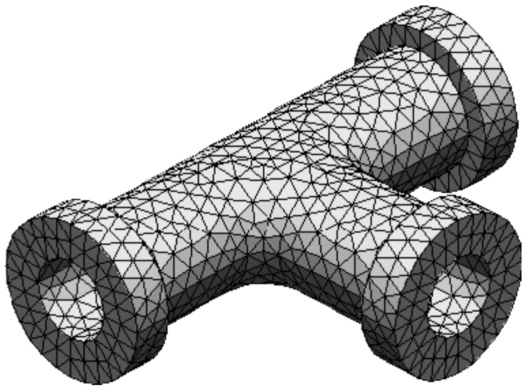
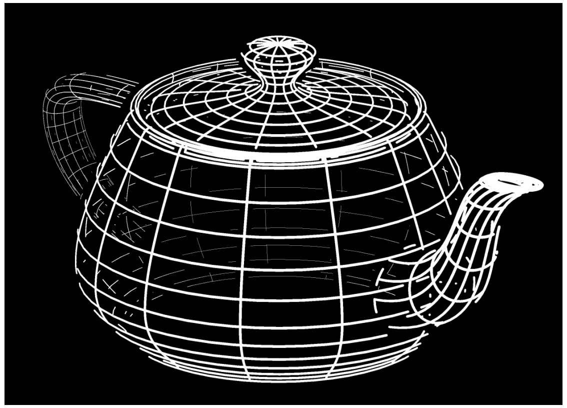
Advanced Computer Graphics
Boundary Representations for Graphical Models

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

The Problem

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

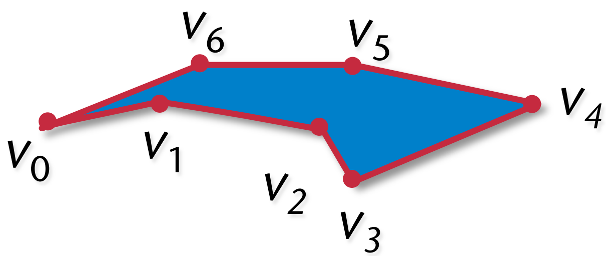
- Definition *Boundary-Representation* (*B-Rep*):
  Objects "consist" of
  1. Triangles, quadrangles, and polygons (i.e., geometry)
  2. Incidence and adjacency relationships (i.e., "topology", "connectivity")
- By contrast, there are also representations that try to model the volume directly, or that 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_iv_j.
- The degree of a node/vertex := number of incident edges

Polygons

- A polygon is a geometric graph $P=(V, E)$, where $V=\{v_0,v_1,\ldots,v_{n-1}\} \subset \mathbb{R}^d$, $d \geq 2$, and $E=\{ (v_0,v_1), \ldots, (v_{n-1}, v_0) \}$.
- Nodes are called vertices (sometimes points or corners).



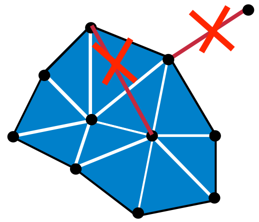
- A polygon is called
  - flat, if all vertices lie in the same plane;
  - simple, if it is flat and if the *intersection of every two edges* in E is either empty or a vertex in V, and if every vertex is incident to exactly two edges (i.e., if the polygon does not have self intersections).
- By definition, we will consider only closed polygons

Mesh (Polygonal Mesh)

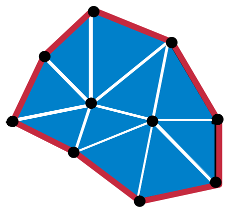
- Let M be a set of closed, simple polygons P_i ;

  let $V = \bigcup_i V_i \quad E = \bigcup_i E_i$

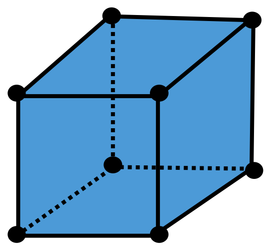
- M is called a mesh iff
  - the intersection of two polygons in M is either empty, a point $v \in V$ or an edge $e \in E$; and
  - each edge $e \in E$ belongs to at least one polygon (no dangling edges)



- The set of all edges, belonging to one polygon only, is called the border of the mesh

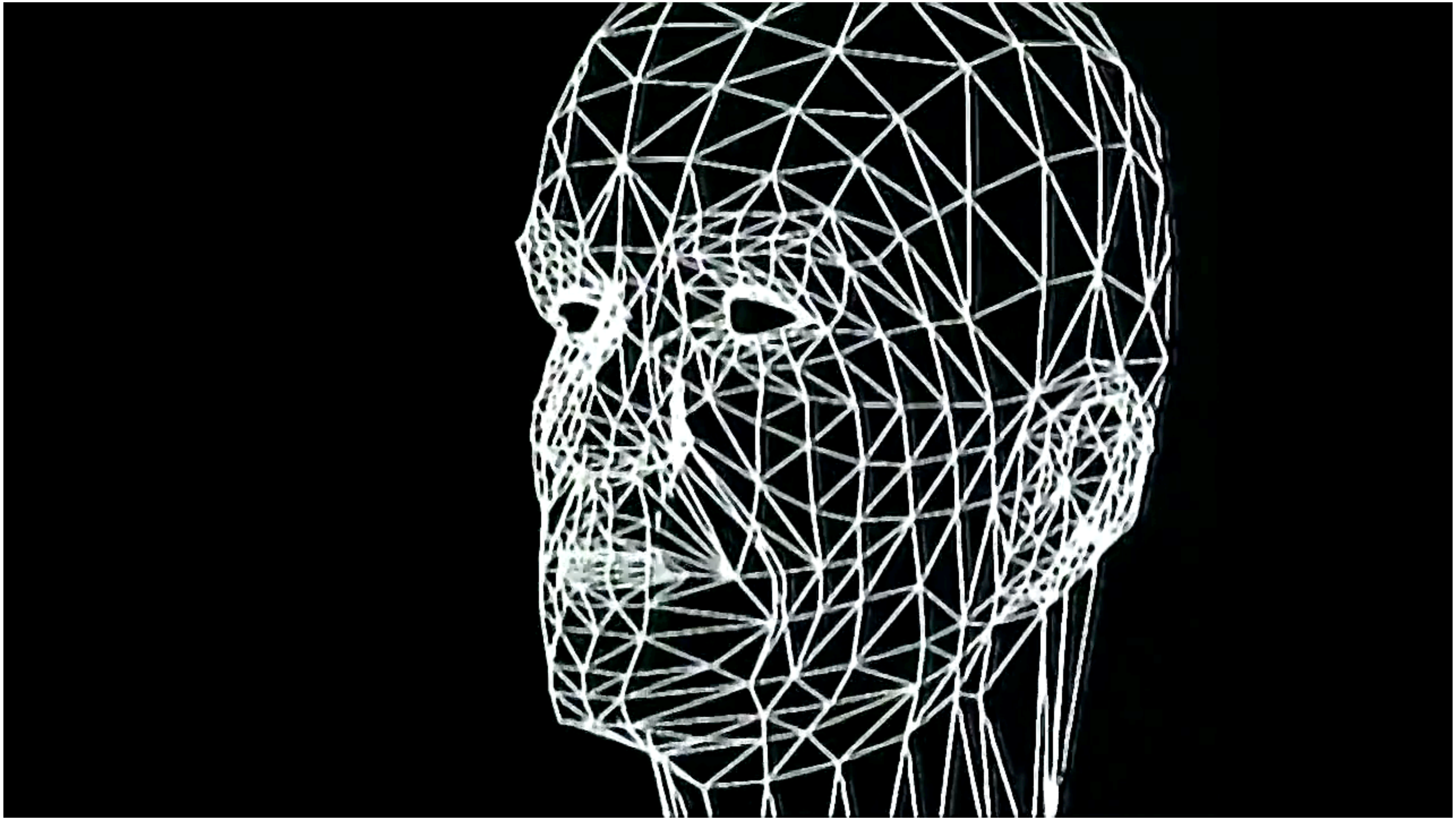


- A mesh with no border is called a closed mesh



- The set of all points V and edges E of a mesh constitute a graph, too

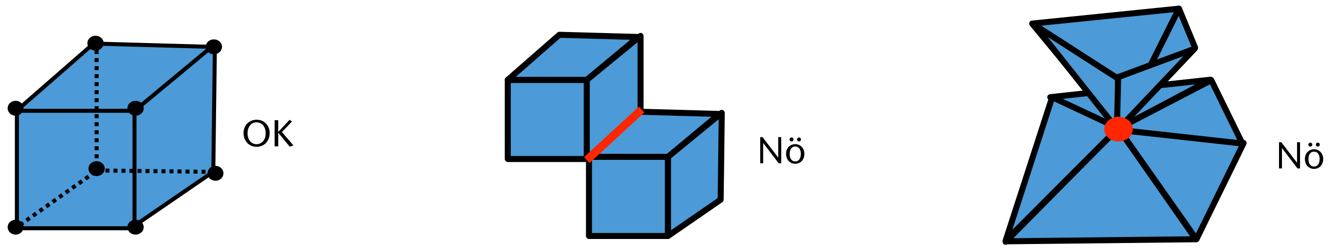
First Explicit Application of a Mesh for a Music Video



Kraftwerk: Musique non Stop, 1986. Musikvideo von Rebecca Allen.

Definition: Polyhedron

- A mesh is called polyhedron, if
  1. each edge $e \in E$ is incident to exactly two polygons (i.e., the mesh is closed); and
  2. no subset of the mesh fulfills condition (1).

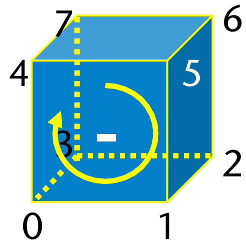


- The polygons are also called *facets / faces* (Facetten)

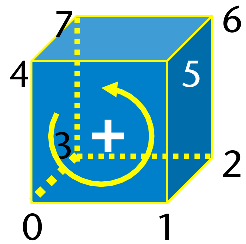
- Theorem (w/o proof):
  Each polyhedron P partitions space into three subsets: its surface, its interior, and its exterior.

Orientation

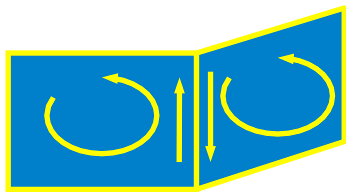
- Each facet of a mesh can be oriented by the definition of a vertex order
  - Each facet can have exactly *two* orientations



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

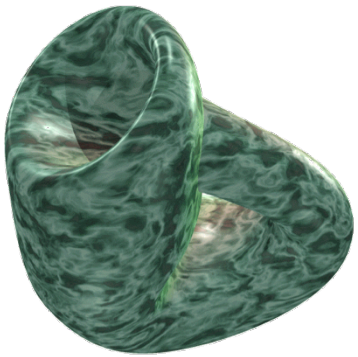


- The orientation determines the surface normal of a facet. By convention, it is obtained using the right-hand-rule



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

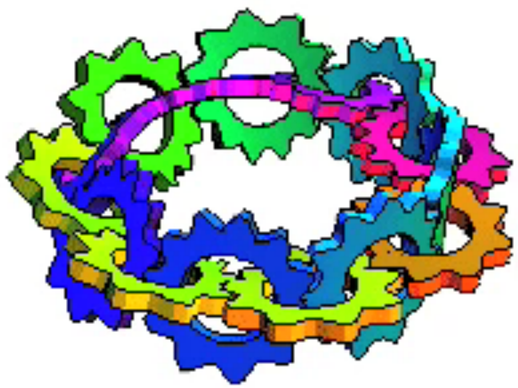
- Theorems (w/o proof):
  - Each non-orientable surface that is embedded in three-dimensional space and closed must have a self-intersection
  - The surface of a polyhedron is always orientable



Digression: the Möbius Strip in the Arts

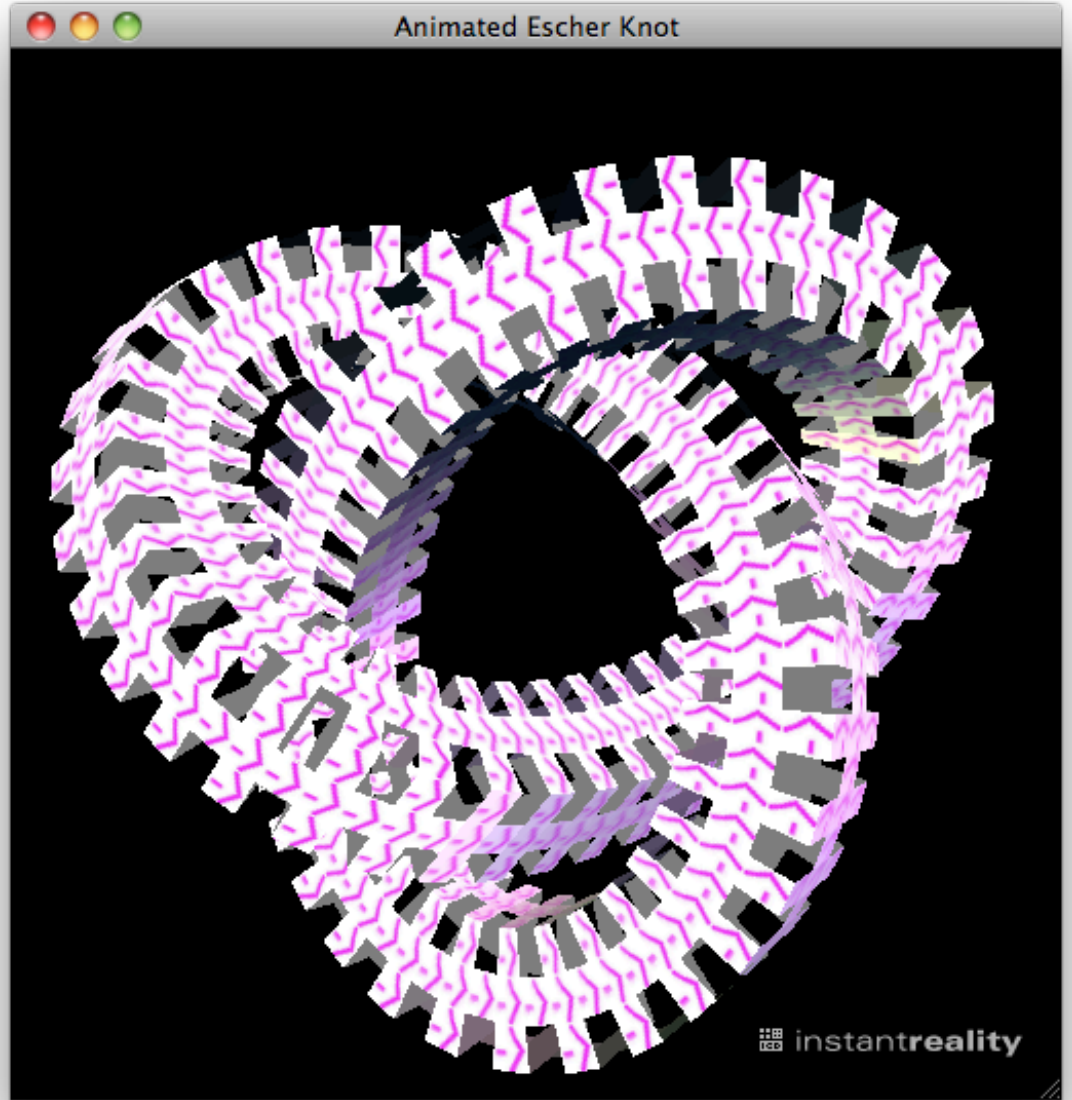
Möbius Strip II, woodcut, 1963

Max Bill



Interlocked Gears,
Michael Trott, 2001

Is the Escher Knot an Orientable Mesh or Not?

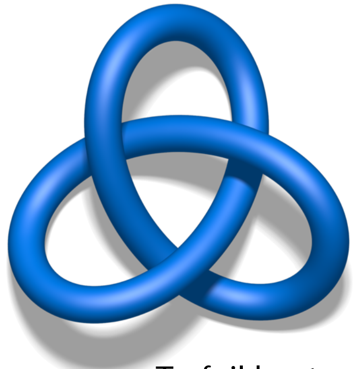


http://homepages.sover.net/~tlongtin

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
- Note: don't confuse this with *homomorphism* or *homotopy*!
- Illustration:
  - 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

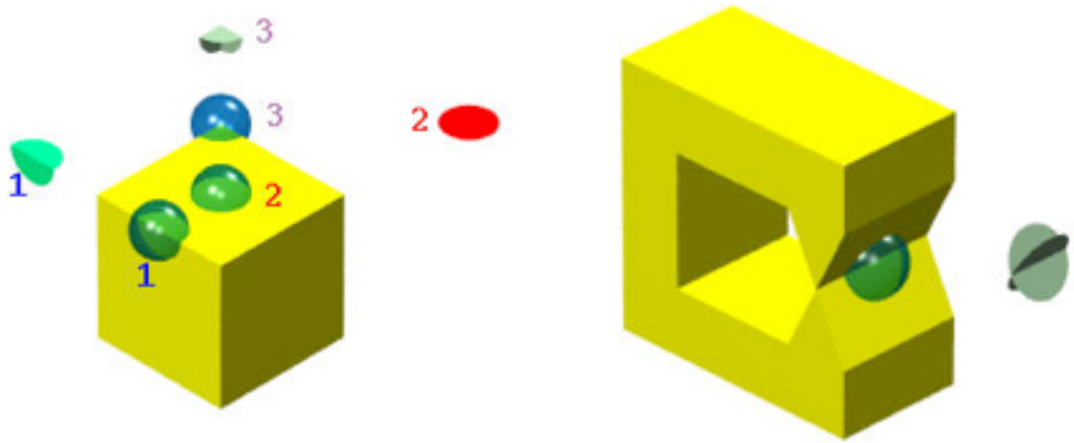
- Homeomorph objects are also called topologically equivalent
- Examples:
  - Disc and square
  - Cup and torus
  - An object and its mirror object
  - *Trefoil knot* and .... ?
  - The border of the Möbius strip and ... ?
- All *convex* polyhedra are homeomorphic to a sphere (and some non-convex ones are 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 at two-dimensional disc
- Examples:

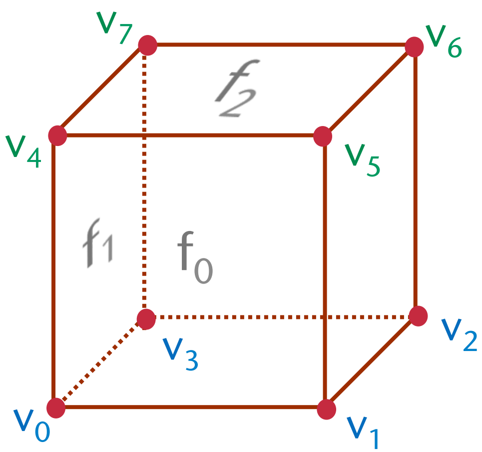


- Notice: in computer graphics, often the term "manifold" is used when 2-manifold is meant!
- The term "*piecewise linear manifold*" is sometimes used by people, to denote just a mesh …

Data Structures for Meshes

- The most naïve data structure:
  - Array of polygons; each polygon = array of vertices
  - Example:

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



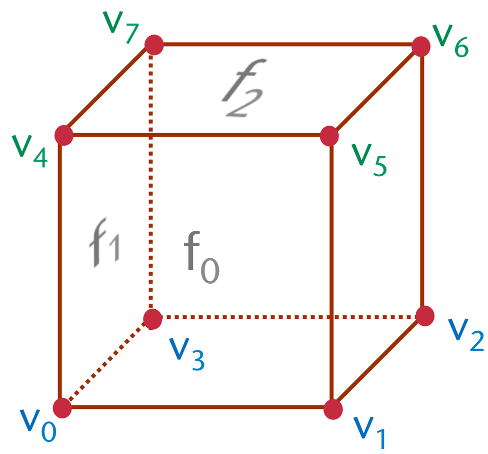
- Problems:
  - Vertices occurr several times!
    - Waste of memory, problems with animations, ...
  - How to find all faces, incident to a given vertex?
  - Different array sizes for polygons with different numbers of vertices

The Indexed Face Set

- Idea: 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
. . .	



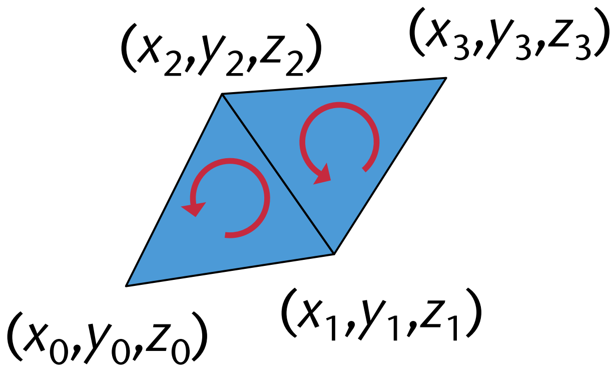
- Advantage: significant memory savings
  - 1 vertex = 1 point + 1 vector (v.-normal) + uv-texture coord. = 32 bytes
  - 1 index = 1 integer = 4 bytes
- Deformable objects / animations are mcuch easier
- Probably the most common data structure

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

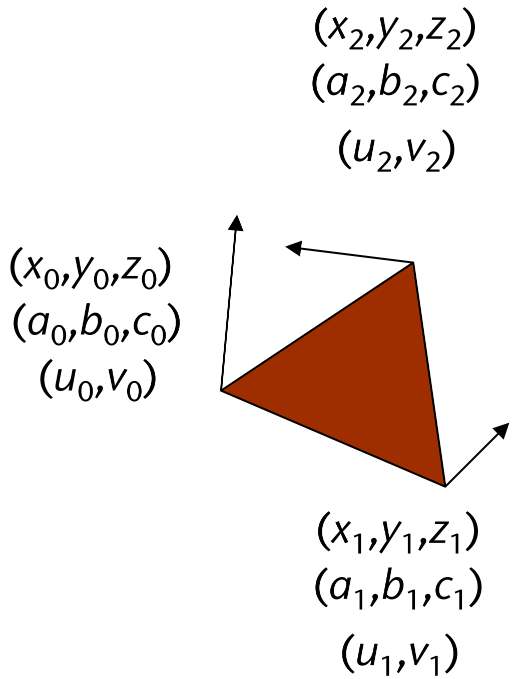


v $x_0\ y_0\ z_0$
v $x_1\ y_1\ z_1$
v $x_2\ y_2\ z_2$
v $x_3\ y_3\ z_3$

```
f 0 1 2
f 1 3 2
```

More Attributes

- Vertex normals:
  - prefix"vn"
  - contains x, y, z for the normalen
  - not necessarily normalized
  - not necessarily in the same in the same order as the vertices
  - indizes similar to vertex indices
- Texture coordinates:
  - prefix "vt"
  - not necessarily in the same 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 normls, if only texture coords are given

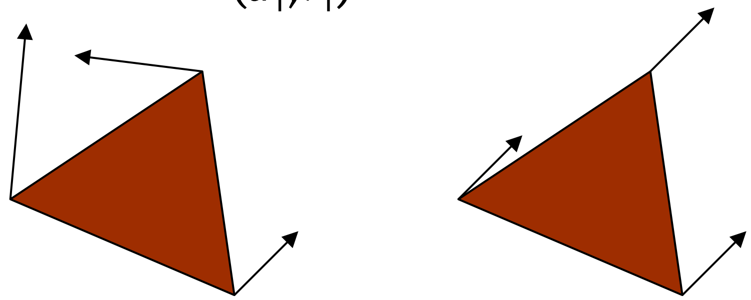


v x_0 y_0 z_0
v x_1 y_1 z_1
v x_2 y_2 z_2

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 …



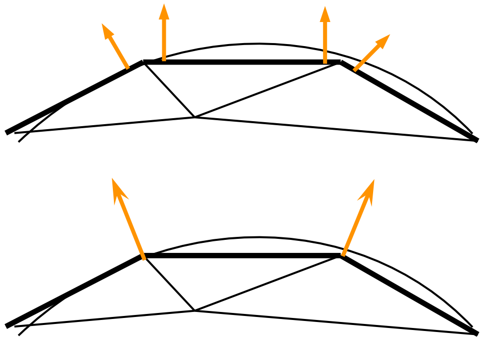
f 0/0/0 1/1/1 2/2/2

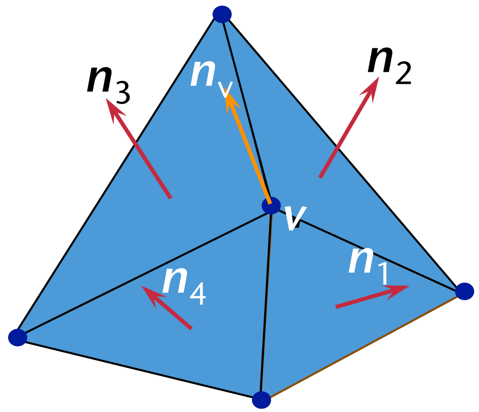
f 0/1/0 1/1/1 2/1/2

- Problems:
  - Edges are (implicitly) stored two times
  - Still no adjacency information (no "topology")
- Consequence:
  - Finding all facets incident to a given vertex takes time $O(\ n\ )$, where n = # facets of the mesh
  - Dito finding all vertices adjacent to another given vertex
  - 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 breadhth first

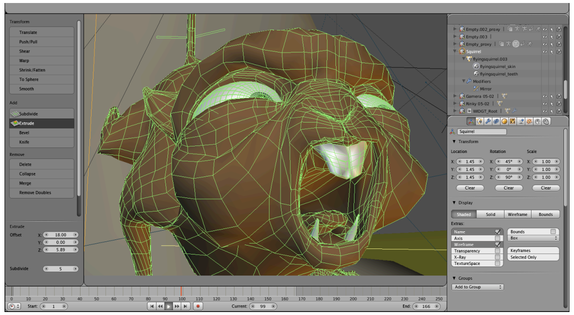
Examples Where Adjacency Information is Needed

- Computing vertex normals

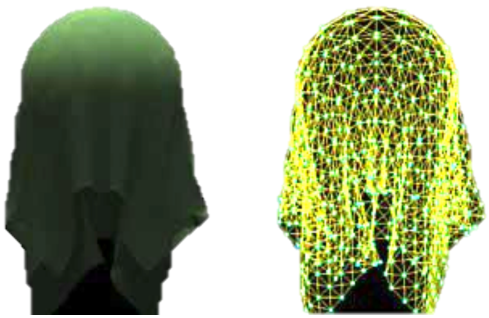




- Editing meshes

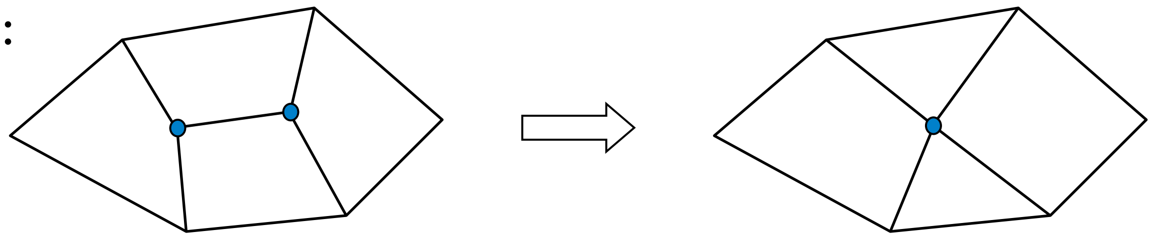


- Simulation, e.g., mass-spring systems

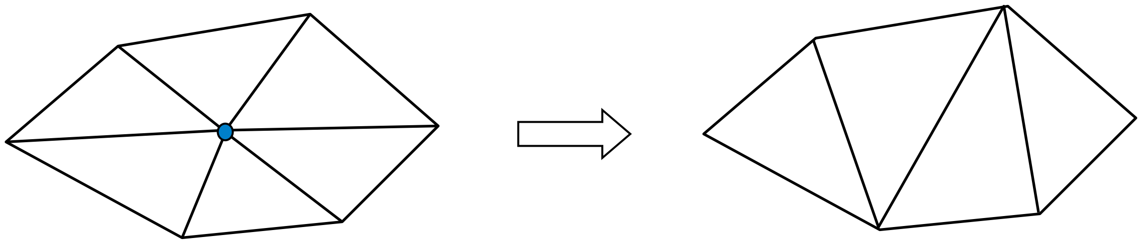


Example Application: Simplification

- Simplification: Generate a coarse mesh from a fine mesh
  - While maintaining certain critera (will not be discussed further here)
- Elementary operations:
  - Edge collapse:



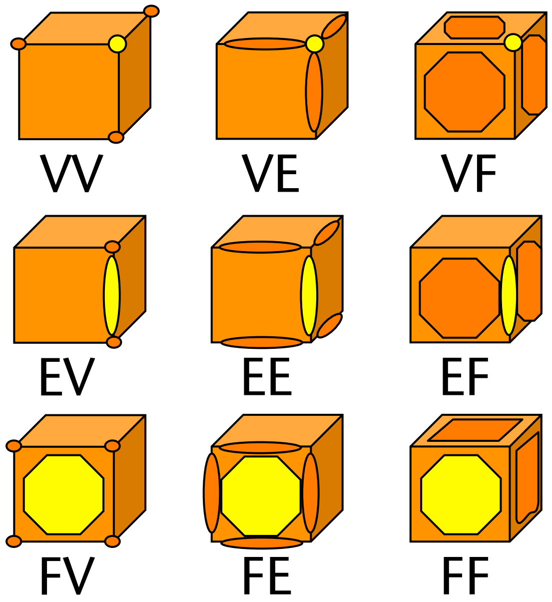
    - All edges adjacent to the edge are required
  - Vertex removal:



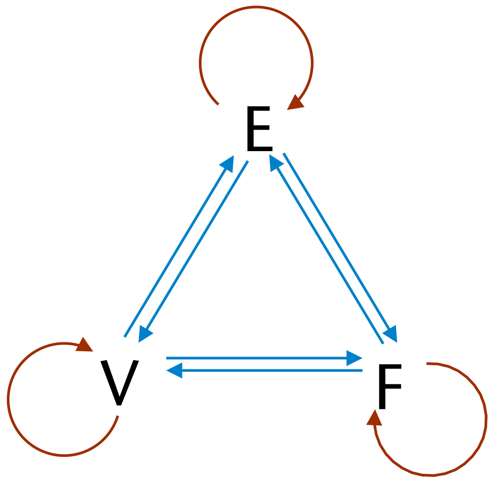
    - All edges incident to the vertex are needed

All Possible Connectivity Relationships

	Given	Looking for ("all neighbours ..")	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

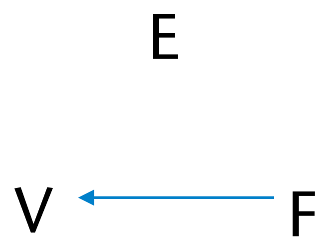


- Example: the Indexed Face Set

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
. . .	

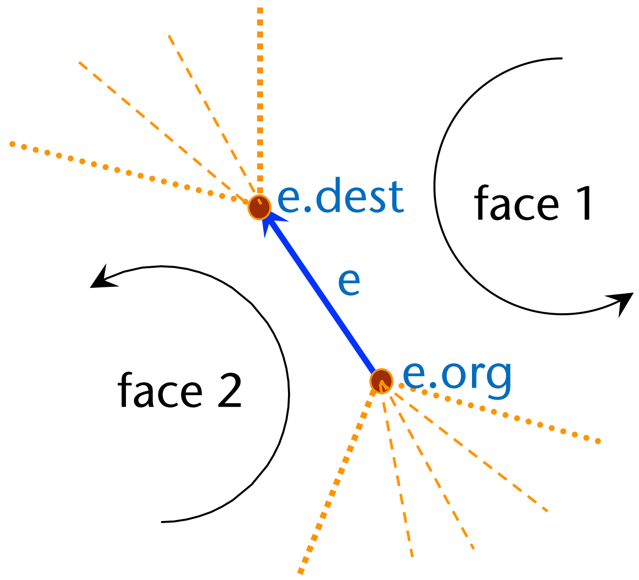
=



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

The Winged-Edge Data Structure

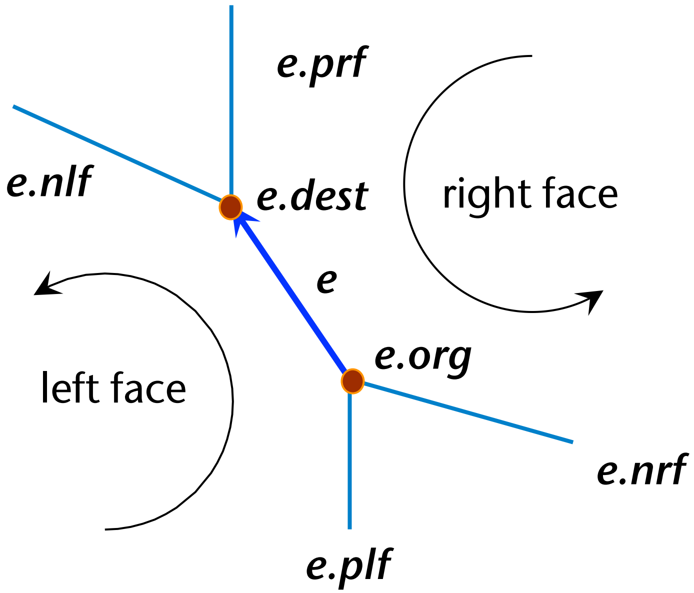
- Idea: edge-based data structure (in contrast to face-based)
- Observations:
  - An edge stores two indices to 2 vertices: ***e.org* , *e.dest***
    → yields an orientation of the edge
  - In a closed polyhedron, each edge is incident to exactly 2 facets
  - If it is oriented, then one of these facets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



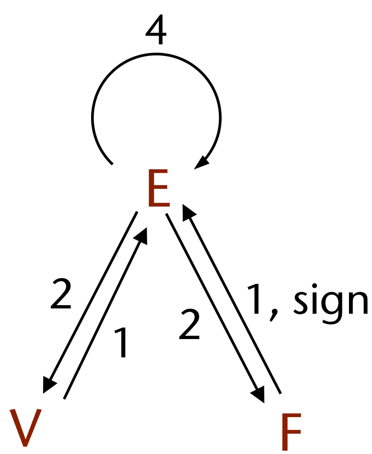
- Each edge has 4 pointers to 4 adjacent edges:
  1. **e.prf** = edge adjacent to ***e.dest*** and incident to *right face*
     (prf = "previous right face")
  2. **e.nrf** = edge adjacent to ***e.org*** and incident to *right face*
     ("next right face")

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

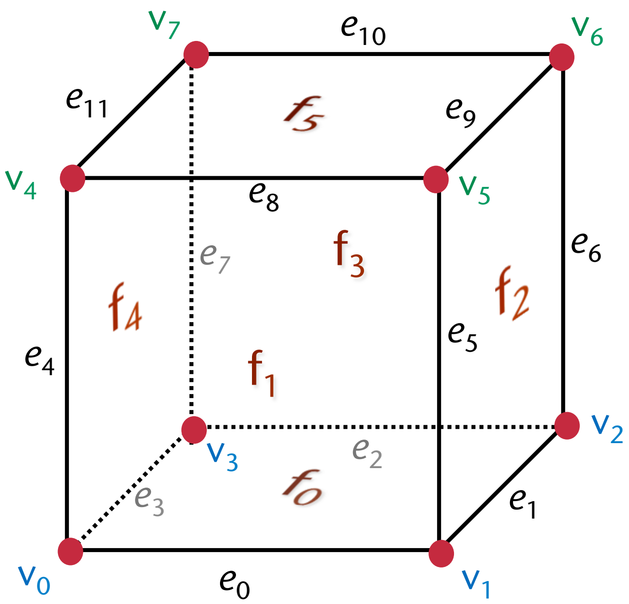
- Observation: if all facets are oriented consistently, then each edge occurs once from org⟶dest and once from dest⟶org



- 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 a arbitrary edge incident to it
- Abstract representation of the data structure:



Example



List of vertices

v	coord			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	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

Facets

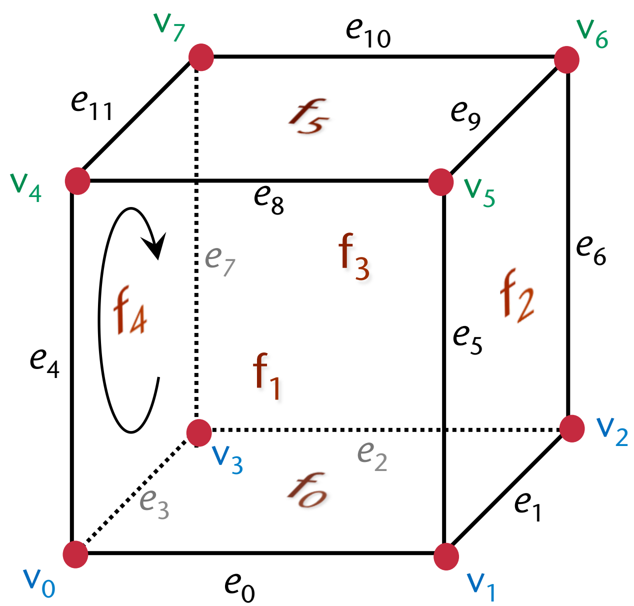
0	e0	-
1	e8	-
2	e5	-
3	e6	-
4	e11	-
5	e8	+

List of edges

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

Example for Traversing that Data Structure

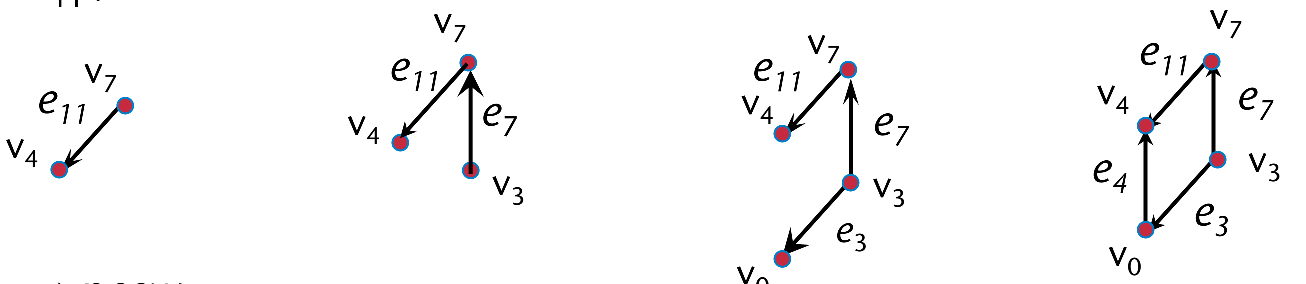
- Example task: enumerate all edges of f_4 in CCW order:



Edge list

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

$f_4 \rightarrow e_{11}$ / "-" :



→ pccw → pccw → nccw → nccw

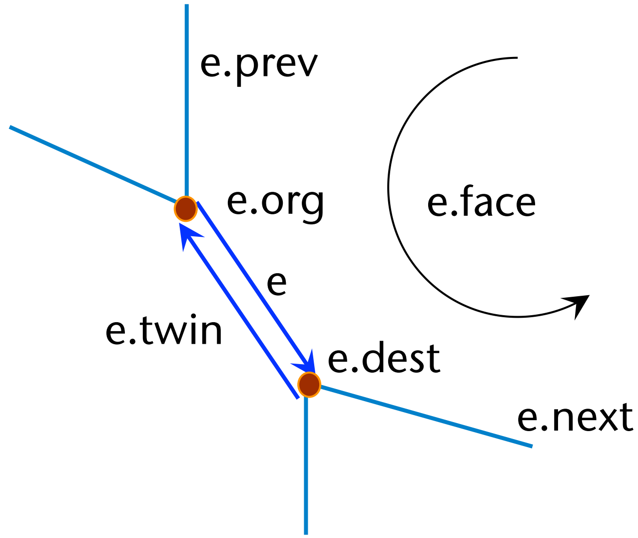
Finish

- 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 $O(k)$

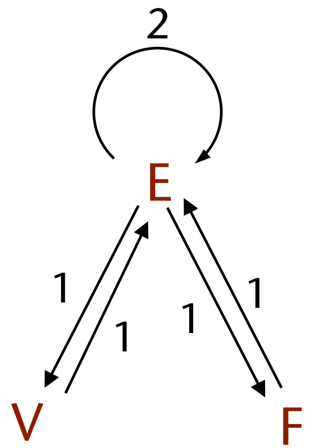
- Problem: When following edges, one has to test for each edge *how* it is oriented, in order to determine whether to follow ***n[c]cw*** or ***p[c]cw***!

Doubly Connected Edge List [Preparata & Müller, 1978]

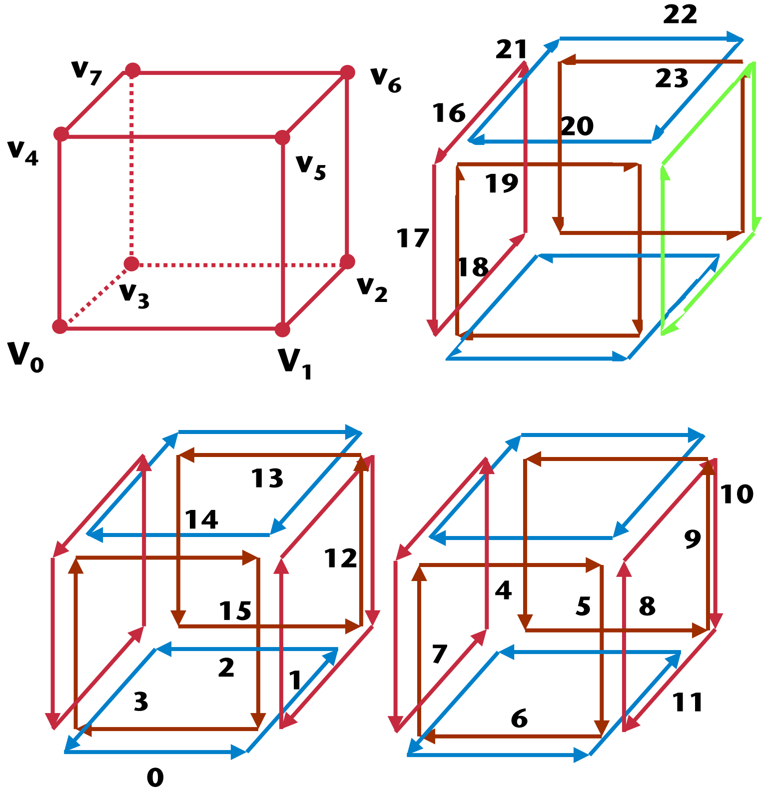
- In computer graphics rather known as *"half-edge data structure"*
- Arguably the easiest and most efficient neighborhood data structure
- Idea:
  - Like the winged-wdge DS, but with "split" edges
  - One half-edge (= entry in the edge table) represents only one direction and one "side" of the complete edge
  - The pointers stored with each half-edge:
    - Start (**org**) and end vertex (**dest**)
    - Incident **face** (on the left-hand side)
    - **Next** und **prev**ious edge (in traversal order)
    - (Originating vertex can be omitted, because e.org = e.twin.dest)



- Abstract notation:
  - Here without pointer to originating vertex (org)
  - Requires twice as many entries in the edge table as the winged-edge DS



Example (Here in CW Order!)



List of Vertices

v	coord			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

Facets

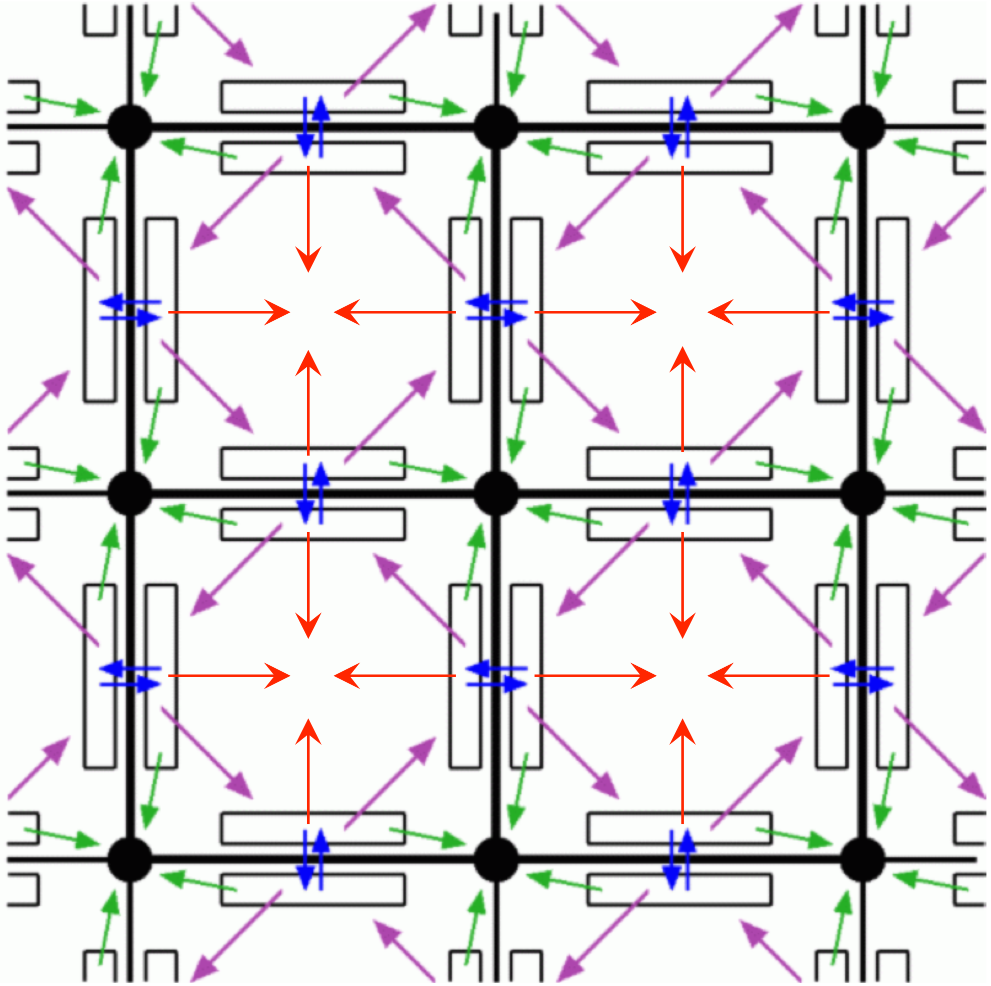
0	e20
1	e4
2	e0
3	e15
4	e16
5	e8

List of Half-Edges

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

Also note the demo on
http://www.holmes3d.net/graphics/dcel/

- Visualization for a quad mesh:



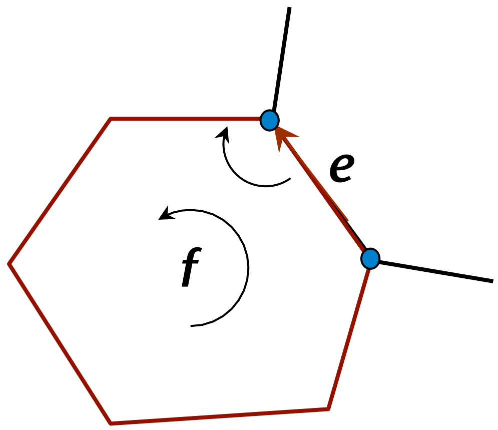
Invariants in a DCEL

- Here, we will use the "functional notation", i.e.,
  twin(e) = e.twin
- Invariants (= axioms in an ADT "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   [v 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:

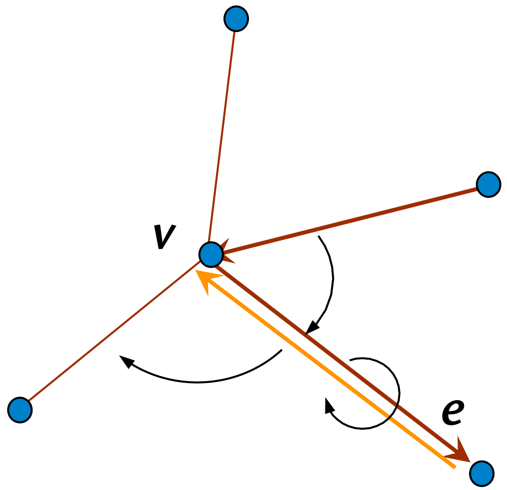
```
e_start ← f.edge
e ← e_start
repeat
  output e.dest
  e ← e.next
until e == e_start
```



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

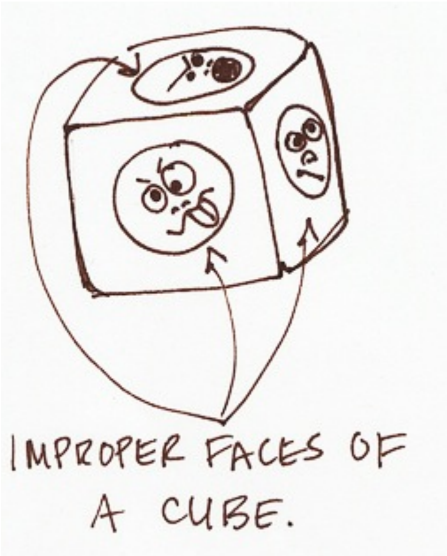
- Task: report all vertices adjacent to a given vertex ***v***
- Algorithm (w.l.o.g., ***v*** points to a leaving edge):

```
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***

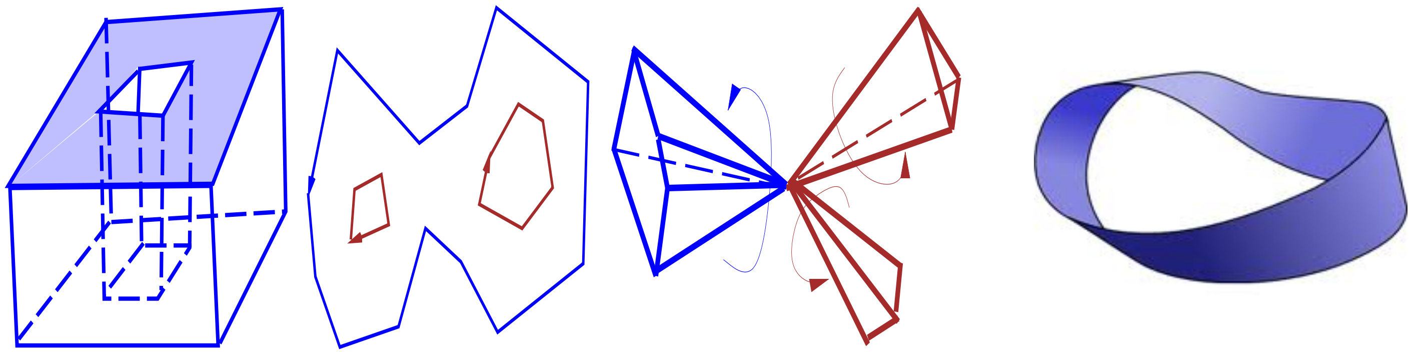
- Terminology: a feature = a vertex or an edge or a facet
- Theorem:
  A DCEL over a 2-manifold mesh supports all incidence and adjacency queries for a given feature in time $O(1)$ or $O(k)$, where k = # neighbours.



Courtney Gibbons 2007

Limitations / Extensions of the DCEL

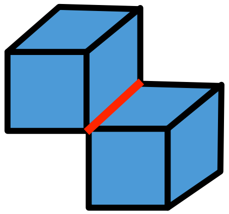
- A DCEL can store only meshes that are ...
  1. two-manifold and
  2. orientable, and
  3. the polygons of which do not have "holes"!

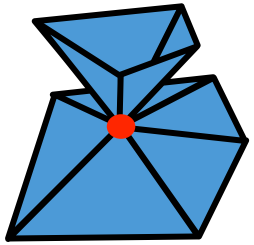


- Extensions: lots of them, e.g. those of Hervé Brönnimann
  - For non-2-manifold vertices, store several pointers to incident edges
  - Dito for facets with holes
  - Yields several cycles of edges for such vertices/faces

A DCEL Data Structure for Non-2-Manifolds

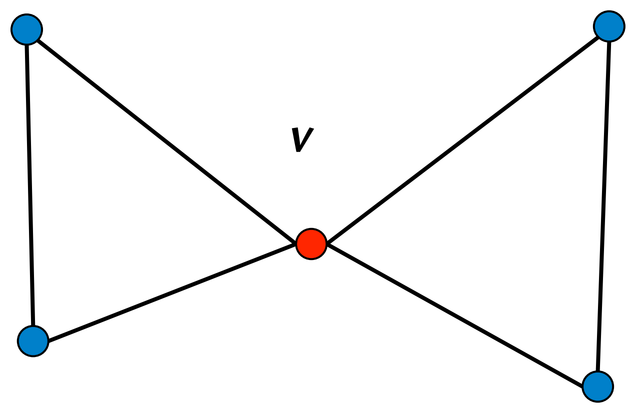
- Directed Edge DS: extension of half-edge DS for meshes that are not 2-manifold at just a few extraordinary places





- Idea:
  - Store pointers to other edges (e.next, e.prev, v.edge, f.edge) as integer indices into the edge array
  - Use the *sign* of the index as a flag for additonal information
  - Interpret negative indices as pointers into additonal arrays, e.g.,
    - a list of all edges eminating 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 ...

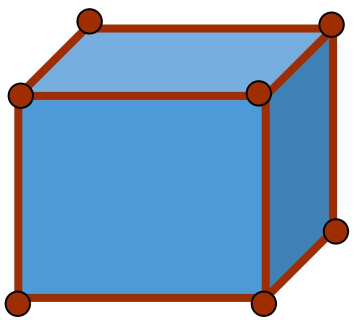
The Euler Equation

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

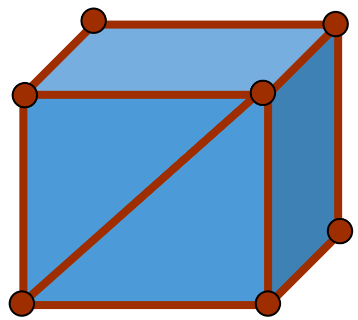
  Then,

$$V - E + F = 2$$

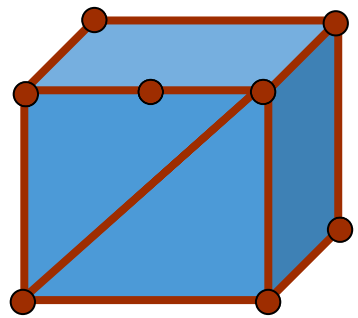
- Examples:



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



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



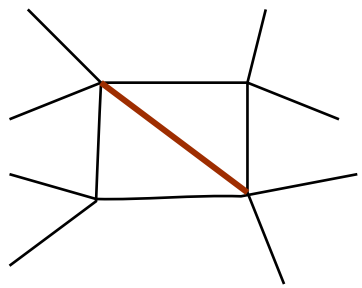
$V = 8+1$
$E = 12+1+1$
$F = 6+1$

Proof (given by Cauchy)

- Given: a closed mesh (Polyhedron)
- 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:

$$V - E + F = 1$$

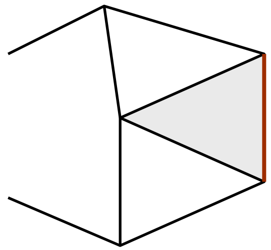
- Second Idea: triangulate the graph (i.e., the mesh)
  - Draw diagonals in all facets with more than 3 vertices
  - For the new feature count we have



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

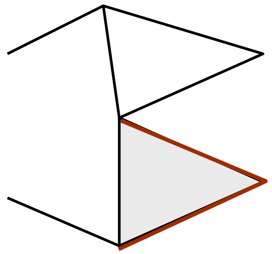
- The graph has a border; triangles have 0, 1, or 2 "border edges"
- Repeat one of the following two transformations:
  - If there is a triangle with exactly one border edge, remove this triangle ; it follows that

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



  - If there is a triangle with exactly two border edges, remove the triangle ; it follows that

$$V' - E' + F' = (V-1) - (E-2) + (F-1) = V - E + F$$

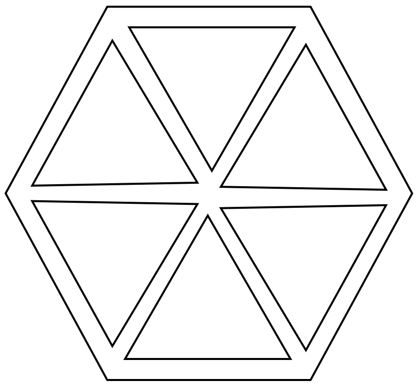


- Repeat, until only one triangle remains
  - For that triangle, the Euler equation is obviously correct
  - Because each of the above transformations did not change the value of *V-E+F*, the equation is also true for the original graph, hence for the original mesh

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

$$F \approx 2V$$

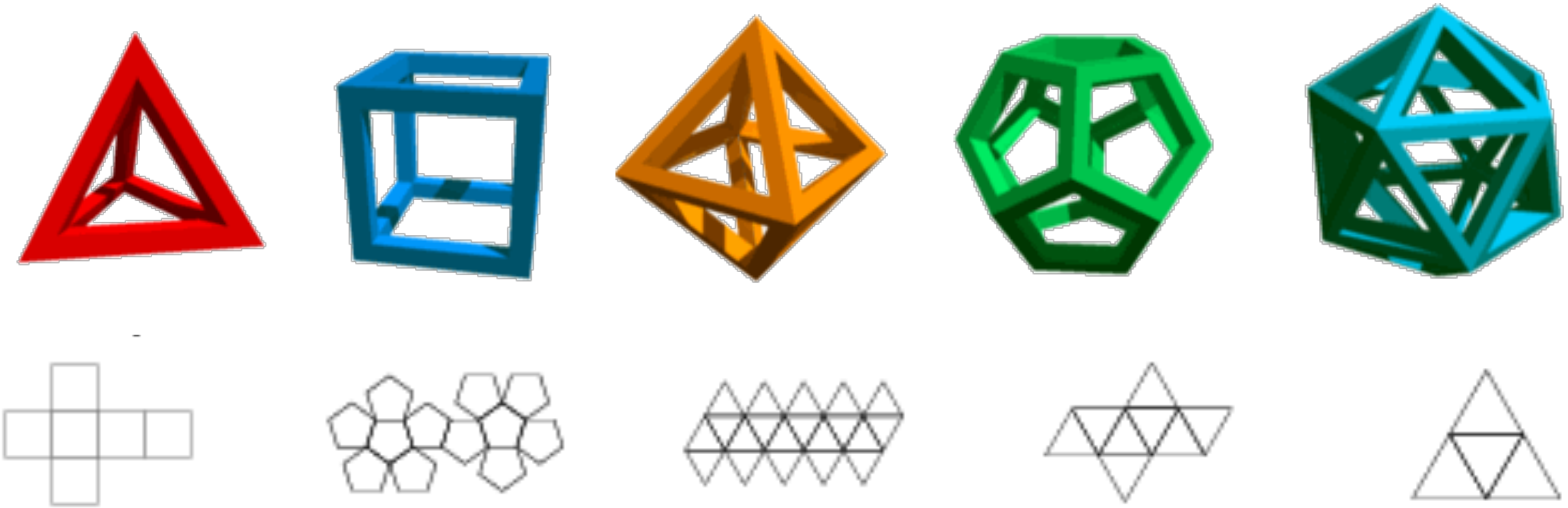
Application of Euler's Equation to the Platonic Solids

- Definition Platonic Solid:

  a convex polyhedron, consisting of a number of congruent regular polyhedra

- Theorem (Euklid):

  There are exactly *five* platonic solids.



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:

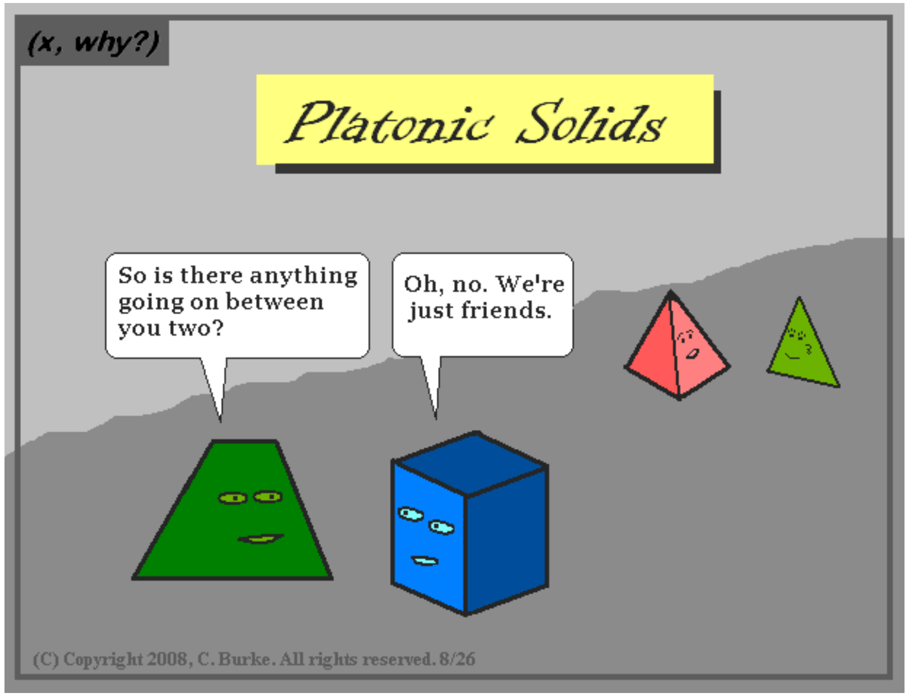
$$2 = V - E + F = \frac{2}{m}E - E + \frac{2}{n}E \Leftrightarrow \frac{2}{E} = \frac{2}{m} - 1 + \frac{2}{n}$$

- 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:

  $\{3,3\}$ $\{3,4\}$ $\{4,3\}$ $\{5,3\}$ $\{3,5\}$



Digression: Platonic Solids in the Arts

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

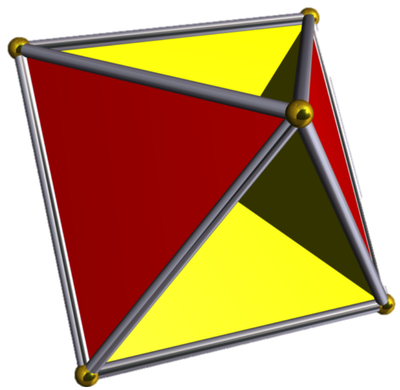
Portrait of Johannes Neudörfer and his Son
Nicolas Neufchatel, 1527—1590

Dürer: *Melencolia I*

The Euler Characteristic

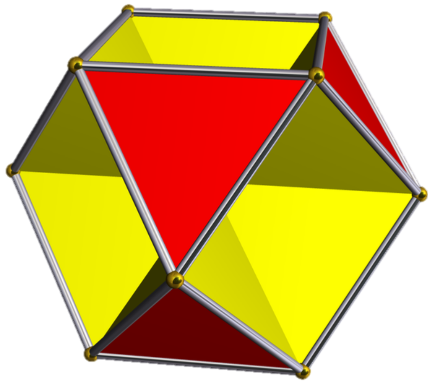
- Caution: the Euler equation holds only for polyhedra, that are topologically equivalent to a sphere!
- Examples:

Tetrahemihexahedron



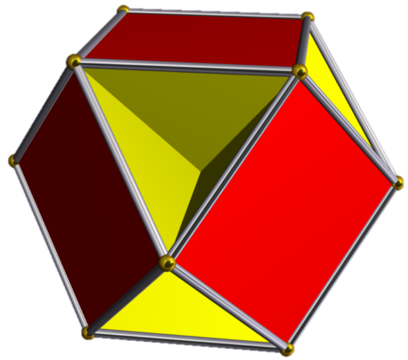
6 - 12 + 7 = 1

Octahemioctahedron



12 - 24 + 12 = 0

Cubohemioctahedron



12 - 24 + 10 = -2

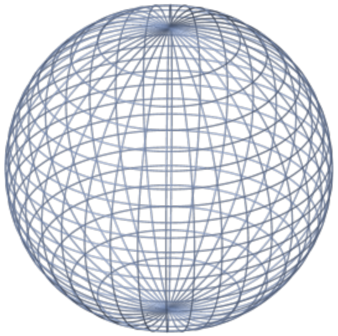
V-E+F

- But: the quantity *V-E+F* stays the same no matter how the polyhedron is deformed (homeomorph)
  → so the quantity *V-E+F* is a topologic invariant

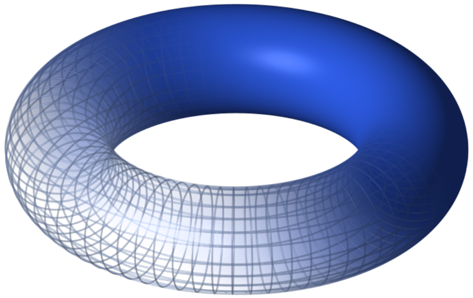
- Definition Euler characteristic:

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

- Examples:

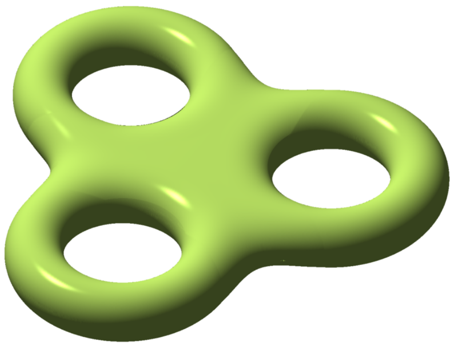


2

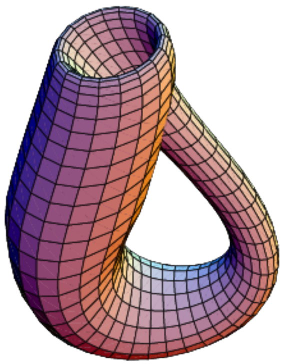


0

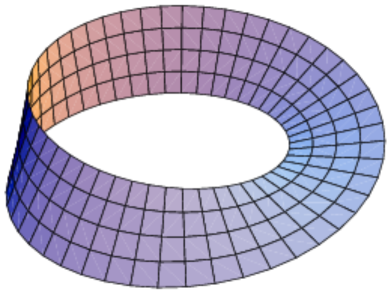
-2



-4

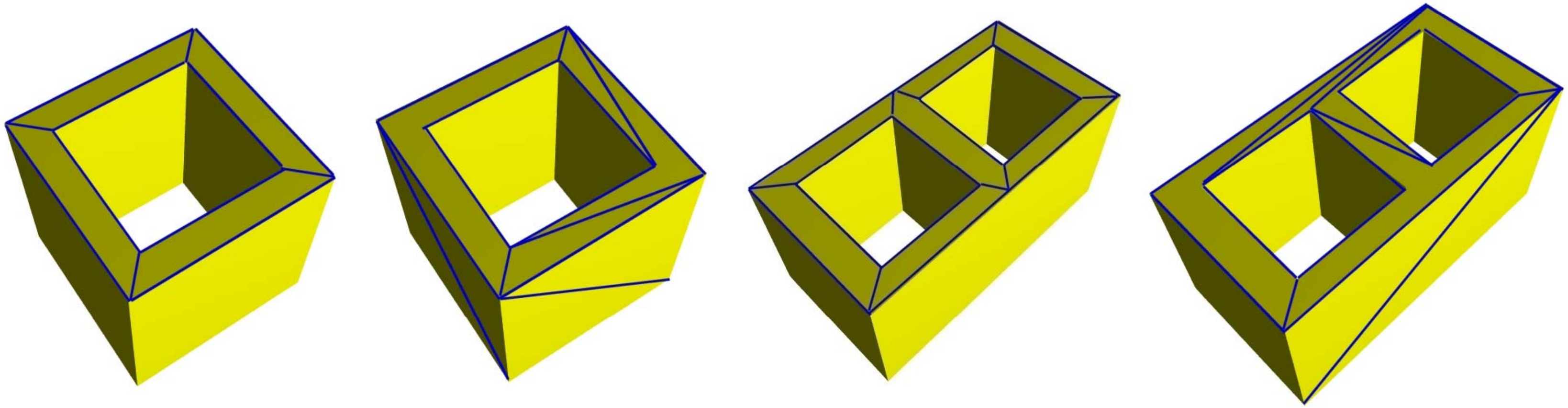


0



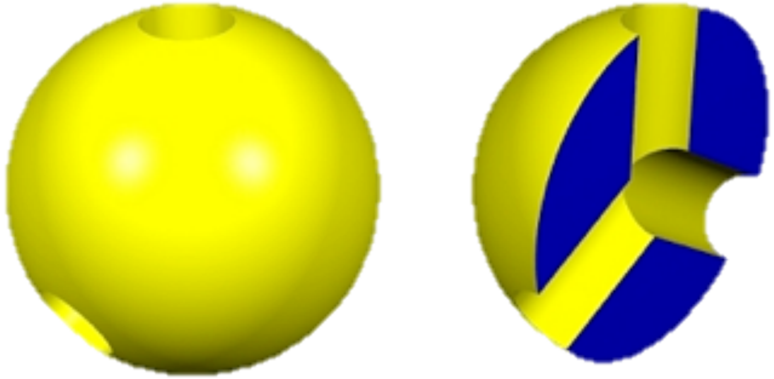
0

- The Euler characteristic is even independent of the tessellation!

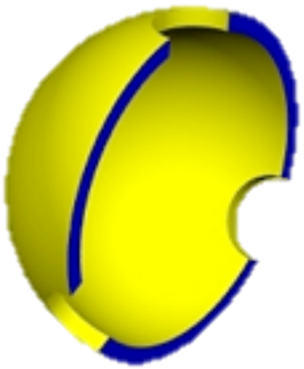


V = 16	V = 16	V = 28	V = 24
E = 32	E = 36	E = 56	E = 48
F = 16	F = 20	F = 26	F = 22
$\chi = 0$	$\chi = 0$	$\chi = -2$	$\chi = -2$

- Beware: sometimes it is not easy to determine the genus!
- Example: genus = 2



  - "Proof": deform topologically equivalently, until the genus is obvious



1.

2.

3.

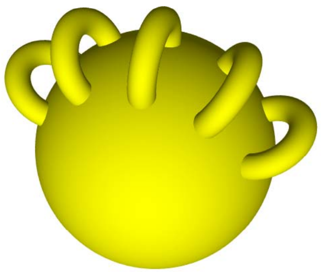
- What is the genus of this object?

The Euler-Poincaré Equation

- Generalization of the Euler equation for 2-manifold closed surfaces (possibly with several components):

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

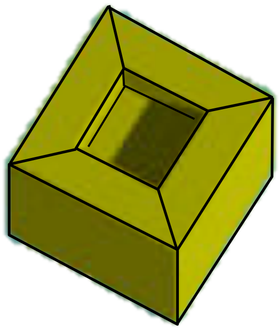
  - G = # handles, S = # shells (Schalen / Komponenten)
  - G is called "Genus"
  - Handle (hole, Loch): a piece of string inside a handle cannot be shrunk towards a single point
  - Shell (Schale): by walking on the surface of a *shell*, each point can be reached
  - We can even cut out so-called "voids" (Aushöhlungen) by "inner" shells



- There are many more generalizations!

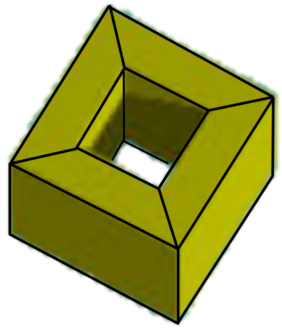
- Examples:
  - $V = 16,\ E = 28,\ F = 14,\ S = 1,\ G = 0$:

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



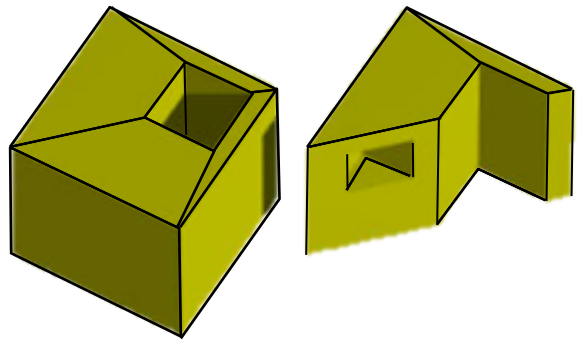
  - $V = 16,\ E = 32,\ F = 16,\ S = 1,\ G = 1$:

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



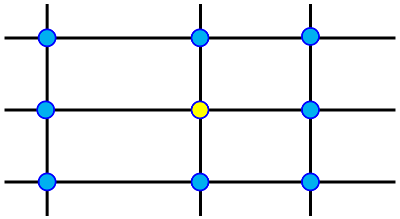
  - $V = 16{+}8,\ E = 32{+}12,\ F = 16{+}6,\ G = 1,\ S = 2$:

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

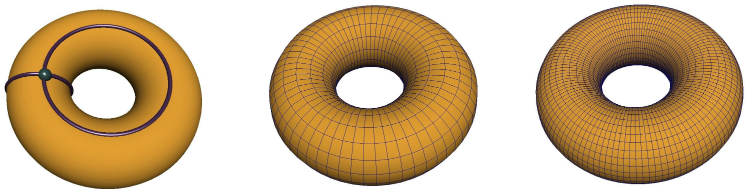


- Theorem:
  Assume we are given a *closed* and *orientable* mesh consisting of just one *shell*. Then the following holds:
  The Euler characteristic $\chi = 2, 0, -2, \ldots$ ⇔
  the mesh is topologically equivalent to a sphere, a torus, a double torus, etc. …

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



- Application 1:
  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 $\chi(M) = V - E + F = 0 \longrightarrow M =$ 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

$$nf = 2e = mv \tag{1}$$

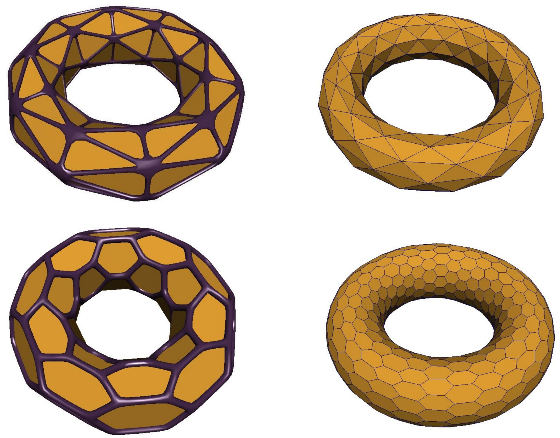
- Plugging that into the Euler equation, we obtain

$$\left(\frac{1}{n} + \frac{1}{m} - \frac{1}{2}\right) e = 1 - g. \tag{2}$$

- 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 $nm - 2n - 2m > 0$.
- Proof:
  - Rewrite equations (1) and (2)

$$e = \frac{2nm}{nm - 2n - 2m}(g-1)$$

$$f = \frac{4m}{nm - 2n - 2m}(g-1)$$

$$v = \frac{4n}{nm - 2n - 2m}(g-1)$$

- 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: a (4,5,2)-mesh