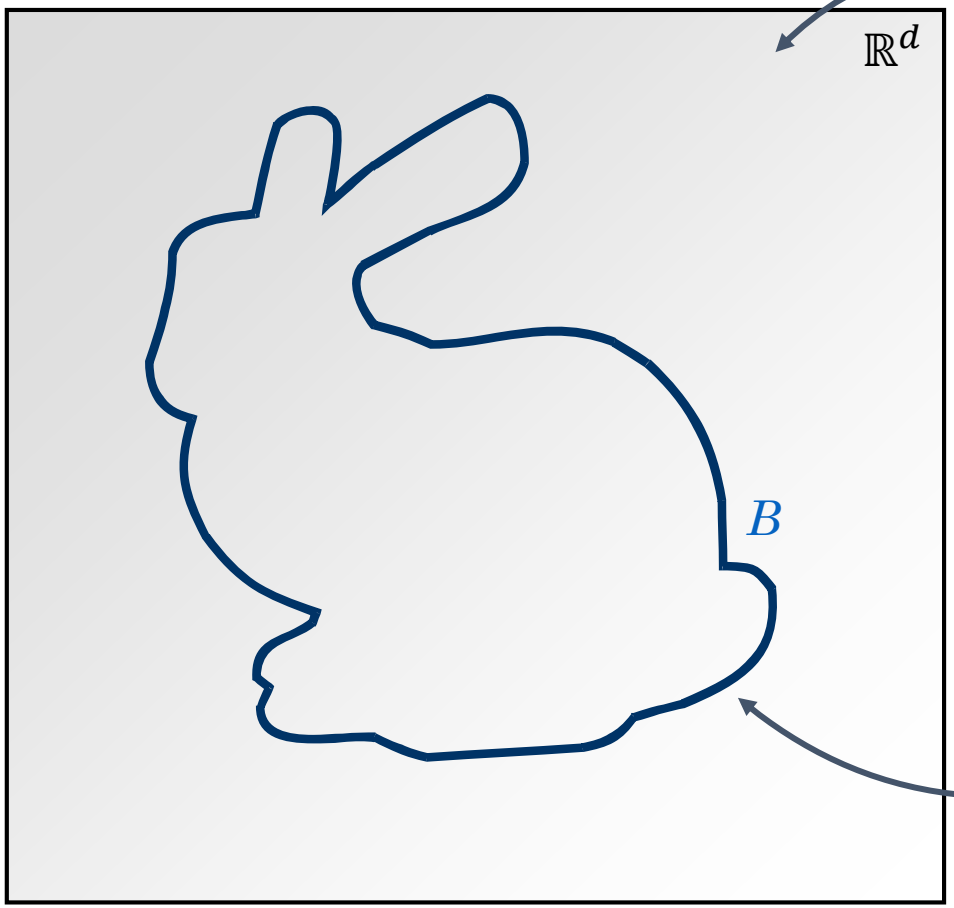
Geometric Representations

3D Graphics

Motivation

Geometric representation

- What do we want to do?

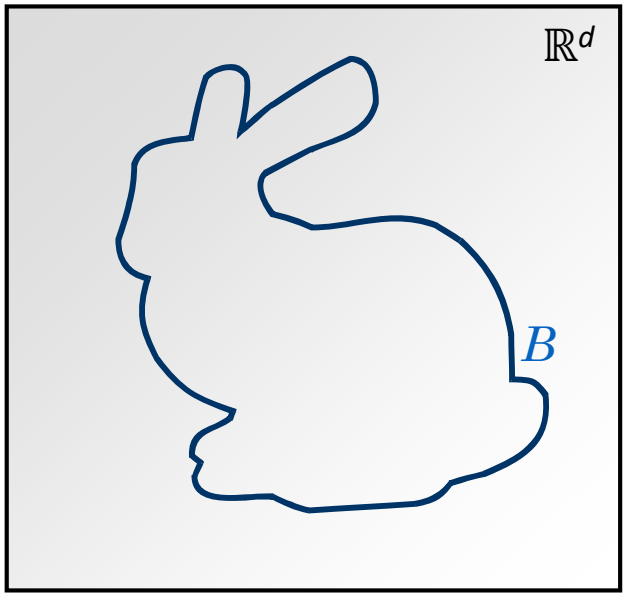


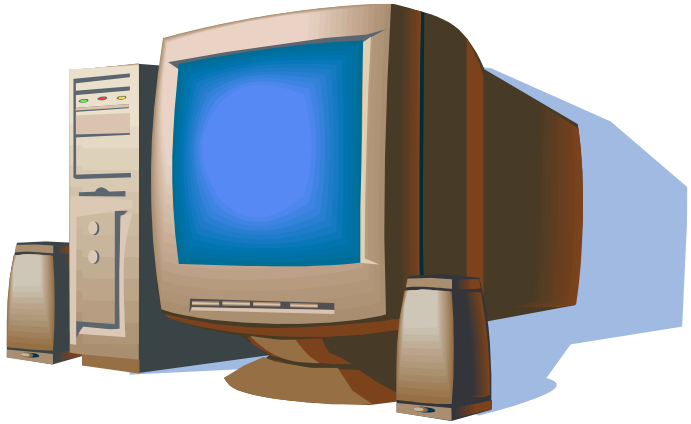
empty space
(typically $\mathbb{R}^3$)

geometric object
$B \subseteq \mathbb{R}^3$

Fundamental problem

- The Problem:





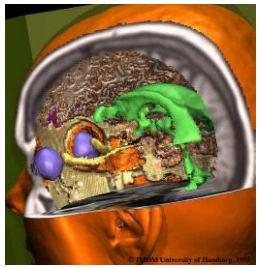
infinite number of points

my computer: 8GB of memory

We need to encode a continuous model with a finite amount of information

Where does this problem occur?

- Reconstruction from real data
  - Modeling or analyses based on acquired scenes
- Procedural modeling
  - Automatic modeling of structured scenes or objects
- Interactive modeling
  - Develop tools for computer artists



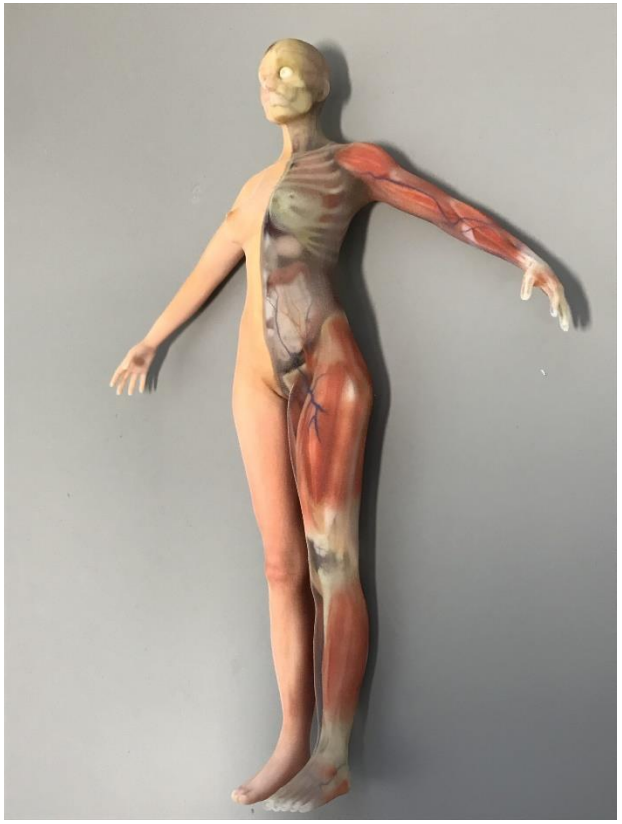
Overview

	Volume Representation	Surface Representation
Implicit	• Voxel grid	• Level sets
Parametric	• Tetrahedral meshes	• Spline surfaces • Subdivision surfaces • Primitive meshes • Triangle meshes

Volume representations

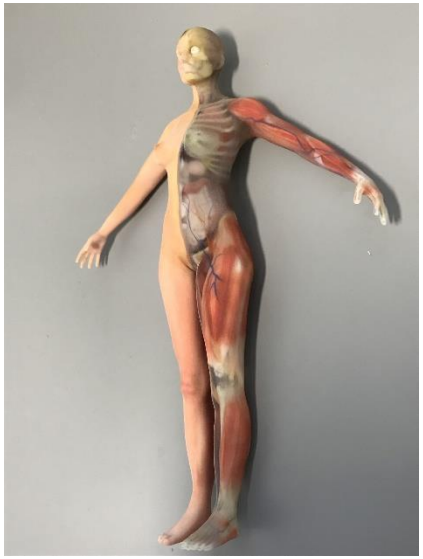
Volume

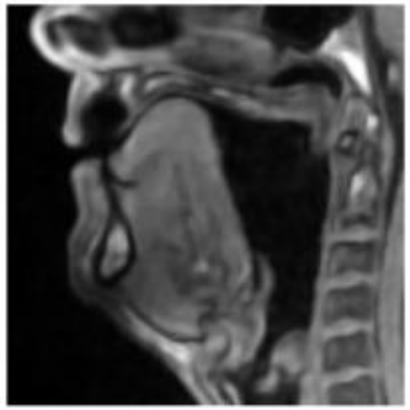
Definition: compact subset of $\mathbb{R}^3$



Interest of volumetric representation

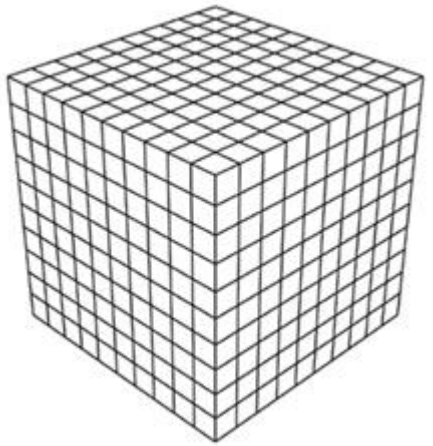
- Allows to model interior information of shapes, e.g. shape densities, colors etc.
- For modeling and simulation, can constrain unrealistic shape variation
- Many of the models used in graphics (animals, plants) are volumetric in reality





Implicit voxel grid

- Defined by function
$$F: \mathbb{R}^3 \rightarrow \mathbb{R}$$
- Discretized voxel grid assigns a value v to position (x, y, z)



Implicit voxel grid

Advantages

- Straight forward to extract or change value v associated to position (x, y, z)

Disadvantages

- Difficult to modify shape as no high-level information is available
- For irregular shapes, requires voxelization of large volume

Parametric tetrahedral mesh

Defined by a function

$f: \Omega \rightarrow \mathcal{L}$

with $\Omega \subset \mathbb{R}^3$

$\mathcal{L} \subset \mathbb{R}^3$ = volume of interest

Defined by a function

$f: \Omega \rightarrow \mathcal{L}$

with $\Omega \subset \mathbb{R}^3$

$\mathcal{L} \subset \mathbb{R}^3$ = volume of interest

Parametric tetrahedral mesh

Advantages

- Allows to modify the shape by changing the mapping function
- Used for deformations and simulations

Disadvantages

- Difficult to store associated information v associated to position (x, y, z) at non-vertex positions
- Interpolating values requires computation

Surface representations

Surface

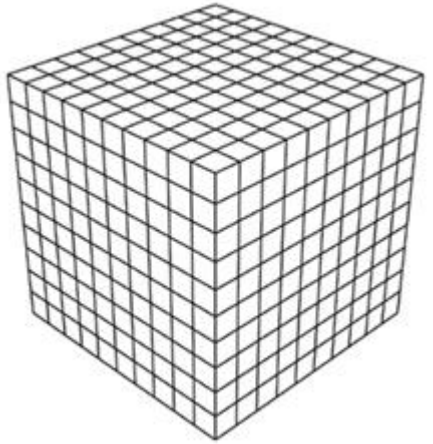
Definition: orientable 2D manifold embedded in $\mathbb{R}^3$

Intuitively:

- Boundary surface of non-degenerate 3D solid
- Non-degenerate: no infinitely thin parts, i.e. solid has a clearly defined interior and exterior

Implicit Level Sets

- Defined by function
$$F: \mathbb{R}^3 \rightarrow \mathbb{R}$$
- Discretized voxel grid assigns a distance d from the surface to each position (x, y, z)



Implicit Level Sets

Advantages

- Straight forward to extract or change value v associated to position (x, y, z)
- Allows for fast Boolean operations of surfaces

Disadvantages

- Difficult to modify surface as no high-level information is available
- For irregular shapes, requires voxelization of large volume

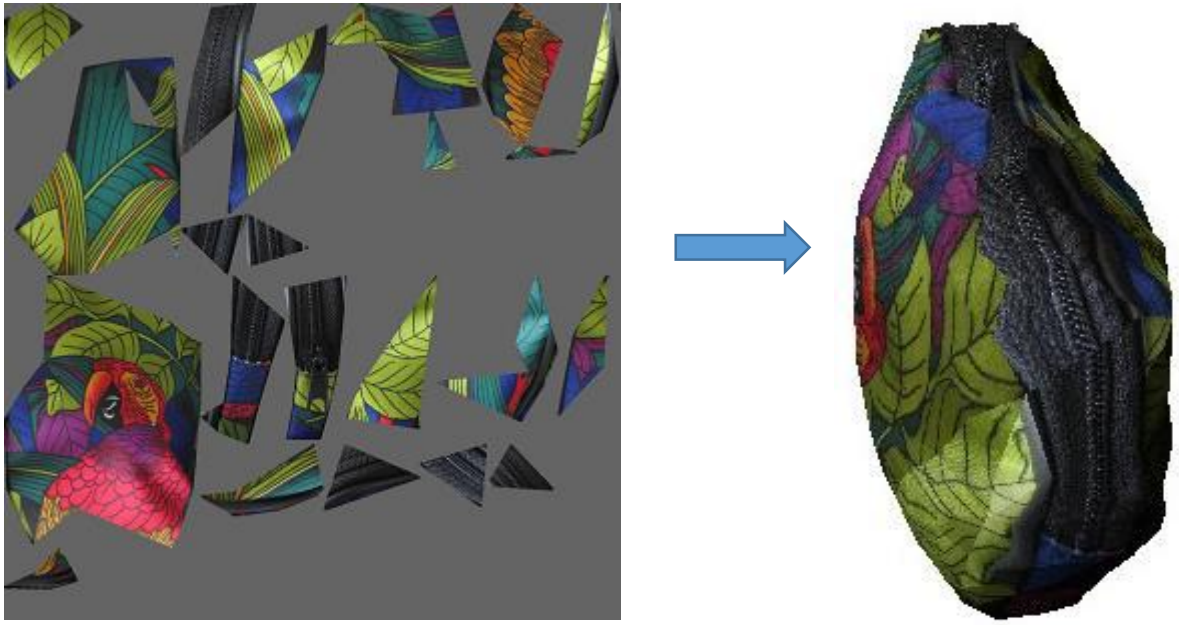
Parametric Surface Representations

Defined by a function

$f: \Omega \rightarrow \mathcal{L}$

with $\Omega \subset \mathbb{R}^2$

$\mathcal{L} \subset \mathbb{R}^3$ = full embedding space



Spline surfaces

- Parameter domain $\Omega = [u_n, u_m] \times [v_n, v_k]$
- Polynomial or rational basis functions $N_i^n(.)$
- $(u, v) \rightarrow \sum_{i=0}^{m} \sum_{j=0}^{k} c_{(ij)} \, N_i^n(u) N_j^n(v)$
- $c_{(ij)}$ are called control points and define a control mesh

Historic example
« Utah teapot »

Spline Surfaces

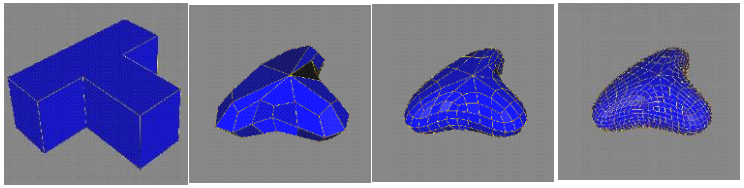
Advantages

- Allows to model smooth surfaces
- Easy to evaluate points at any position
- Allows for deformation by changing control points

Disadvantages

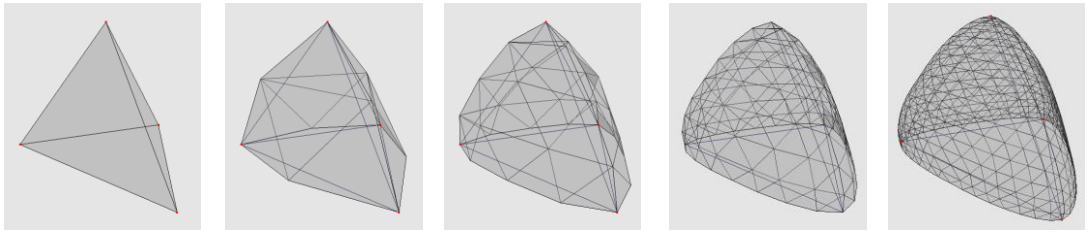
- Difficult to fit to acquired scan data efficiently

Subdivision surfaces

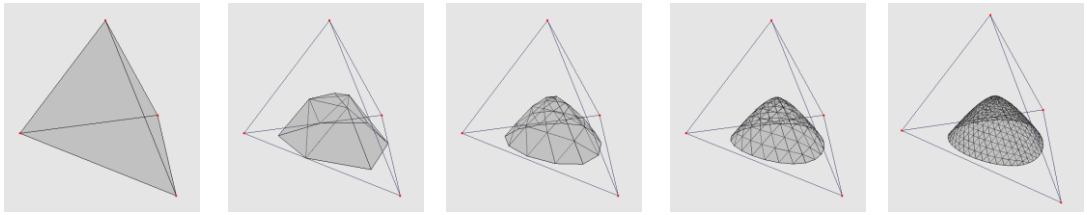


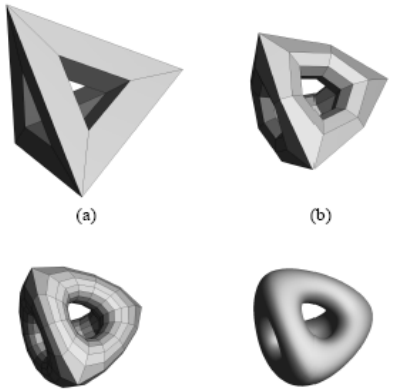
- Topology defined by the control polygon
- Progressive refinement (interpolation or approximation)

Butterfly



Loop





Catmull-Clark

Subdivision surfaces

- Like splines, they are hence controlled by coarse *control mesh*
- Each step involves
  - Subdivision (insertion of new vertices)
  - Adjustment of vertex positions (new only for interpolation schemes, all for approximation schemes)
- Provably converge to smooth limit surfaces

Subdivision Surfaces

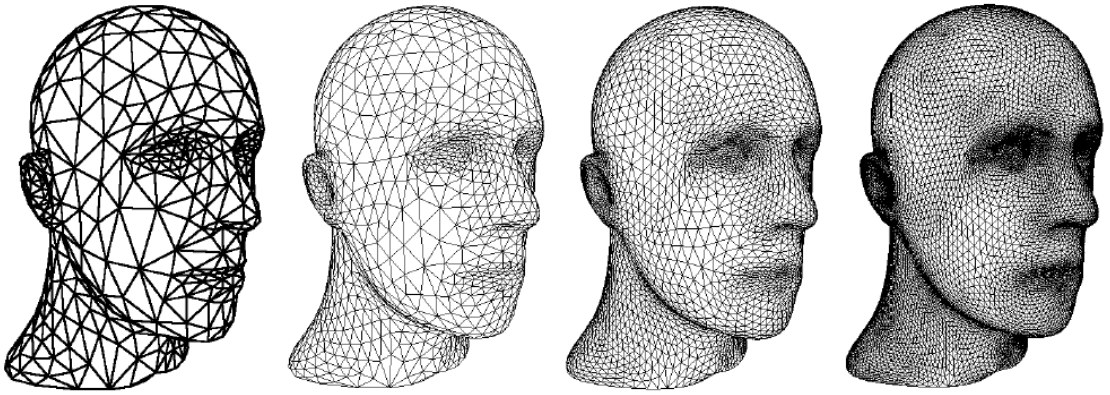
Advantages

- Arbitrary geometry and *topology* can be modeled
- Approximation at different level of detail

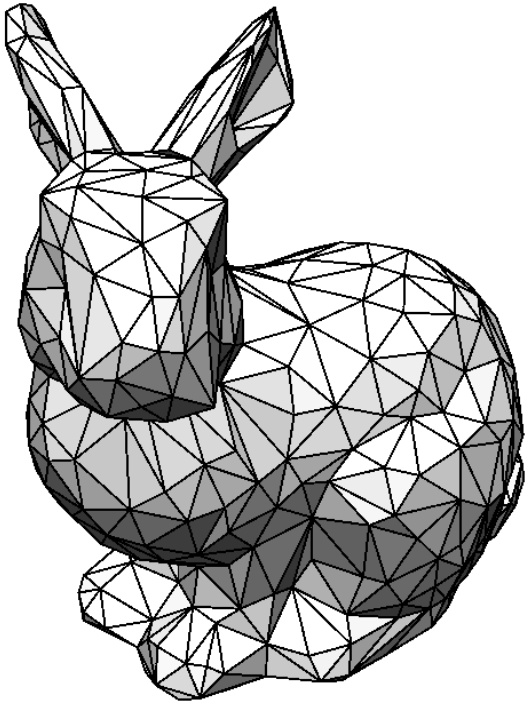
Disadvantages

- No parameterization
- Some unexpected results

Loop

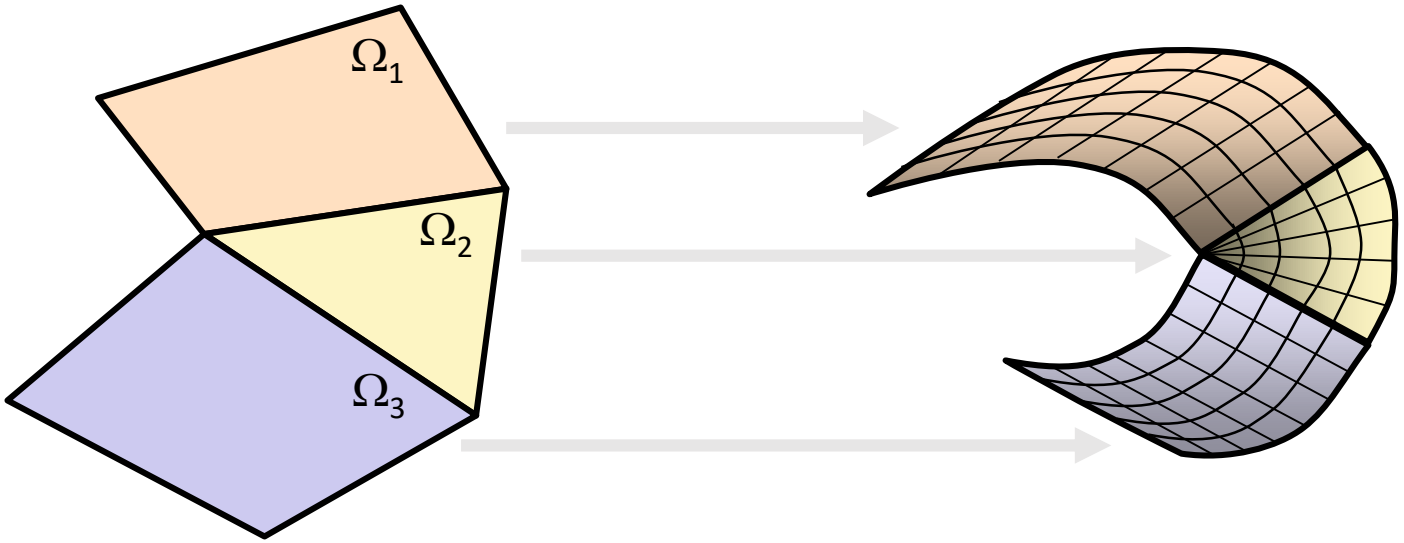


Primitive meshes



- Primitive Meshes
  - Collection of geometric primitives
    - Triangles
    - Quadrilaterals
  - Typically, primitives are parametric surfaces
  - Composite model:
    - Mesh encodes topology, rough shape
    - Primitive parameter encodes local geometry
  - Triangle meshes rule the world (including “triangle soups”)

Primitive meshes



- Complex Topology for Parametric Models
  - Mesh of parameter domains attached in a mesh
  - Domain can have complex shape (“trimmed patches”)
  - Separate mapping function *f* for each part (typically of the same class)

Primitive meshes

Advantages

- Compact representation (usually)
- Can represent arbitrary topology

Disadvantages

- Need to specify a mesh first, then edit geometry
- Problems
  - Mesh structure needs to be adjusted to fit shape
  - Mesh encodes object topology
    $\Rightarrow$ Changing object topology is difficult
- Examples
  - Surface reconstruction
  - Fluid simulation (surface of splashing water)

Special case: Triangle mesh

Triangle meshes

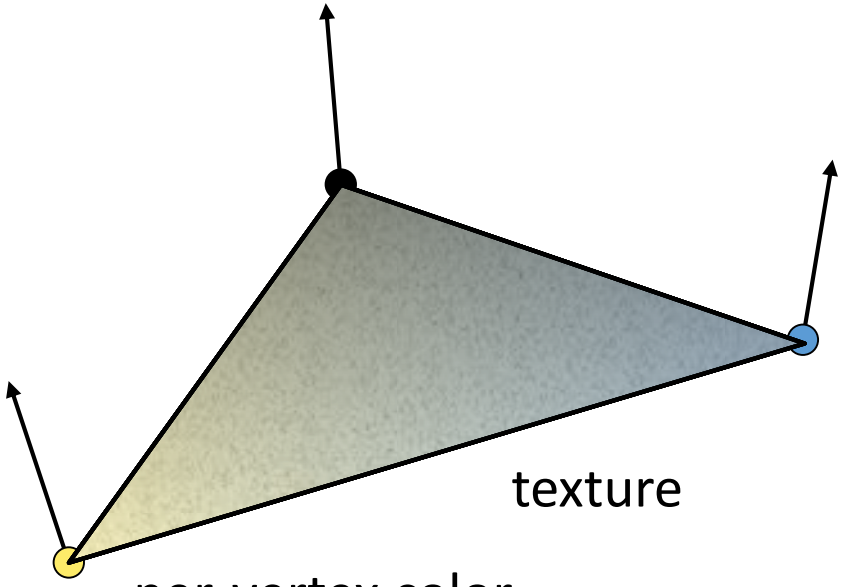
- (Probably) most common representation
- Simplest surface primitive
  that can be assembled into meshes
  - Rendering in hardware (z-buffering)
  - Simple algorithms for intersections (raytracing, collisions)
- Piecewise linear surface representation
- Each triangle $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ defines points

$$\boldsymbol{p} = \alpha\boldsymbol{a} + \beta\boldsymbol{b} + \gamma\boldsymbol{c}$$

  with $\alpha + \beta + \gamma = 1, \alpha + \beta + \gamma > 0$

Attributes

- How to define a triangle?
  - We need three points in $\mathbb{R}^3$ (obviously).
  - But we can have more:



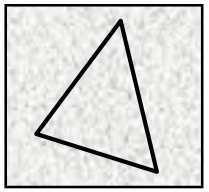
per-vertex normals
(represent smooth surfaces more accurately)

texture

per-vertex color

per-vertex texture coordinates

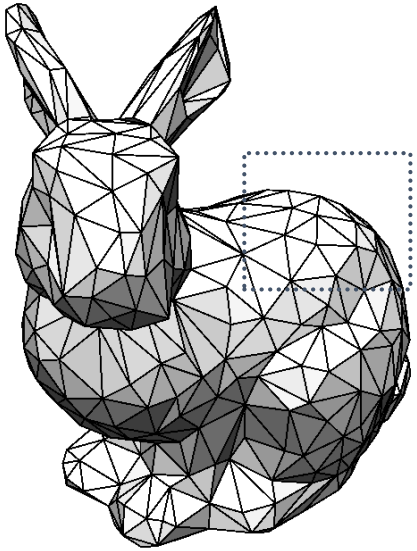
(etc...)

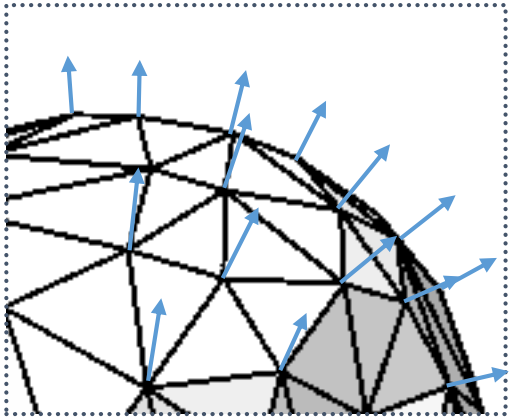


Shared Attributes in Meshes

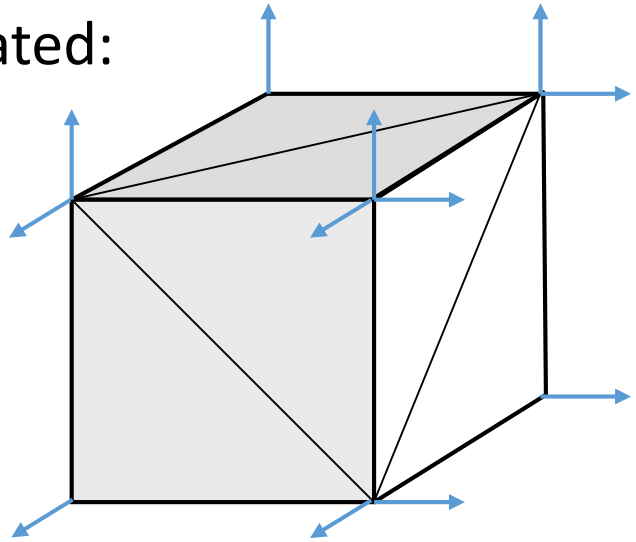
In Triangle Meshes:

- Attributes might be shared or separated:





adjacent triangles
share normals



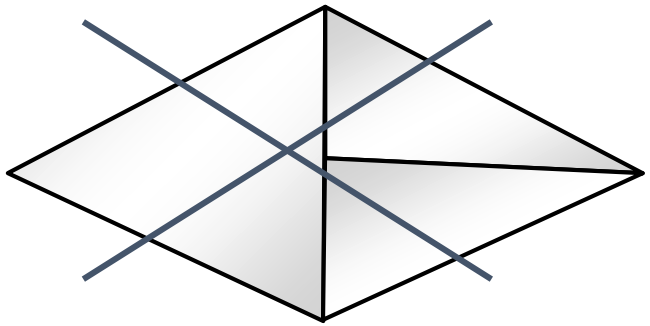
adjacent triangles
have separated normals

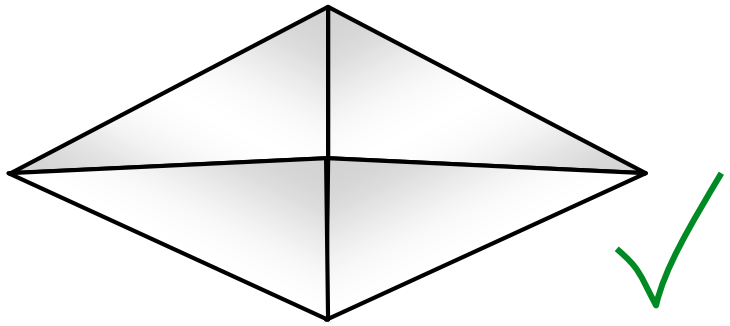
“Triangle Soup”

- Variants in triangle mesh representations:
  - “*Triangle Soup*”
    - A set $S = \{t_1, ..., t_n\}$ of triangles
    - No further conditions
    - Does **not** represent a surface
  - *Triangle Meshes*: Additional consistency conditions
    - Conforming meshes: Vertices meet only at vertices
    - Manifold meshes: No intersections, no T-junctions

Conforming Meshes

- Conforming Triangulation:
  - Vertices of triangles must only meet at vertices, not in the middle of edges:



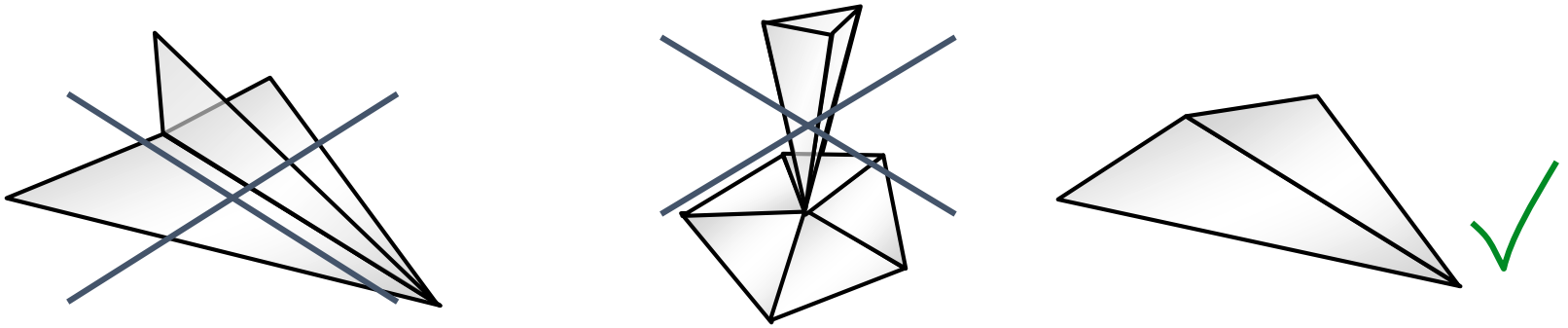


  - This makes sure that we can move vertices around arbitrarily without creating holes in the surface

Manifold Meshes

Triangulated two-manifold:

- Every edge is incident to exactly 2 triangles (closed manifold)
- ...or to at most two triangles (manifold with boundary)
- No triangles intersect (other than along common edges or vertices)
- Two triangles that share a vertex must share an edge



Attributes

- In general:
  - Vertex attributes:
    - Position (mandatory)
    - Normals
    - Color
    - Texture Coordinates
  - Face attributes:
    - Color
    - Texture
  - Edge attributes (rarely used)
    - E.g.: Visible line

In-class exercise

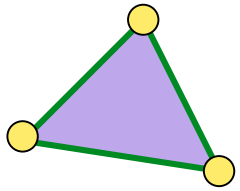
In-class exercise

- How would you describe a triangle mesh whose attributes are only vertex positions?
- What data structure would you use?
- What are the advantages and disadvantages?

Some common data structures

- List of vertices, triangles, edges
- Half-edge data structure

Simple data structure



- The simple approach: List of vertices, edges, triangles

v_1: (posx posy posz), $attrib_1$, ..., $attrib_{n_{av}}$
...
v_{n_v}: (posx posy posz), $attrib_1$, ..., $attrib_{n_{av}}$

e_1: ($index_1$ $index_2$), $attrib_1$, ..., $attrib_{n_{ae}}$
...
e_{n_e}: ($index_1$ $index_2$), $attrib_1$, ..., $attrib_{n_{ae}}$

t_1: (idx_1 idx_2 idx_3), $attrib_1$, ..., $attrib_{n_{at}}$
...
t_{n_t}: (idx_1 idx_2 idx_3), $attrib_1$, ..., $attrib_{n_{at}}$

Pros & Cons

Advantages:

- Simple to understand and build
- Provides exactly the information necessary for rendering

Disadvantages:

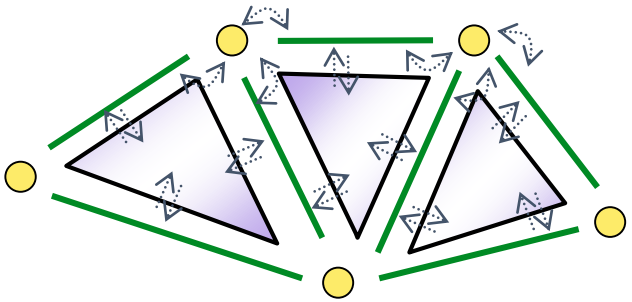
- Dynamic operations are expensive:
  - Removing or inserting a vertex
    $\rightarrow$ renumber expected edges, triangles
- Adjacency information is one-way
  - Vertices adjacent to triangles, edges $\rightarrow$ direct access
  - Any other relationship $\rightarrow$ need to search
  - Can be improved using hash tables (but still not dynamic)

Adjacency data structures

Alternative:

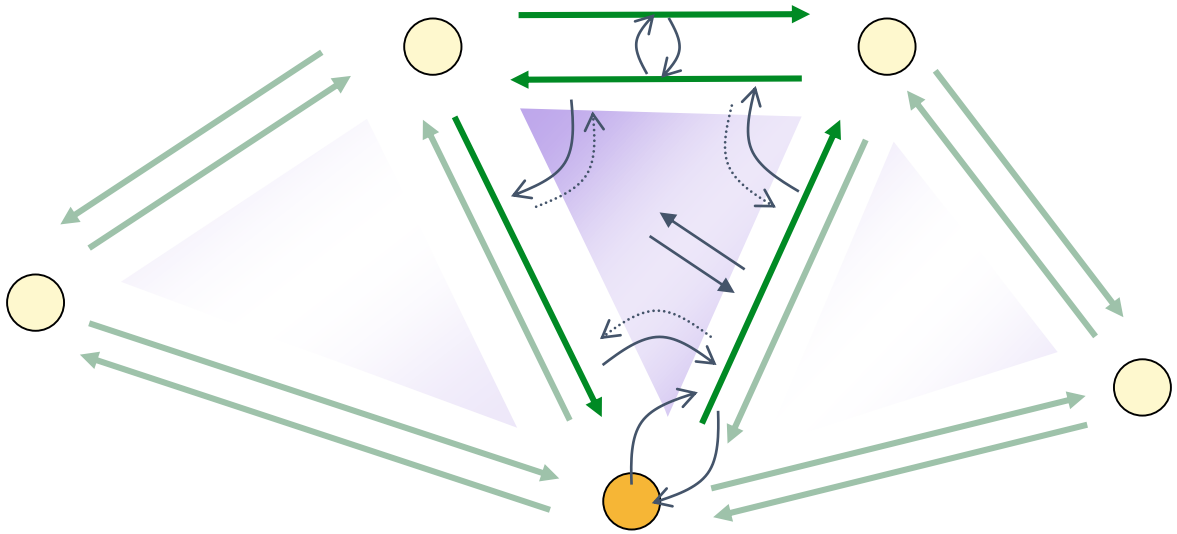
- Some algorithms require extensive neighborhood operations (get adjacent triangles, edges, vertices)
- ...as well as dynamic operations (inserting, deleting triangles, edges, vertices)
- For such algorithms, an *adjacency based* data structure is usually more efficient
  - The data structure encodes the graph of mesh elements
  - Using pointers to neighboring elements

First try...



- Straightforward Implementation:
  - Use a list of vertices, edges, triangles
  - Add a pointer from each element to each of its neighbors
  - Global triangle list can be used for rendering
- Remaining Problems:
  - Lots of redundant information – hard to keep consistent
  - Adjacency lists might become very long
    - Need to search again (might become expensive)
    - This is mostly a “theoretical problem” (O(n) search)

Less Redundant Data Structures



Half edge data structure:

- Half edges, connected by clockwise / ccw pointers
- Pointers to opposite half edge
- Pointers to/from start vertex of each edge
- Pointers to/from left face of each edge

Implementation

```
•// a half edge
•struct HalfEdge {
•    HalfEdge* next;
•    HalfEdge* previous;
•    HalfEdge* opposite;

•    Vertex* origin;
•    Face* leftFace;
•    EdgeData* edge;
•};

•// the data of the edge
•// stored only once
•struct EdgeData {
•    HalfEdge* anEdge;
•    /* attributes */
•};
```

```
// a vertex
struct Vertex {
    HalfEdge* someEdge;
    /* vertex attributes */
};

// the face (triangle, poly)
struct Face {
    HalfEdge* half;
    /* face attributes */
};
```

Implementation

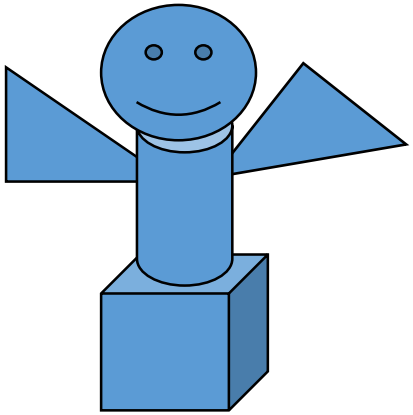
Implementation:

- The data structure should be encapsulated
  - To make sure that updates are consistent
  - Implement abstract data type with more high level operations that guarantee consistency of back and forth pointers
- Free Implementations are available, for example
  - OpenMesh
  - CGAL
- Many alternative data structures exist: for example winged edge (Baumgart 1975)

Hierarchical modeling

Hierarchical modeling

- Basic example: OpenGL programming
  - Describe each object in the frame where it is the simplest
  - Assemble them in a hierarchy for consistency!



Example of hierarchical modeling

- Character = hierarchy of limbs
- The foot remains connected to the hip articulates !