252-0538-00L, Spring 2025

Shape Modeling and Geometry Processing

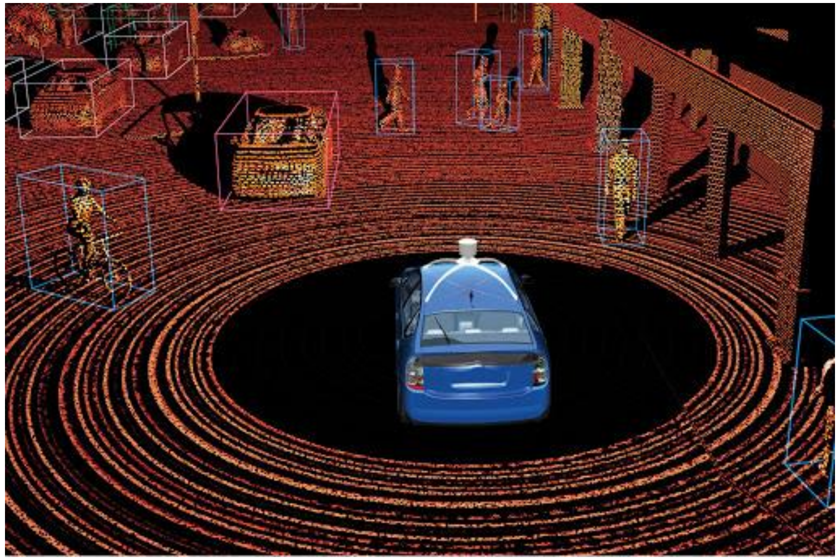
Geometry Acquisition
Meshes

INTERACTIVE GEOMETRY LAB

February 26, 2025

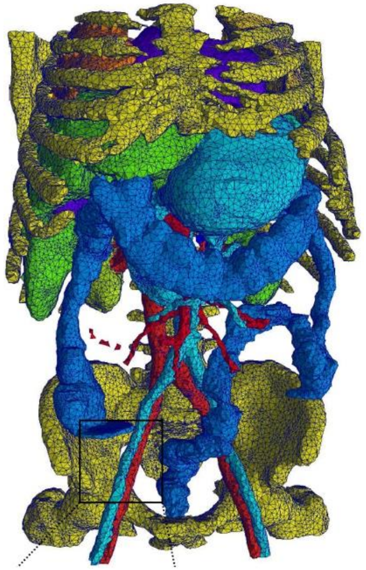
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Geometry Acquisition is Everywhere



By Google

By Elysium Co. Ltd.

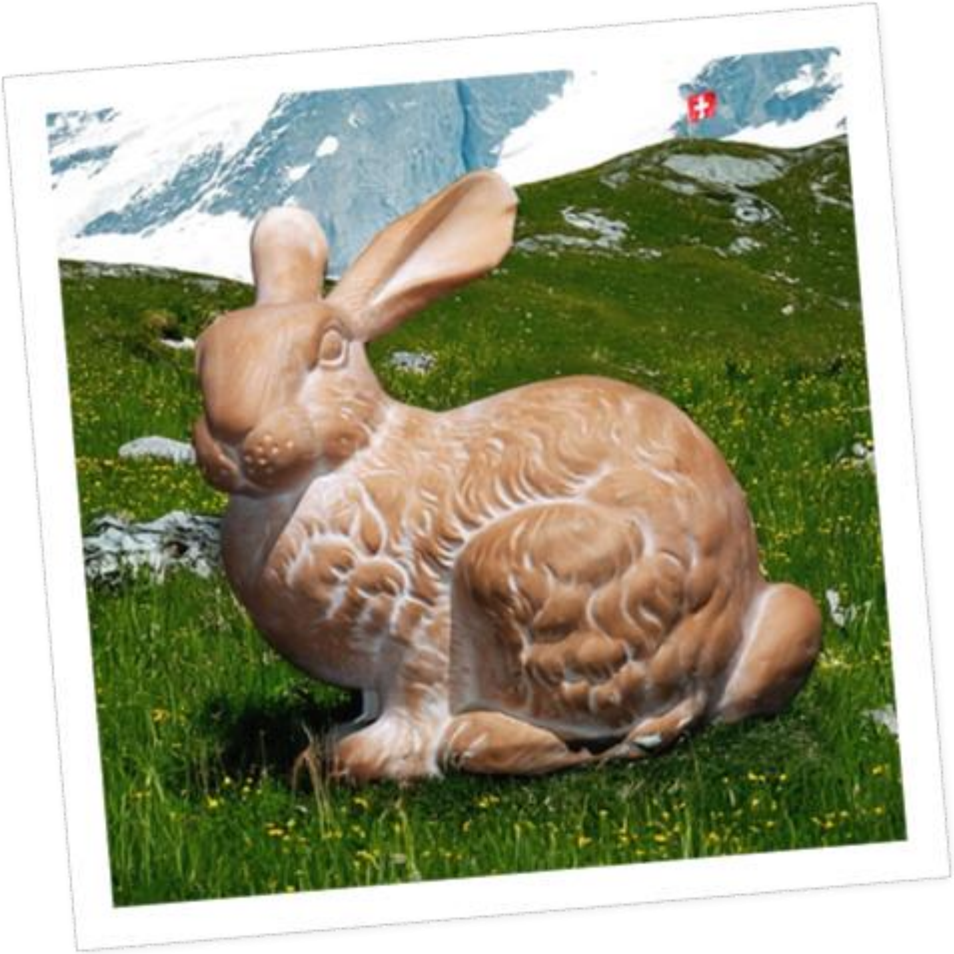


By MeshMed 2012

Goal: low-cost, fast, accurate, dense


80
80
1.5km
80
km/h
80

From physical to digital



Geometry Acquisition Pipeline

Scanning
results in range images

⇨

Registration
bring all range images to one coordinate system

⇨

Stitching/reconstruction
Integration of scans into a single mesh

⇨

Postprocess
Topological filtering
Geometric filtering
Remeshing
Compression

Geometry Acquisition Pipeline

Scanning
results in range images

⇨

Registration
bring all range images to one coordinate system

⇨

Stitching/reconstruction
Integration of scans into a single mesh

⇨

Postprocess
Topological filtering
Geometric filtering
Remeshing
Compression

Geometry Acquisition Pipeline

Scanning
results in range images

⇨

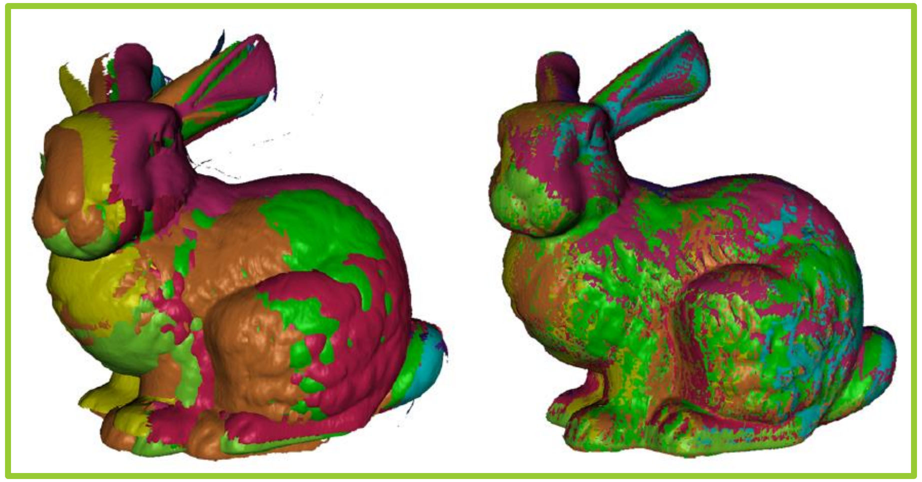
Registration
bring all range images to one coordinate system

⇨

Stitching/reconstruction
Integration of scans into a single mesh

⇨

Postprocess
Topological filtering
Geometric filtering
Remeshing
Compression



Geometry Acquisition Pipeline

Scanning
results in range images

⇨

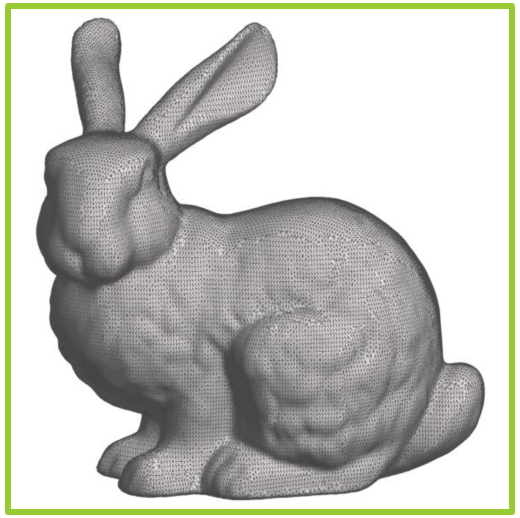
Registration
bring all range images to one coordinate system

⇨

Stitching/reconstruction
Integration of scans into a single mesh

⇨

Postprocess
Topological filtering
Geometric filtering
Remeshing
Compression



Geometry Acquisition Pipeline

Scanning
results in range images

⇨

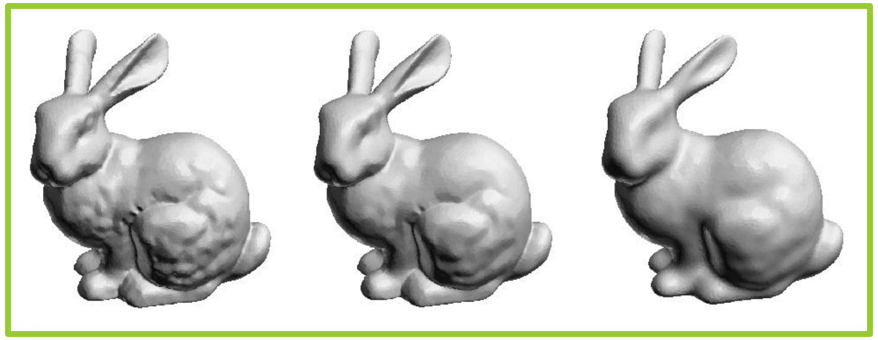
Registration
bring all range images to one coordinate system

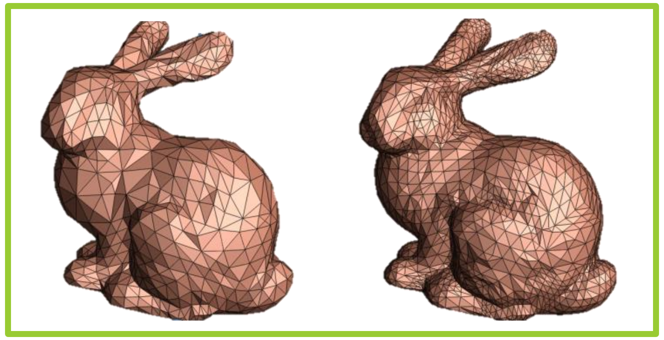
⇨

Stitching/reconstruction
Integration of scans into a single mesh

⇨

Postprocess
Topological filtering
Geometric filtering
Remeshing
Compression





Geometry Acquisition Pipeline

Scanning
results in range images

⇨

Registration
bring all range images to one coordinate system

⇨

Stitching/reconstruction
Integration of scans into a single mesh

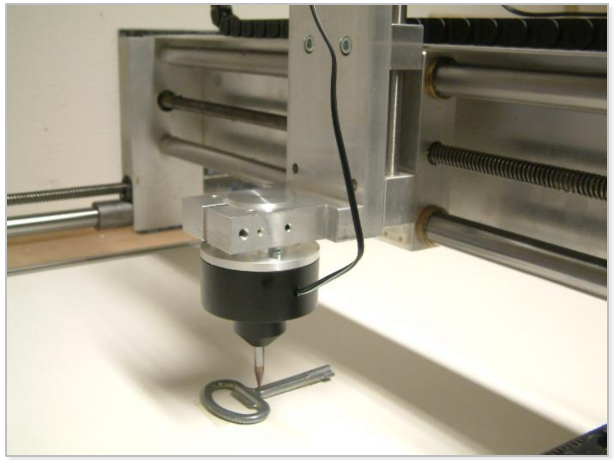
⇨

Postprocess
Topological filtering
Geometric filtering
Remeshing
Compression

Touch Probes

Touch Probes (Contact-based)

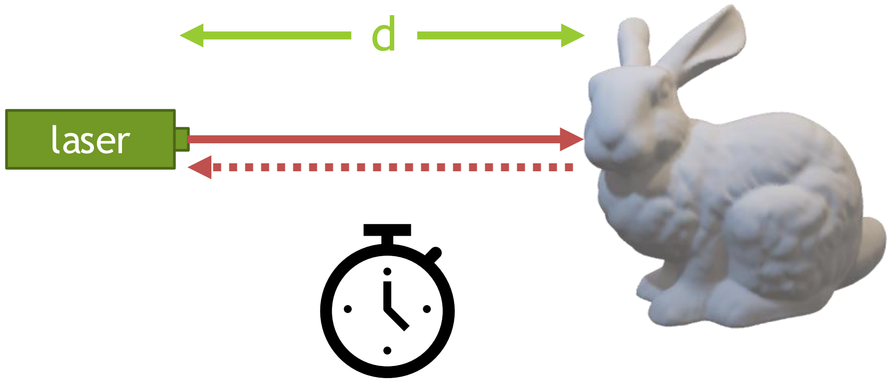
- Physical contact with the object
- Manual or computer-guided
- Advantages:
  - Can be **very precise**
  - Can scan **any** solid surface
- Disadvantages:
  - Slow, small scale
  - Can’t use on fragile objects



Optical Scanning

- Infer the geometry from light reflectance
- Advantages:
  - Less invasive than touch
  - Fast, large scale possible
- Disadvantages:
  - Difficulty with transparent, fuzzy and shiny objects

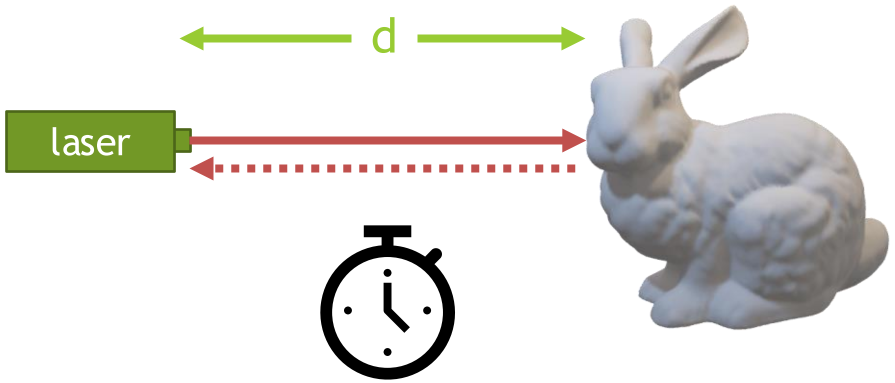
Optical scanning – active lighting

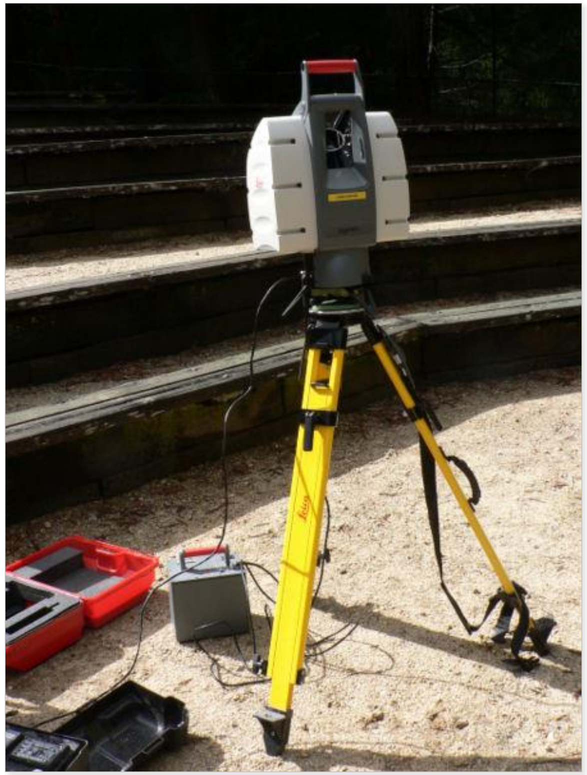


Time of flight laser

Optical scanning – active lighting

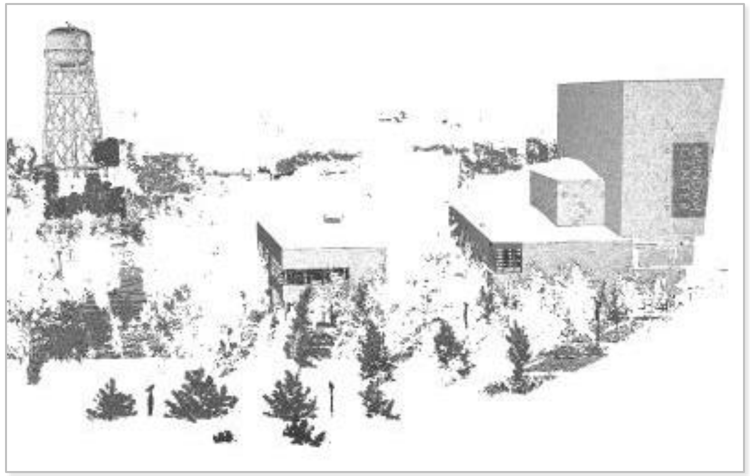
- A type of laser pulse-based rangefinder (LIDAR)
- Measures the time it takes the laser beam to hit the object and come back

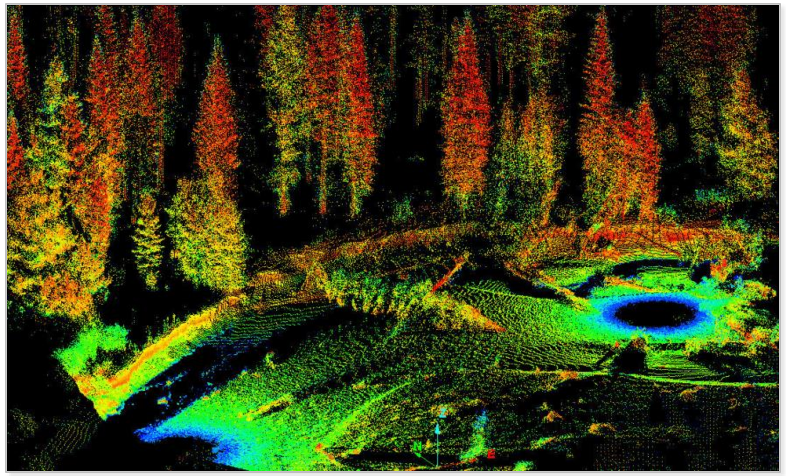




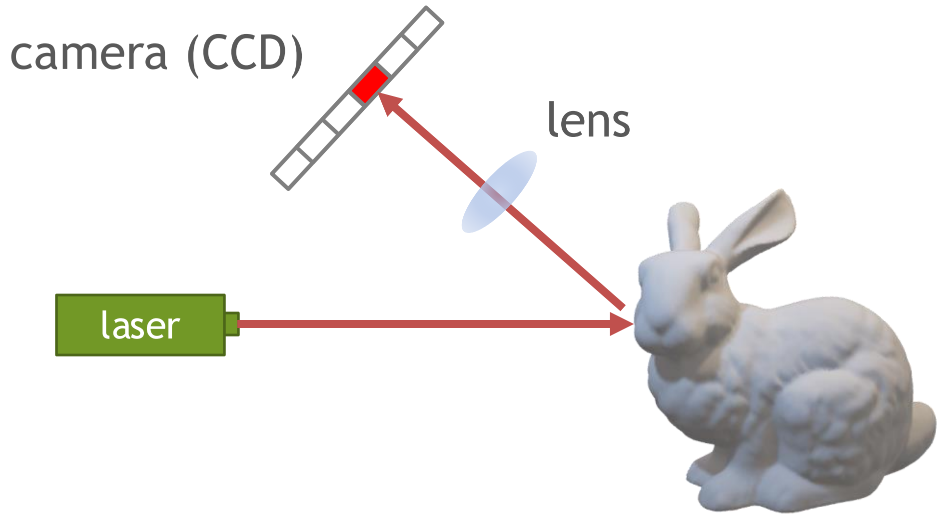
Optical scanning – active lighting

- Accommodates large range – up to several miles (suitable for buildings, rocks)
- Lower accuracy in large range
  - objects move while scanning



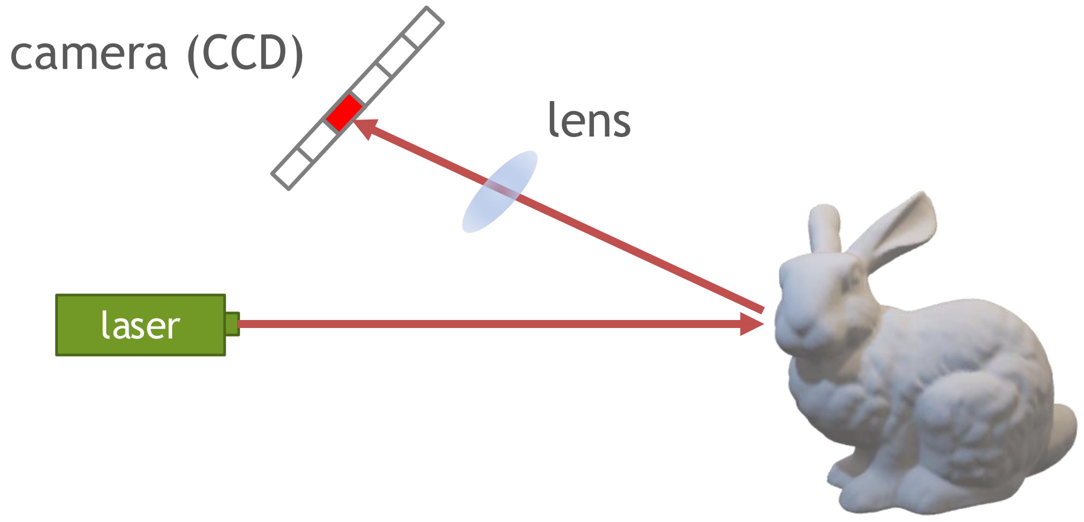


Optical scanning – active lighting



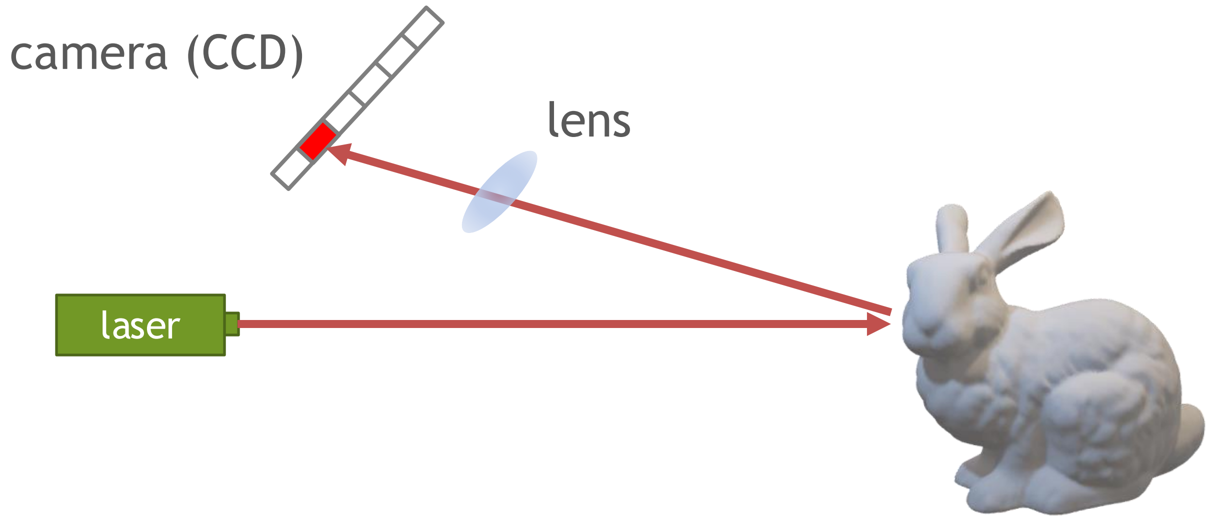
Triangulation laser

Optical scanning – active lighting



Triangulation laser

Optical scanning – active lighting

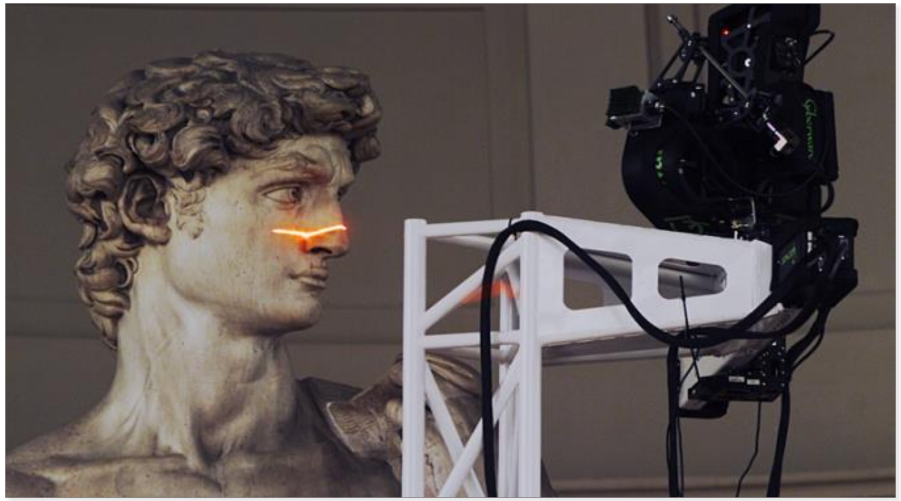


Triangulation laser

Optical scanning – active lighting

Triangulation laser

- Laser beam and camera
- Laser dot is photographed
- The location of the dot in the image allows triangulation: we get the distance to the object



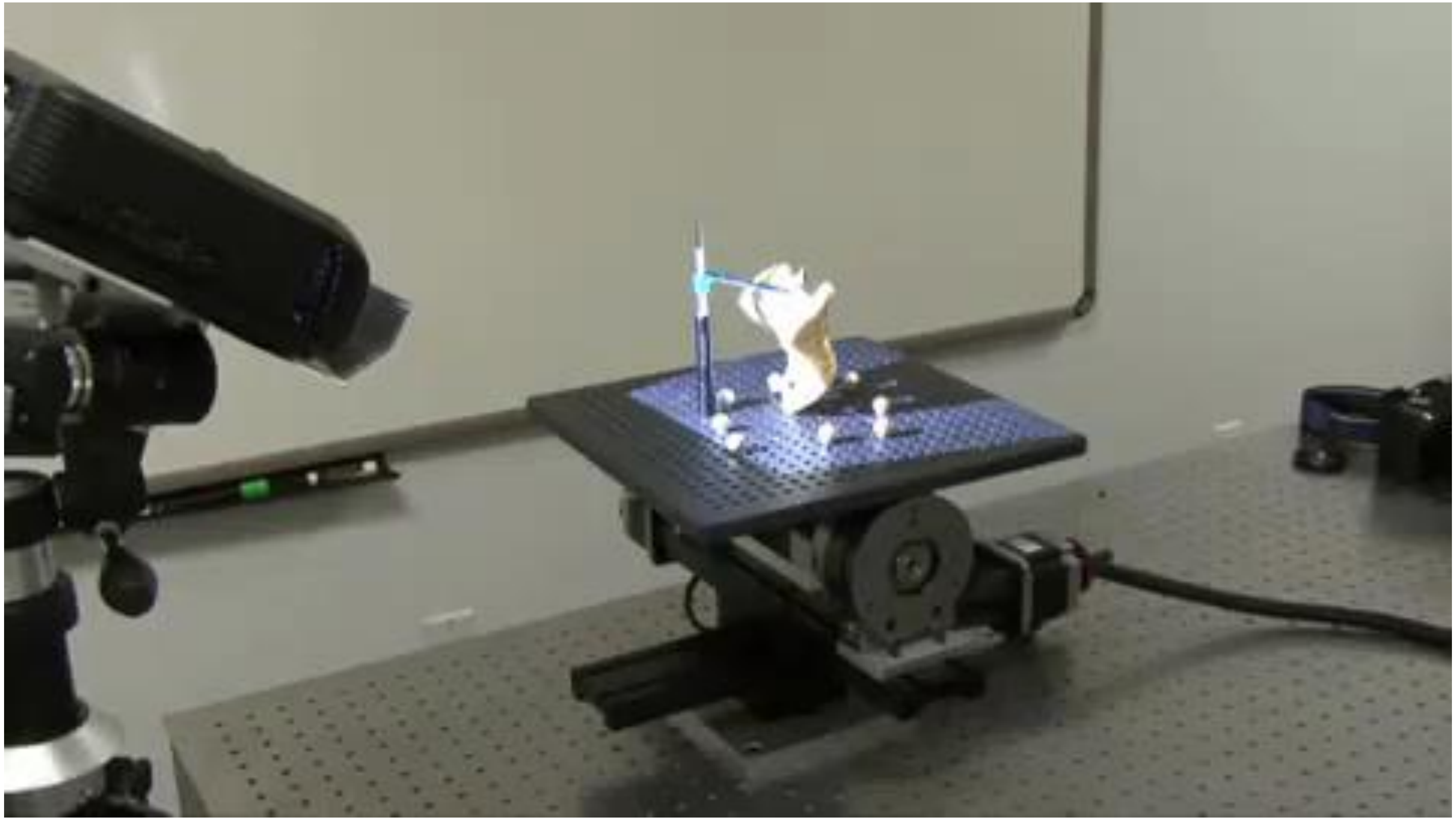
Optical scanning – active lighting

Triangulation laser

- Very precise (tens of microns)
- Works well for small distances (meters)
- Scanning is tough for surfaces (shiny or dark)

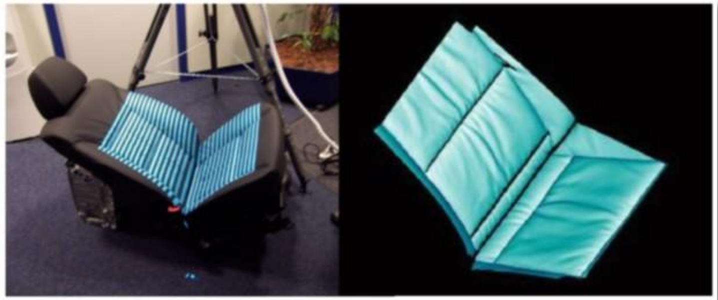
Optical scanning – active lighting

Structured light



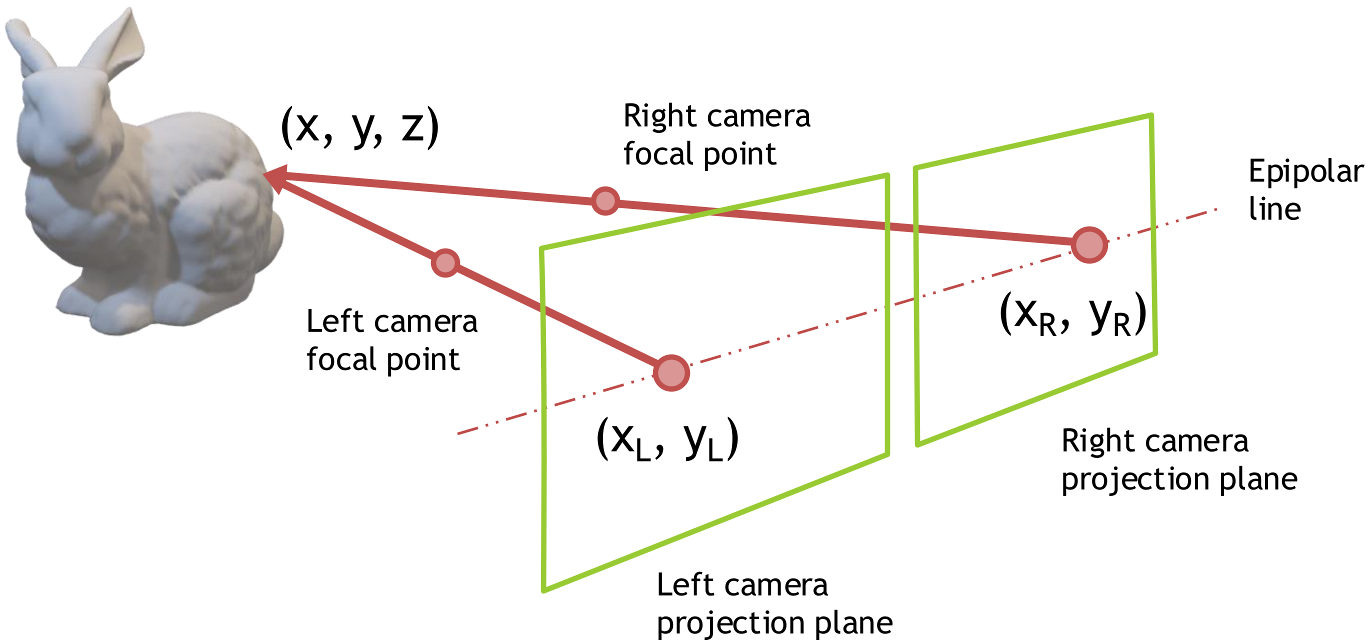
Optical scanning – active lighting
Structured light (depth camera)

- Pattern of visible or infrared light is projected onto the object (larger scanning area)
- The distortion of the pattern, recorded by the camera, provides geometric information
- Very fast – 2D pattern at once
  - Even in real time, like Intel RealSense
- Complex distance calculation, prone to noise, problems outdoors



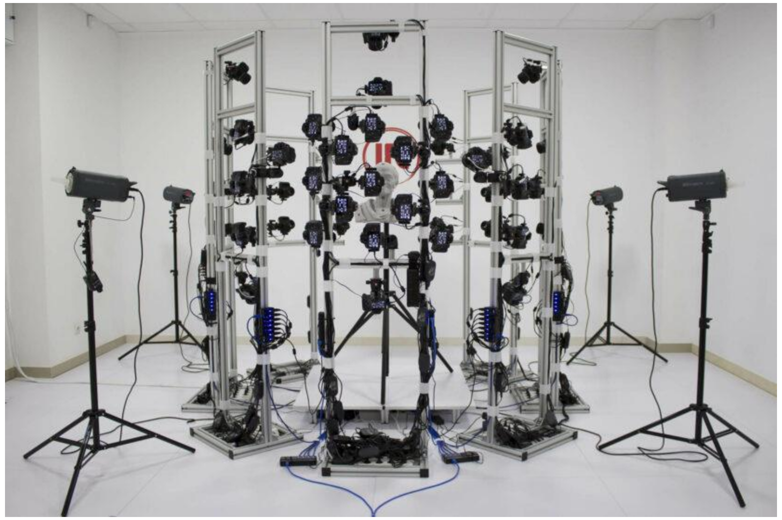
Optical scanning – passive stereo

- No need for special lighting/radiation (* but good ambient lighting helps)
- Requires two (or more) cameras
  - Feature matching and triangulation



Optical scanning – passive stereo

- Photogrammetry, multi-view reconstruction
- Sensitive to changing light conditions and ambient light
- Sensitive to density of features
- Relatively slow and inaccurate, requires significant compute resources

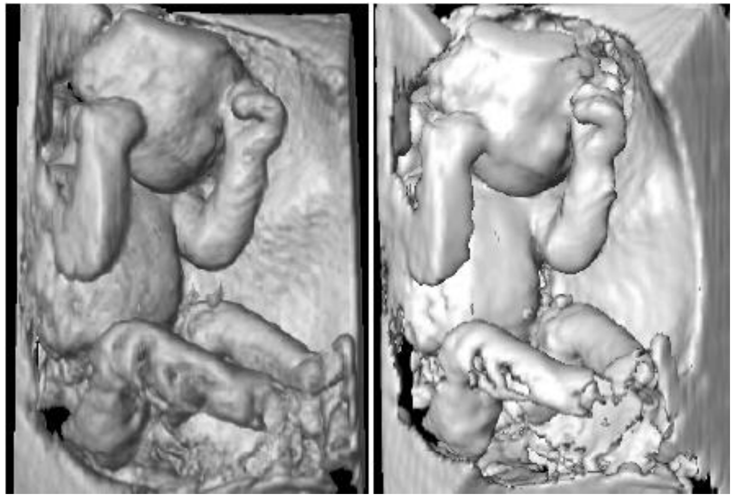


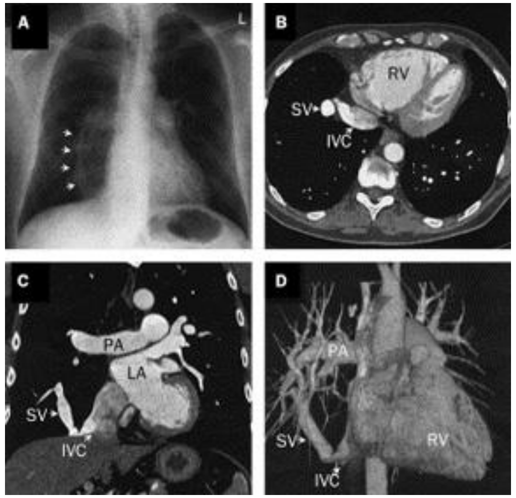
By Fxguide

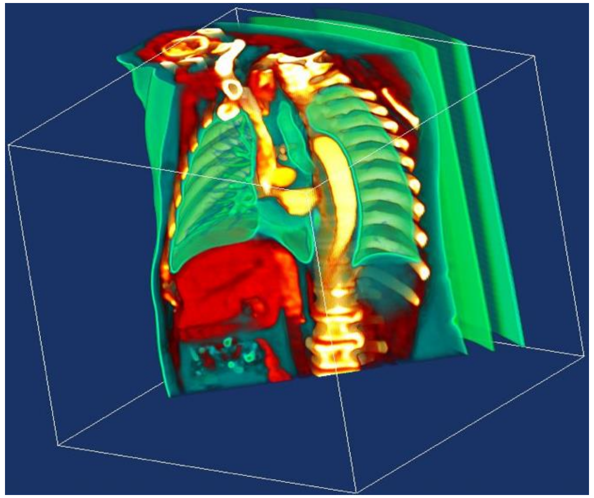
By bitfab

Imaging

- Ultrasound, CT, MRI
- Discrete volume of density data
- First need to segment the desired object (contouring)



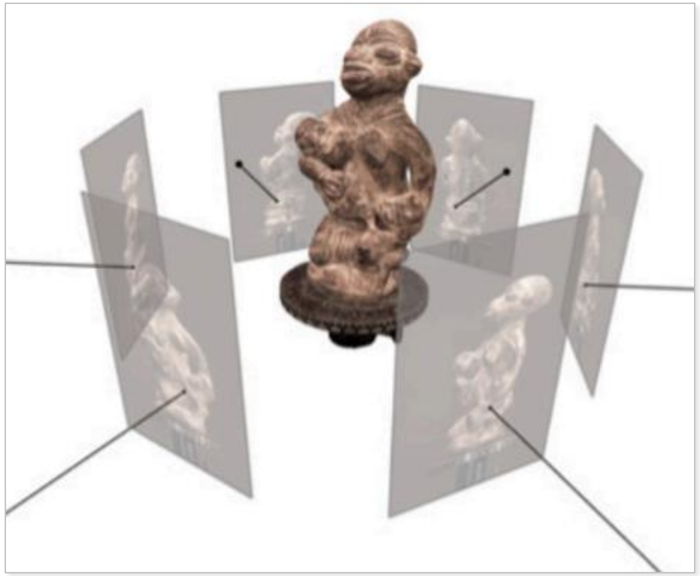




Challenges

Noise & Outliers

Incompleteness



Inconsistency

Geometry Acquisition Pipeline

Scanning
results in range images

⇨

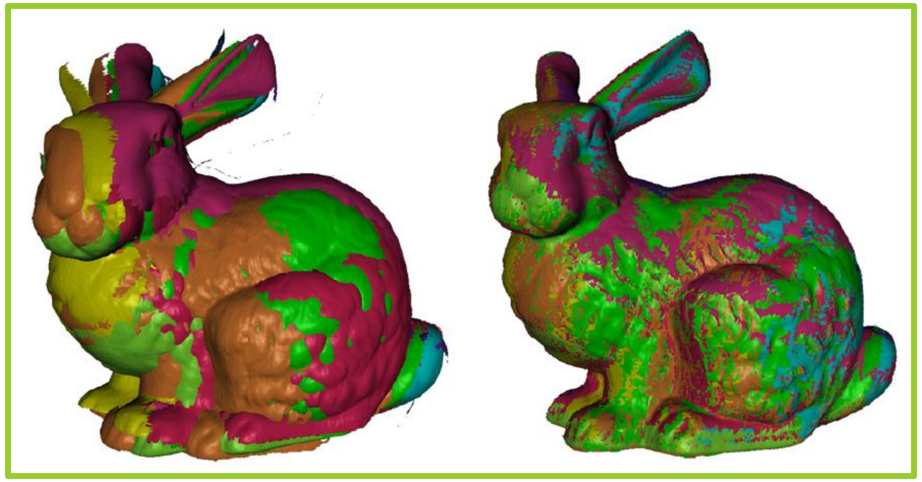
Registration
bring all range images to one coordinate system

⇨

Stitching/reconstruction
Integration of scans into a single mesh

⇨

Postprocess
Topological filtering
Geometric filtering
Remeshing
Compression



G… …eline

Scanning

results in range imag…

R…

bring…
one…

Postprocess

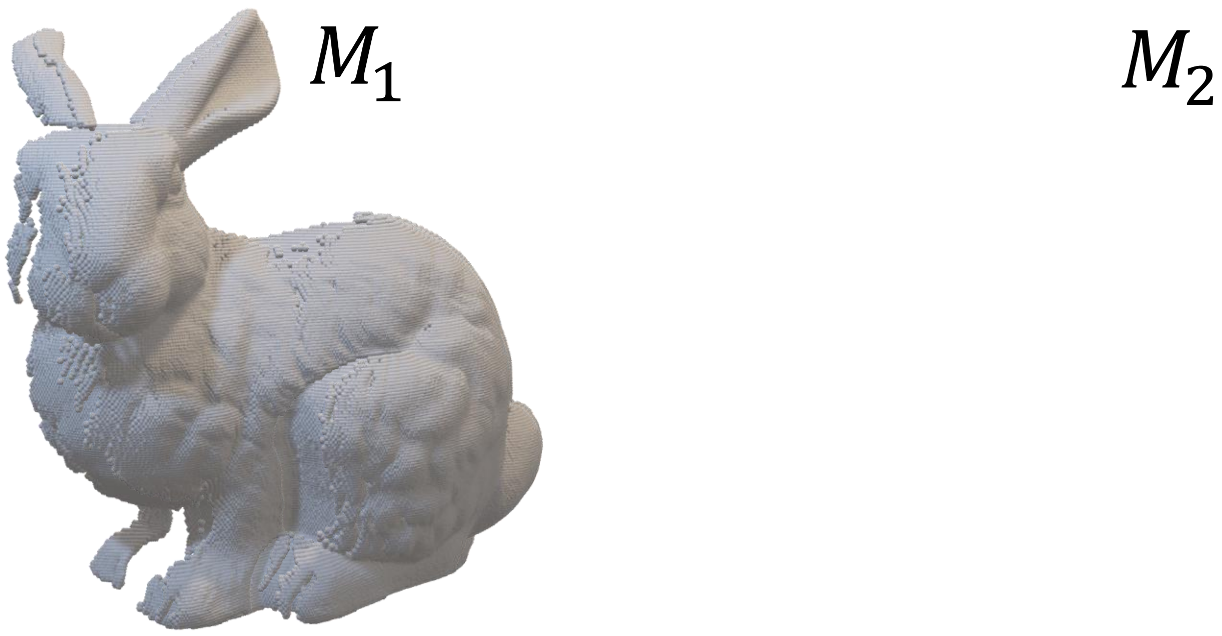
Topological filtering
Geometric filtering
Remeshing
Compression

Problem Statement

$$M_1 \approx T(M_2)$$

T: translation + rotation

Problem Statement



$$M_1 \approx T(M_2)$$

T: translation + rotation

Problem Statement

...

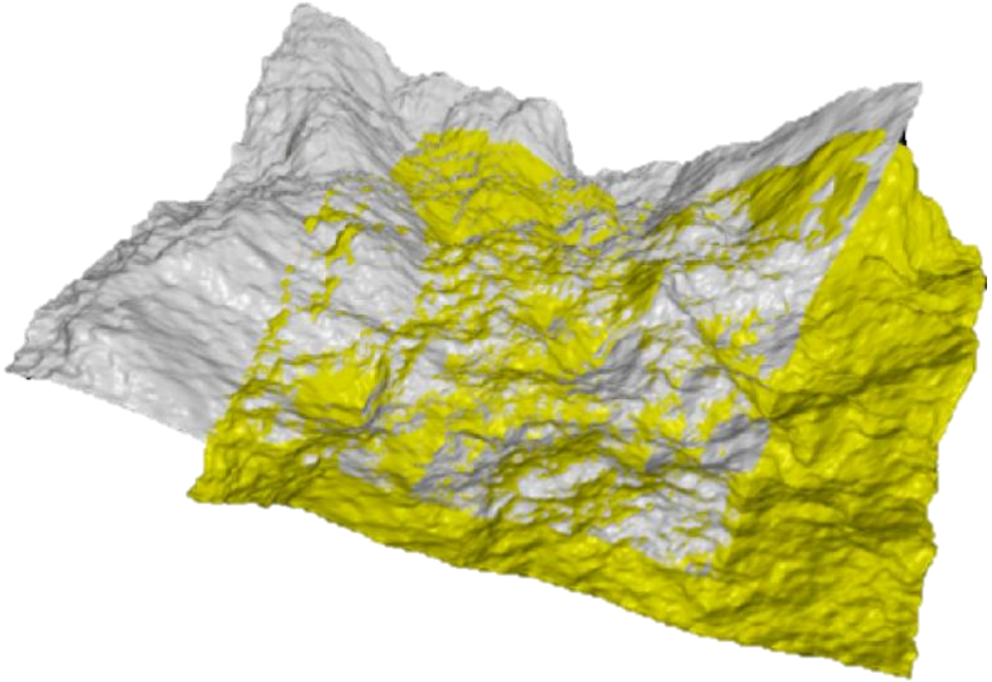
$$M_1 \approx T_2(M_2) \approx \cdots \approx T_n(M_n)$$

Given $M_1, \ldots, M_n$ find $T_2, \ldots, T_n$ such that the overlapping parts of the shapes match.

Correspondences

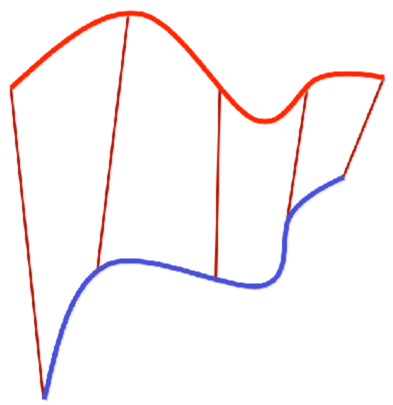
- How many points define a rigid transformation? 6 DOF
- The first problem is finding corresponding pairs!

$$
\begin{aligned}
p_1 &\rightarrow q_1 \\
p_2 &\rightarrow q_2 \\
p_3 &\rightarrow q_3 \\
&\vdots \\
R\,p_i + t &\approx q_i
\end{aligned}
$$



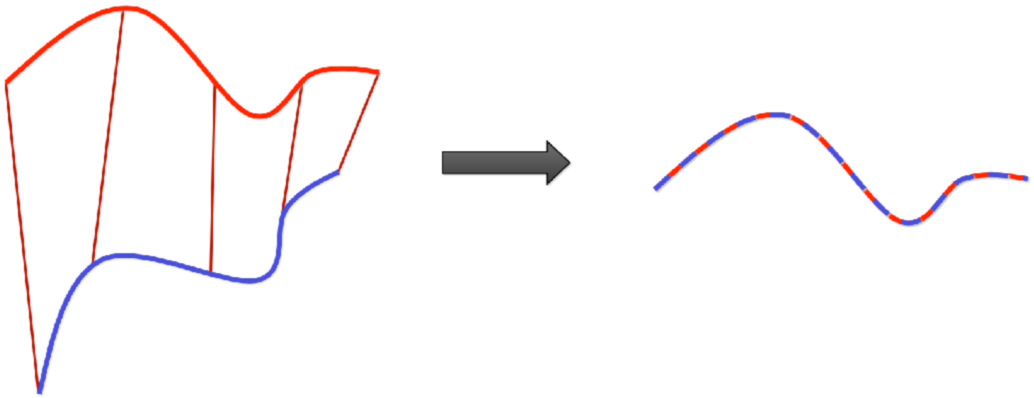
ICP: Iterative Closest Point

- Idea: Iterate
  - (1) Find correspondences
  - (2) Use them to find a transformation
- Intuition:
  - With right correspondences, problem solved



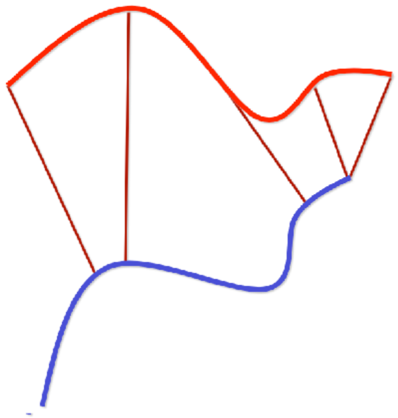
ICP: Iterative Closest Point

- Idea: Iterate
  - (1) Find correspondences
  - (2) Use them to find a transformation
- Intuition:
  - With right correspondences, problem solved



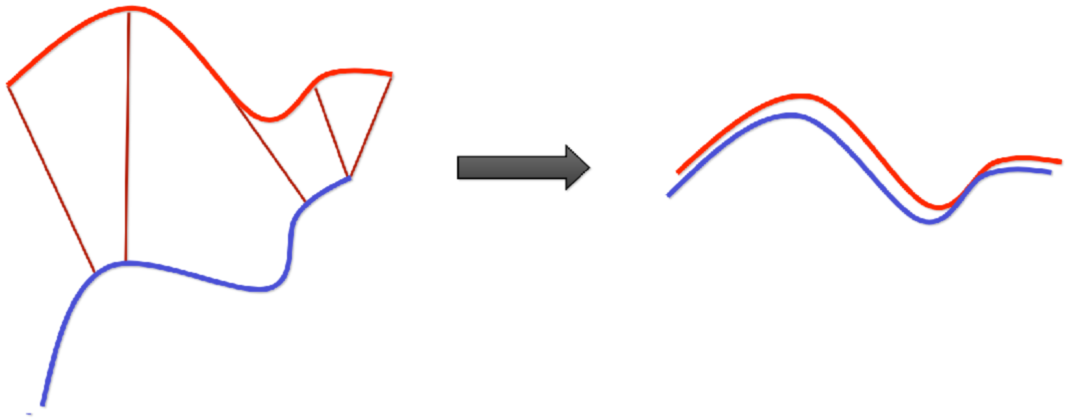
ICP: Iterative Closest Point

- Idea: Iterate
  - (1) Find correspondences
  - (2) Use them to find a transformation
- Intuition:
  - **Don’t have the right correspondences? Can still make progress!**

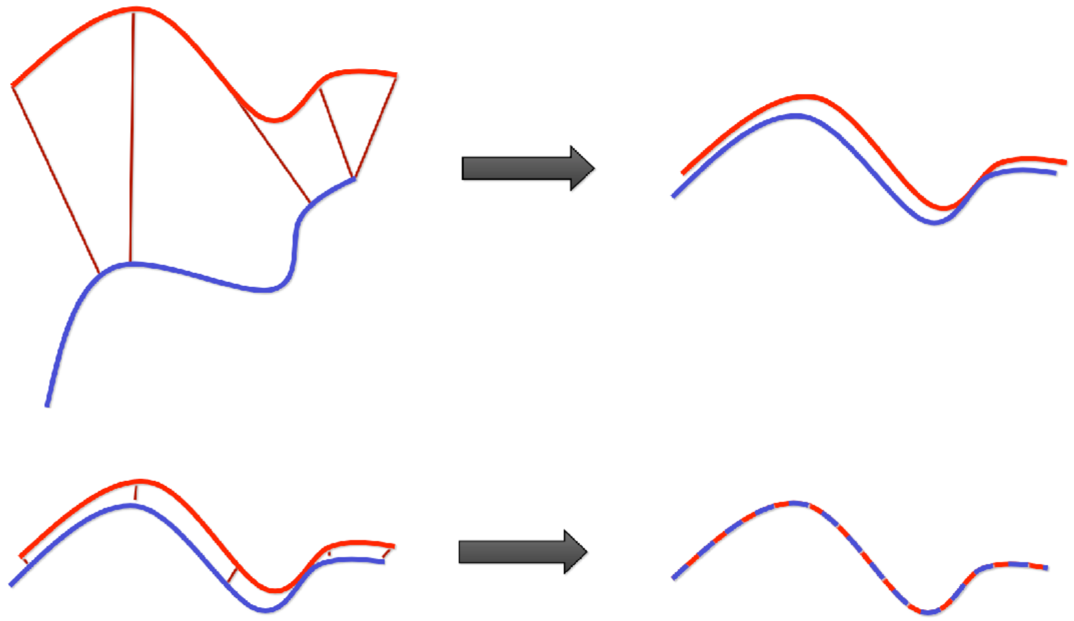


ICP: Iterative Closest Point

- Idea: Iterate
  - (1) Find correspondences
  - (2) Use them to find a transformation
- Intuition:
  - **Don’t have the right correspondences? Can still make progress!**



ICP: Iterative Closest Point



This algorithm converges to the correct solution
if the starting scans are “close enough”

ICP: Basic Algorithm

- Select (e.g., 1000) random points
- Match each point to closest point on other scan
- Reject pairs with distance too big
  - Why? How?
- Construct error function:

$$E(R,t) := \sum_{i=1}^{n} \|(Rp_i + t) - q_i\|^2$$

- Minimize
  - closed form solution in: http://dl.acm.org/citation.cfm?id=250160

ICP: Basic Algorithm

- Select (e.g., 1000) random points
- Match each point to closest point on other scan
- Reject pairs with distance too big
  - Why? How?
- Construct error function:

$$E(R,t) := \sum_{i=1}^{n} \|(Rp_i + t) - q_i\|^2$$

- Minimize
  - We will revisit this solution later: http://igl.ethz.ch/projects/ARAP/svd_rot.pdf

Geometry Acquisition Pipeline

Scanning
results in range images

⇨

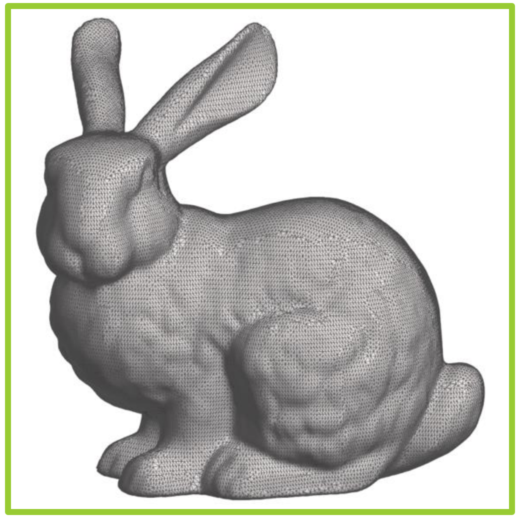
Registration
bring all range images to one coordinate system

⇨

Stitching/reconstruction
Integration of scans into a single mesh

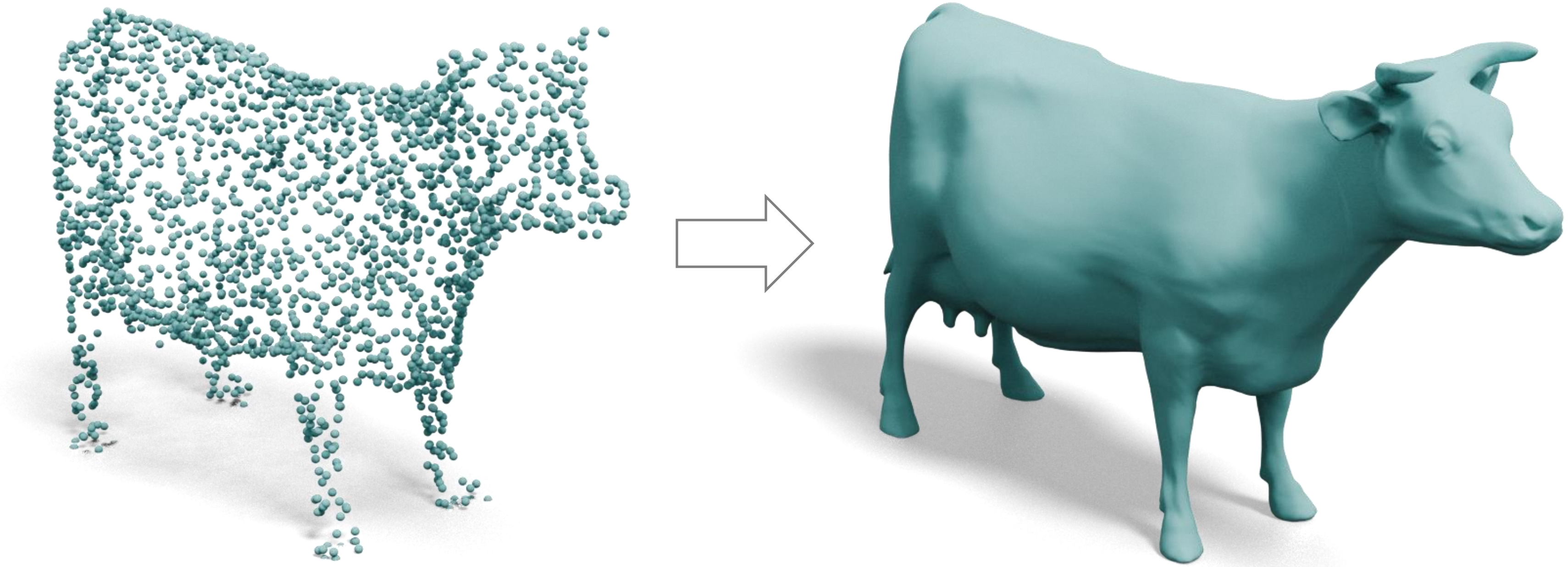
⇨

Postprocess
Topological filtering
Geometric filtering
Remeshing
Compression

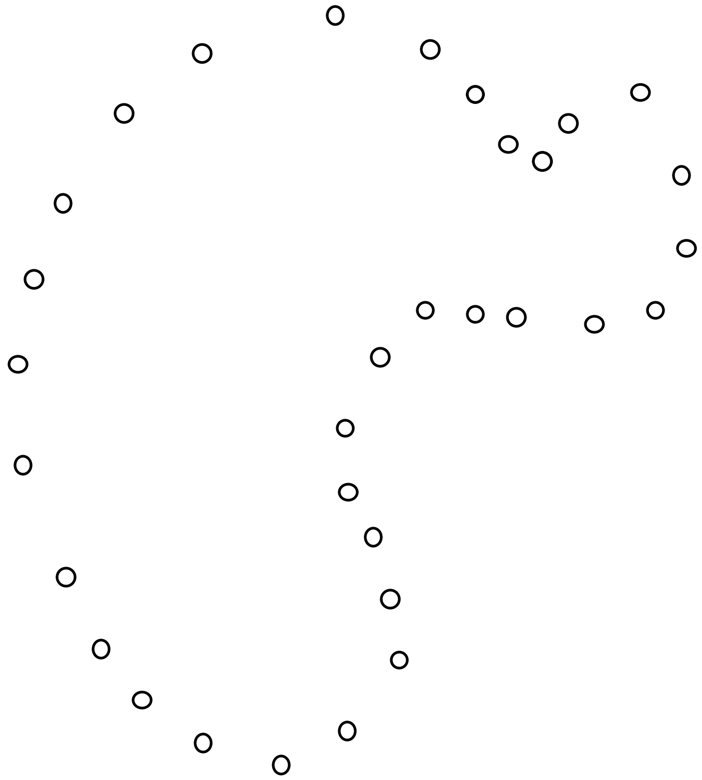


Surface Reconstruction

- Generate a mesh from a set of surface samples



Implicit Function Approach

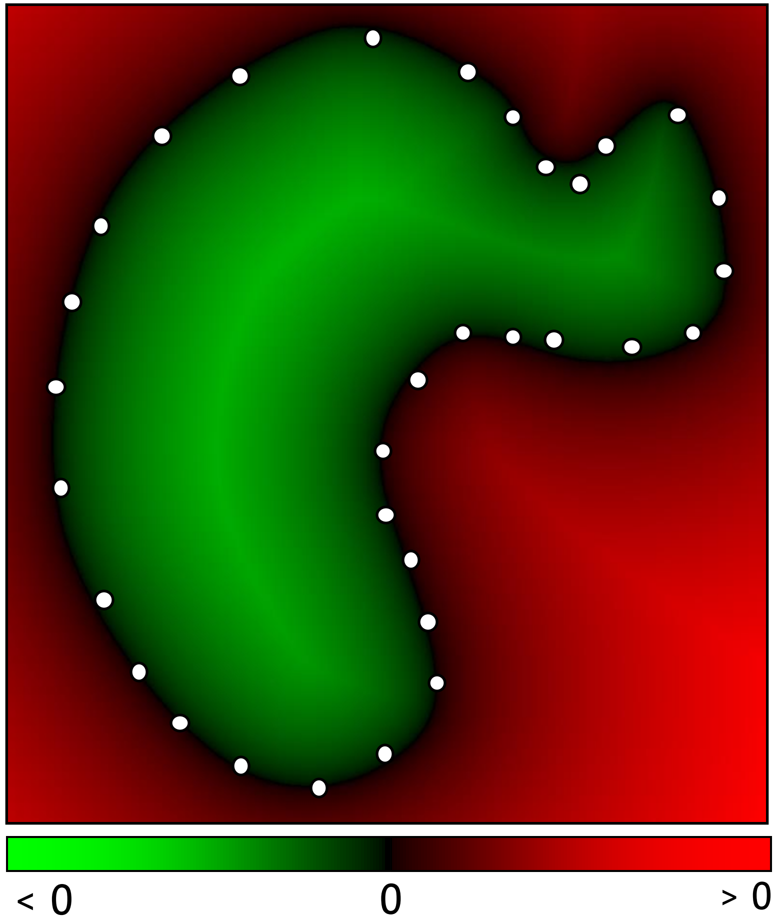


Implicit Function Approach

- Define a function

$$f : \mathbb{R}^3 \rightarrow \mathbb{R}$$

(typically with value > 0 outside the shape and < 0 inside)



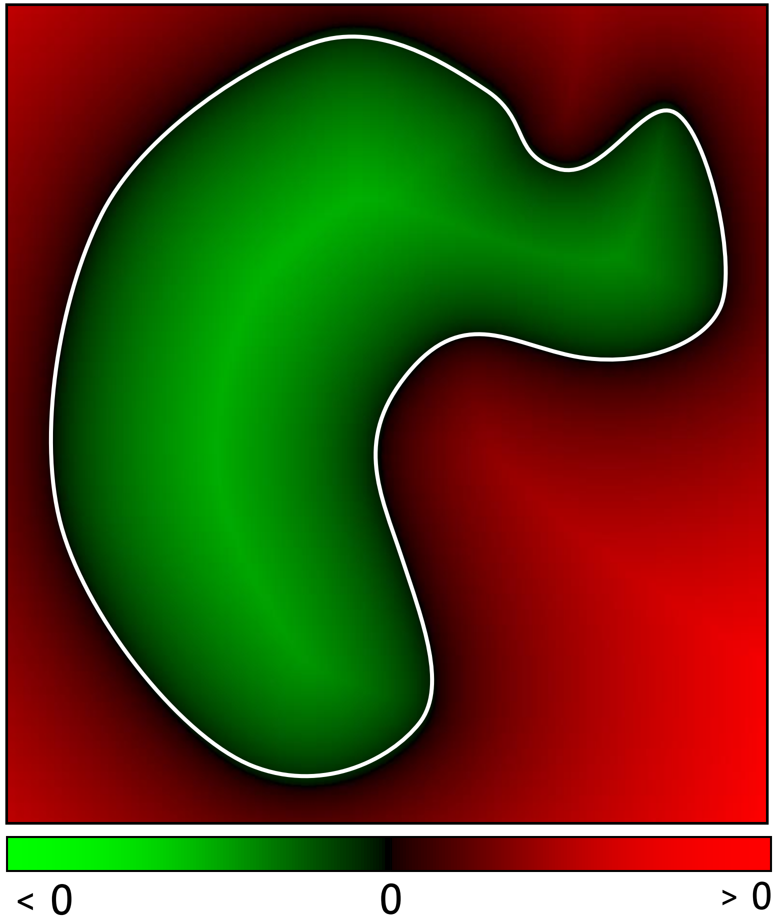
Implicit Function Approach

- Define a function

  $$f : \mathbb{R}^3 \rightarrow \mathbb{R}$$

  (typically with value > 0 outside the shape and < 0 inside)
- Extract the zero-set

  $$\{\mathbf{x} : f(\mathbf{x}) = 0\}$$



Implicit Function Approach

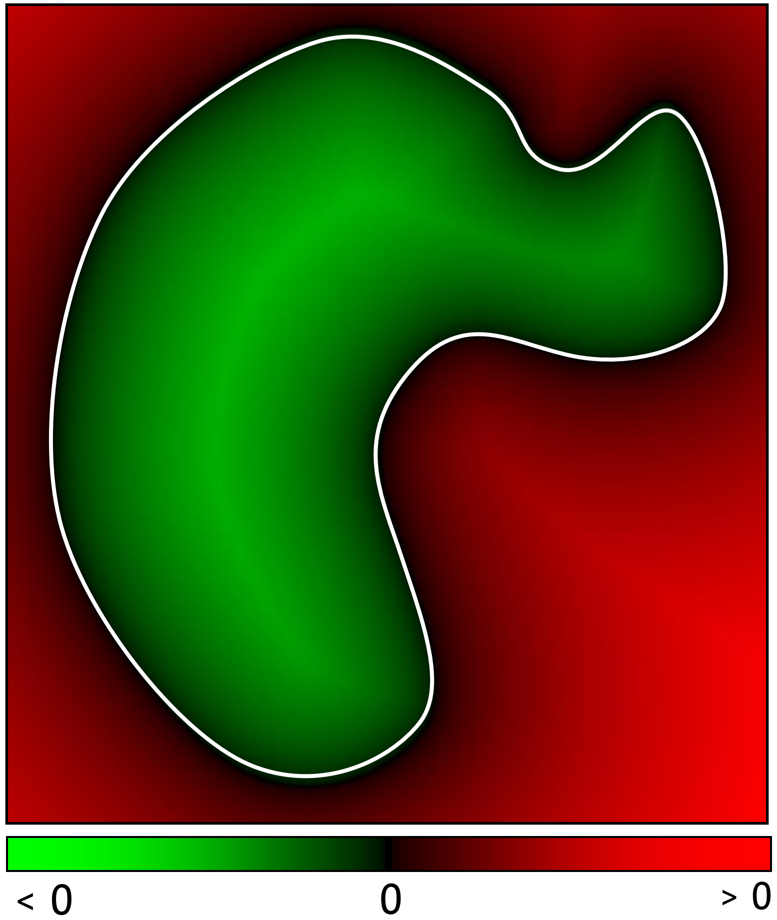
- Define a function

  $$f : \mathbb{R}^3 \rightarrow \mathbb{R}$$

  (typically with value > 0 outside the shape and < 0 inside)
- Extract the zero-set

  $$\{\mathbf{x} : f(\mathbf{x}) = 0\}$$

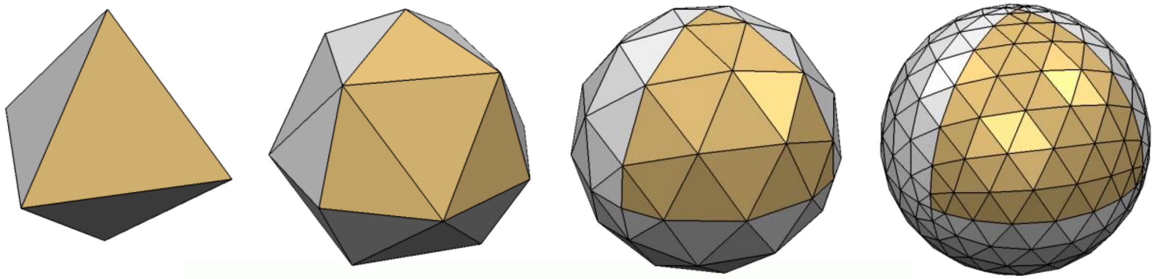
➔ Get mesh with Marching Cubes!
More on all this next week.

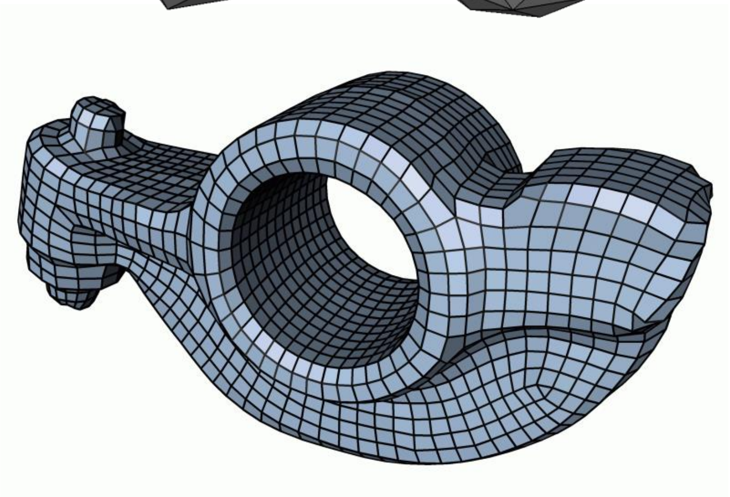


Meshes

Polygonal Meshes

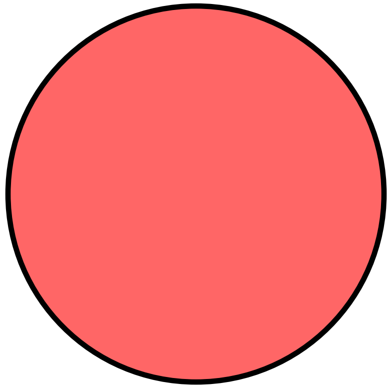
- Boundary representations of objects





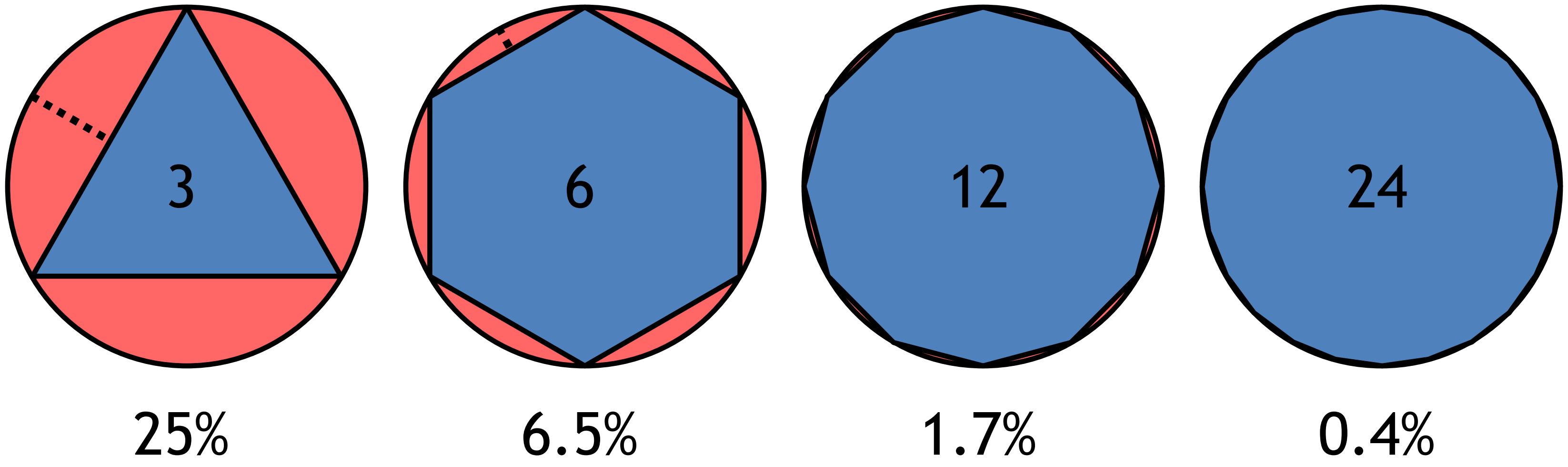
Meshes as Approximations of Smooth Surfaces

- Piecewise linear approximation
  - Error is $O(h^2)$, where h is edge-length



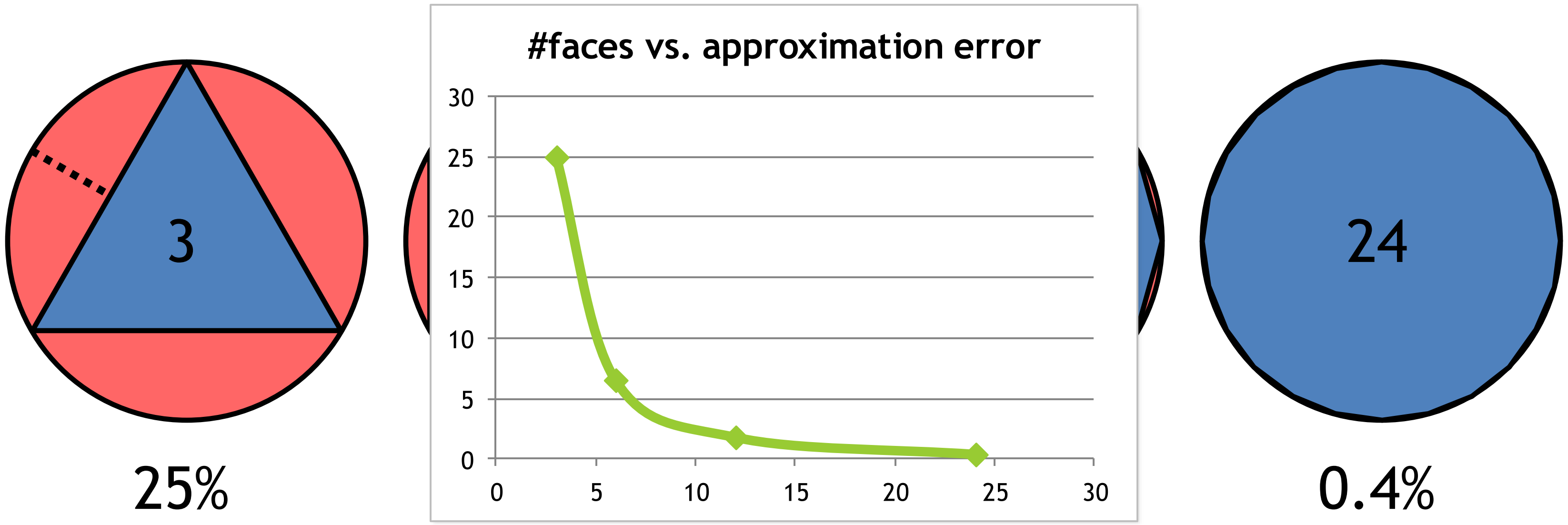
Meshes as Approximations of Smooth Surfaces

- Piecewise linear approximation
  - Error is $O(h^2)$, where h is edge-length



Meshes as Approximations of Smooth Surfaces

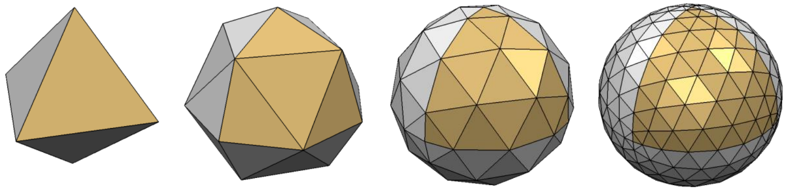
- Piecewise linear approximation
  - Error is $O(h^2)$, where h is edge-length

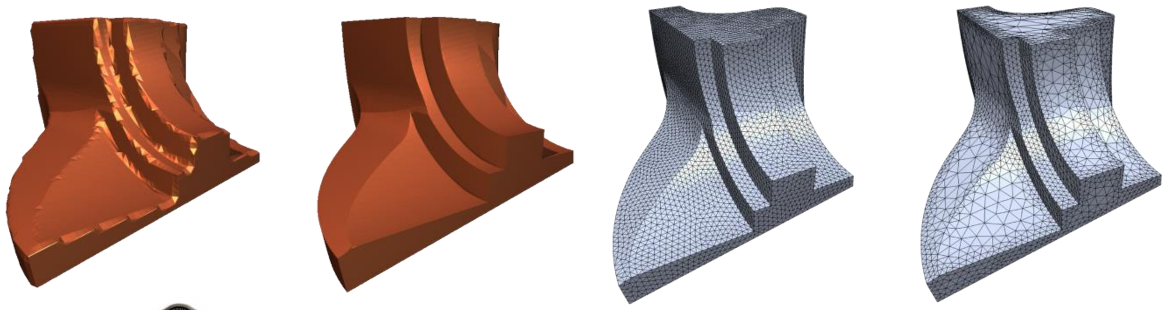


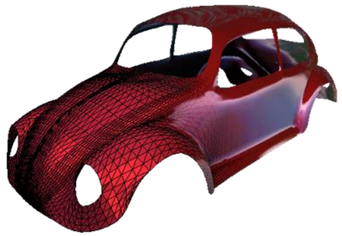
Polygonal Meshes

Polygonal meshes are a good representation

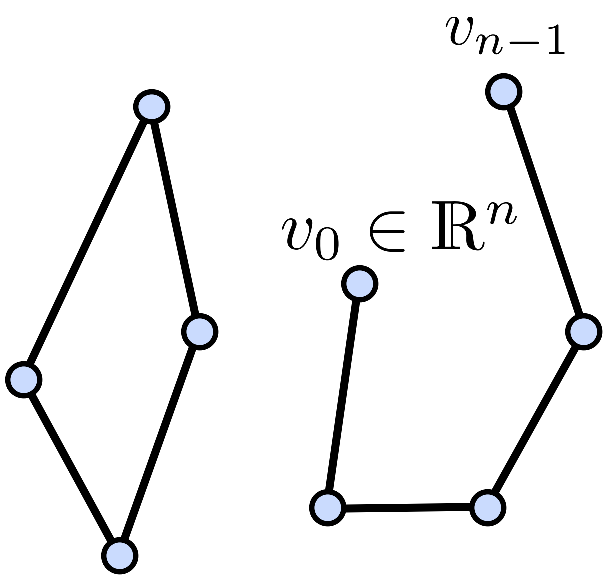
- approximation $O(h^2)$
- arbitrary topology
- piecewise smooth surfaces
- adaptive refinement
- efficient rendering







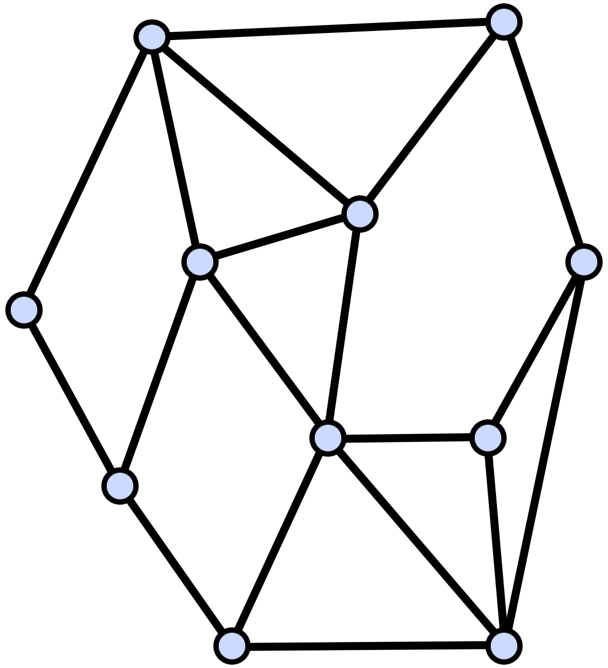
Polygon



- Vertices: $v_0, v_1, \ldots, v_{n-1}$
- Edges: $\{(v_0, v_1), \ldots, (v_{n-2}, v_{n-1})\}$

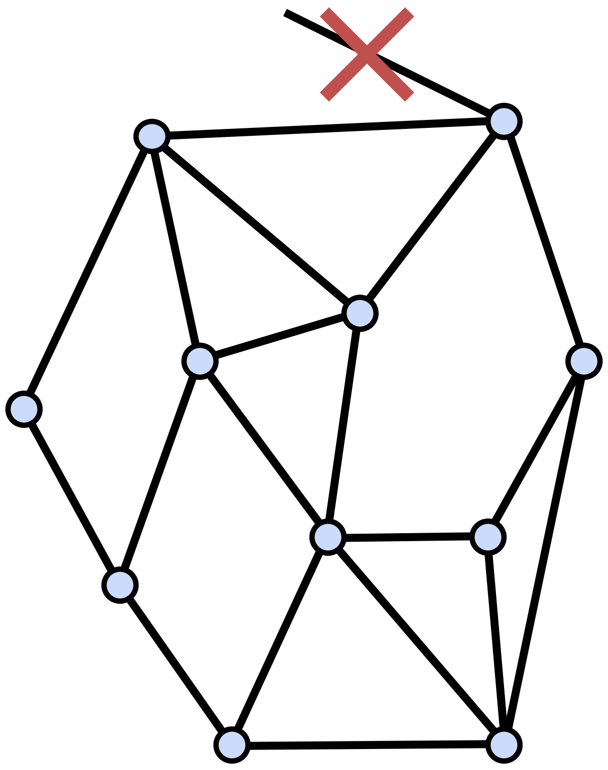
- Closed: $v_0 = v_{n-1}$
- Planar: all vertices on a plane
- Simple: not self-intersecting

Polygonal Mesh



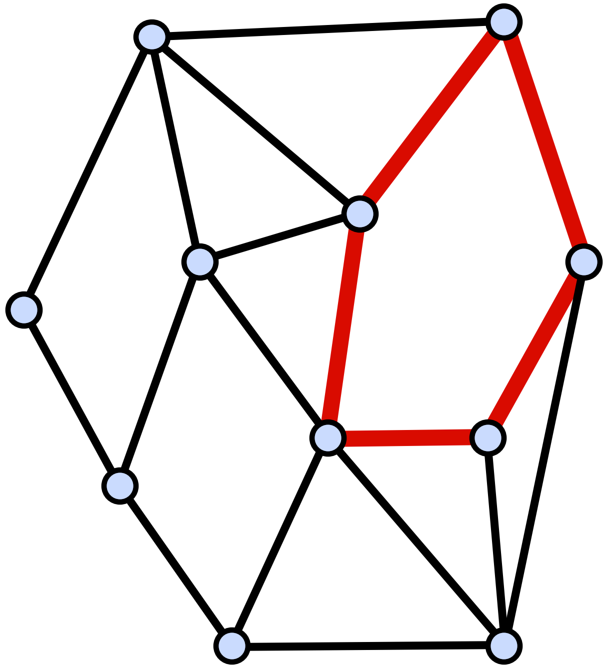
- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge

Polygonal Mesh



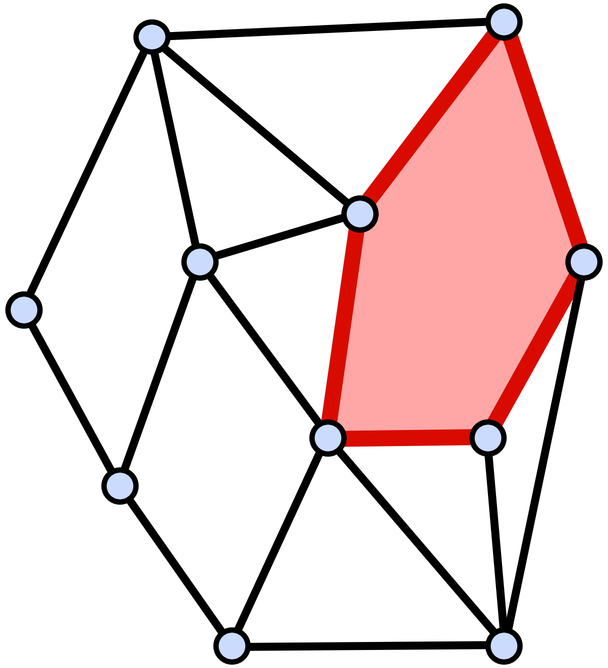
- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon

Polygonal Mesh



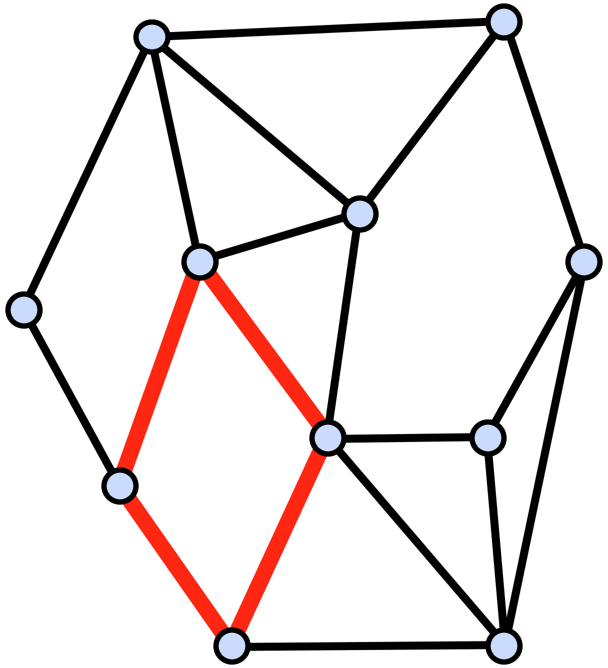
- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon
- Each Q_i defines a **face** of the polygonal mesh

Polygonal Mesh



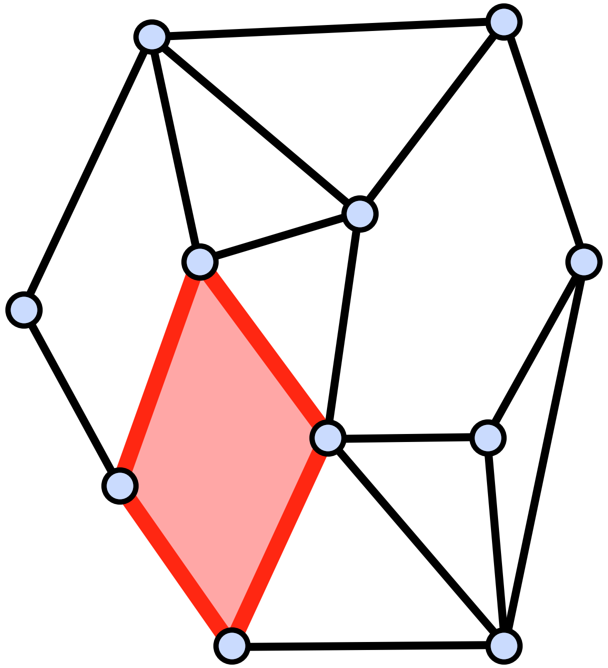
- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon
- Each Q_i defines a **face** of the polygonal mesh

Polygonal Mesh



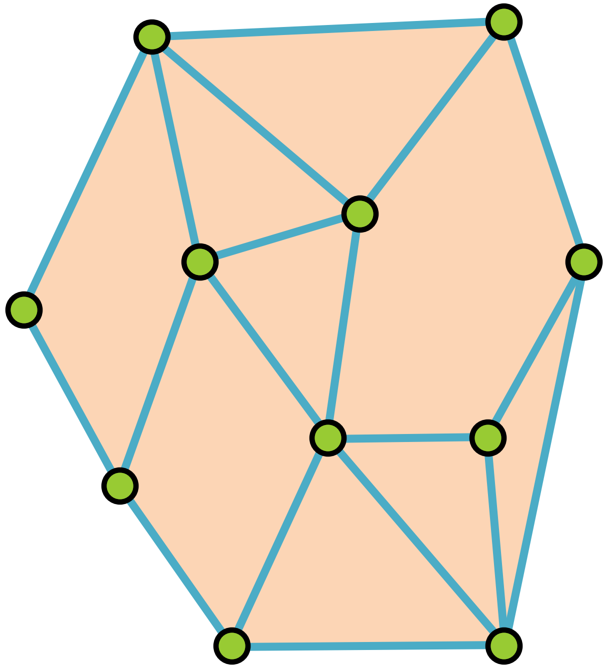
- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon
- Each Q_i defines a **face** of the polygonal mesh

Polygonal Mesh



- A finite set M of closed, simple polygons Q_i is a polygonal mesh
- The intersection of two polygons in M is either empty, a vertex, or an edge
- Every edge belongs to at least one polygon
- Each Q_i defines a **face** of the polygonal mesh

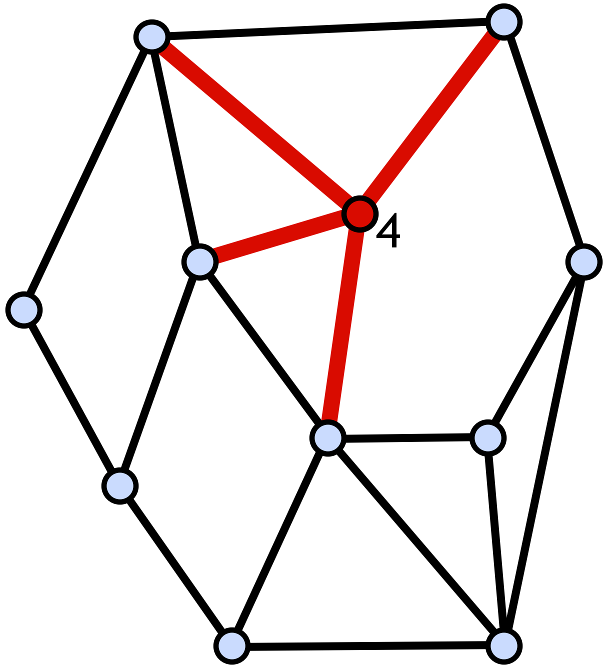
Polygonal Mesh



$$M = \langle V, E, F \rangle$$

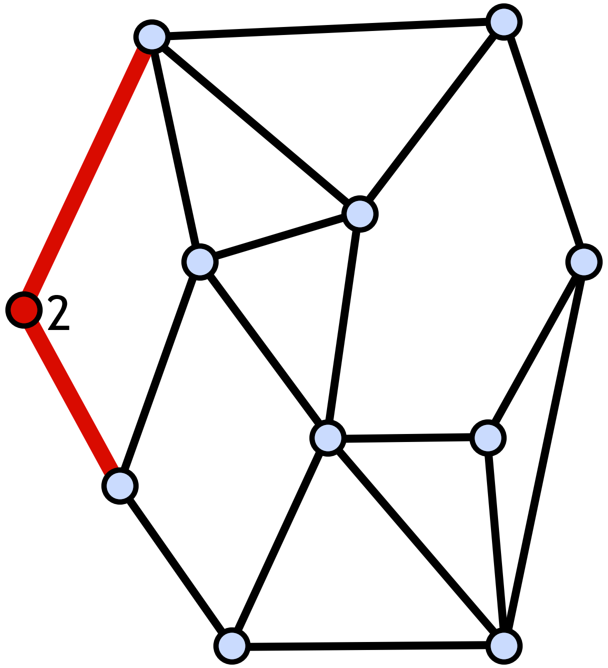
vertices — edges — faces

Polygonal Mesh



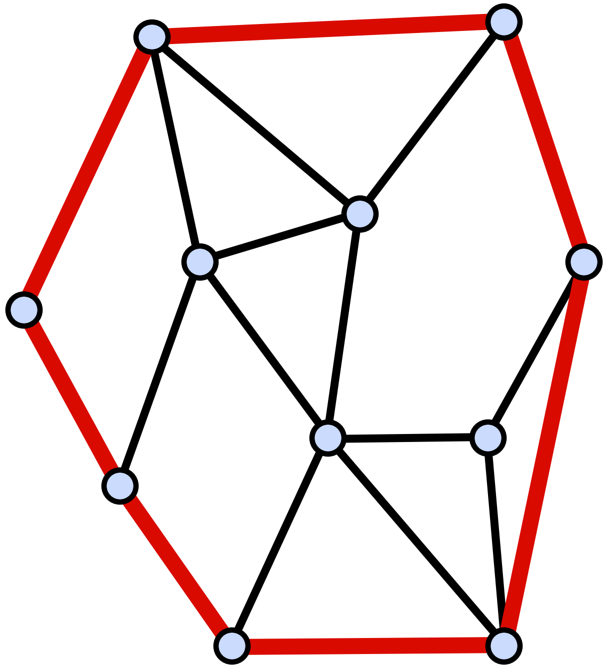
- Vertex **degree** or **valence** = number of incident edges

Polygonal Mesh

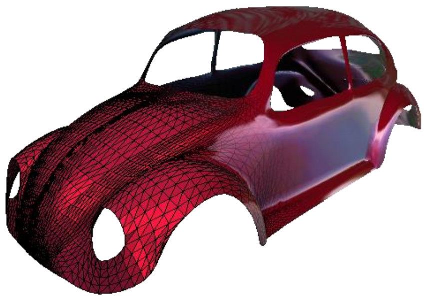


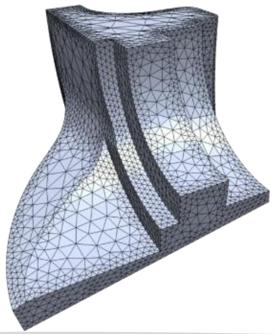
- Vertex **degree** or **valence**
  =
  number of incident edges

Polygonal Mesh



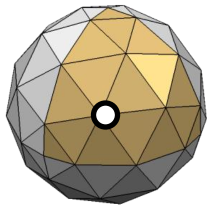
- **Boundary:** the set of all edges that belong to only one polygon
  - Either empty or forms closed loops
  - If empty, then the polygonal mesh is closed





Triangle Meshes

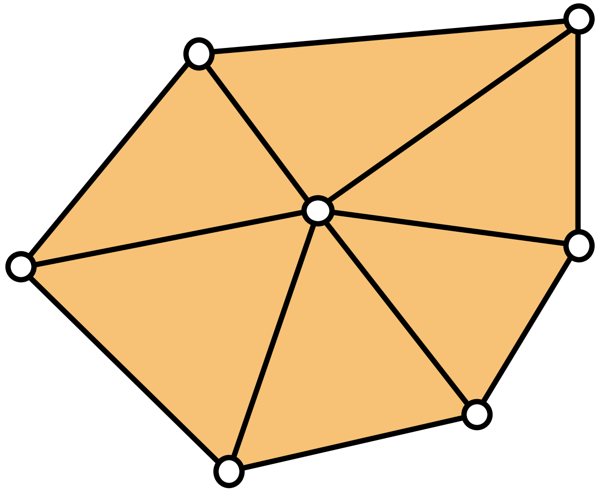
- Connectivity: vertices, edges, triangles
- Geometry: vertex positions



$$V = \{v_1, \ldots, v_n\}$$

$$E = \{e_1, \ldots, e_k\}, \quad e_i \in V \times V$$

$$F = \{f_1, \ldots, f_m\}, \quad f_i \in V \times V \times V$$

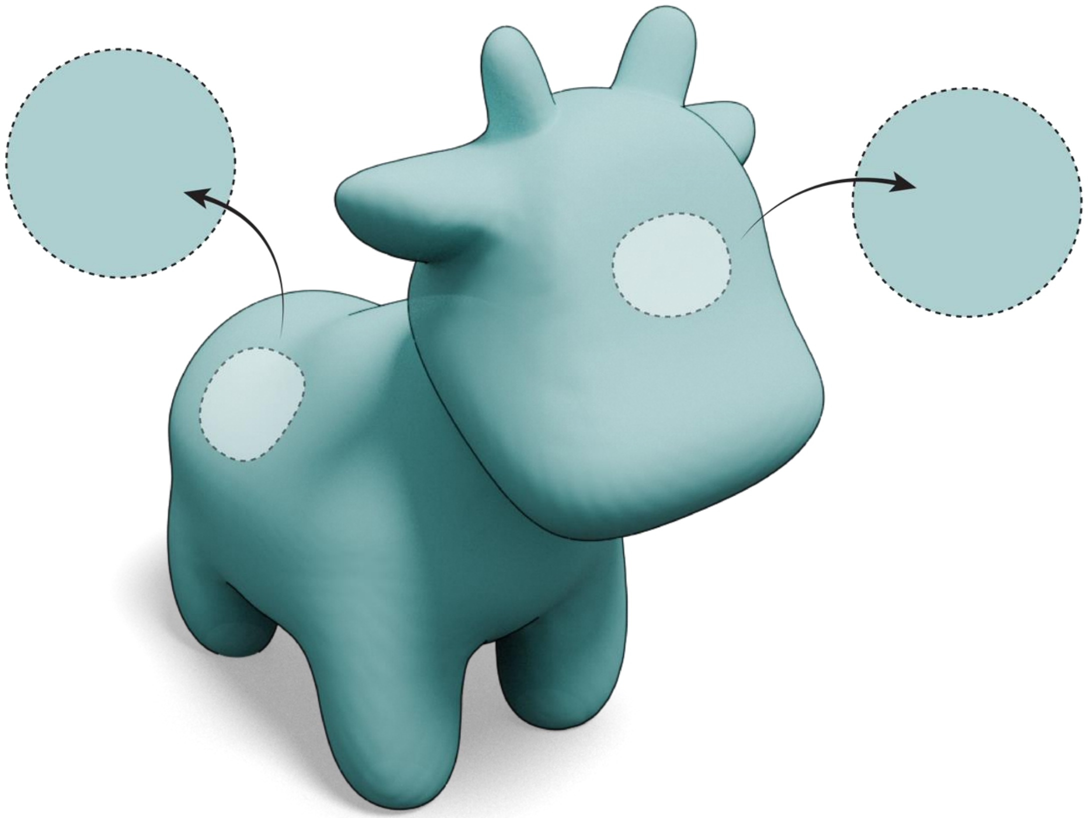


$$P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}, \quad \mathbf{p}_i \in \mathbb{R}^3$$

Manifolds

- A surface is a closed **2-manifold** if it is everywhere locally homeomorphic to a disk

$$B_{\mathbf{x}}(r) = \{\mathbf{y} \in \mathbb{R}^3 \ s.t. \ \|\mathbf{y} - \mathbf{x}\| < r\}$$

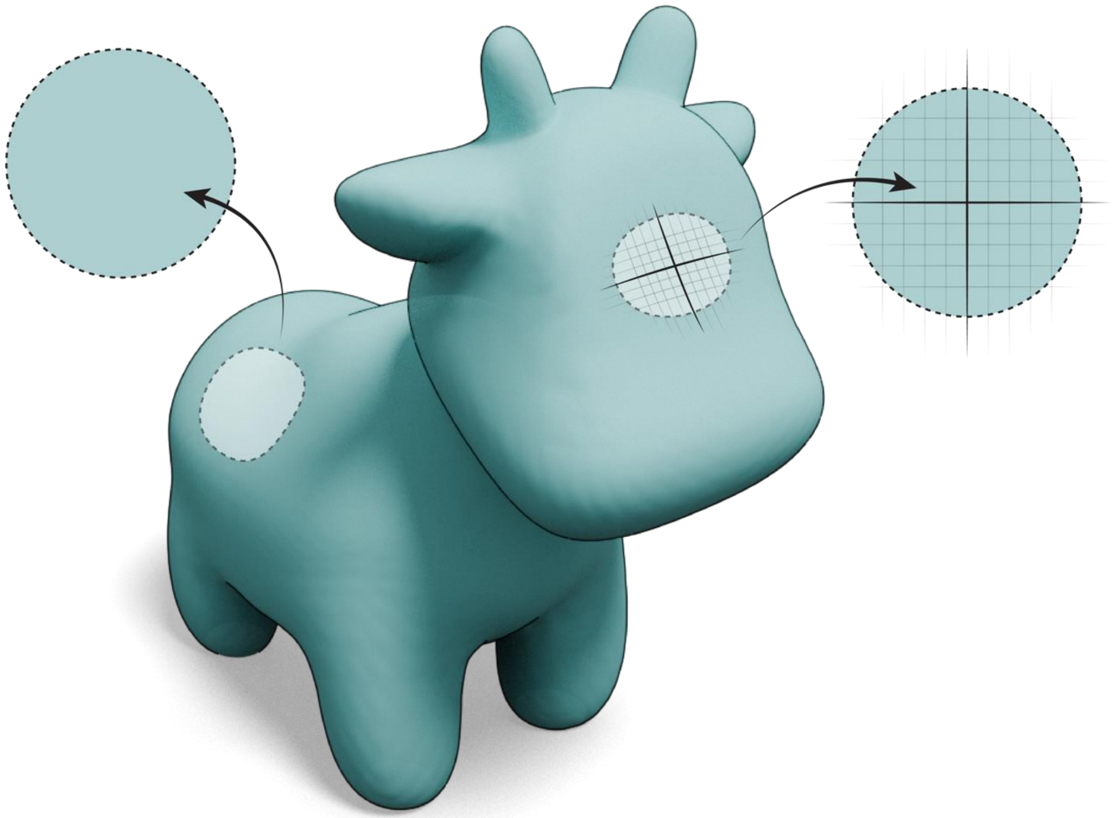


Manifolds

- A surface is a closed **2-manifold** if it is everywhere locally homeomorphic to a disk

$$B_{\mathbf{x}}(r) = \{\mathbf{y} \in \mathbb{R}^3 \; s.t. \; \|\mathbf{y} - \mathbf{x}\| < r\}$$

Homeomorphic
- one-to-one (bijective)
- continuous in both directions

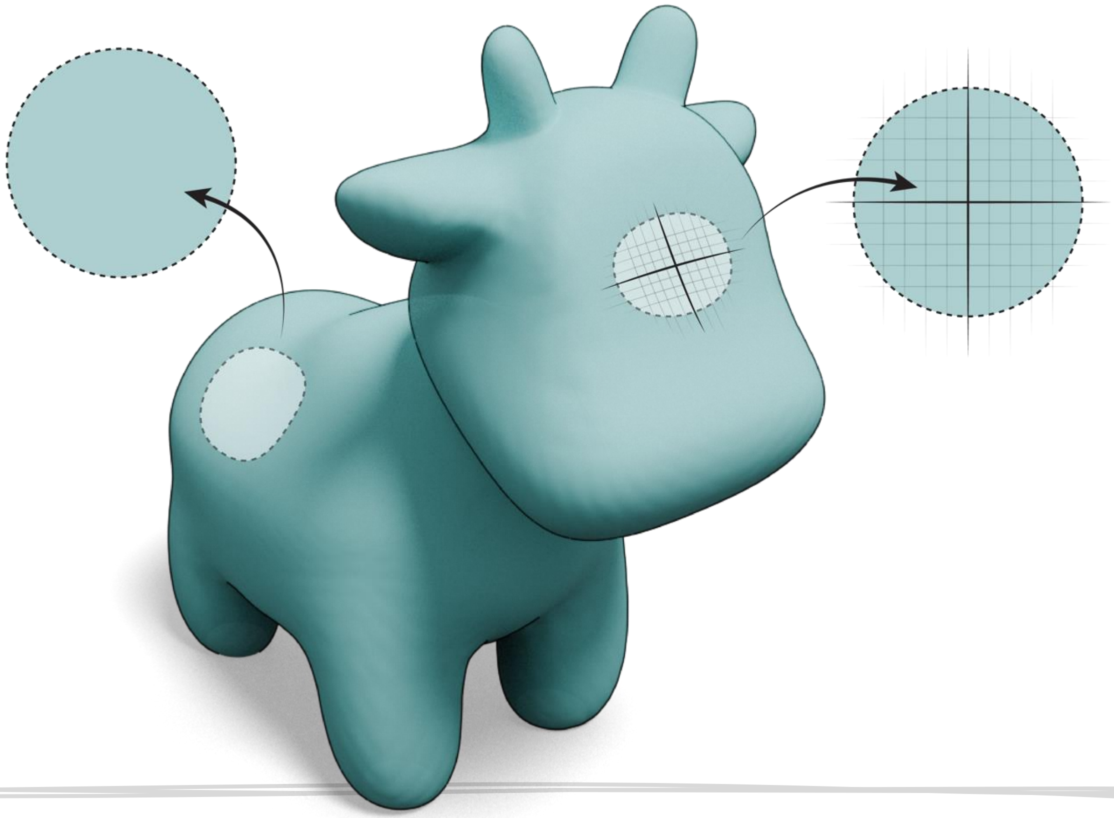


Manifolds

- For every point $\mathbf{x}$ in M, there is an **open** ball $B_{\mathbf{x}}(r)$ of radius $r > 0$ centered at $\mathbf{x}$ such that $M \cap B_{\mathbf{x}}$ is homeomorphic to an open disk

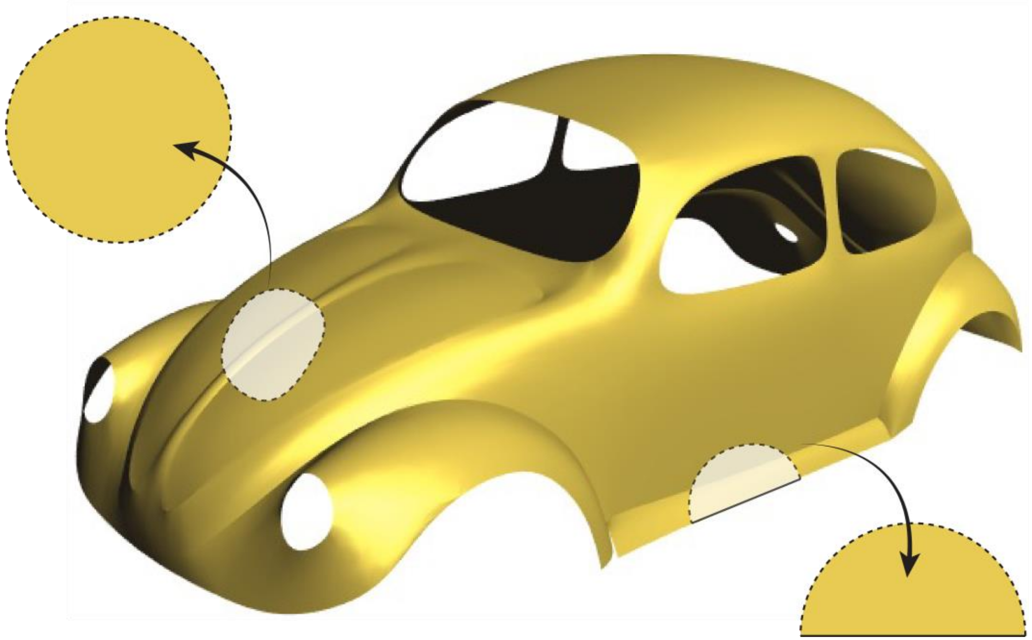
$$B_{\mathbf{x}}(r) = \{\mathbf{y} \in \mathbb{R}^3 \;\; s.t. \;\; \|\mathbf{y} - \mathbf{x}\| < r\}$$

Homeomorphic
- one-to-one (bijective)
- continuous in both directions

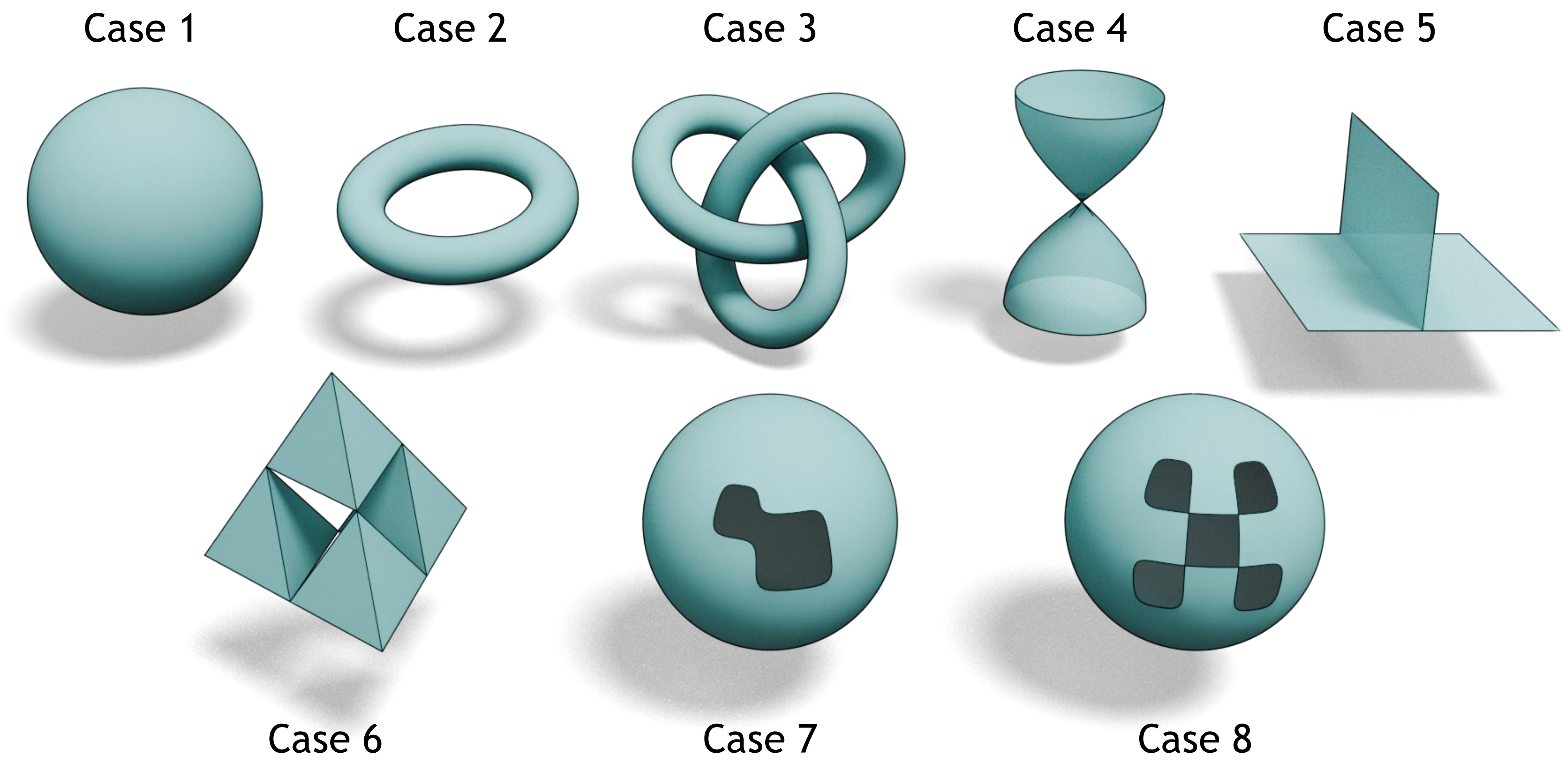


Manifolds

- Manifold with boundary: a vicinity of each boundary point is homeomorphic to a half-disk

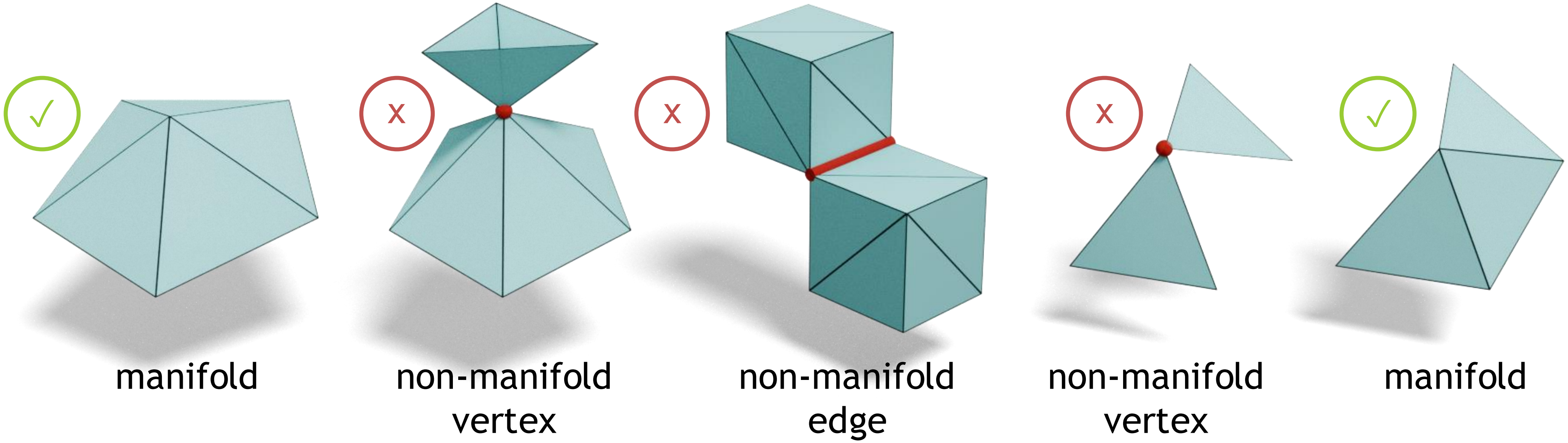


Is it 2-manifold or not? Why?



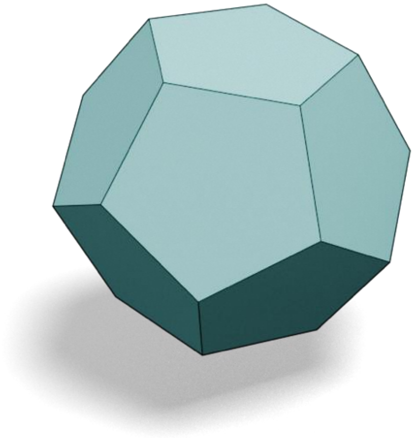
Manifold meshes

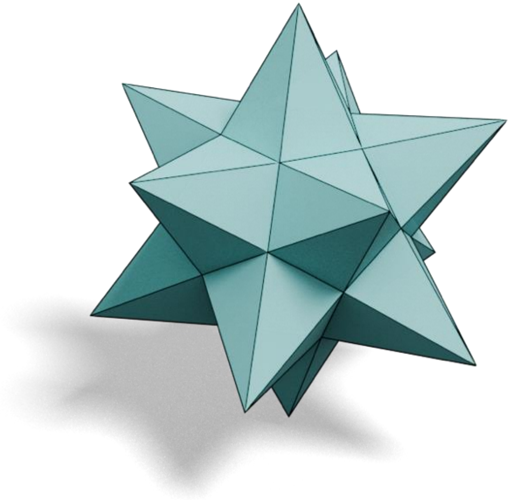
- Manifold: at most 2 faces sharing an edge
  - Boundary edges have one incident face
  - Inner edges have two incident faces
- A manifold vertex has 1 connected (half-)ring of faces

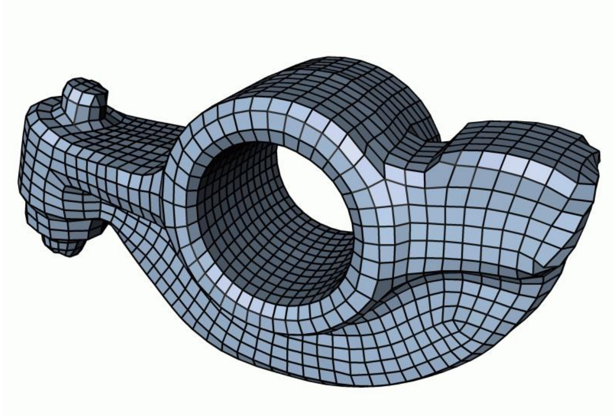


Manifolds

- If closed and not self-intersecting, a manifold divides the space into inside and outside
- A closed manifold polygonal mesh is also called **polyhedron**

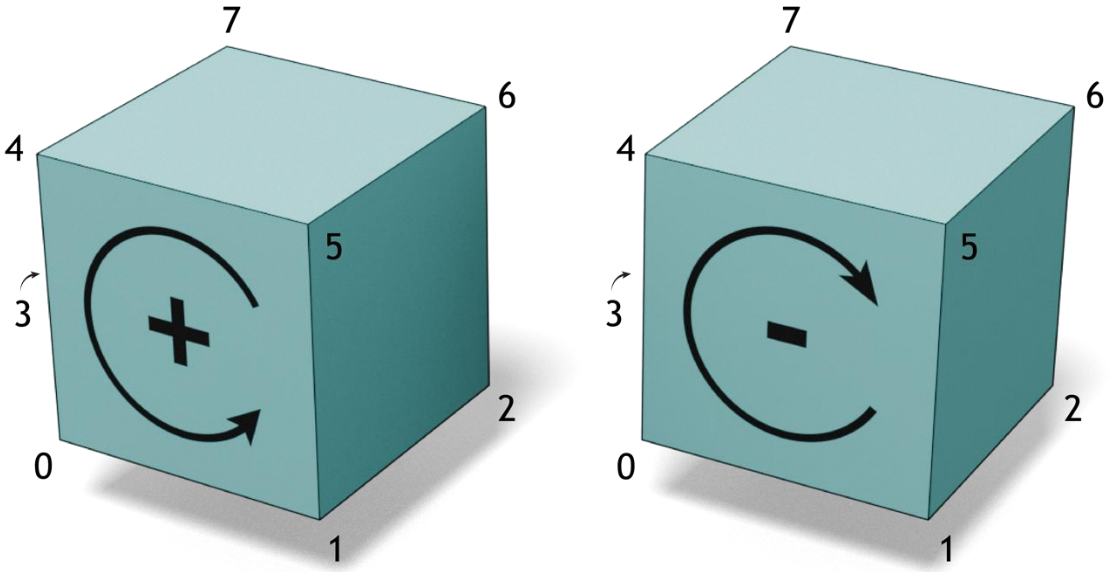






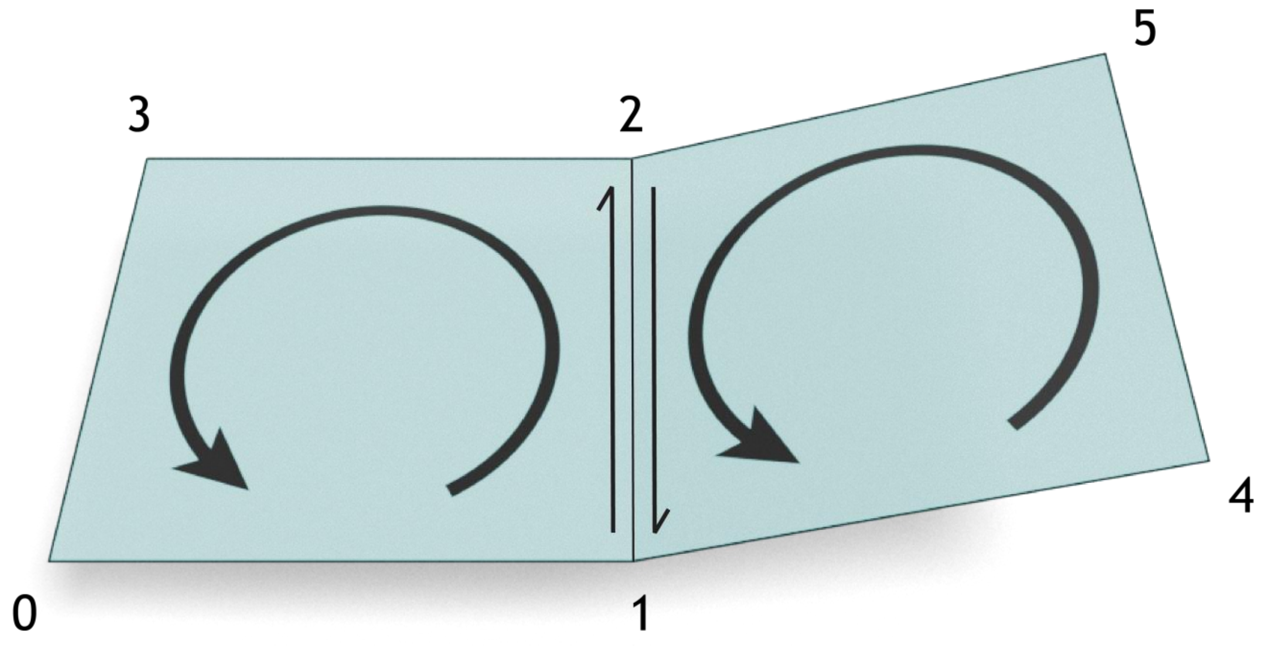
Orientation

- Every face of a polygonal mesh is orientable
  - Clockwise vs. counterclockwise order of face vertices
  - Defines sign/direction of the surface normal



Orientation

- Consistent orientation of neighboring faces:



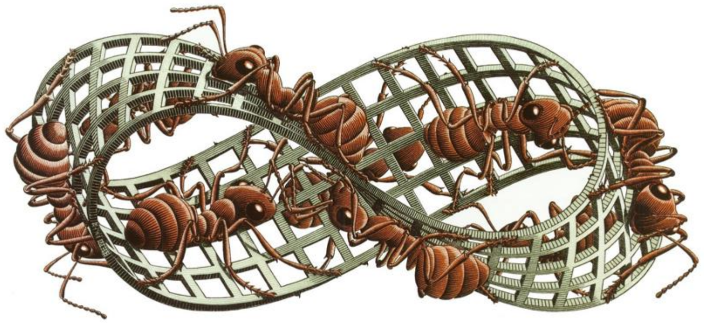
face 1: (3,0,1,2) face 2: (2,1,4,5)

Orientability

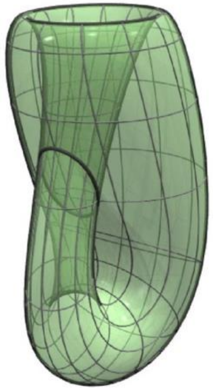
- A polygonal mesh is *orientable*, if all faces can be oriented such that the incident faces to every edge are *consistently* oriented
  - If the faces are consistently oriented for every edge, the mesh is oriented

- Note
  - Every non-orientable *closed* mesh embedded in $\mathbb{R}^3$ intersects itself
  - A non-self-intersecting *polyhedron* is always orientable

Möbius strip

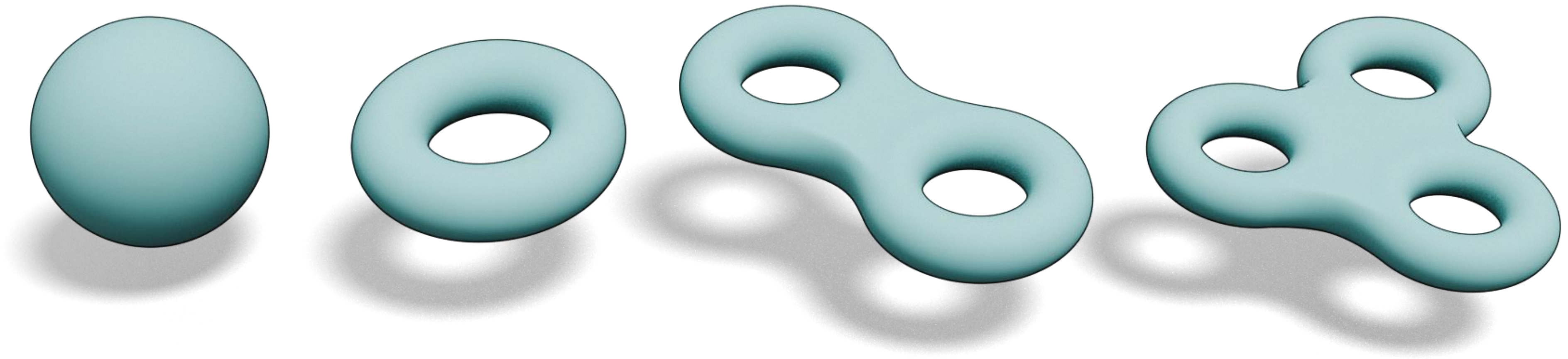


Klein bottle



Global Topology of Meshes

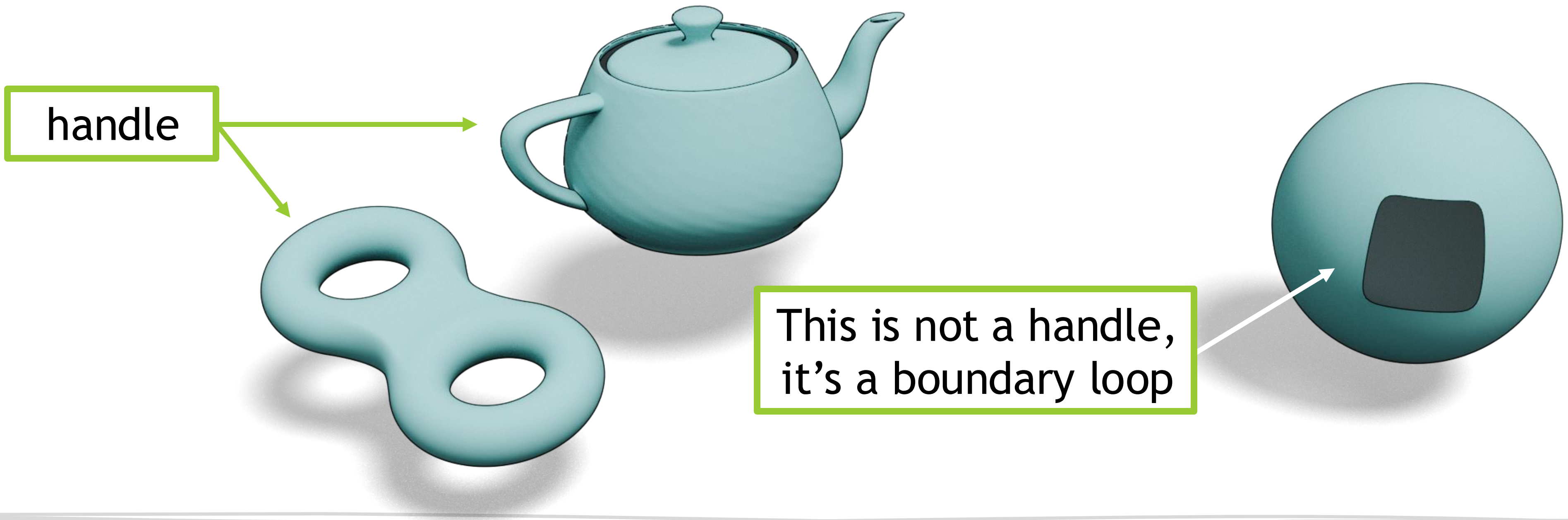
- **Genus:** ½ × the maximal number of closed paths that do not disconnect the graph
  - Informally, the number of handles ("donut holes")



Genus 0 Genus 1 Genus 2 Genus 3

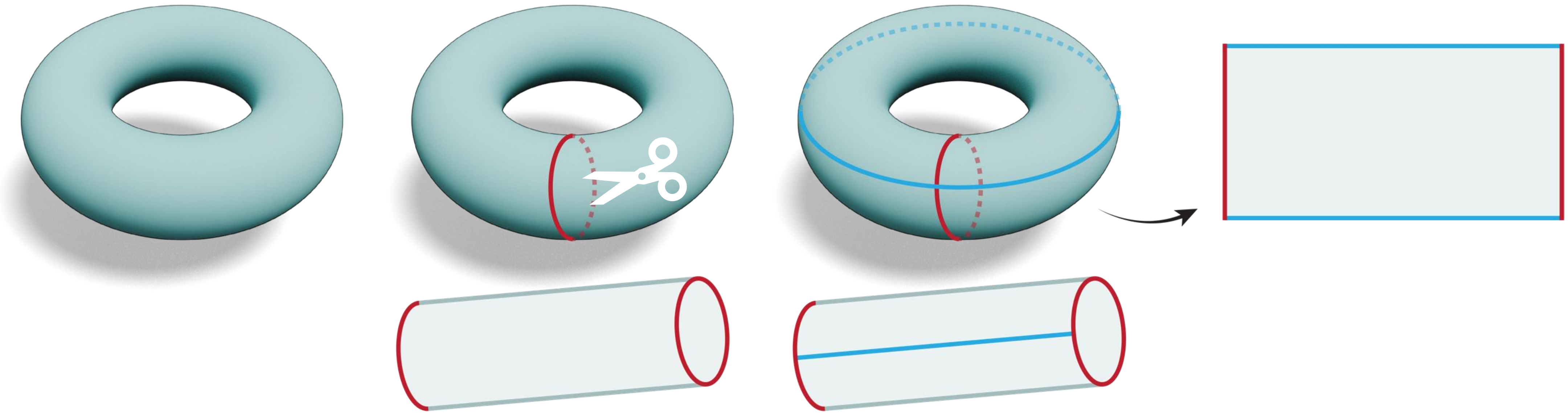
Global Topology of Meshes

- **Genus:** ½ × the maximal number of closed paths that do not disconnect the graph
  - Informally, the number of handles ("donut holes")



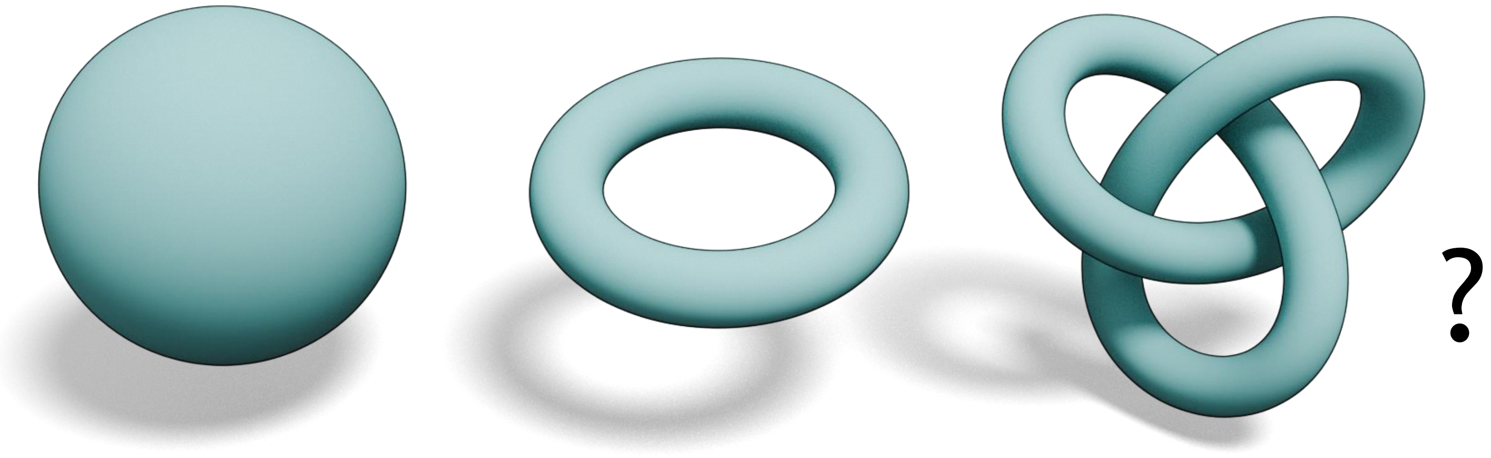
Global Topology of Meshes

- **Genus:** ½ × the maximal number of closed paths that do not disconnect the graph
  - Informally, the number of handles ("donut holes")



Global Topology of Meshes

- **Genus:** ½ × the maximal number of closed paths that do not disconnect the graph



Genus 0 Genus 1

Euler-Poincaré Formula

- Theorem (Euler): The sum

$$\chi(M) = v - e + f$$

is **constant** for a given surface topology, no matter which (manifold) mesh we choose

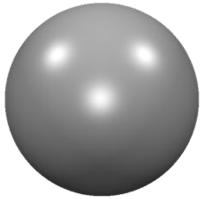
  - v = number of vertices
  - e = number of edges
  - f = number of faces

Euler-Poincaré Formula

- For orientable meshes:

$$v - e + f = 2(c - g) - b = \chi(M)$$

  - c = number of connected components
  - g = genus
  - b = number of boundary loops

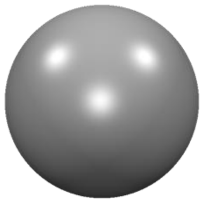
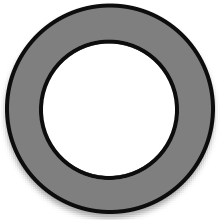
$\chi($  $) = 2$ $\chi($  $) = 0$

Euler-Poincaré Formula

- For orientable meshes:

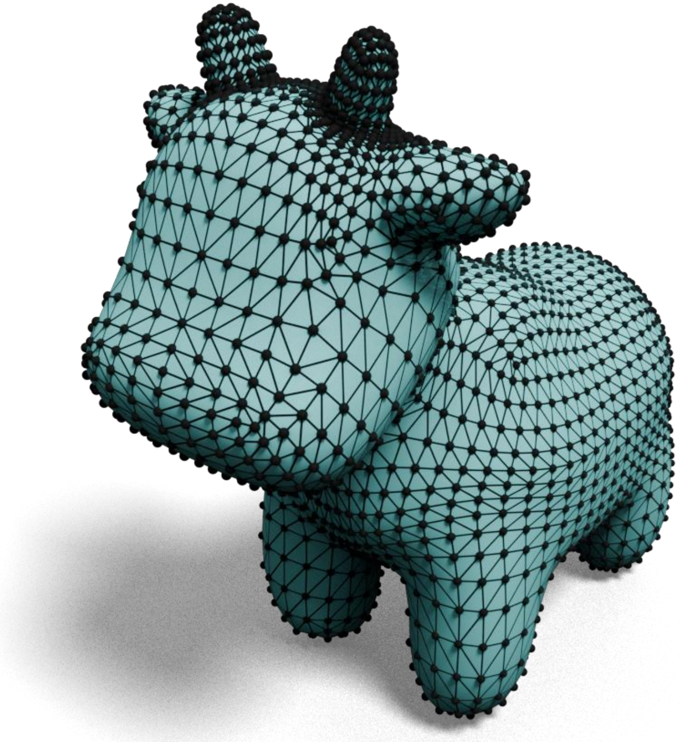
$$v - e + f = 2(c - g) - b = \chi(M)$$

  - c = number of connected components
  - g = genus
  - b = number of boundary loops

$\chi($  $) = 2$ $\chi($  $) = 0$ $\chi($  $) = ?$

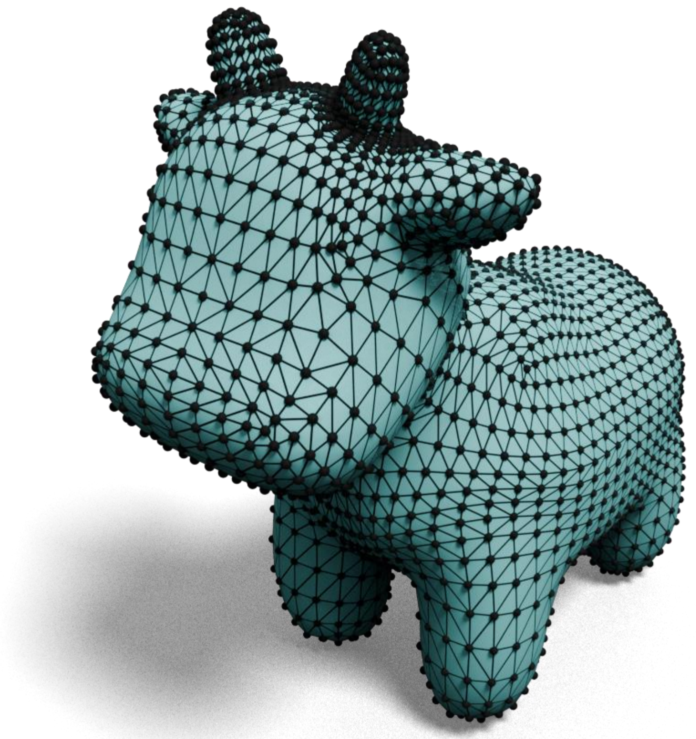
Implication for Mesh Storage

- Let‘s count the edges and faces in a closed **triangle mesh:**
  - Ratio of edges to faces: $\boldsymbol{e} = 3/2\,\boldsymbol{f}$
    - each edge belongs to exactly 2 triangles
    - each triangle has exactly 3 edges



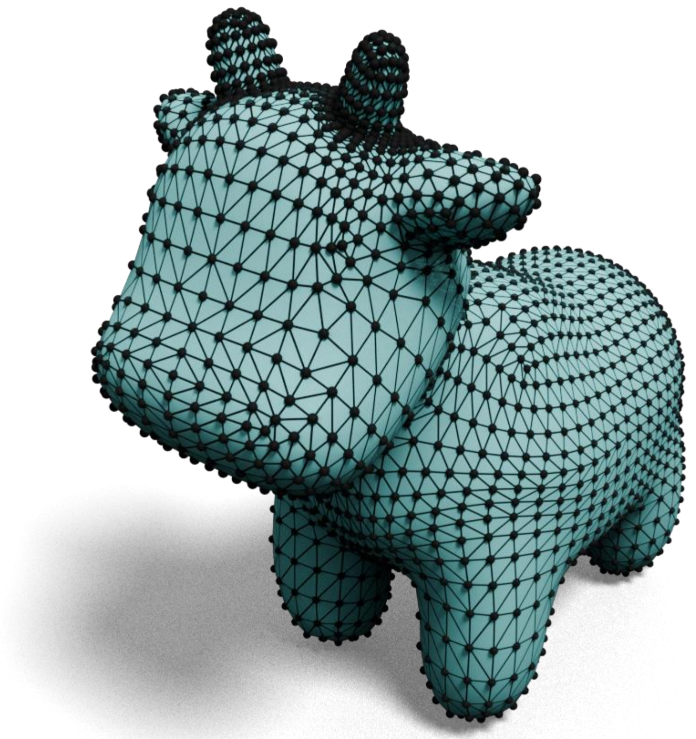
Implication for Mesh Storage

- Let's count the edges and faces in a closed **triangle mesh:**
  - Ratio of edges to faces: $\boldsymbol{e = 3/2\, f}$
    - each edge belongs to exactly 2 triangles
    - each triangle has exactly 3 edges
  - Ratio of vertices to faces: $\boldsymbol{f \sim 2v}$
    - $2 = v - e + f = v - 3/2\, f + f$
    - $2 + f / 2 = v$



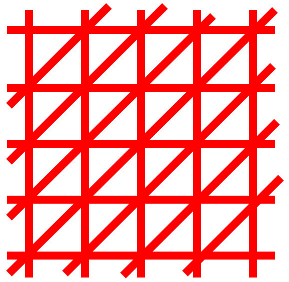
Implication for Mesh Storage

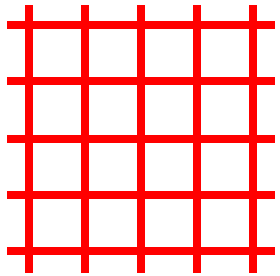
- Let's count the edges and faces in a closed **triangle mesh:**
  - Ratio of edges to faces: $\boldsymbol{e = 3/2\, f}$
    - each edge belongs to exactly 2 triangles
    - each triangle has exactly 3 edges
  - Ratio of vertices to faces: $\boldsymbol{f \sim 2v}$
    - $2 = v - e + f = v - 3/2\, f + f$
    - $2 + f\,/\,2 = v$
  - Ratio of edges to vertices: $\boldsymbol{e \sim 3v}$
  - **Average degree of a vertex: 6**



Regularity

- Triangle mesh: average valence = 6
- Quad mesh: average valence = 4

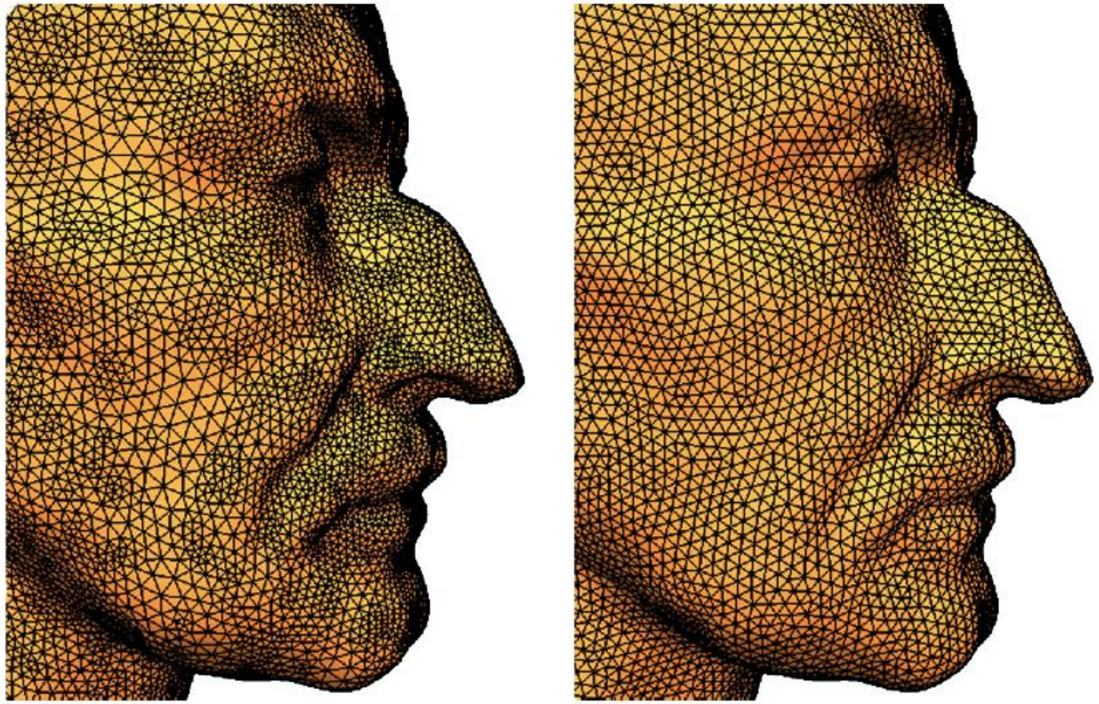




- **Regular mesh**: all faces have the same number of edges and all vertex degrees are equal.
  - Not possible for all topologies
- **Regular mesh with singularities:**
  - all faces have same number of sides;
  - small number of vertices has a different valence (e.g. for quad meshes: degree 3 or 5).

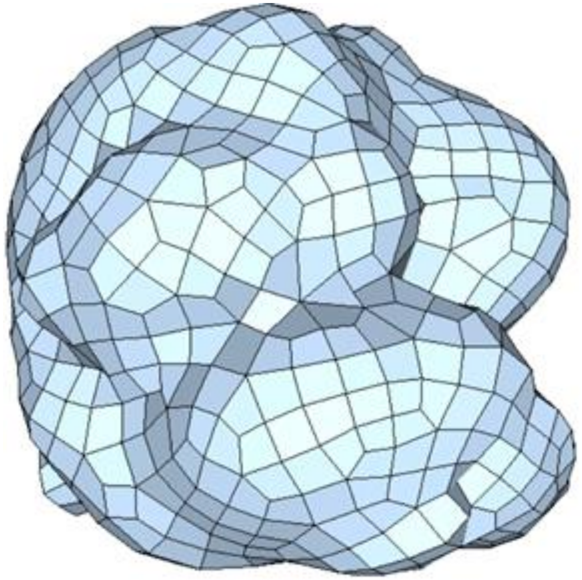
Regularity

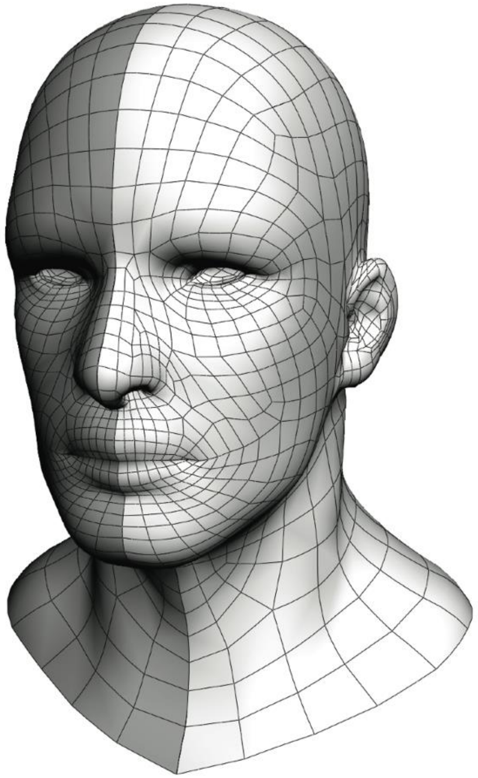
- "Nice mesh" (sometimes colloquially called "regular")



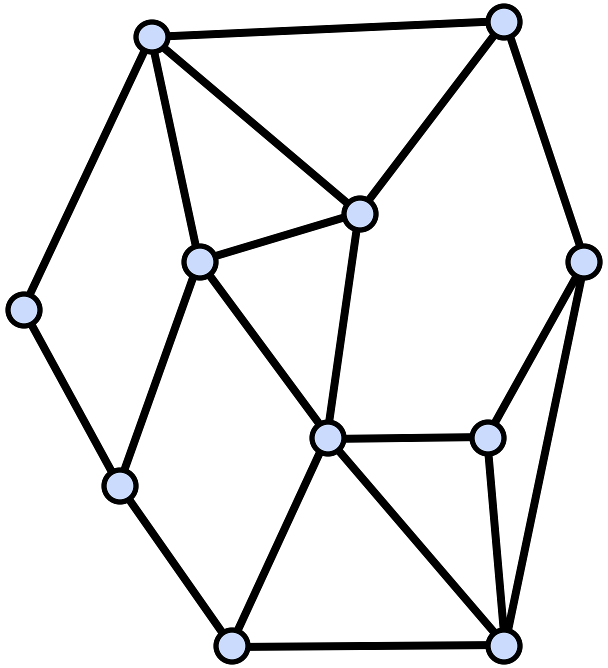
Regularity

- Regular mesh with singularities (different valence)
  - a.k.a. “nearly regular”





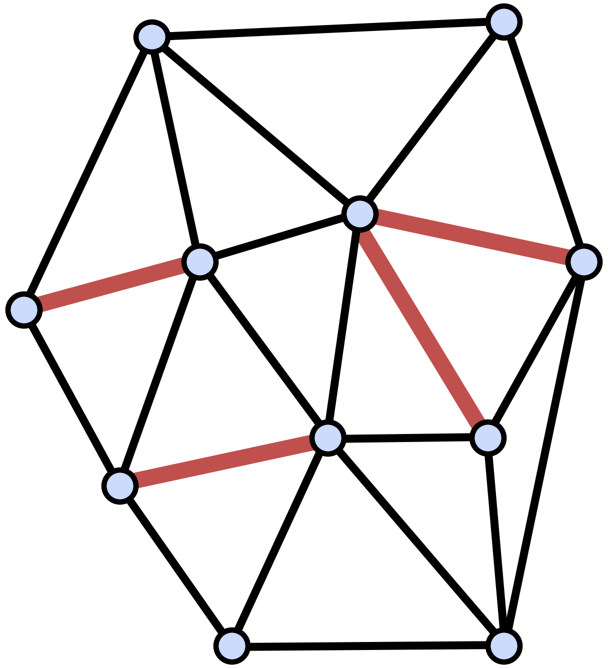
Triangulation



- Polygonal mesh where every face is a triangle

- Simplifies data structures
- Simplifies rendering
- Simplifies algorithms
- Each face planar and convex
- Any polygon can be triangulated

Triangulation



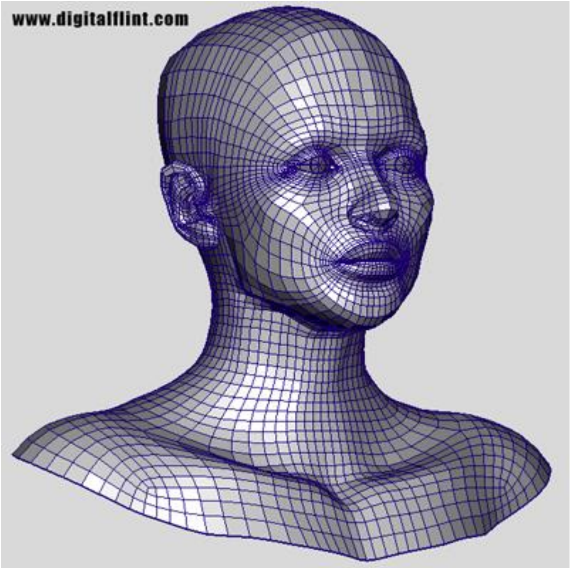
- Polygonal mesh where every face is a triangle

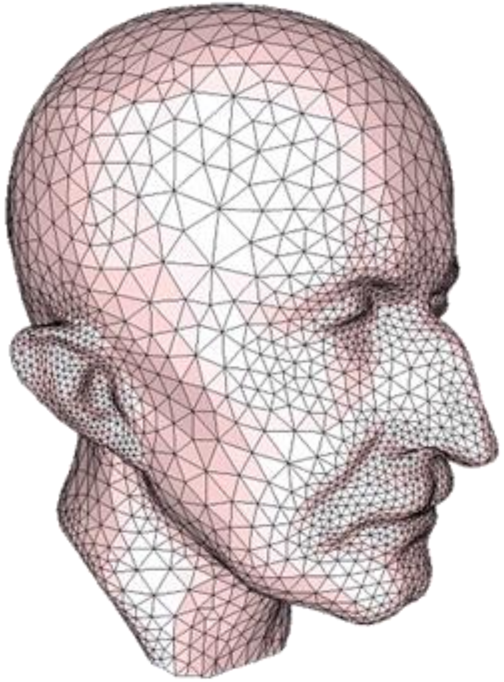
- Simplifies data structures
- Simplifies rendering
- Simplifies algorithms
- Each face planar and convex
- Any polygon can be triangulated

Polygonal vs. Triangle Meshes

- Triangles are flat and convex
  - Easy rasterization, normals
  - Uniformity (same # of vertices)
- 3-way symmetry is less natural

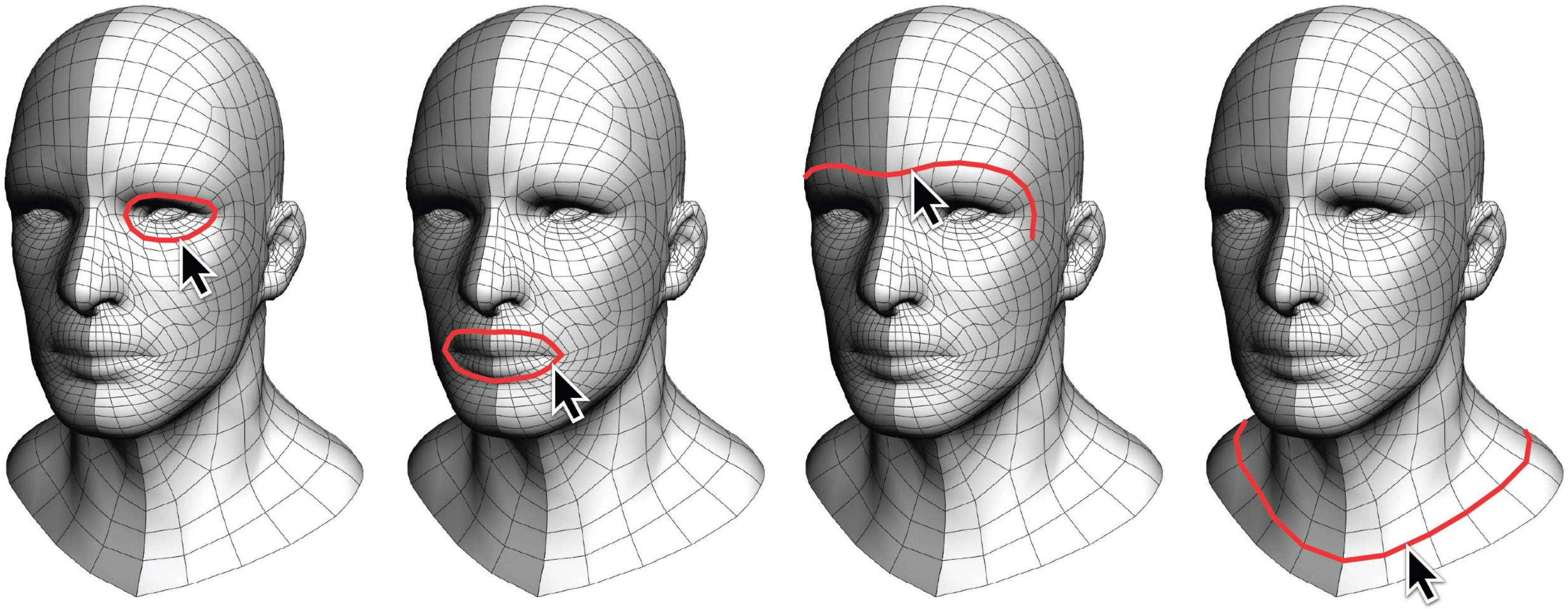
- General polygons are flexible
  - Quads have natural symmetry
- Can be non-planar, non-convex
  - Difficult for graphics hardware
- Varying number of vertices





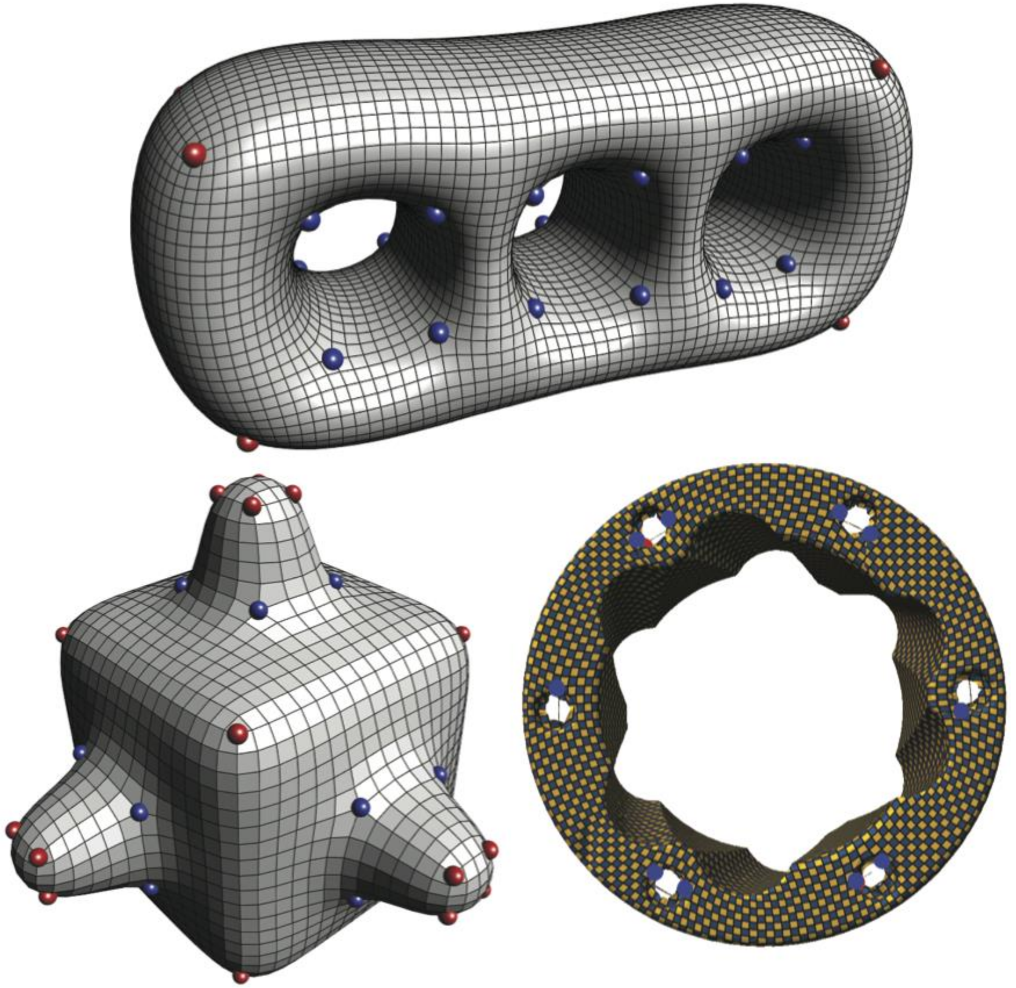
Polygonal vs. Triangle Meshes

- Edge loops are convenient for editing and animation

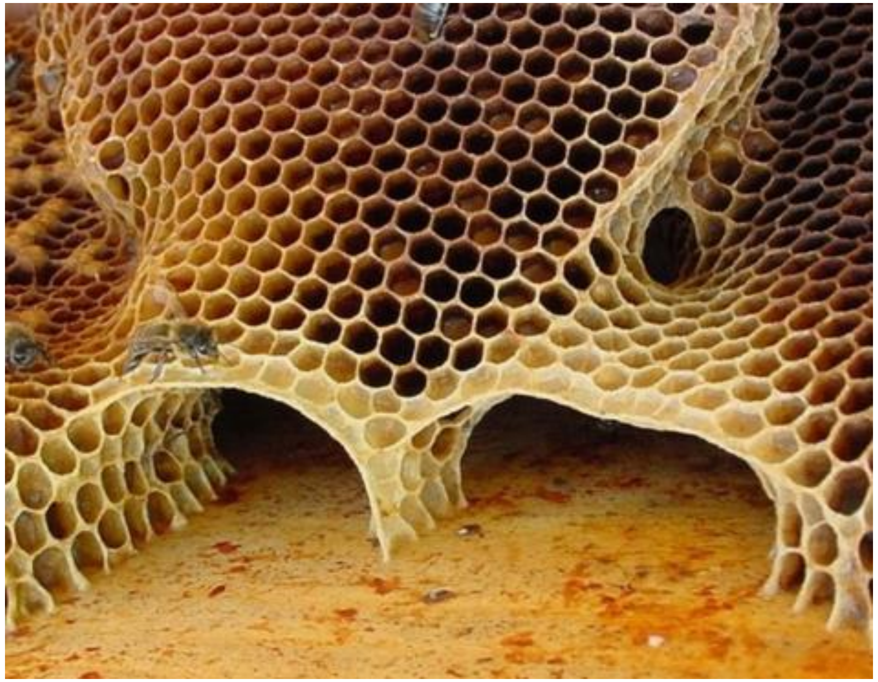


Polygonal vs. Triangle Meshes

- Quality of triangle meshes
  - Uniform area
  - Angles close to 60
- Quality of quadrilateral meshes
  - Number of irregular vertices
  - Angles close to 90
  - *Good edge flow*



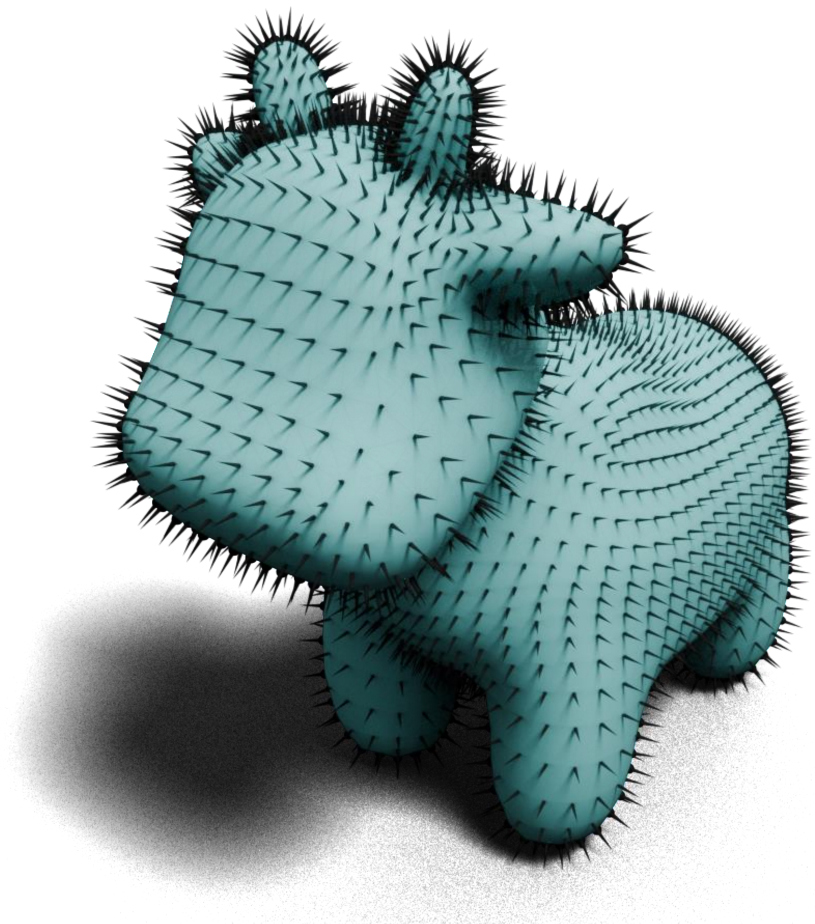
Polygonal (hex) Meshes



E. Van Egeraat

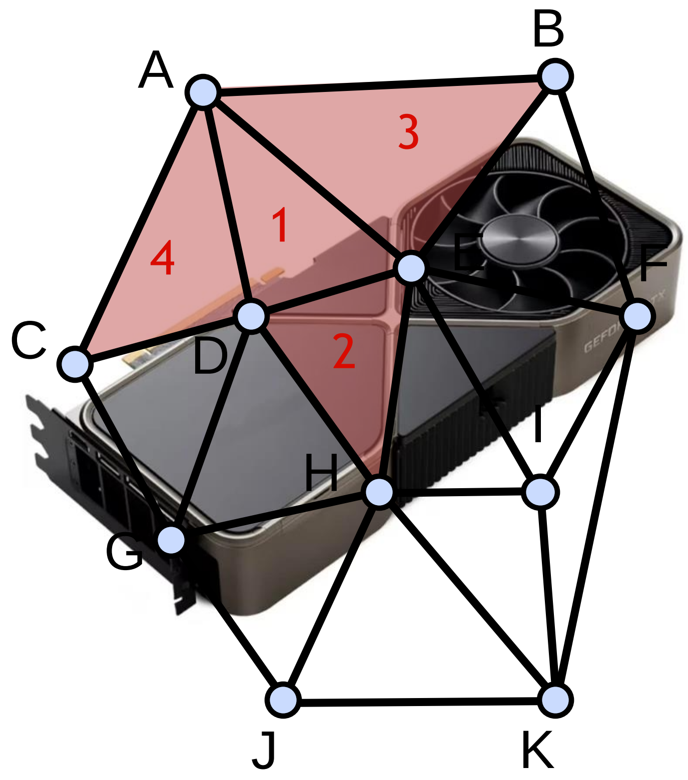
Data Structures

- What should be stored?
  - Geometry: 3D coordinates
  - Connectivity
    - Adjacency relationships
  - Attributes
    - Normal, color, texture coordinates
    - Per vertex, face, edge



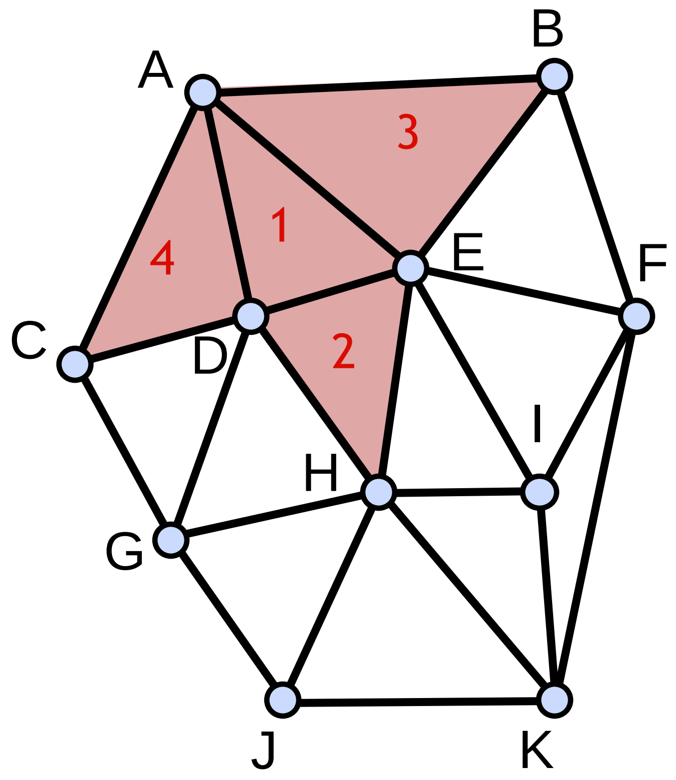
Data Structures

- What should be supported?
  - Rendering
  - Queries
    - What are the vertices of face #2?
    - Is vertex *A* adjacent to vertex *H*?
    - Which faces are adjacent to face #1?
  - Modifications
    - Remove/add a vertex/face
    - Vertex split, edge collapse



Data Structures

- How good is a data structure?
  - Time to construct
  - Time to answer a query
  - Time to perform an operation
  - Space complexity
  - Redundancy
- Criteria for design
  - Expected number of vertices
  - Available memory
  - Required operations
  - Distribution of operations



Triangle List

- STL format (used in CAD)
- Storage
  - Triangular face: 3 positions
  - 4 bytes per coordinate
  - 36 bytes per face
    - Euler: $f = 2v$
    - $72*v$ bytes for a mesh with v vertices
- No connectivity information

Triangles			
0	x0	y0	z0
1	x1	x1	z1
2	x2	y2	z2
3	x3	y3	z3
4	x4	y4	z4
5	x5	y5	z5
6	x6	y6	z6
…	…	…	…

Indexed Face Set

- Used in formats

  OBJ, OFF, WRL…

- Storage
  - Vertex: position
  - Face: vertex indices
  - 12 bytes per vertex (single precision)
  - 12 bytes per face
  - 36*v bytes for the mesh

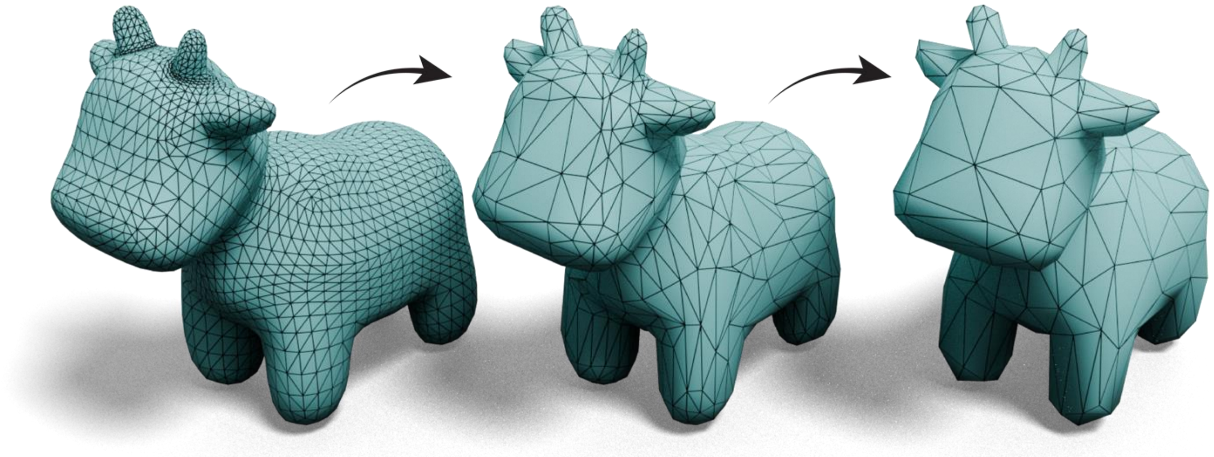
- No *explicit* neighborhood info

Vertices			
v0	x0	y0	z0
v1	x1	x1	z1
v2	x2	y2	z2
v3	x3	y3	z3
v4	x4	y4	z4
v5	x5	y5	z5
v6	x6	y6	z6
…	…	…	…

Triangles			
t0	v0	v1	v2
t1	v0	v1	v3
t2	v2	v4	v3
t3	v5	v2	v6
…	…	…	…

Indexed Face Set: Problems

- Information about neighbors is not explicit
  - Finding neighboring vertices/edges/faces costs $O(V)$ time!
  - Local mesh modifications cost $O(V)$

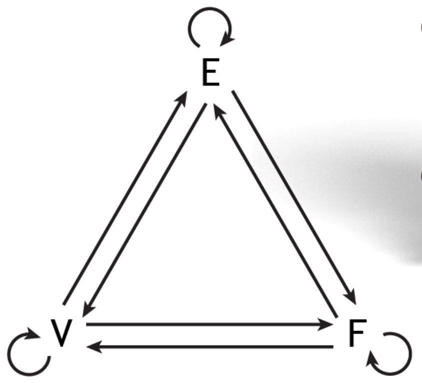


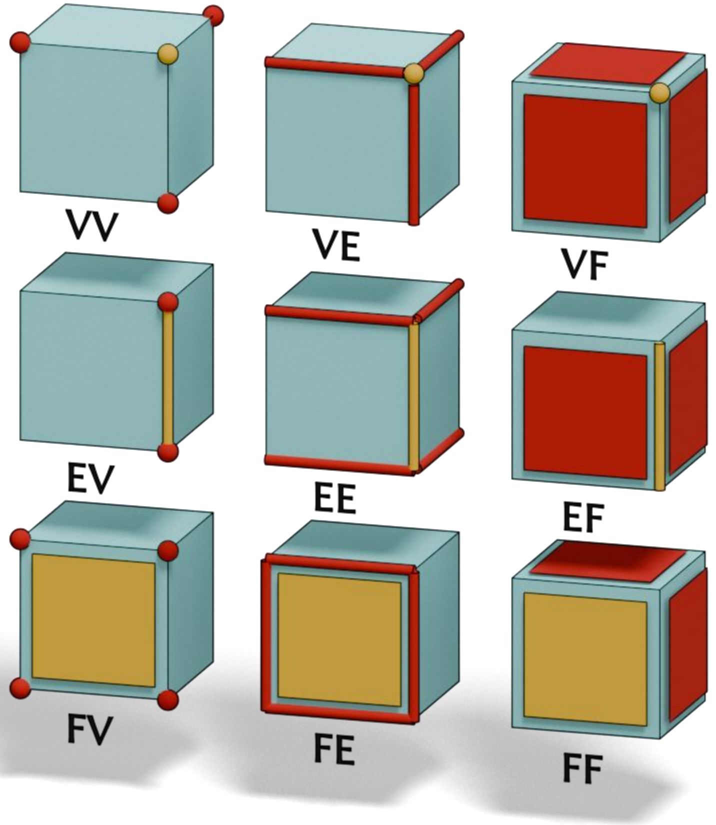
  - Breadth-first search costs $O(kV)$ where k = # found vertices

Neighborhood Relations

- All possible neighborhood relationships:
  1. Vertex – Vertex VV
  2. Vertex – Edge VE
  3. Vertex – Face VF
  4. Edge – Vertex EV
  5. Edge – Edge EE
  6. Edge – Face EF
  7. Face – Vertex FV
  8. Face – Edge FE
  9. Face – Face FF

We'd like $O(1)$ time for queries and local updates of these relationships





The Classics

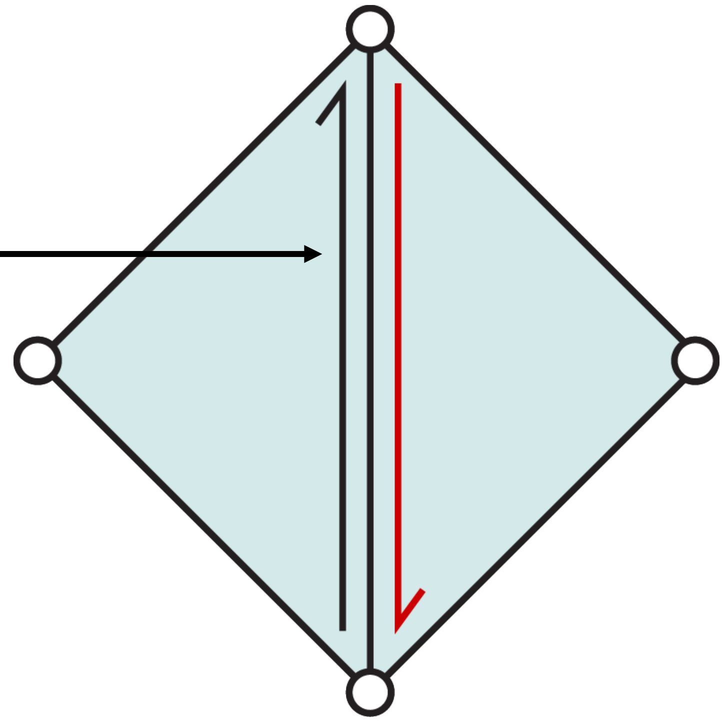
- Which data structure?
  - O(1) query for adjacency
  - O(1) insertion, deletion

Linked List

Halfedge data structure

- Split edges in **oriented halfedges**
  - New 'core' element

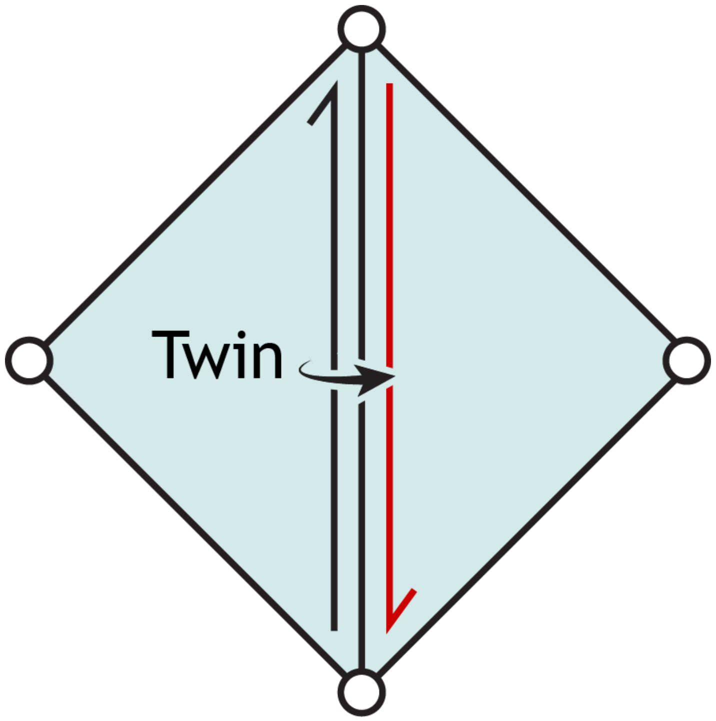
```
struct Halfedge {

};
```



Halfedge data structure

- Split edges in **oriented halfedges**
  - New ‘core’ element

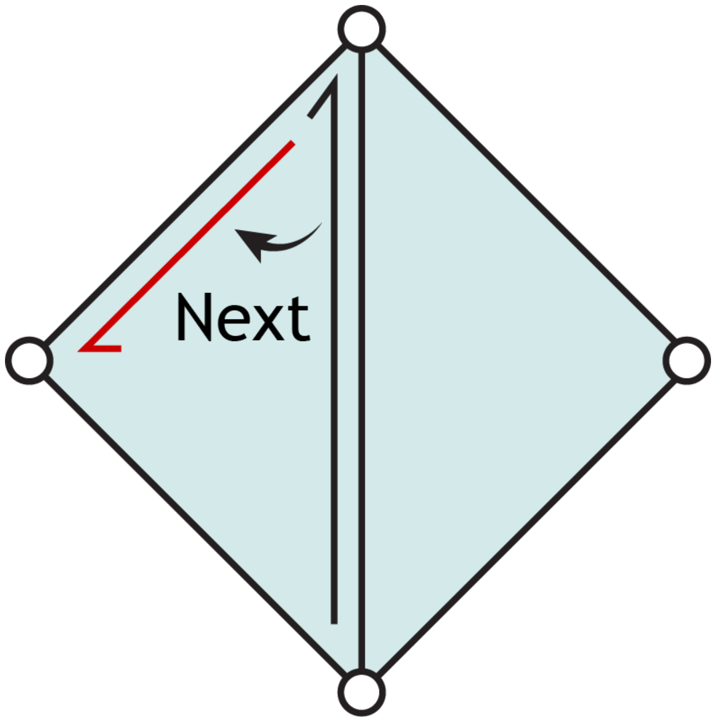
```
struct Halfedge {
    Halfedge* twin;



};
```



Halfedge data structure

- Split edges in **oriented halfedges**
  - New 'core' element

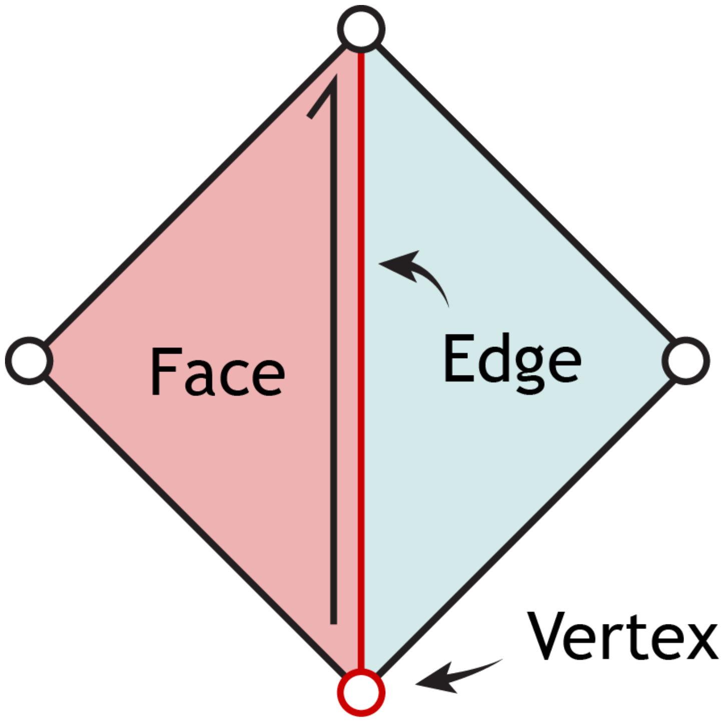
```
struct Halfedge {
    Halfedge* twin;
    Halfedge* next;



};
```



Halfedge data structure

- Split edges in **oriented halfedges**
  - New 'core' element

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



Halfedge data structure

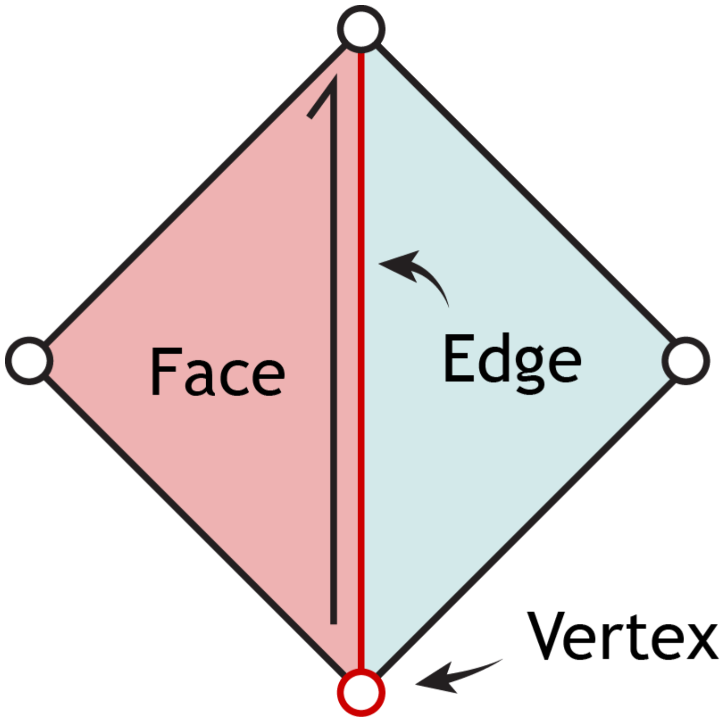
- Split edges in ori
  - New 'core' eleme

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

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

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

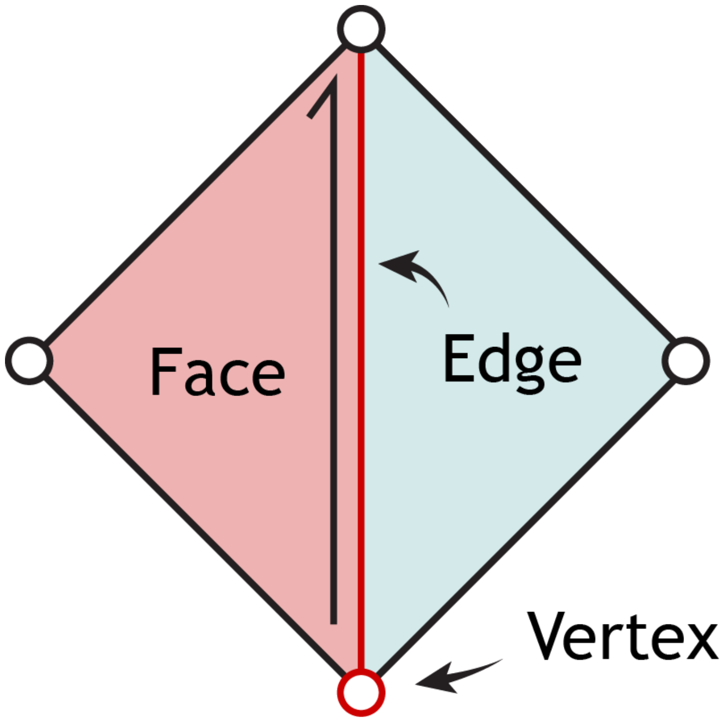
```
struct Face {
    Halfedge* halfedge;
};
```



Halfedge data structure

- Split edges in **oriented halfedges**
  - New ‘core’ element

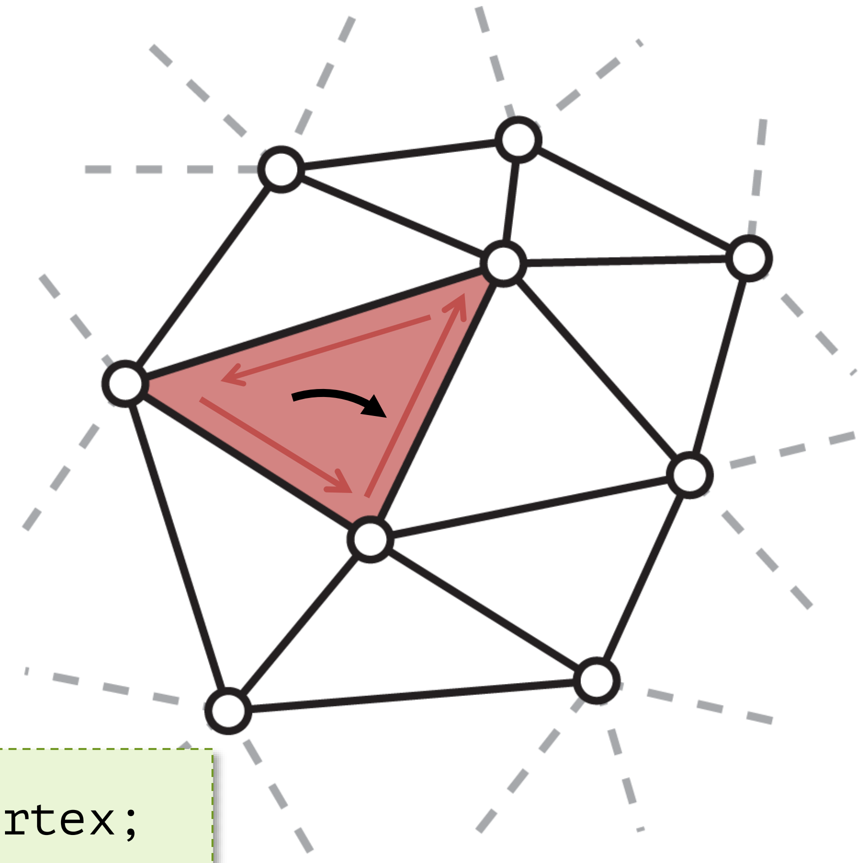
```
struct Halfedge {
    Halfedge* twin;
    Halfedge* next;
    Vertex* vertex;
    Edge* edge;
    Face* face;
};
```



Easy to traverse

- Over a face
  - `face`
  - `halfedge`
  - `next`
  - `next`
- Vertices?

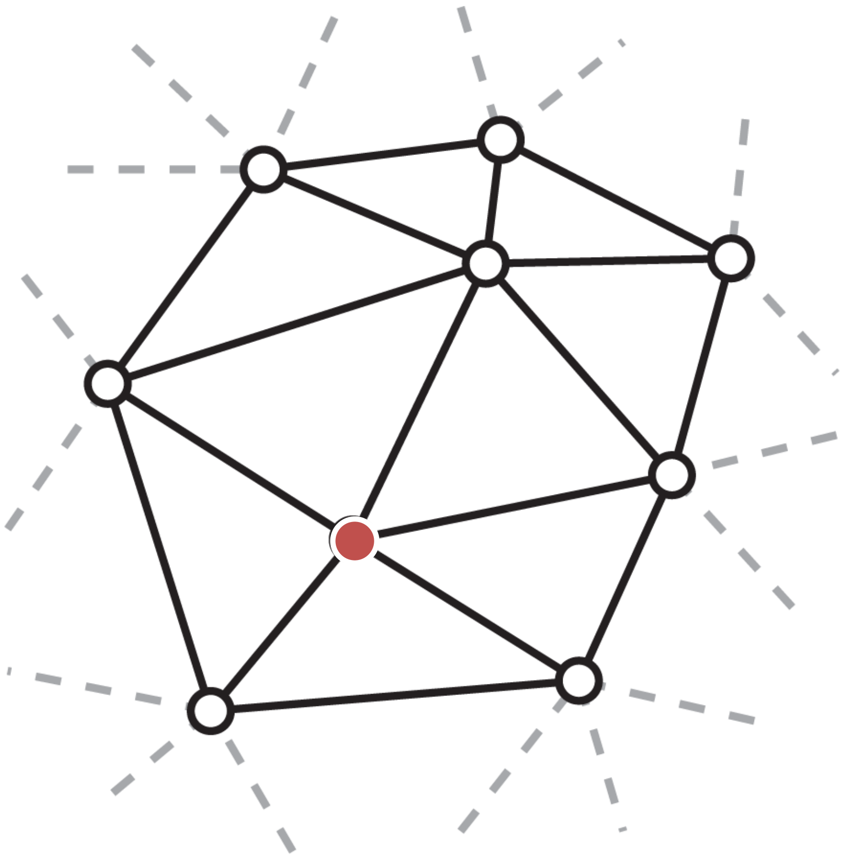
```
Vertex v = halfedge.vertex;
```



Easy to traverse

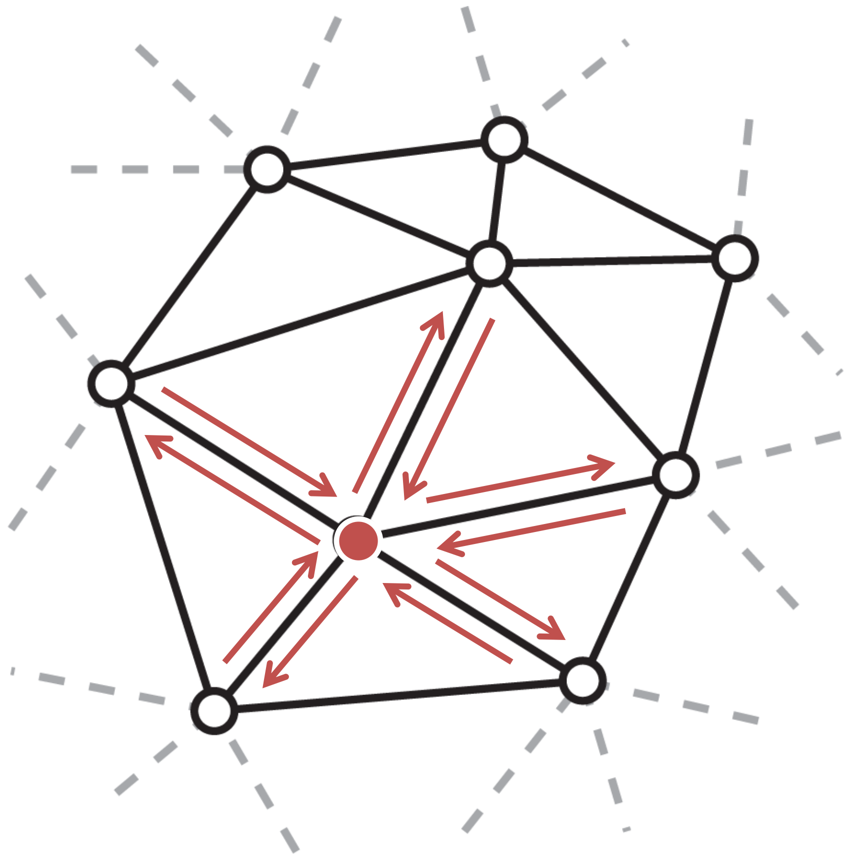
- Around a vertex?

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



Easy to traverse

- Around a vertex?
  - `halfedge`
  - `twin`
  - `next`
  - `twin`
  - `next`
  - …

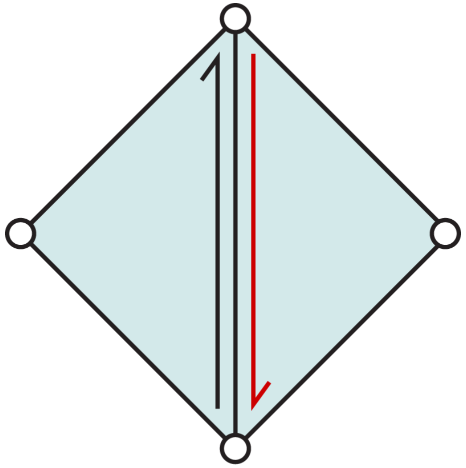


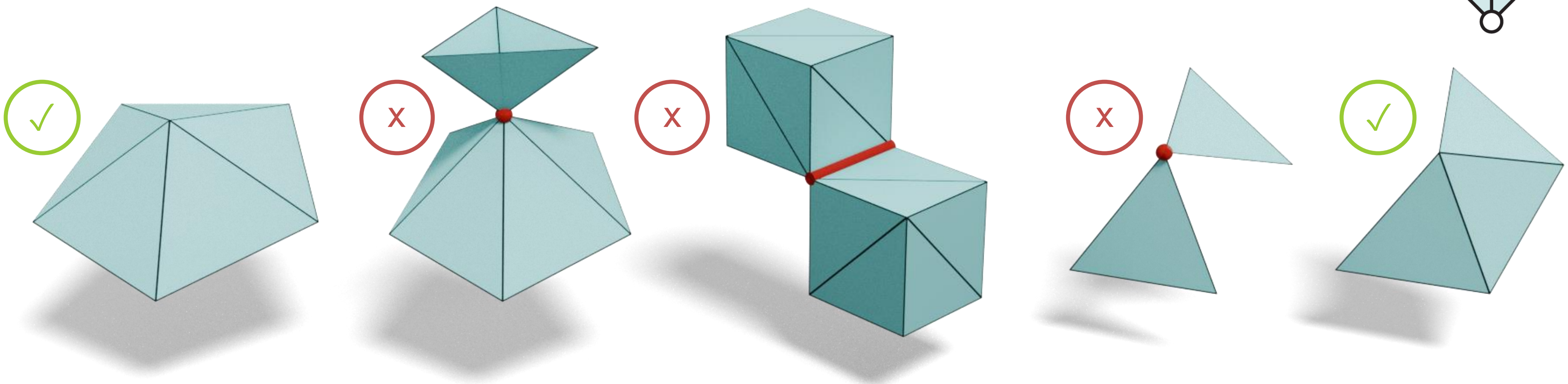
Halfedge data structure

- Pros: (assuming bounded vertex valence)
  - $O(1)$ time for neighborhood relationship queries
  - $O(1)$ time and space for local modifications (edge collapse, vertex insertion…)
- Cons:
  - Heavy – requires storing and managing extra pointers.
  - Not as trivial as Indexed Face Set for rendering with GPUs

Manifold...

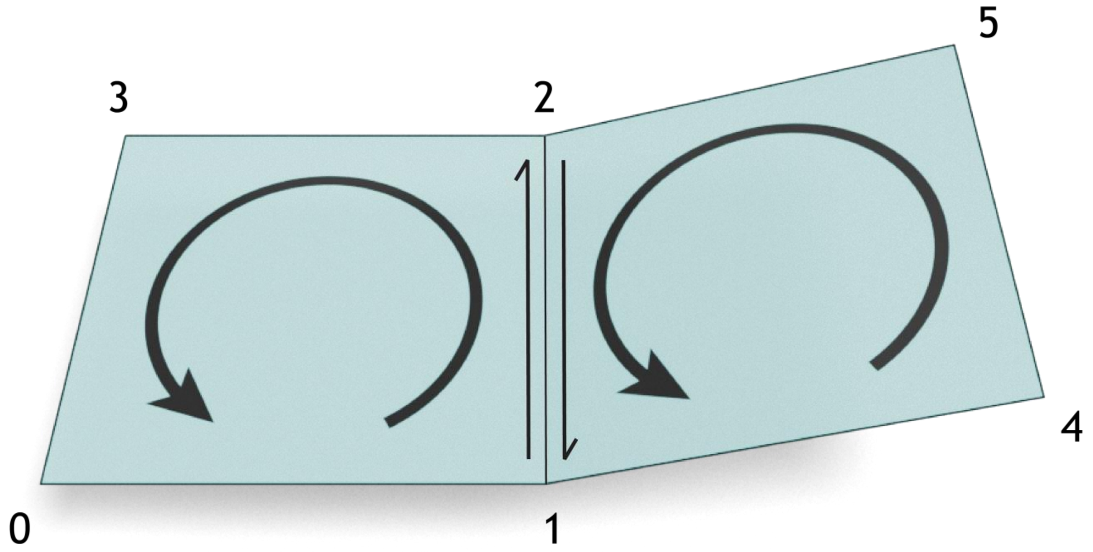
- At most two faces on an edge
- Each vertex has only one halfedge

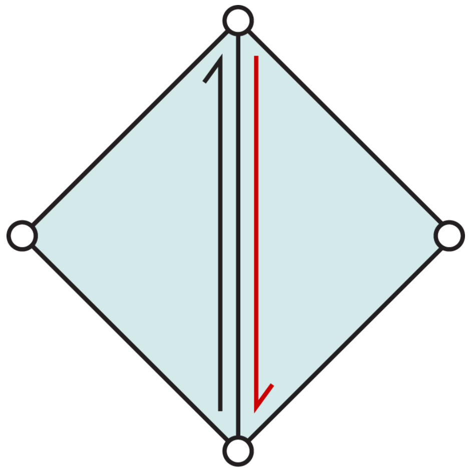




Manifold and oriented

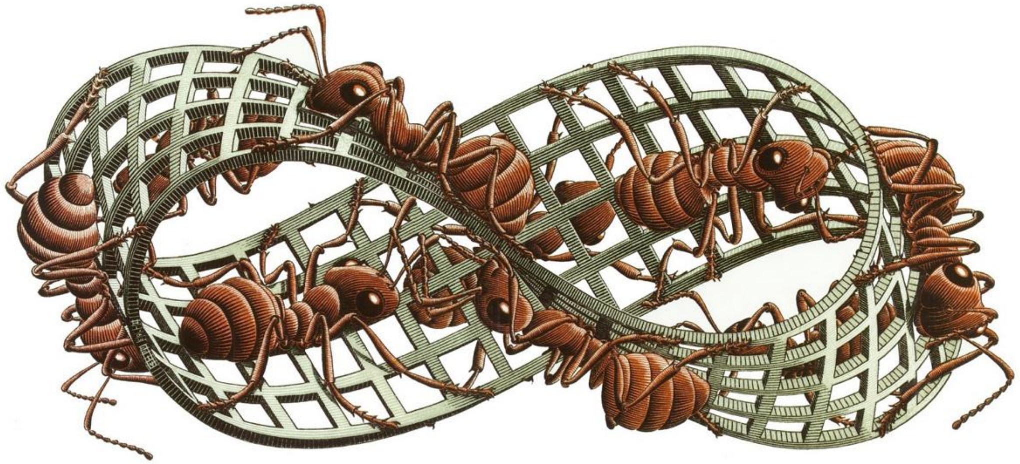
- Data structure guarantees orientation



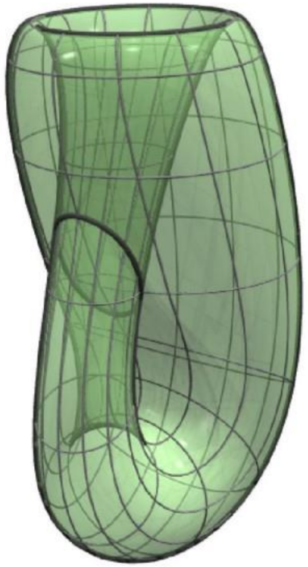


Does it halfedge?

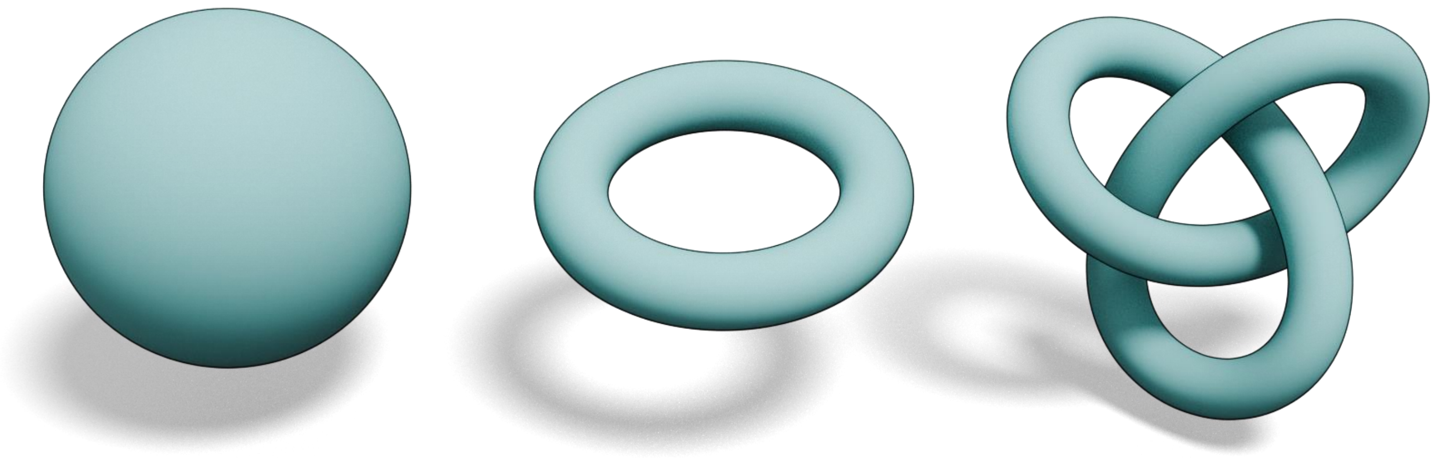
Möbius strip



Klein bottle



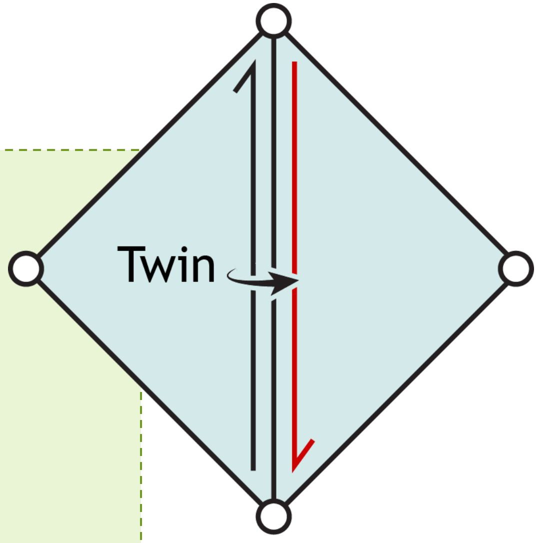
Minimum number of halfedges?



Halfedge Libraries

- CGAL
  - www.cgal.org
  - Computational geometry
- OpenMesh
  - www.openmesh.org
  - Mesh processing
- Geometry Central
  - www.geometry-central.net

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



- **Not used in class.**
- Instead, Indexed Face Set augmented with tables for fast queries.

Thank you