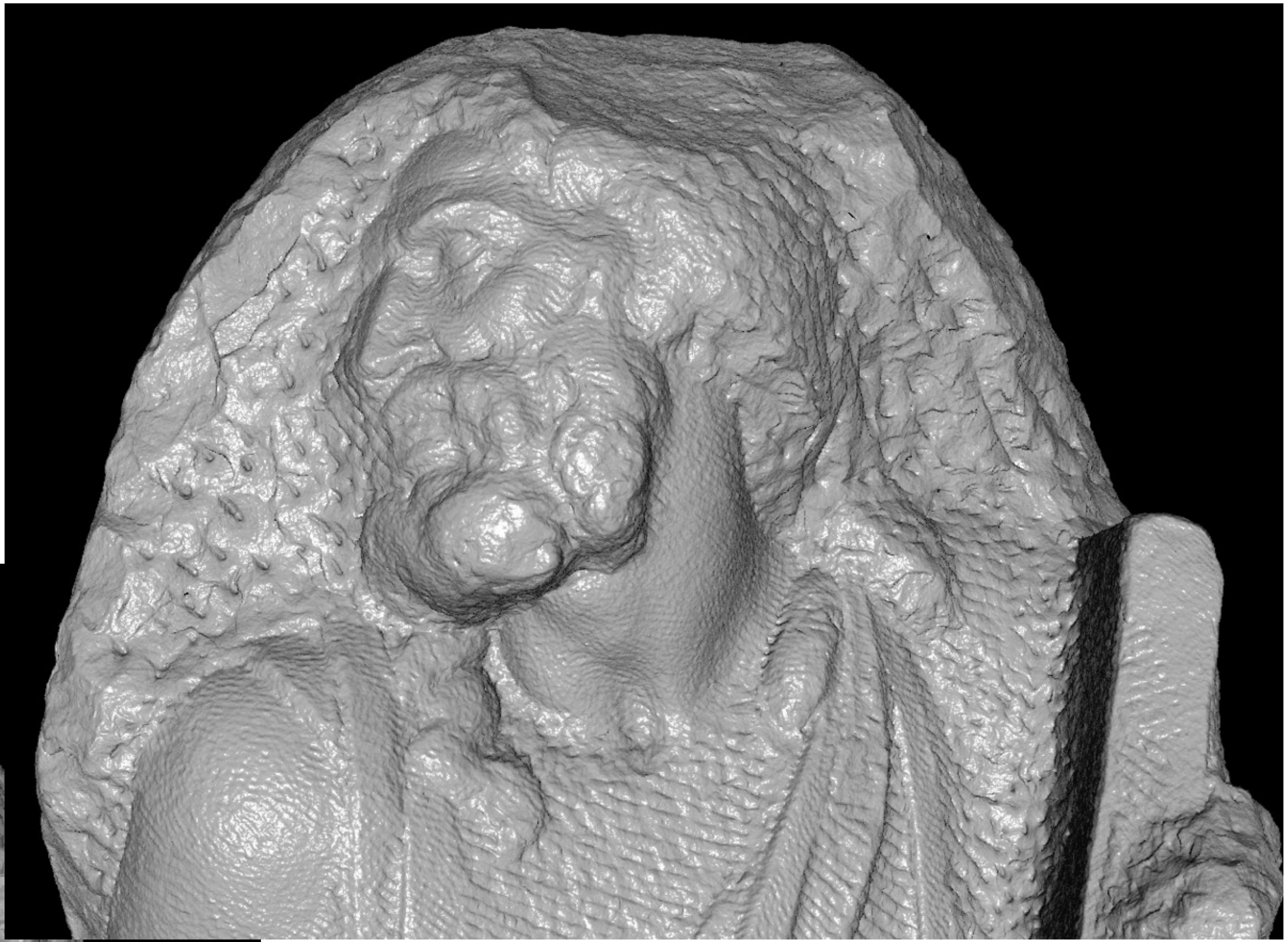
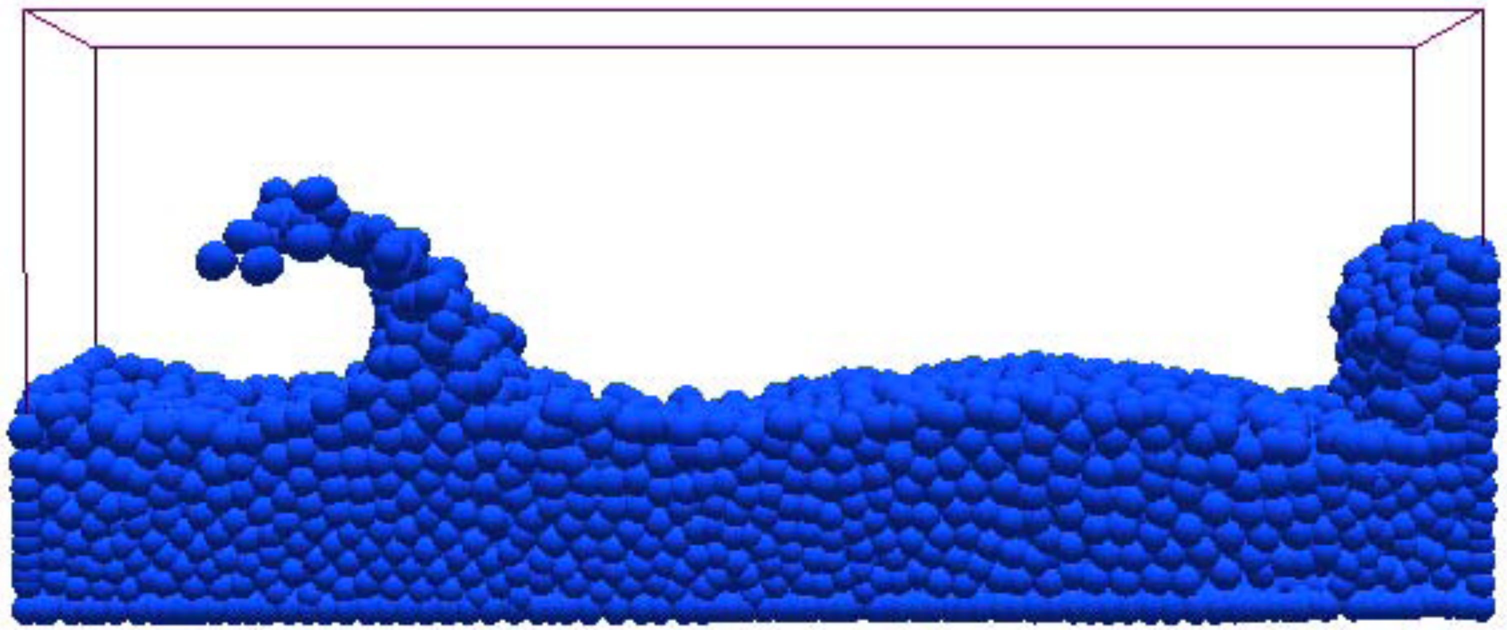
Highlights

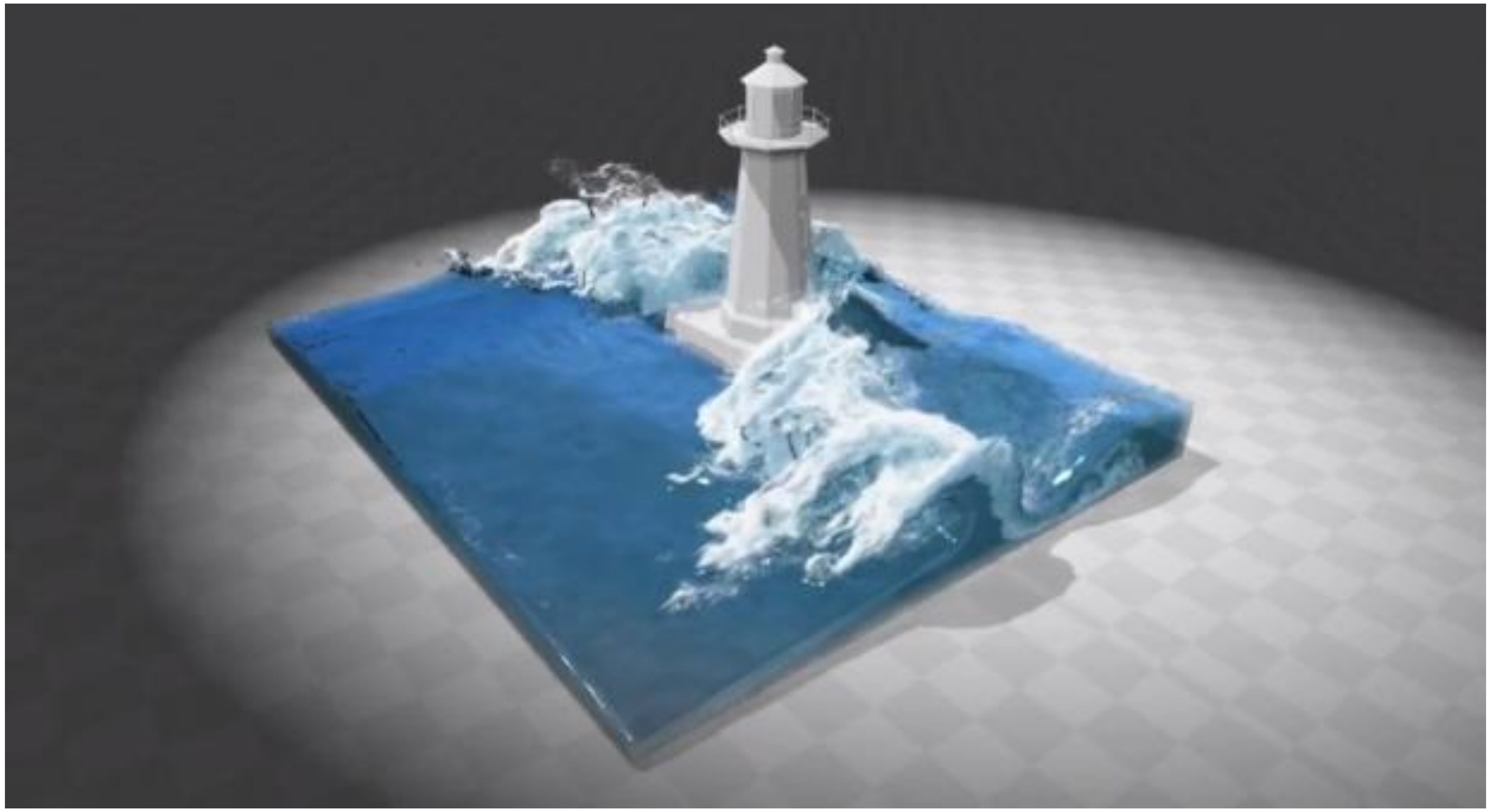
- Point clouds : why do we use them?
  - Many scanning devices produce them
  - Simplest graphics primitive to argue about
  - Can render directly (do we?)
  - Can manipulate directly (if desired)
- What are other geometry representations?
  - Splines?
  - Meshes?
  - Implicit surfaces?

Highlights



Highlights

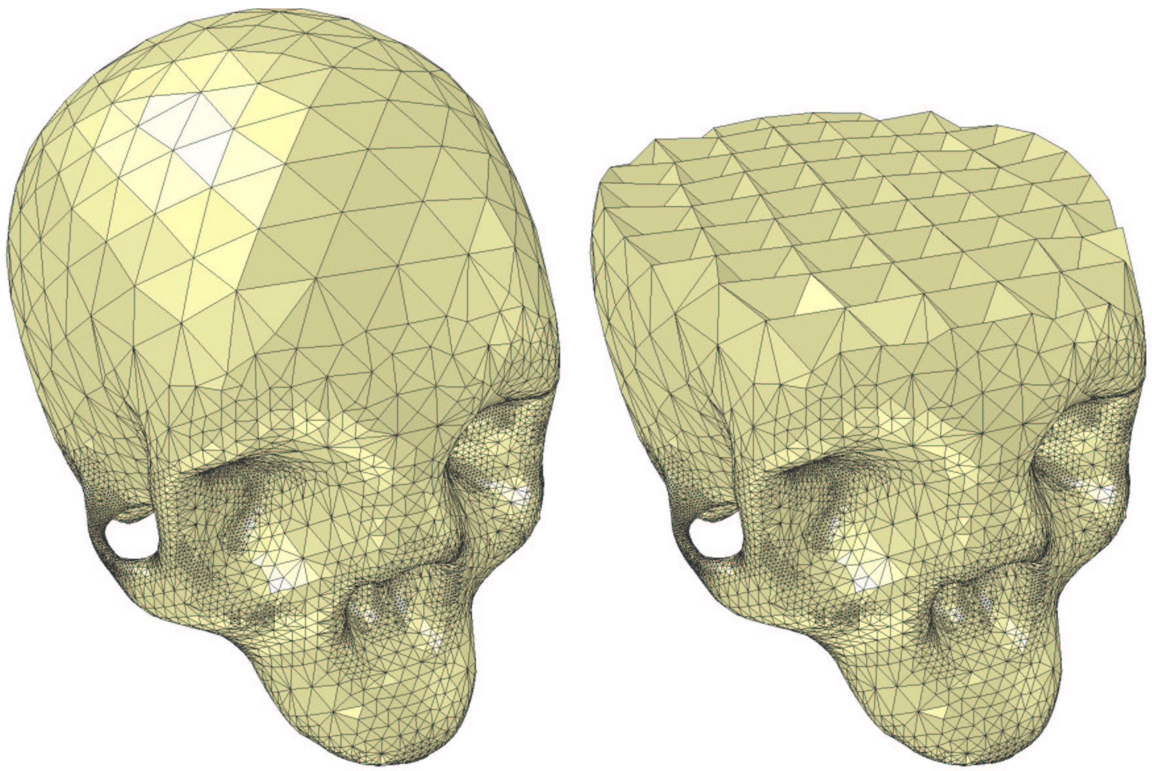




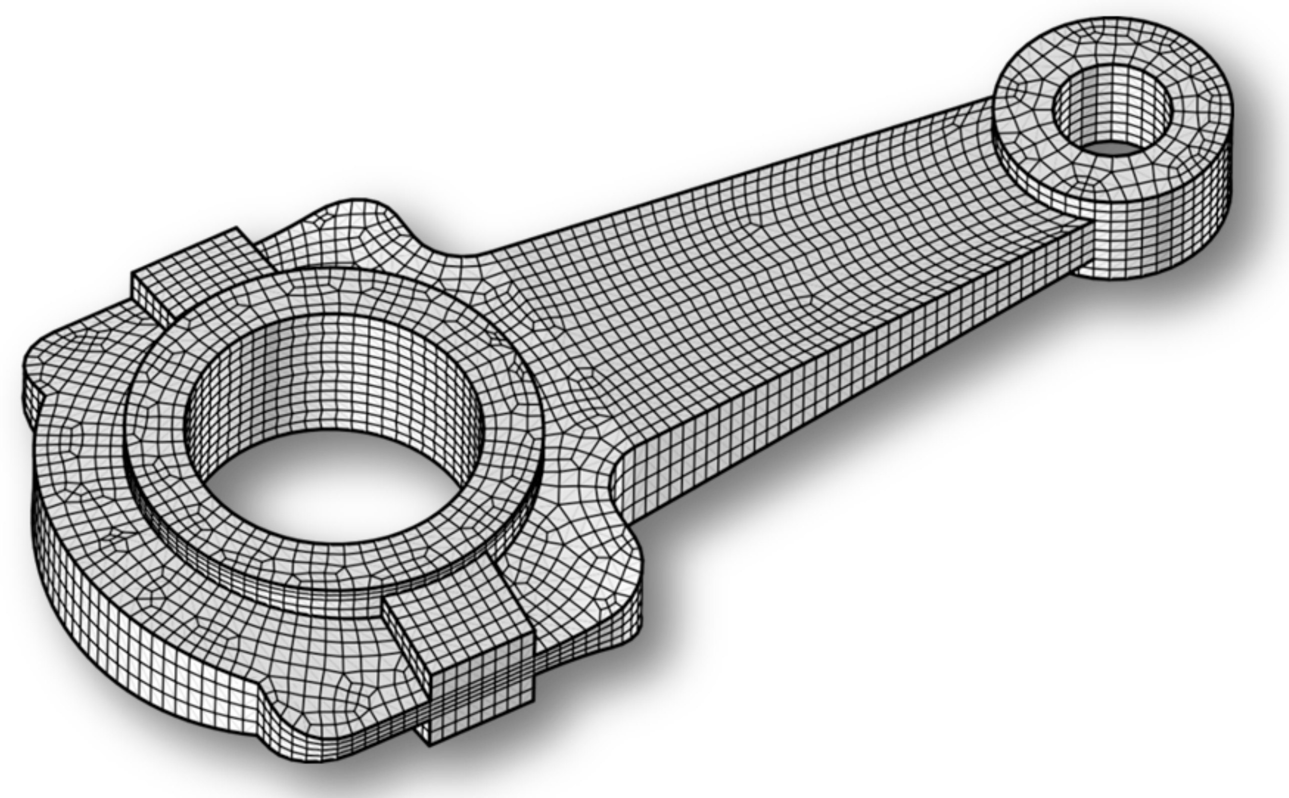
Highlights

- Describe a number of discrete representations used to encode geometric objects for modeling and simulation purposes
  - Meshes
  - Implicit surfaces
  - Point clouds
- Explain the features that make one representation better than another for certain tasks (e.g. meshes vs. implicit surfaces)
  - Static vs. dynamic topology (connectivity)
  - "Shape memory" and deformation drift
  - Regular, structured storage
  - Efficiency of geometric queries

Geometry representations



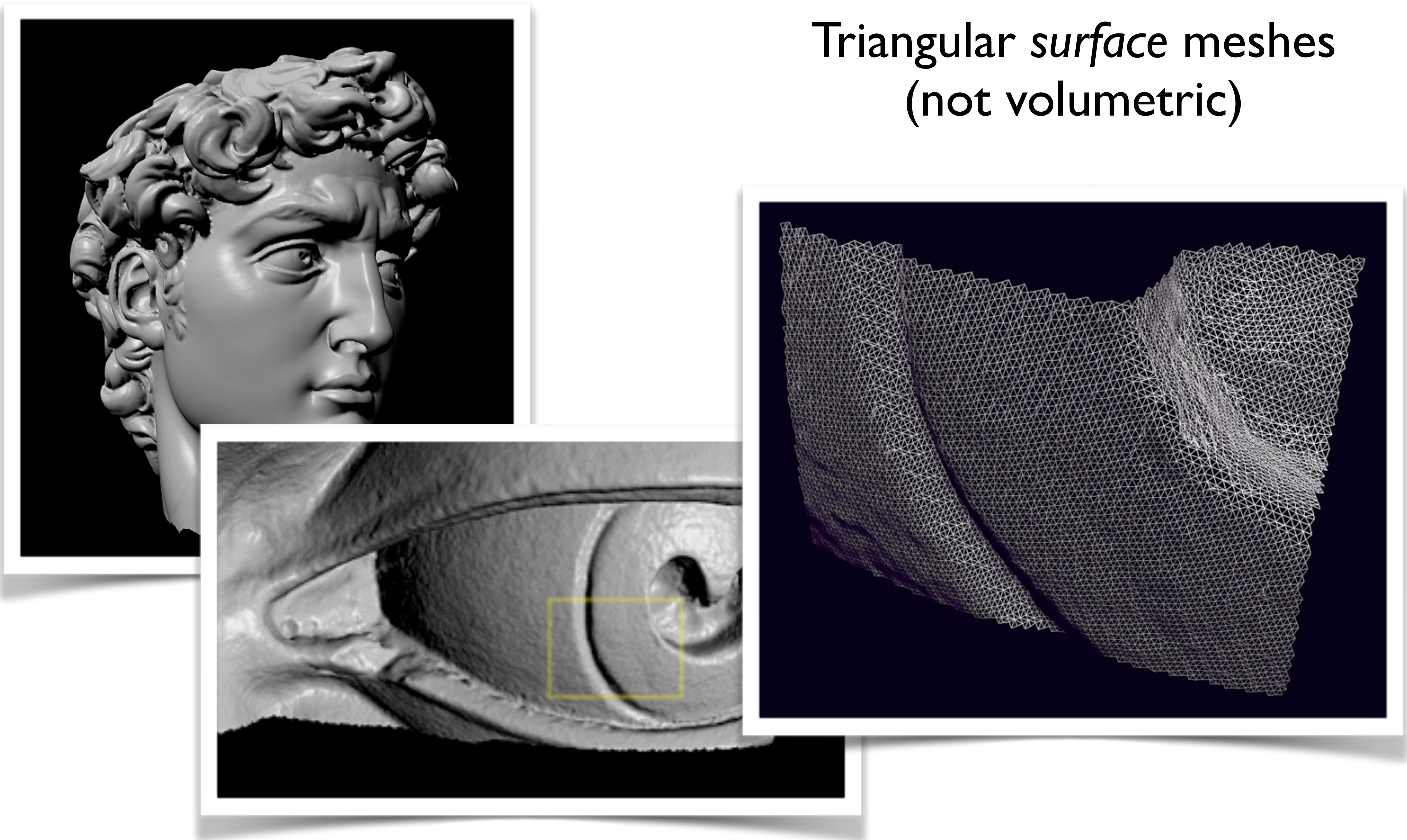
Tetrahedral meshes
(volumetric)



Hexahedral meshes
(volumetric)

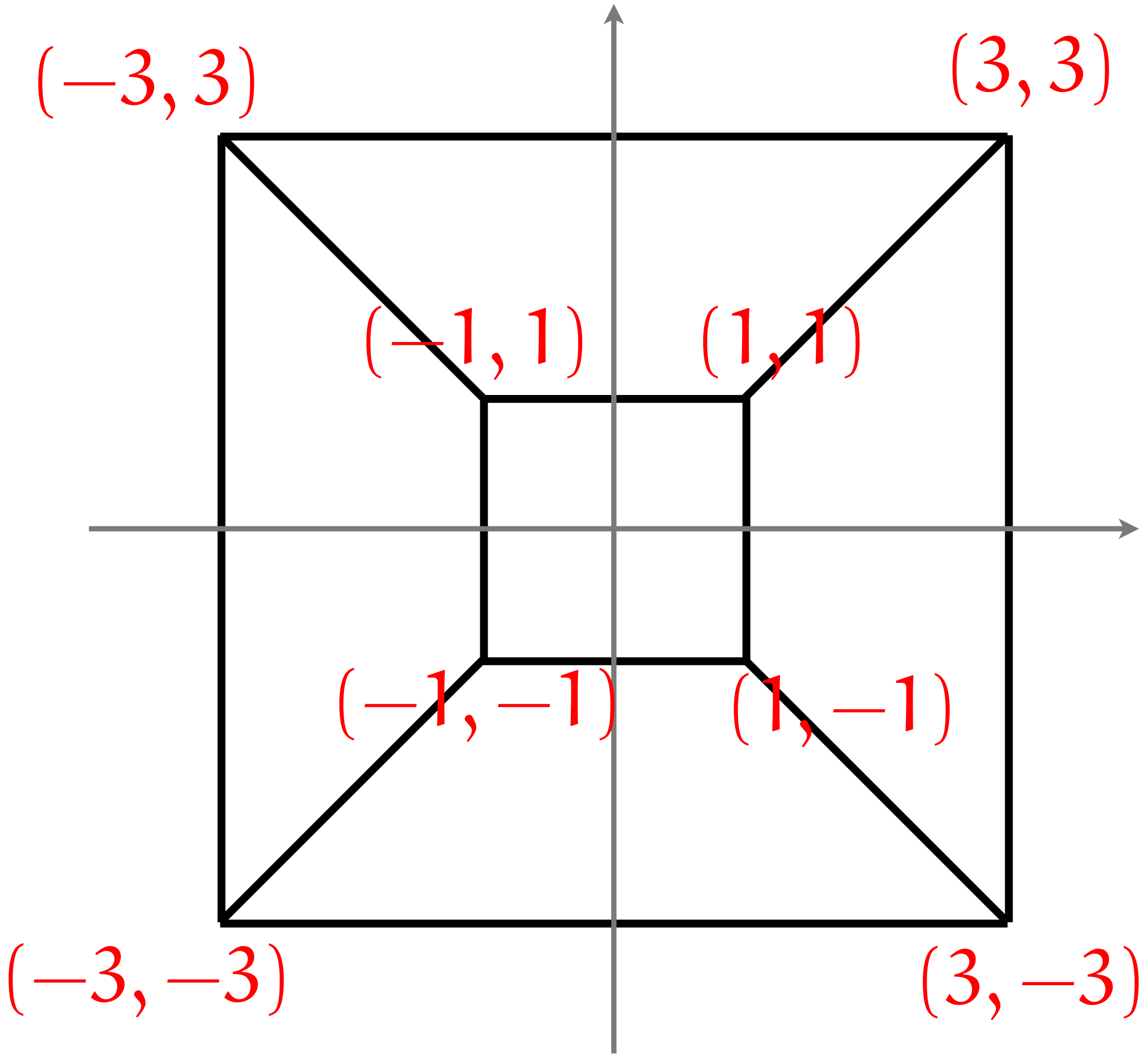
Geometry representations

Triangular *surface* meshes
(not volumetric)



Explicit meshes

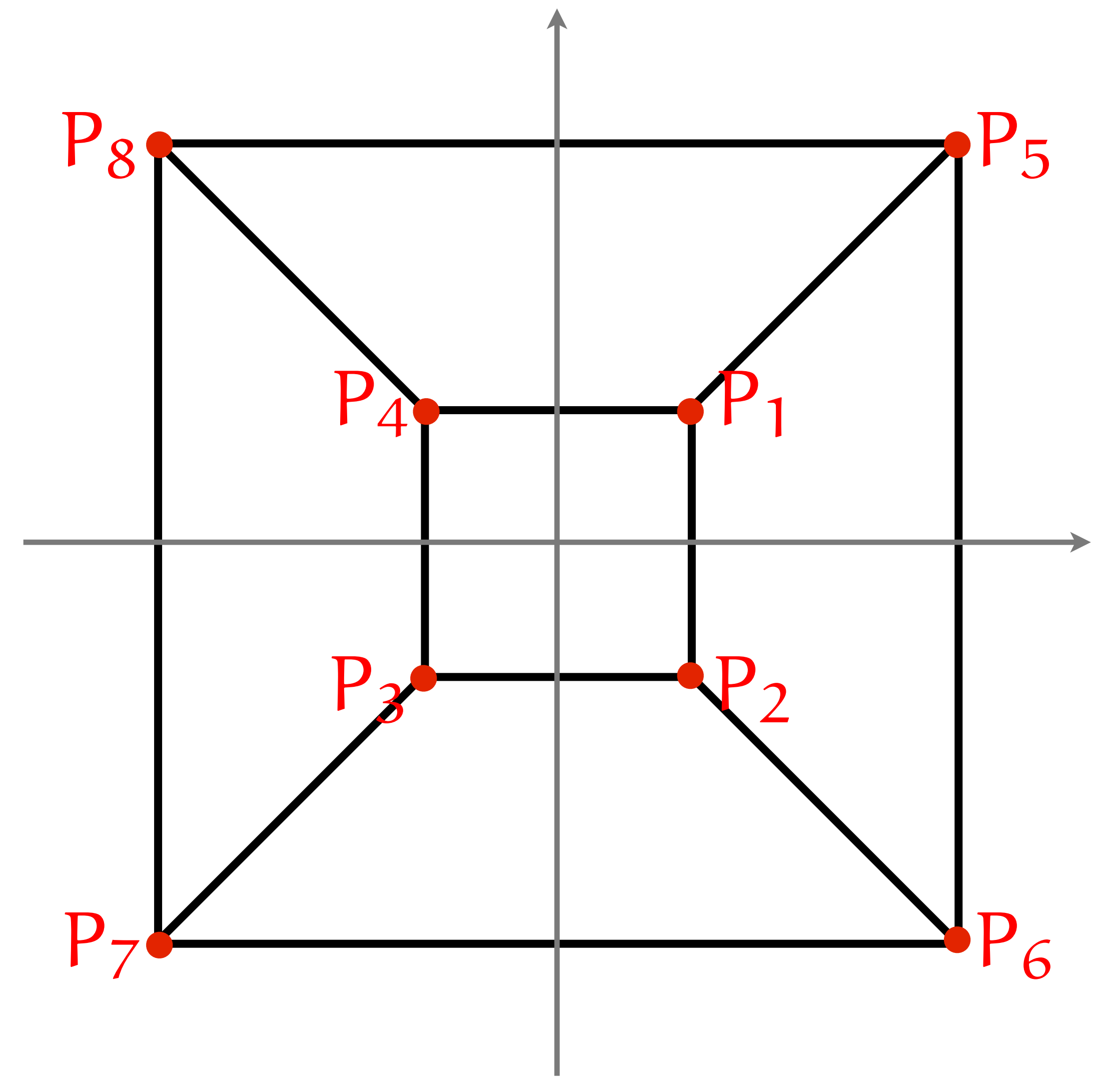
Example:
A quadrilateral mesh



Explicit meshes

Vertex data structure

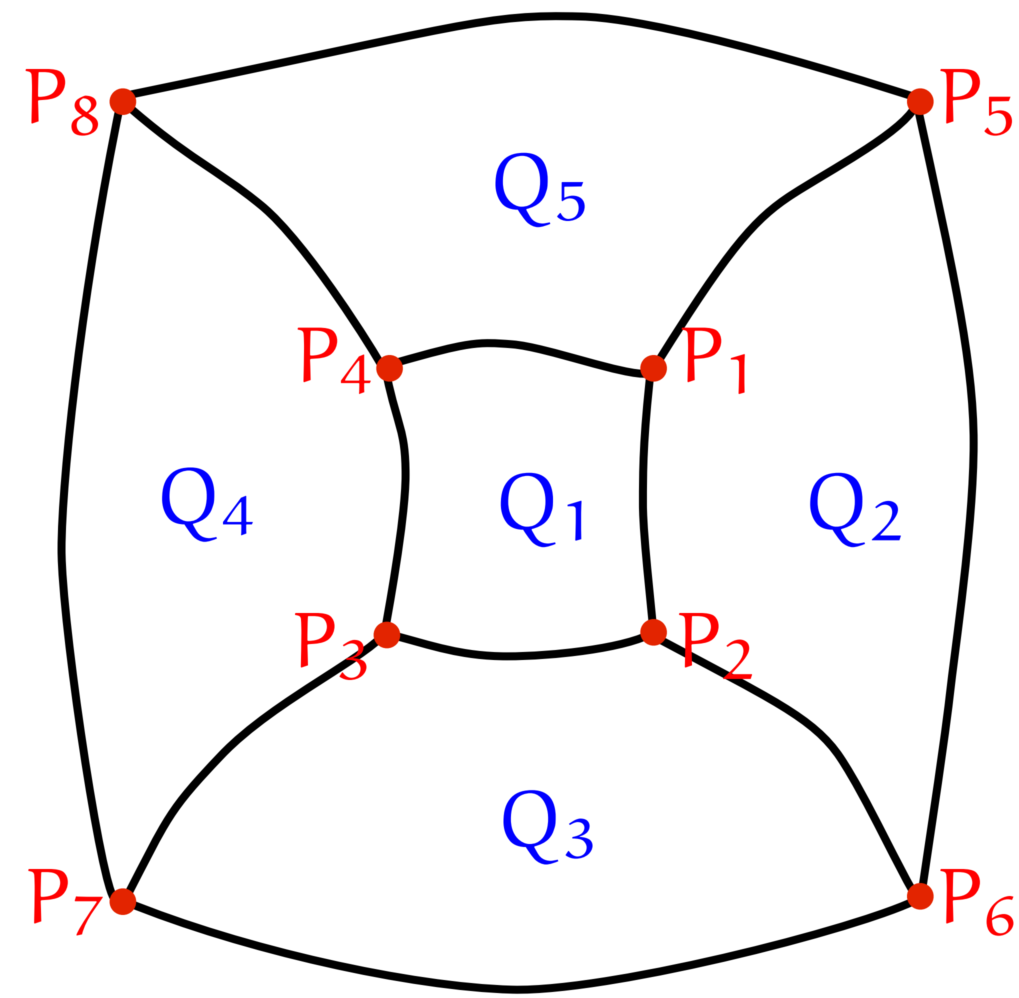
Vertex ID	Position
P_1	(1, 1)
P_2	(1, -1)
P_3	(-1, -1)
P_4	(-1, 1)
P_5	(3, 3)
P_6	(3, -3)
P_7	(-3, -3)
P_8	(-3, 3)



Explicit meshes

Mesh data structure

Element (Quad) ID	Vertices
Q_1	(P_1, P_2, P_3, P_4)
Q_2	(P_1, P_5, P_6, P_2)
Q_3	(P_2, P_6, P_7, P_3)
Q_4	(P_3, P_7, P_8, P_4)
Q_5	(P_4, P_8, P_5, P_1)



Implicit surfaces - levelsets

- Motivation
  - ✓ Accelerated geometric queries for problems such as:
    - ➡ Is a point (x*,y*) inside the object?
    - ➡ Is a point (x*,y*) within a distance of d* from the object surface?
    - ➡ What is the point on the surface which is closest to the query point (x*,y*)?

Implicit surfaces - levelsets

- Motivation
  - ✓ Easy modeling of motions that involve topological change, e.g. shapes splitting or merging

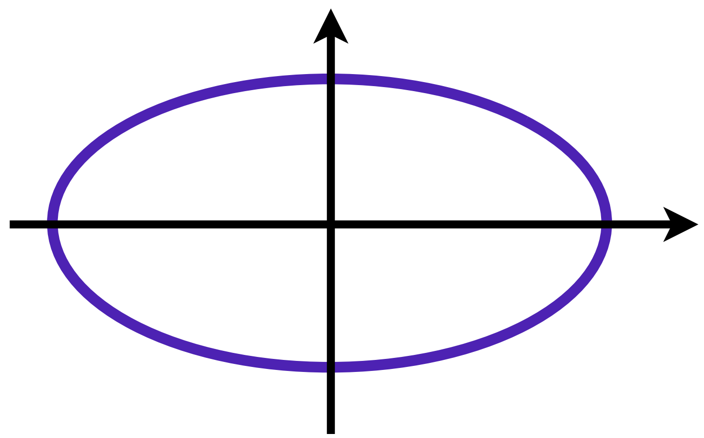
  - ✓ Such operations are difficult to encode with meshes, since they don't "split" or "merge" unless we force them to

Implicit surfaces - levelsets

- Familiar representations address some of these demands:

  ✓ e.g. Analytic equations

  ➡ For an ellipsis:



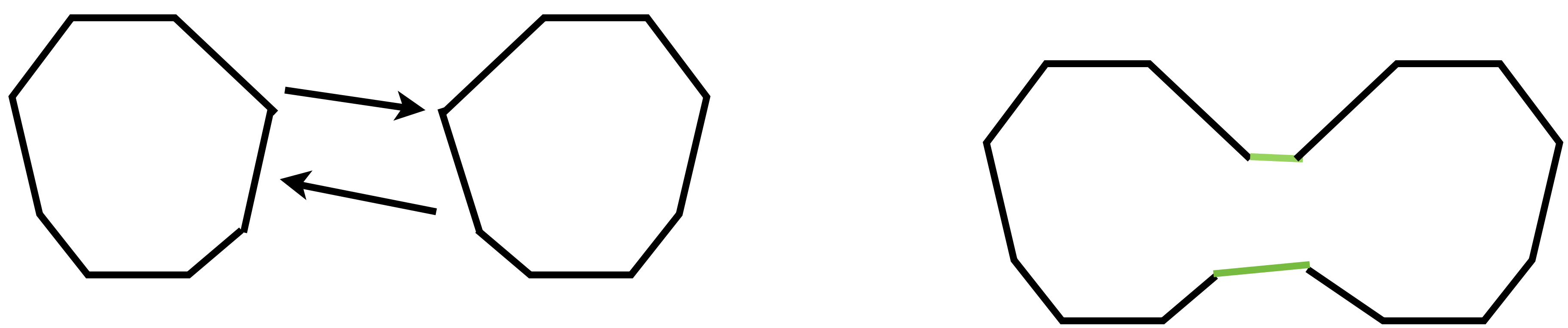
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

  ➡ Easy inside/outside tests

$$\frac{x_*^2}{a^2} + \frac{y_*^2}{b^2} < 1 \Leftrightarrow (x_*, y_*) \text{ is inside}$$

Implicit surfaces - levelsets

- Familiar representations address some of these demands:
  - ✓ Describe a closed region via its boundary; split and reconnect when necessary



➡ This may be tractable in isolated cases, but very cumbersome and impractical for more complicated cases, and with 3-dimensional surfaces

Implicit surfaces - levelsets

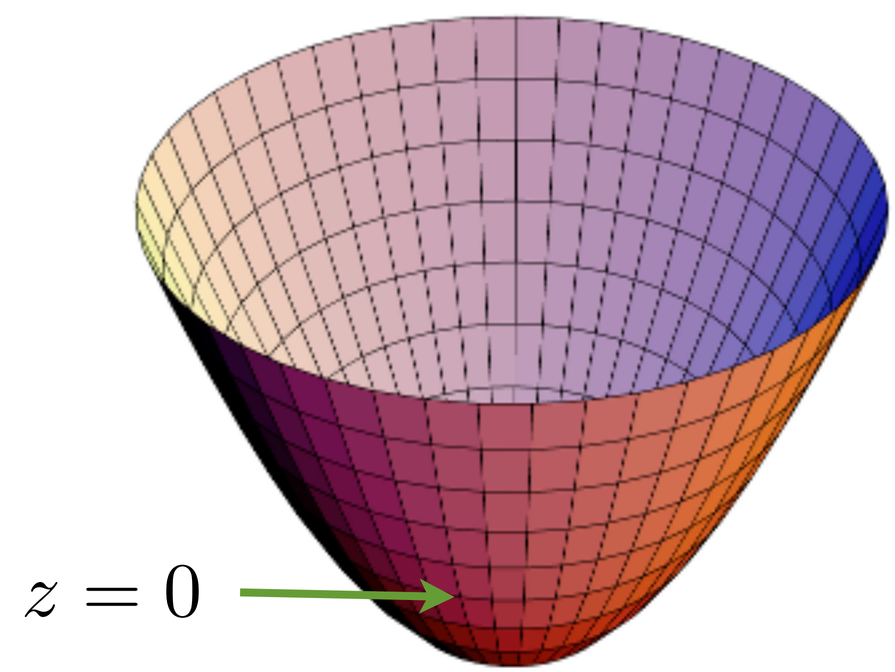
- Represent a curve in 2D (or, a surface in 3D) as the zero isocontour of a (continuous) function, i.e.

$$\mathcal{C} = \{(x.y) \in \mathbf{R}^2 : \phi(x,y) = 0\}$$

e.g.

$$\text{circle } x^2 + y^2 = R^2 \equiv \{(x,y) : \phi(x,y) = 0\}$$

$$where\ \phi(x,y) = x^2 + y^2 - R^2$$



Implicit surfaces - levelsets

- This representation may seem redundant (we store information everywhere, just to capture a curve), but it conveys important benefits:
  - ➡ Containment queries

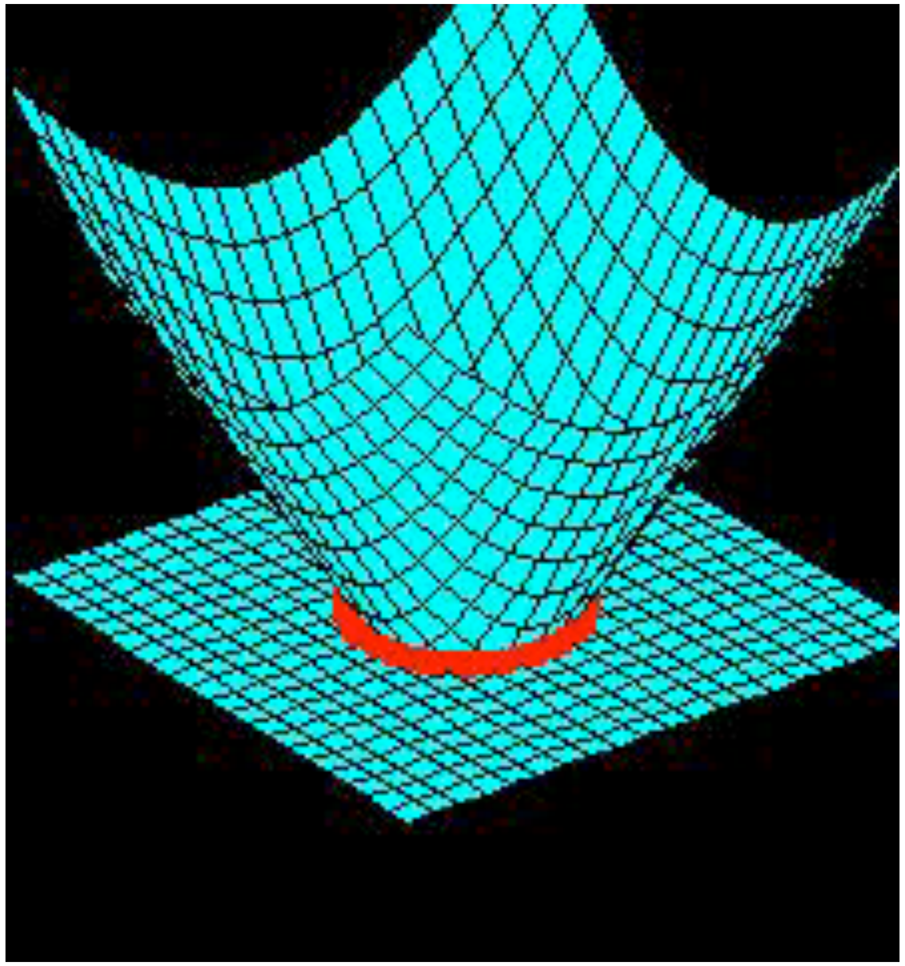
$$\text{Is } (x_*, y_*) \text{ inside } \mathcal{C}? \Leftrightarrow \phi(x_*, y_*) < 0$$

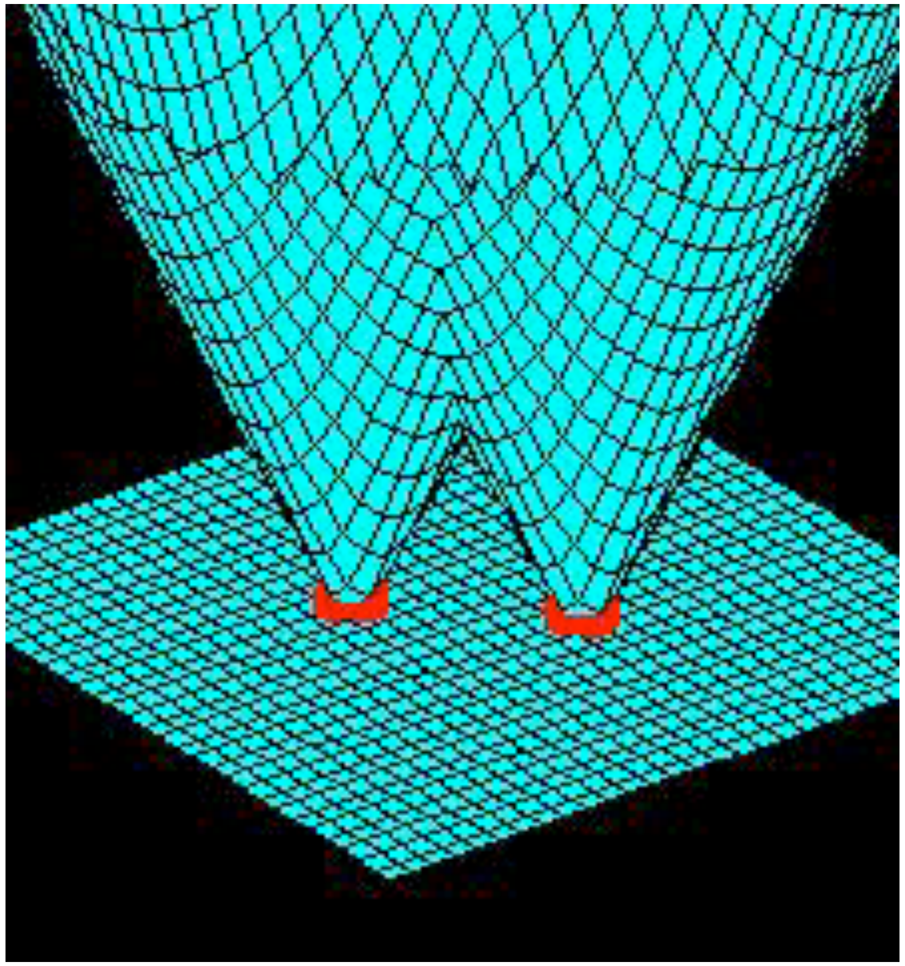
  - ➡ Composability

$$\left.\begin{array}{l} \phi_1(x,y) \text{ encodes } \Omega_1 \\ \phi_2(x,y) \text{ encodes } \Omega_2 \end{array}\right\} \Rightarrow \quad \begin{array}{l} \max(\phi_1, \phi_2) \text{ encodes } \Omega_1 \cap \Omega_2 \\ \max(\phi_1, \phi_2) \text{ encodes } \Omega_1 \cup \Omega_2 \end{array}$$

  - ➡ We model both shape & topology change by simply varying the level set function

Implicit surfaces - levelsets





Contouring - marching cubes