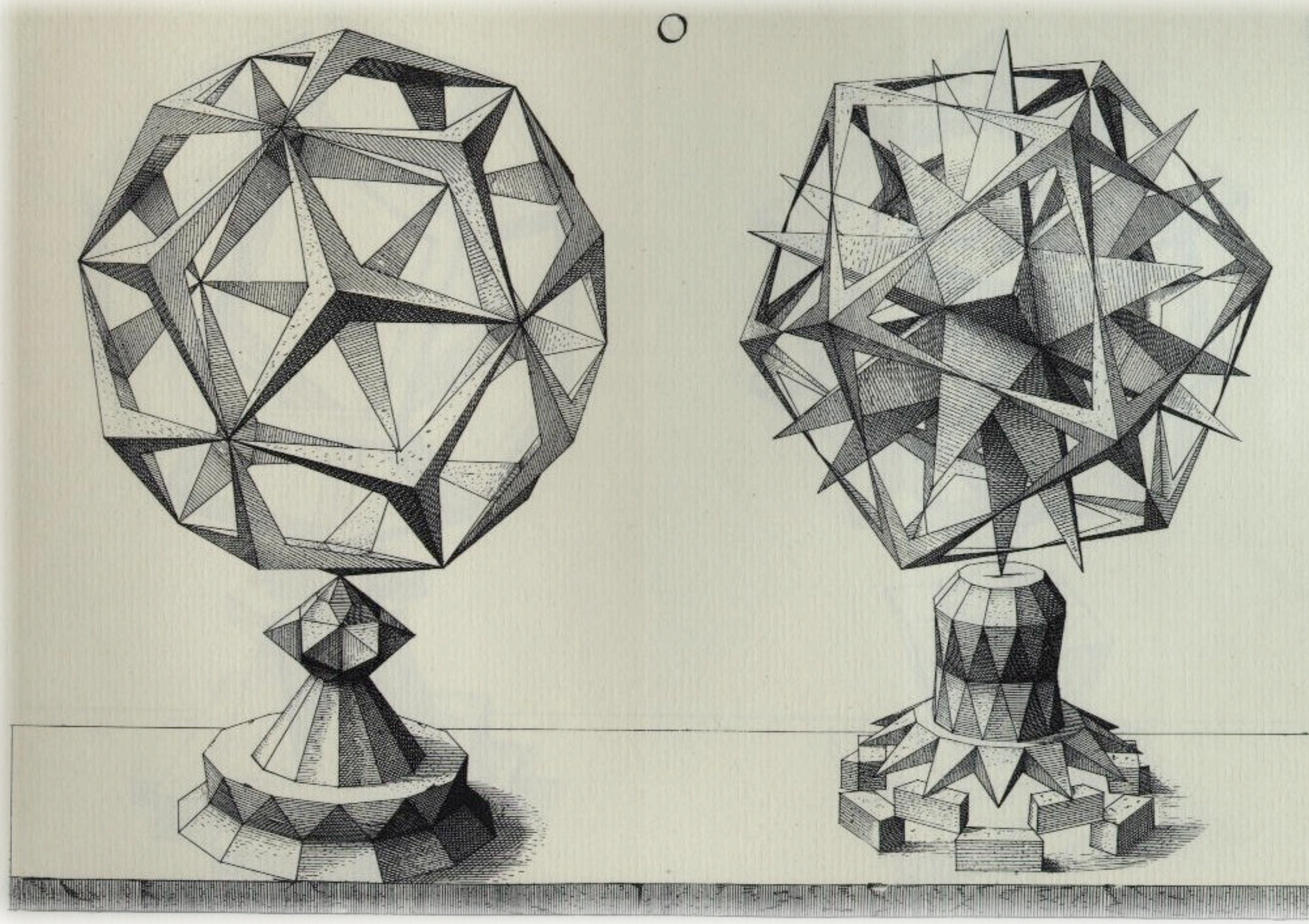
CS277 - Experimental Haptics
Lecture 7

Collision Detection II

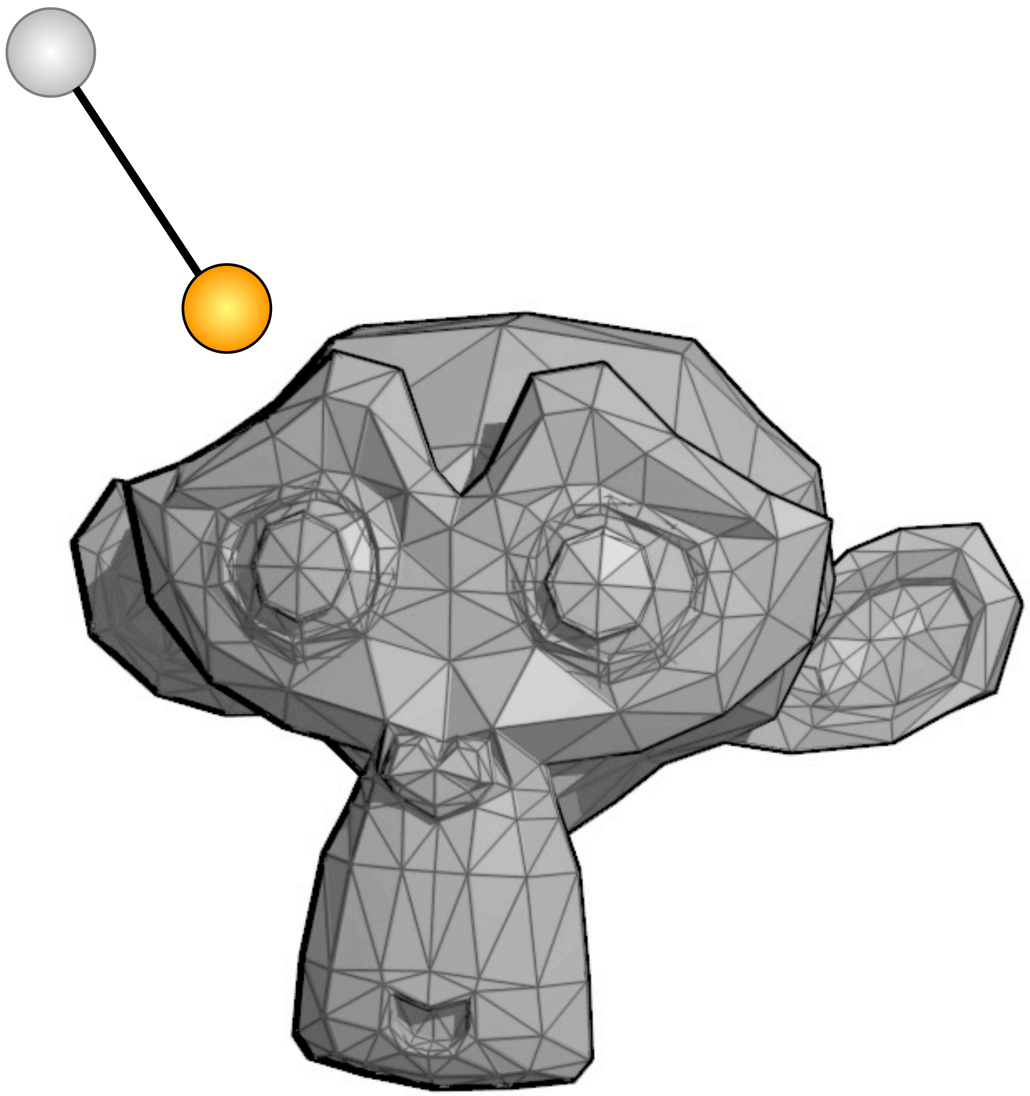
Outline

- Problem definition and motivation
- Bounding volume hierarchies
- Spatial partitioning approaches
- Point-sampled surfaces

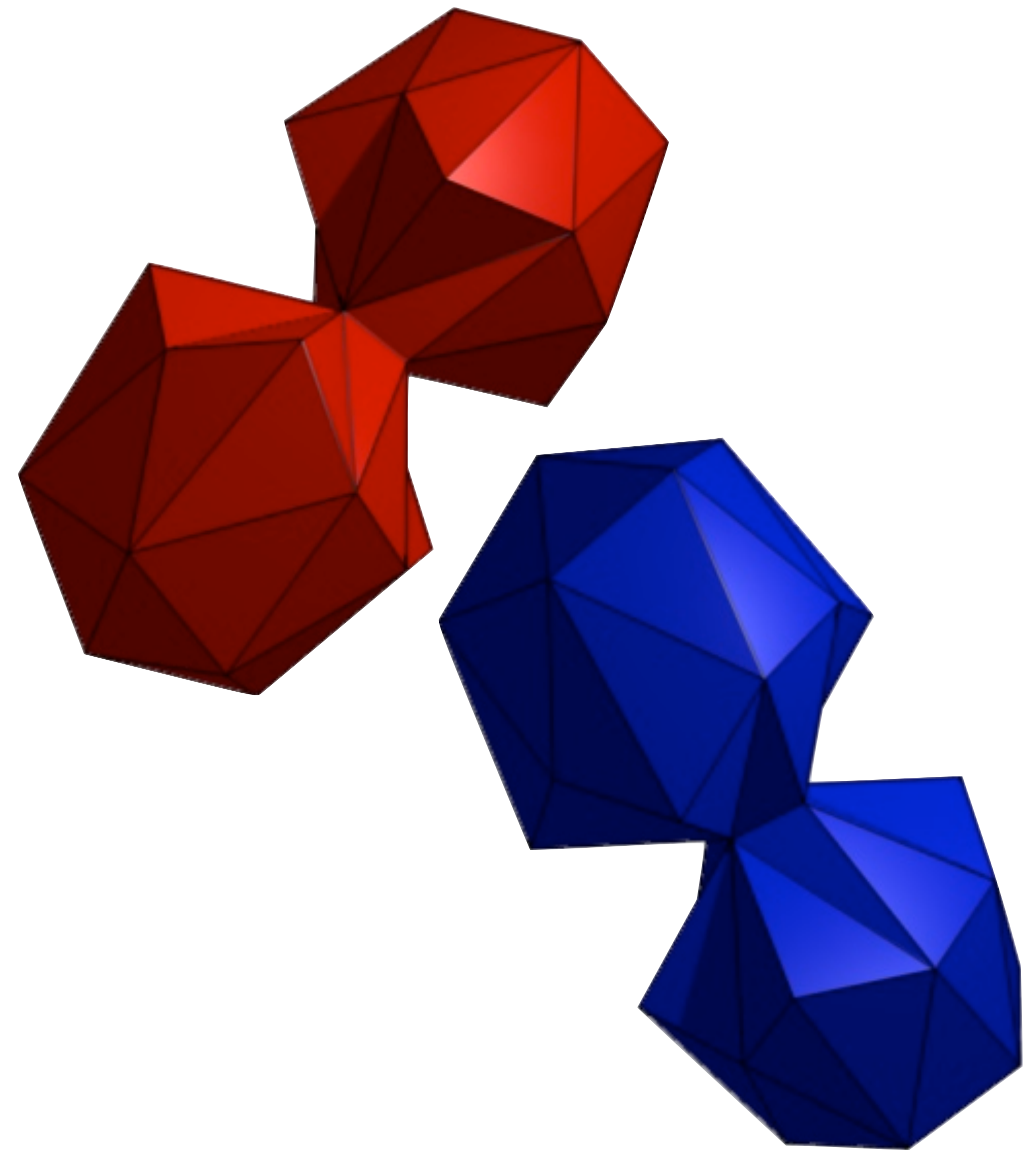


Polyhedron Tests

Polyhedron Intersection Tests

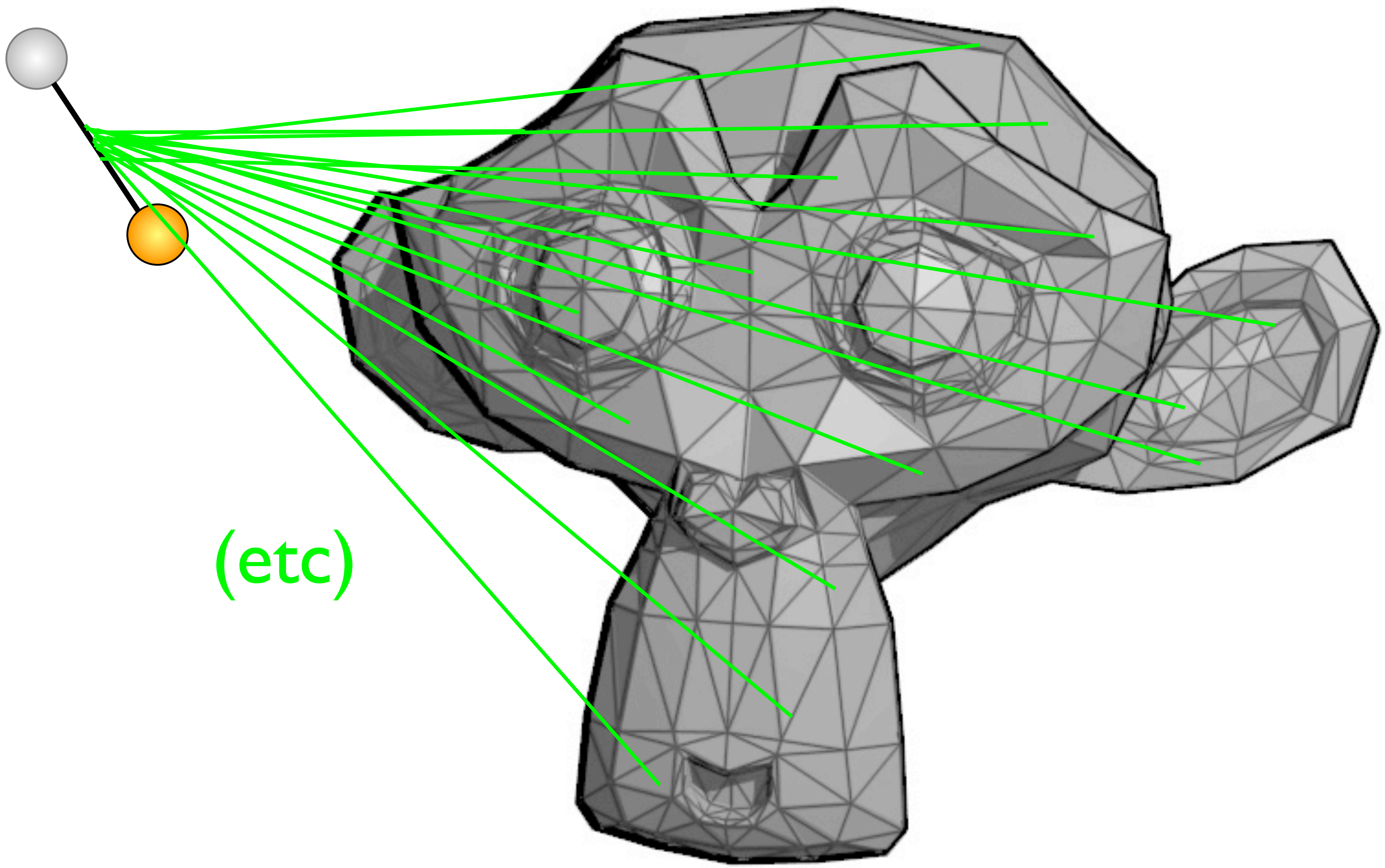


Segment-Mesh



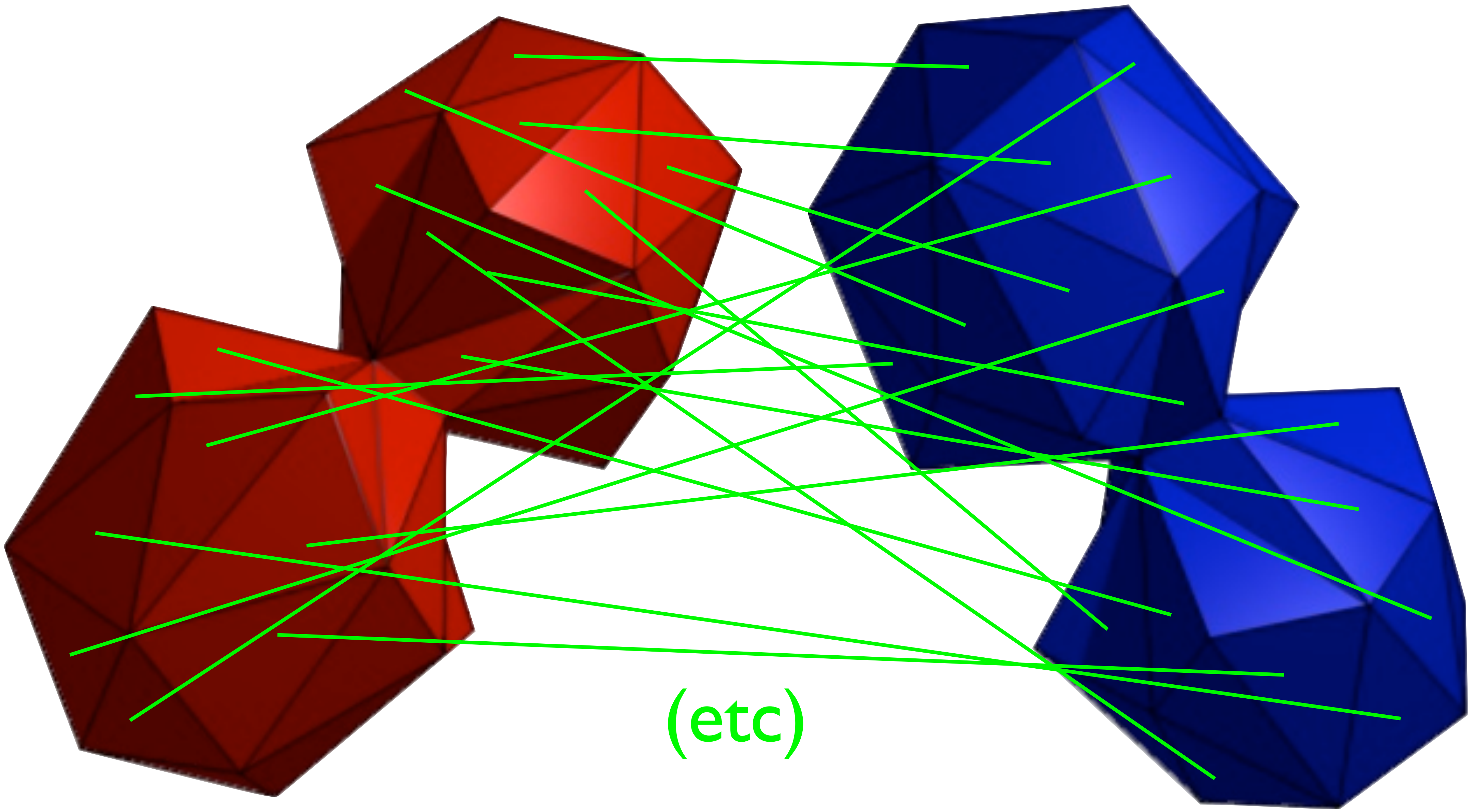
Mesh-Mesh

Brute-Force Approach



Test segment against every primitive: O(n) complexity

Brute-Force Approach



Test every pair of primitives for possible intersection:
O(mn) complexity

Too Slow!

- Haptic rendering requires us to compute collisions within a millisecond time interval
- Typical meshes have thousands of primitives
- Collision detection is a search problem
  - Recall what you learned in CS161
- Divide-and-conquer paradigm:
  - We can accelerate the operation by organizing our geometry into a tree data structure!

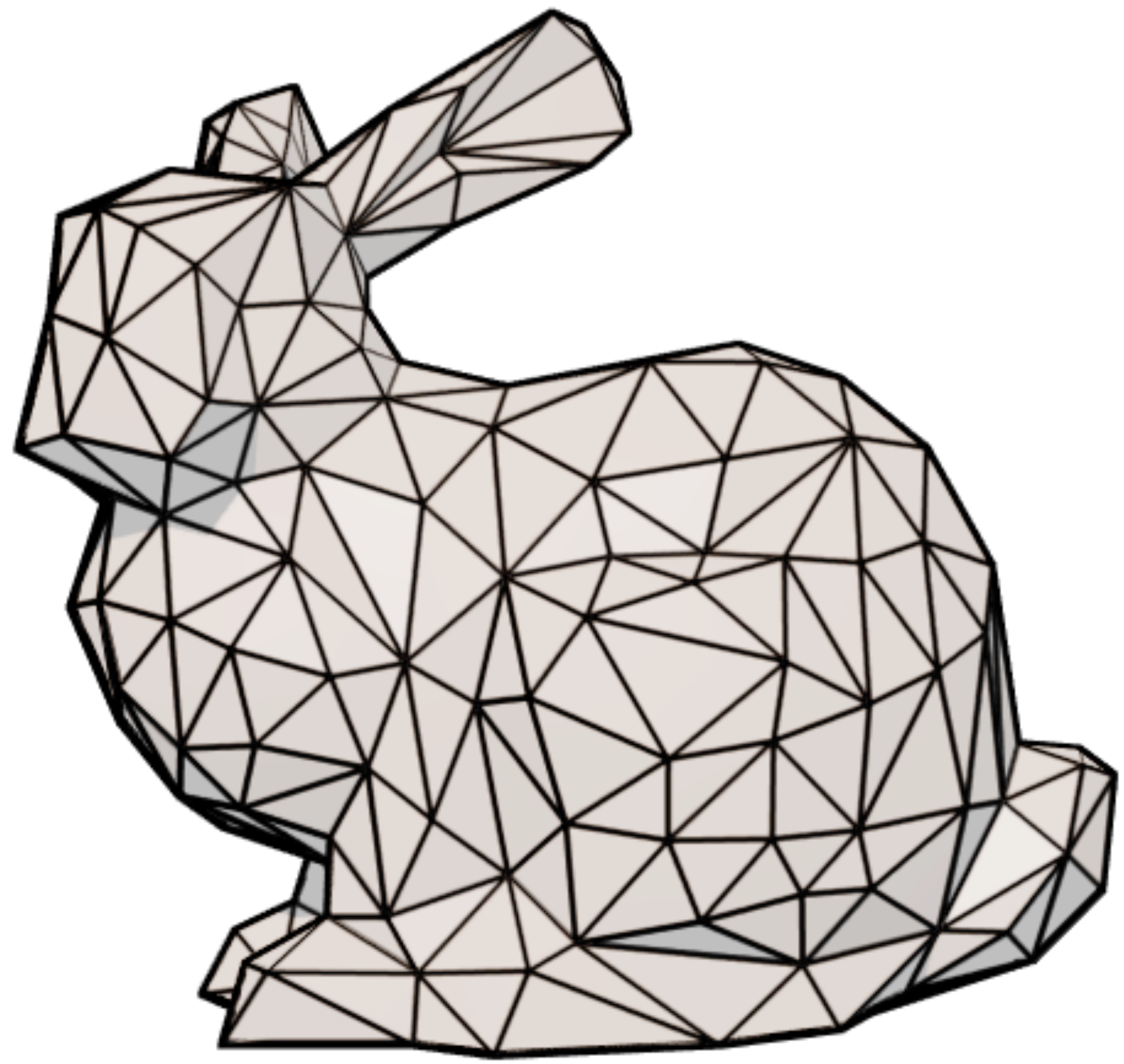
Two Approaches

- Bounding volume hierarchy
  - Partitions the object itself into smaller chunks that are fit within simple geometric primitives
- Spatial subdivision
  - Partitions the underlying space the object sits in

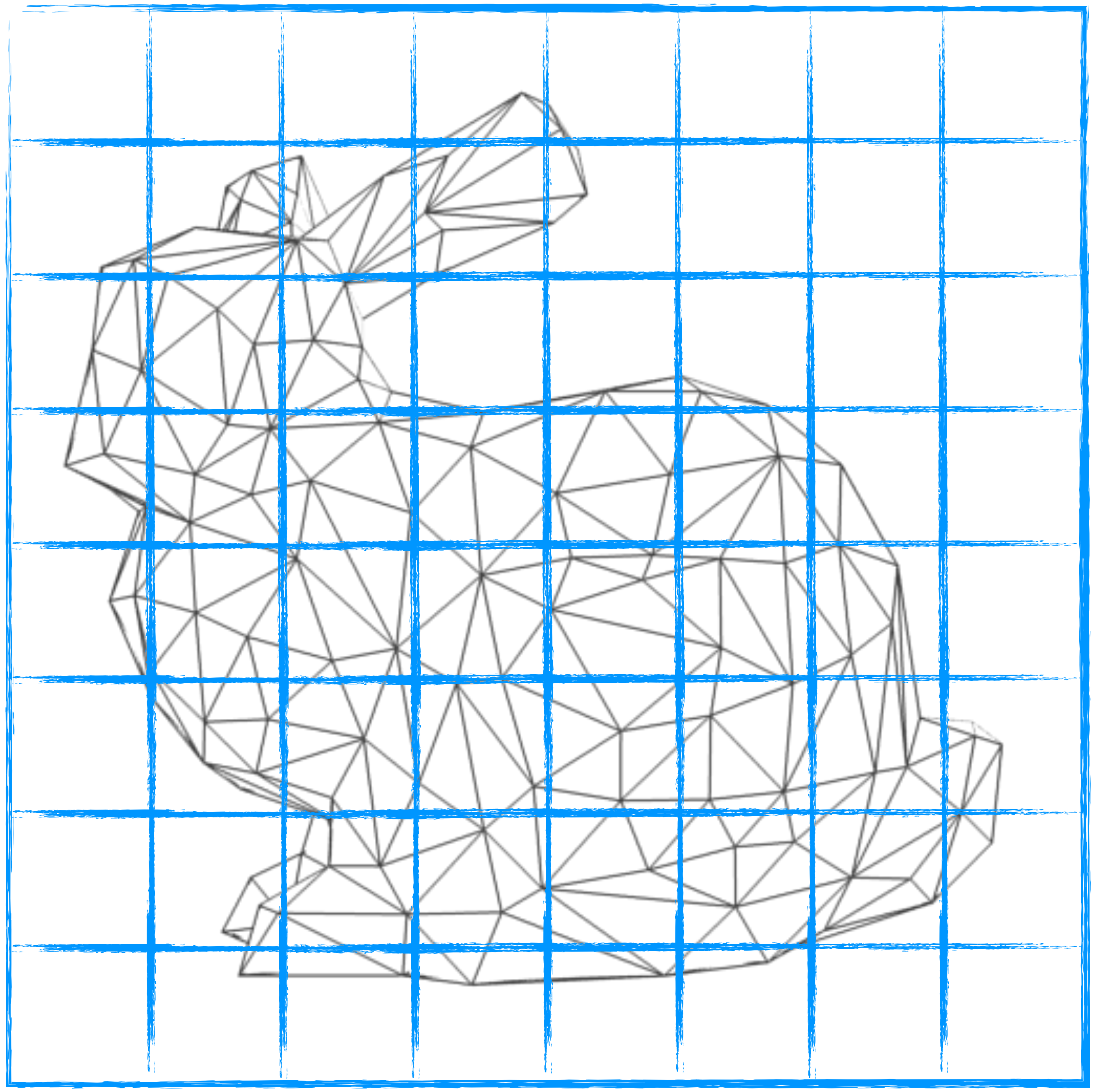
Spatial Partitioning

- Most direct extension of a binary search tree to three (or more!) dimensions
- Partitioning is more flexible, and can cake different forms:
  - Spatial hash (not really a tree)
  - Quadtree / octree
  - *k*-dimensional (*k*-D) tree
  - Binary space partition (BSP) tree

A Few Examples...



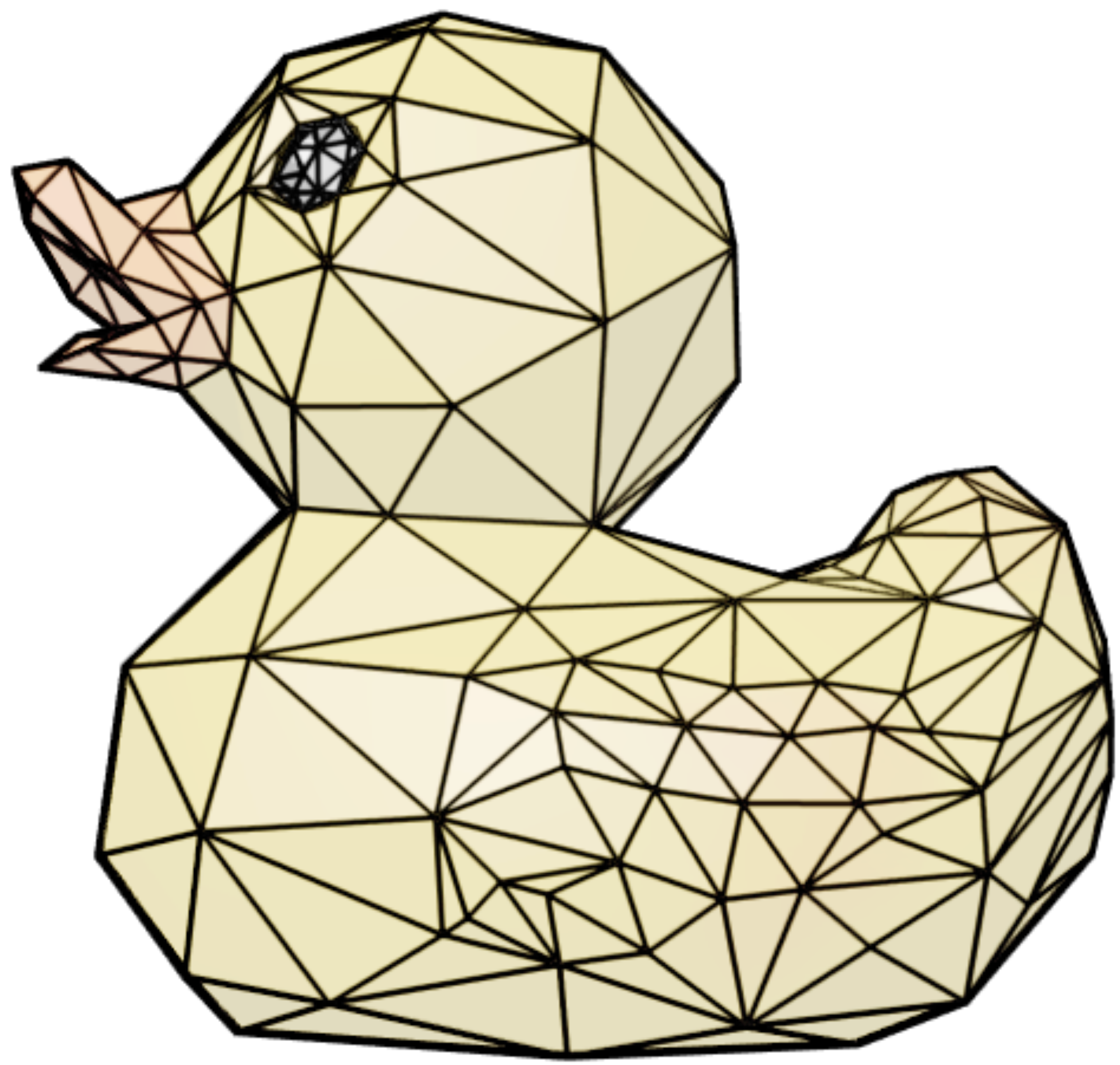
Spatial Hashing



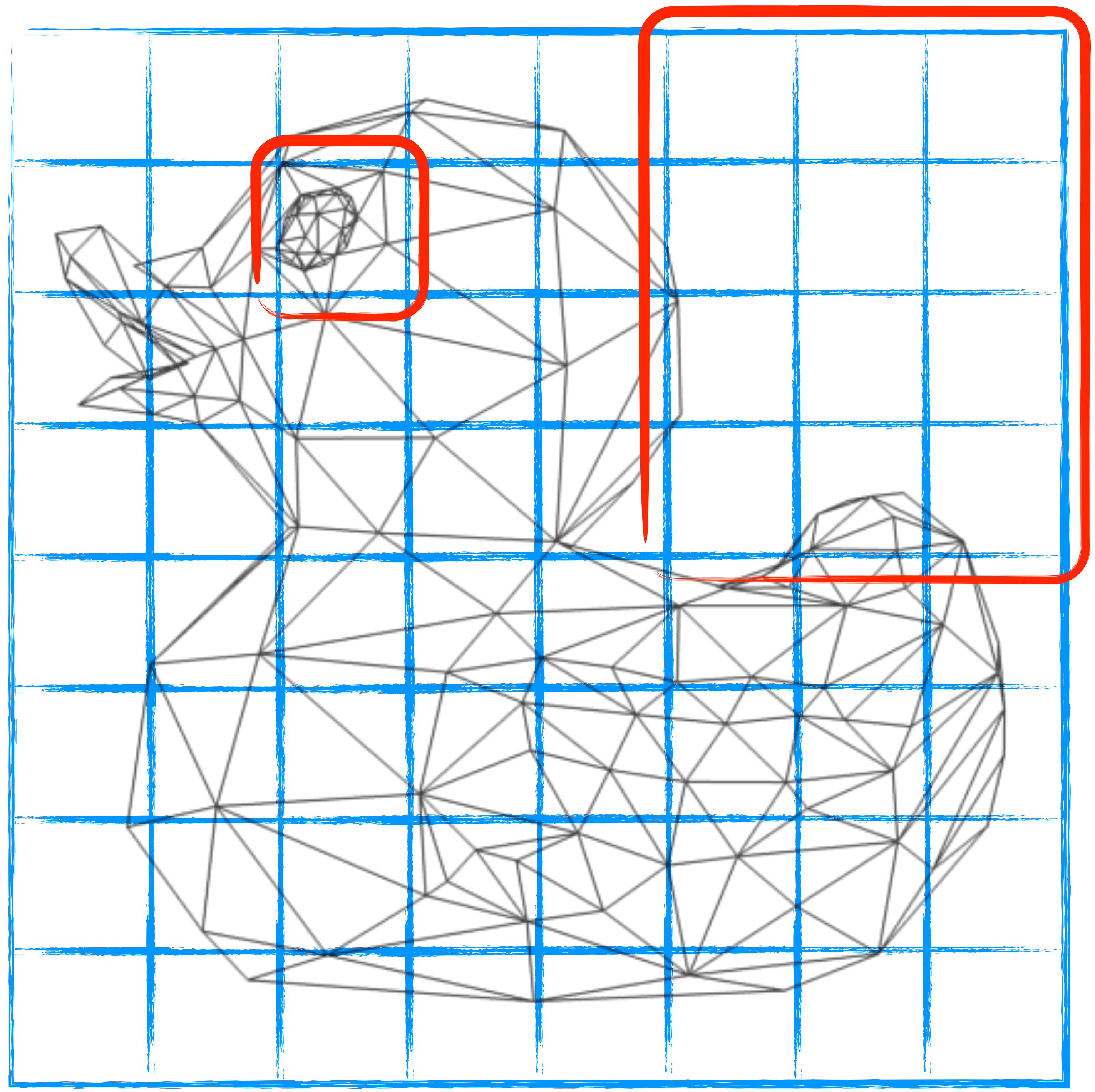
Spatial Hashing

- Extremely easy to implement
- Can provide constant time collision queries in the ideal case
- How do we decide what the grid spacing should be?

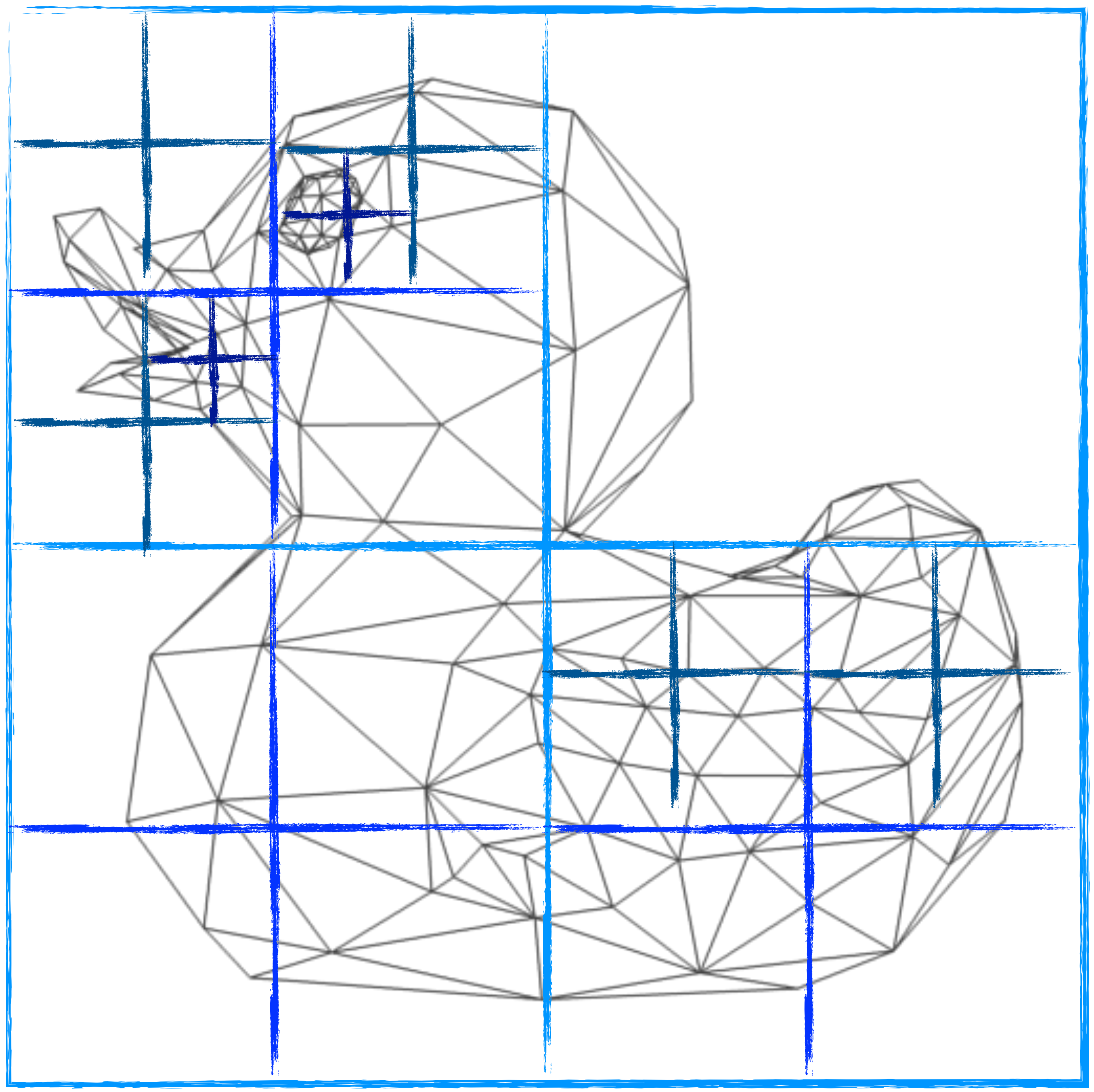
What about our other friend?



Spatial Hashing Limitations



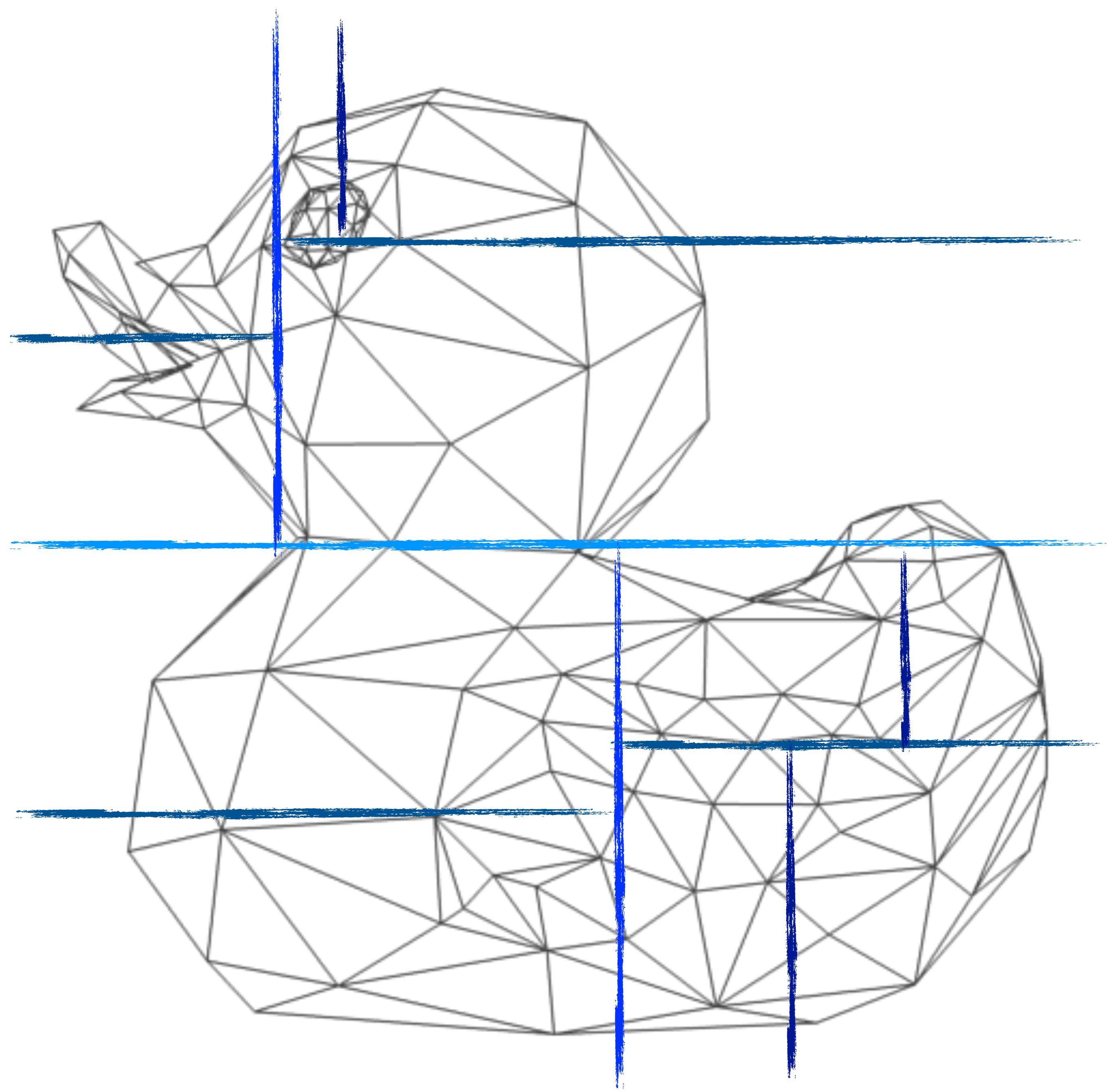
Quadtree / Octree



Quadtree / Octree

- Very simple to implement
- Does not make any effort to partition the space efficiently
- Has a high branching factor
- Can be efficient when data is uniform

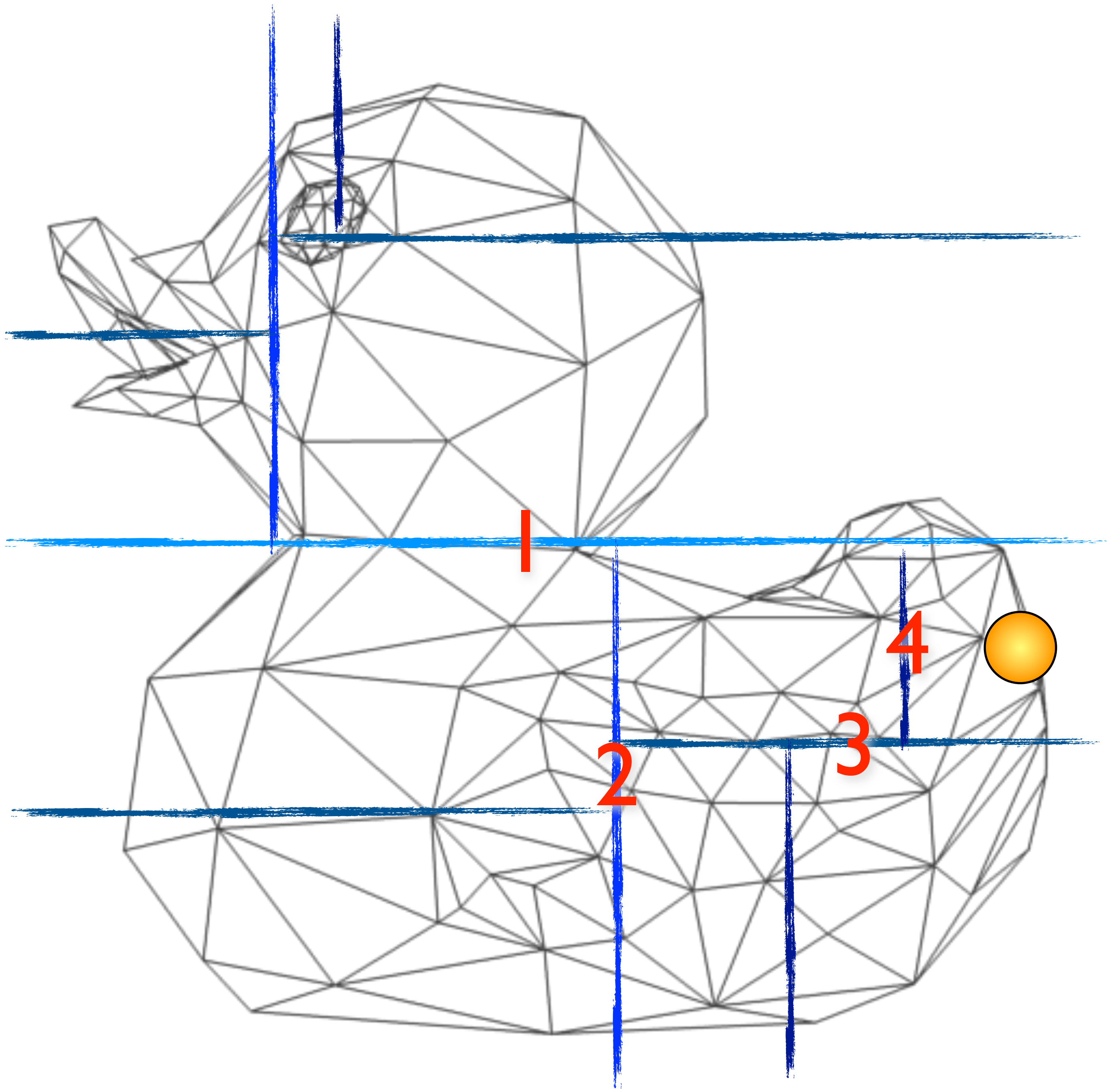
k-Dimensional Tree



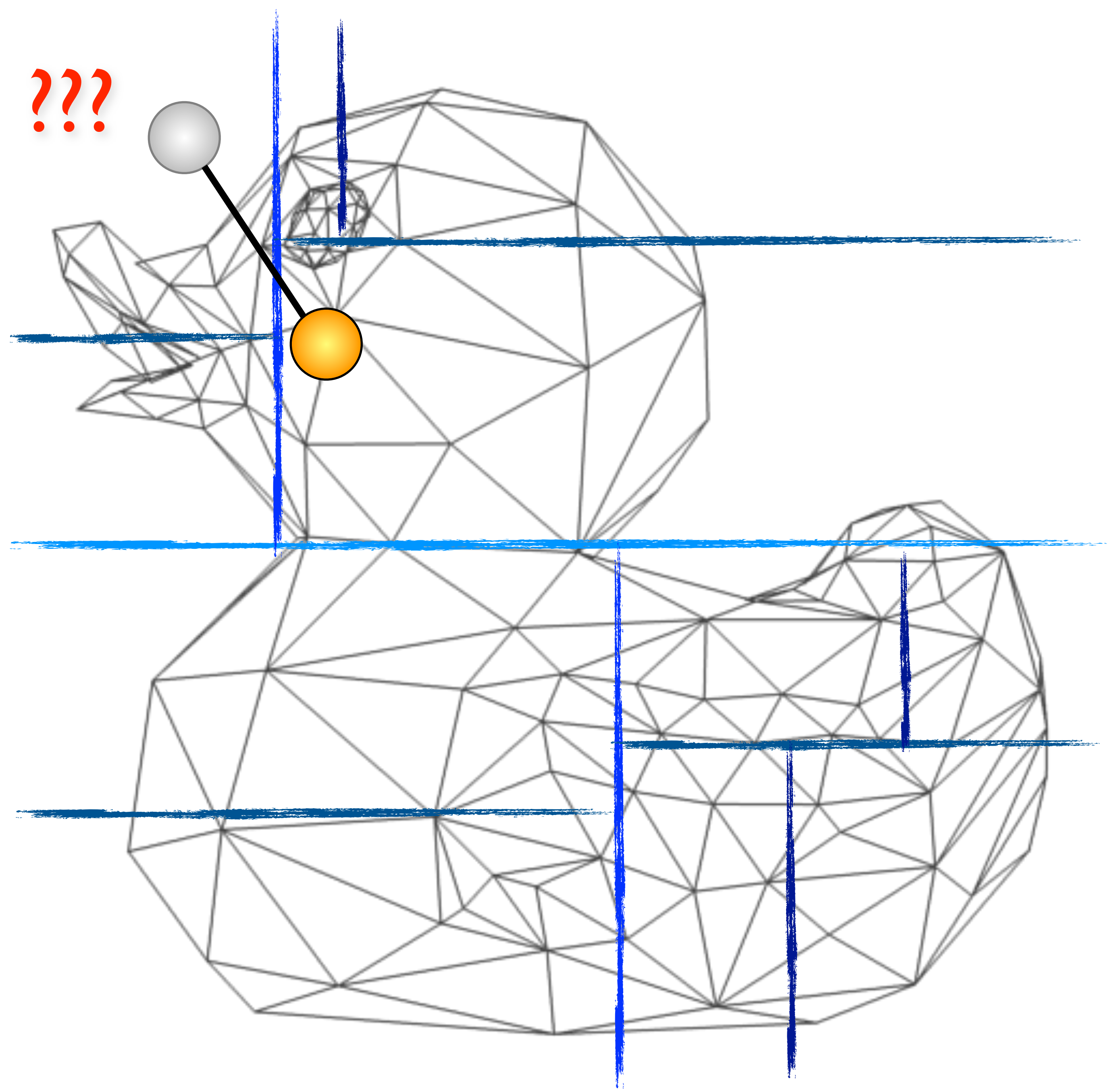
k-Dimensional Trees

- Binary tree that partitions space along an axis-aligned plane
- Adaptive to the characteristics of the input geometry (more balanced tree)
- Many partitioning heuristics for construction:
  - Alternating x-y-z axes
  - Equal count vs. equal volume

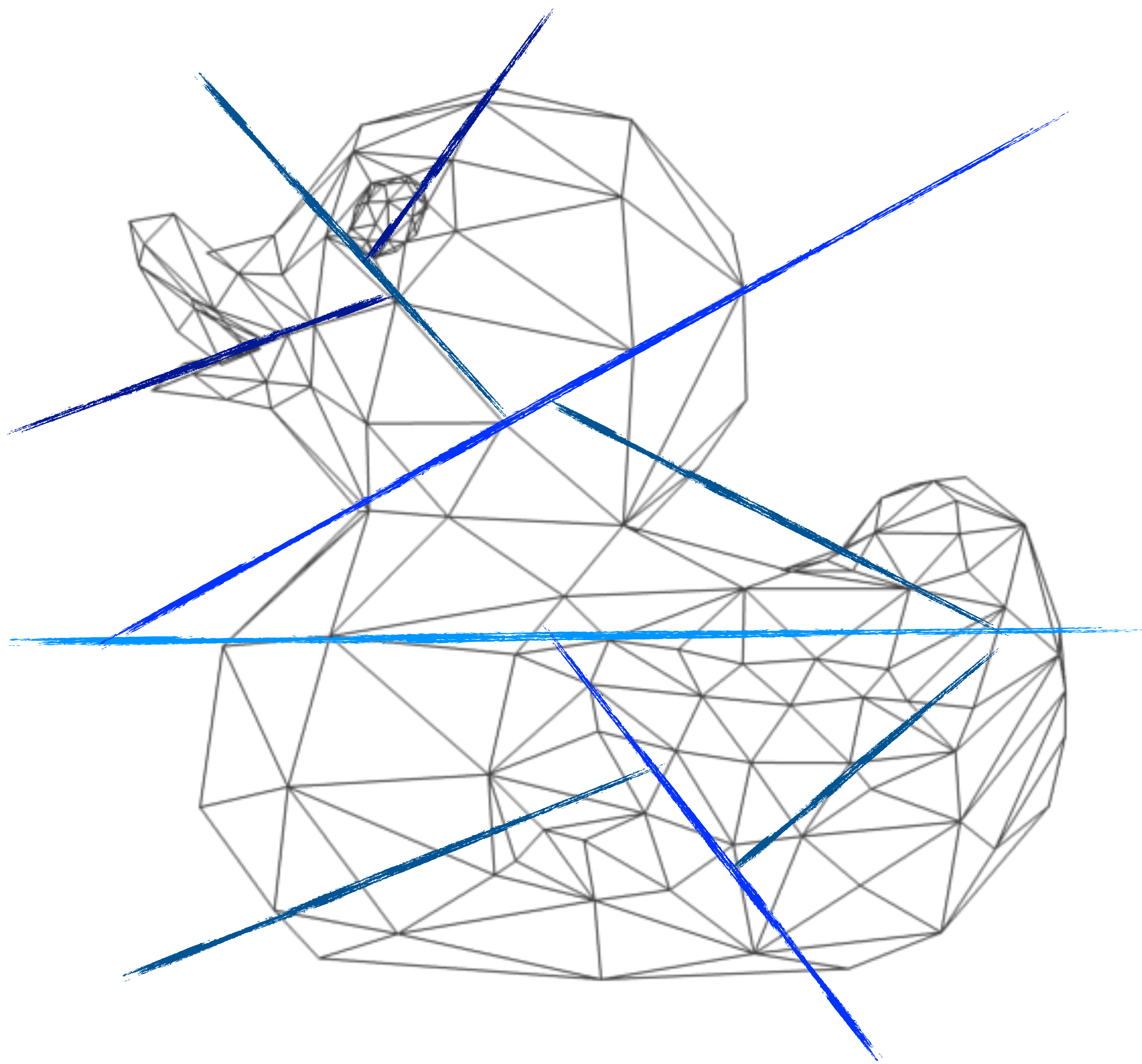
Searching a k-D Tree



What About a Segment?



Binary Space Partition Tree



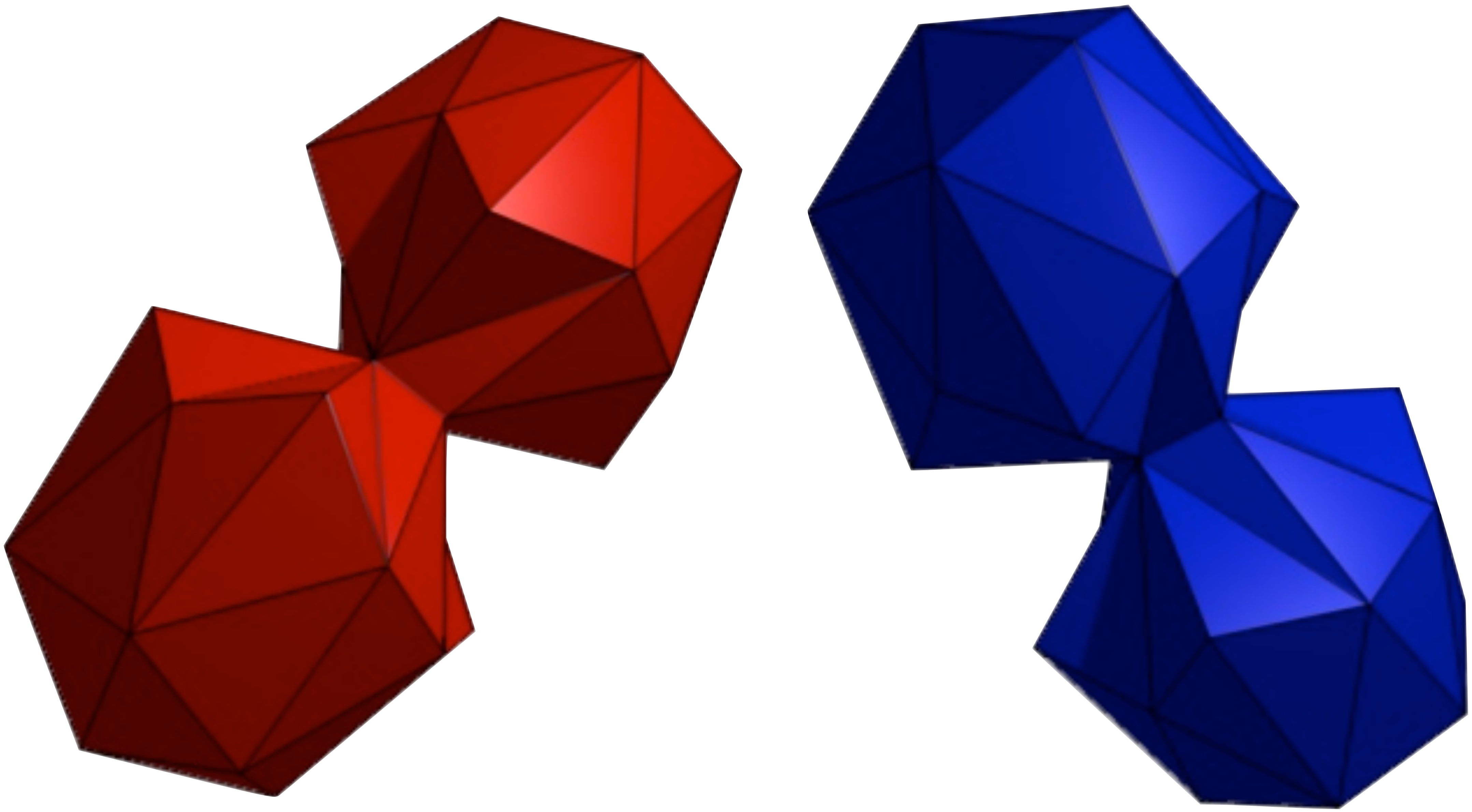
Binary Space Partition Tree

- Allows splitting along arbitrary plane
- Fewer objects or primitives are "split in the middle"
- Can require more effort to construct
- Slightly more storage overhead than a k-D tree

Spatial Partitioning Summary

- Different partitioning structures are embodiments of the same principle
- Supports O(log n) time query for a point and expected logarithmic time for a ray or segment
- Choose which one to use based on the characteristics of the geometry

The (Second) Task at Hand



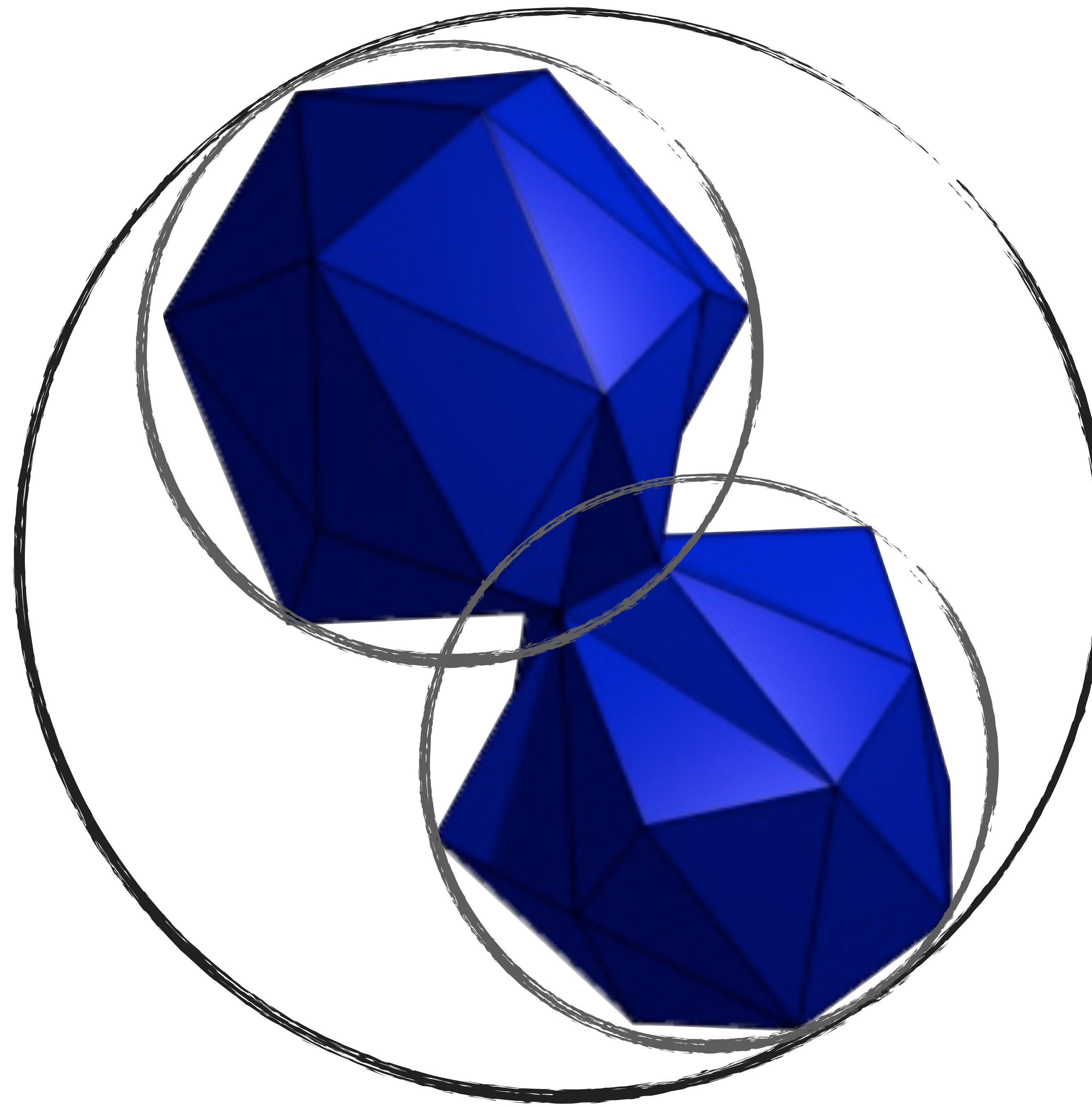
How do we detect collision between two complex meshes?

Bounding Volume Hierarchies

- Similar idea to spatial partitioning, but break up the object instead
- Takes advantage of *spatial coherence*
- When objects collide, the contact set is generally small relative to the mesh size

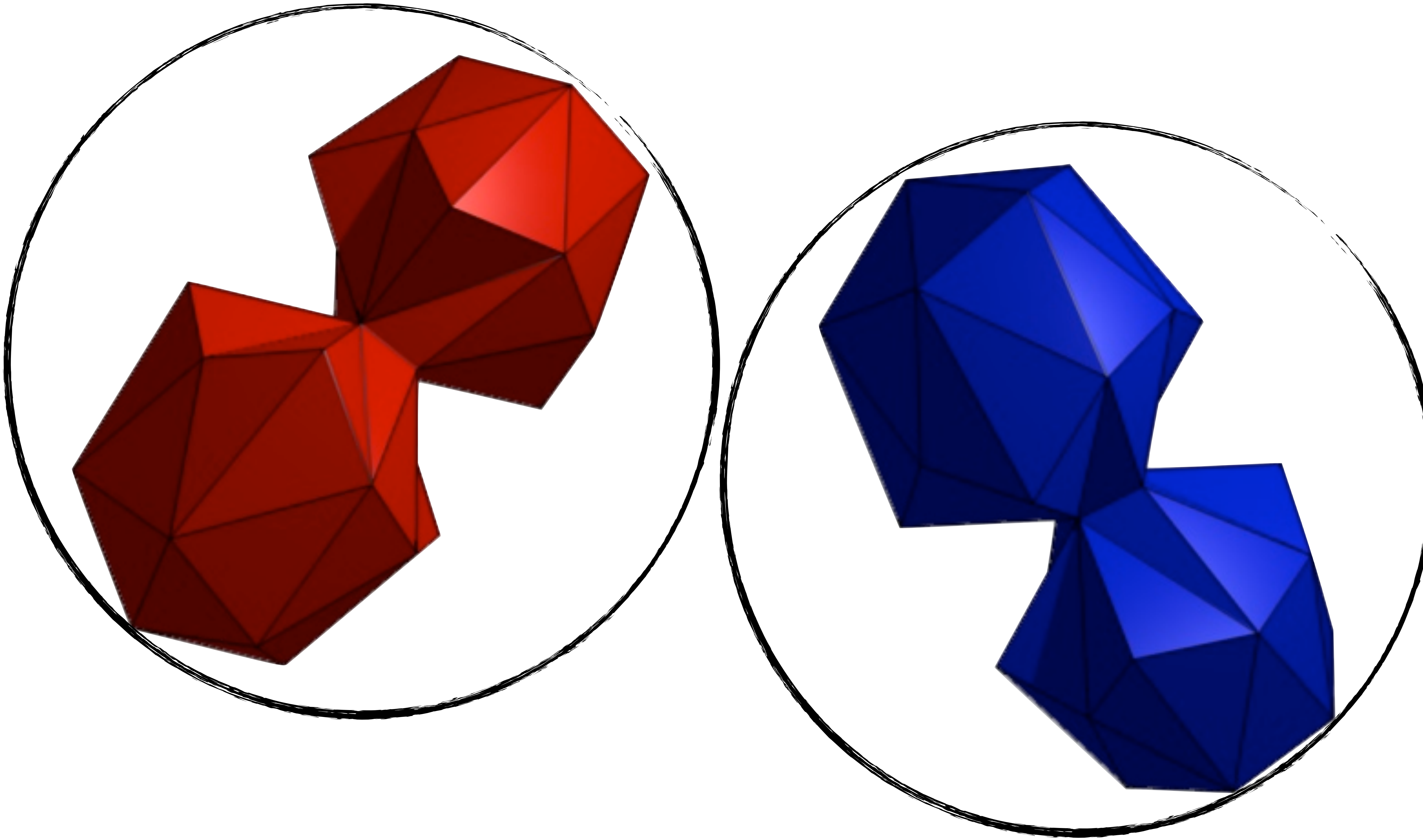
Bounding Volume Hierarchies

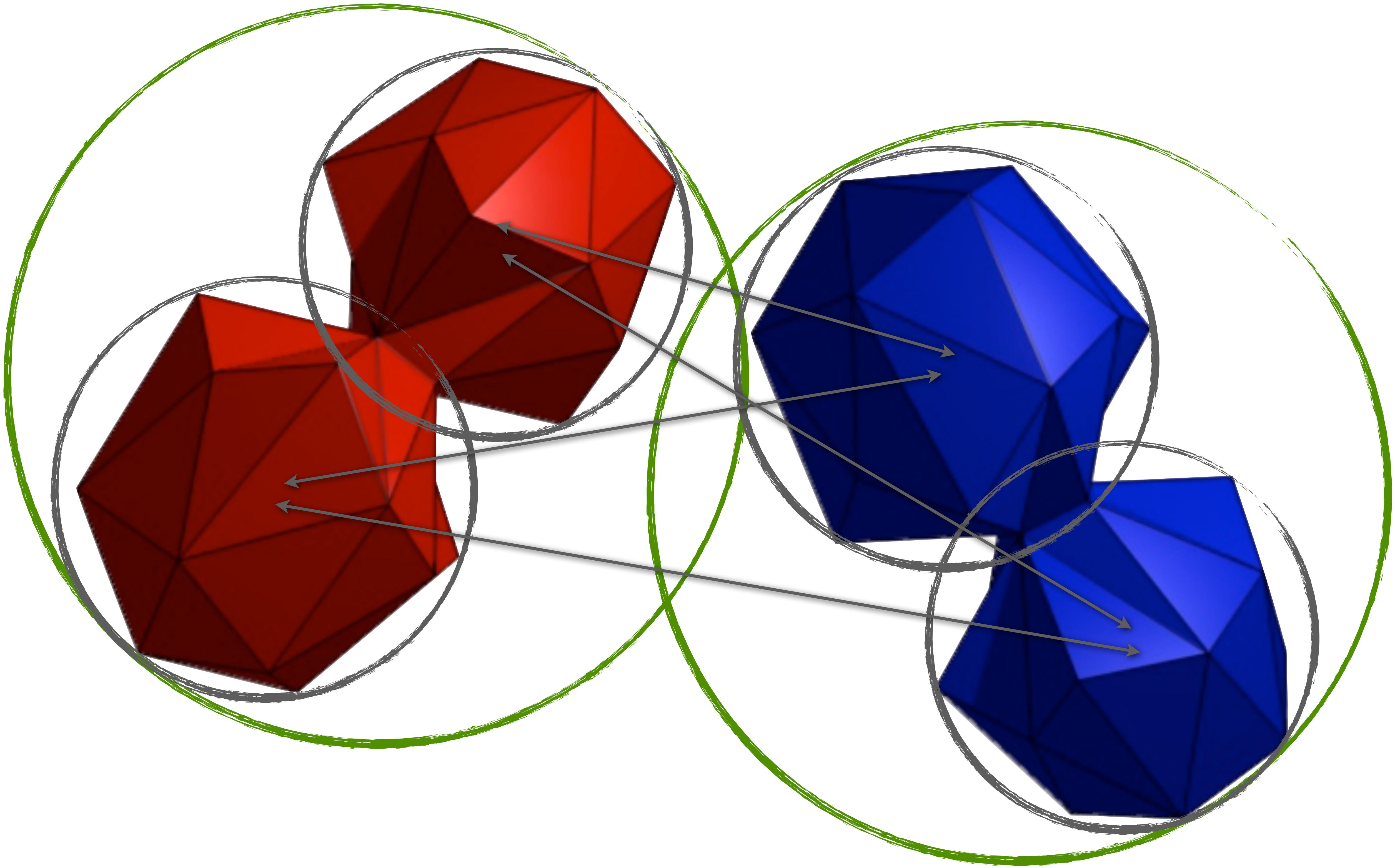
- Many flavours:
  - Bounding spheres
  - Axis-aligned box
  - Oriented box
  - Polytope / convex hull
- Allows mesh collision detection using one common algorithm

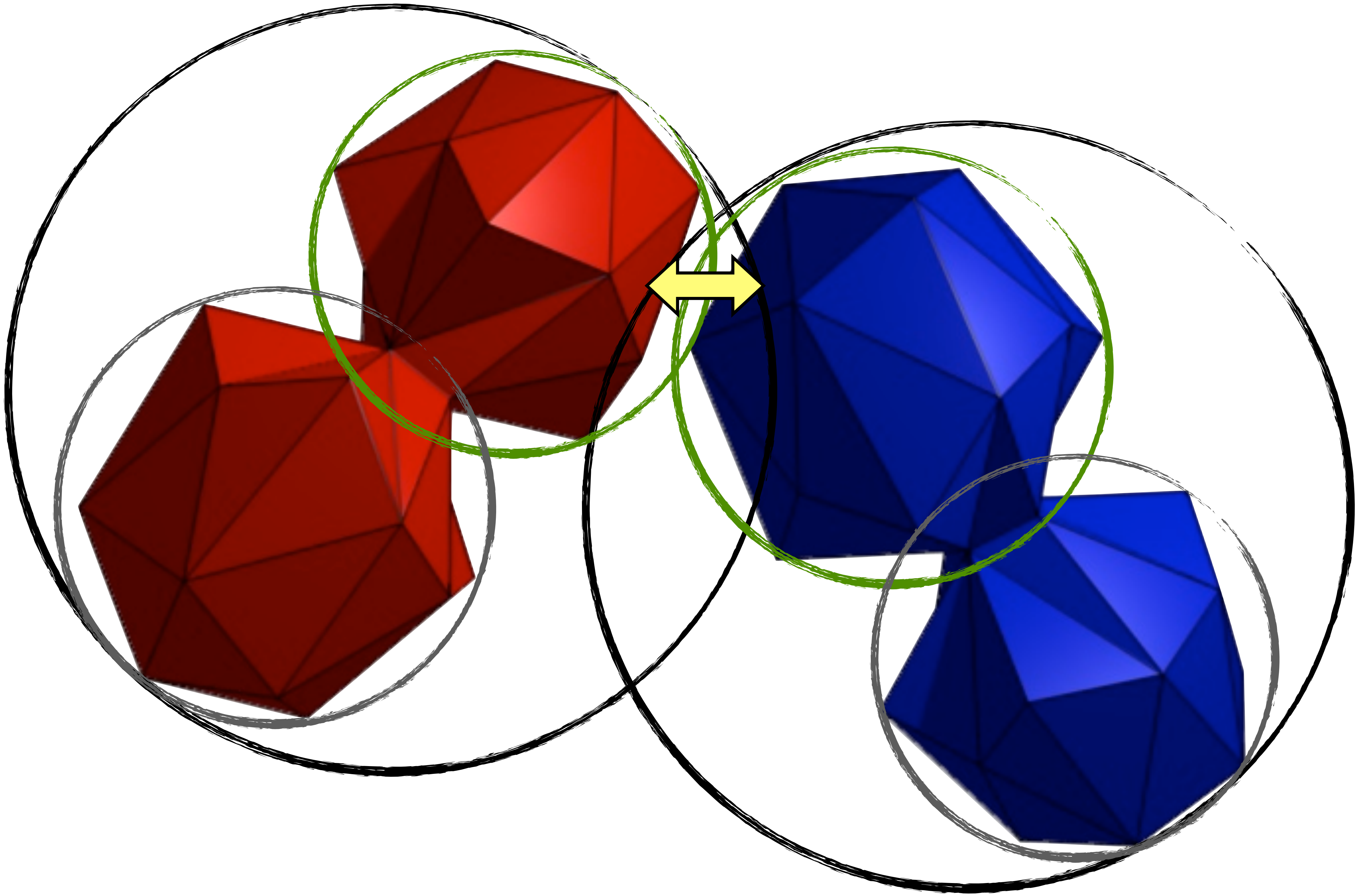


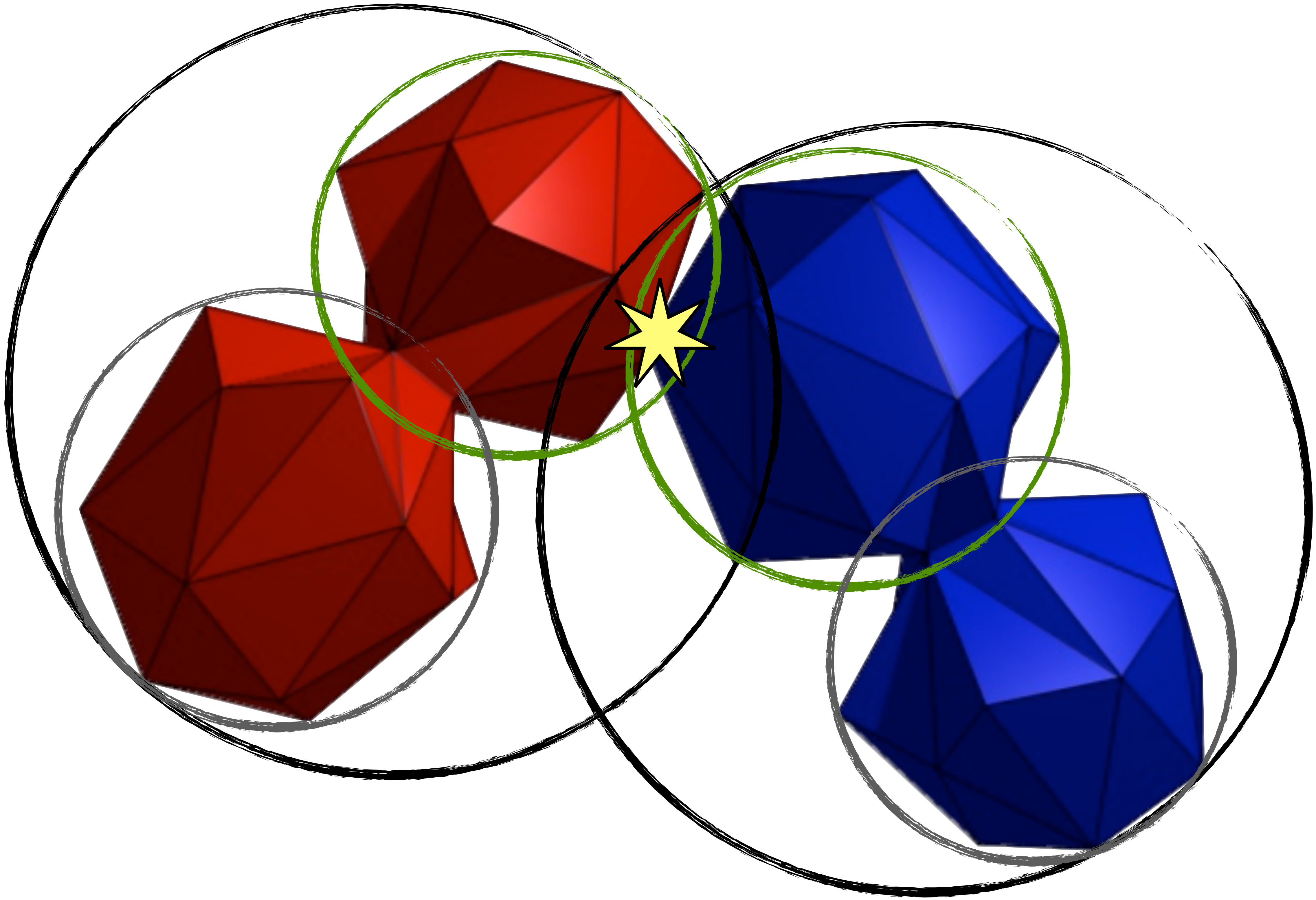
BVH Collision Queries

- Rejection test: If bounding volumes do not intersect, then the objects (or parts within) cannot intersect
- If bounding volumes intersect, recursively query all pairs of bounding volumes at the next hierarchy level in each object
- Can track and report an (approximate) minimum separation distance, or simply report interference









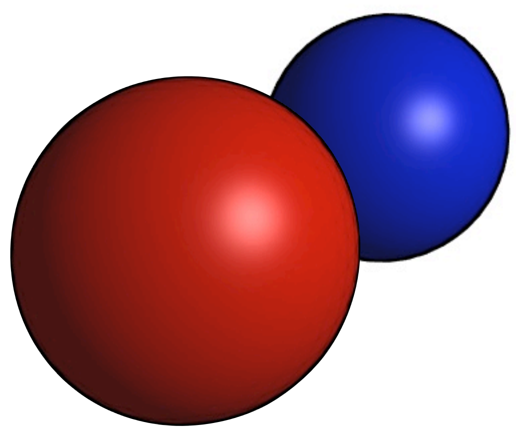
Example: Bounding Spheres

- One large sphere surrounds the mesh
- Geometry within is partitioned into two parts
- The structure is recursive: spheres enclose sub-parts
- Leaf spheres contain one triangle, a few elements, or a small convex component

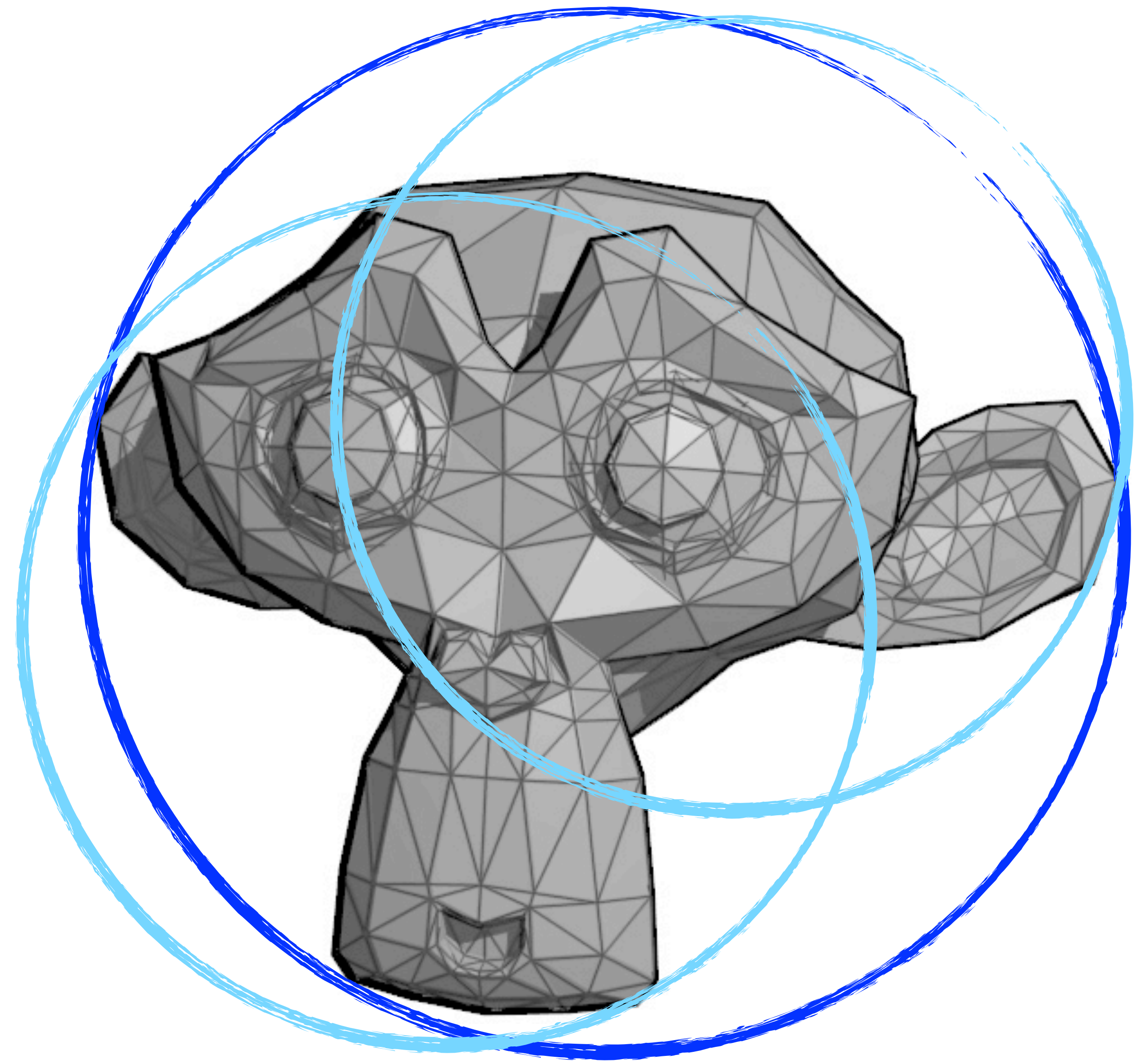
[from D. Ruspini *et al.*, *Proc. ACM SIGGRAPH*, 1997.]

Bounding Sphere Construction

- Easiest intersection test in the book, but...



- How do we determine the bounding sphere?
- How do we partition the object geometry?

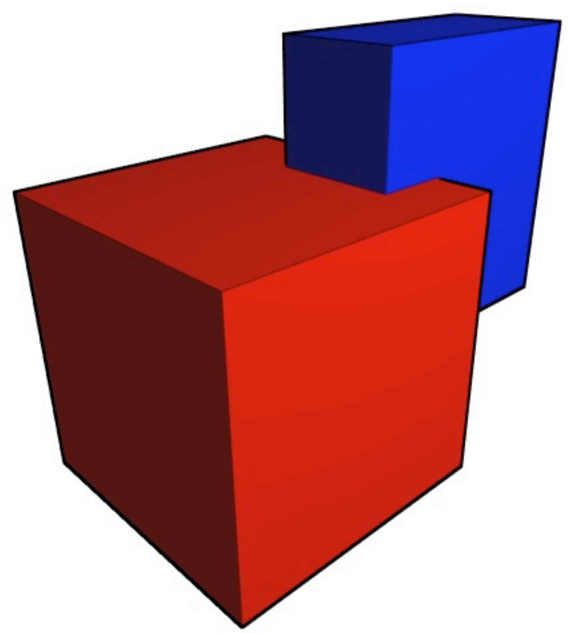


Bounding Sphere Construction

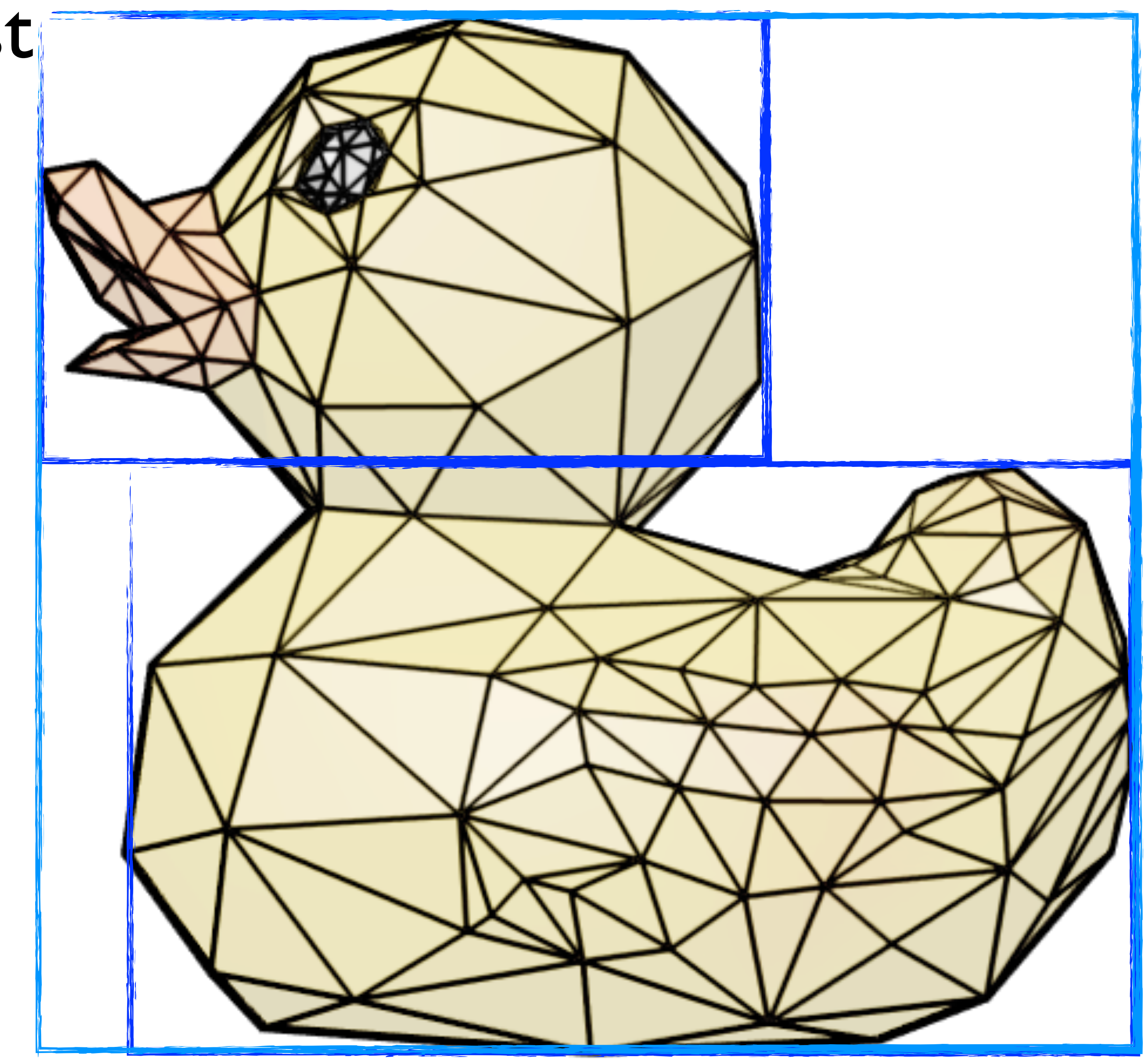
- Building the tree is expensive and often done as an offline preprocessing step
- If you have all the time in the world...
  - Try every possible partition
  - Compute the tightest bounding sphere
- In practice, heuristics are used for partitioning and a "good enough" bounding sphere is computed

Axis-Aligned Bounding Box

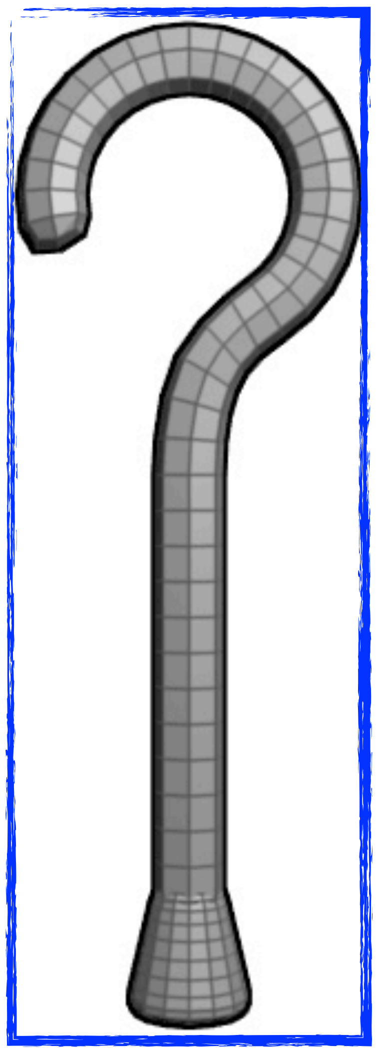
- Intersection test is just as easy as spheres...

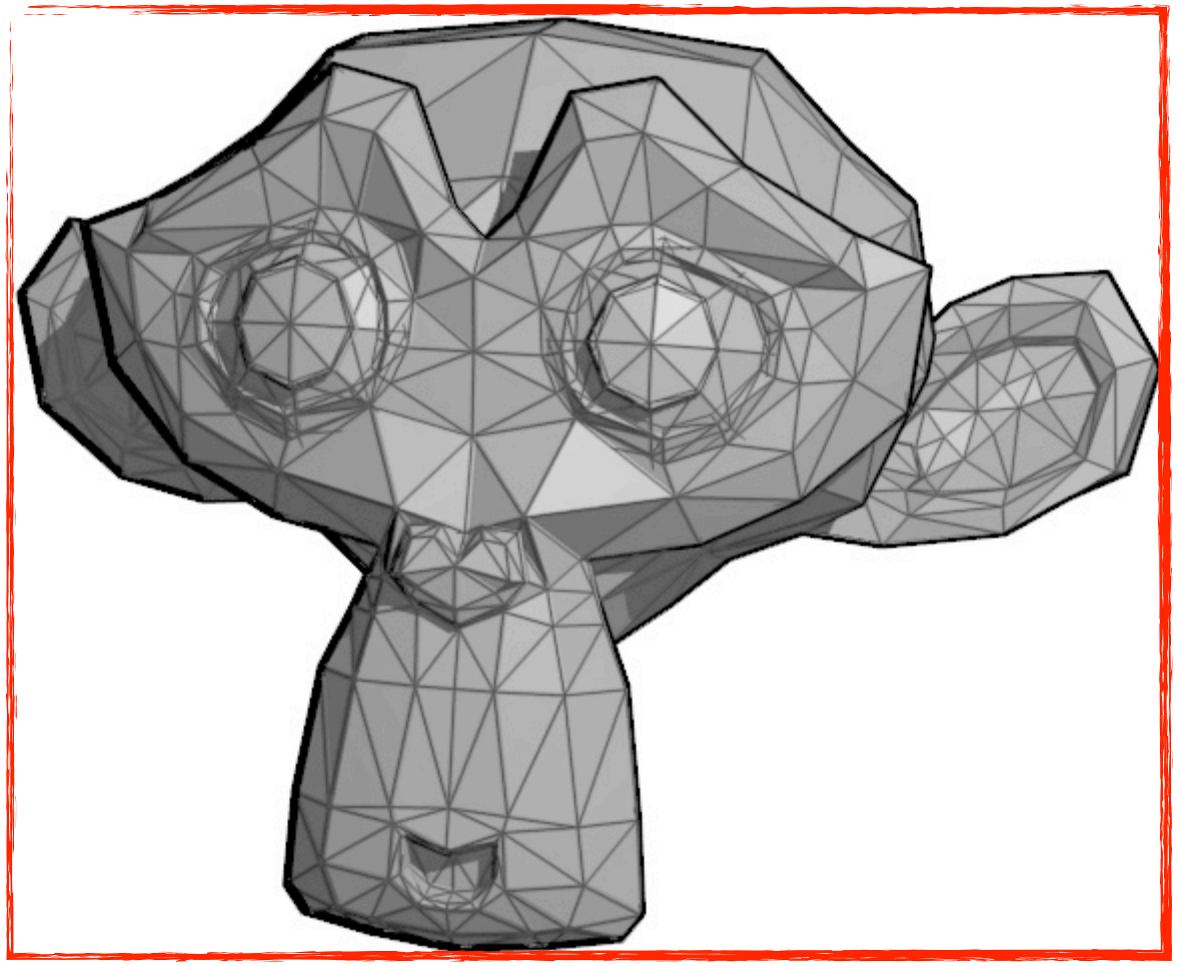


- but paritioning and bounding is much easier!



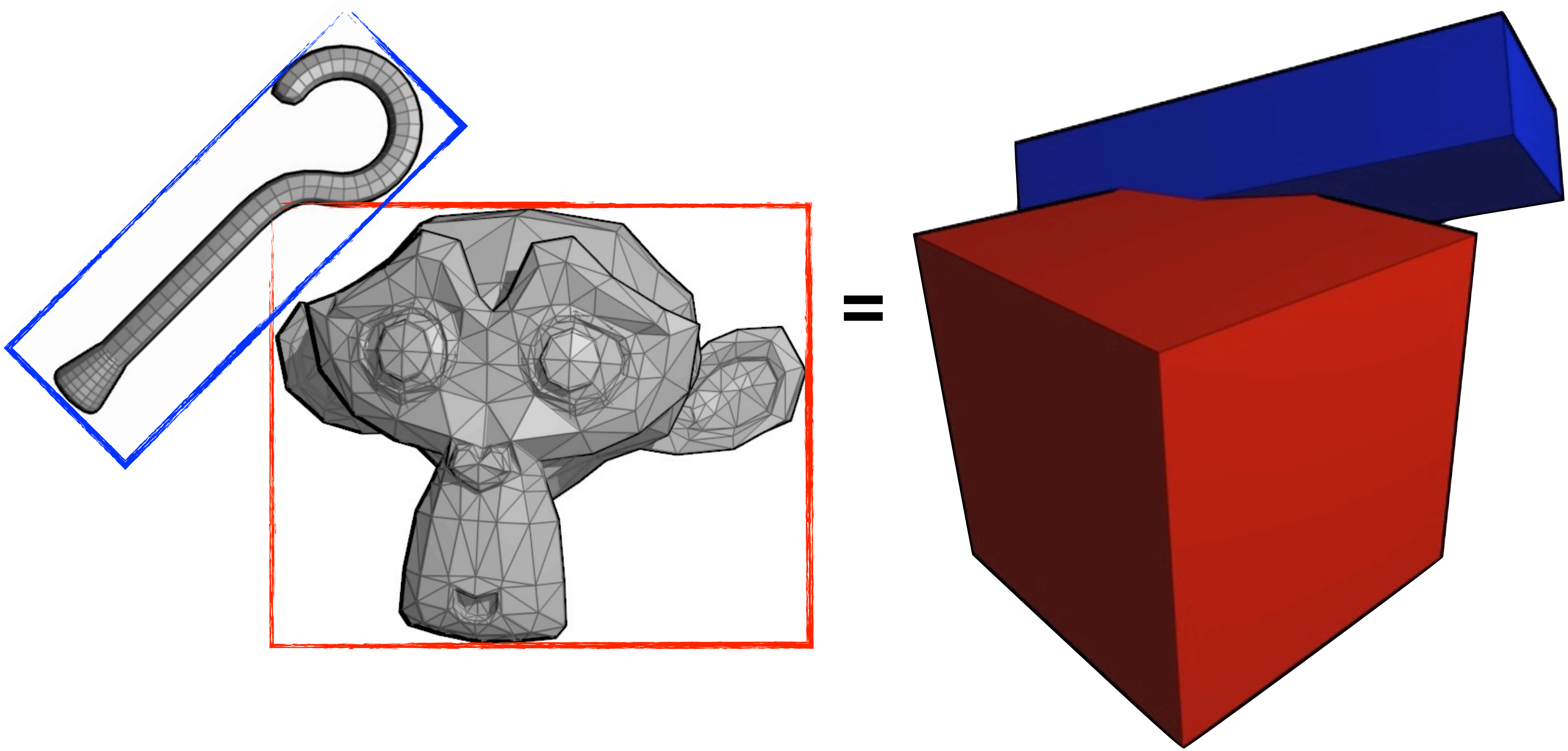
AABB Collision Detection





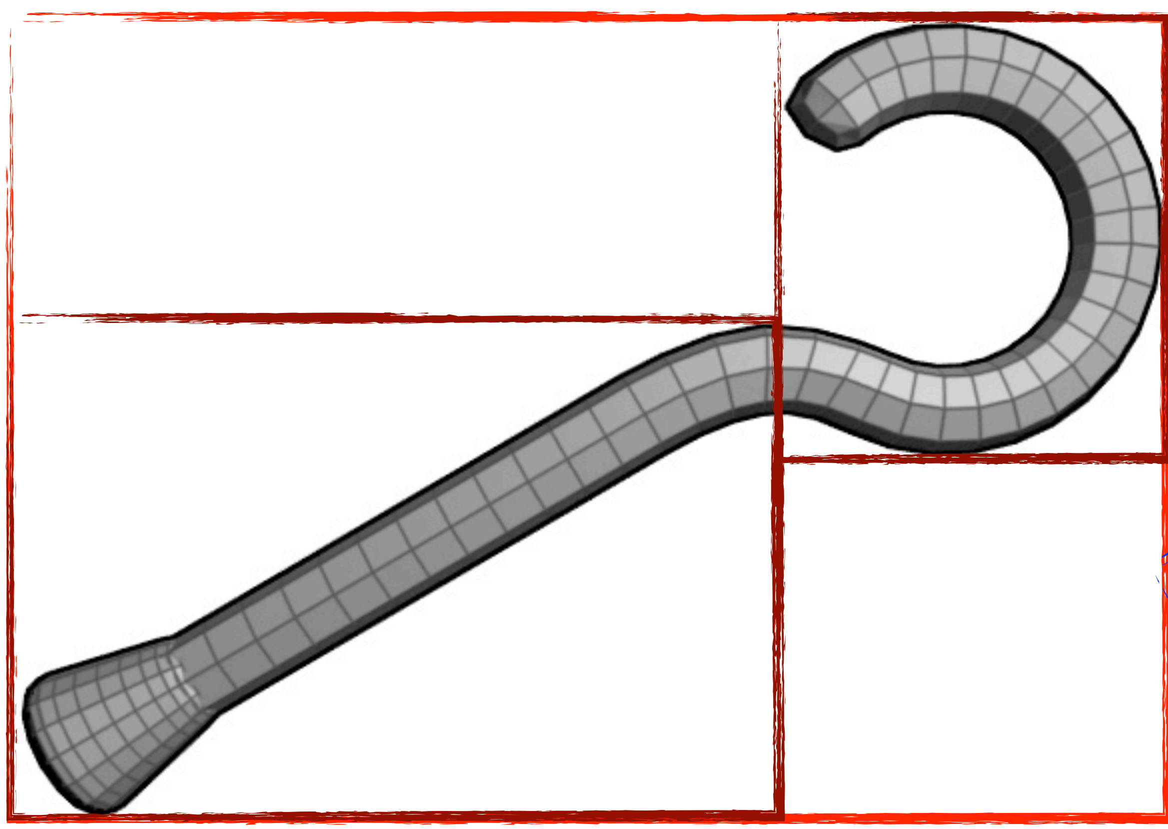
So why doesn't everyone just use axis-aligned bounding boxes?

Rotation Dependent!

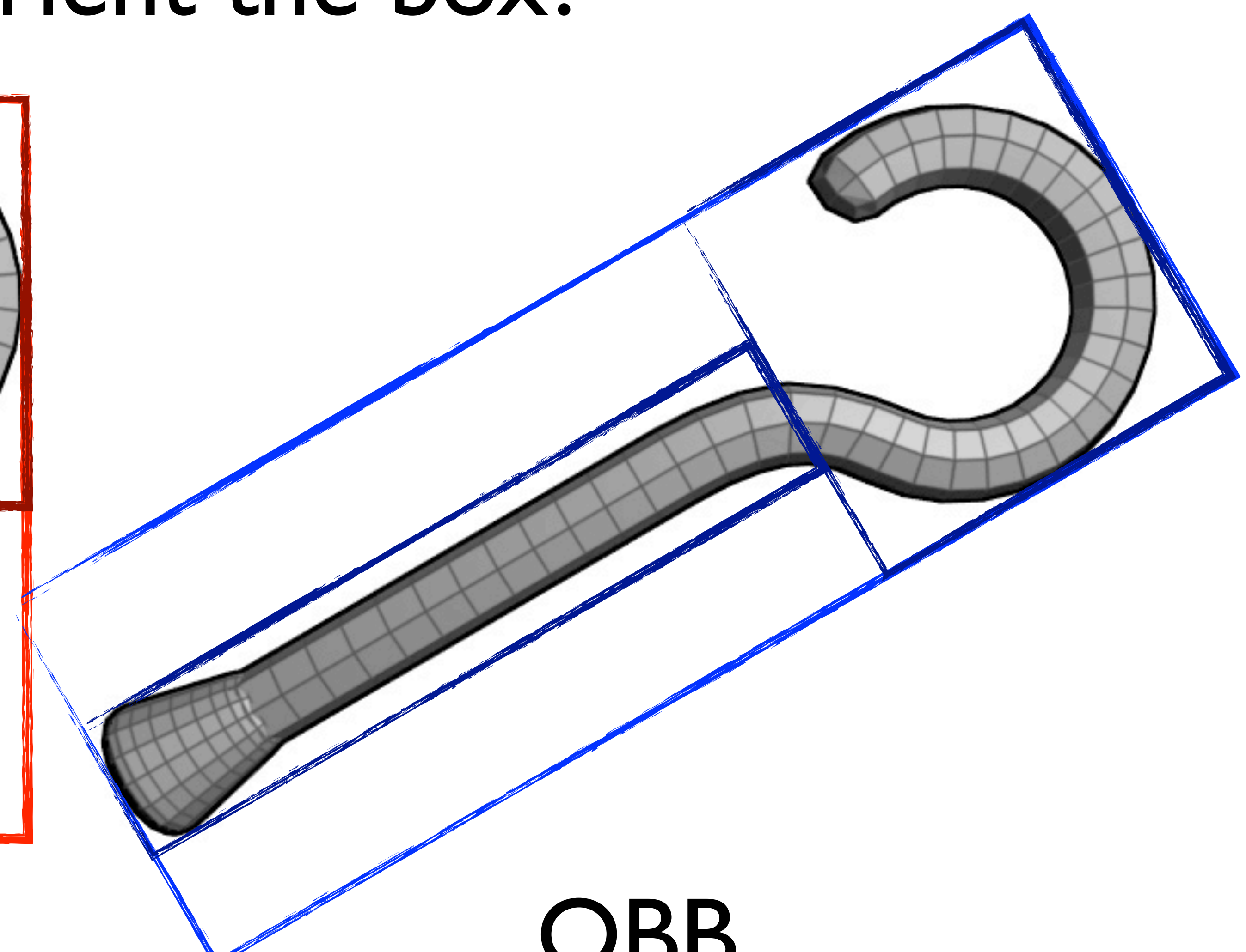


Oriented Bounding Boxes

- Tighter fit than spheres, axis-aligned boxes
- How would you orient the box?



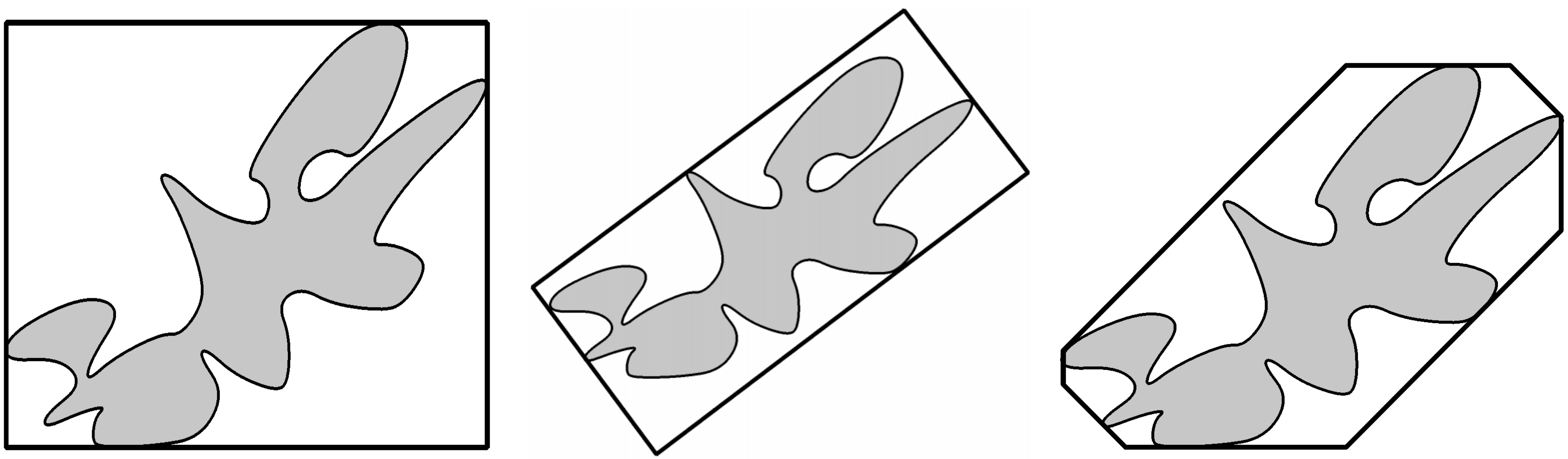
AABB



OBB

Discrete Oriented Polytopes

- An even tighter fit than oriented boxes

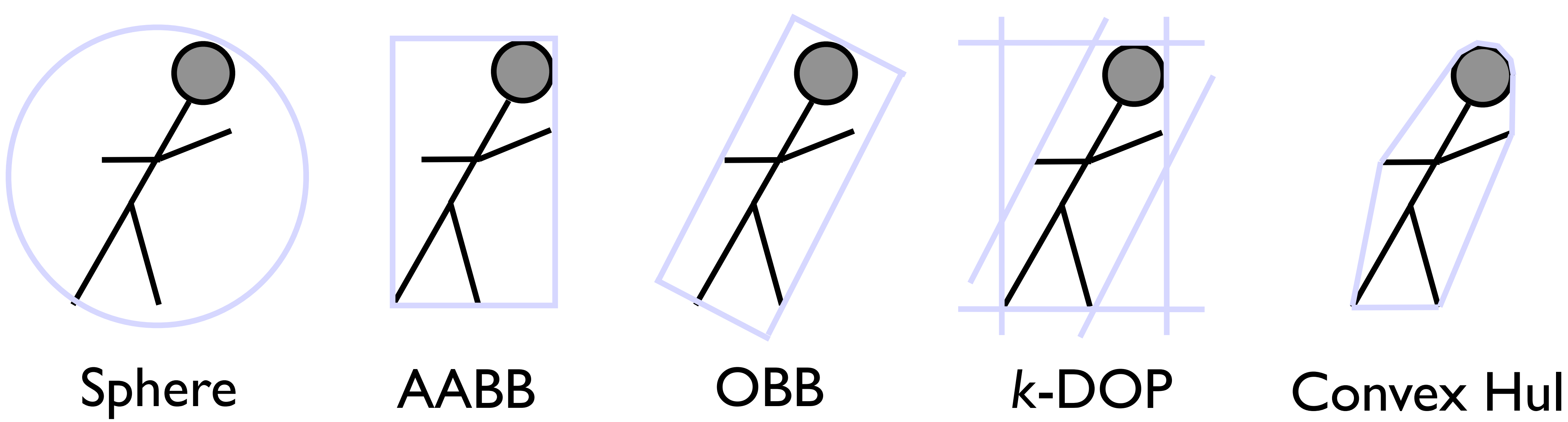


AABB OBB 8-DOP

- How would you do an intersection test?

Types of Bounding Volumes

- Many shapes (primitives) can be used as bounding volumes
- Choice of bounding volume has computational efficiency tradeoffs

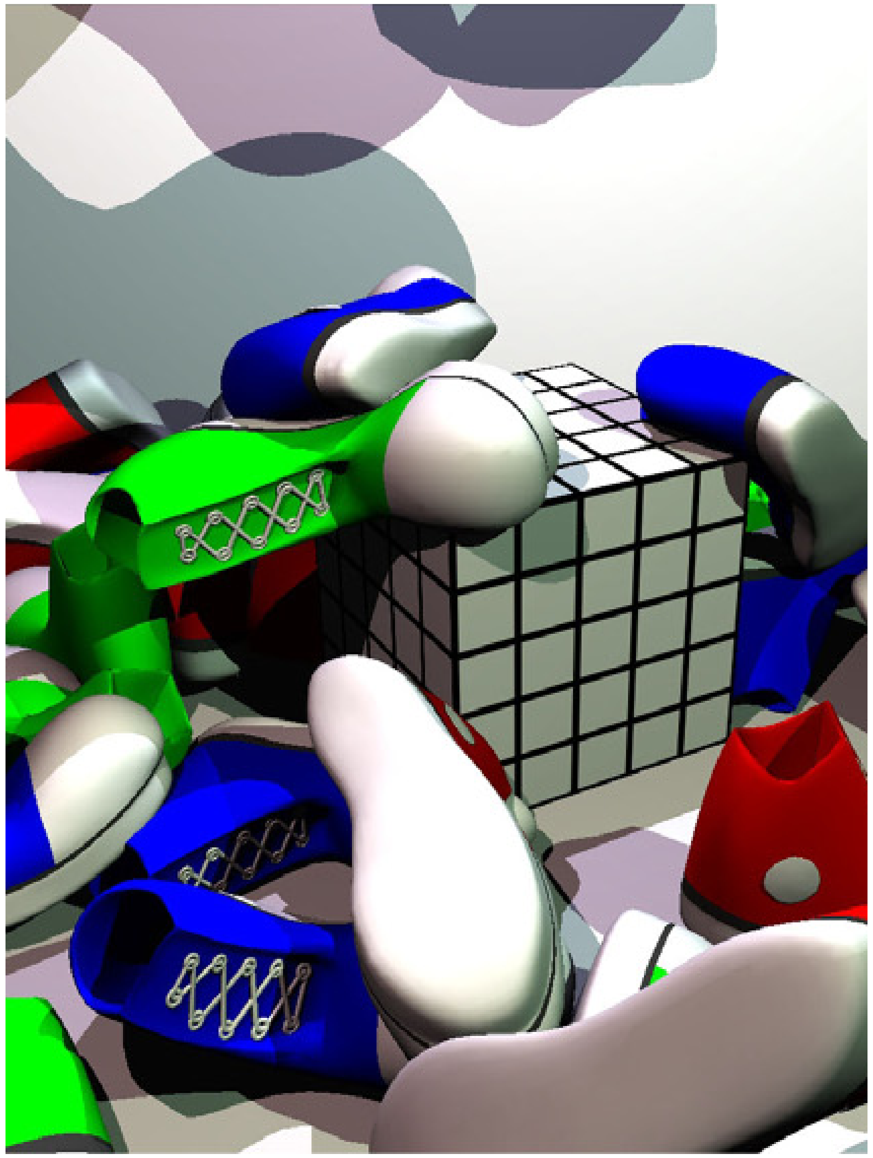


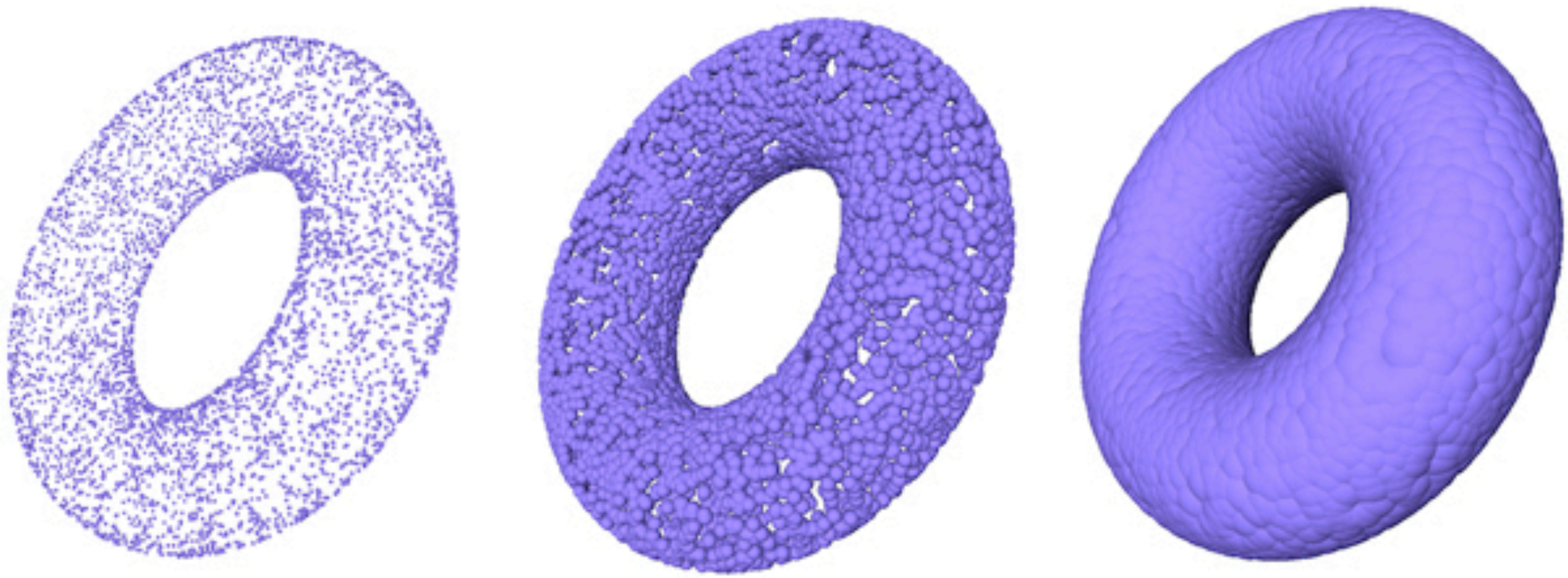
Bounding Volumes Summary

- Carefully crafted BVHs can facilitate fast mesh-mesh collision detection
- Choose the best variant for your geometry
- What is the algorithm's time complexity...
  - for typical queries?
  - in the worst case?
- What are the implications for their use in haptic rendering?

Summary

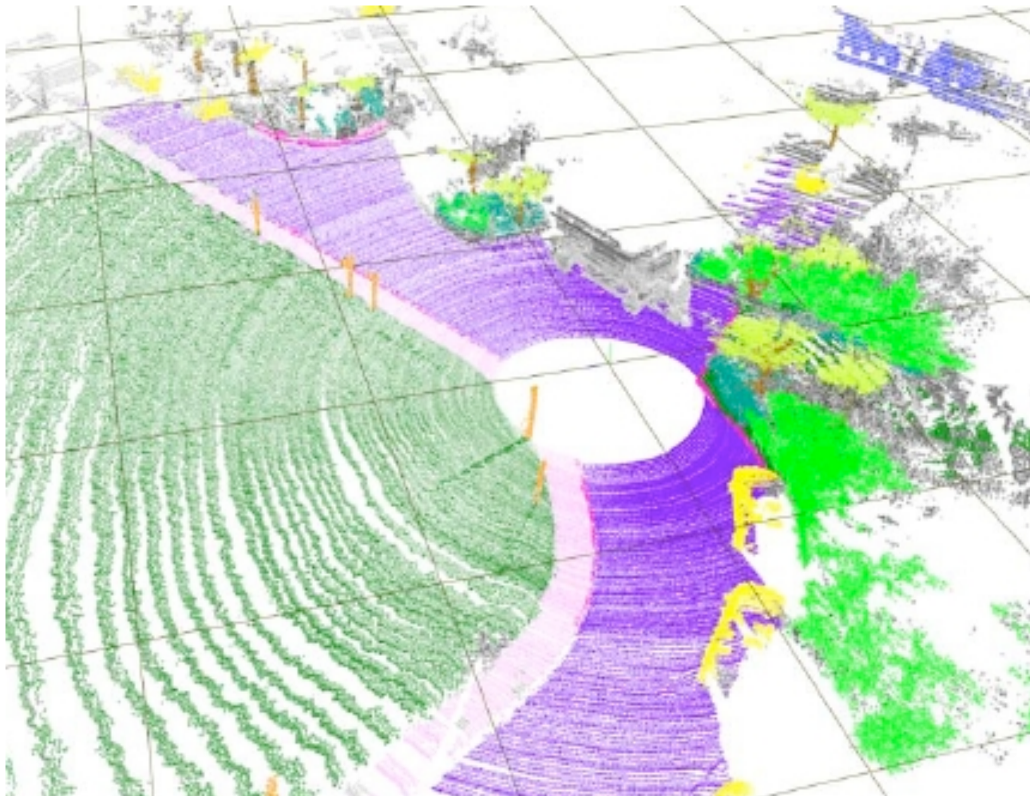
- Explored methods for mesh collision queries:
  - Spatial partitioning methods for segments
  - Bounding volume hierarchies for meshs
- Do they still work for deformable objects?

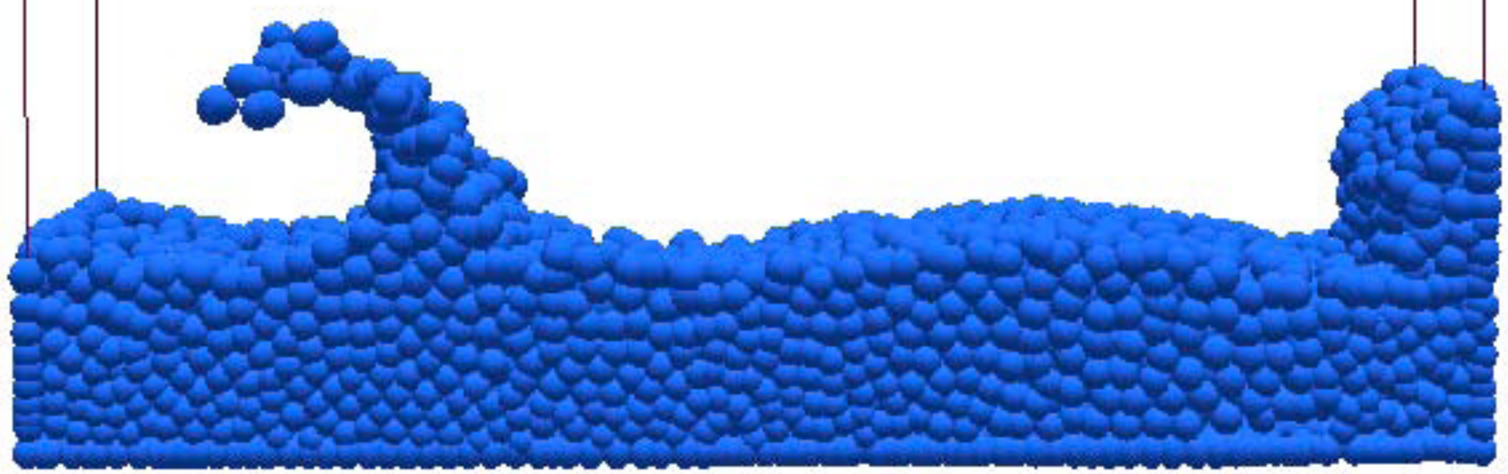




Unstructured Point Sets

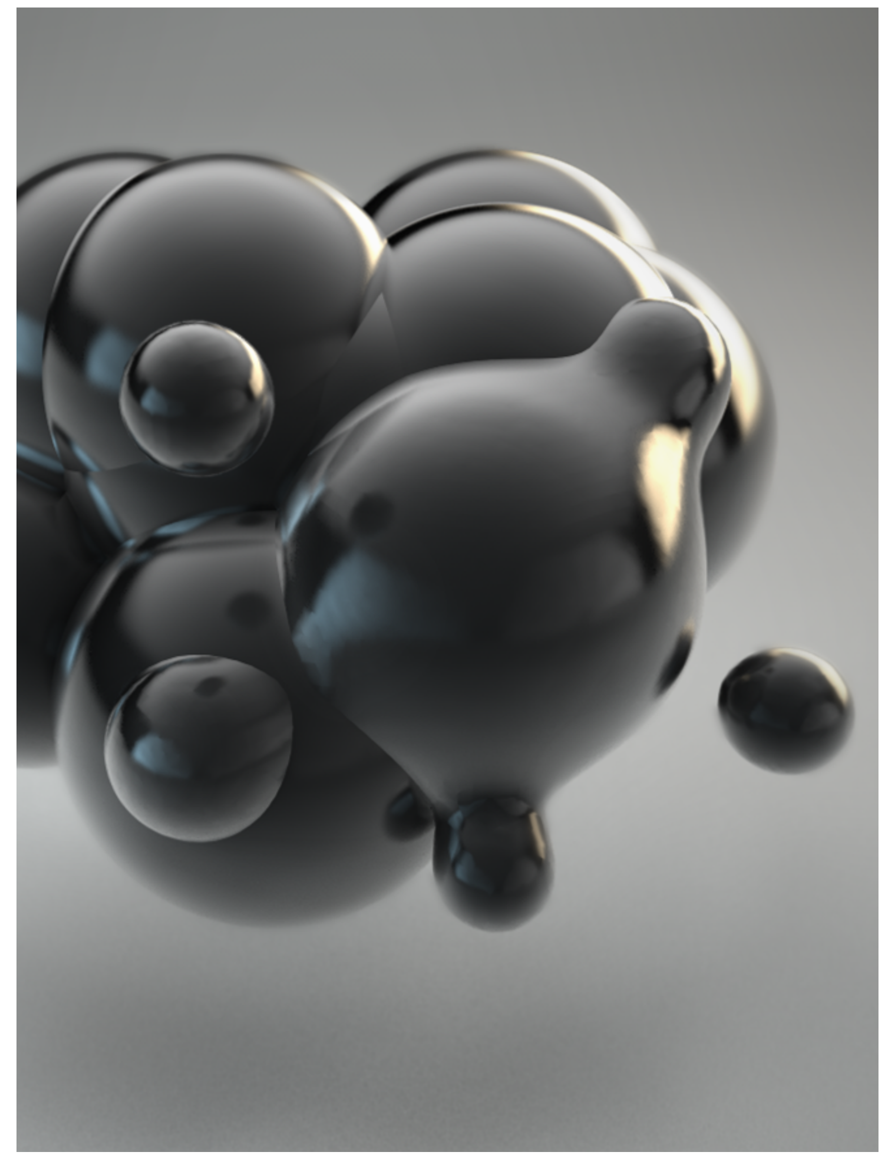
Data Sources





Metaball Surfaces

- Soft objects proposed by Wyvill et al. 1986
- Radial basis functions with compact support
- Surface is implicitly defined by a threshold on the intensity field



Compact Support Function

- Define a field contribution function, *C*(*r*), for a given distance *r* from a point
- If the point has radius of influence *R*, we desire the function to be
  - compact: $C(0) = 1 \quad \text{and} \quad C(R) = 0$
  - smooth: $C'(0) = 0 \quad \text{and} \quad C'(R) = 0$

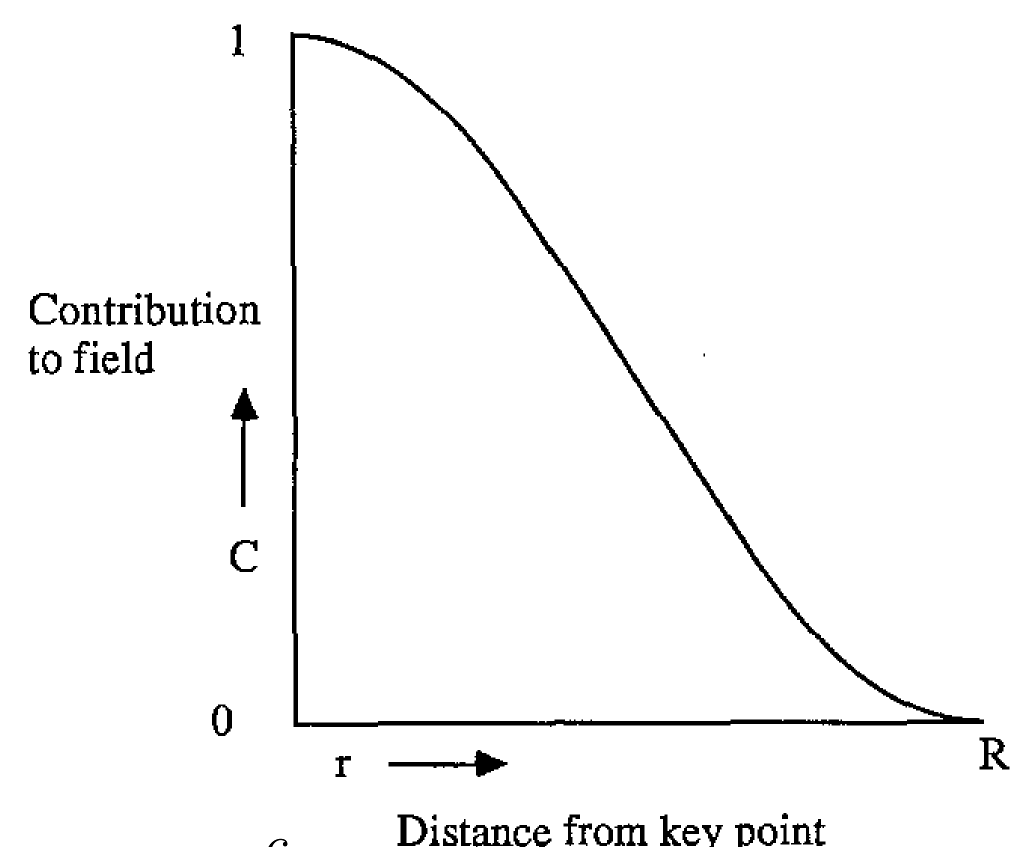
Contribution Function

- Domain is r ∈ [0, *R*]
- If we want the function to be cubic in r^2, then

$$C(r) = 1 - \frac{22}{9}r^2 + \frac{17}{9}r^4 - \frac{4}{9}r^6$$

  - with radius of influence

$$C(r, R) = 1 - \frac{22}{9}\left(\frac{r}{R}\right)^2 + \frac{17}{9}\left(\frac{r}{R}\right)^4 - \frac{4}{9}\left(\frac{r}{R}\right)^6$$



[from G. Wyvill *et al.*, *The Visual Computer*, 1986.]

Metaball Implicit Surface

- The field function is defined by

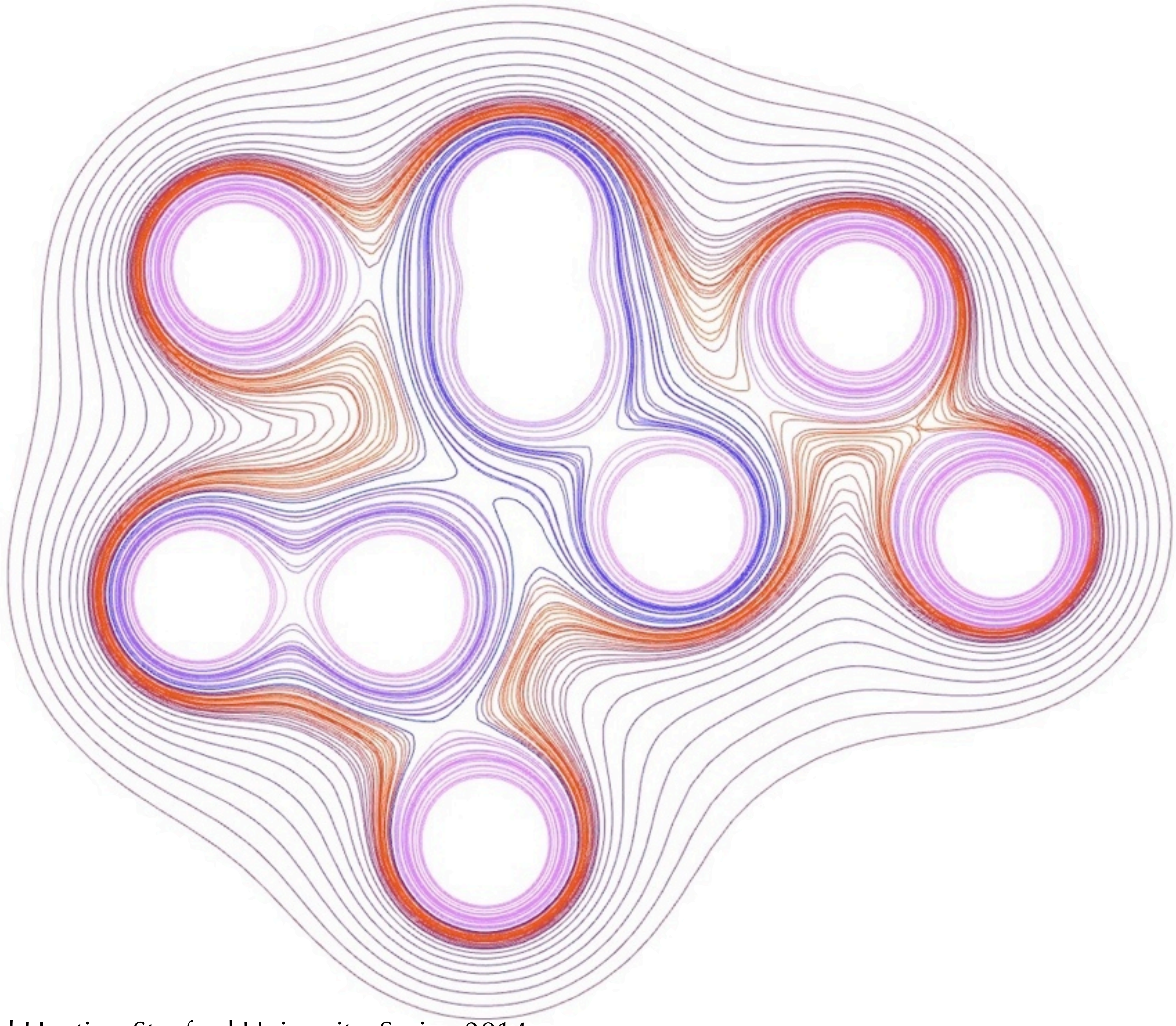
$$f(\mathbf{x}) = \sum_{i}^{n} C\left(\|\mathbf{x} - \mathbf{p}_i\|, R_i\right)$$

- And the implicit surface by

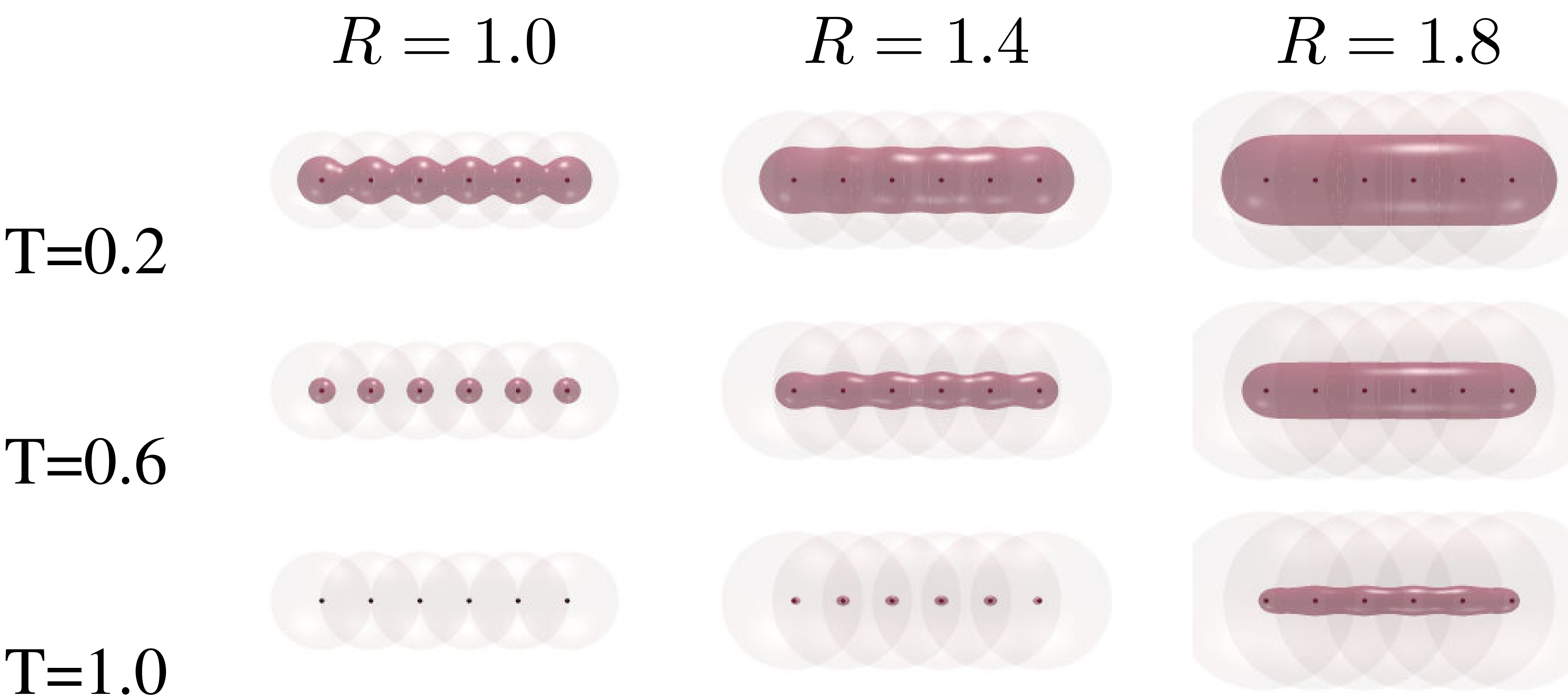
$$S(\mathbf{x}) = T - f(\mathbf{x}) = 0$$

- How many points do we need to consider to evaluate the surface function at *x*?

Surface Threshold

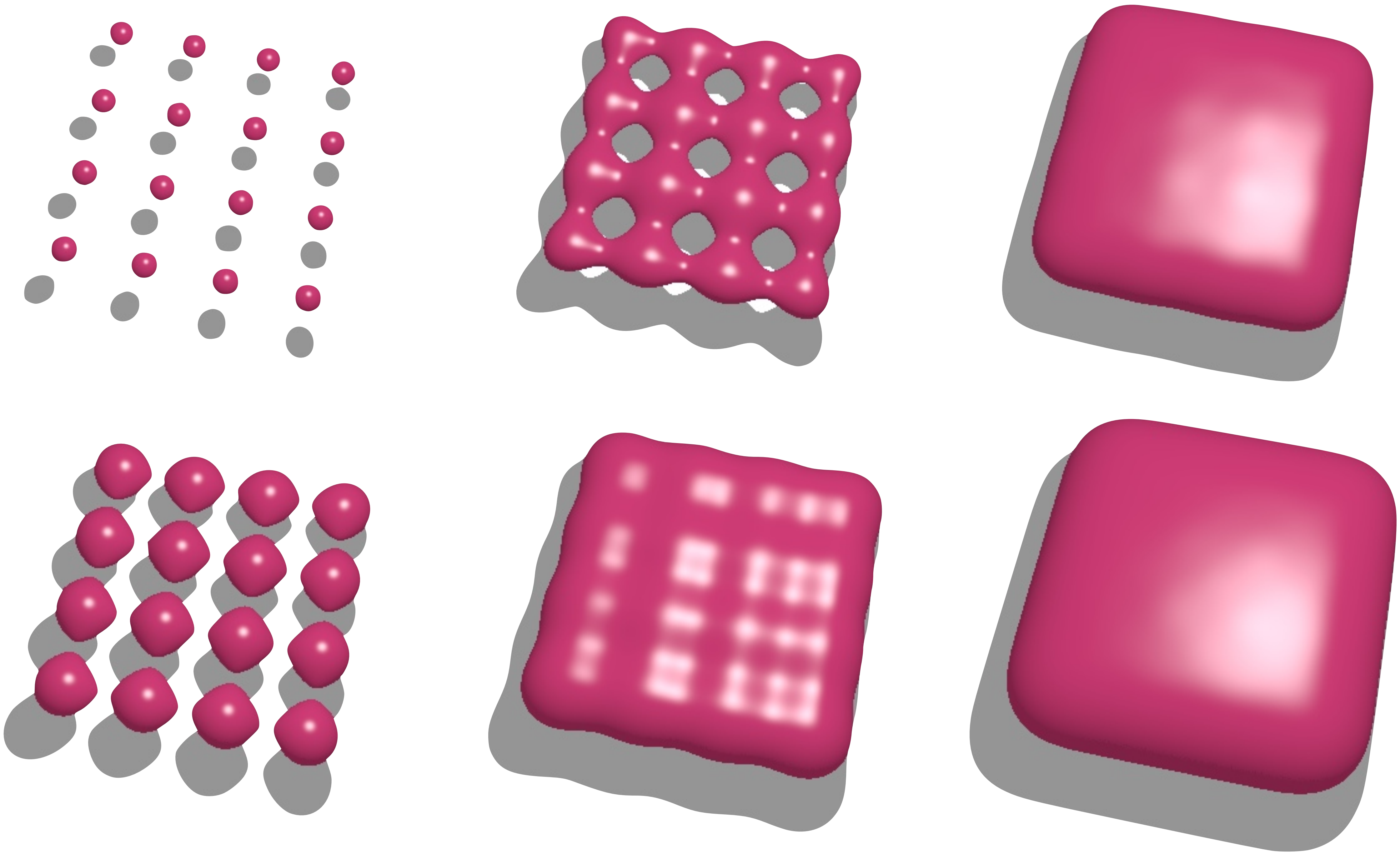


Choosing a Threshold



[from A. Leeper *et al.*, *Proc. IEEE Intl. Conf. on Robotics and Automation*, 2012.]

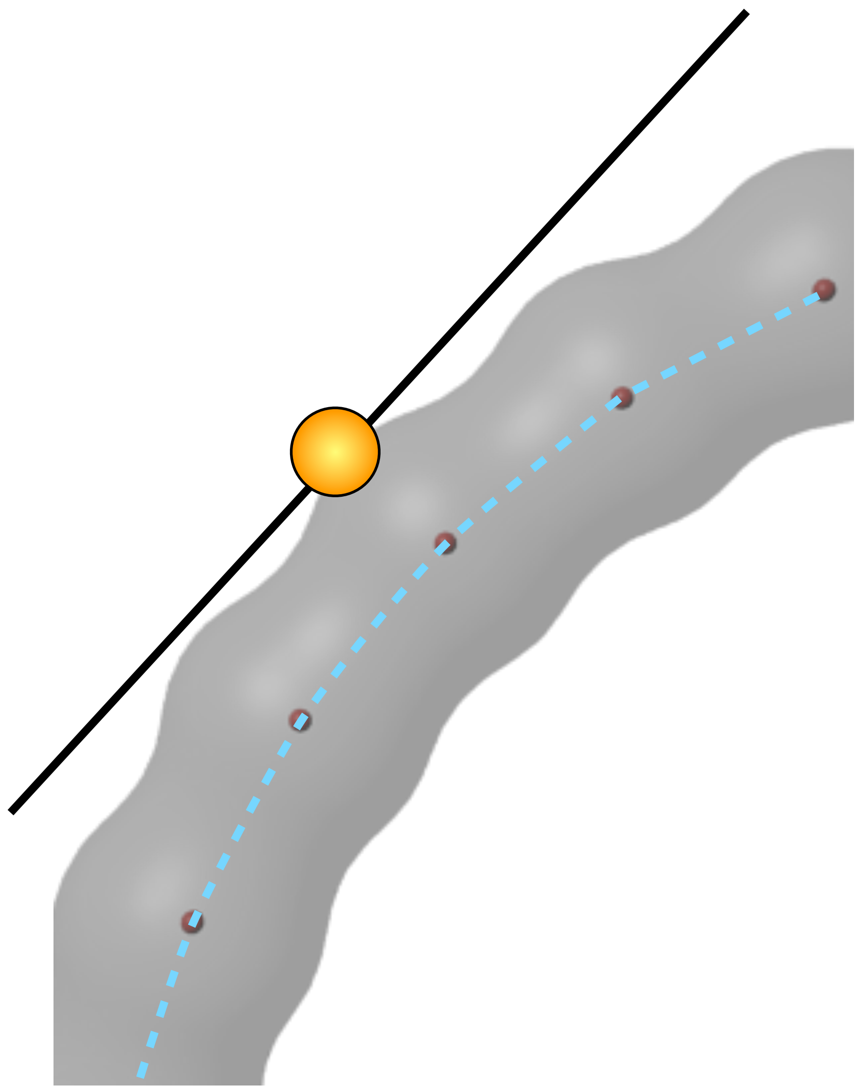
Choosing a Threshold

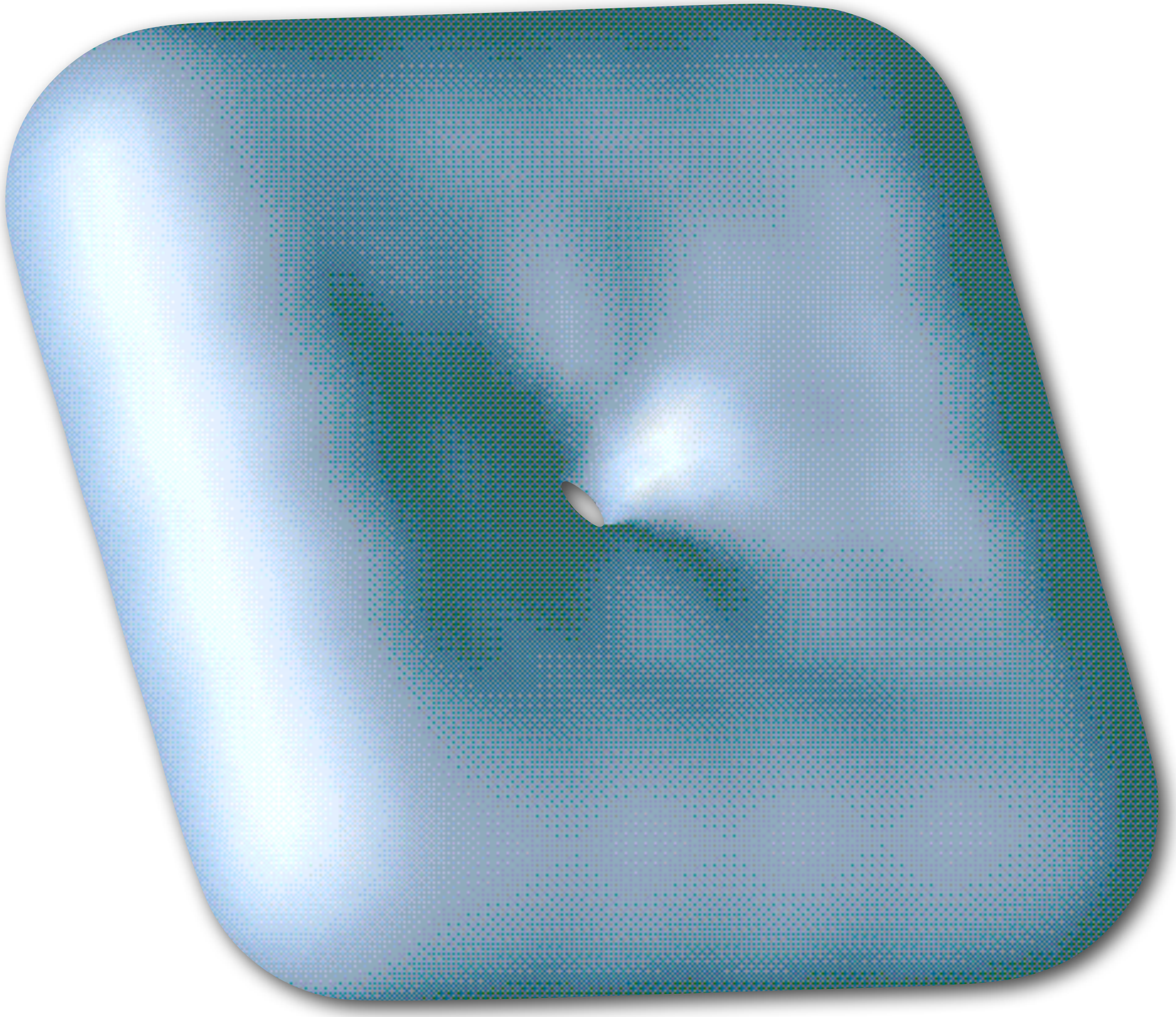


Metaball Surfaces

- Metaballs are good for
  - Results of fluid simulation
  - Noisy data
  - Sparsely sampled data
- Can you think of types of point data that a metaball surface would poorly represent?

Limitations with Metaballs





Point Set Surfaces

- Approximates a *smooth* surface from irregularly sampled points
- Create a local estimate of the surface at every point in space
- Test for intersection with the approximation

[from M. Alexa *et al.*, *IEEE Trans. on Visualization and Computer Graphics* 9(1), 2003.]

Estimating Surface Position

- Weighted average of nearby points
- If we are at position **x**, estimate a point on the surface at

$$\mathbf{a}(\mathbf{x}) = \frac{\sum_i^n \theta_i(\|\mathbf{x} - \mathbf{p}_i\|)\mathbf{p}_i}{\sum_i^n \theta_i(\|\mathbf{x} - \mathbf{p}_i\|)}$$

  - where θ is a weighting function of distance

Estimating Surface Normal

- Direction of smallest weighted covariance of nearby points
- If the weighted covariance is expressed as

$$\sigma^2_{\mathbf{n}}(\mathbf{x}) = \frac{\sum\limits_i^n \theta_i(\|\mathbf{x} - \mathbf{p}_i\|)\left(\mathbf{n} \cdot (\mathbf{x} - \mathbf{p}_i)\right)^2}{\sum\limits_i^n \theta_i(\|\mathbf{x} - \mathbf{p}_i\|)}$$

- Then the surface normal direction is

$$\min_{\mathbf{n}} \sigma_{\mathbf{n}}(\mathbf{x})$$

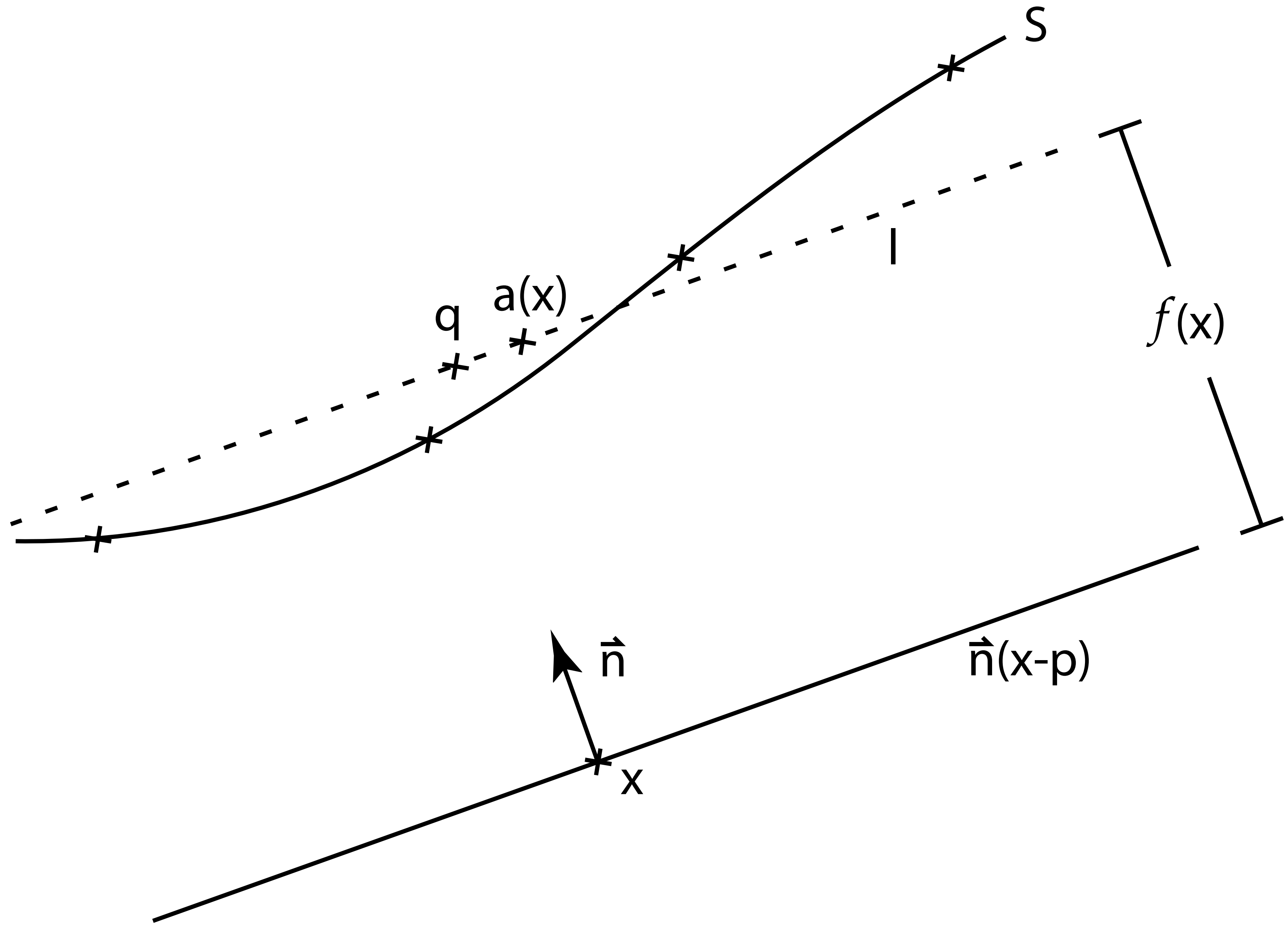
Point Set Implicit Surface

- We can now define the surface by the implicit function

$$S(\mathbf{x}) = \mathbf{n}(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{a}(\mathbf{x})) = 0$$

- This surface approximates the original shape if it was *well-sampled* with points
  - *i.e.* If the normals are well-defined within a neighborhood of the surface

Point Set Implicit Surface



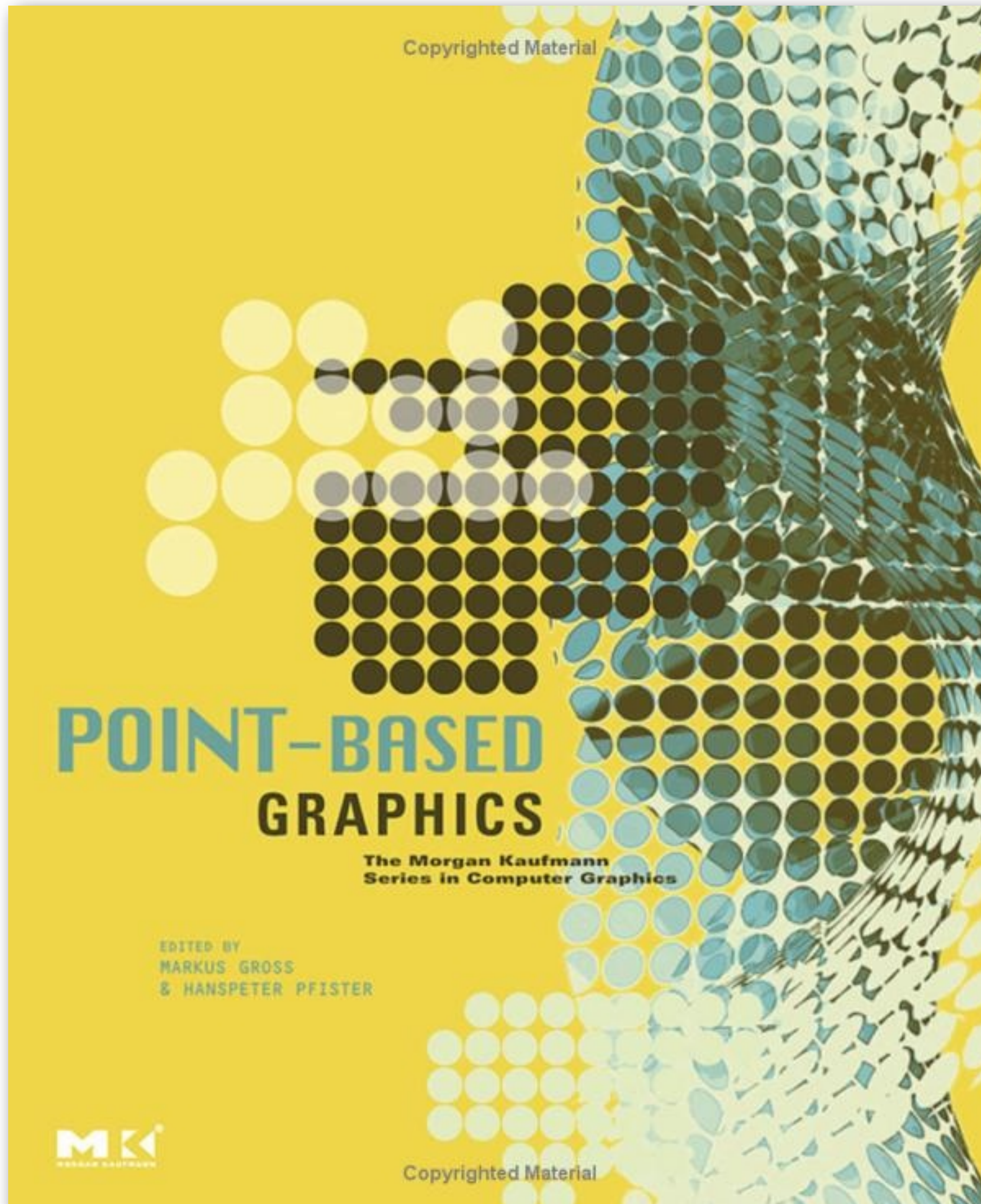
[from A. Adamson & M. Alexa, *Proc. Eurographics Symp. on Geometry Processing*, 2003.]

Choosing a Weighting Function

Cyberware Rabbit, 67038 points

[from A. Adamson & M. Alexa, *Proc. Eurographics Symp. on Geometry Processing*, 2003.]

If you want to learn more...



Summary

- Point sets can be haptically rendered as implicit surfaces
- We examined two methods of formulation:
  - Metaballs (a.k.a. blobs, soft objects)
  - Point-sampled surface reconstruction