CS709a: Algorithms and Complexity

Focus: Spatial Data Structures and Algorithms

Instructor: Dan Coming

dan.coming@dri.edu

Thursdays 4:00-6:45pm

Office hours after class

(or by appointment)

Today

- Finish Project 1 Presentations
- Project 2
- Bounding Volumes
  - Fitting
  - Intersections

Tentative Calendar

- 2/12 – Paper selection due
- 2/19 – Paper Presenter: Joe
- 2/26 – Paper Presenter: Matt
  - Present Project 1 in class (Project 1 Due 2/25)
- 3/5 – Bounding volumes
- 3/12 – Nearest neighbor
- 3/14-22 Spring Break
- 3/26 – Midterm review
  - Present Project 2 in class (Project 2 Due 3/25)
- 4/2 – Paper Presenters: Mark, Scott, Cody, and Steve
- 4/9 – Midterm
- 4/16 – Present Project 3 in class (Project 3 Due 4/15)
- 4/23 – Paper Presenter: Roger
- 4/30
- 5/7-13 Finals Week
  - Final Projects and Presentations Due

Project 2: Due 3/25 @11:59pm

- Adopt another team's Project 1 code and extend it
- Base functions to add:
  - iterator insert(const point_t & x)
  - iterator insert(iterator hint, const point_t & x)
  - void erase(iterator position)
  - size_type erase(const point_t & x) // erase all matching and report how many were erased
- Useful support functions for the above:
  - iterator find(const point_t & x)
  - size_type count(const point_t & x)

Project 2 (continued)

- Ray Cast:

  template <intersect_info>

  bool intersect(const point_t & origin, const point_t & direction, intersect_info & info)

  - Returns whether there was a hit before info.hit_time and if there was a hit, info contains the result
  - template <class T> struct intersect_info {

    T hit_time;

    point_t hit_location;

    };

Project 2 (continued)

- Nearest neighbor
  - vector<iterator> find_nearest(const point_t & x, size_type count)
- Extra credit (5%) if these are done in a better way than a for loop of calls to insert/erase:
  - template <class InputIterator> void insert(InputIterator first, InputIterator last)
  - void erase (iterator first, iterator last)

Project 2 (continued)

- Data
  - Assume point_t is assignable and comparable and has:
    - template <class intersect_info>
      ```
      bool point_t::intersect(const point_t & ray_origin,
                              const point_t & ray_direction,
                              T ray_thickness,
                              intersect_info & info)
      ```
  - Extra credit (10%) if your data structure can handle non-point data - black box like point_t, plus:
    - T* box::get_min_bound(); T* box::get_max_bound();
    - void split(T * plane_normal, T plane_offset,
      box & left, box & right)

Project 2 (continued)

- Add unit tests and don't break existing tests
- As before:
  - Documentation in code and a separate document providing design, implementation decisions, complexity analysis, and anything that will help the next group add collision detection
  - Presentation in class (20 minutes, plus time for questions)
- Additional details TBA as necessary – ask questions early if instructions are unclear

Project 2 (continued)

- Thinking ahead for project 3
  - Project 3 will use non-point data
  - Project 3 will track pair-wise intersections between data (collision detection)

Bounding Volumes (BV)

- Objects are likely to be non-convex
- Convex is easier

→ Convex decomposition

- Bounding Volumes
  - Convex shapes
  - Simple operations
  - Completely contain arbitrary geometry

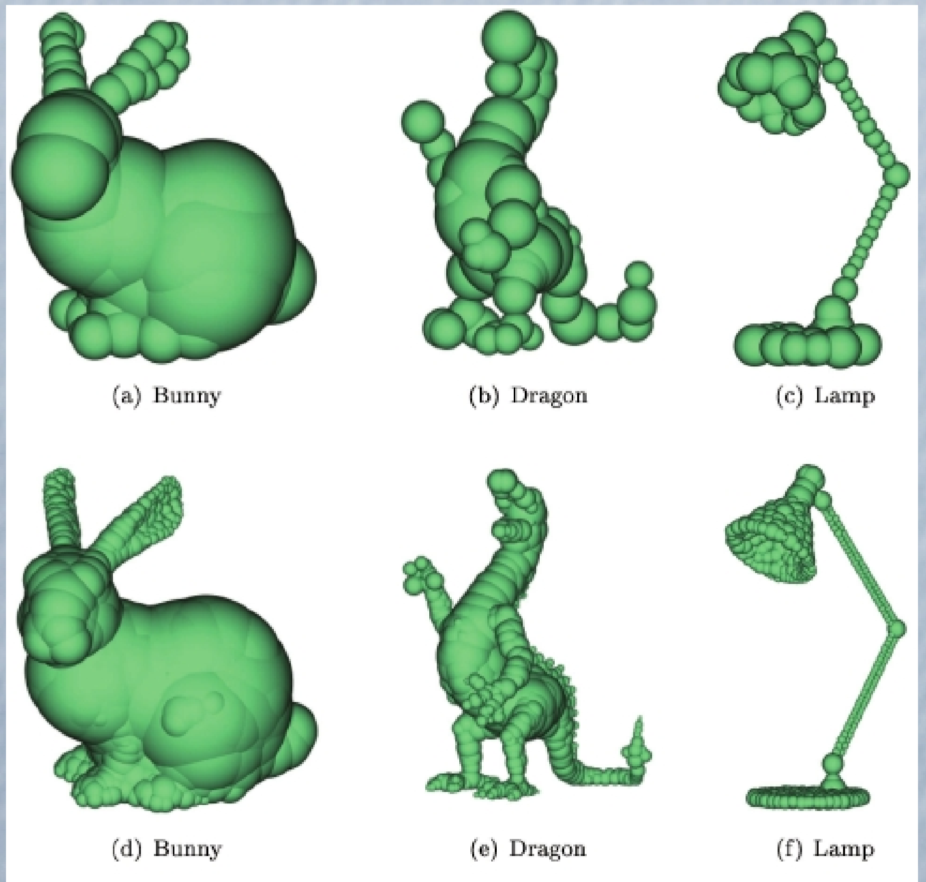
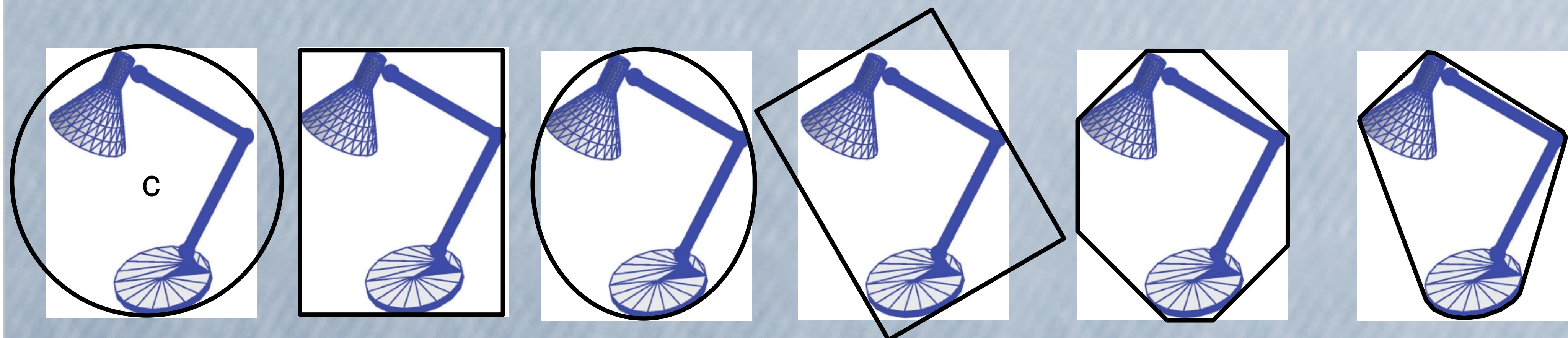


Fig. 24. Sample approximations for levels 2 and 3 of the test models.

Image: Bradshaw, G. and O'Sullivan, C. 2004. Adaptive medial-axis approximation for sphere-tree construction. ACM Trans. Graph. 23, 1 (Jan. 2004), 1-26.

Complexity – Fitness Tradeoff



Circle/Sphere | Axis-Aligned Bounding Box(AABB) | Ellips{e|oid} | Oriented Box | Discrete Oriented Polytope (DOP) | Convex Hull

Simple — Complex

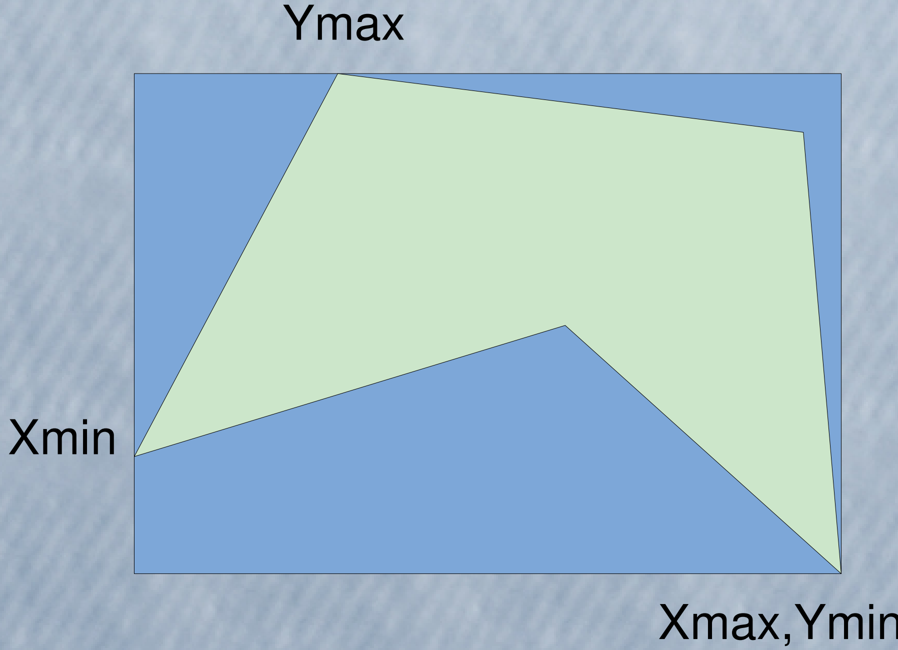
Loose fitting — Tight fitting

Fitting Bounding Volumes

- Finding a valid bounding volume is easy
- Minimizing its area/volume can be hard
- Approximate bounding volumes
  - Leave wiggle room for moving / deforming objects
  - Save build time in BVH

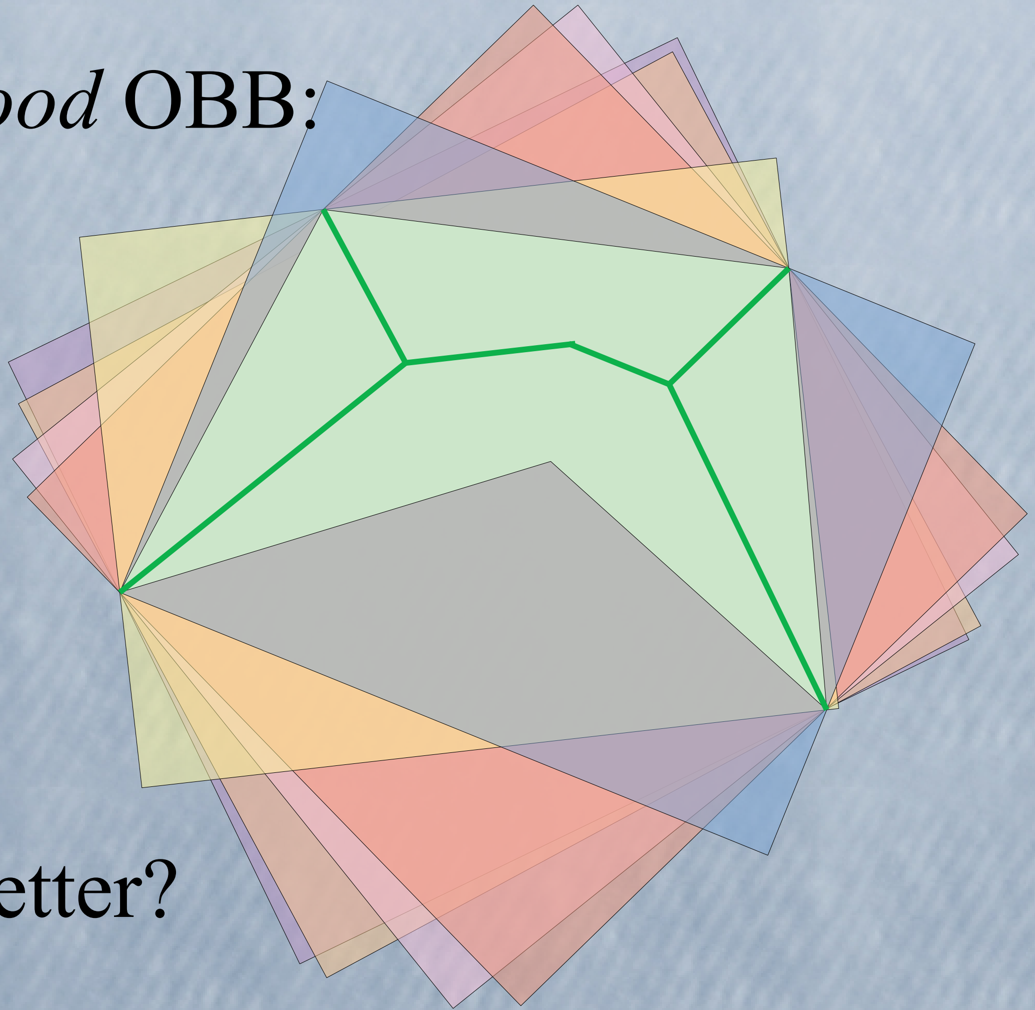
Axis-Aligned Bounding Box (AABB)

- Fitting minimum AABB:
  - Find min/max coordinate in each dimension (x_0, x_1, ...) (e.g., by looping over points)
  - Make a box from:
    (x_{min0}, x_{min1}, ...) to
    (x_{max0}, x_{max1}, ...)



Oriented Bounding Box (OBB)

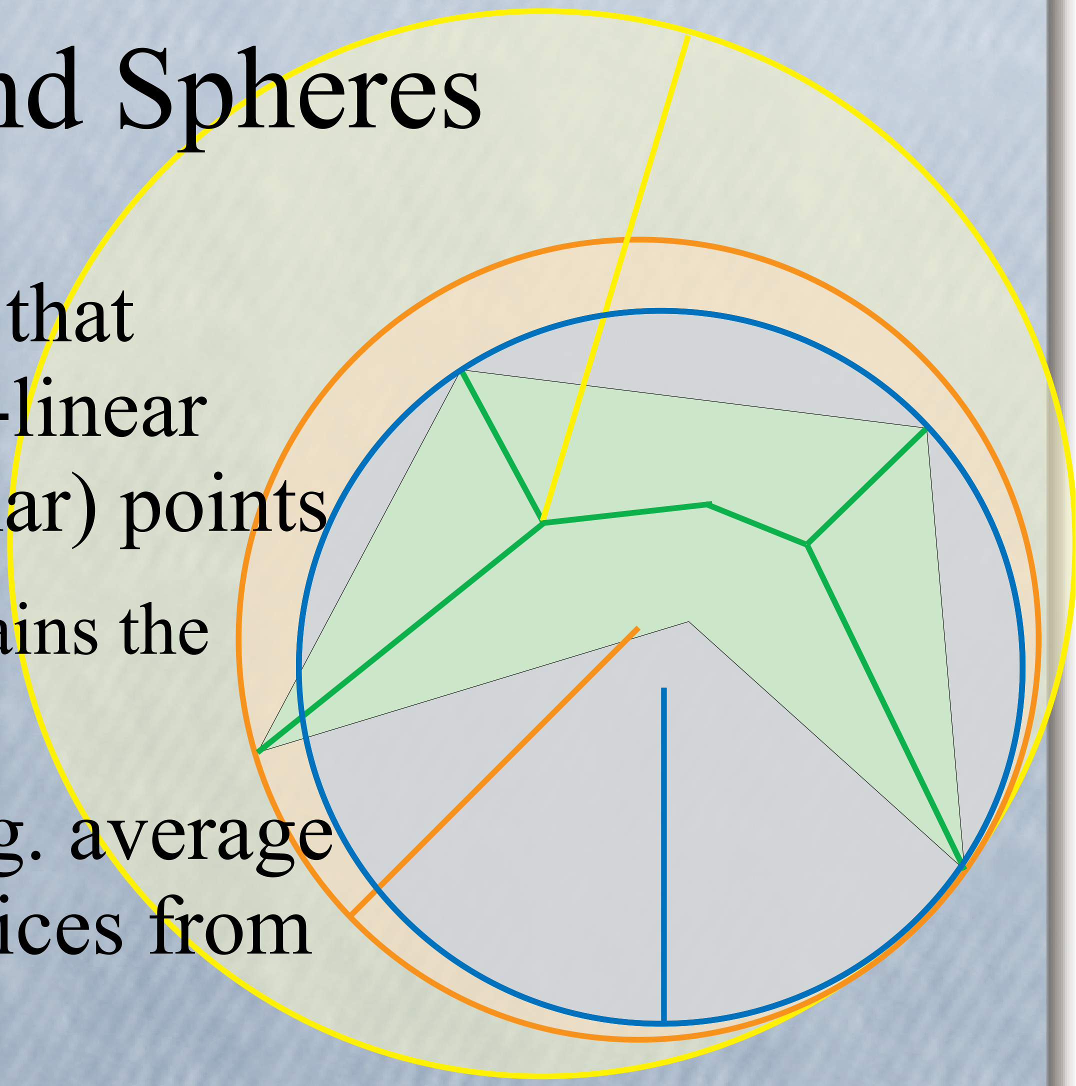
- So many possible orientations
- One way to fitting a *good* OBB:
  - Find medial axes
  - Consider each as a possible major axis for OBB



- Could we have done better?

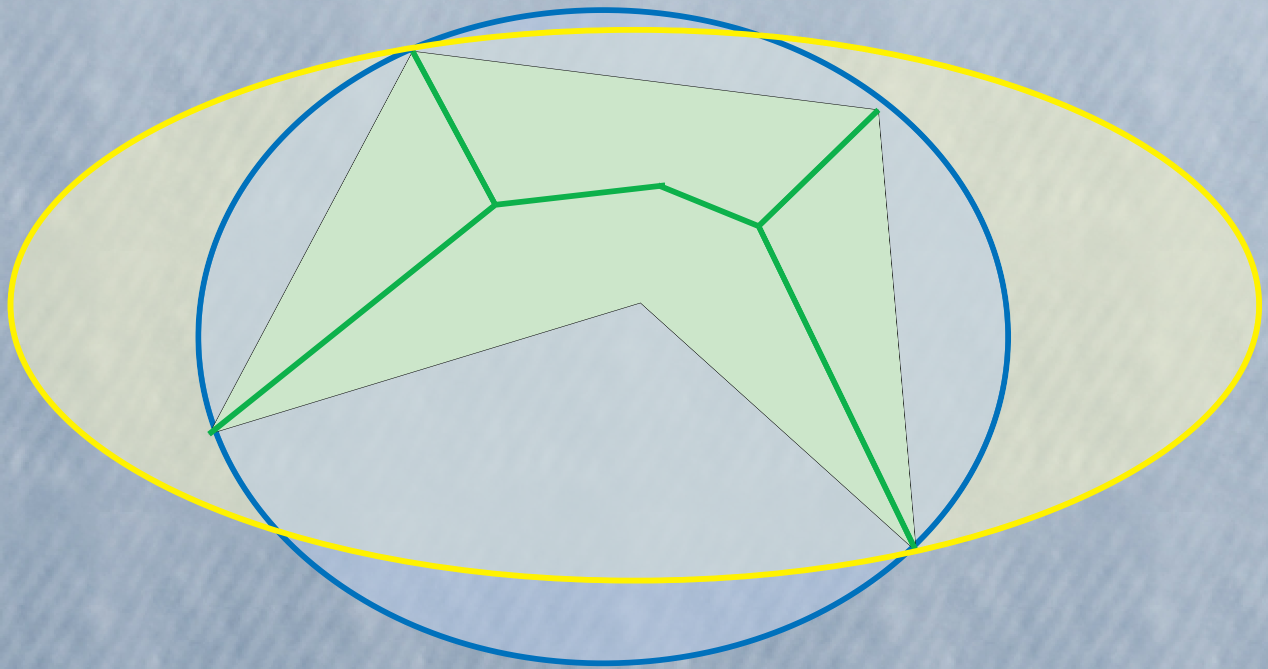
Circles and Spheres

- Alt-1: Consider circles that intersect any three non-linear (spheres: four non-planar) points
  - Must check that it contains the whole object
- Alt-2: Pick a center (e.g. average point or one of the vertices from topological skeleton)
  - Farthest point defines radius



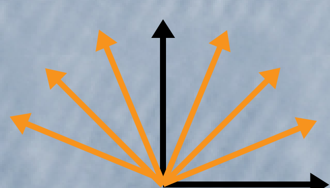
Ellips{e|oid}s

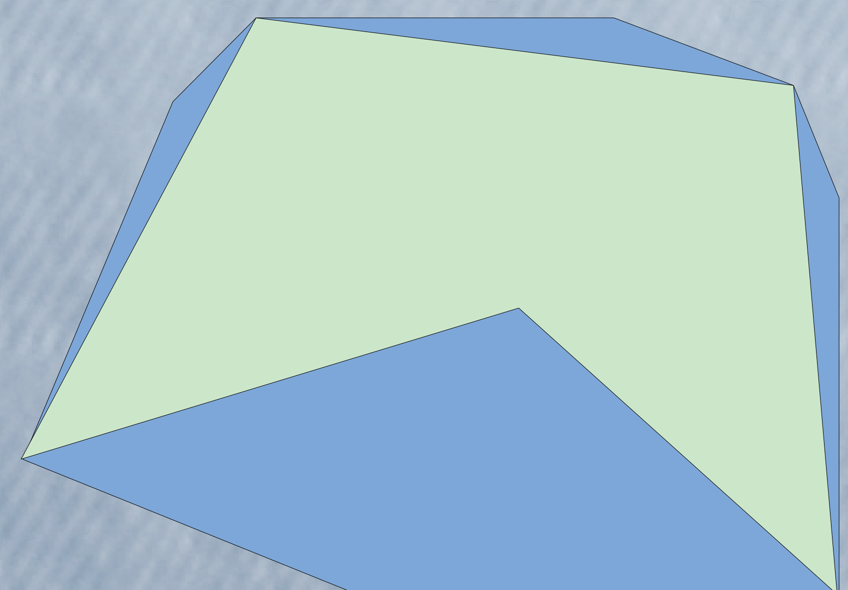
- Similar to circle/sphere – picking position
- Also have to pick an orientation and length of major/minor axes



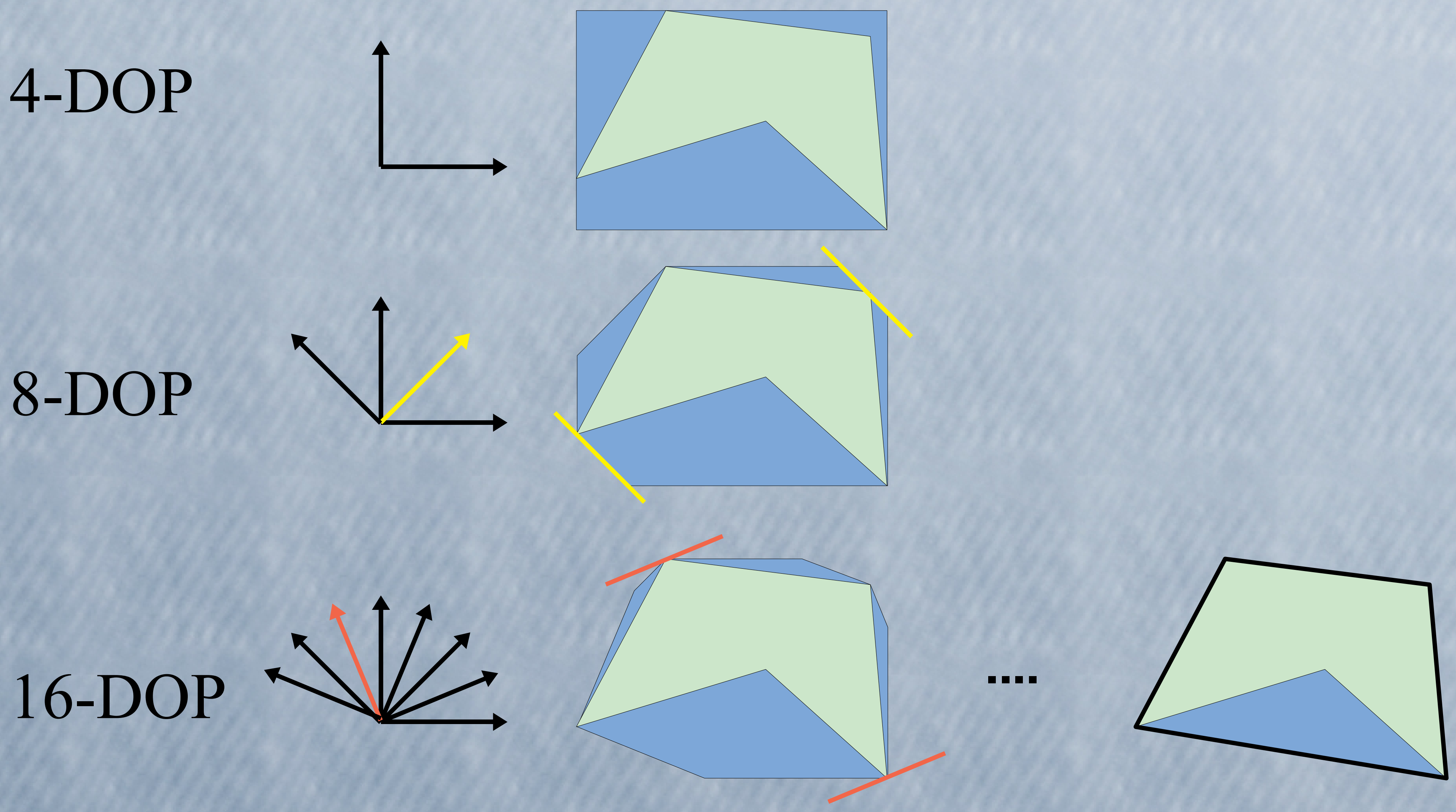
Discrete Oriented Polytopes (DOP)

- Called *k*-DOP, where *k*/2 is the number of directions
- Generalization of AABB (4-DOP in 2D)
  - more directions to choose from than the dimensionality
- Fitting exactly the same as AABB but for more directions



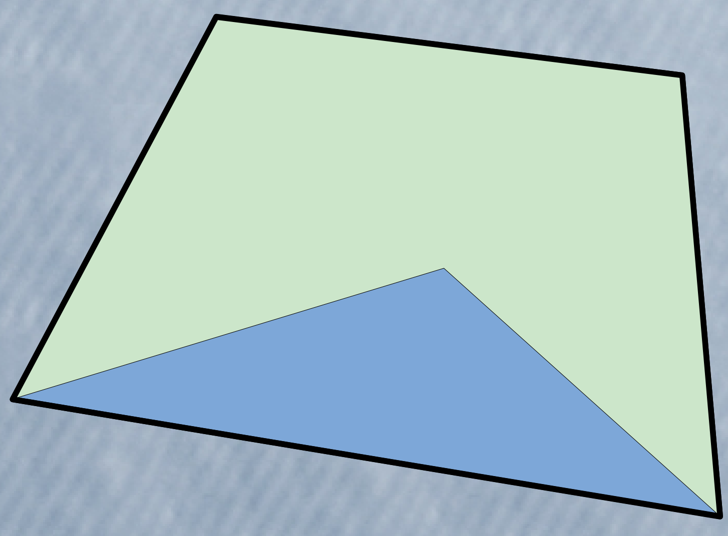


k-DOPs



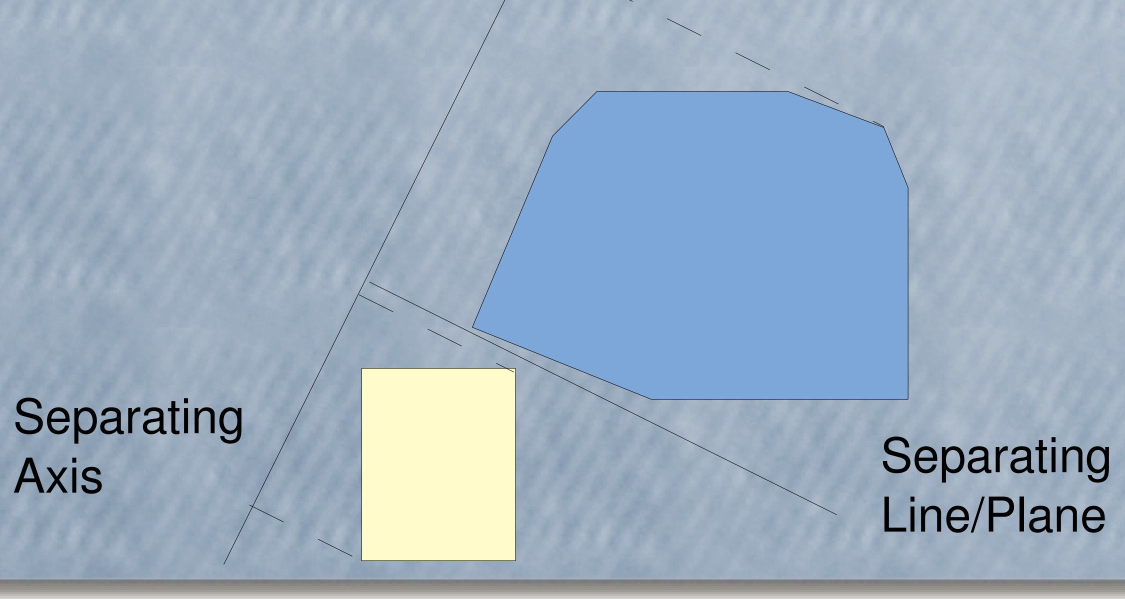
Convex Hulls

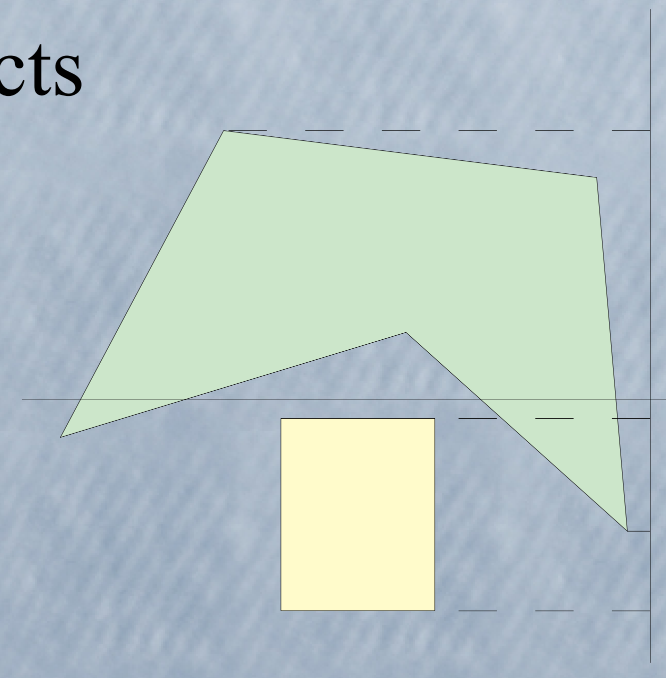
- Composed of the extremal points of the object (in all directions)
- Expensive to compute
- Tightext convex bounding volume possible



Separating Axis Theorem (SAT)

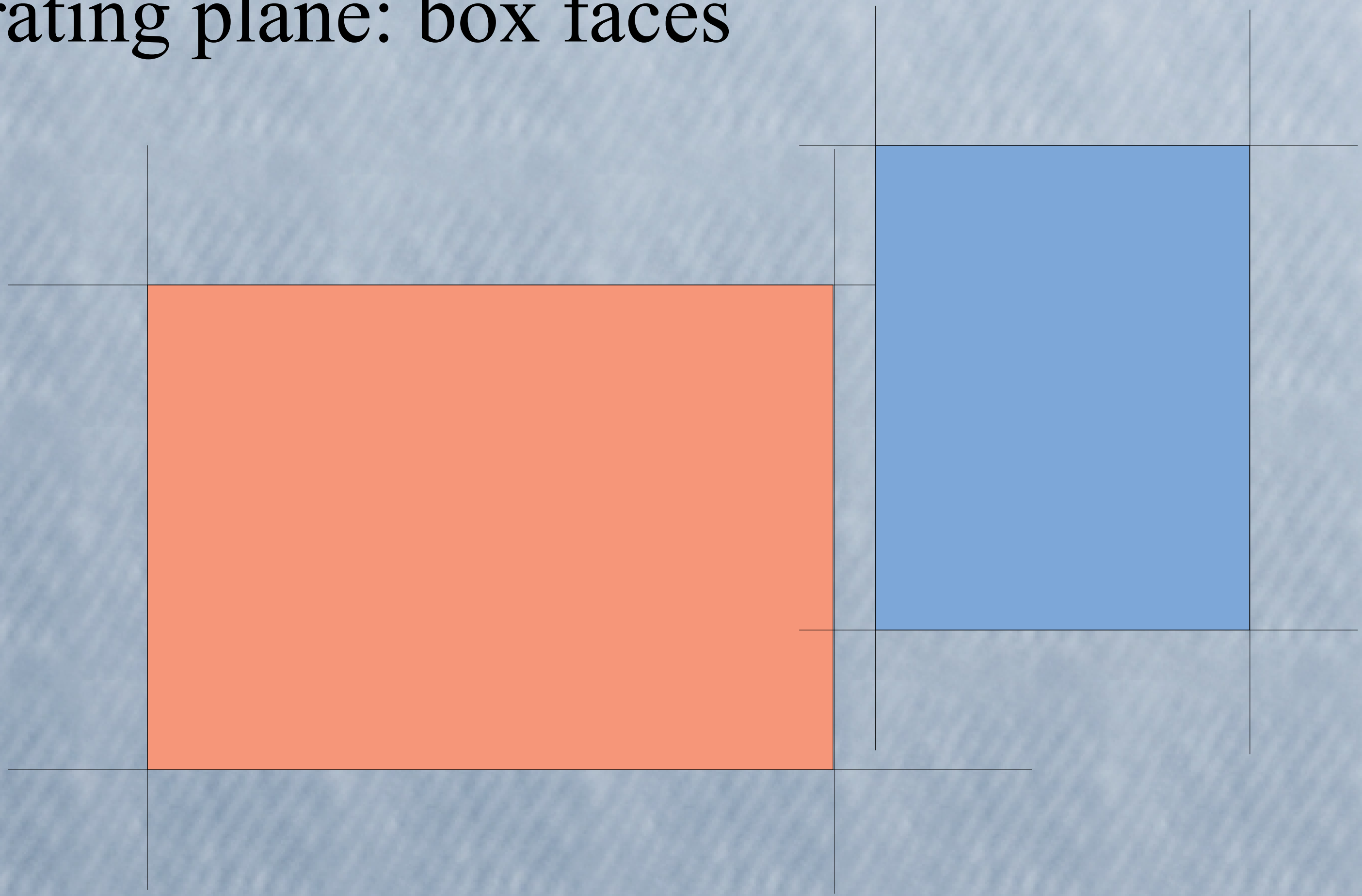
- *Separating axis* – a direction in which the projection of two objects does not overlap
- Quick test: two convex objects intersect iff there does not exist a separating axis for them
- Does not work on concave objects





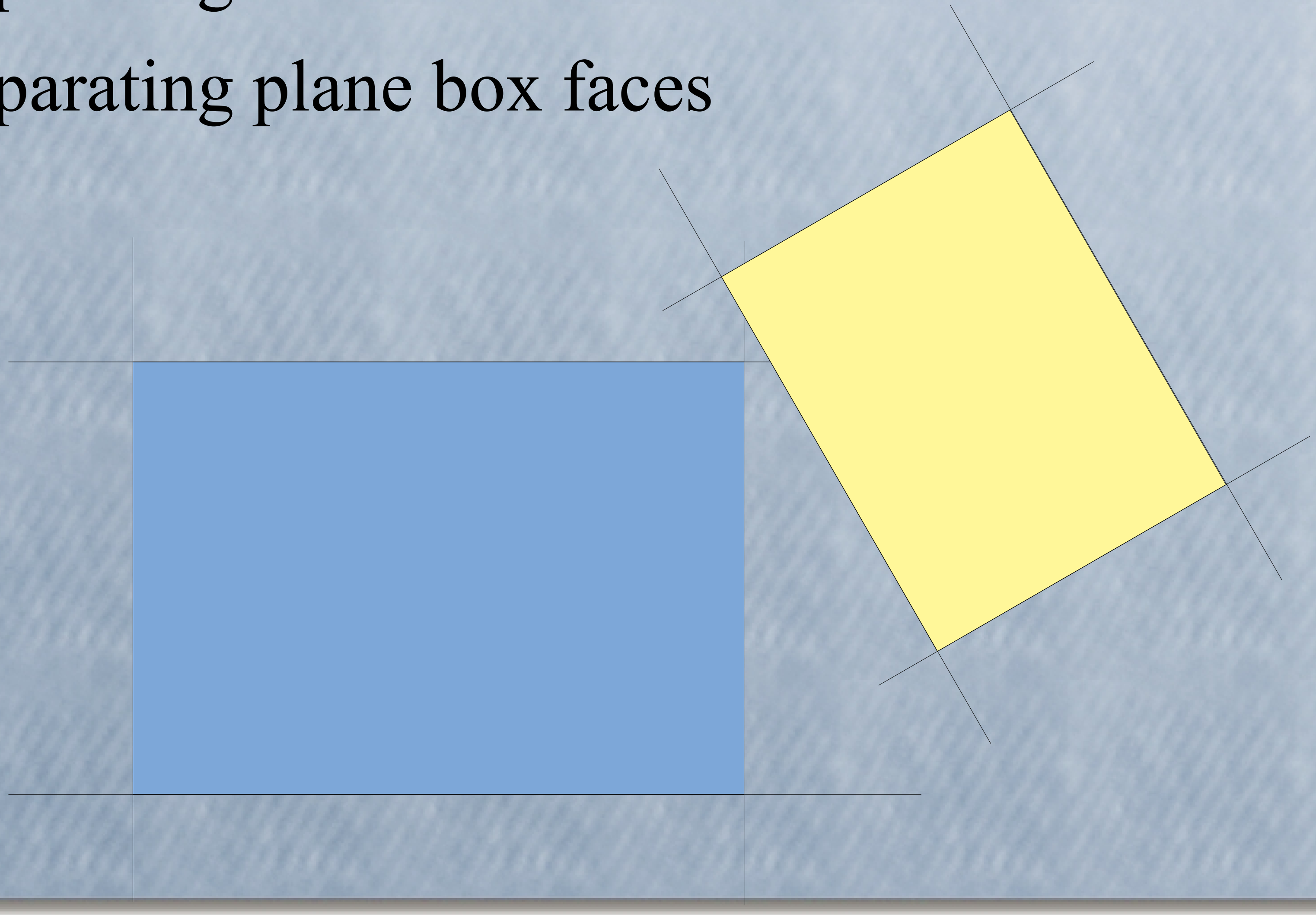
Axis-Aligned Bounding Box (AABB)

- Separating axis candidates: x,y,z...
- Separating plane: box faces



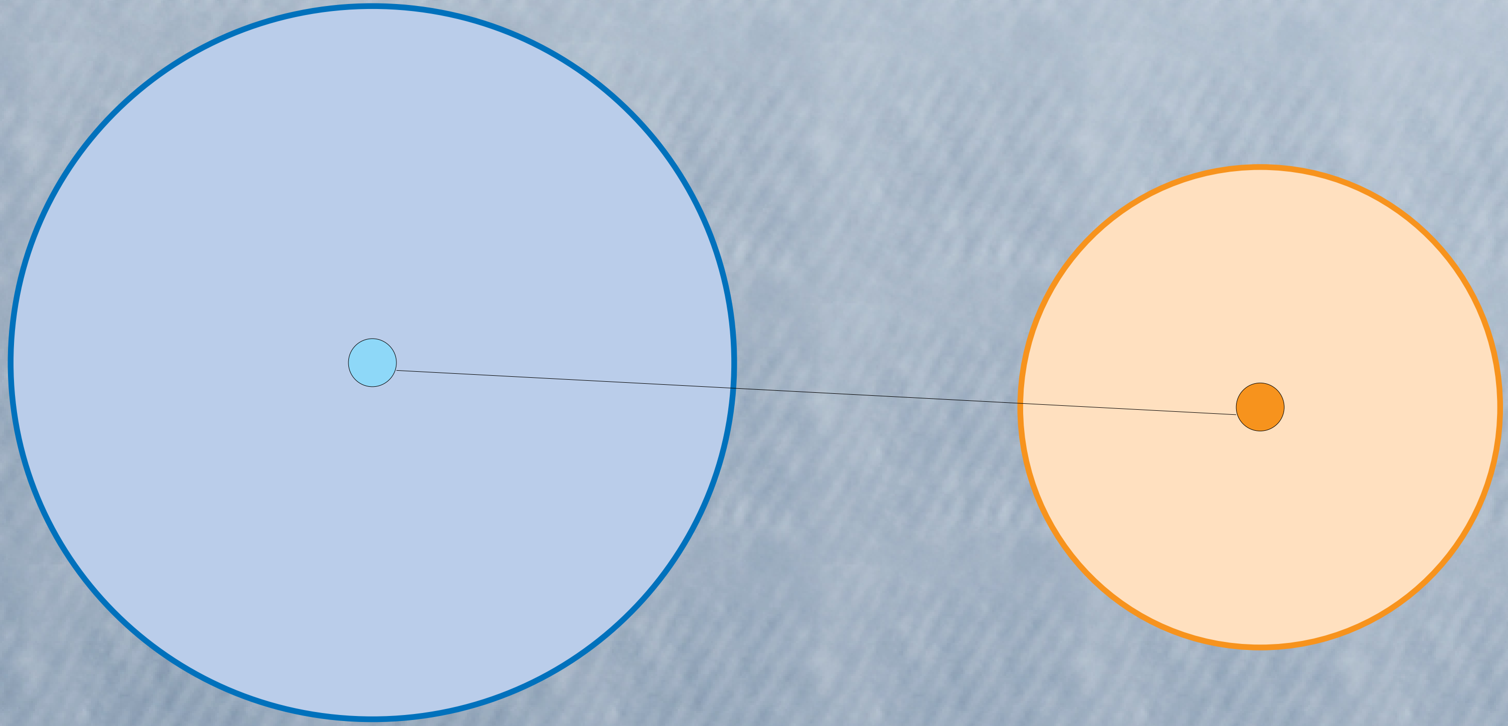
Oriented Bounding Box (OBB)

- Separating axis candidates normals of box faces
- Separating plane box faces



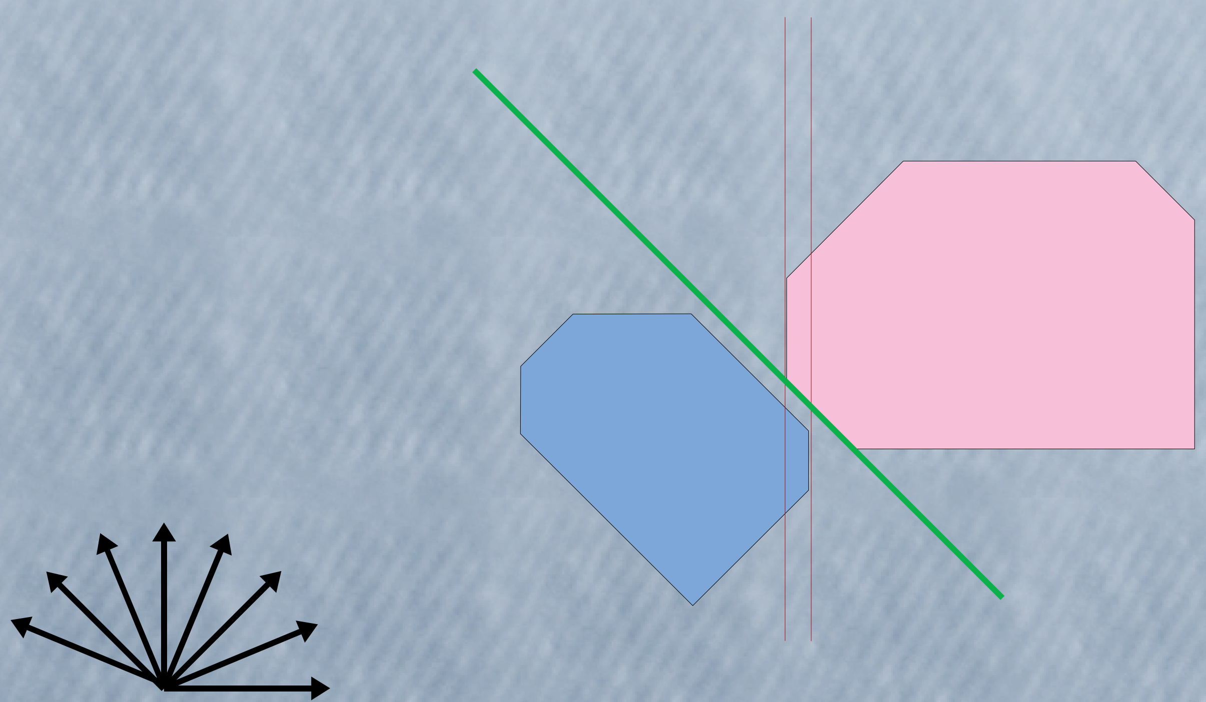
Circles and Spheres

- Separating axis candidate: line between the centers
- Separating plane: tangent planes



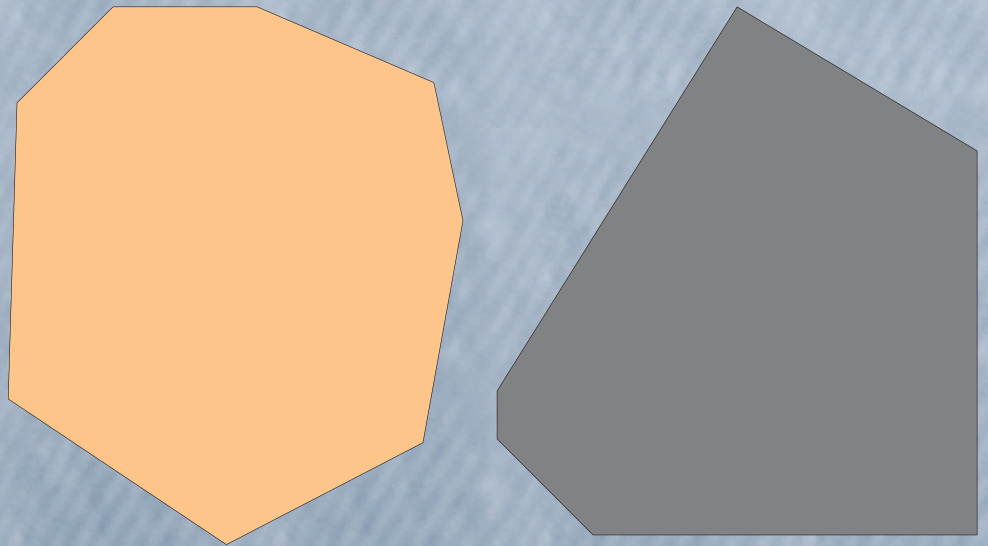
Discrete Oriented Polytopes (DOP)

- Candidate separating axes: AABB, but more directions to test than just coordinate axes
- Separating planes: polytope faces



Convex Hulls

- Candidate separating axes: many, face normals
- Separating planes: hull faces
- Better to find closest points...



Next Time

- Nearest neighbor searches
- Reading: TBA