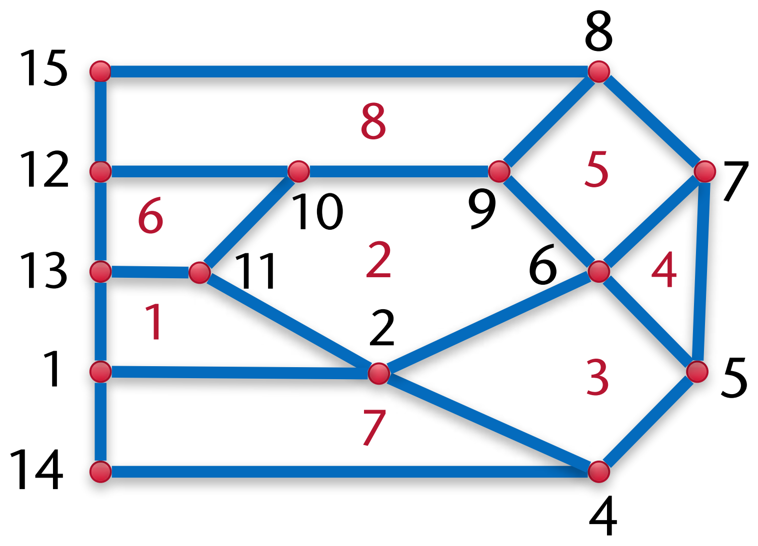
Example: Mesh Simplification

- For instance, collapse edge (2,3)
- Topology change: remove vertex 3, connect all edges incident to 3 with vertex 2 – except edge (2,3), of course
- Change face-vertex incidence matrix:
  - Add all incidences from row 3 to row 2, then clear row 3



	1	2	3	4	5	6	7	8
1	1							
2	1	1						
3		1	1					
4			1					
5			1	1	1			
6		1	1	1	1			
7				1	1			
8					1			
9		1						
10		1						
11	1	1						
12								
13	1							
14								
15								

	1	2	3	4	5	6	7	8
1	1							
2	1	2	1					
3								
4			1					
5			1	1	1			
6		1	1	1	1			
7				1	1			
8					1			
9		1						
10		1						
11	1	1						
12								
13	1							
14								
15								



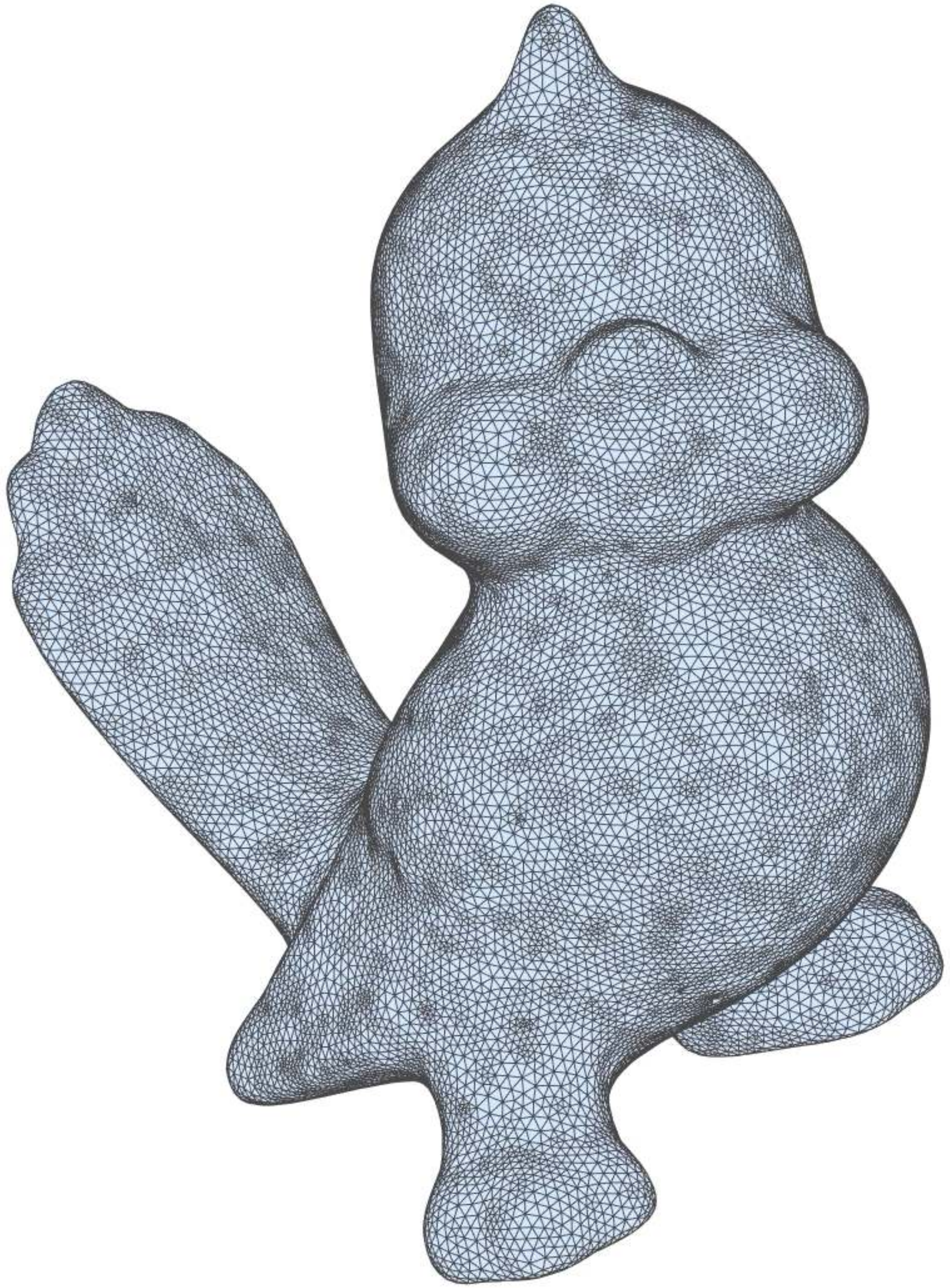
- Edge collapse (*i*,*j*) (where vertex *j* is removed) can be achieved by

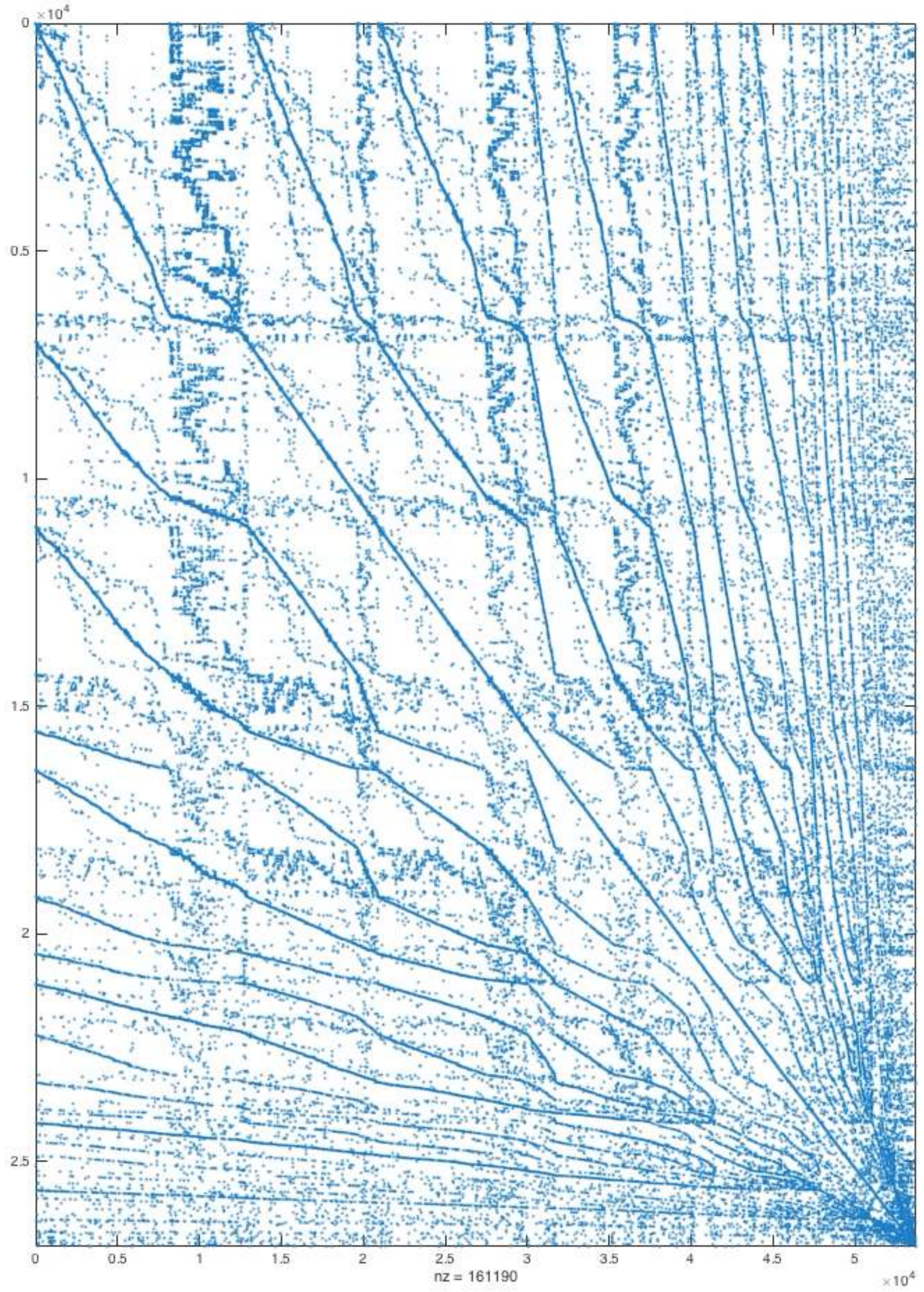
$$\bar{M}' = K \bar{M}$$

where

$$K = \begin{array}{cc} \begin{pmatrix} 1 & & & & & \\ & \ddots & & 1 & & \\ & & 1 & & & \\ & & & 0 & & \\ & & & & 1 & \\ & & & & & \ddots \end{pmatrix} & \begin{array}{l} \\ \longleftarrow i \\ \\ \longleftarrow j \\ \\ \\ \end{array} \\ \begin{array}{cccccc} & & & \uparrow & & \\ & & & j & & \end{array} & \end{array}$$

Mesh simplification demo using mesh matrices





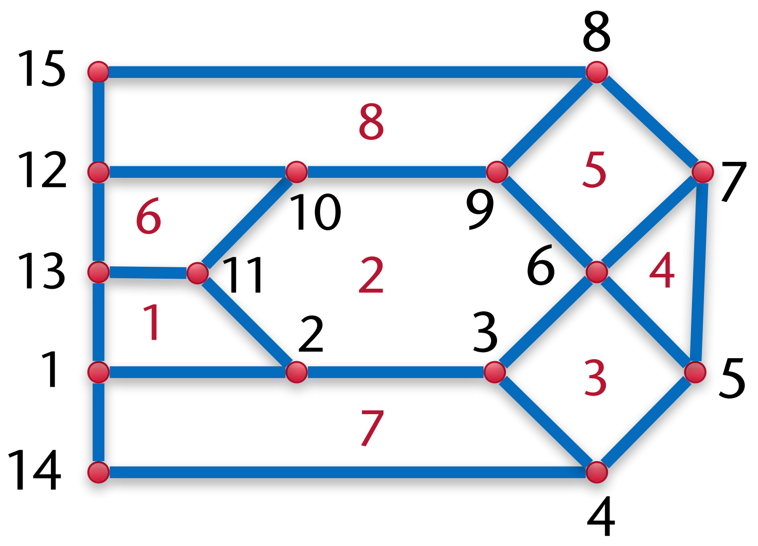
[Rhaleb Zayer, 2017]

Operations using Matrix-Matrix Multiplication

- Definition: generalized vertex-vertex adjacency

$$S_v = \bar{M}\bar{M}^{\mathsf{T}}$$

- Meaning: $S_v(i,j) \neq 0 \Leftrightarrow$ vertex i and vertex j are incident to the same face ("share a face") in M (*not* necessarily do they form an edge!)
- In the case of triangle meshes: $S_v(i,j) \neq 0 \;\Leftrightarrow\; (i,j)$ is an edge in M



$$\begin{array}{c|ccccccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
1 & 2 & 2 & 1 & 1 & & & & & & & 1 & & 1 & 1 & \\
2 & 2 & 3 & 2 & 1 & & 1 & & & 1 & 1 & 2 & & 1 & 1 & \\
3 & 1 & 2 & 3 & 2 & 1 & 2 & & & 1 & 1 & 1 & & & 1 & \\
4 & 1 & 1 & 2 & 2 & 1 & 1 & & & & & & & & 1 & \\
5 & & & 1 & 1 & 2 & 2 & 1 & & & & & & & & \\
6 & & 1 & 2 & 1 & 2 & 4 & 2 & 1 & 2 & 1 & 1 & & & & \\
7 & & & & & 1 & 2 & 2 & 1 & 1 & & & & & & \\
8 & & & & & & 1 & 1 & 2 & 2 & 1 & & 1 & & & 1 \\
9 & & 1 & 1 & & & 2 & 1 & 2 & 3 & 2 & 1 & 1 & & & 1 \\
10 & & 1 & 1 & & & 1 & & 1 & 2 & 3 & 2 & 2 & 1 & & 1 \\
11 & 1 & 2 & 1 & & & 1 & & & 1 & 2 & 3 & 1 & 2 & & \\
12 & & & & & & & & 1 & 1 & 2 & 1 & 2 & 1 & & 1 \\
13 & 1 & 1 & & & & & & & & 1 & 2 & 1 & 2 & & \\
14 & 1 & 1 & 1 & 1 & & & & & & & & & & 1 & \\
15 & & & & & & & & 1 & 1 & 1 & & 1 & & & 1
\end{array}$$

- Definition: generalized face-face adjacency matrix

$$S_f = \bar{M}^\mathsf{T} \bar{M}$$

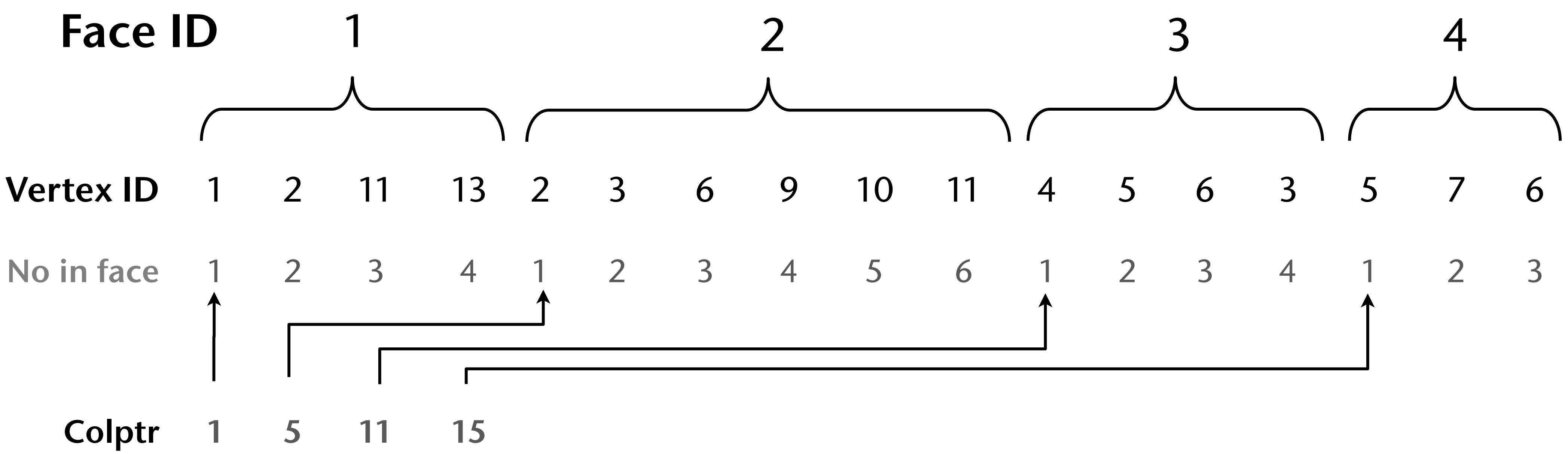
- Meaning:
  - $S_f(i,j)$ = # vertices shared by face i and face j in M
  - In particular: $S_f(i,i)$ = # vertices of face i
  - And for different faces i, j: $S_f(i,j) = 2 \Leftrightarrow i$ and j share an edge

    (provided all faces are convex)
  - Face i is an interior face ⇔

    number of off-diagonal elements equaling $2 = S_f(i,i)$;
    otherwise it has an edge on the border of the mesh

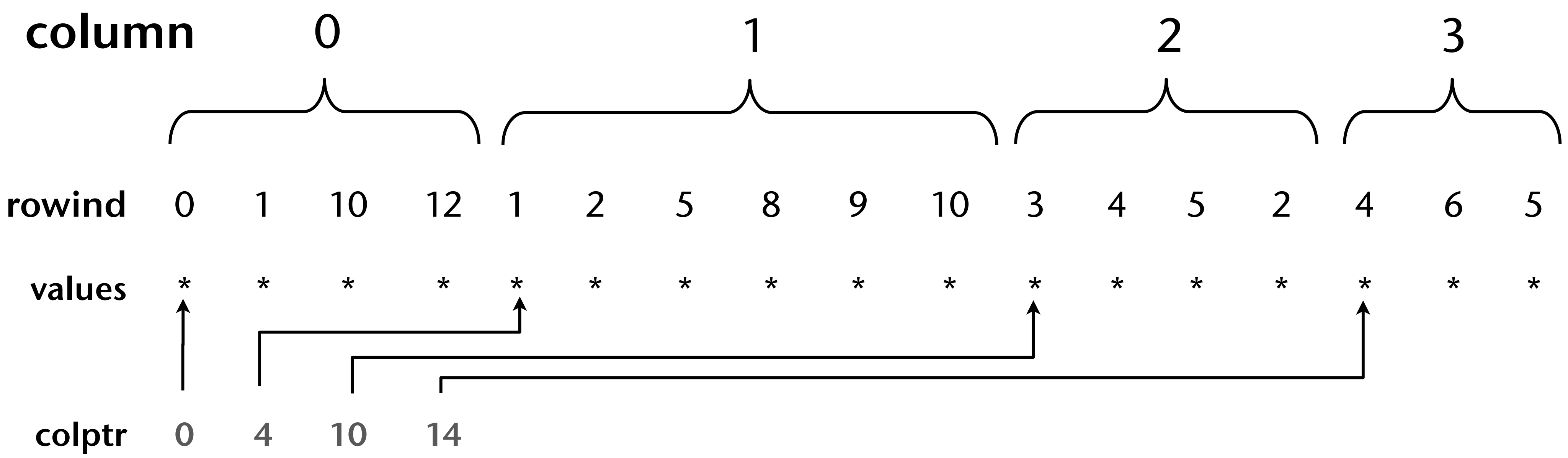
Implementation: Matrix-Vector Multiplication using CSC

- Consider the case $y = M^{\mathrm{T}} \cdot x$ ($M \cdot x$ works similarly)
- Remember the CSC representation



Face ID	1				2						3				4	
Vertex ID	1	2	11	13	2	3	6	9	10	11	4	5	6	3	5	7	6
No in face	1	2	3	4	1	2	3	4	5	6	1	2	3	4	1	2	3
Colptr	1	5	11	15													

- For generality, rename things and allow arbitrary values
  - Also, use *base 0* indices!



column	0				1						2				3		
rowind	0	1	10	12	1	2	5	8	9	10	3	4	5	2	4	6	5
values	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
colptr	0	4	10	14													

Starting Point: Standard Matrix-Vector Multiplication

```
$y = M^T \cdot x$ :

for j = 0..n-1:
 s = 0
 for i = 0..m-1:
   s += M[i][j] * x[j]
 y[j] = s
```

m = # rows (vertices) in M

n = # columns (faces) in M

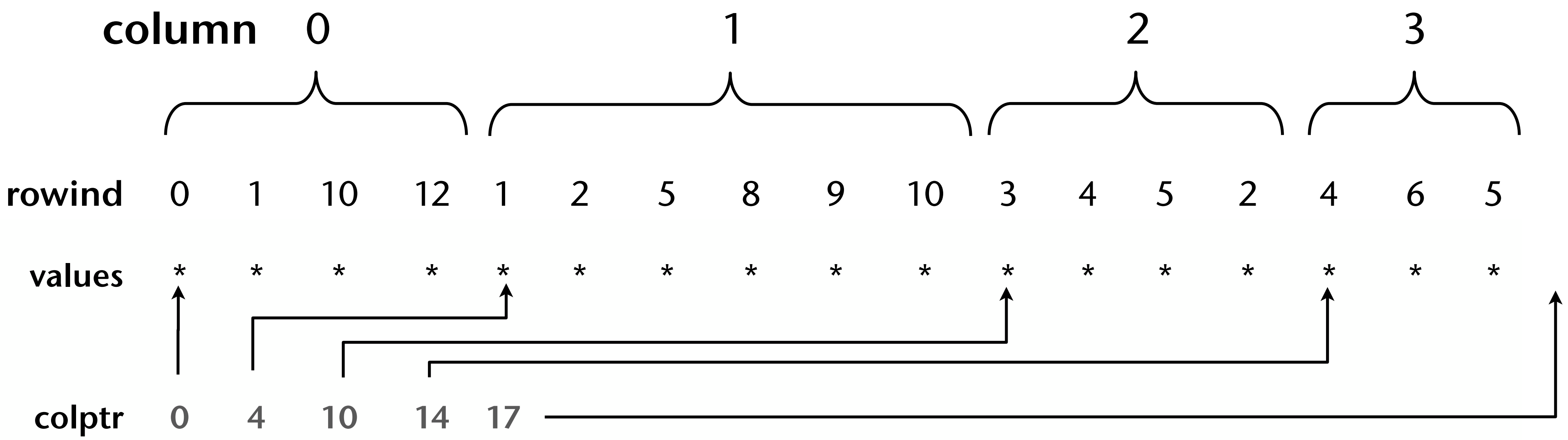
= # entries in colptr

Sparse Matrix-Vector Multiplication (with Arbitrary Values in *M*)

```
$y = M^T \cdot x$ : (SpMV)

for j = 0..n-1:
  s = 0
  for k = colptr[j]..colptr[j+1]-1:
    s += values[k] * x[ rowind[k] ]
  y[j] = s
```

n = # columns (faces) in *M*
= # entries in colptr



column	0				1						2				3		
rowind	0	1	10	12	1	2	5	8	9	10	3	4	5	2	4	6	5
values	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
colptr	0	4	10	14	17												

Sparse Matrix-Vector Multiplication for Mesh Matrices

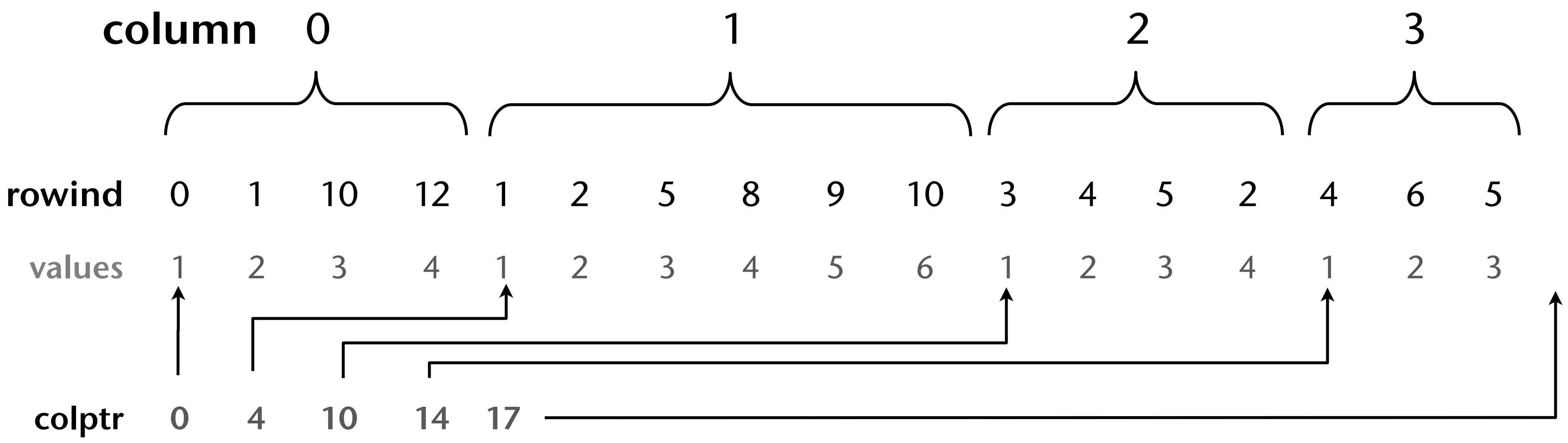
$y = M^T \cdot x$:

```
for j = 0..n-1:
 s = 0
 value = 1
 for k = colptr[j]..colptr[j+1]-1:
   s += value * x[ rowind[k] ]
   value ++
 y[j] = s
```

n = # columns (faces) in *M*
   = # entries in colptr

Exploit the fact that

1. values in *M* (and, thus, in the values array) are incrementing by 1, starting with 1 in every column; and,
2. all rotations of the values within a column represent the same face



column	0				1						2				3		
rowind	0	1	10	12	1	2	5	8	9	10	3	4	5	2	4	6	5
values	1	2	3	4	1	2	3	4	5	6	1	2	3	4	1	2	3
colptr	0	4	10	14	17												

Sparse Matrix-Vector Multiplication for Triangle Meshes

```
y = M^T·x :

for j = 0..n-1:
 s = 0
 value = 1
 for k = 3*j,..,3*(j+1)-1:
   s += value * x[ rowind[k] ]
   value ++
 y[j] = s
```

n = # columns (faces) in M
= # entries in colptr

In addition, exploit the fact that all faces have 3 vertices
⟶ can omit **`colptr`**, too.
(And good compilers can unroll the inner loop automatically!)

column	0			1			2			3			4			5	
rowind	0	1	10	12	1	2	5	8	9	10	3	4	5	2	4	6	5

Note: this is just a placeholder,
not a real mesh!

Implementation Details

- Mult. with replacements: can be integrated into the SpMV routine directly
- We *could* use libraries for general sparse matrix-vector multiplication
- Experience shows: specialized versions for the case of mesh matrices pays off, especially in case of multiplication with replacements
- Parallelization for the GPU:
  - In case of $y=M^Tx$: every thread i computes one $y[i]$ (outer loop), thread i reads one column of M, no inter-thread communication and synchronization needed
    - Unfortunately, no coalesced memory access
  - In case of $y=Mx$: again one thread per column of M, but threads accumulate their results into y, thus atomic add is necessary
  - For more, see the course "Massively Parallel Algorithms"

Performance

- Computing vertex normals:

	Trimesh2	ours(CPU)	ours(GPU)
Embreea Orchid ($4M\Delta$)	.61	.32	0.0078
Earhart Flight Suit ($21,5M\Delta$)	3.31	1.93	0.046
Pergolesi side chair ($29M\Delta$)	4.45	2.81	0.073

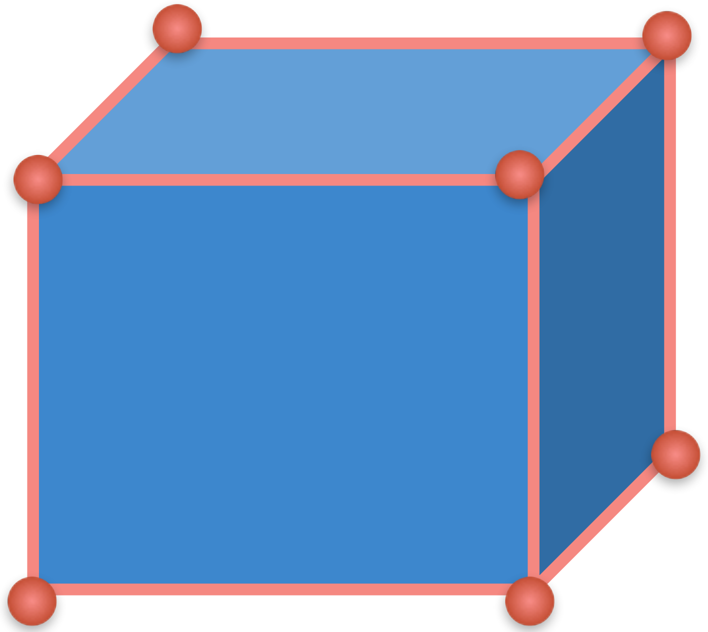
The Euler Equation for Graphs / Polyhedra

- Theorem (Euler's Equation):
  Let V, E, F = number of vertices, edges, faces in a polyhedron that is homeomorph to a sphere. Then,

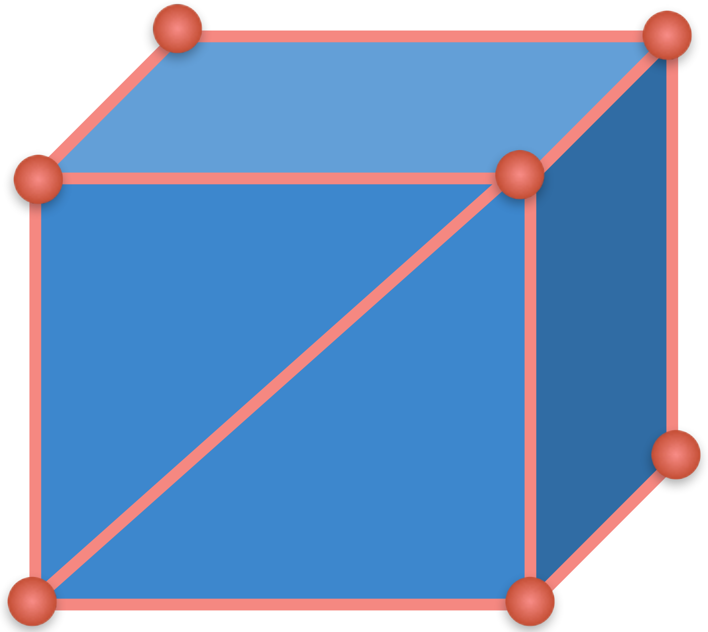
$$V - E + F = 2$$

Voted 2nd most beautiful theorem!

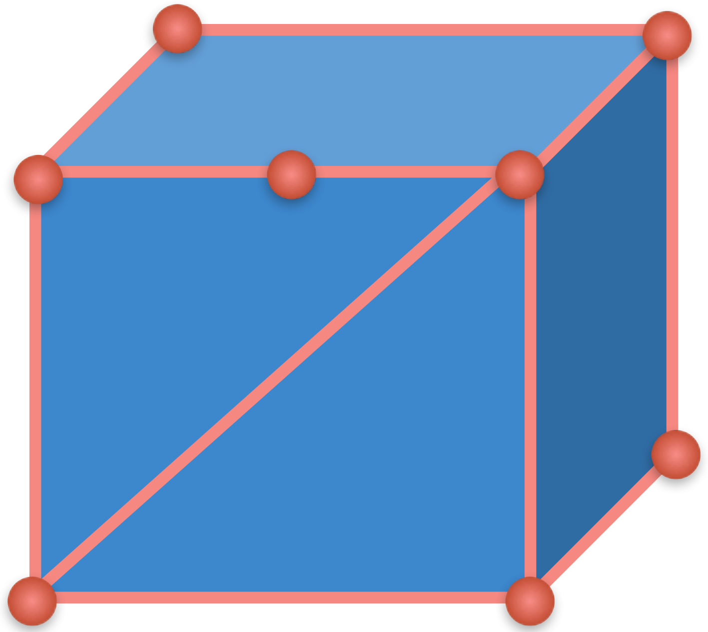
- Examples:



$V = 8$
$E = 12$
$F = 6$



$V = 8$
$E = 12+1$
$F = 6 +1$

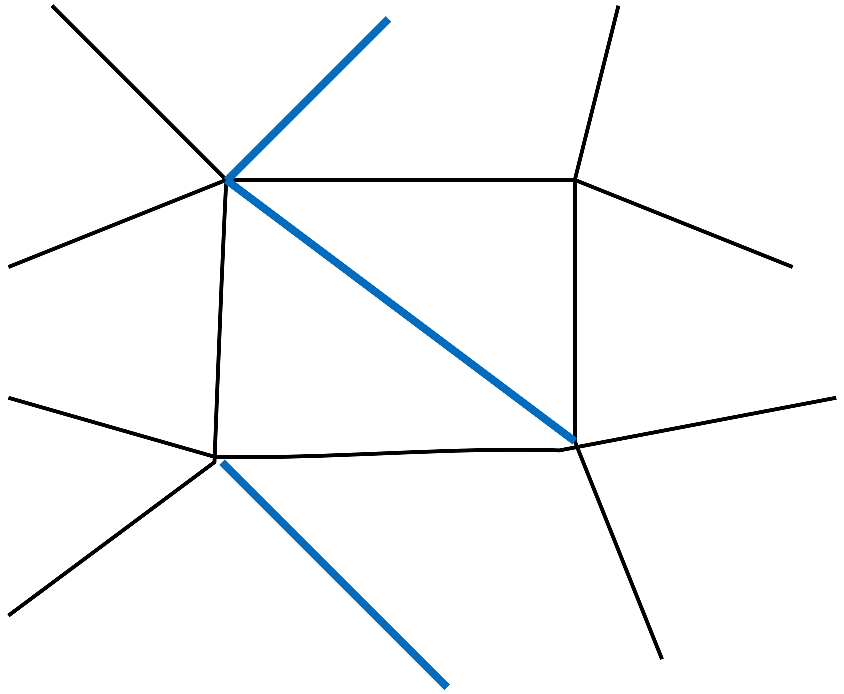


$V = 8+1$
$E = 12+1+1$
$F = 6 +1$

Proof (given by Cauchy)

- Given: a closed mesh (polyhedron, equivalent to sphere)
- First idea:
  - Remove one facet → yields an open mesh; the border is exactly the edge cycle of the removed facet)
  - Stretch the mesh by pulling its border apart until it becomes a planar graph (works only if the polyhedron is homeomorph to a sphere)
  - It remains to show: $V - E + F = 1$
- Second idea: triangulate the graph (i.e., the mesh)
  - Draw diagonals in all facets with more than 3 vertices
  - For each new diagonal, we have

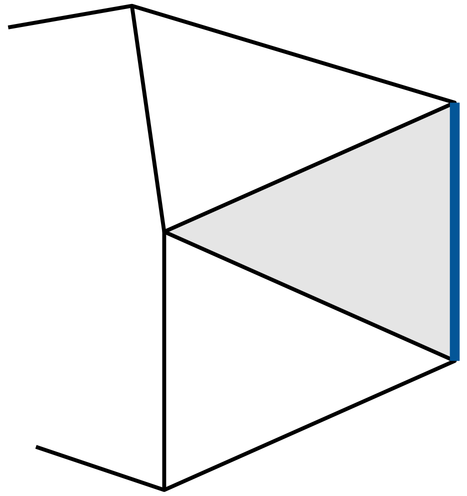
$$V' - E' + F' = V - (E+1) + (F+1) = V - E + F$$

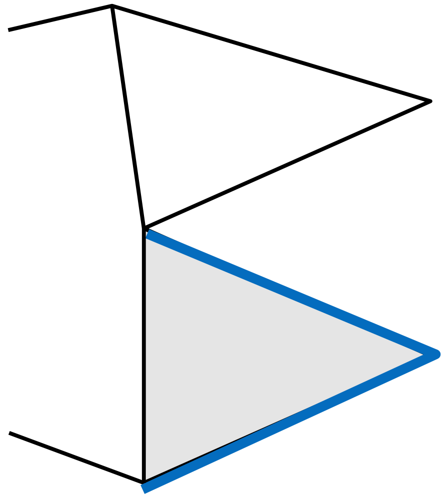


- The graph has a border; triangles have 0, 1, or 2 "border edges"
- Repeat one of the following two transformations:
  - If there is a triangle with exactly one border edge, remove this triangle

    It follows that $V' - E' + F' = V - (E-1) + (F-1) = V - E + F$
  - If there is a triangle with exactly two border edges,remove the triangle

    It follows that $V' - E' + F' = (V-1) - (E-2) + (F-1) = V - E + F$
- Repeat, until only one triangle remains
  - For that triangle, the Euler equation is obviously correct
  - Because the value of *V-E+F* is invariant through all above transformations, the equation is also true for the original graph, hence for the original mesh





Application of Euler's Equation to Meshes

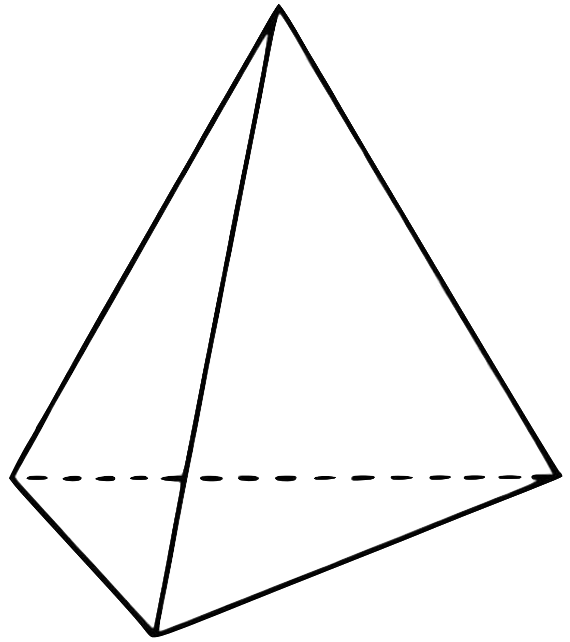
- Euler's Equation → relationship between #triangles and #vertices in a closed *triangle* mesh
- In a closed triangle mesh, each edge is incident to exactly 2 triangles , so

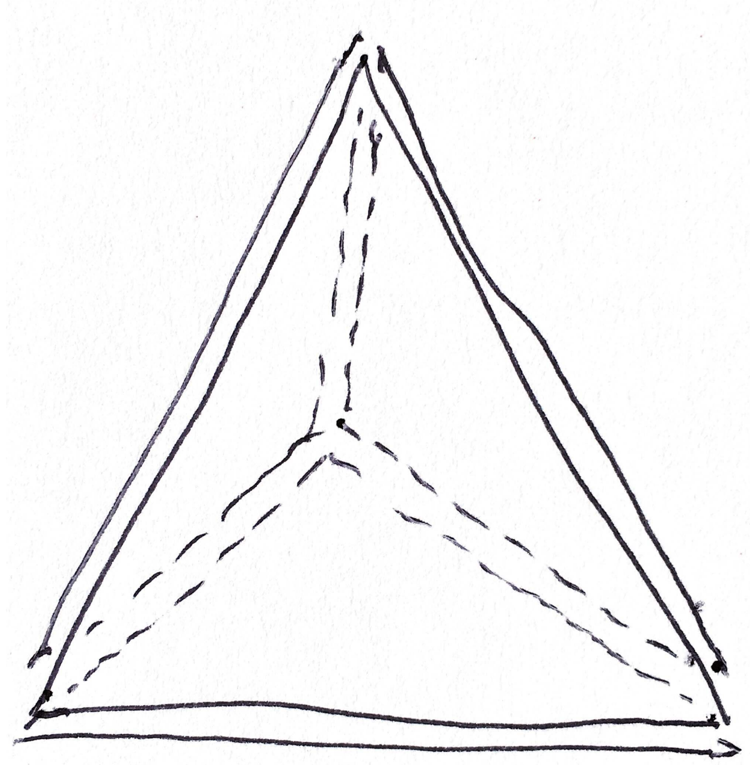
$$3F = 2E$$

- Plug this into Euler's equation:

$$2 = V - \frac{3}{2}F + F \;\Leftrightarrow\; \frac{1}{2}F = V{-}2$$

- Therefore, for large *triangle* meshes $F \approx 2V$



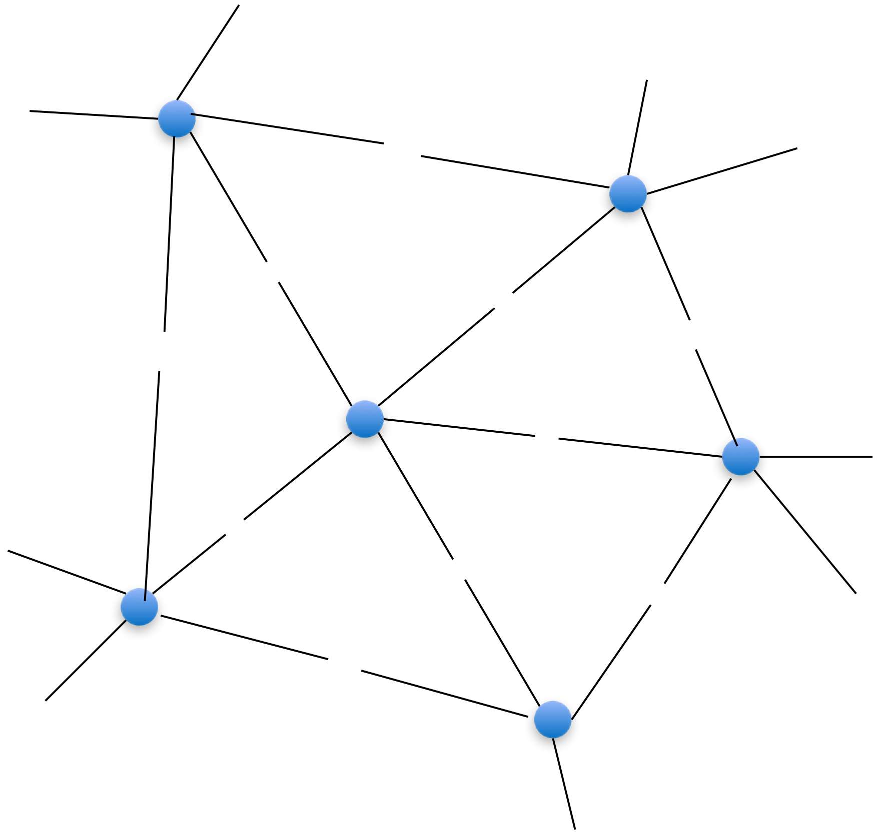


Average Vertex Degree

- Theorem:

  In all closed, two-manifold, triangle meshes,
  the average vertex degree ≈ 6 (even with higher genus).

- Proof:
  - We know: $3F = 2E$
  - Plugging this into Euler: $V - E + \frac{2}{3}E = 2$
  - Solve for E: $E = 3(V-2)$
  - Calculate average degree by edge-splitting trick:



$$\text{average degree} = \frac{\#\text{ split edges}}{V} = \frac{2E}{V} = \frac{6V-12}{V} \approx 6$$

Complexity of Polyhedra

- Similarly, we can derive (for closed triangle meshes)

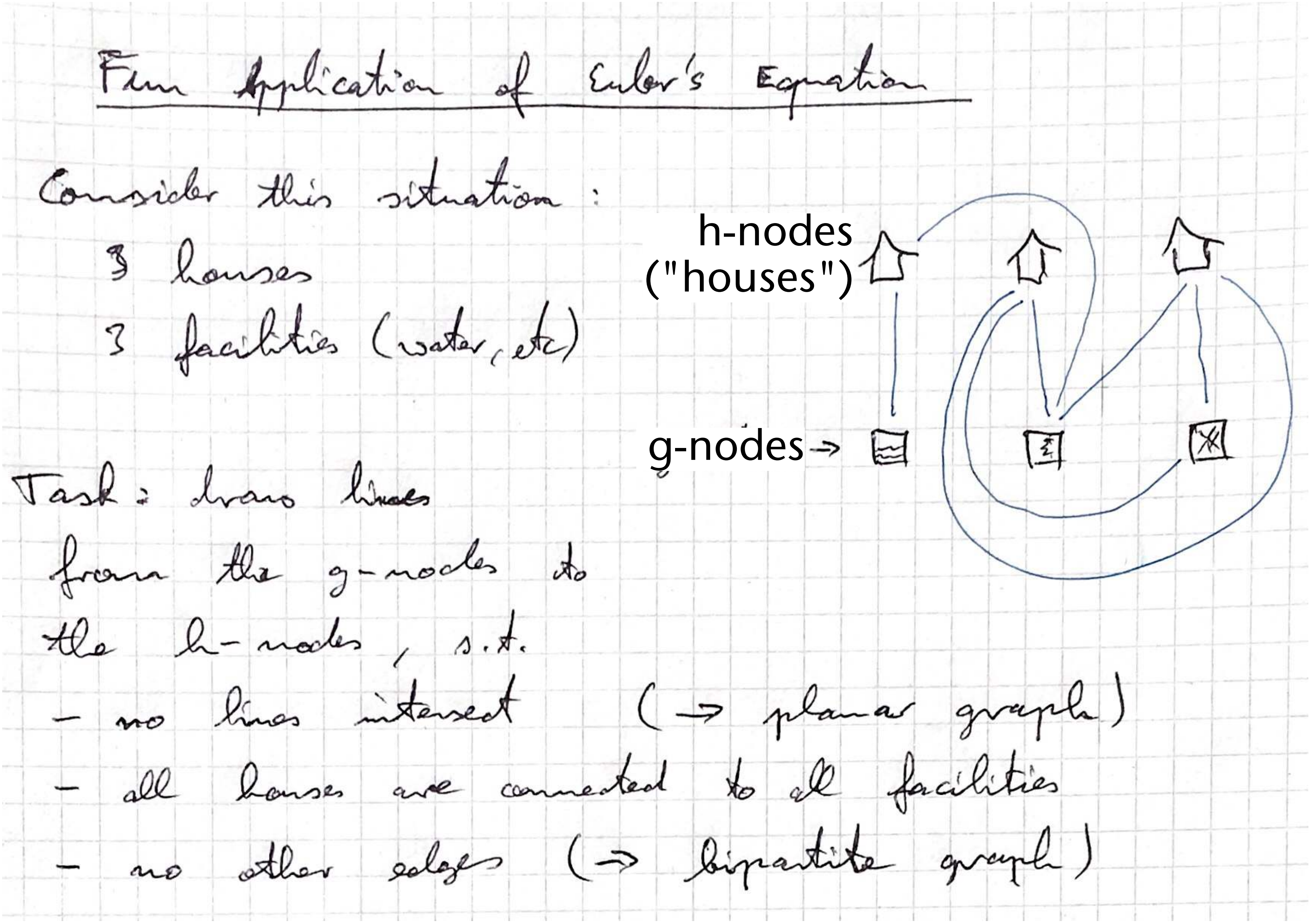
$$E \approx 3V$$

- Theorem (on the complexity of polyhedra):

  For all closed triangle meshes, $O(F) = O(E) = O(V)$

- Analogously, we can prove this for quad meshes
- We can even prove it for all closed meshes!
- Usually, it is also true for open meshes, where the border is not "too long" relative to the total number of edges (i.e., not too many faces are border faces)

Fun Application of Euler's Equation



Fun Application of Euler's Equation

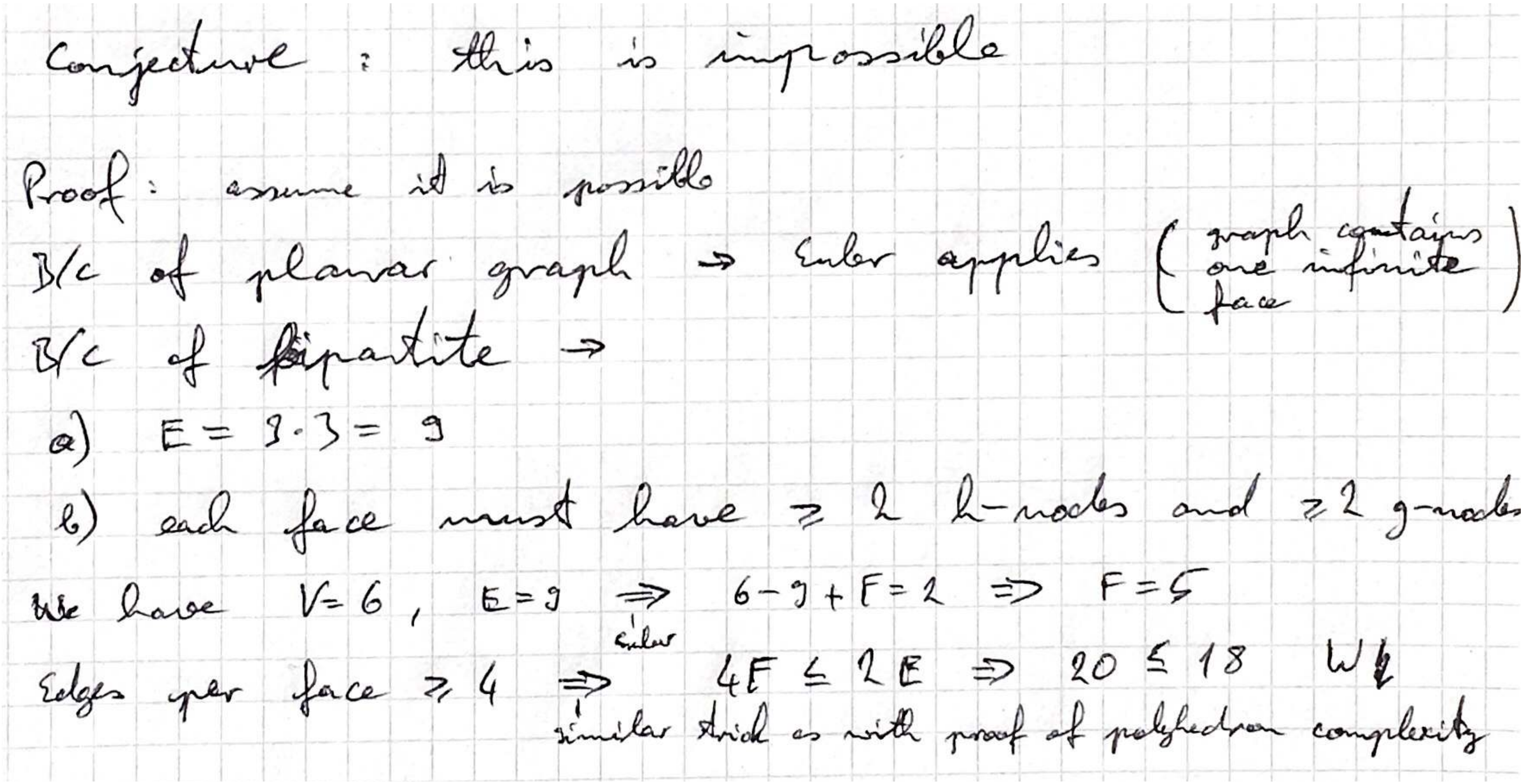
Consider this situation:

3 houses

3 facilities (water, etc)

Task: draw lines from the g-nodes to the h-nodes, s.t.

- no lines intersect (⇒ planar graph)
- all houses are connected to all facilities
- no other edges (⇒ bipartite graph)



Conjecture: this is impossible

Proof: assume it is possible

B/c of planar graph ⇒ Euler applies (graph contains one infinite face)

B/c of bipartite ⇒

a) $E = 3 \cdot 3 = 9$

b) each face must have ≥ 2 h-nodes and ≥ 2 g-nodes

We have $V = 6$, $E = 9$ $\overset{\text{Euler}}{\Rightarrow}$ $6 - 9 + F = 2$ ⇒ $F = 5$

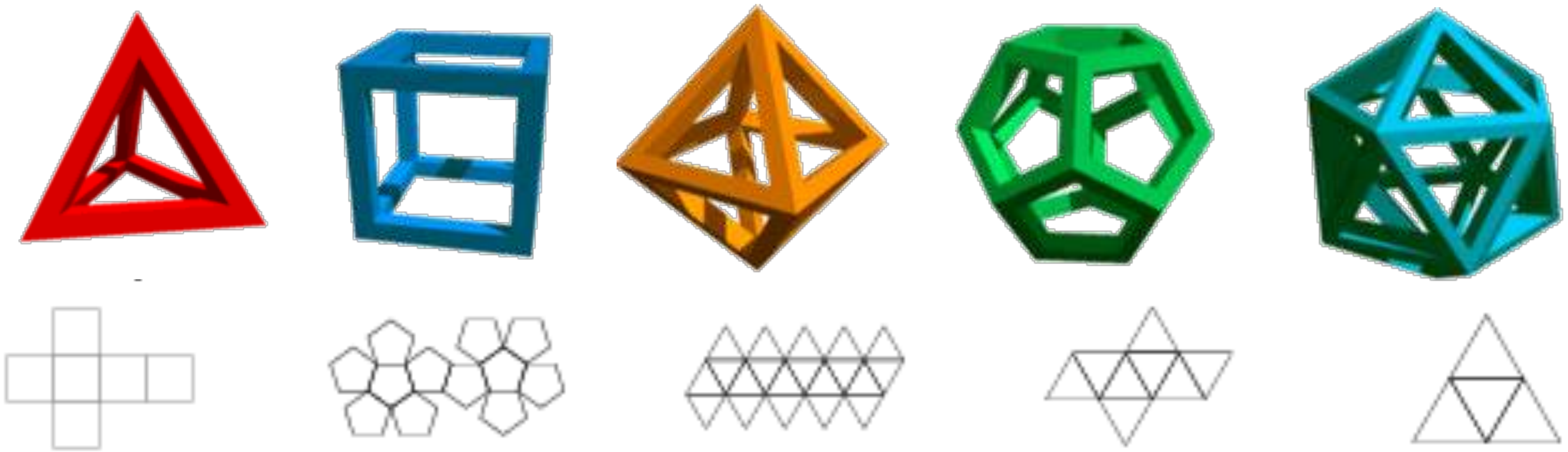
Edges per face ≥ 4 $\underset{\text{similar trick as with proof of polyhedron complexity}}{\Rightarrow}$ $4F \leq 2E$ ⇒ $20 \leq 18$ W↯

The Platonic Solids

- Definition Platonic Solid:
  A convex polyhedron consisting of a number of *congruent* & *regular* polygons, with the same number of faces meeting at each vertex.
  - Regular polygon = all sides are equal, all angles are equal
- Theorem (Euklid):

  There are exactly *five* platonic solids.

Voted 4th most beautiful theorem!



Proof

- All facets have the same number of edges = *n*; therefore:

$$2E = nF \;\Leftrightarrow\; F = \frac{2}{n}E$$

- All vertices have the same number of incident edges = *m*; therefore

$$2E = mV \;\Leftrightarrow\; V = \frac{2}{m}E$$

- Plugging this into Euler's equation:

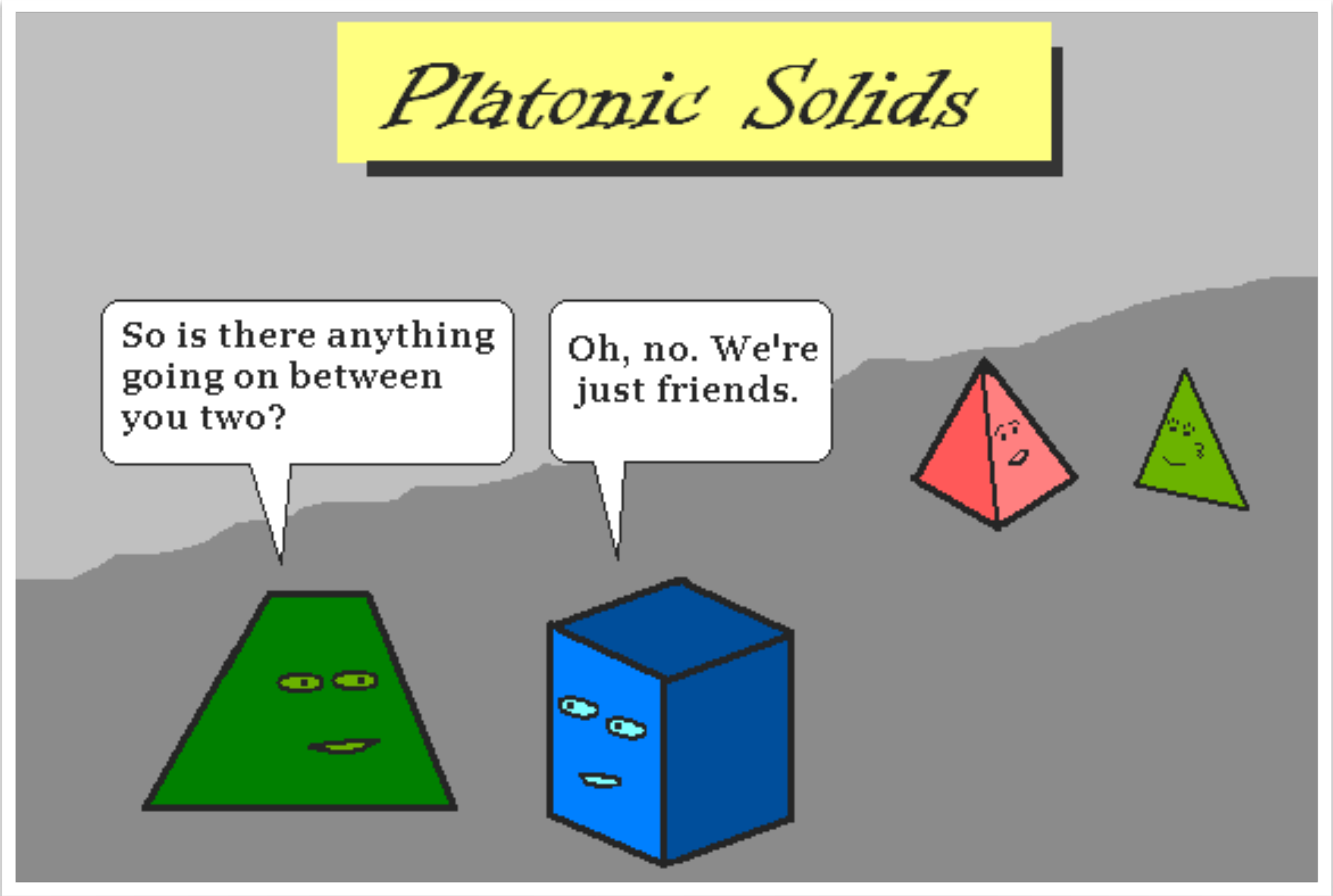
$$2 = V - E + F = \frac{2}{m}E - E + \frac{2}{n}E \;\Leftrightarrow\; \frac{2}{E} = \frac{2}{m} - 1 + \frac{2}{n}$$

- Yields the following condition on *m* and *n*:

$$\frac{1}{m} + \frac{1}{n} = \frac{1}{2} + \frac{1}{E} > \frac{1}{2}$$

- Additional condition: *m* and *n* both must be ≥ 3
- Which {*m*,*n*} fulfill these conditions:

{3,3} {3,4} {4,3} {5,3} {3,5} ⟵ Schläfli symbols



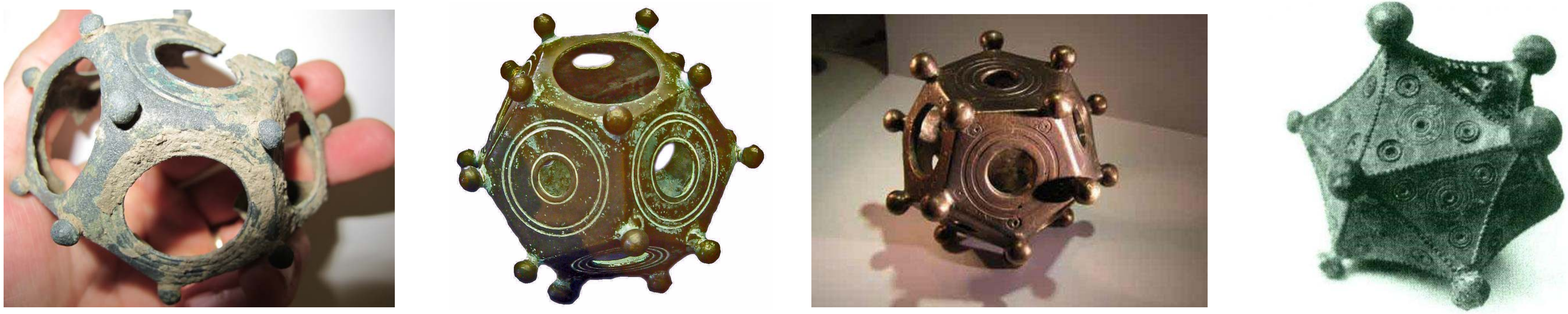
C. Burke, 2008

Digression: Platonic Solids in History & the Arts

- A specimen of an icosahedron from Egypt, 2nd century B.C. – 4th century A.D.

- The platonic solids have been known at least 1000 years before Plato in Scotland

The Curious Case of the Roman Dodecahedrons



Nobody has a convincing answer as to what their purpose was!

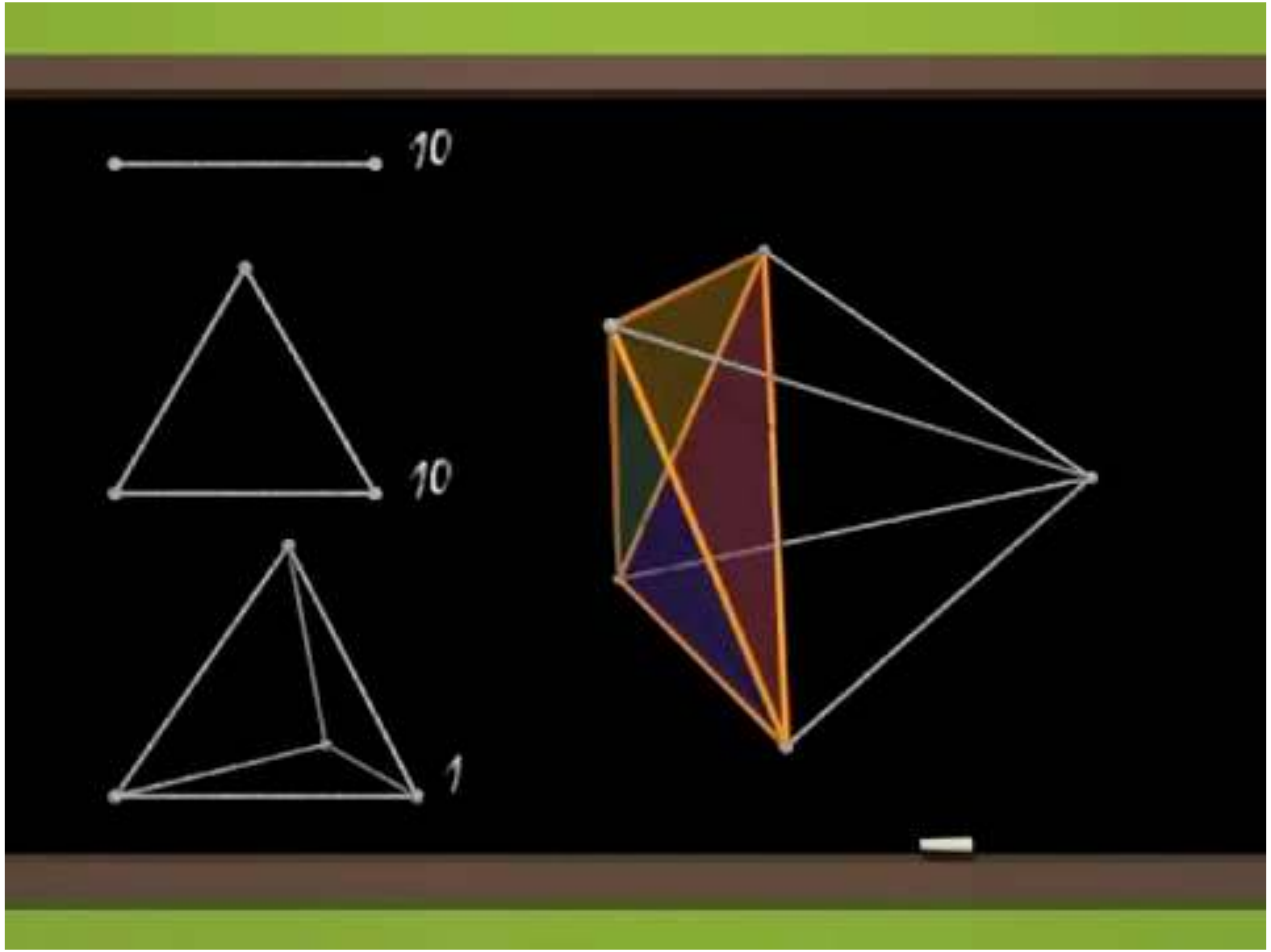
Platonic Solids in the Arts and Architecture

Portrait of Johannes Neudörfer and his Son
Nicolas Neufchatel,
1527—1590

Oceano, CA, USA

Dürer: *Melencolia I*

Regular Polyhedra in 4-Dimensional Space

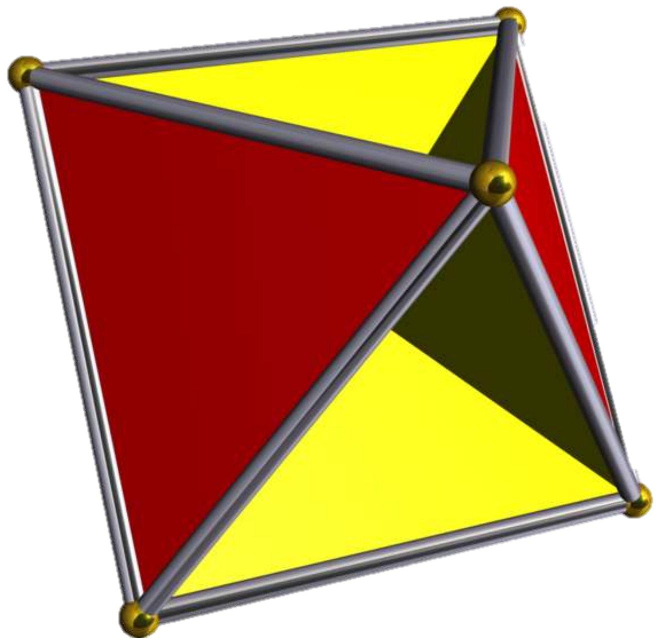


"Dimensions" by Jos Leys, Étienne Ghys, Aurélien Alvarez
http://www.dimensions-math.org/Dim_E.htm

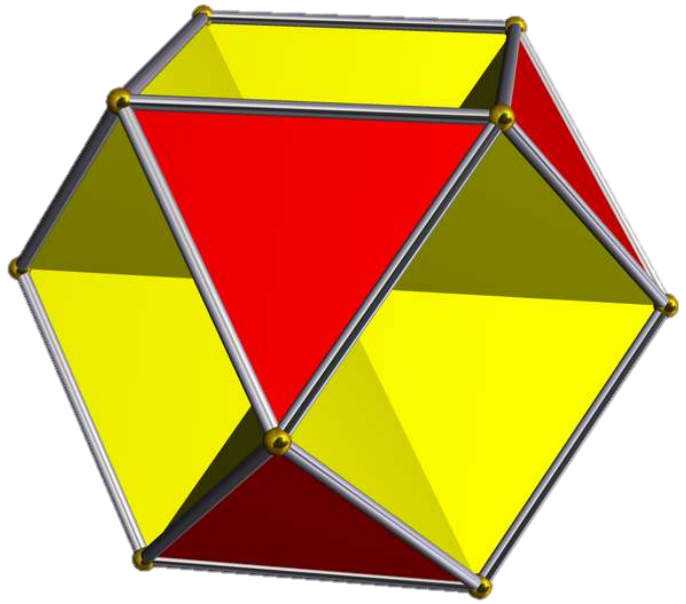
The regular polyhedra (sort of the generalization of Platonic solids in higher dimensions) were discovered by Schläfli.

The Euler Characteristic

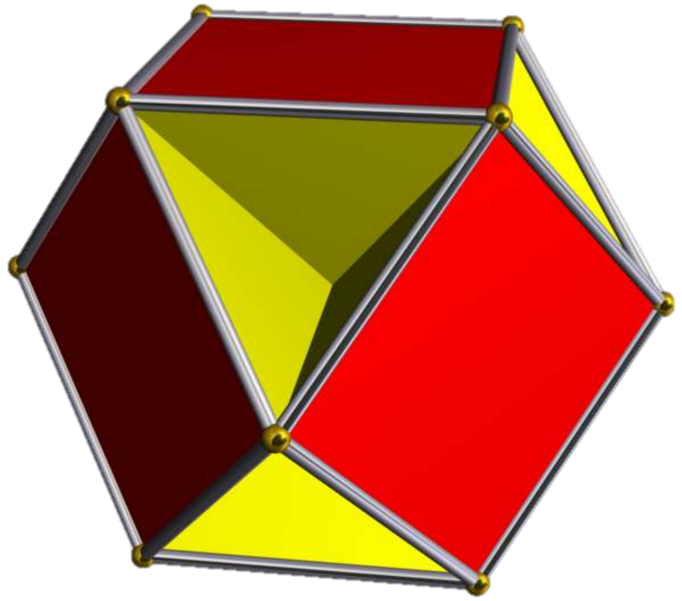
- Caution: the Euler equation holds only for polyhedra, that are topologically equivalent to a sphere!
- Examples:



Tetrahemihexahedron
6 - 12 + 7 = 1



Octahemioctahedron
12 - 24 + 12 = 0



Cubohemioctahedron
12 - 24 + 10 = -2

V-E+F

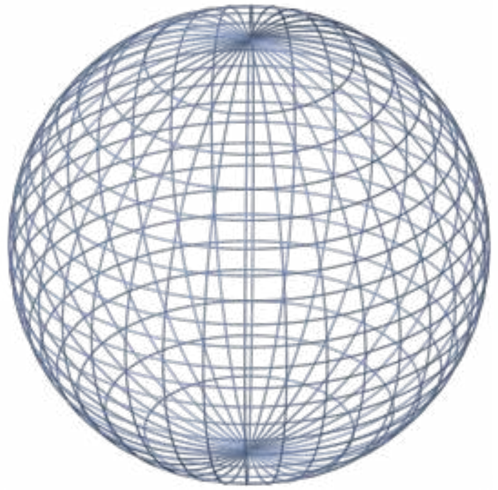
- *But*: the quantity *V-E+F* stays the same no matter how any given polyhedron is deformed (homeomorphically!)

  → so the quantity *V-E+F* is a topologic invariant

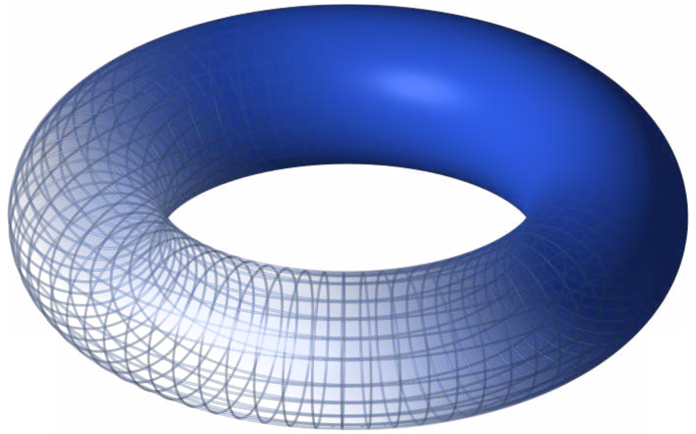
- Definition Euler characteristic:

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

- Examples:

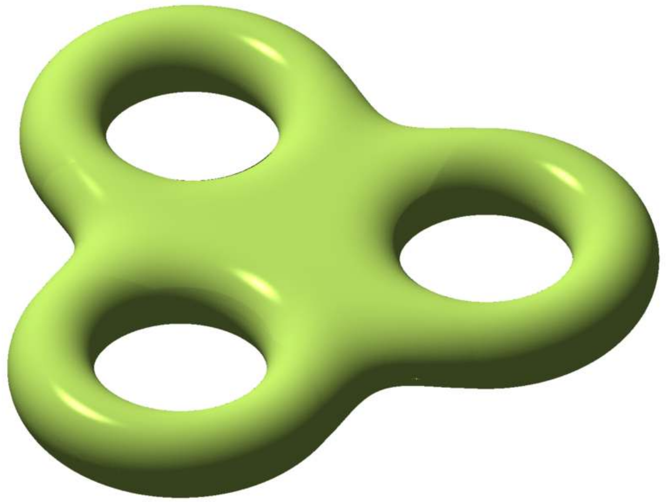


2

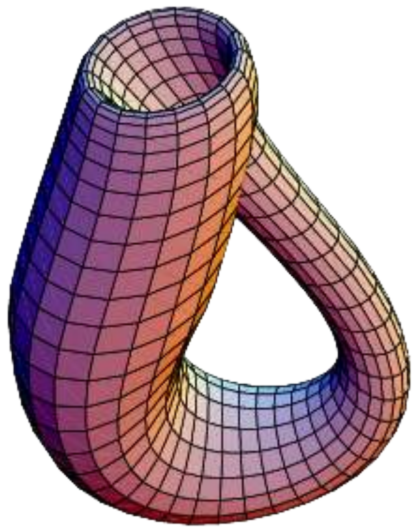


0

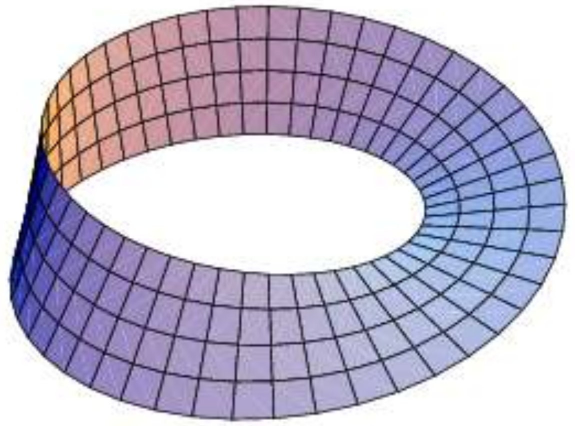
-2



-4

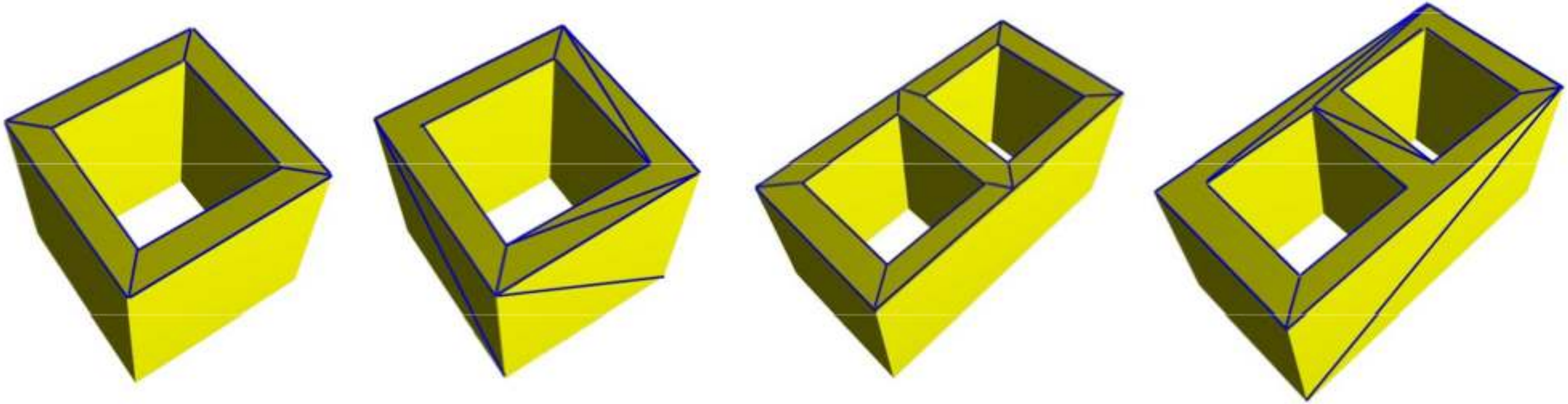


0



0

- The Euler characteristic is even independent of the tessellation!



V = 16	V = 16	V = 28	V = 24
E = 32	E = 36	E = 56	E = 48
F = 16	F = 20	F = 26	F = 22
$\chi = 0$	$\chi = 0$	$\chi = -2$	$\chi = -2$

The Classification Theorem

Assume we are given a *closed* and *orientable* mesh consisting of just one *shell*. Then the following holds:

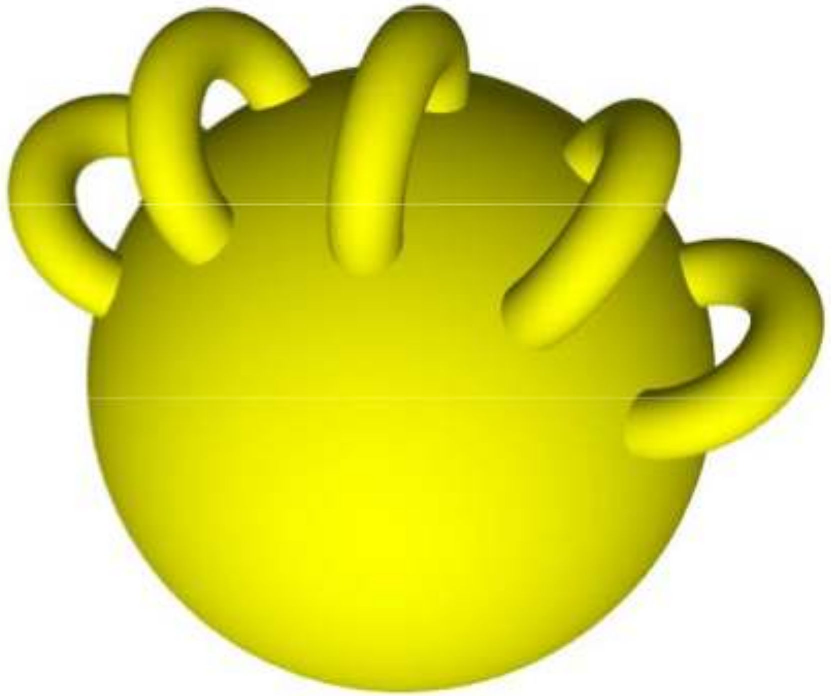
The Euler characteristic $\chi = 2, 0, -2, \ldots$ ⇔

the mesh is topologically equivalent to a sphere, a torus, a double torus, etc.

The Euler-Poincaré Equation

- Generalization of the Euler equation for 2-manifold, closed objects (possibly with several, non-contiguous "surfaces"):

$$V - E + F = 2(S - G)$$

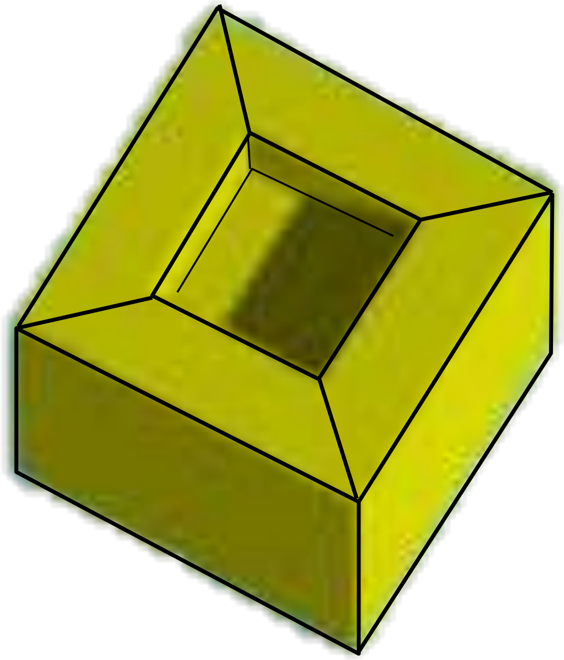


- G = # handles, S = # shells/surfaces (Schalen)
- G is called "Genus"
- Handle (hole, Loch): a piece of rubber string *inside* a handle cannot be shrunk towards a single point
- Shell (Schale) = 2-manifold, contiguous surface without self-intersection; by walking on the surface of a shell, *each and every point on it can be reached*, without ever leaving it
- We can cut out so-called "voids" (Aushöhlungen) by "inner" shells
  - In CG, we usually consider only meshes consisting of a single shell!

Examples

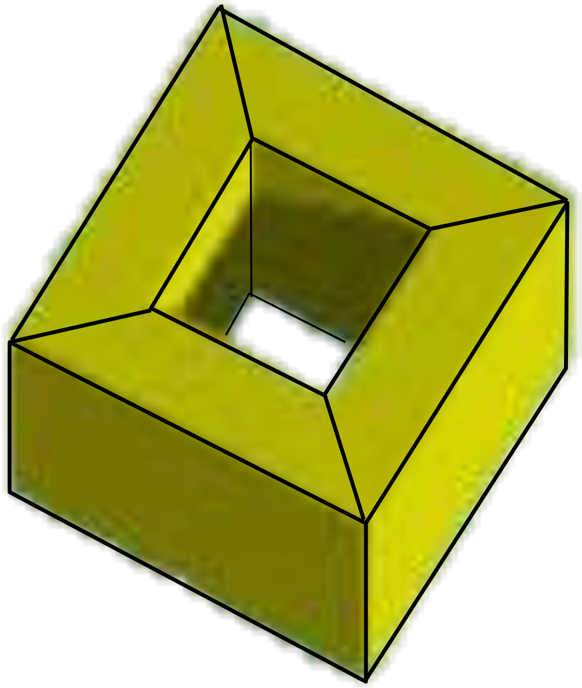
- $V = 16,\ E = 28,\ F = 14,\ S = 1,\ G = 0$:

  $V - E + F = 2 = 2(S - G)$



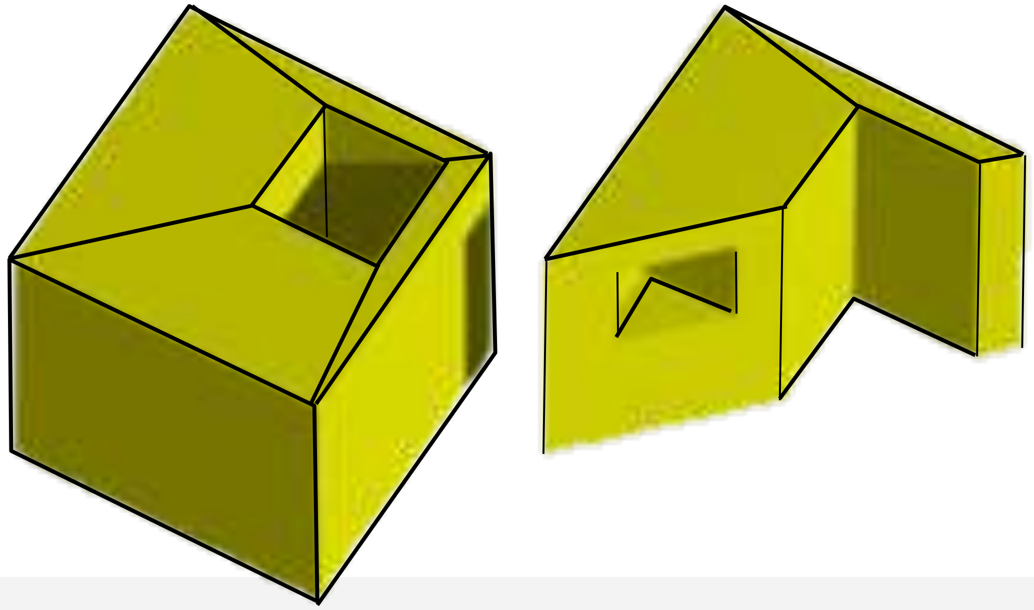
- $V = 16,\ E = 32,\ F = 16,\ S = 1,\ G = 1$:

  $V - E + F = 0 = 2(S - G)$

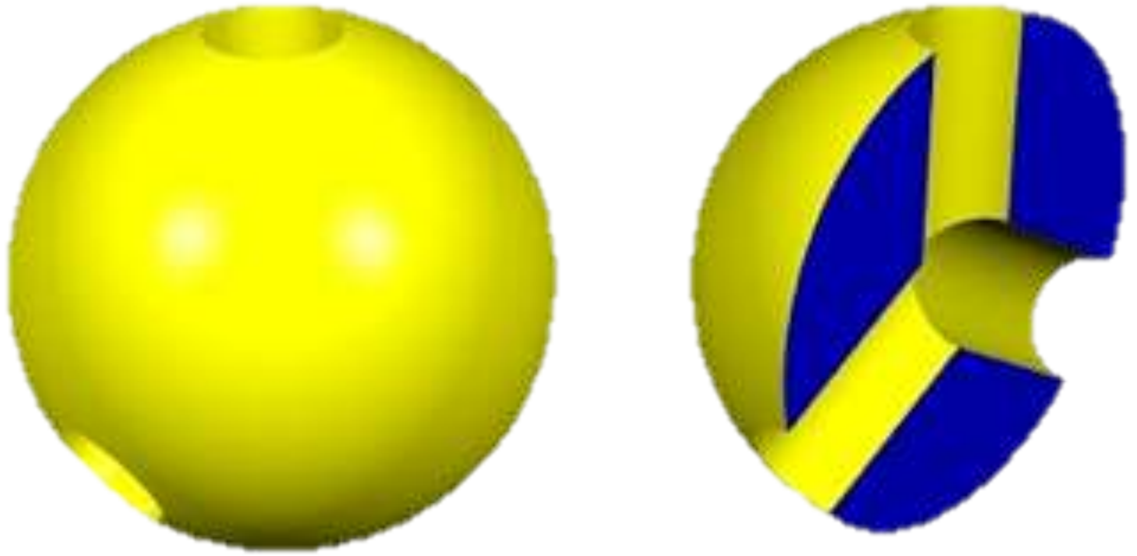


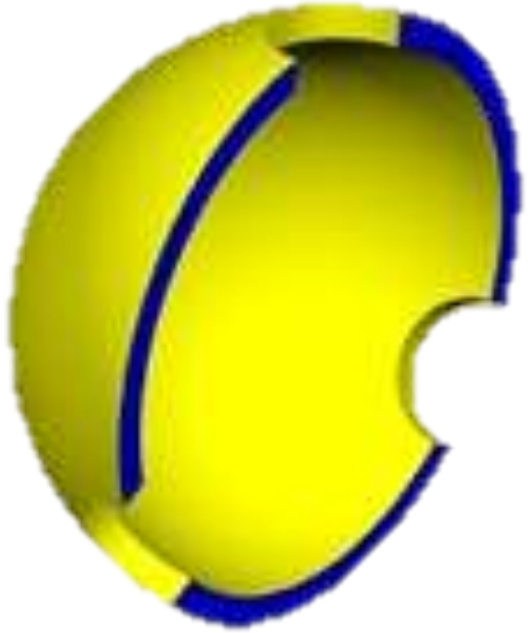
- $V = 16+8,\ E = 32+12,\ F = 16+6, G = 1,\ S = 2$:

  $V - E + F = 2 = 2(S - G)$



- Beware: sometimes it is not obvious to determine the genus!
- Approach: find homeomorphism, so that genus becomes obvious
- Example:





1.

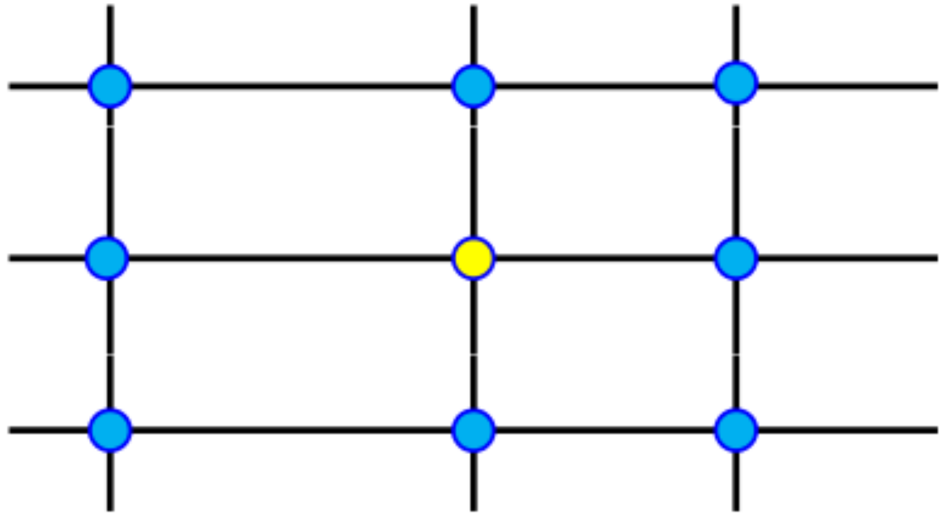
2.

3.

  - Genus = 2

- What is the genus of this object?

Regular Quad Meshes



- Definition "regular quad mesh": Each face of the mesh is a quadrangle (a.k.a. *quad, quadrilateral*), and each vertex has degree 4.

- Theorem:
  Each closed, orientable, regular quad mesh must be topologically equivalent to a torus
- Proof:
  - In such a mesh we have: $4V = 2E \longrightarrow V = \frac{1}{2} E$
  - By counting the edges via the faces: $4F = 2E \longrightarrow F = \frac{1}{2} E$
  - Therefore $\chi(M) = V - E + F = 0 \longrightarrow$ mesh = torus (by previous theorem)

Regular Meshes

- Definition:
  A regular (n,m,g)-mesh is a closed, orientable mesh, with genus g, where each facet has exactly n edges, and each vertex has exactly degree m.

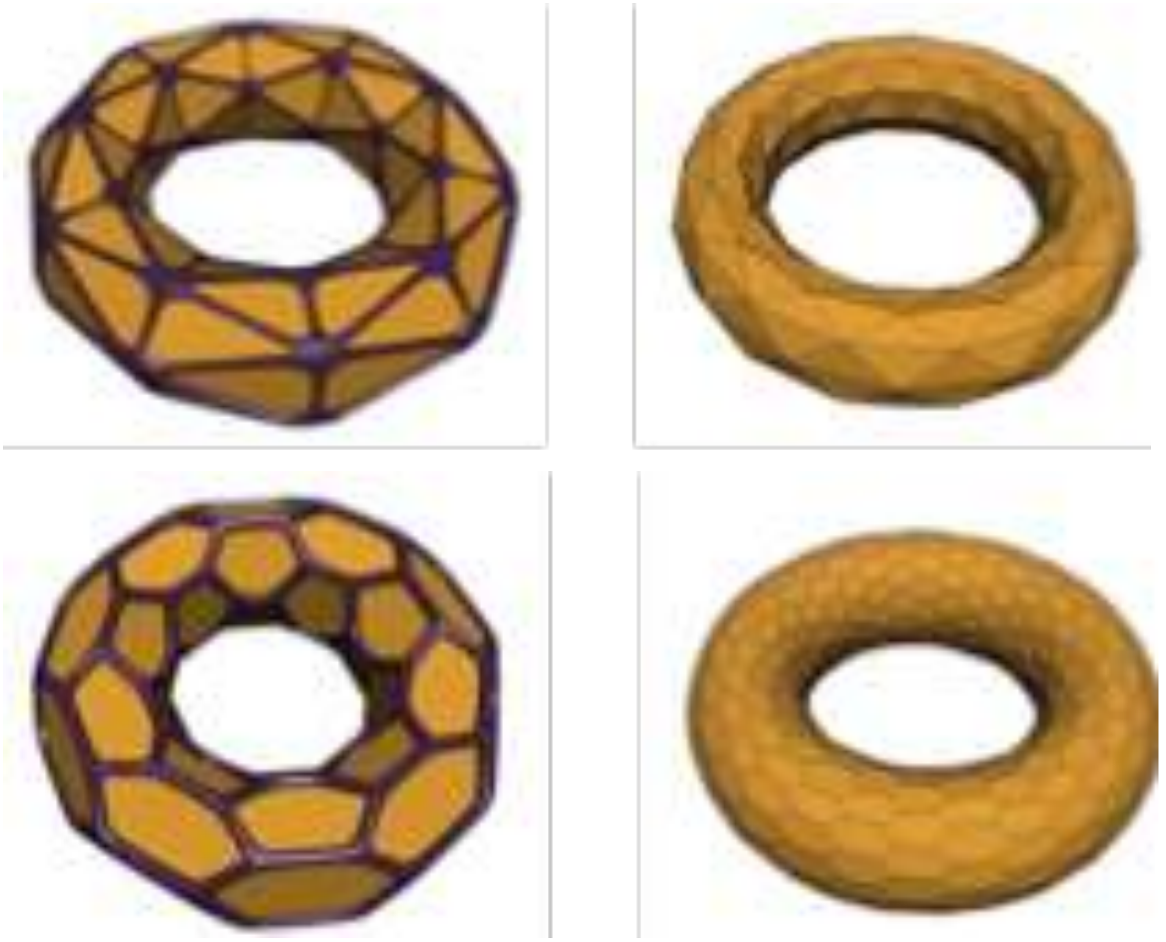
- Examples:
  - The (n,m,0)-meshes are exactly the Platonic solids
  - The regular quad mesh is a regular (4,4,1)-mesh

- In a regular mesh we have

$$nF = 2E = mV$$

- Plugging that into the Euler equation, we obtain

$$\left(\frac{1}{n} + \frac{1}{m} - \frac{1}{2}\right) E = 1 - G$$



- For regular genus-1 meshes we have:

$$nm - 2n - 2m = 0$$

  - The only possible integer solutions are: (4, 4, 1) (3, 6, 1) (6, 3, 1)

- Theorem:
  There are infinitely many regular (n,m,g)-meshes for all pairs (n,m) with $nm - 2n - 2m > 0$ (genus $G > 1$).

- Proof:
  - Rewrite equations (1) and (2) from previous slide:

$$H = \frac{g - 1}{nm - 2n - 2m} \qquad E = 2nmH\ , \quad F = 4mH\ , \quad V = 4nH$$

  - Let $g_1 = nm - 2n - 2m$;
    then $E_1 = 2nm$, $V_1 = 4n$, $F_1 = 4m$ are solutions of the 3 equations
  - Let $g_k = k(g_1 - 1) + 1$, $k = 1, 2, \ldots$;
    then $E_k = kE_1$, $V_k = kV_1$, $F_k = kF_1$ are solutions, too

- Remark: the proof does not tell us how to construct such meshes
- Example of a (4,5,2)-mesh

  - We can add an arbitrary number of such "chain rings" to achieve an arbitrarily high genus and, thus, a (4,5,*g*)-mesh with *g*>=2