Proximity Queries

Sung-Eui Yoon
(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/MPA

KAIST

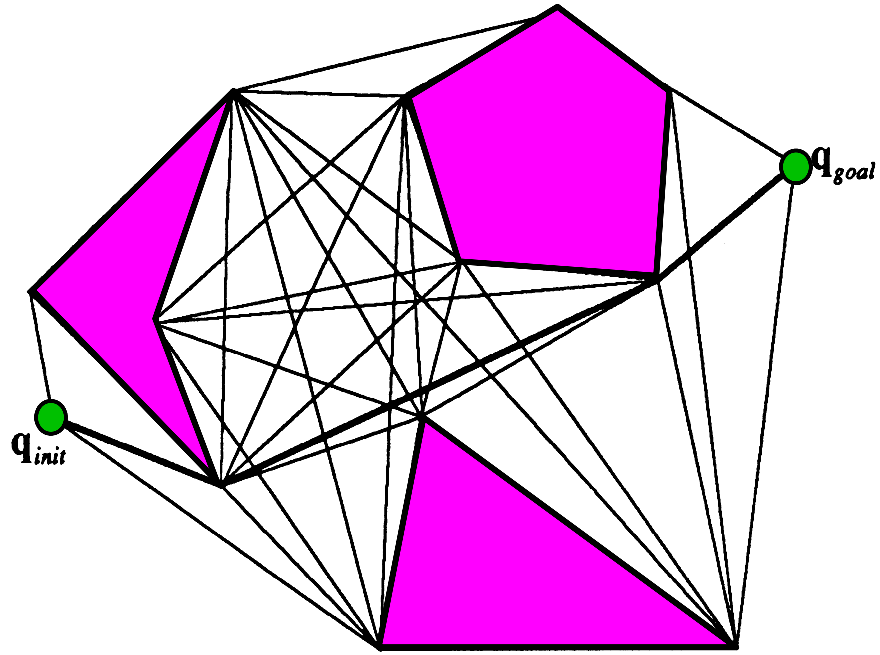
Class Objectives

- **Understand collision detection and distance computation**
  - **Bounding volume hierarchies**
  - **Tracking features**

Two geometric primitives in configuration space

- **CLEAR(q)**
  Is configuration q collision free or not?

- **LINK(q, q')**
  Is the straight-line path between q and q' collision-free?



Problem

- **Input: two objects A and B**
- **Output:**
  - **Distance computation: compute the distance (in the workspace) between A and B**

OR

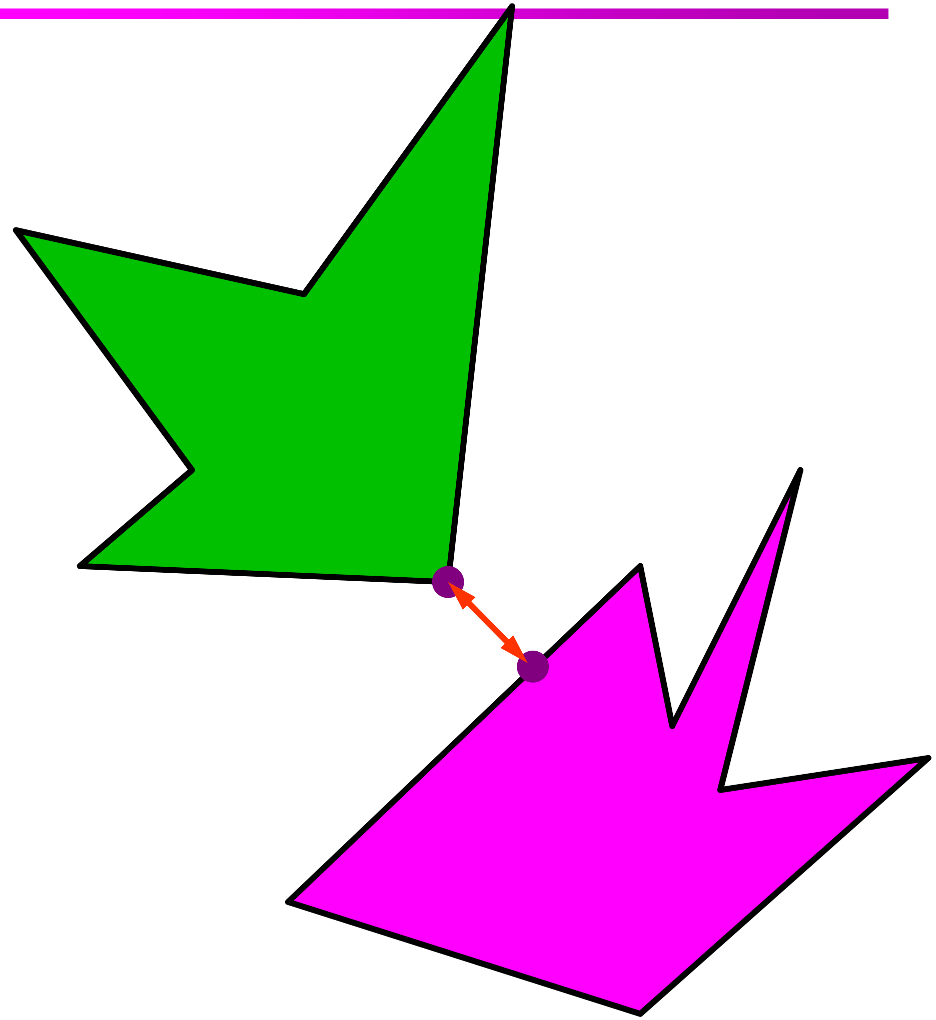
  - **Collision detection: determine whether A and B collide or not**

KAIST

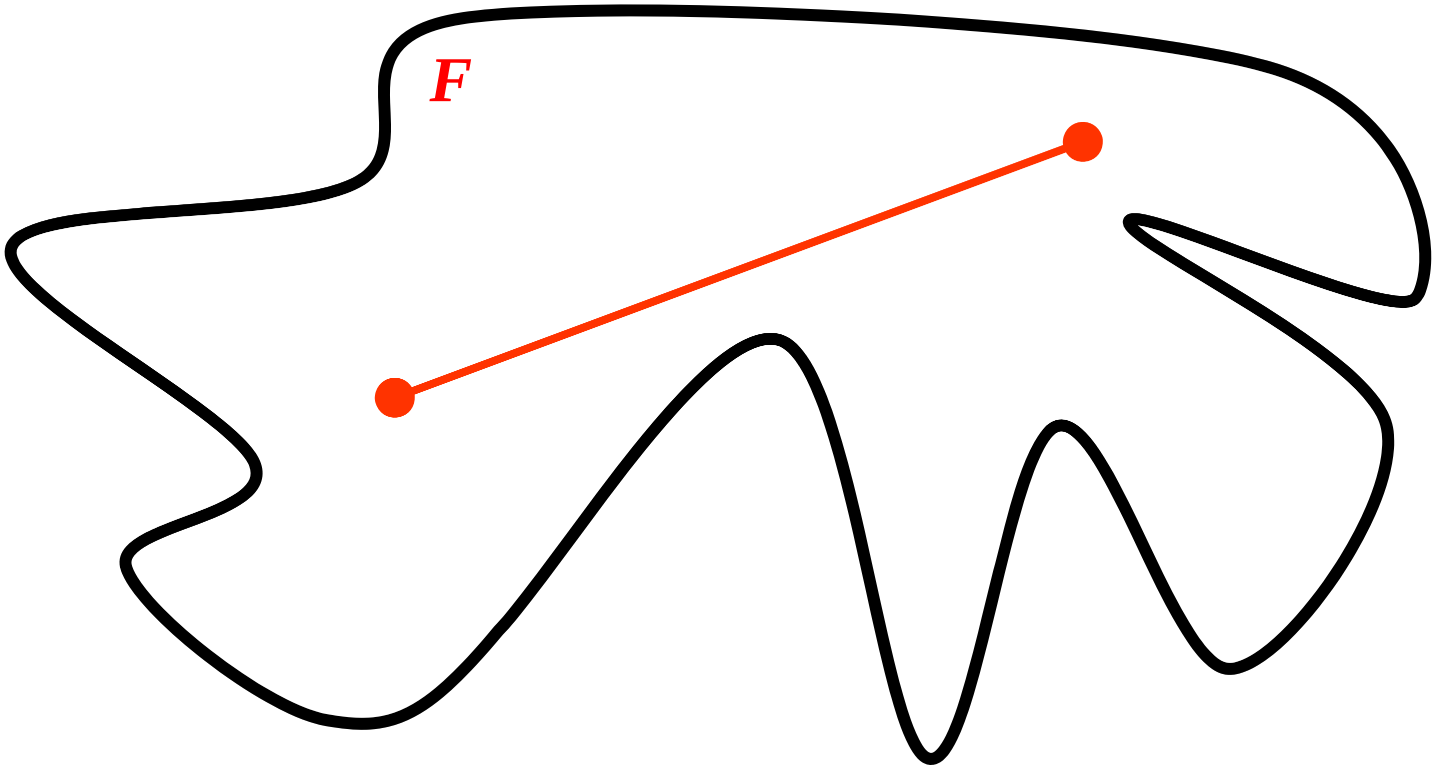
Collision detection vs. distance computation

- **The distance between two objects (in the workspace) is the distance between the two closest points on the respective objects**

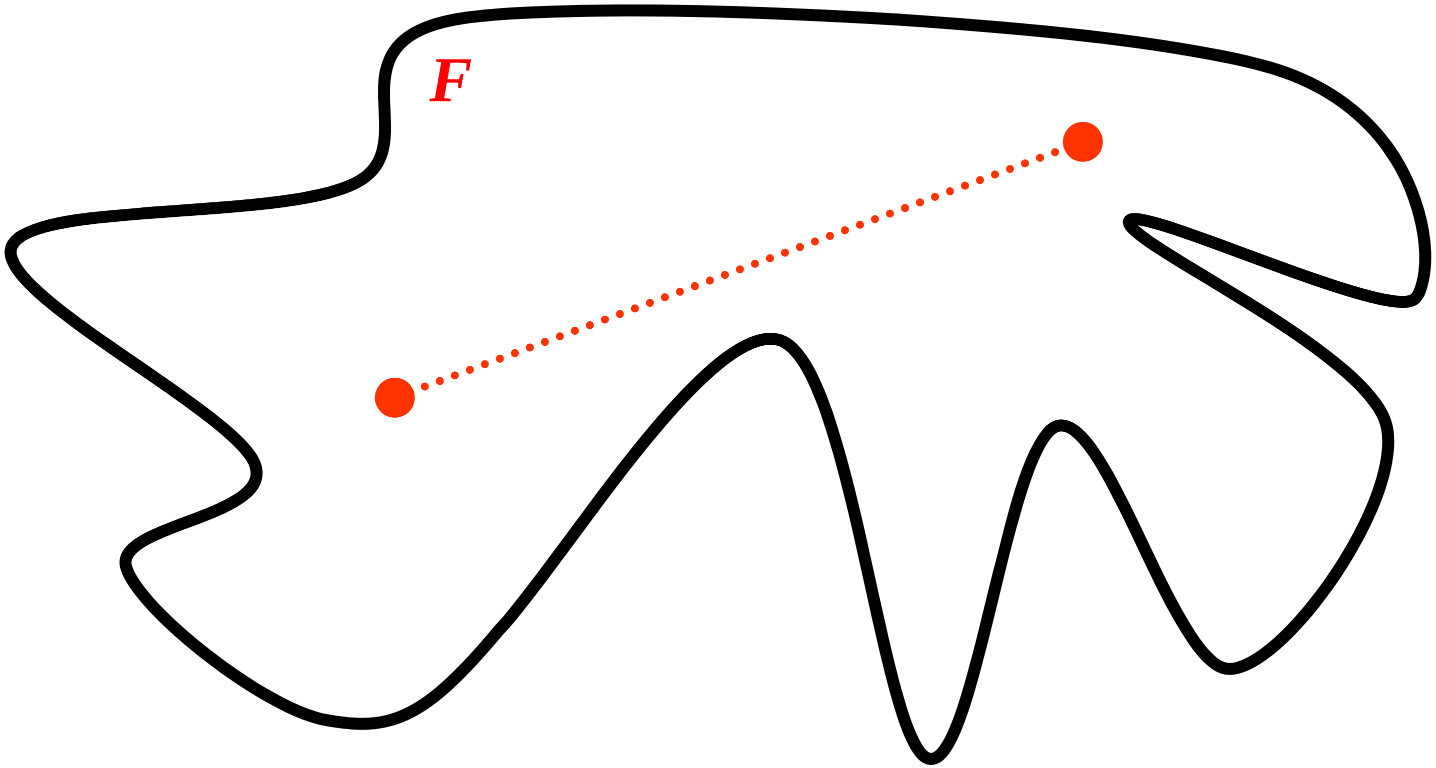
- **Collision if and only if distance = 0**



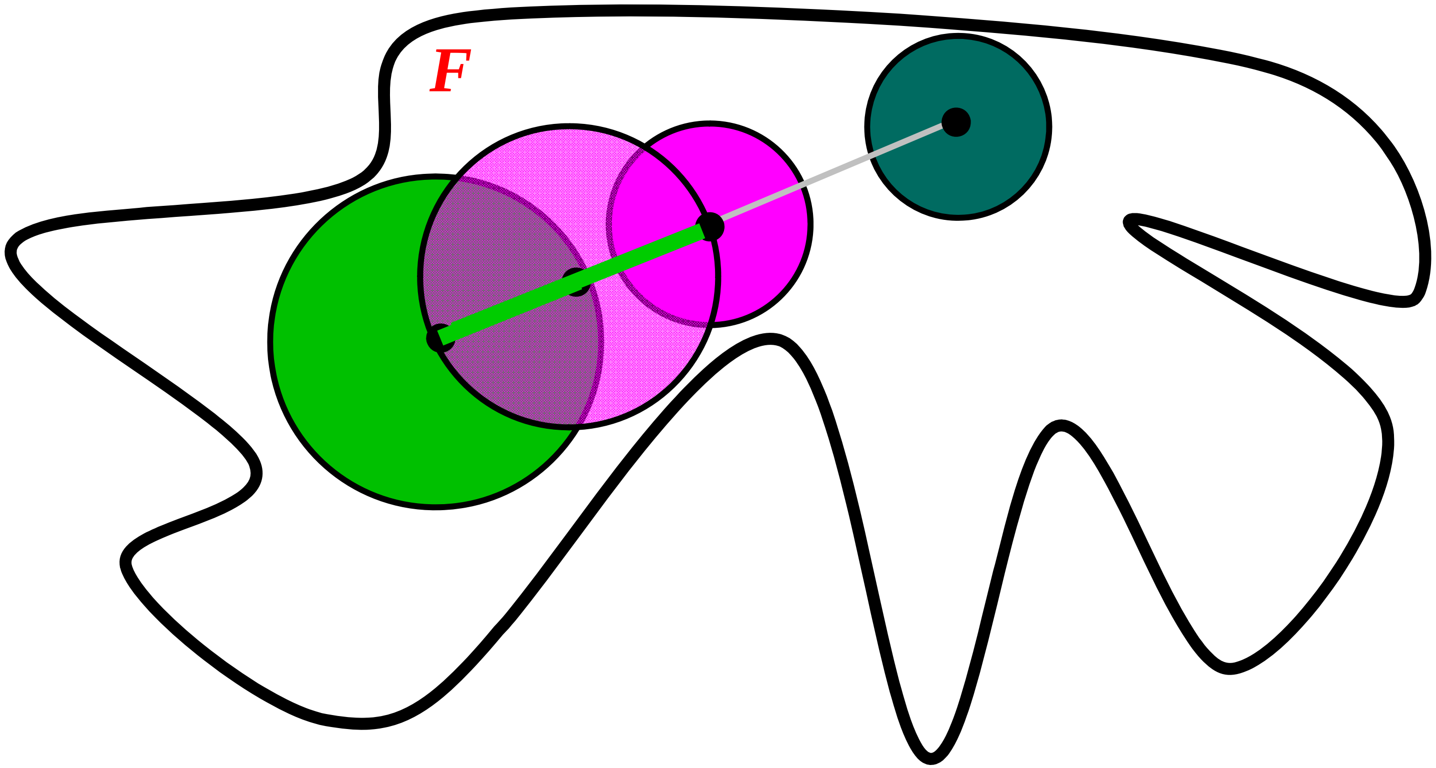
Collision detection does not allow us to check for free path rigorously



Collision detection does not allow us to check for free path rigorously



Use distance to check for free path rigorously



Use distance to check for free path rigorously

```
Link(q0, q1)
if q0∈N(q1) or q1∈N(q0)
then
  return TRUE.
else
  q' = (q0+q1)/2.
  if q' is in collision
  then
    return FALSE
  else
    return Link(q0, q') && Link(q1,q').
```

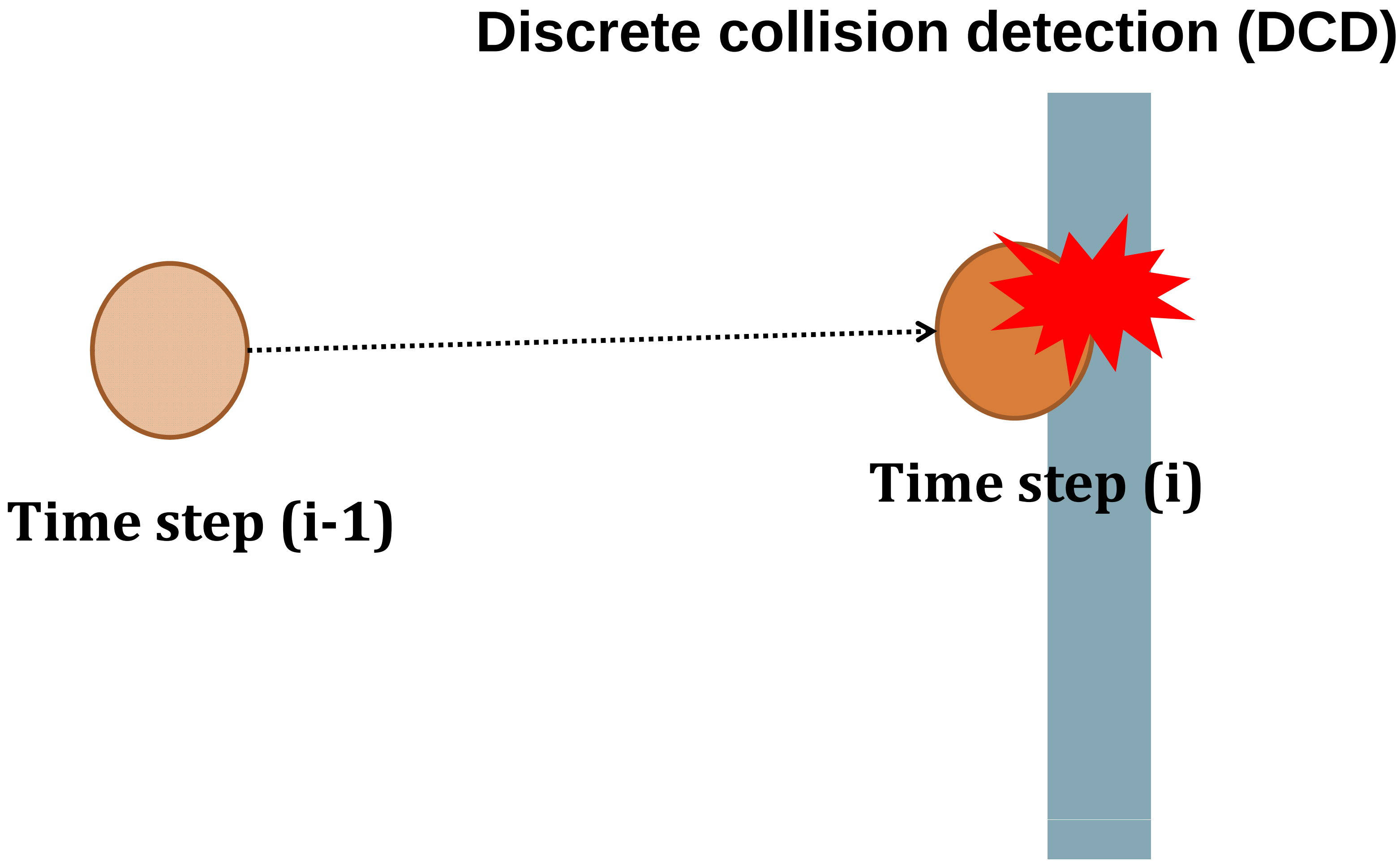
Applications

- Robotics
  - Collision avoidance
  - Path planning
- Graphics & virtual environment simulation
- Haptics
  - Collision detection
  - Force proportional to distance

Collision Detection

- Discrete collision detection
- Continuous collision detection

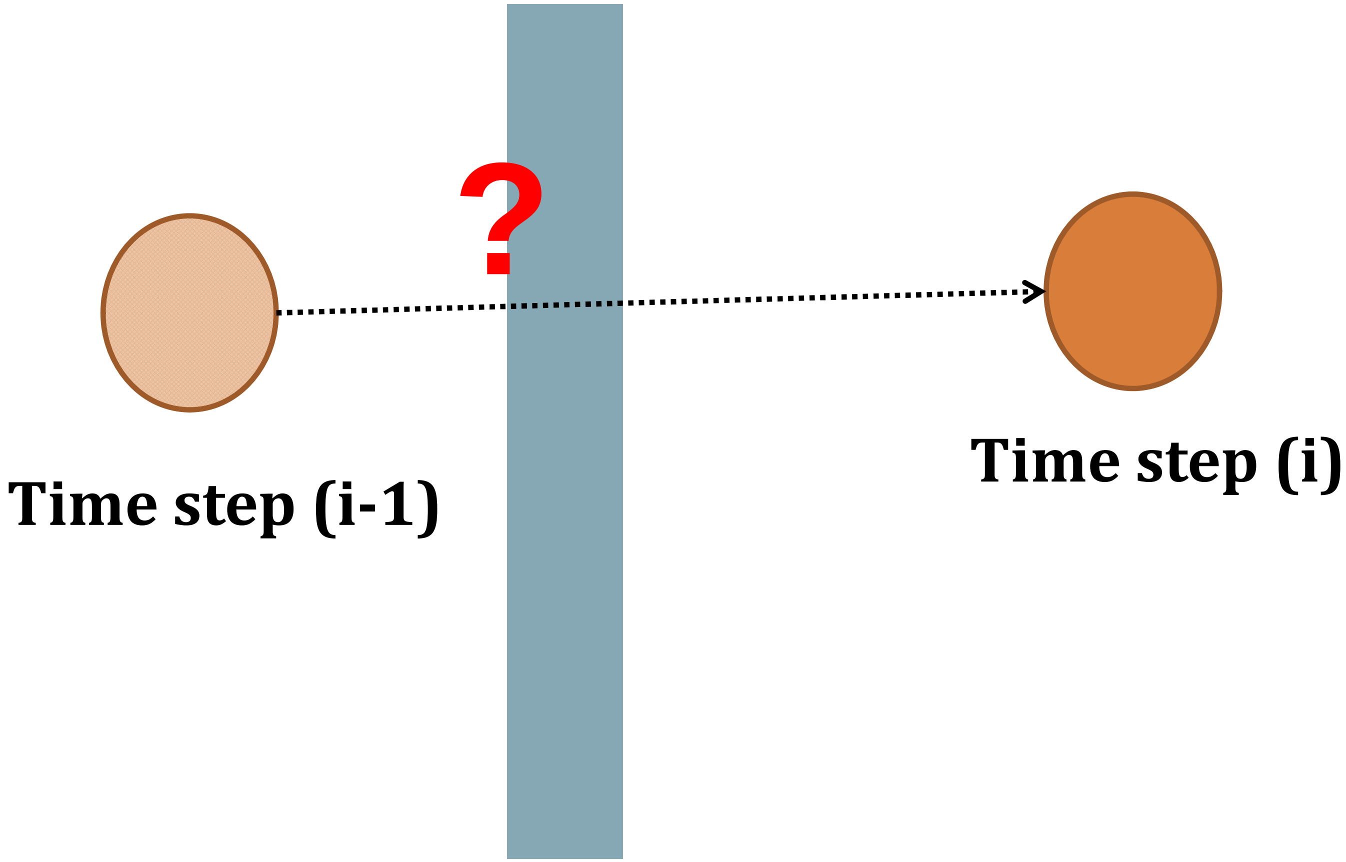
Discrete VS Continuous



From Duksu's slides

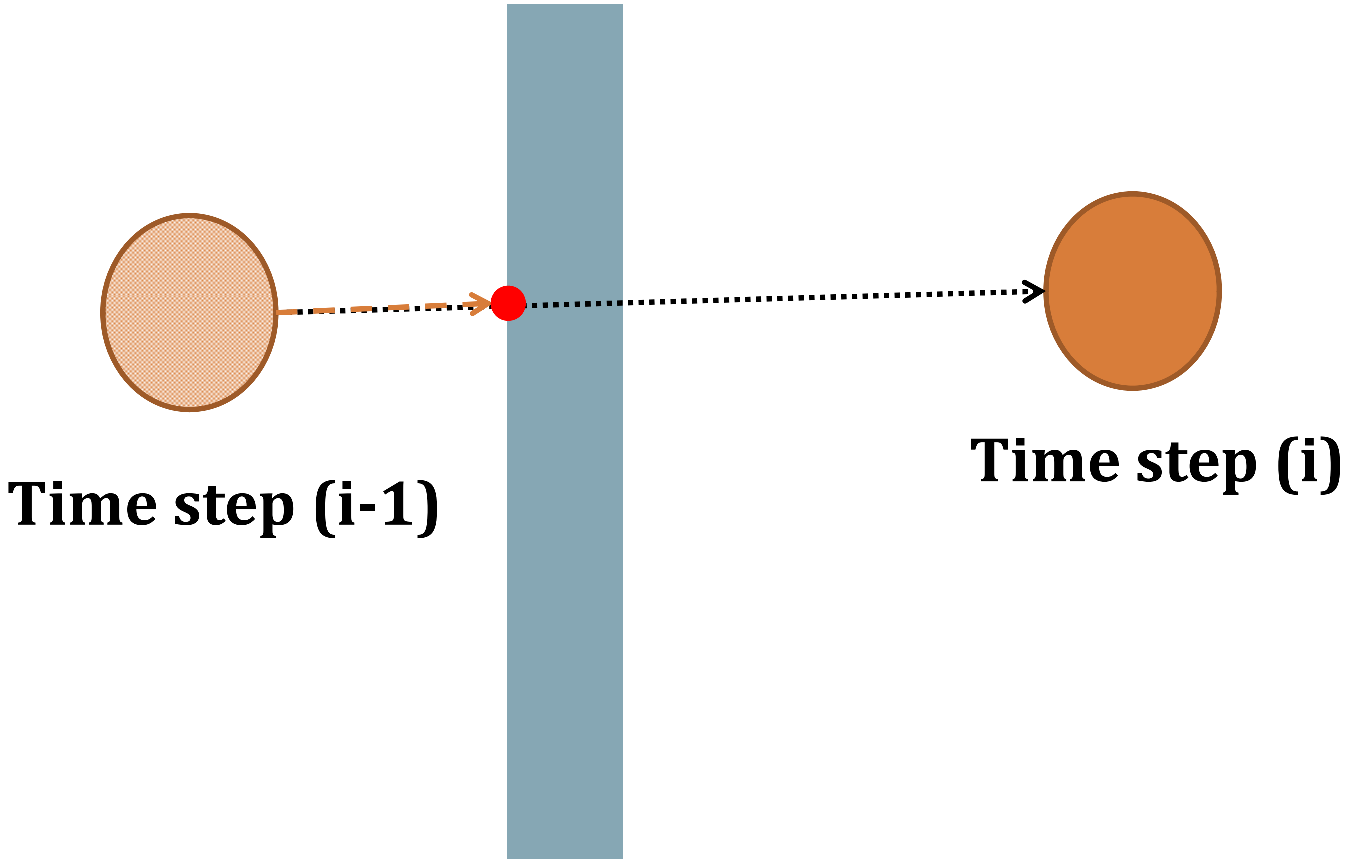
Discrete VS Continuous

Discrete collision detection (DCD)



Discrete VS Continuous

Continuous collision detection(CCD)



Discrete VS Continuous

	Continuous CD	Discrete CD
Accuracy	Accurate	May miss some collisions
Computation time	Slow	Fast

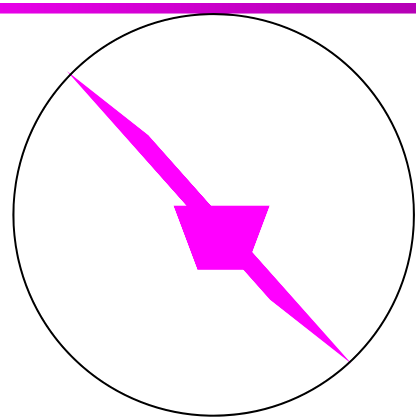
Collision Detection

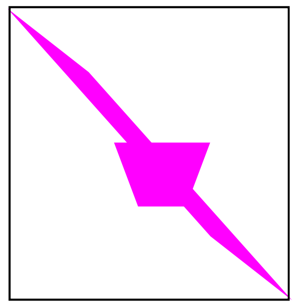
- **Discrete collision detection**
- **Continuous collision detection**

- **These are typically accelerated by bounding volume hierarchices (BVHs)**

Bounding Volumes

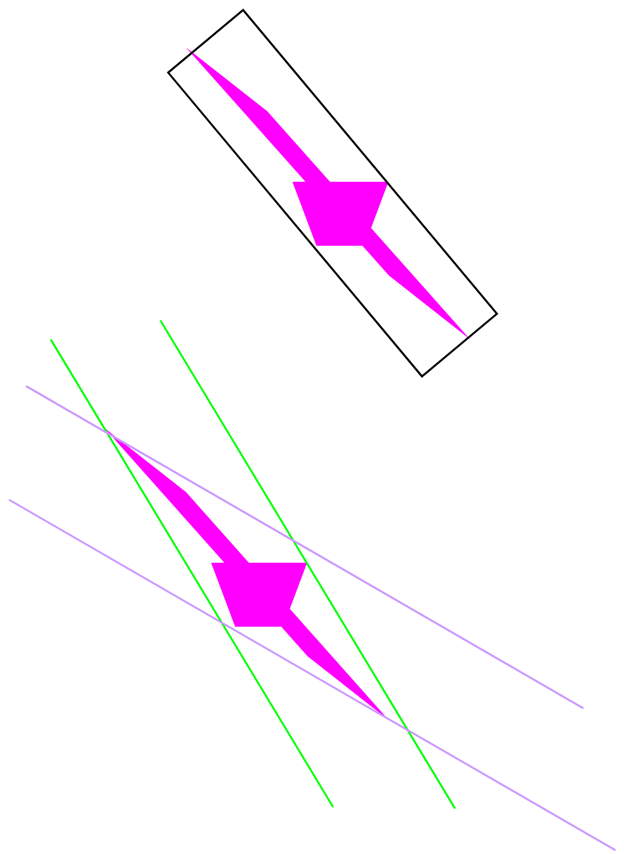
- **Sphere [Whitted80]**
  - **Cheap to compute**
  - **Cheap test**
  - **Potentially very bad fit**
- **Axis-aligned bounding box**
  - **Very cheap to compute**
  - **Cheap test**
  - **Tighter than sphere**





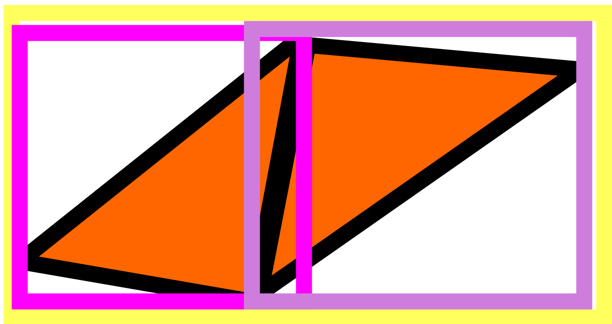
Bounding Volumes

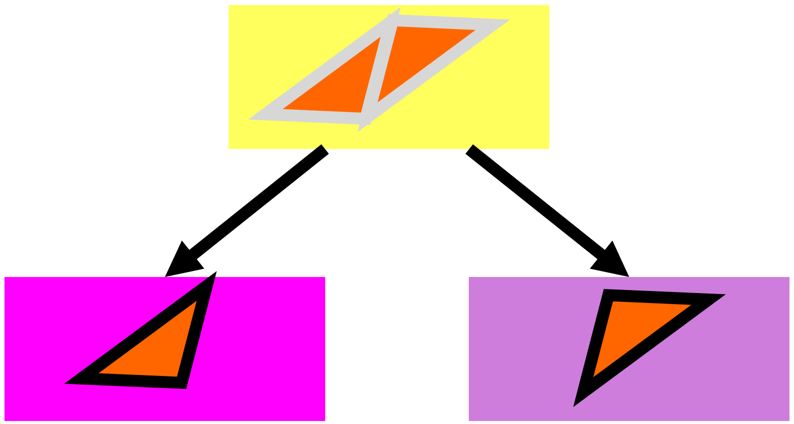
- **Oriented bounding box**
  - **Fairly cheap to compute**
  - **Fairly cheap test**
  - **Generally fairly tight**
- **Slabs / K-dops**
  - **More expensive to compute**
  - **Fairly cheap test**
  - **Can be tighter than OBB**



Bounding Volume Hierarchies (BVHs)

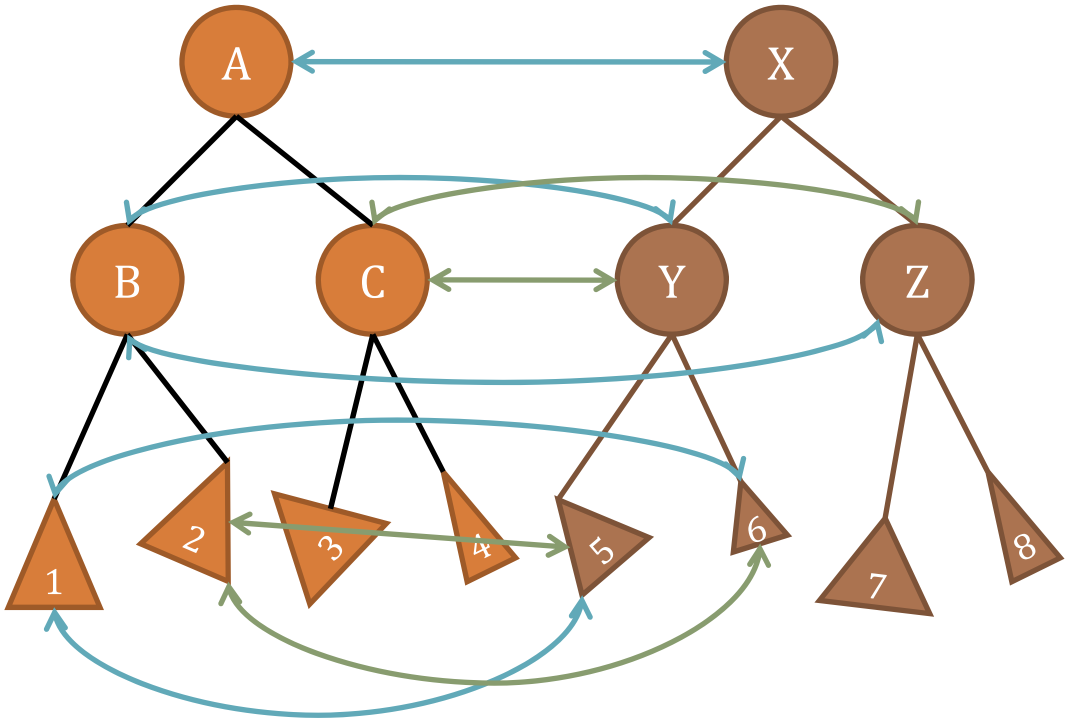
- Organize bounding volumes recursively as a tree
- Construct BVHs in a top-down manner
  - Use median-based partitioning or other advanced partitioning methods



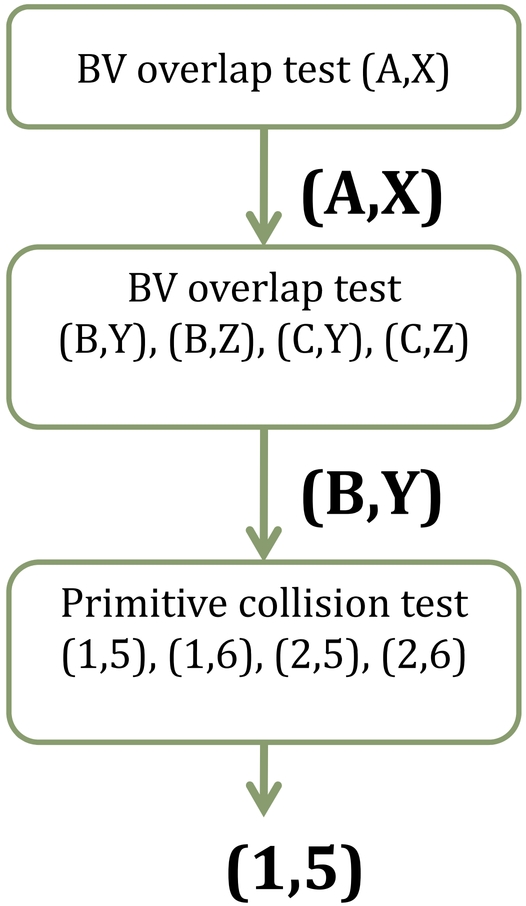


A BVH

Collision Detection with BVHs



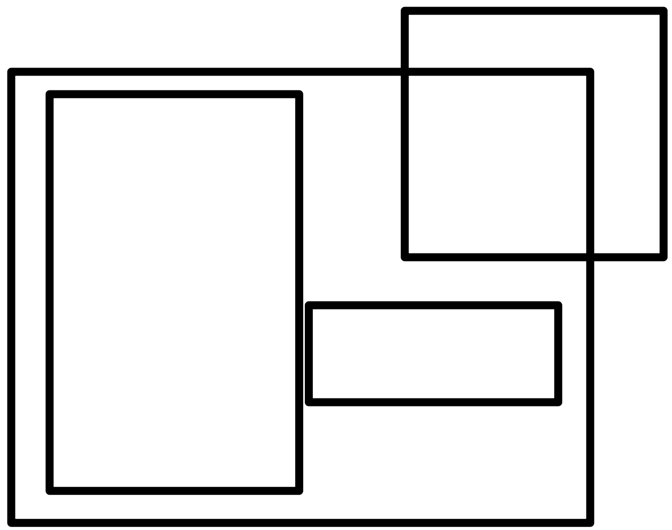
Triangle 1 and 5 have a collision!



From Duksu's slides

BVH Traversal

- Traverse BVHs with depth-first or breadth-first
- Refine a BV node first that has a bigger BV



Continuous Collision Detection

- BVHs are also widely used
- Models a continuous motion for a primitive, whose positions are defined at discrete time steps
  - E.g., linear interpolation

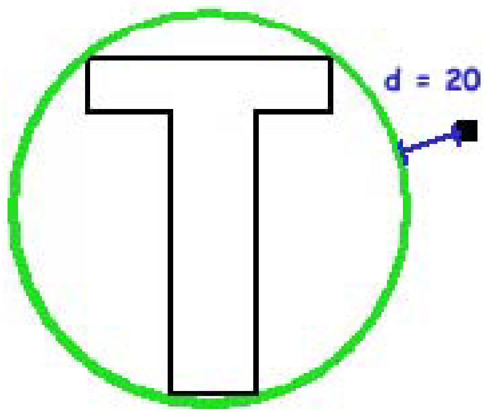
KAIST

Computing distances

- Depth-first search on the binary tree
  - Keep an updated minimum distance
  - Depth-first → more pruning in search
- Prune search on branches that won't reduce minimum distance
- Once leaf node is reached, examine underlying convex polygon for exact distance

Simple example

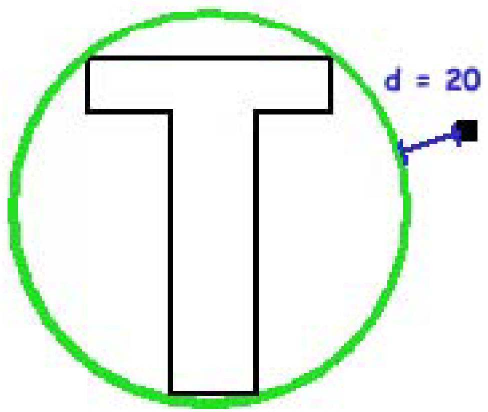
- **Set initial distance value to infinity**



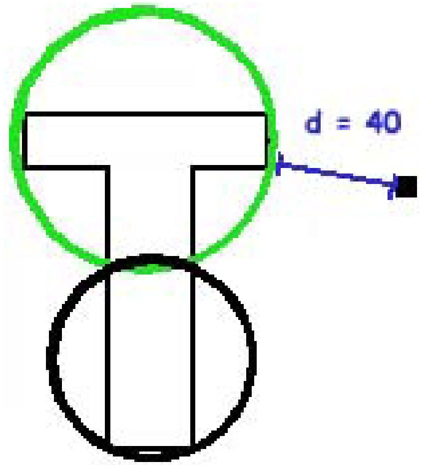
Start at the root node.
20 < infinity, so continue searching

Simple example

- **Set initial distance value to infinity**



Start at the root node.
20 < infinity, so continue searching.

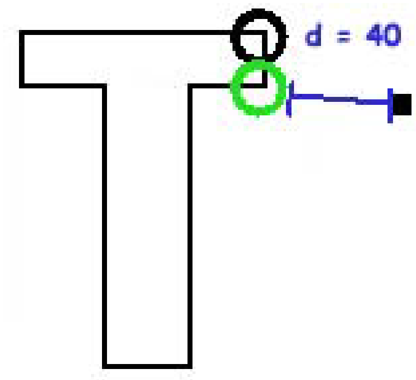


40 < infinity, so continue searching recursively.

- **Choose the nearest of the two child spheres to search first**

Simple example

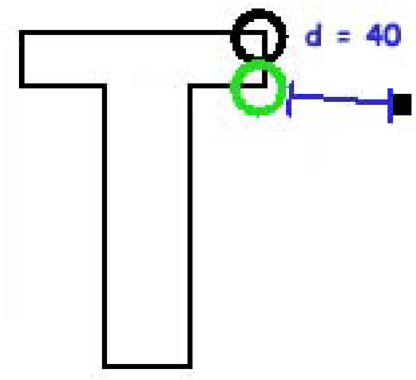
- **Eventually reach a leaf node**



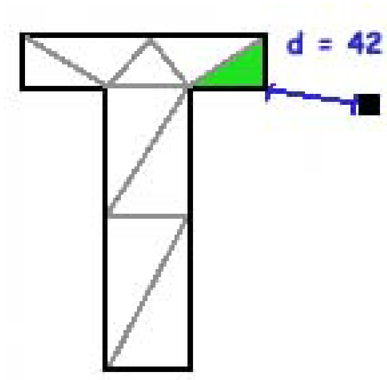
40 < infinity; examine the polygon to which the leaf node is attached.

Simple example

- **Eventually reach a leaf node**



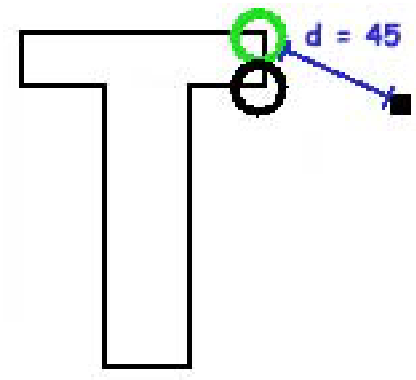
40 < infinity; examine the polygon to which the leaf node is attached.



Call algorithm to find exact distance to the polygon. Replace infinity with new minimum distance (42 in this case).

Simple example

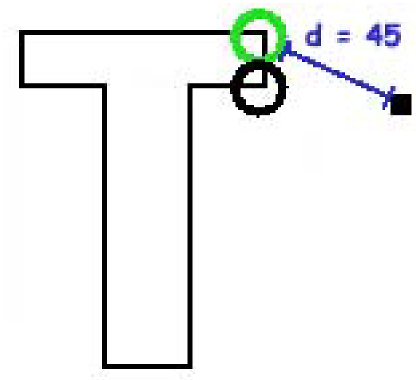
- **Continue depth-first search**



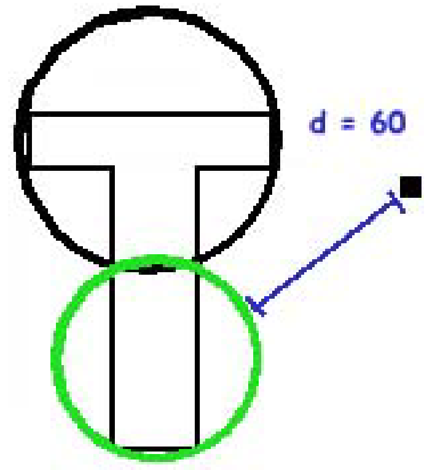
45 > 42; don’t search this branch any further

Simple example

- **Continue depth-first search**



45 > 42; don't search this branch any further



60 > 42; we can prune this half of our tree from the search

Running time: build the tree

- Roughly balanced binary tree
- Expected time $O(n \log n)$
  - Time to generate node v is proportional to the number of leaf nodes descended from v.
- Tree is built only once and can often be pre-computed.

Running time: search the tree

- **Full search**
  - $O(n)$ time to traverse the tree, where n = number of leaf nodes
  - Plus time to compute distance to each polygon in the underlying model
- **The algorithm allows a pruned search:**
  - Worst case as above; occurs when objects are close together
  - Best case: $O(\log n)$ + a single polygon calculation
  - Average case ranges between the two.

KAIST

General case

- If second object is not a single point, then search & compare 2 trees
  - Use two BVHs and perform the BVH traversal

Extension: relative error

- **When updating the minimum distance d' between objects, set** $d' = (1-a)d$ **(*d* = actual distance).**
  - ***a* is our relative error, why?**
  - **Guarantee that objects are at least *d'* apart**

$$d_{\min} \geq d' \Rightarrow d_{\min} \geq (1-a)d \Rightarrow (d - d_{\min})/d \leq a$$

  - **$(1-a)d = 0$ iff $d = 0$; correctly detects collisions**
- **Improves performance by pruning search**

Empirical results

- Tested on a set of six 3D chess pieces
  - Non-convex
  - Each piece has roughly 2,000 triangles
  - Each piece has roughly 5750 leaf nodes
- Relative error of 20% → more pruning in search → speedup of 2 orders of magnitude
- Objects close together → less pruning in search → less efficient

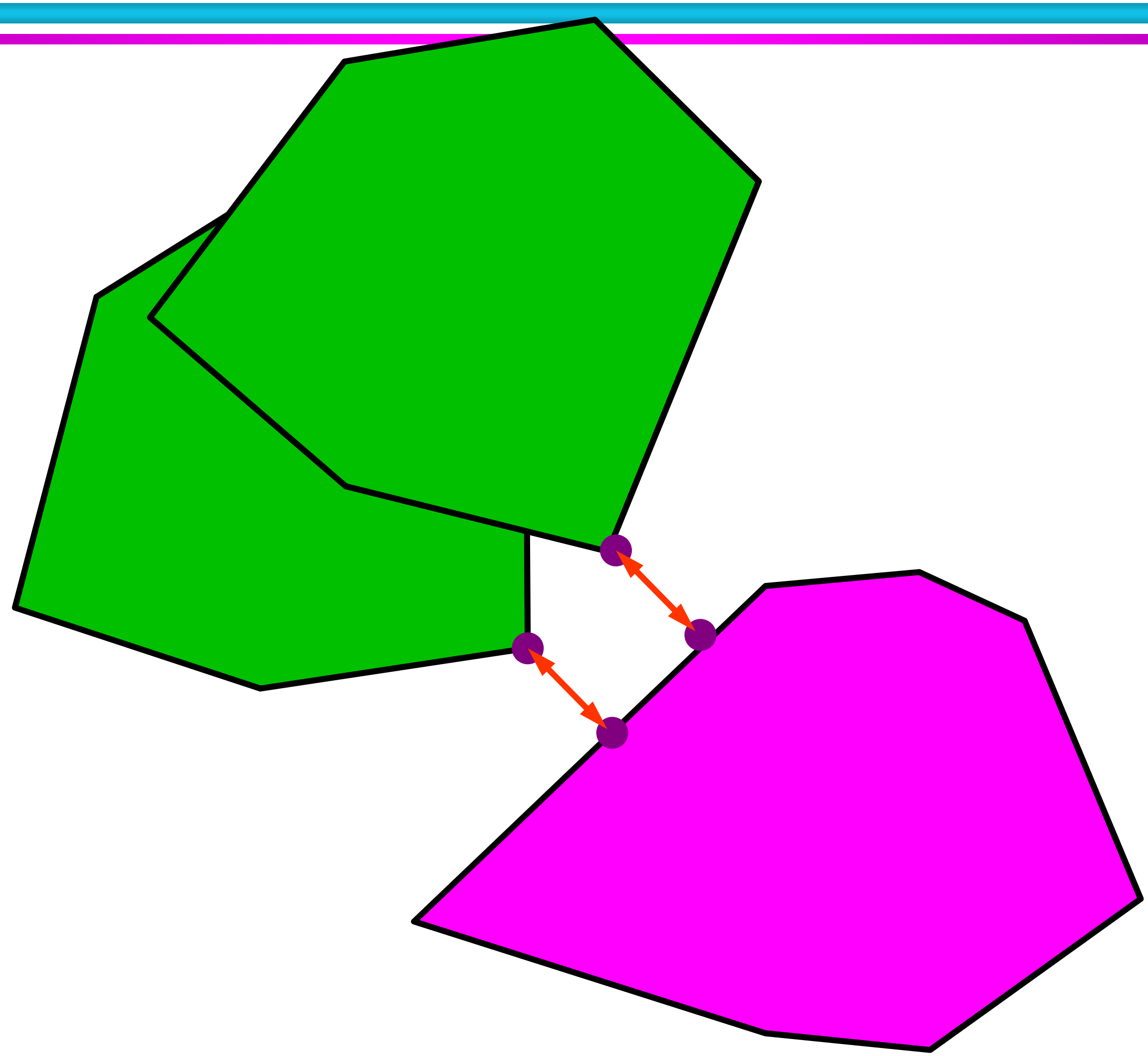
Tracking the closest pair

- ***V-Clip: Fast and Robust Polyhedral Collision Detection,*** **B. Mirtich, 1997**

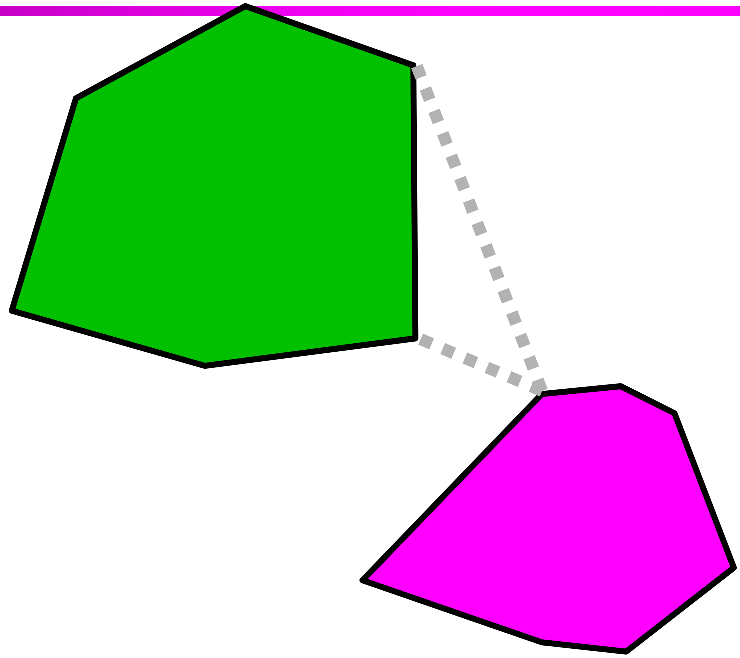
Key features

- Work for convex objects in 2-D or 3-D environements
- Compute the exact distance
- Efficiency from motion coherence

Motion coherence



Iterative improvement



- For convex objects, an iterative step always results in a decrease in the candidate “feature” pair.

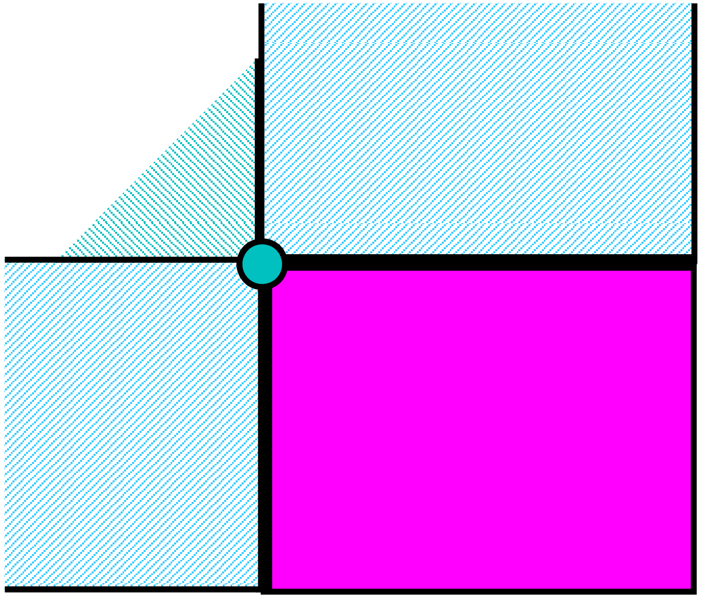
Features and their Voronoi regions

- **Features**
  - **Vertices**
  - **Edges**
- **For a feature X in a convex polygon, the Voronoi region $\mathrm{vor}(X)$ is the set of points outside of the polygon that are as close to X as to any other feature on the polygon.**

KAIST

Voronoi regions of points and edges

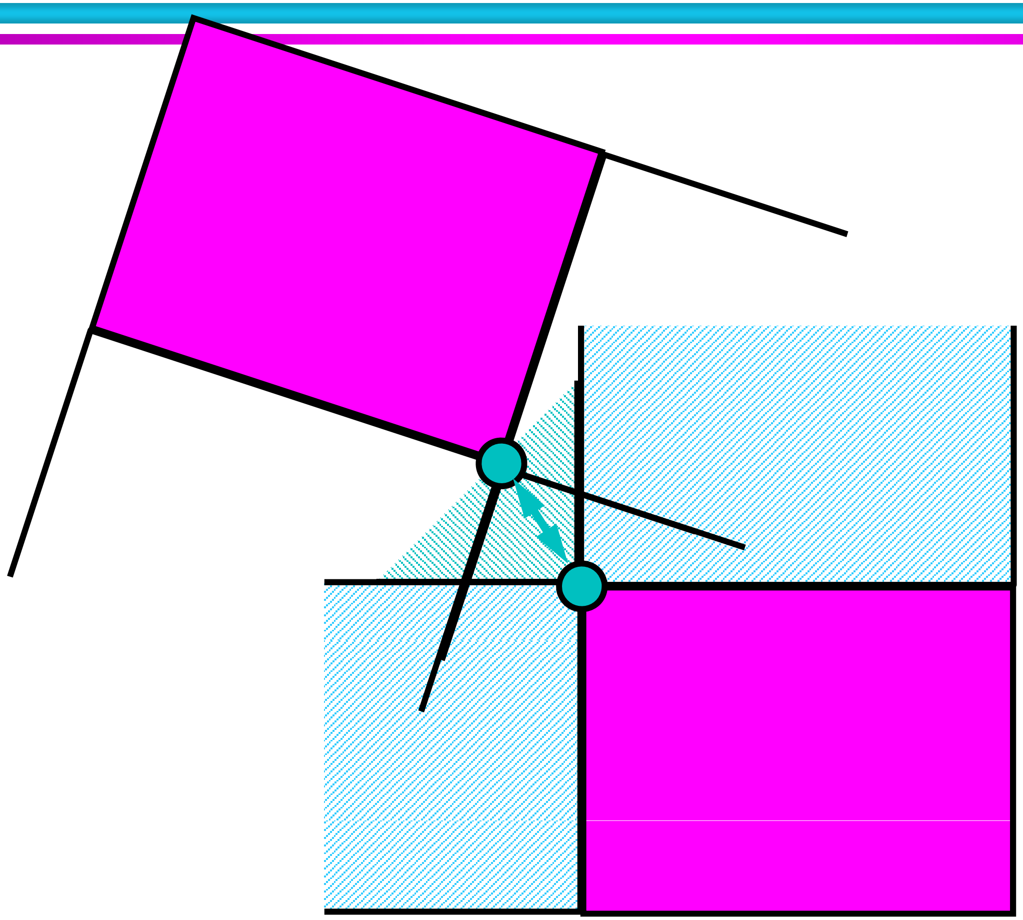
- **Voronoi region of a point**
- **Voronoi region of an edge**



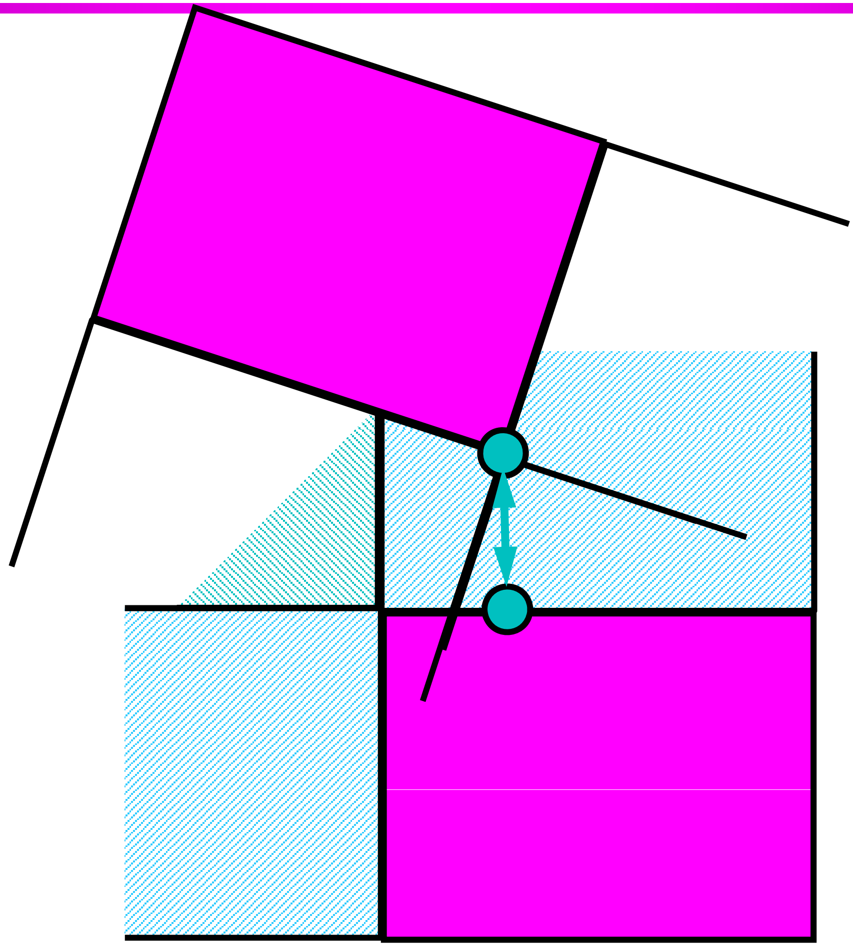
Critical condition

- **Theorem: Let X and Y be a pair of features from disjoint convex polygons and let $x \in X$ and $y \in Y$ be the closest pair of points between X and Y. If $x \in \mathrm{vor}(Y)$ and $y \in \mathrm{vor}(X)$, then x and y are a globally closest pair of points between the polygons.**

Critical condition: vertex-vertex



Critical condition: vertex-edge



Sketch of the algorithm

```
1: Start with a candidate feature pair
   (X,Y).
2: if (X,Y) satisfies the critical condition
3: then
     return (X,Y) as the closest pair.
4: else
     Update either X or Y to its neighboring
     feature. Go to (2).
```


KAIST

3-D case

- More features
  - Vertices
  - Edges
  - Faces
- More cases for the critical conditions
  - Vertex-vertex
  - Vertex-edge
  - Vertex-face
  - Edge-edge
  - Edge-face

KAIST

Class Objectives were:

- Understand collision detection and distance computation
  - Bounding volume hierarchies
  - Tracking features

Next Time…

- Probabilistic Roadmaps