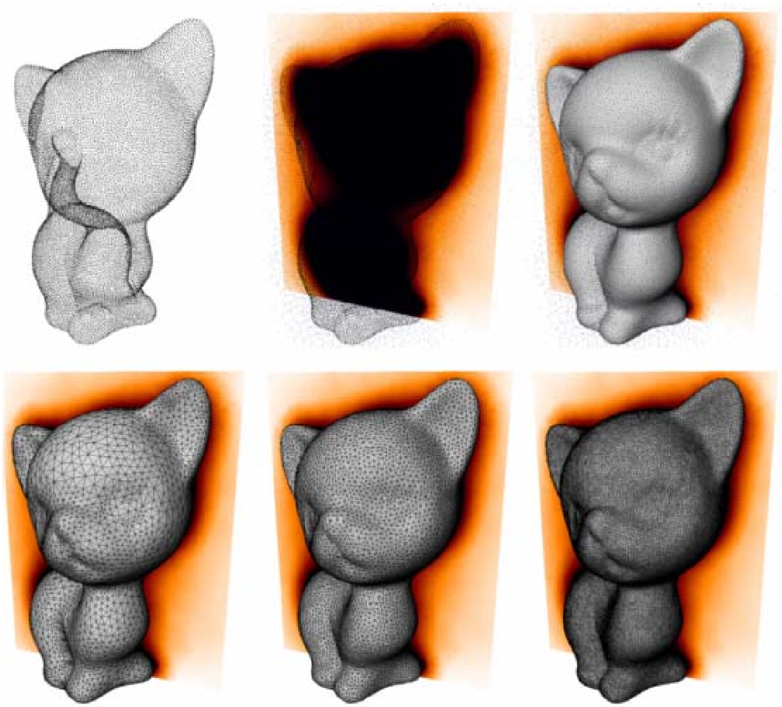
G22.3033-008, Spring 2010

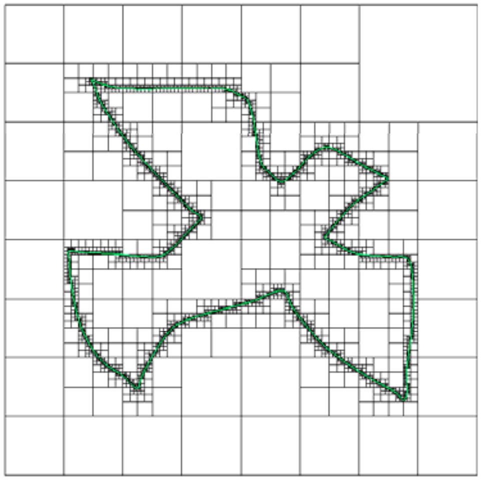
Geometric Modeling

Shape Acquisition and Meshes

Course Topics

- Shape acquisition
  - Scanning/imaging
  - Reconstruction





Data 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 and geometric filtering
- Remeshing
- Compression

Data Acquisition

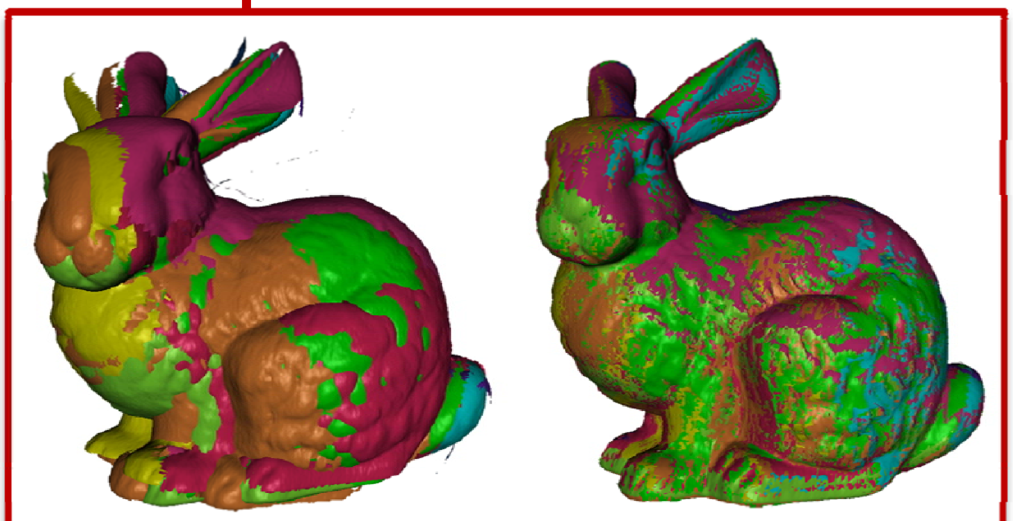
Pipeline

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 and geometric filtering
- Remeshing
- Compression

Data 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 and geometric filtering
- Remeshing
- Compression

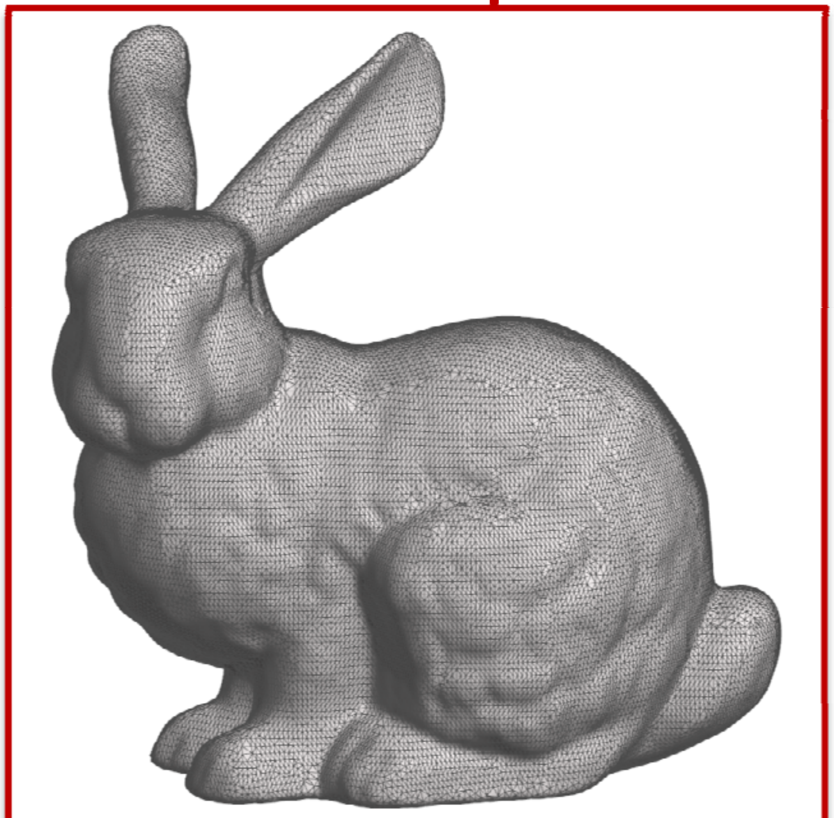


Data Acquisition
Pipeline

Scanning: results in range images

Registration: bring all range images to one coordinate system

Stitching/reconstruction: Integration of scans into a single mesh



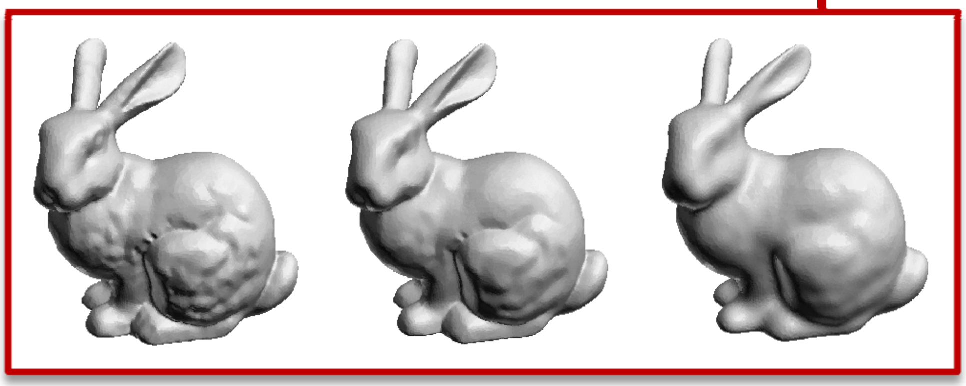
Postprocess:
- Topological and geometric filtering
- Remeshing
- Compression

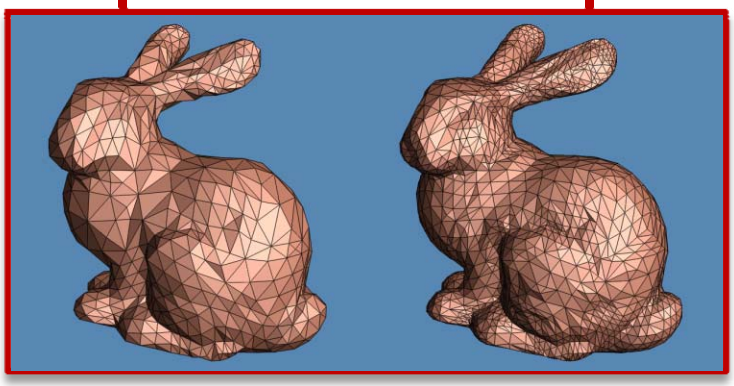
Data Acquisition

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 and geometric filtering
- Remeshing
- Compression





Touch probes

- 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

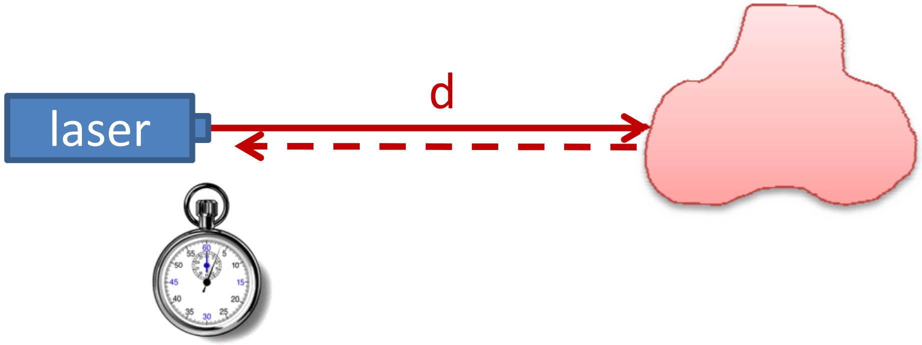
Optical scanning

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

Optical scanning – active lighting

Time of flight laser

- Laser rangefinder (lidar)
- Measures the time it takes the laser beam to hit the object and come back
- Scans one point at a time; mirrors used to change beam direction

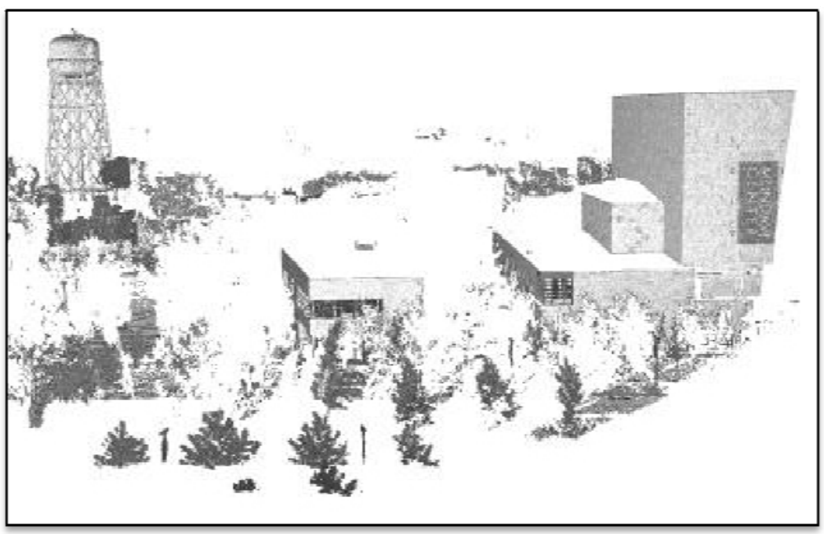


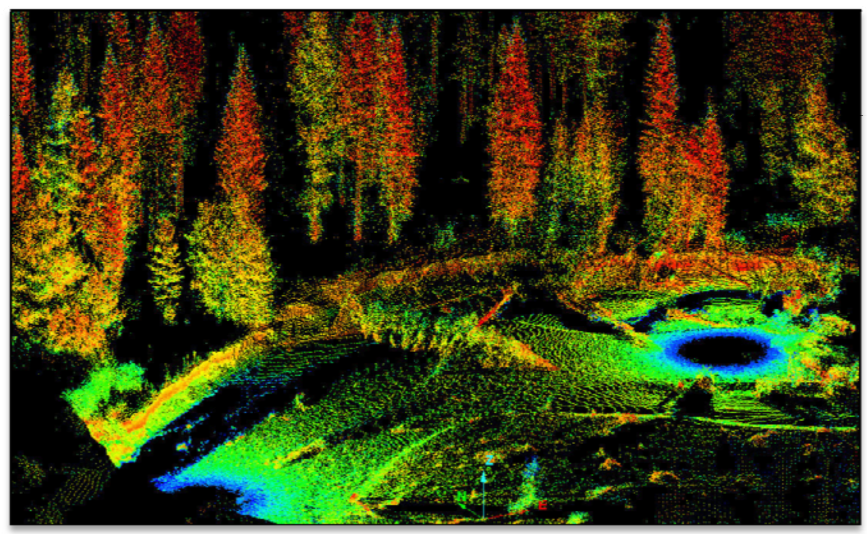
$d = 0.5\, t \cdot c$

Optical scanning – active lighting

Time of flight laser

- Accommodates large range – up to several miles (suitable for buildings, rocks)
- Lower accuracy (light travels really fast)

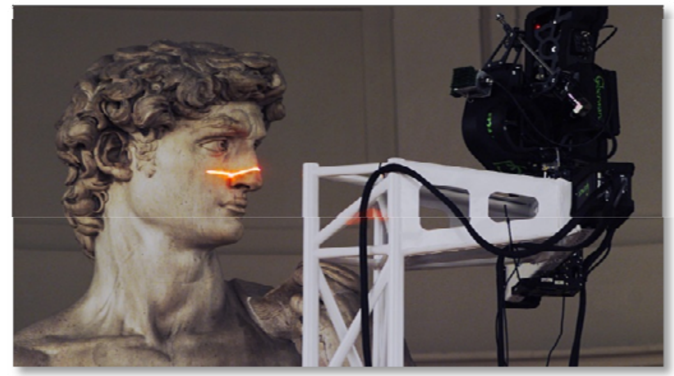


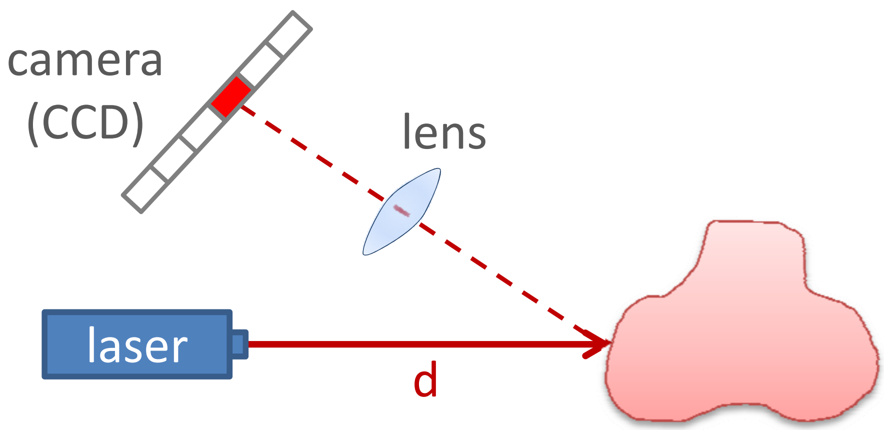


Optical scanning – active lighting

Triangulation laser

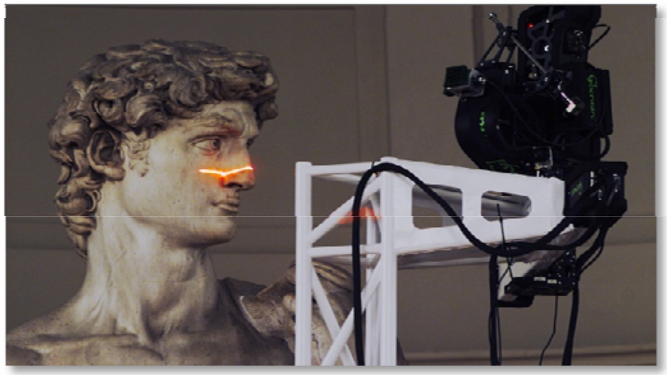
- Laser beam and camera
- Laser dot is photographed
- The location of the dot in the image allows triangulation – so we get the distance to the object



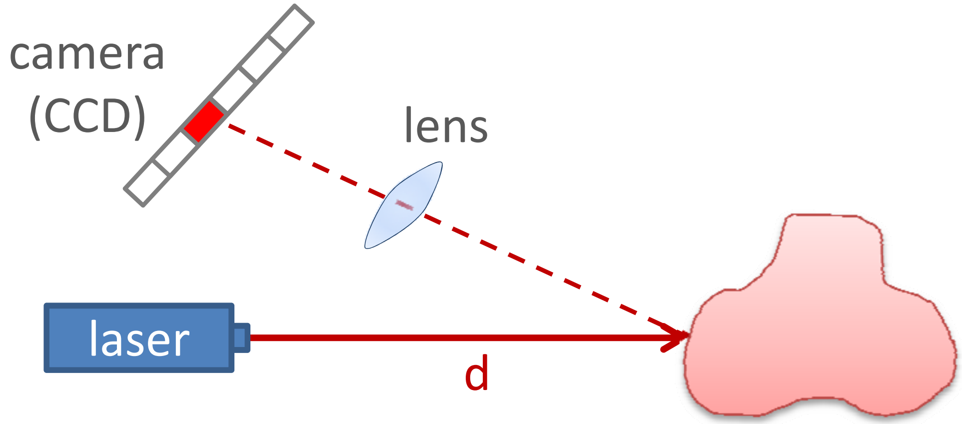


Optical scanning – active lighting

Triangulation laser



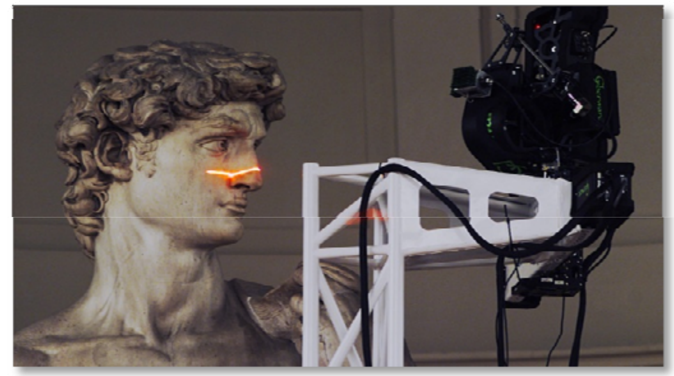
- Laser beam and camera
- Laser dot is photographed
- The location of the dot in the image allows triangulation – so we get the distance to the object

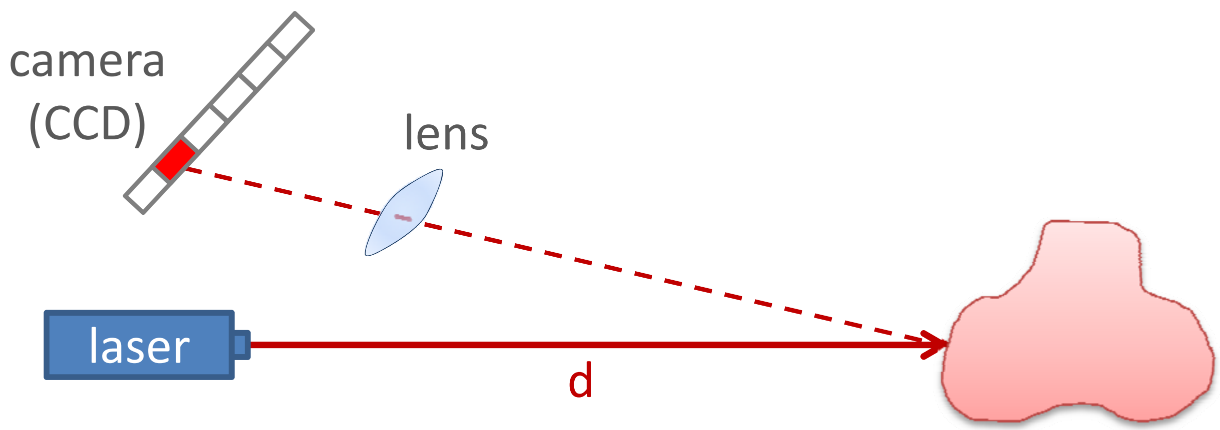


Optical scanning – active lighting

Triangulation laser

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

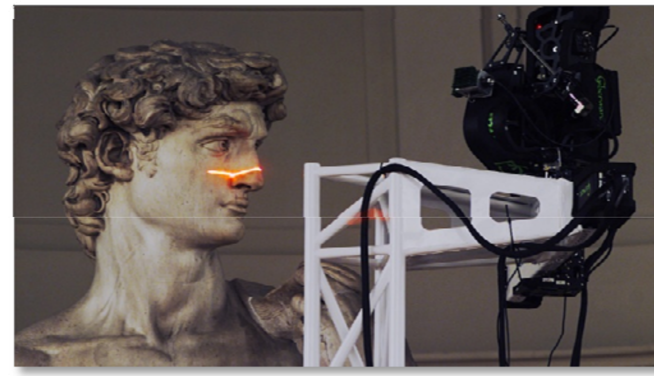




Optical scanning – active lighting

Triangulation laser

- Very precise (tens of microns)
- Small distances (meters)



Optical scanning – active lighting

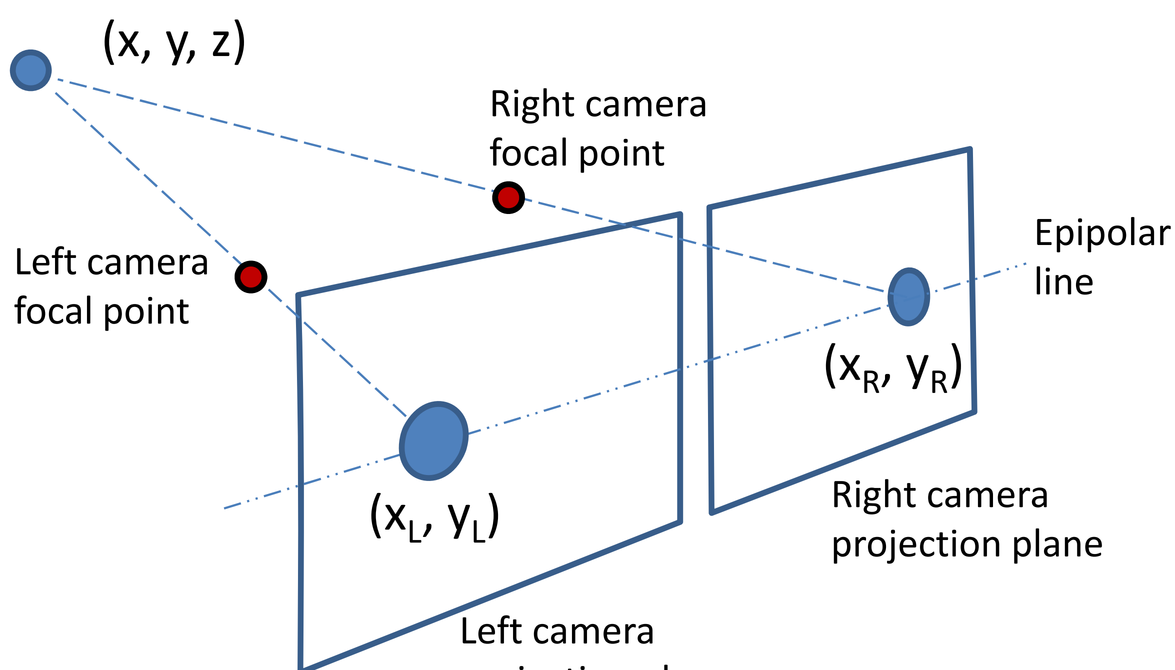
Structured light

- Pattern of visible light is projected onto the object
- The distortion of the pattern, recorded by the camera, provides geometric information
- Very fast – 2D pattern at once, not single dots/lines
  - Even in real time
- Complex distance calculation, prone to noise

Optical scanning – passive

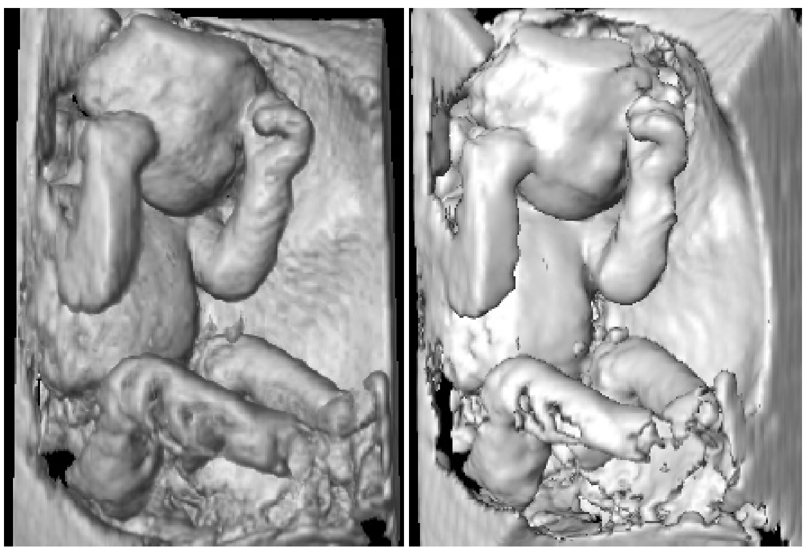
Stereo

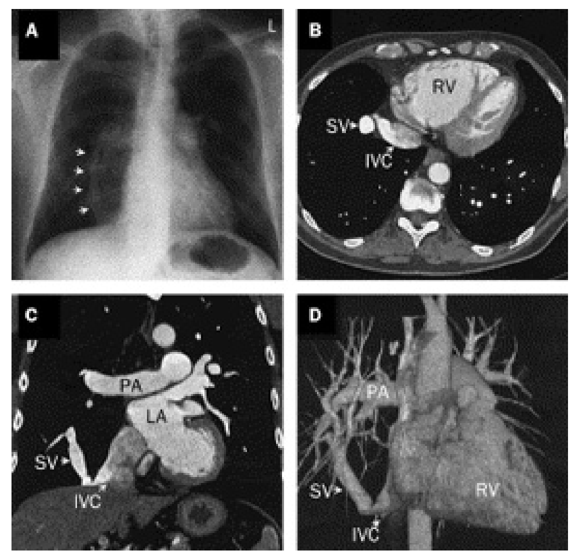
- No need for special lighting/radiation
- Two (or more) cameras
- Feature matching and triangulation



Imaging

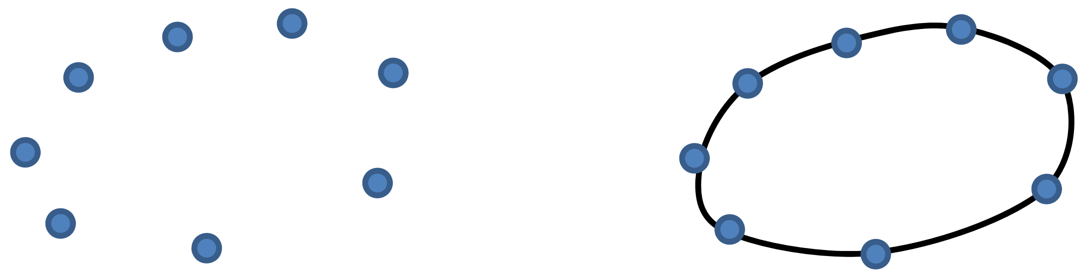
- Ultrasound, CT, MRI
- Discrete volume of density data
- First need to segment the desired object (contouring)





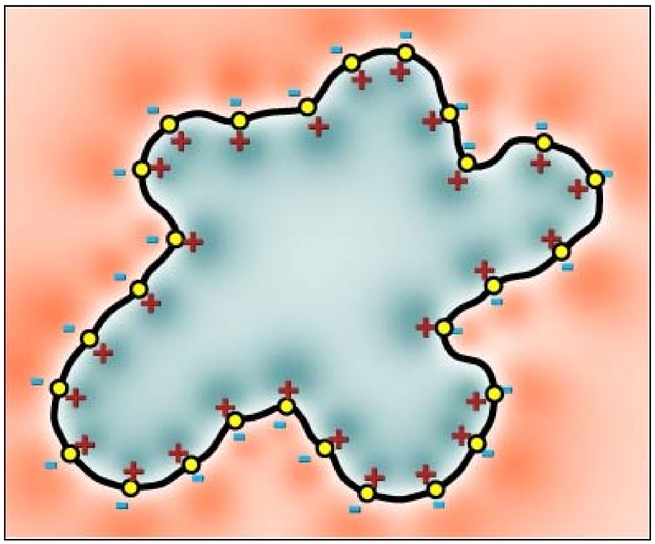
Surface reconstruction

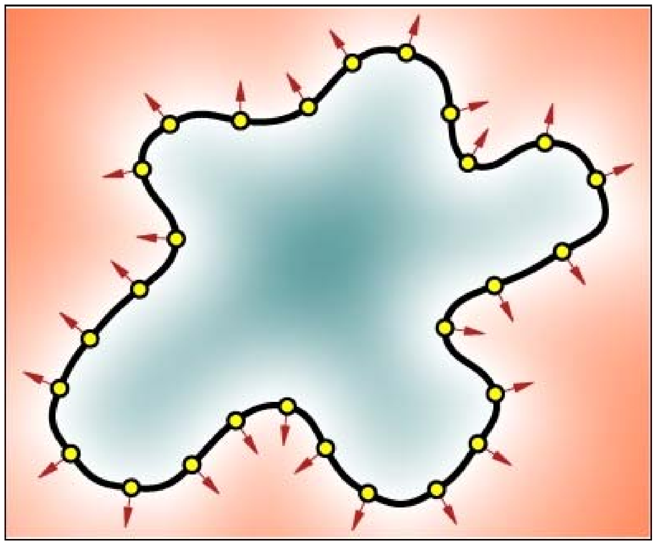
- How to create a single mesh?
  - Surface topology?
  - Smoothness?
  - How to connect the dots?



Distance Field or Implicit Function

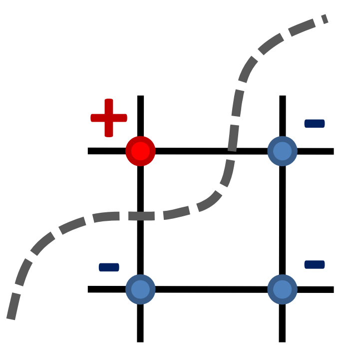
- Fit a function to the point data, such that it's positive inside, negative outside and zero on the surface



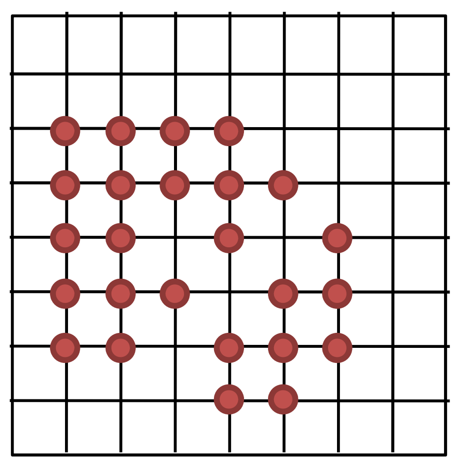


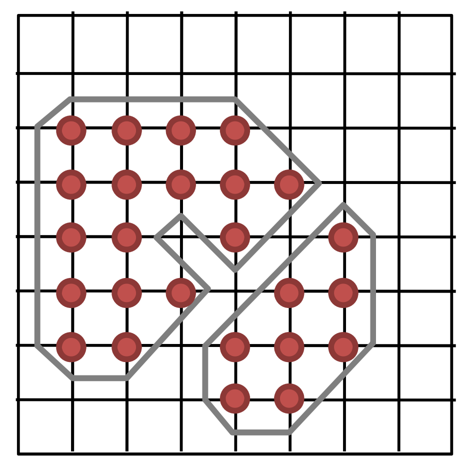
Tessellation of the implicit function

- Want to approximate an implicit surface with a mesh
- Can‘t explicitly compute all the roots
  - Infinite amount (the whole surface)
  - The expression of the implicit function may be complicated
- Solution: find approximate roots by trapping the implicit surface in a grid (lattice)



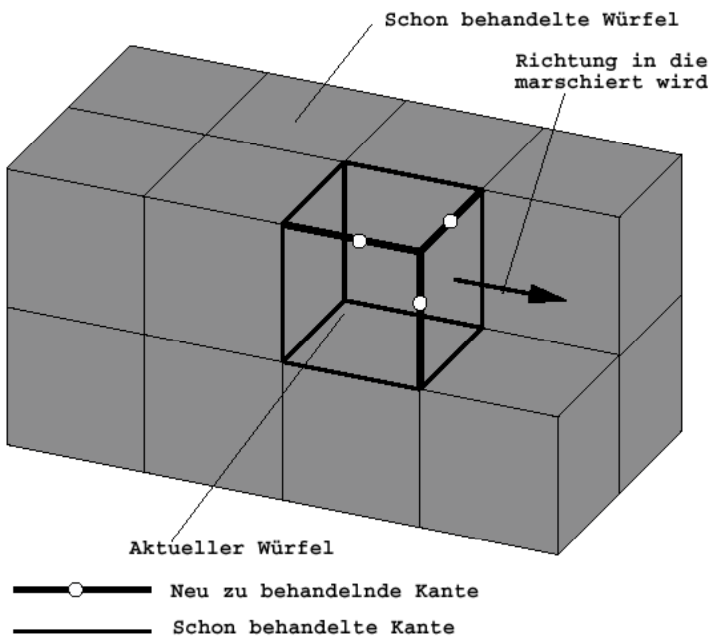
• $f(\mathbf{p}) > 0$

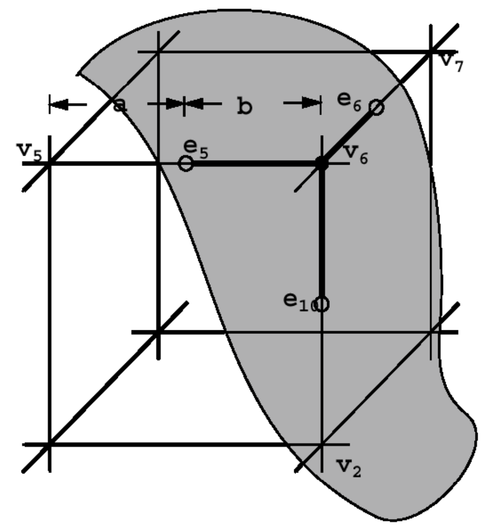




Tessellation

3D – Marching Cubes

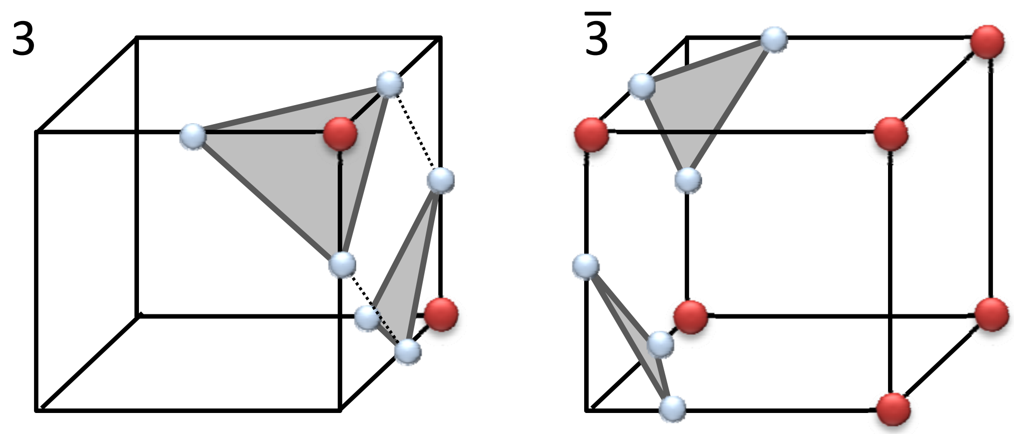




Tessellation

3D – configurations, consistency

- Have to make consistent choices for neighboring cubes
- Prevent " holes" in the triangulation



Surface reconstruction

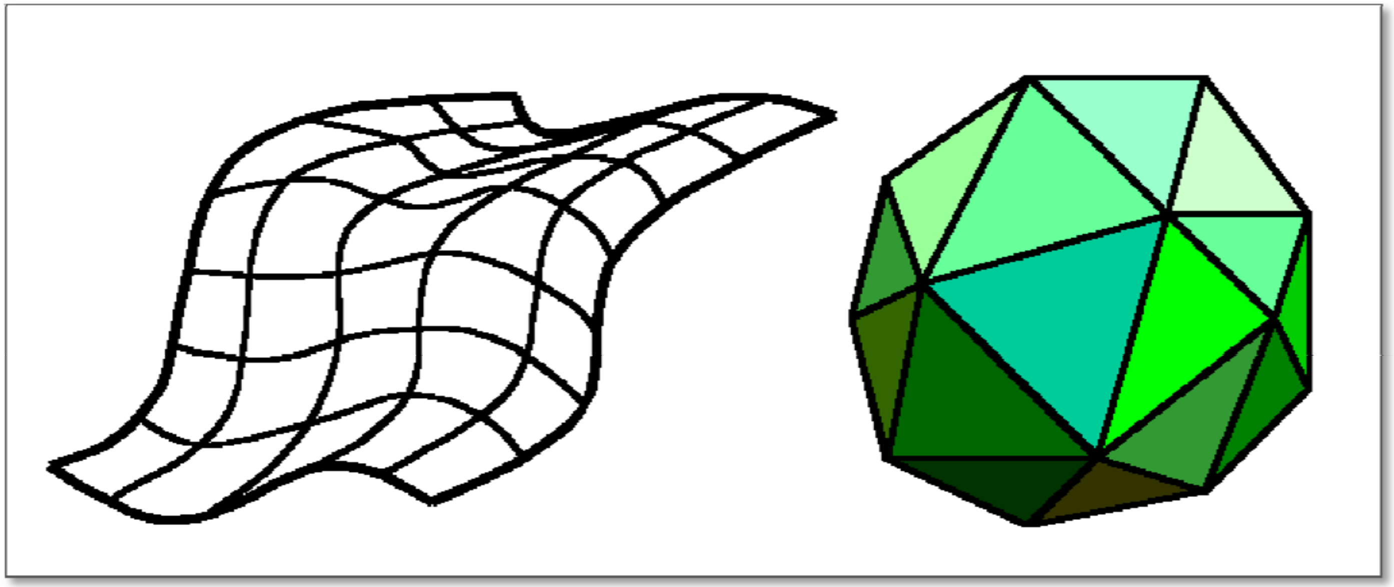
- How to compute the implicit function?
- Details of the Marching Cubes algorithm?

- Next lecture

Polygonal Meshes

Polygonal Meshes

- Boundary representations of objects
  - Surfaces, polyhedrons

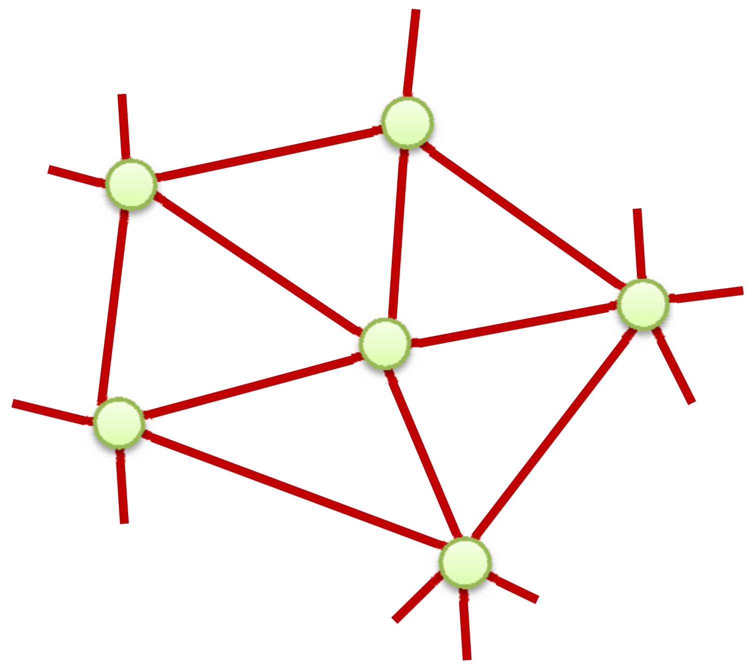


  - How are these objects stored?

Definitions

Geometric graph

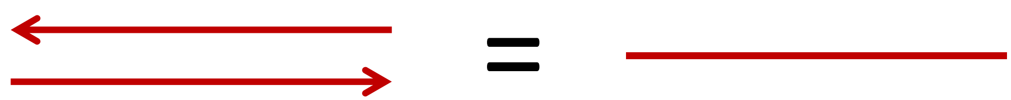
- A graph is a pair G=(V, E)
  - V is a set of *n* distinct vertices $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{n\text{-}1}$
  - E is a set of edges $(\mathbf{v}_i, \mathbf{v}_j)$
- If $V \subset R^d$ with $d \geq 2$, then G=(V, E) is a *geometric graph*
- The *degree* or *valence* of a vertex describes the number of edges incident to this vertex



Definitions

Edges

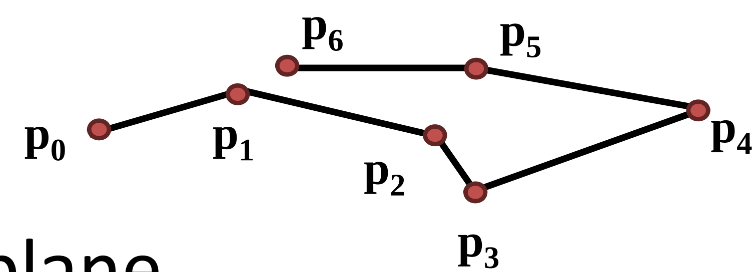
- Two edges are neighbors if they share a common vertex
- Edges are generally not oriented, and are noted as $(\mathbf{v}_i , \mathbf{v}_j)$
- Halfedges are edges with added orientation
- An edge is comprised of two halfedges



Definitions

Polygon

- A geometric graph $Q=(V,E)$ with $E=\{(\mathbf{v}_0, \mathbf{v}_1), (\mathbf{v}_1, \mathbf{v}_2), \ldots, (\mathbf{v}_{n-2}, \mathbf{v}_{n-1})\}$ is a *polygon*

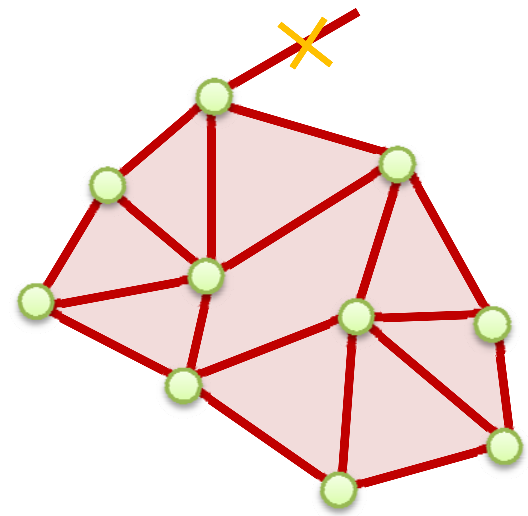


- A polygon is
  - Planar, if all vertices lie on a plane
  - Closed, if $\mathbf{p}_0 = \mathbf{p}_{n-1}$
  - Simple, if the polygon does not self-intersect

Definitions

Polygonal mesh

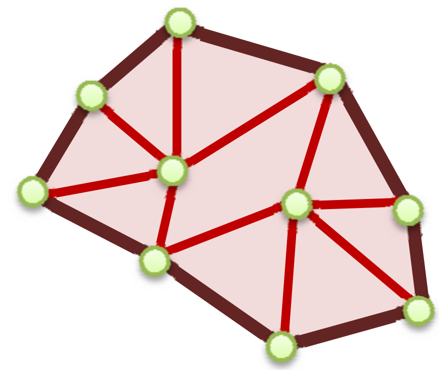
- A finite set M of closed, simple polygons Q_i is a **polygonal mesh** if:
  - The intersection of enclosed regions of any two polygons in M is empty
  - The intersection of two polygons in M is either empty, a vertex $\mathbf{v} \in \mathrm{V}$ or an edge $\mathbf{e} \in \mathrm{E}$
  - Every edge belongs to at least one polygon



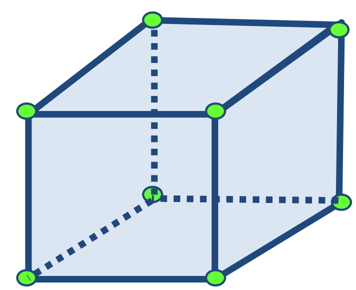
Definitions

Polygonal mesh

- The set of all edges that belong to only one polygon is termed the ***boundary*** of the polygonal mesh, and is either empty or forms closed loops



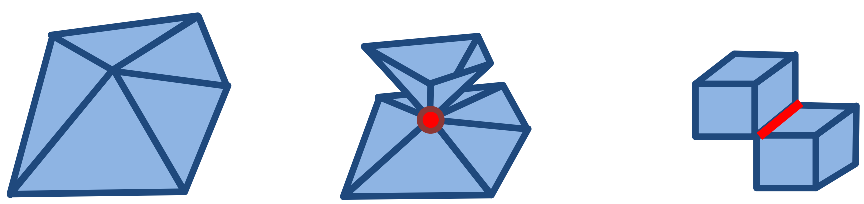
- If the set of edges that belong to only one polygon is empty, then the polygonal mesh is *closed*
- The set of all vertices and edges in a polygonal mesh form a graph



Definitions

Polyhedron

- A polygonal mesh is a polyhedron if
  - Each edge is part of two polygons (it is closed)
  - Every vertex $\mathbf{v} \in V$ is part of finite, cyclic ordered set of polygons $\{Q_i\}$
    - The polygons incident to a vertex $\mathbf{v}$ can be ordered, such that Q_i and Q_j share an edge incident to $\mathbf{v}$

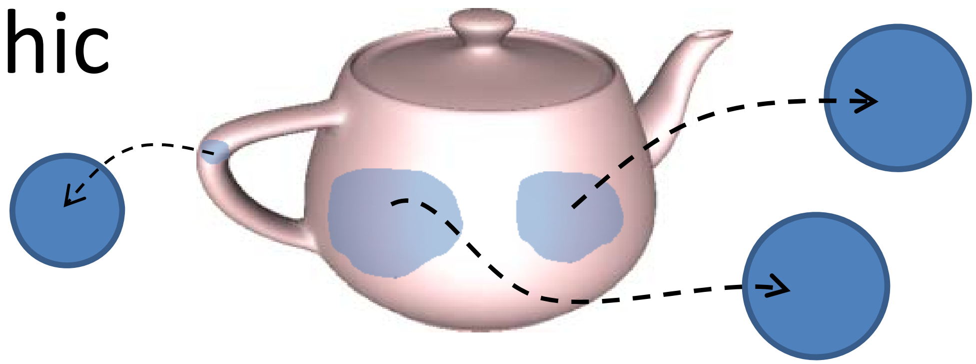


  - The union of all polygons forms a single connected component

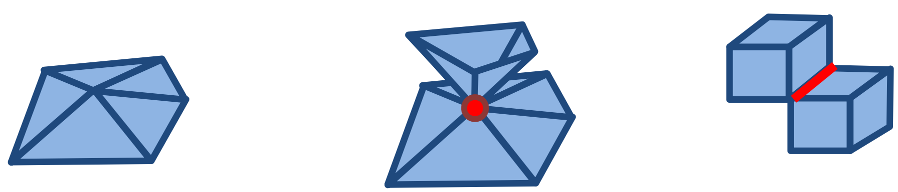
Definitions

Manifold

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

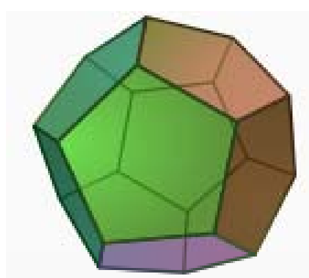


- Examples for a non-manifold vertex and a non-manifold edge



Definitions
Polyhedron

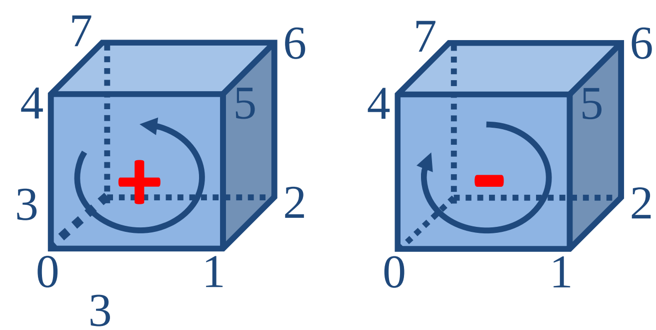
- The union of all polygonal areas is the *surface* of the polyhedron
- The polygonal areas of a polyhedron are also known as *faces*
- Every polyhedron partitions space into two areas; inside and outside the polyhedron



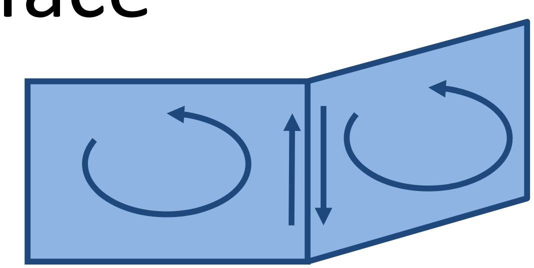
Definitions

Orientation

- Every face of a polygonal mesh is orientable
  - by defining “clockwise” (as opposed to “counterclockwise”). Two possible orientations
  - Defines the sign of the surface normal



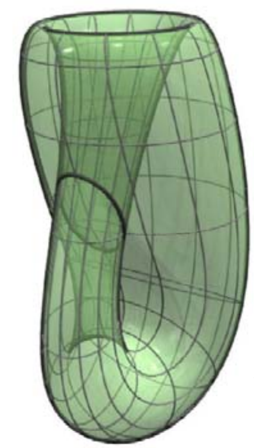
- Two neighboring facets are equally oriented, if the edge directions of the shared edge (induced by the face orientations) are opposing



Definitions

Orientability

- A polygonal mesh is orientable, if the incident faces to every edge can be equally oriented
  - If the faces are equally oriented for every edge, the mesh is *oriented*
- Notes
  - Every **non-orientable closed** mesh embedded in R^3 intersects itself
  - The surface of a polyhedron is always orientable



Klein bottle

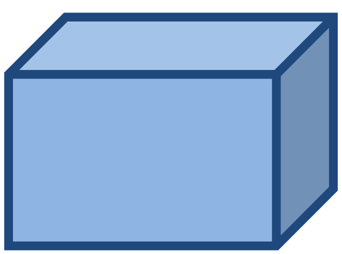
Möbius strip

Euler-Poincaré Formula

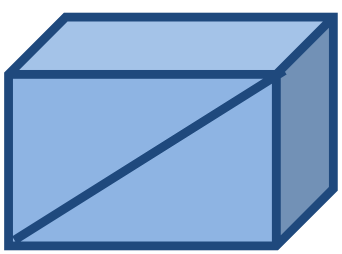
- Relation between #vertices, #edges and #faces of a polygonal mesh

- Example:

v = #vertices
e = #edges
f = #faces



v = 8
e = 12
f = 6



v = 8
e = 12+1
f = 6 +1

Euler-Poincaré Formula

- Theorem (Euler): The sum

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

  is **constant** for a given topology, no matter which mesh we choose
- If M has one boundary loop:

$$\chi(M) = v - e + f = 1$$

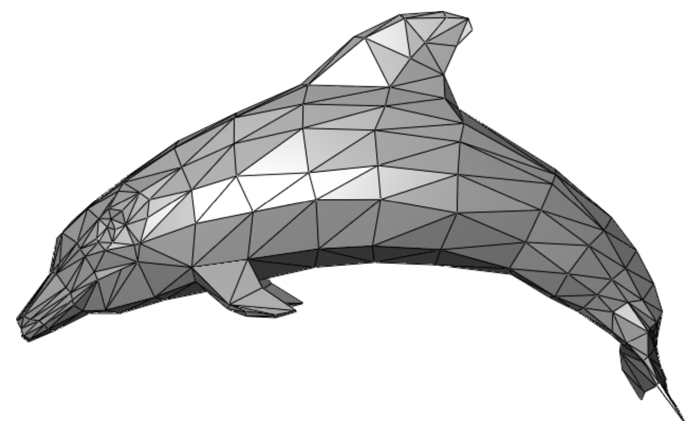
- If M is homeomorphic to a sphere:

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

Euler-Poincaré Formula

Usage

- Let‘s count the edges and faces in a closed triangle mesh:
  - Ratio of edges to faces: e = 3/2 f
    - each edge belongs to exactly 2 triangles
    - each triangle has exactly 3 edges
  - Ratio of vertices to faces: f ~ 2v
    - 2 = v – e + f = v – 3/2 f + f
    - 2 + f/2 = v
  - Ratio of edges to vertices: e ~ 3v
  - Average degree of a vertex: 6
    - 2 vertices incident on each edge



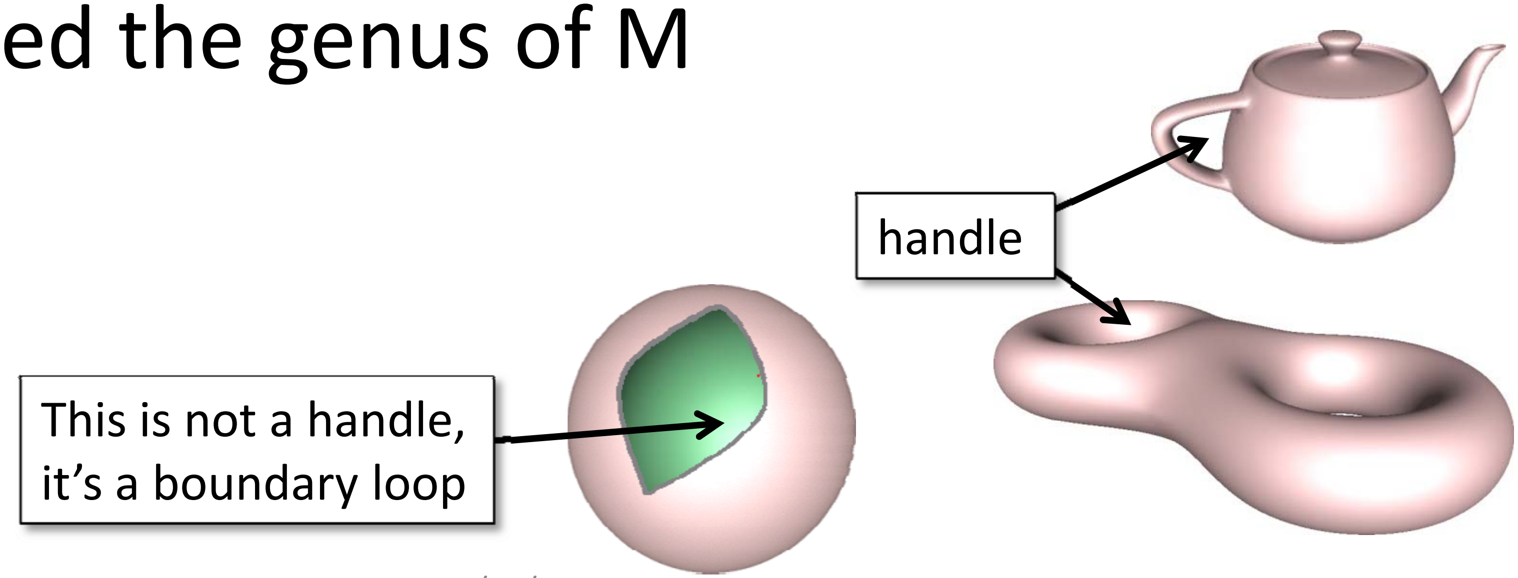
Euler-Poincaré Formula

Genus

- Theorem: if a polyhedron M is homeomorphic to a sphere with g handles (“holes“) then

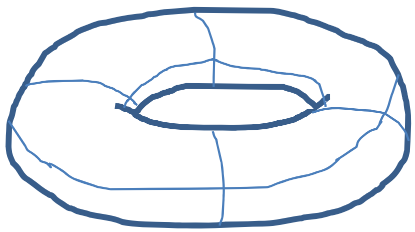
$$\chi(M) = v - e + f = 2(1 - g)$$

- g is called the genus of M



Euler-Poincaré Formula

Example: simple torus



$$v - e + f = 2(1 - g)$$

$$8 - 16 + 8 = 2(1 - 1)$$

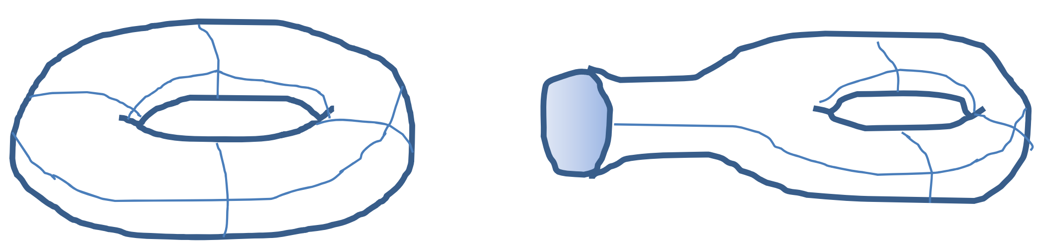
Euler-Poincaré Formula

Generalization

- Theorem: Let
  - v – # vertices
  - e – # edges
  - f – # faces
  - c – # connected components
  - h – # boundary loops
  - g – # handles (the genus)

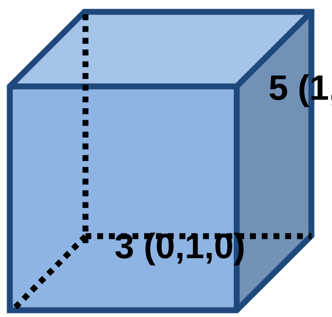
  then:

$$v - e + f - h = 2\,(c - g)$$



Data structures for meshes

Indexed Face Set



Vertex list (Coordinate3)

0	0.0	0.0	0.0
1	1.0	0.0	0.0
2	1.0	1.0	0.0
3	0.0	1.0	0.0
4	0.0	0.0	1.0
5	1.0	0.0	1.0
6	1.0	1.0	1.0
7	0.0	1.0	1.0

Face list (IndexedFaceSet)

0	0	3	2	1
1	0	1	5	4
2	1	2	6	5
3	2	3	7	6
4	0	4	7	3
5	4	5	6	7

Data structures for meshes

Space requirements

- Coordinates/attributes

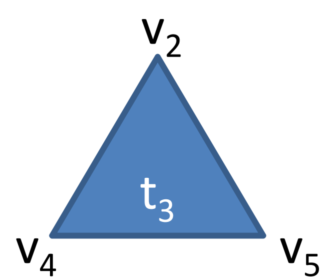
3 x 16 + k bits/vertex

vertex 1	x	y	z	c
vertex 2	x	y	z	c
vertex 3	x	y	z	c

- Connectivity

3 x $\log_2 V$ bits/triangle

triangle 1	1	2	3
triangle 2	3	2	4
triangle 3	4	2	5
triangle 4	7	5	6
triangle 5	6	5	8

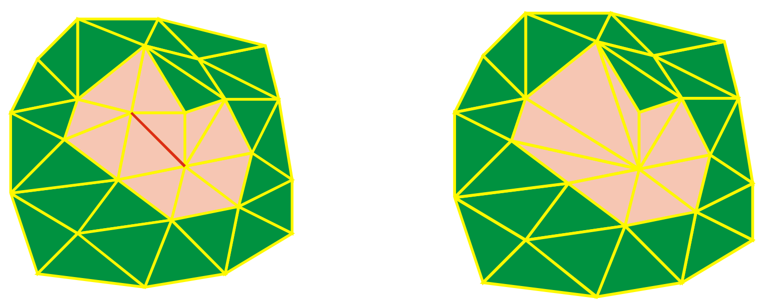


- When uncompressed, connectivity dominates
  - Reminder: f = 2v... so after 256 vertices

Data structures for meshes

Indexed Face Set – Problems

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

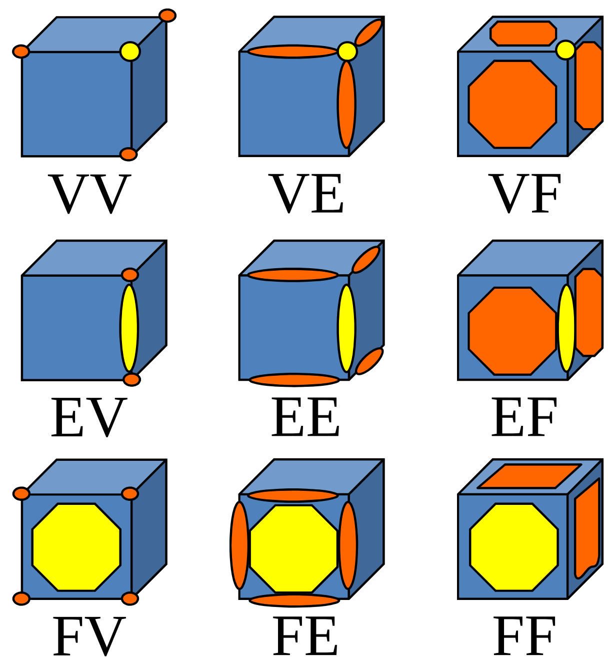


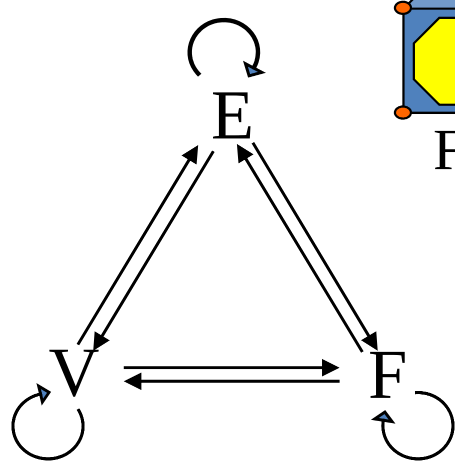
  - Breadth-first search costs O(k*v) where k = # found vertices

Data structures for meshes

Neighborhood relations [Weiler 1985]

- 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

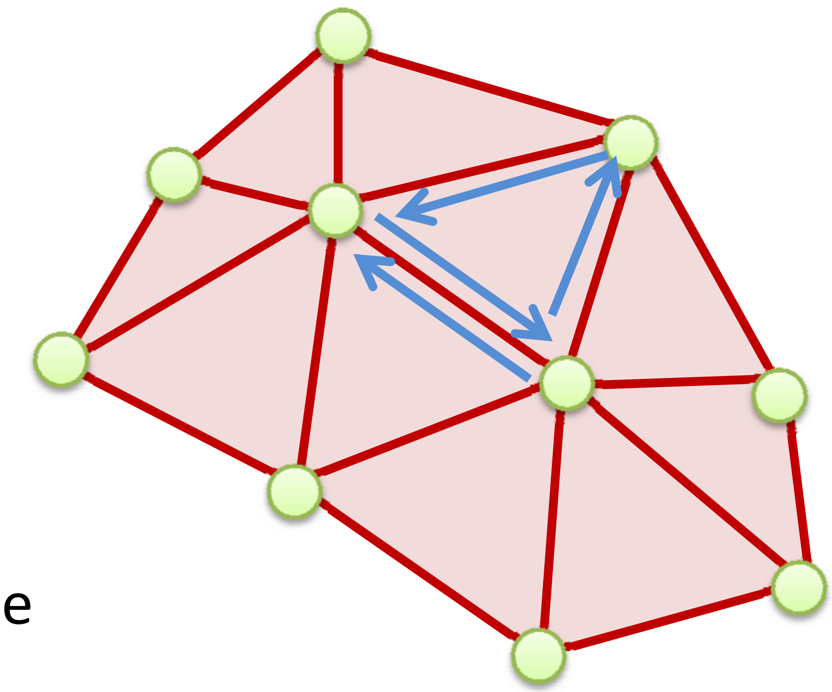




Data structures for meshes

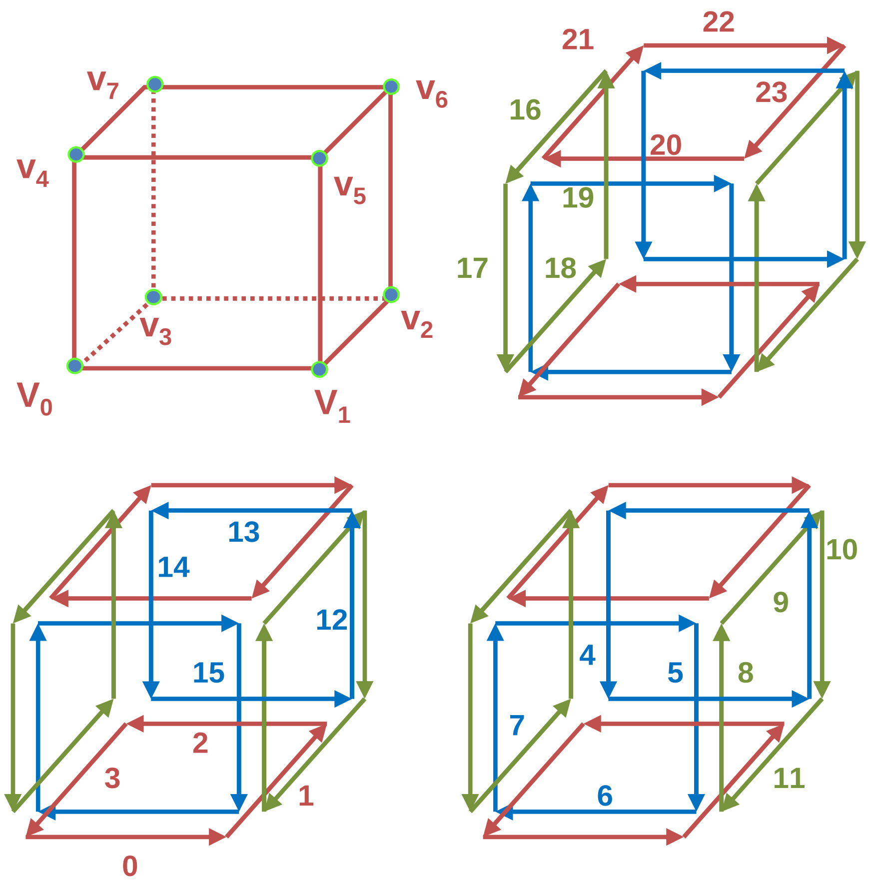
Half-edge data structure

- Half-edge has:
  - Pointer to twin h-e
  - Pointer to origin vertex
  - Pointer to next h-e
  - Pointer to previous h-e
  - Pointer to incident face
- Vertex has:
  - Pointer to one emanating h-e
- Face has:
  - Pointer to one of its enclosing h-e



Data structures for meshes

Half-edge data structure



Vertexlist

v	coord			he
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

Face

f	e
0	e0
1	e8
2	e4
3	e16
4	e12
5	e20

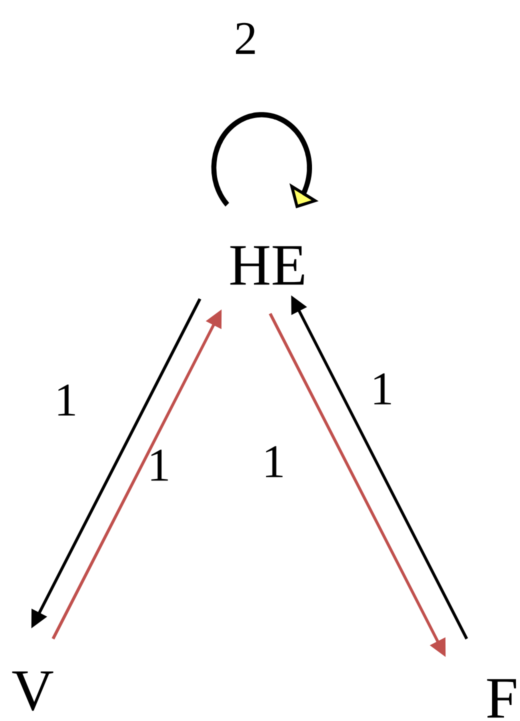
Half-Edgelist

he	vstart	next	prev	opp	he	vstart	next	prev	opp
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

Data structures for meshes

Half-edge data structure

- Each atomic insertion into the data structure (i.e., vertex, edge or face insertion) requires constant space and time

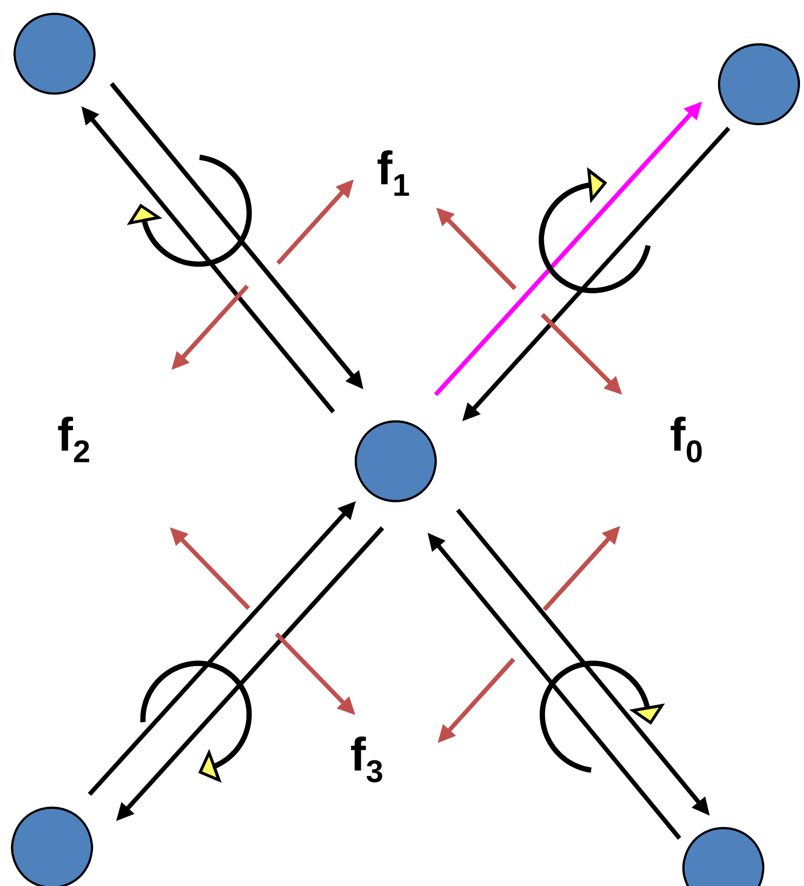


Triangle face:
prev = next->next

Data structures for meshes

Half-edge data structure

- All basic queries take constant O(1) time!
  - In particular, the query time is independent of the model size



Data structures for meshes

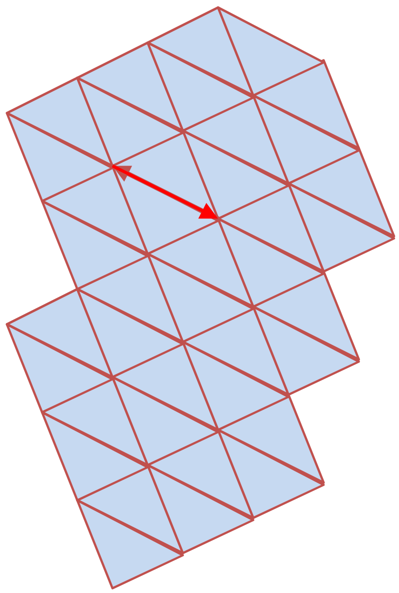
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
   q.append(he.opposite);

while (! q.isEmpty()) {
  he=q.first();
  // do work
  if (he.next.opposite != null)
    q.append(he.next.opposite);
  if (he.next.next.opposite != null)
    q.append(he.next.next.opposite)
}
```



Data structures for meshes

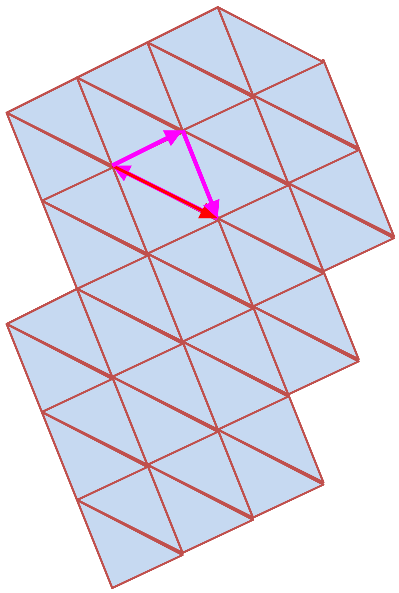
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
   q.append(he.opposite);

while (! q.isEmpty()) {
  he=q.first();
  // do work
  if (he.next.opposite != null)
    q.append(he.next.opposite);
  if (he.next.next.opposite != null)
    q.append(he.next.next.opposite)
}
```



Data structures for meshes

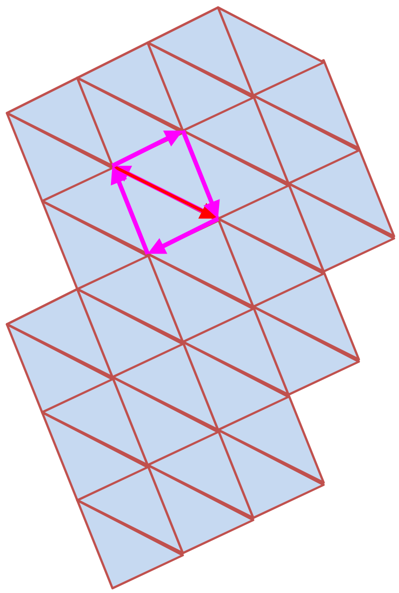
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
   q.append(he.opposite);

while (! q.isEmpty()) {
  he=q.first();
  // do work
  if (he.next.opposite != null)
    q.append(he.next.opposite);
  if (he.next.next.opposite != null)
    q.append(he.next.next.opposite)
}
```



Data structures for meshes

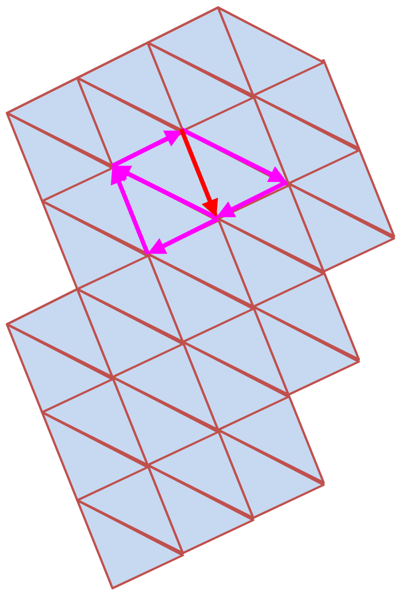
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
   q.append(he.opposite);

while (! q.isEmpty()) {
  he=q.first();
  // do work
  if (he.next.opposite != null)
    q.append(he.next.opposite);
  if (he.next.next.opposite != null)
    q.append(he.next.next.opposite)
}
```



Data structures for meshes

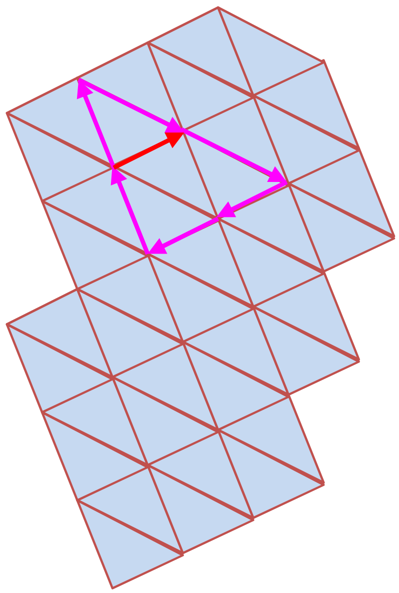
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
   q.append(he.opposite);

while (! q.isEmpty()) {
  he=q.first();
  // do work
  if (he.next.opposite != null)
    q.append(he.next.opposite);
  if (he.next.next.opposite != null)
    q.append(he.next.next.opposite)
}
```



Data structures for meshes

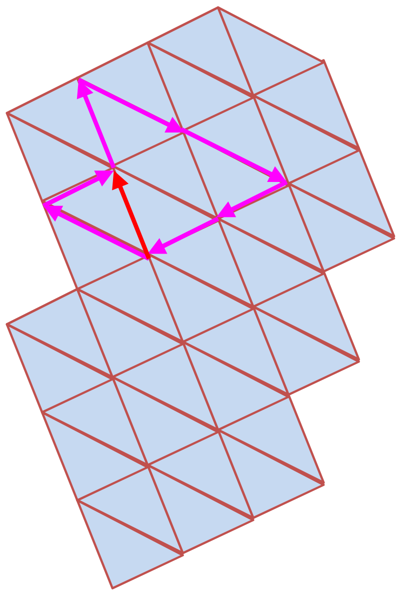
Half-edge data structure

- Example: efficient breadth-first search

```
//q: Queue (FIFO) of HalfEdges

HalfEdge he;
q.append(he);
if (he.opposite != null)
   q.append(he.opposite);

while (! q.isEmpty()) {
  he=q.first();
  // do work
  if (he.next.opposite != null)
    q.append(he.next.opposite);
  if (he.next.next.opposite != null)
    q.append(he.next.next.opposite)
}
```



Data structures for meshes

Criteria for design

- Maximal number of vertices (i.e., how large are the models?)
- Available memory size
- Required operations
  - Mesh updates (edge collapse, edge flip)
  - Neighborhood queries
- Distribution of operations (what are the most common/frequent ones?)
- How can we compare different data structures?

Homework 2

- On the course website
- Familiarize yourself with a mesh library
- Read a mesh in OFF format, render it and compute some basic things

Thank you!