
CS686: Proximity Queries

Sung-Eui Yoon
(윤성의)

Course URL:

<http://sgvr.kaist.ac.kr/~sungeui/MPA>

KAIST



Presentation Guideline: Expectations

- **Good summary, not full detail, of the paper**
 - **Just 20 min for the talk + 3 min for quiz**
 - **Talk about motivations of the work**
 - **Give a broad background on the related work**
 - **Explain main idea and results of the paper**
 - **Discuss strengths and weaknesses of the method**

High-Level Ideas

- **Deliver most important ideas and results**
 - **Do not talk about minor details**
 - **Give enough background instead**

- **Spend most time to figure out the most important things and prepare good slides for them**
 - **If possible, re-use existing slides/videos with acknowledgement**

Overall Structure

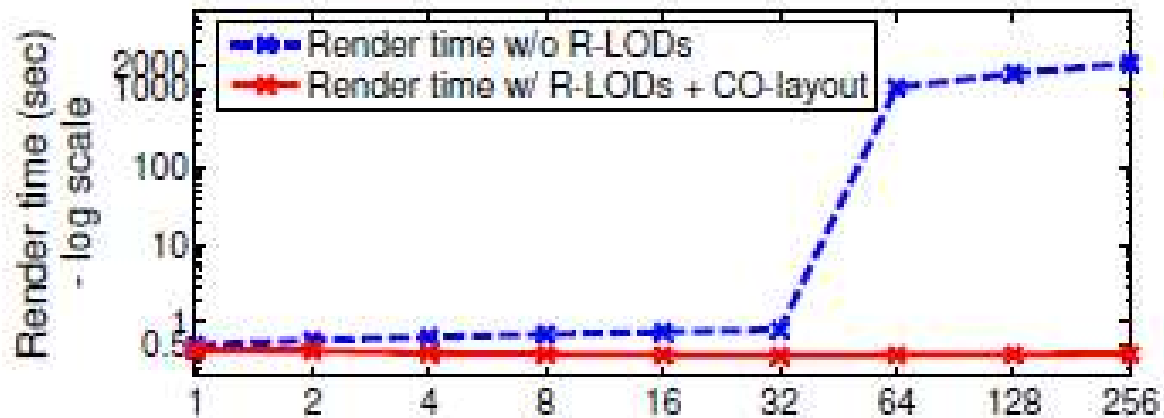
- **Prepare an overview slide**
 - **Talk about most important things and connect them well**

Be Honest

- **Do not skip important ideas that you don't know**
 - **Explain as much as you know and mention that you don't understand some parts**
- **If you get questions you don't know good answers, just say it**
- **In the end, you need to explain them before the semester ends**

Result Presentation

- Give full experiment settings and present data with the related information



- After showing the data, give a message that we can pull of the data
- Show images/videos, if there are

Prepare a Quiz

- **Give two simple questions to draw attentions**
 - **Ask a keyword**
 - **Simple true or false questions**
 - **Multiple choice questions**
- **Grade them in the scale of 0 and 10, and send the score to TA**

Audience feedback form

Date:

Talk title:

Speaker:

A. Was the talk well organized and well prepared?

5: Excellent 4: good 3: okay 2: less than average 1: poor

B. Was the talk comprehensible? How well were important concepts covered?

5: Excellent 4: good 3: okay 2: less than average 1: poor

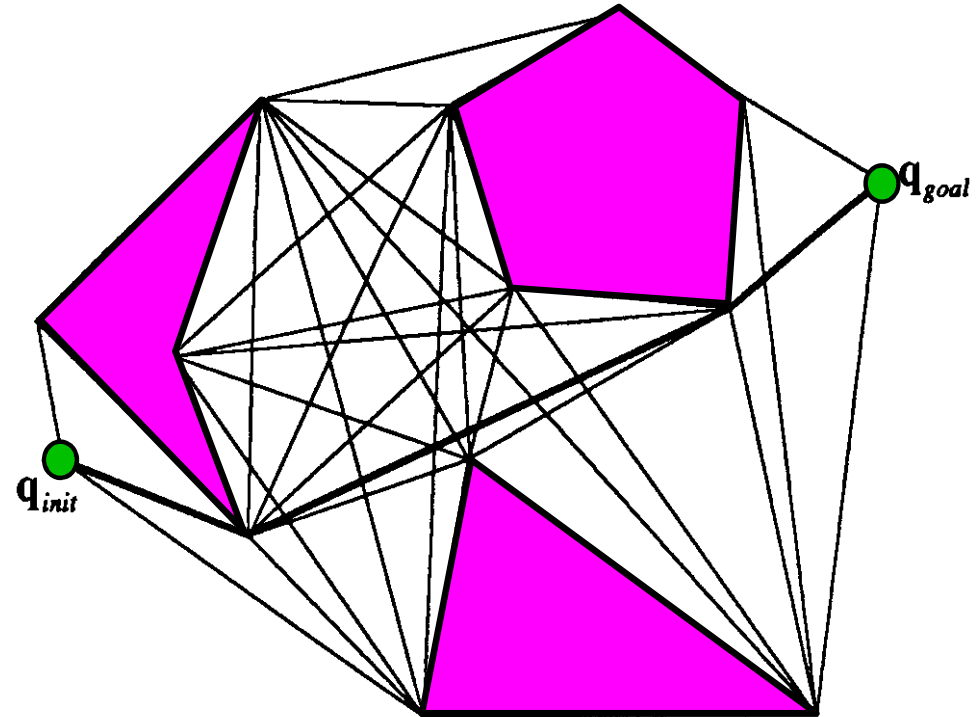
Any comments to the speaker

Class Objectives

- **Understand collision detection and distance computation**
 - **Bounding volume hierarchies**
- **Handle point clouds**

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?

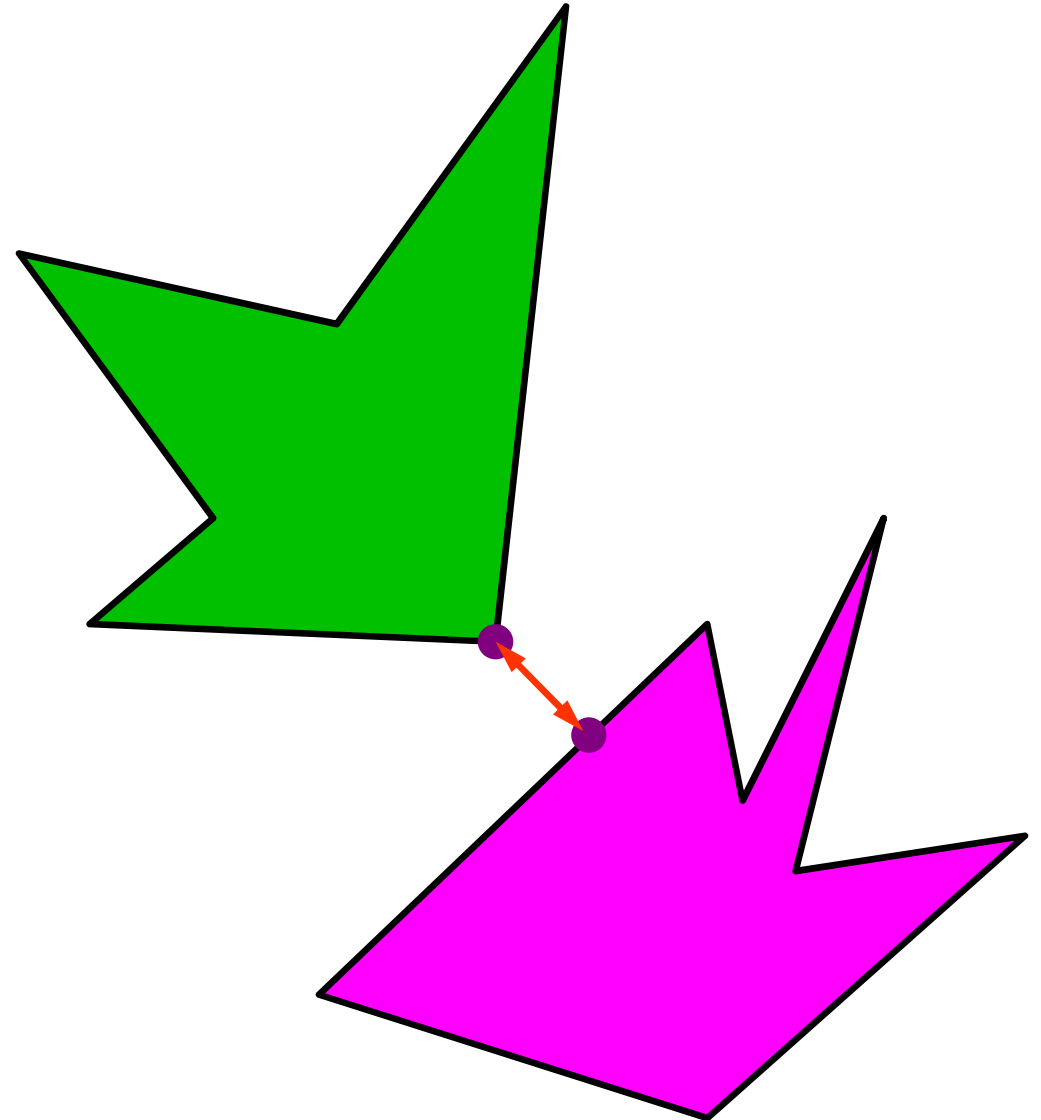


Problem

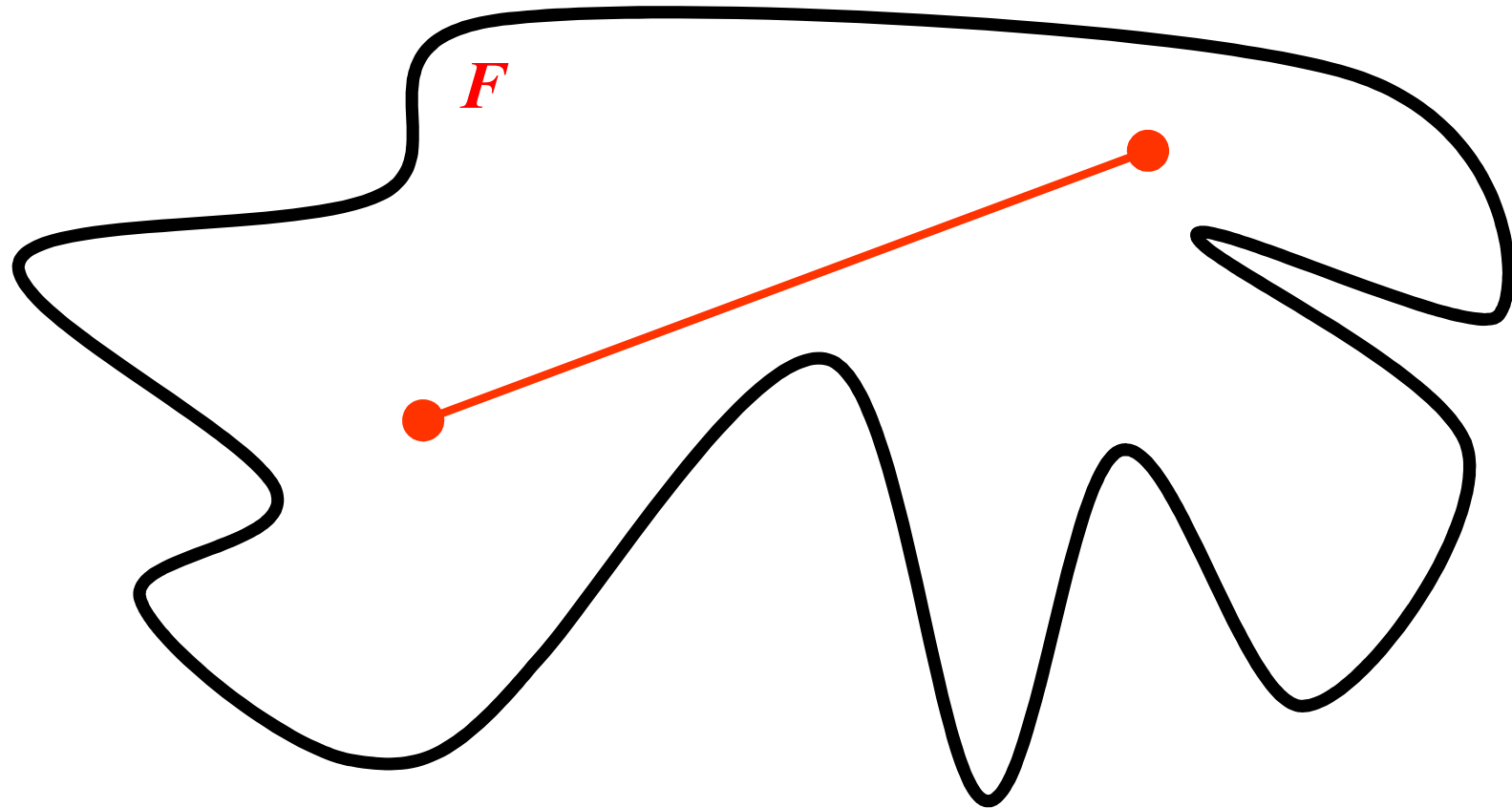
- **Input: two objects A and B**
 - **Output:**
 - **Distance computation: compute the distance (in the **workspace**) between A and B**
- OR**
- **Collision detection: determine whether A and B collide or not**

Collision detection vs. distance computation

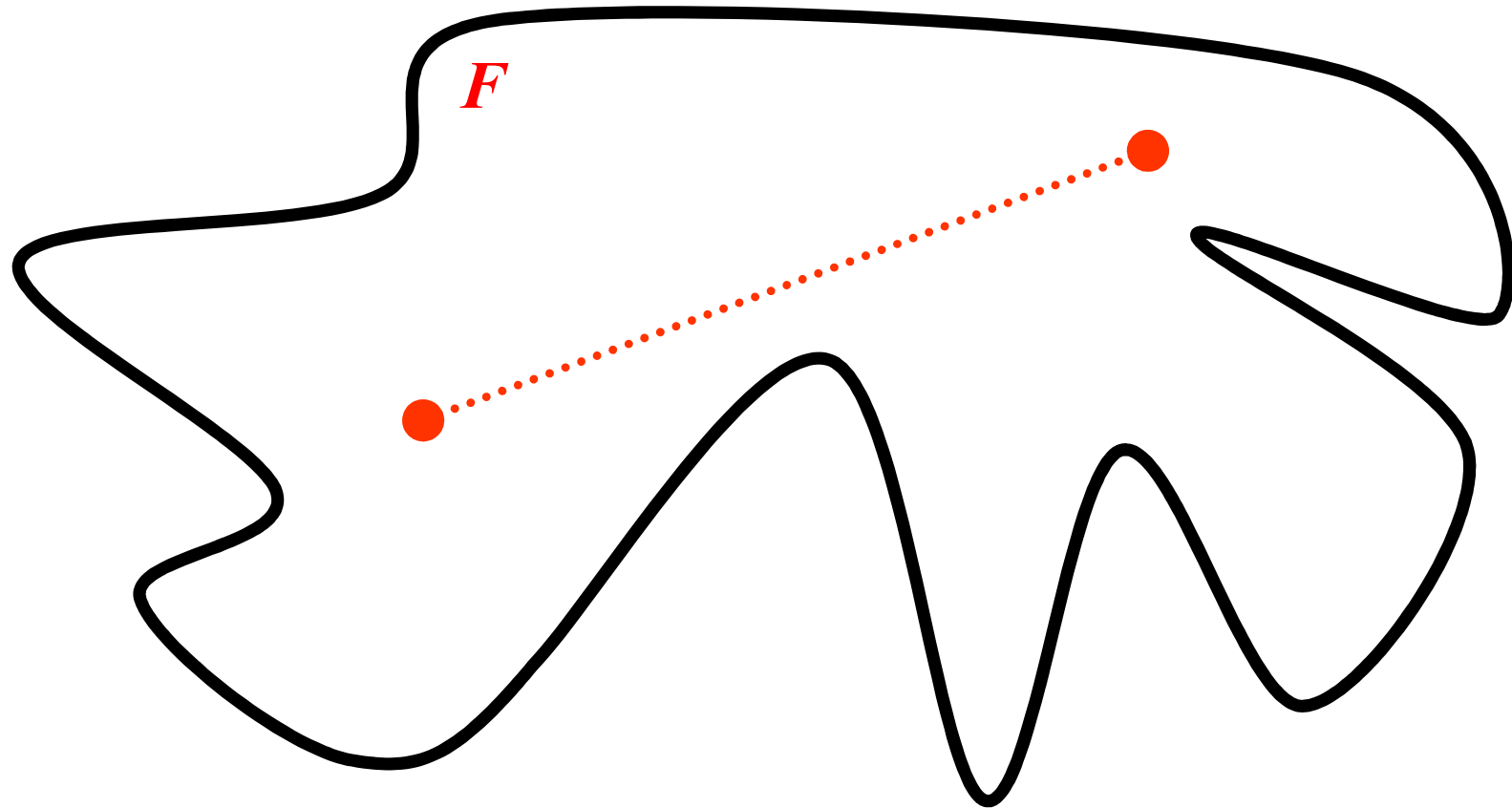
- **The distance between two objects (in the workspace) is the distance between the two closest points on the respective objects**
- **Collision if and only if distance = 0**



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

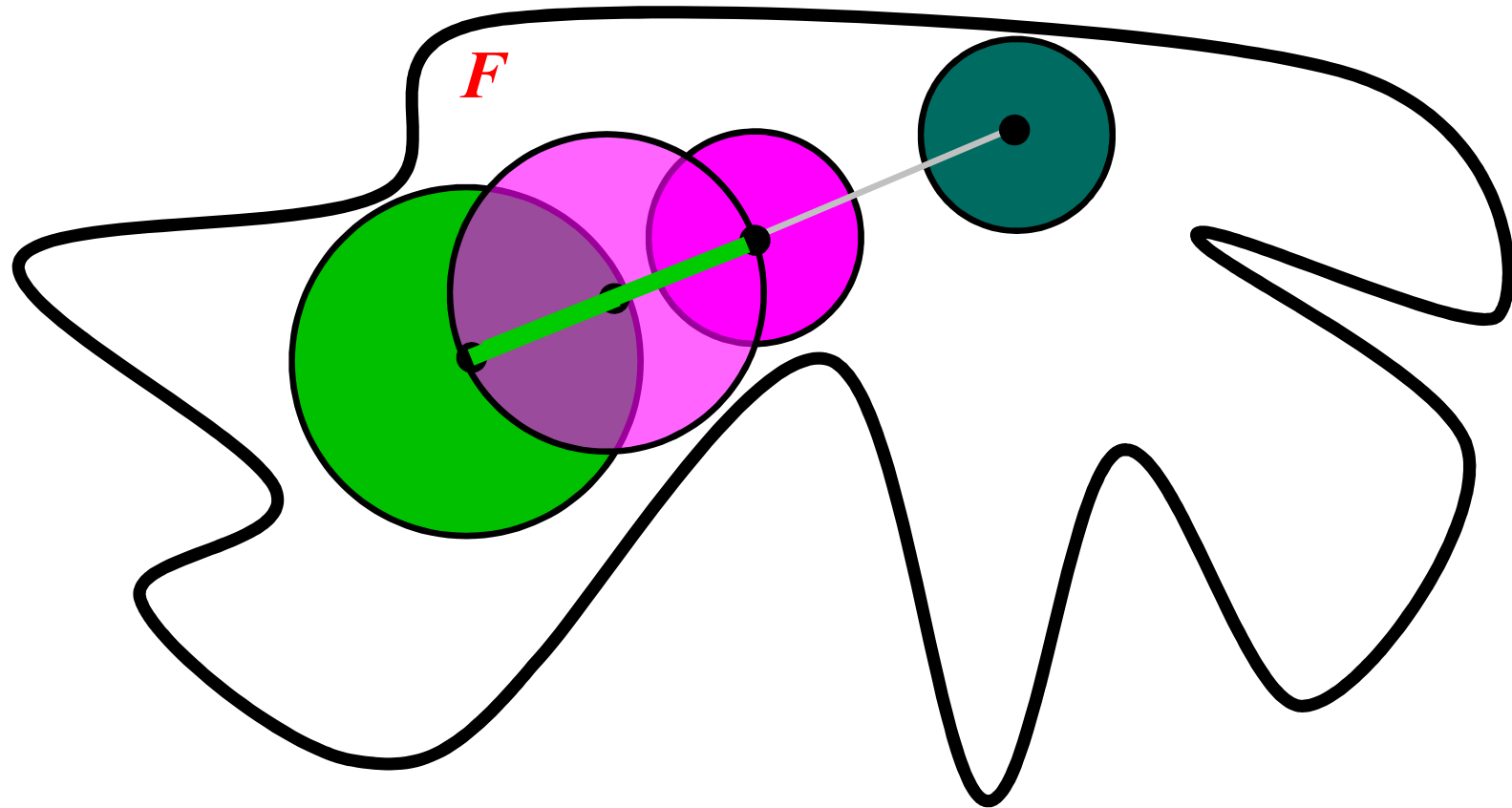


Collision detection does not allow us to check for free path rigorously



Discrete collision checks

Use distance to check for free path rigorously



Use distance to check for free path rigorously

Link(q_0 , q_1)

1: if $q_0 \in N(q_1)$ or $q_1 \in N(q_0)$

2: then

3: return TRUE.

4: else

5: $q' = (q_0 + q_1) / 2$.

6: if q' is in collision

7: then

8: return FALSE

9: else

10: return Link(q_0 , q') && Link(q_1 , q')

Applications

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

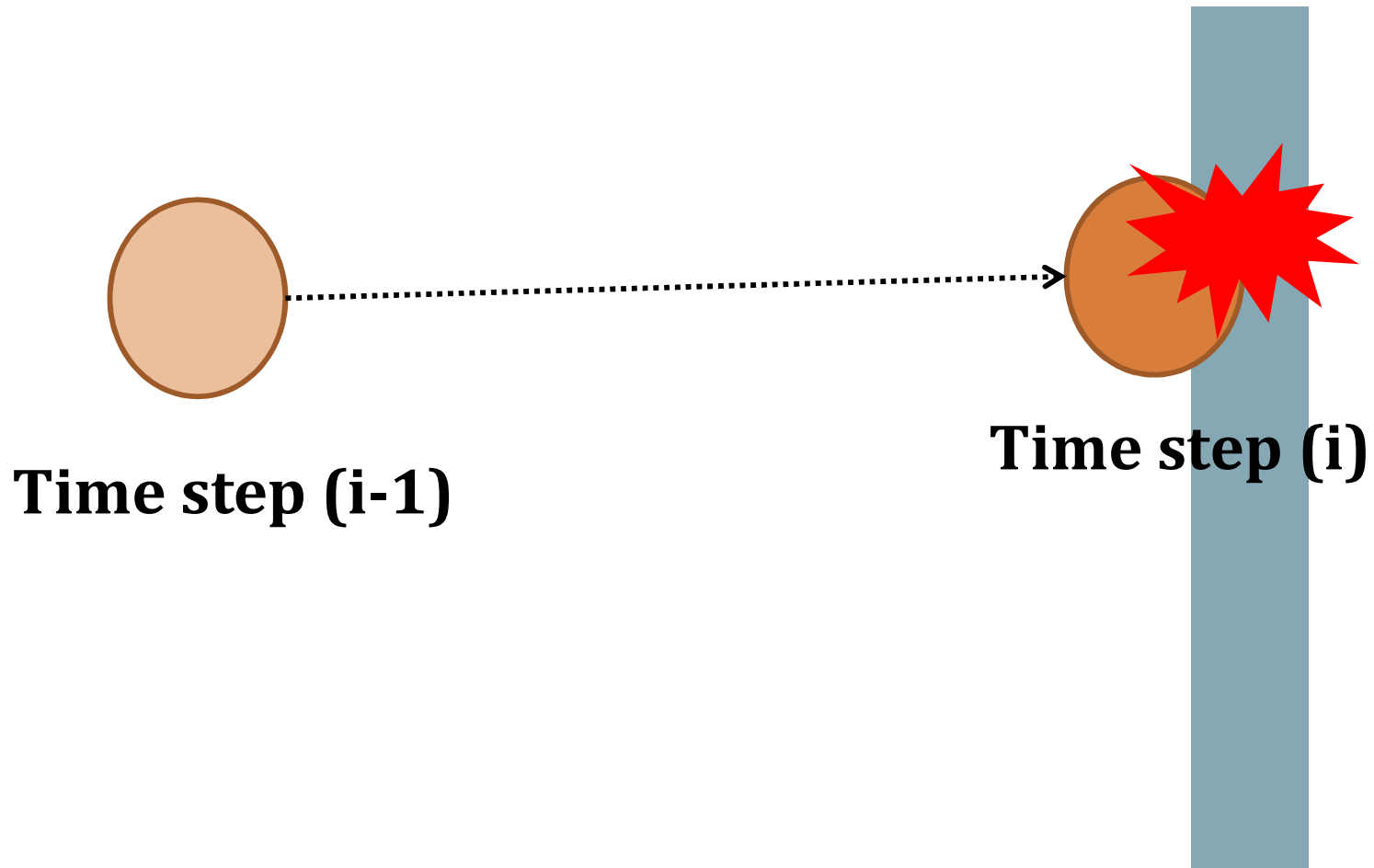


Collision Detection

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

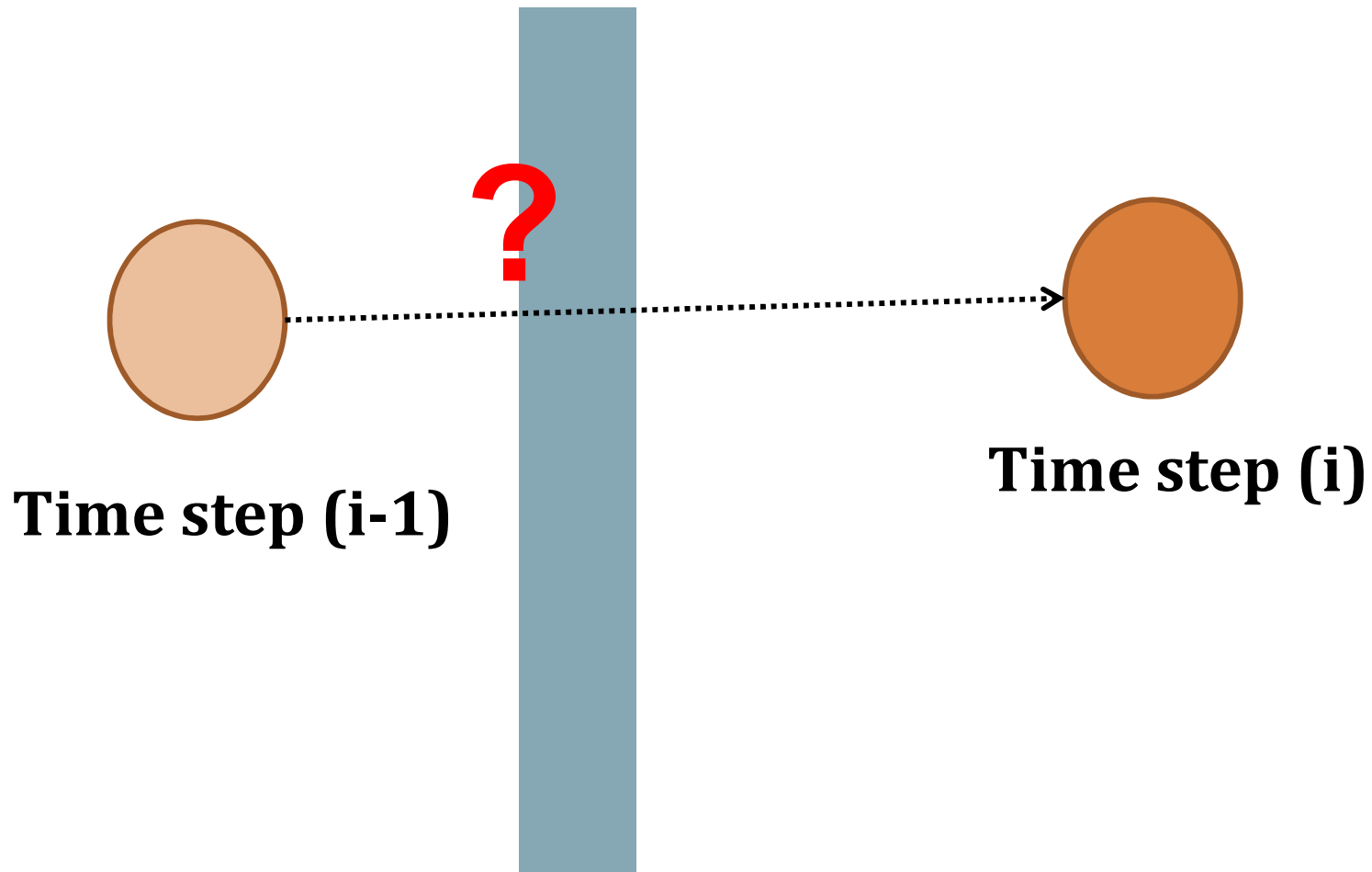
Discrete VS Continuous

Discrete collision detection (DCD)



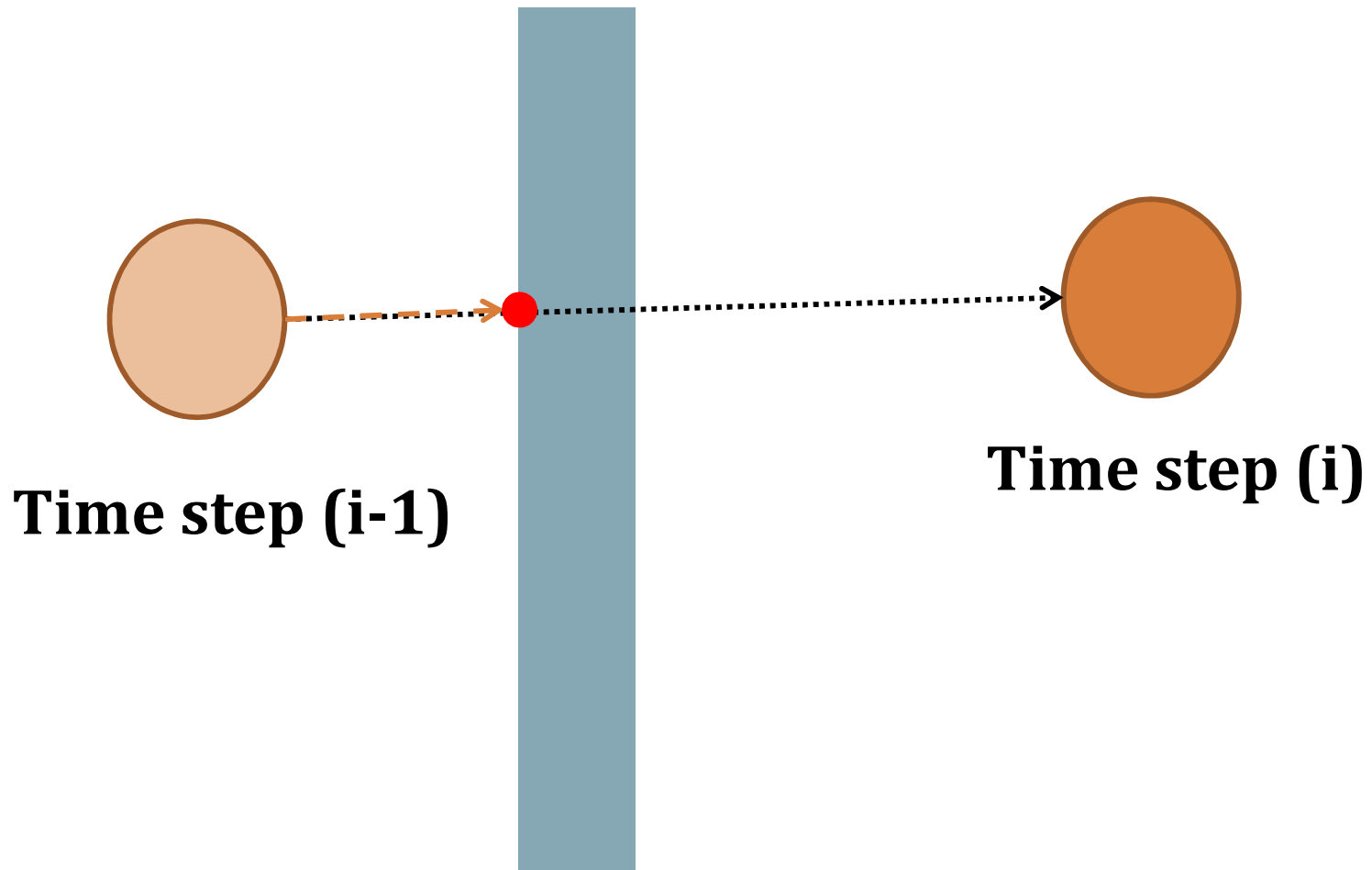
Discrete VS Continuous

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

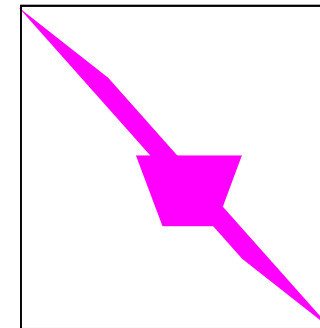
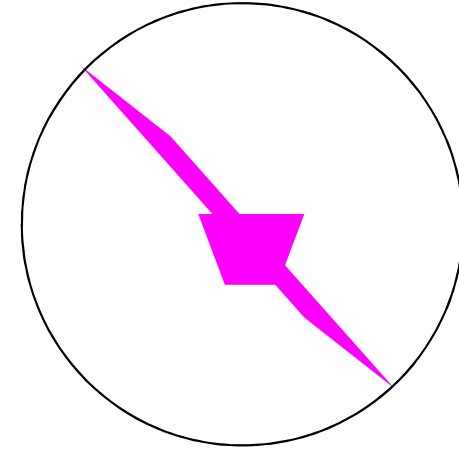
Collision Detection

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

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

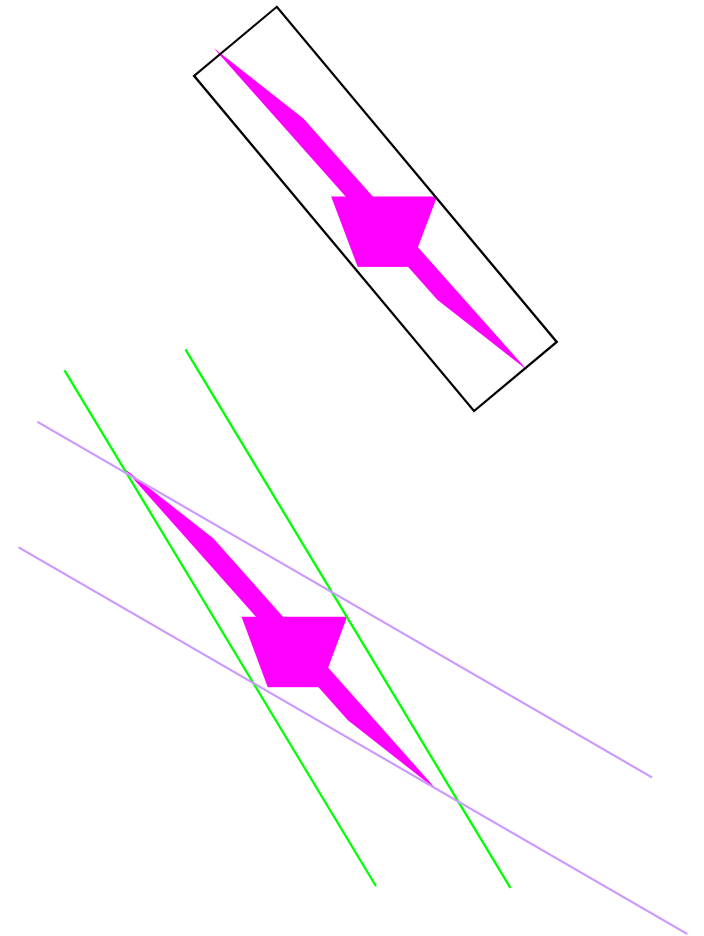
Bounding Volumes

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



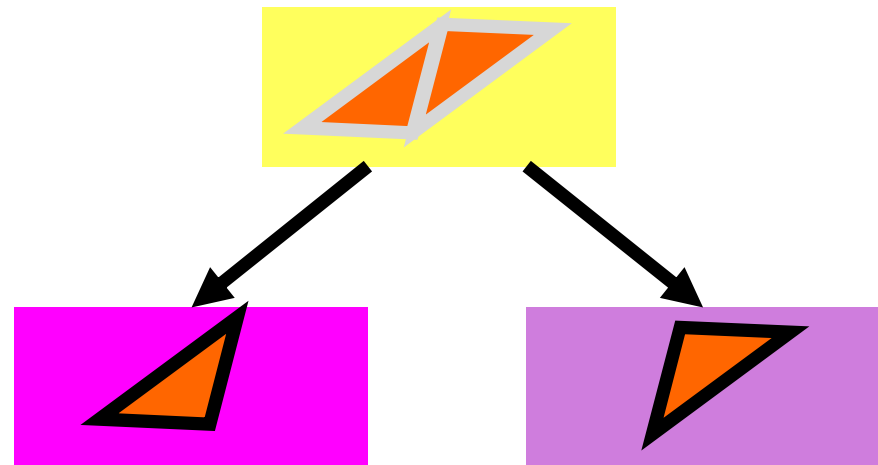
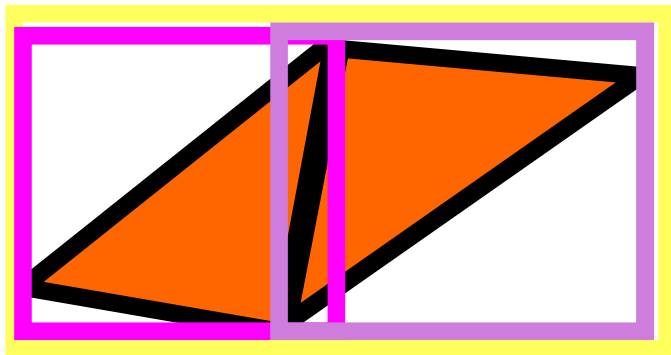
Bounding Volumes

- **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



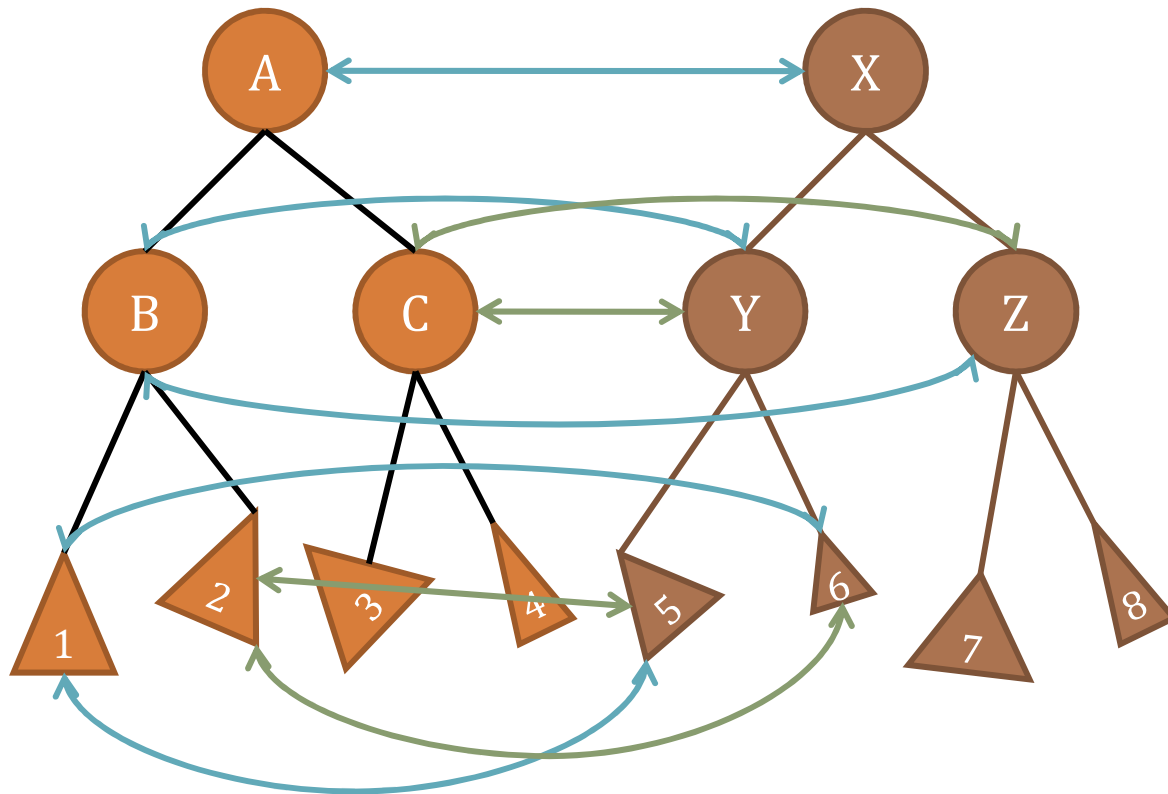
Bounding Volume Hierarchies (BVHs)

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

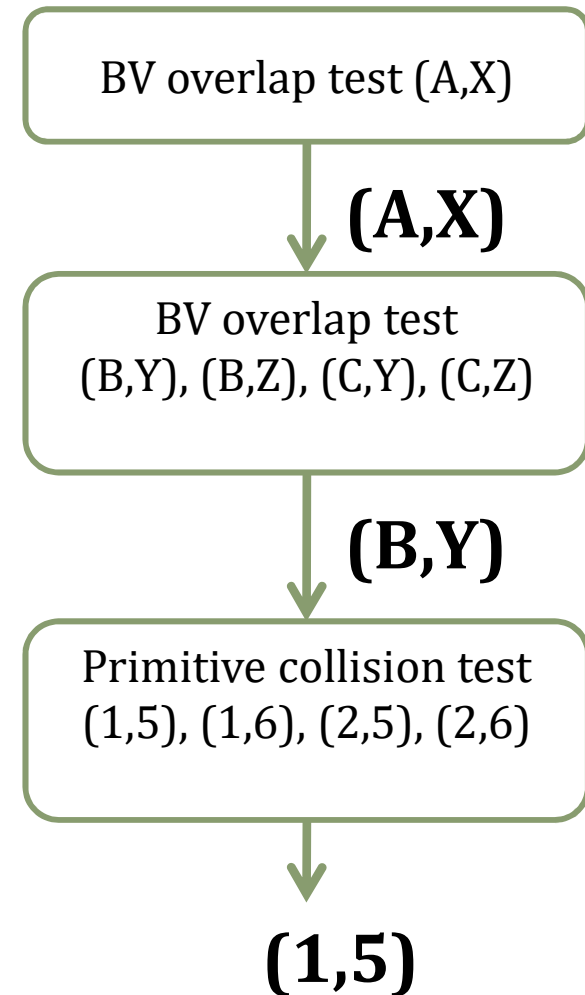


A BVH

Collision Detection with BVHs



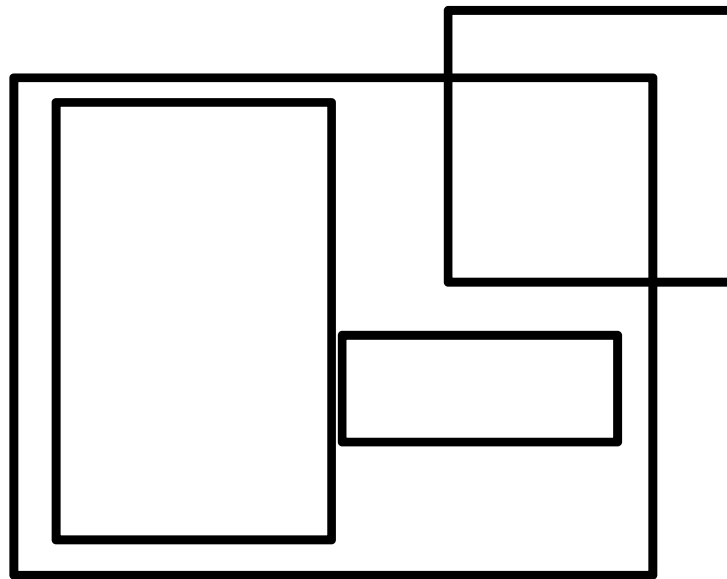
Triangle 1 and 5 have a collision!



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**

Test-Of-Time 2006 Award



RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs

Christian Lauterbach, **Sung-eui Yoon**, David Tuft, Dinesh Manocha

IEEE Interactive Ray Tracing, 2006

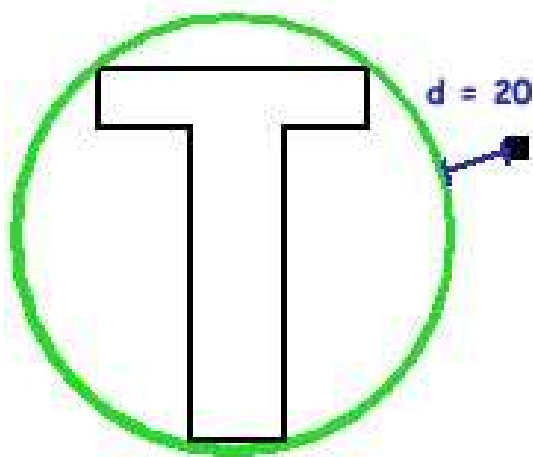


Computing distances

- **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

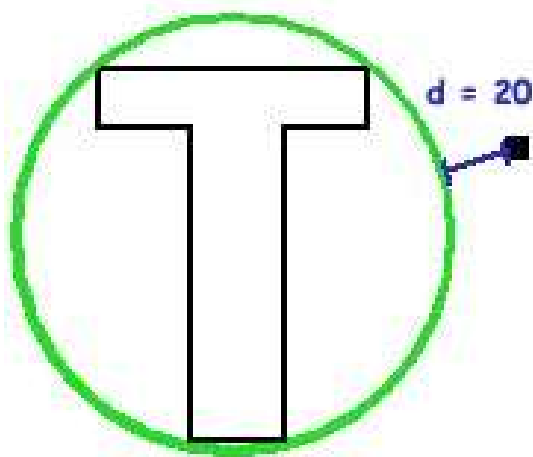
- **Set initial distance value to infinity**



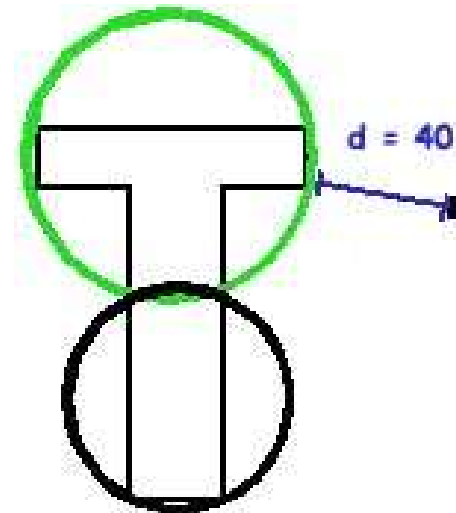
Start at the root node.
 $20 < \text{infinity}$, so continue
searching

Simple example

- **Set initial distance value to infinity**



Start at the root node.
 $20 < \text{infinity}$, so continue searching.

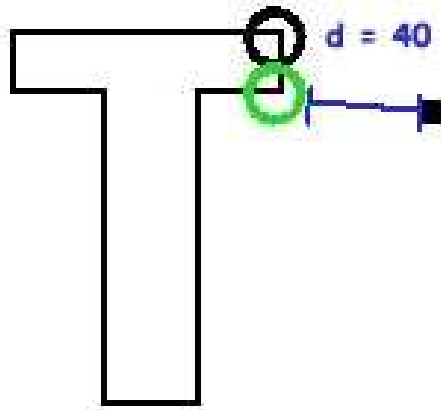


$40 < \text{infinity}$, so continue searching recursively.

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

Simple example

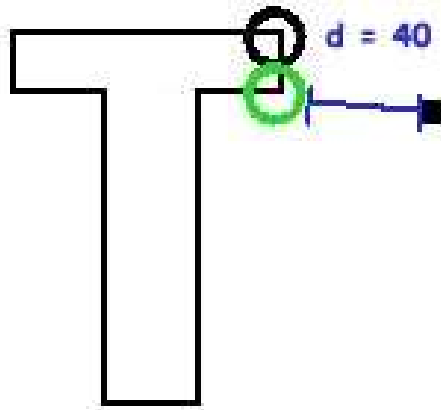
- Eventually reach a leaf node



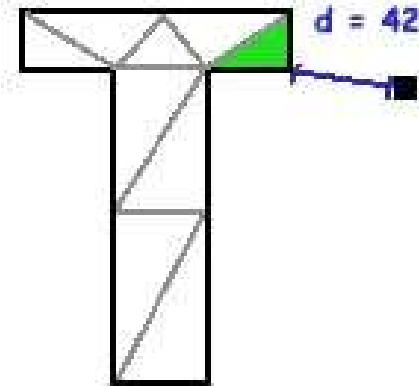
40 < infinity; examine the polygon to which the leaf node is attached.

Simple example

- Eventually reach a leaf node



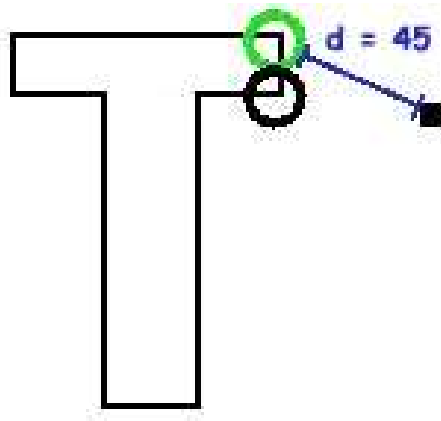
40 < infinity; examine the polygon to which the leaf node is attached.



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

Simple example

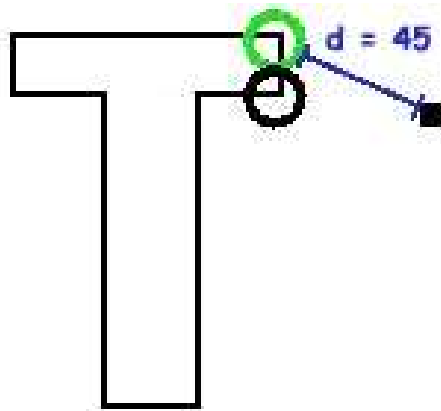
- Continue depth-first search



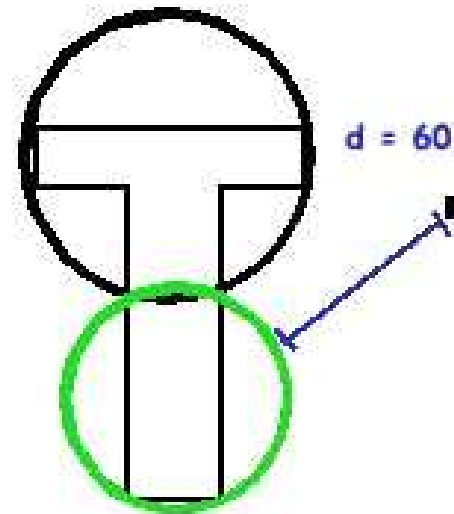
45 > 42; don't search this
branch any further

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**

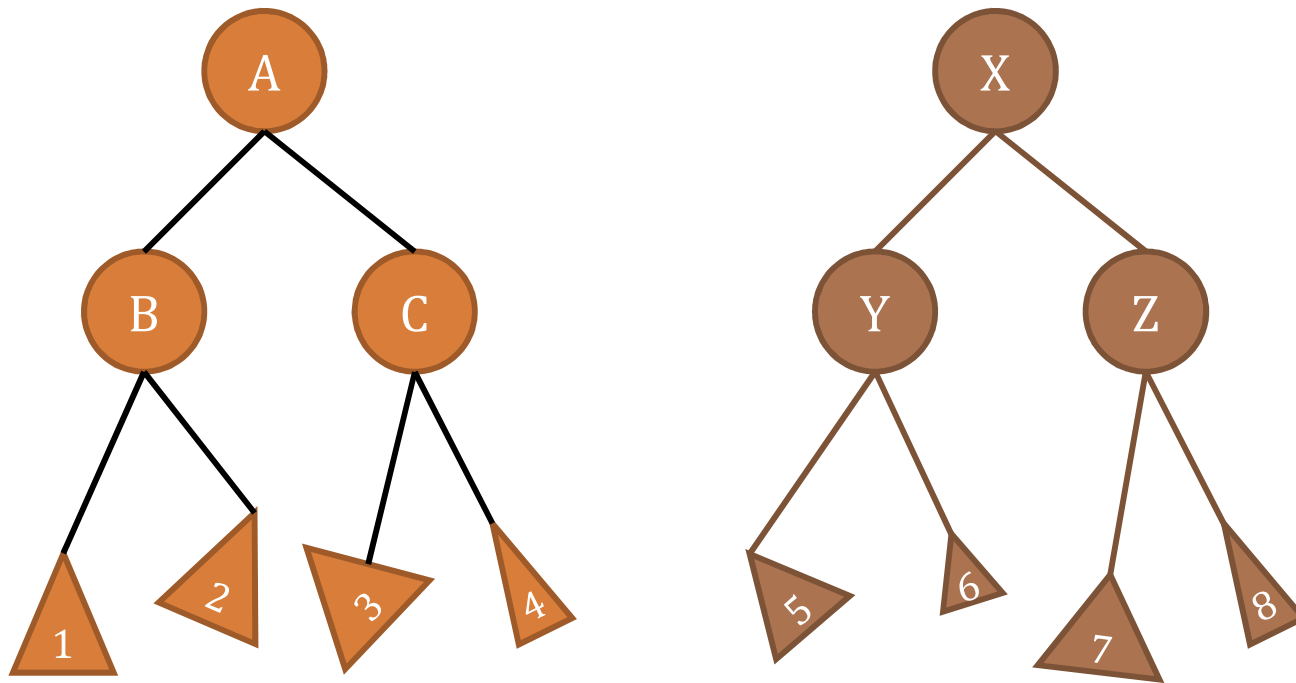
- $O(n)$ time to traverse the tree, where n = number of leaf nodes
- Plus time to compute distance to each polygon in the underlying model

- **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

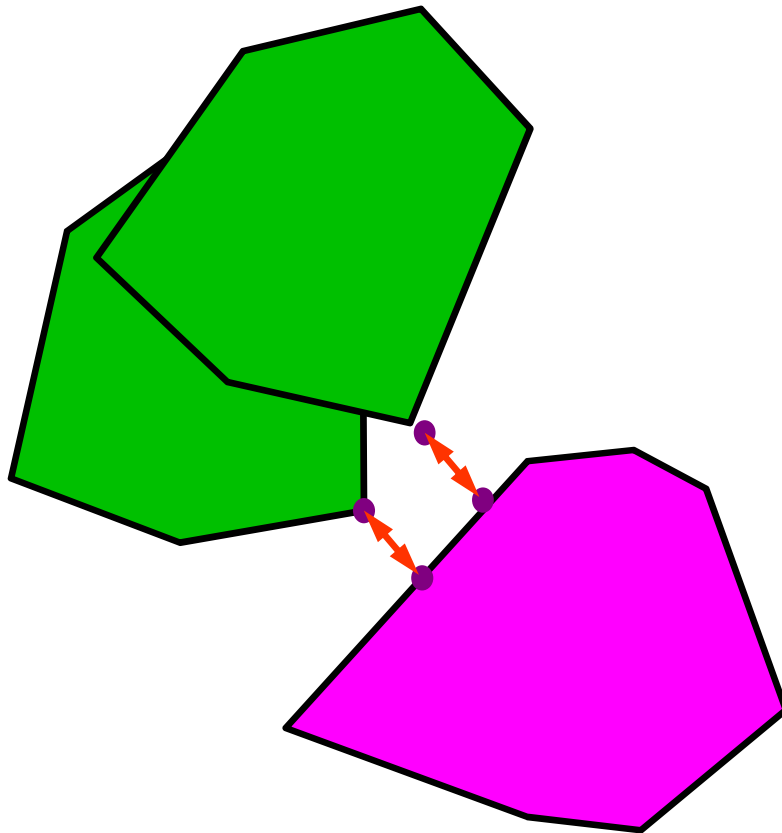
General case

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



Tracking the closest pair

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



Sensor-based Path Planning

- Navigation using 3D depth sensor

Real-Time Navigation in 3D Environments Based on Depth Camera Data

Daniel Maier Armin Hornung Maren Bennewitz

Humanoid Robots Laboratory, University of Freiburg

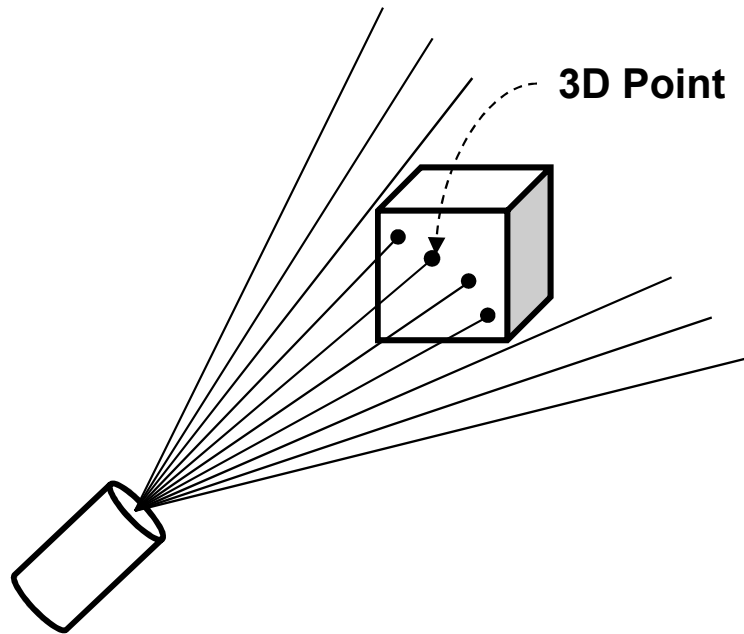


Maier, Daniel, et al.

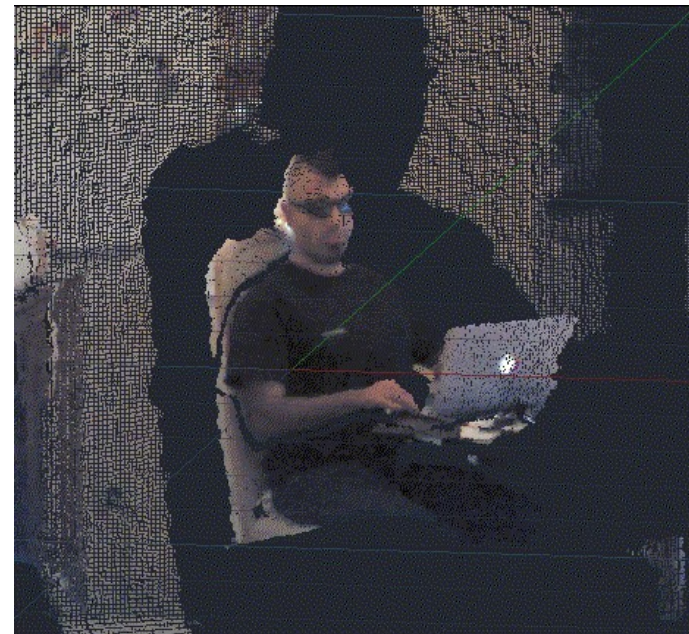
Modified from YongSun's slides

3D Sensor & Point Cloud Data

- **3D sensor generates excessive amount of points with some noise periodically**
 - **300K points / 30FPS with Kinect**

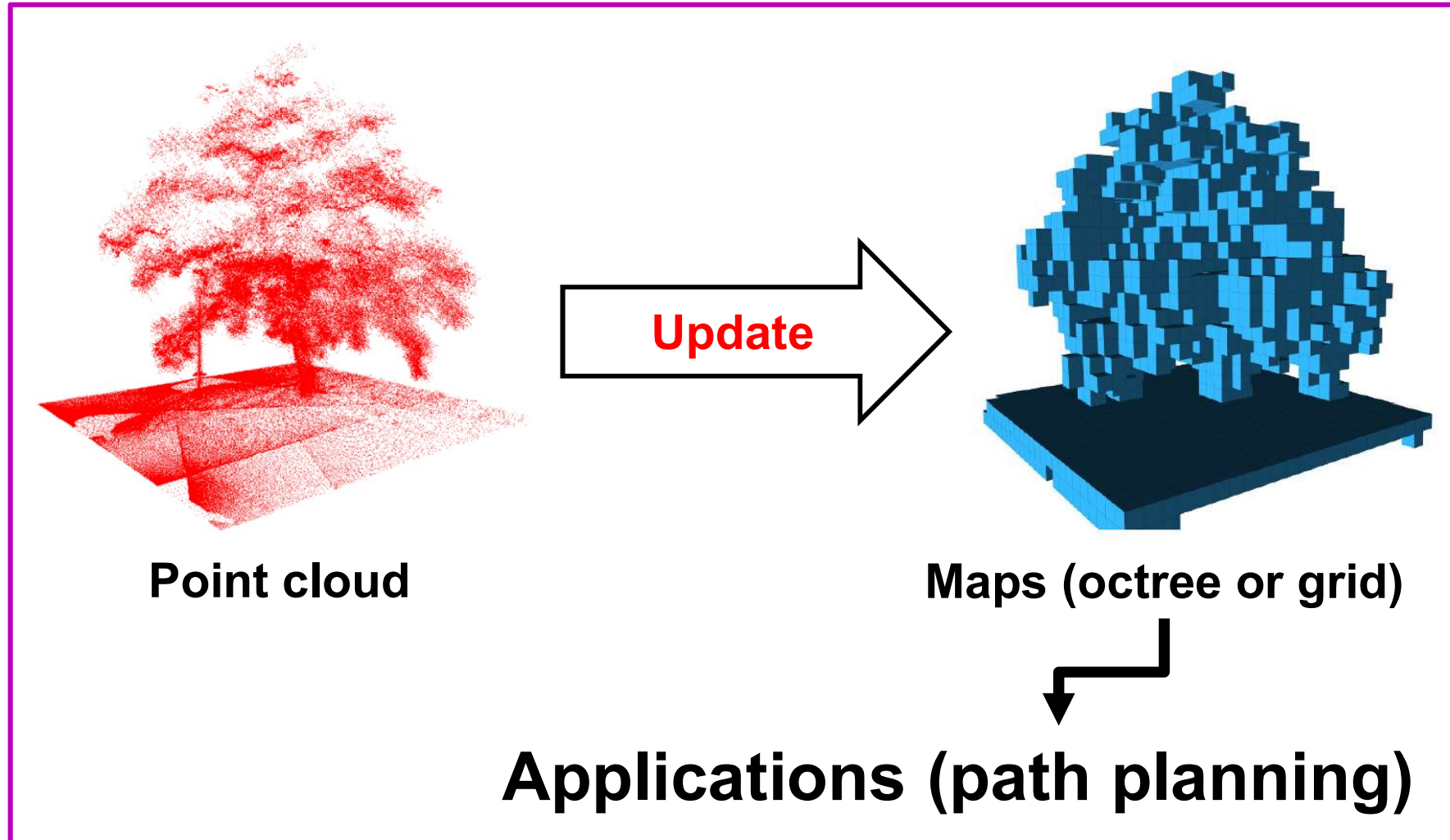


3D Sensor Model

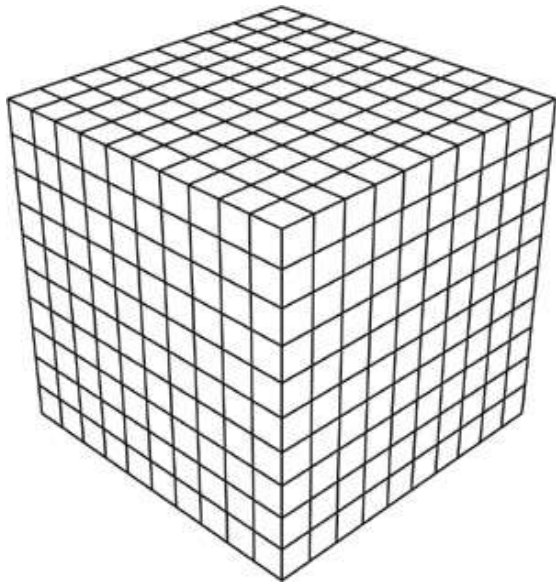


Point Cloud Data

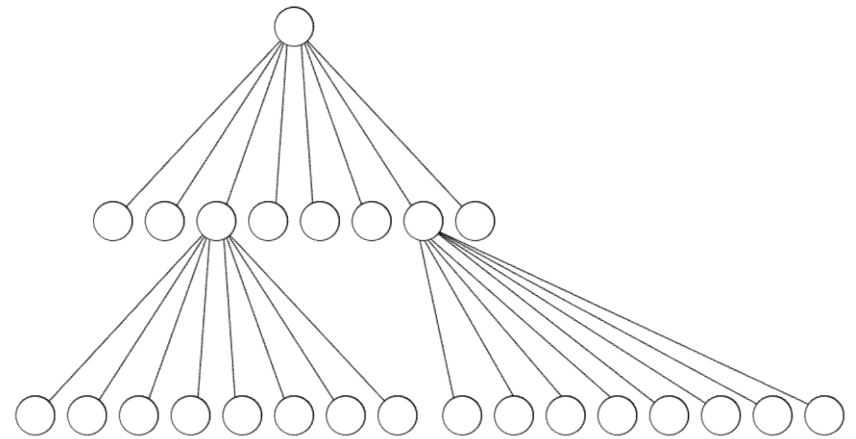
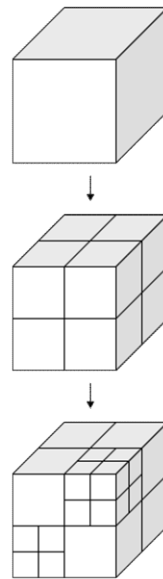
General Flow of Using Point Clouds



Map Representations



3D Grid Map



Octree Data Structure

Occupancy Map Representation

- **OctoMap [Wurm et al., *ICRA, 2010*]**
 - **Encode an occupancy probability of cell n given measurement $z_{1:t}$**

$$L(n | z_{1:t}) = L(n | z_{1:t-1}) + L(n | z_t)$$

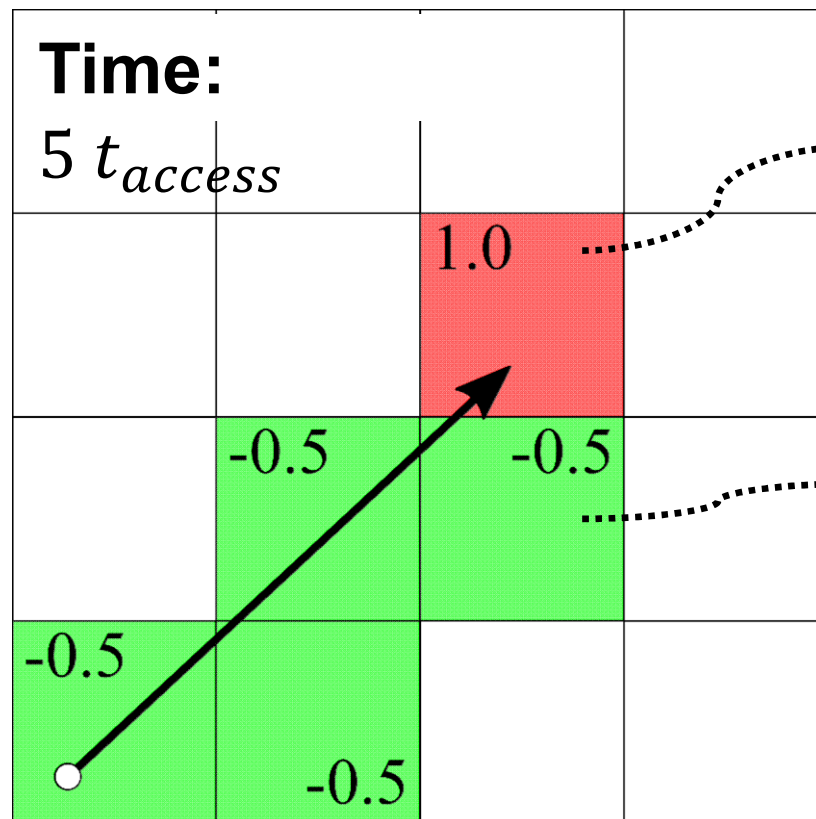
↓
Occupancy probability of the
cell n at time step $t - 1$

↓
New sensor measurement z_t
to be updated at time step t

$$L(n | z_t) = \begin{cases} l_{occ} & \text{occupied state} \\ l_{free} & \text{free state} \end{cases}$$

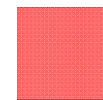
Update Method

- Traverse and update cells
 - Bresenham algorithm [Amanatides et al., *Eurographics, 1987*]



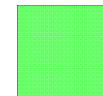
Updated cell to occupied state

$$L(n | z_t) = l_{occ} = 1.0$$



Updated cell to free state

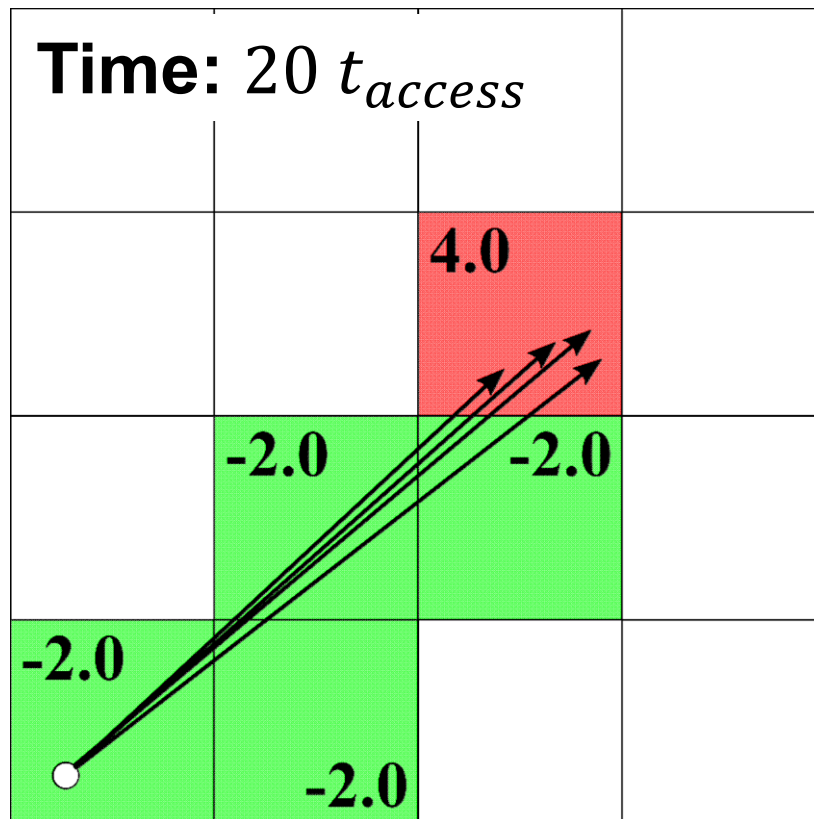
$$L(n | z_t) = l_{free} = -0.5$$



t_{access} : time to update a cell

Update Method

- Traverse and update cells
 - Bresenham algorithm [Amanatides et al., *Eurographics, 1987*]
 - Can be very slow, with many points



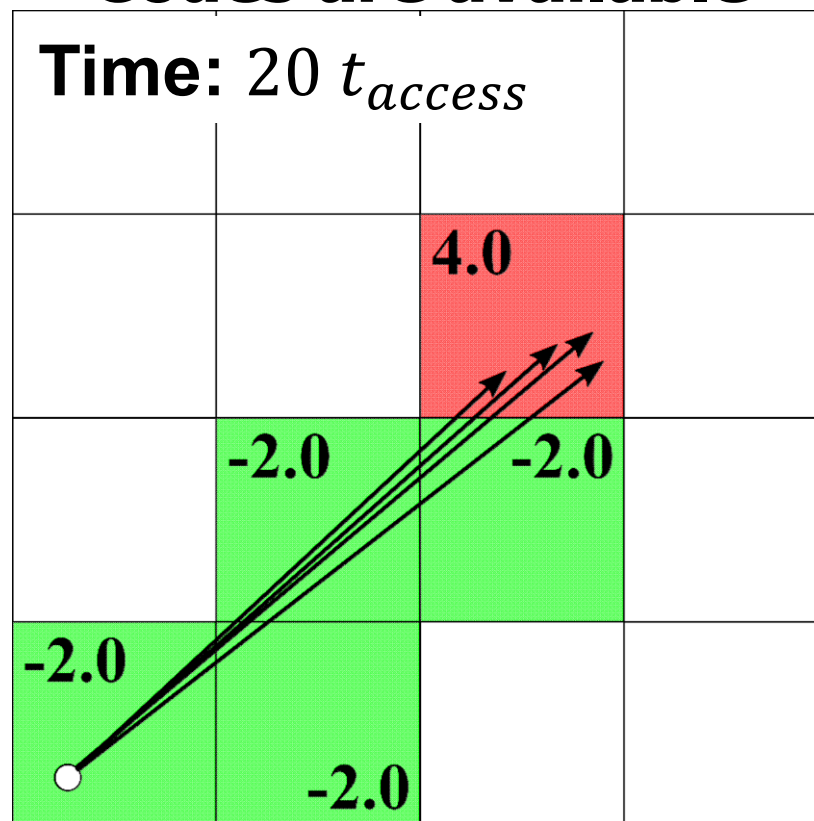
- Visit **the same cells multiple times** for multiple rays

t_{access} : time to update a cell

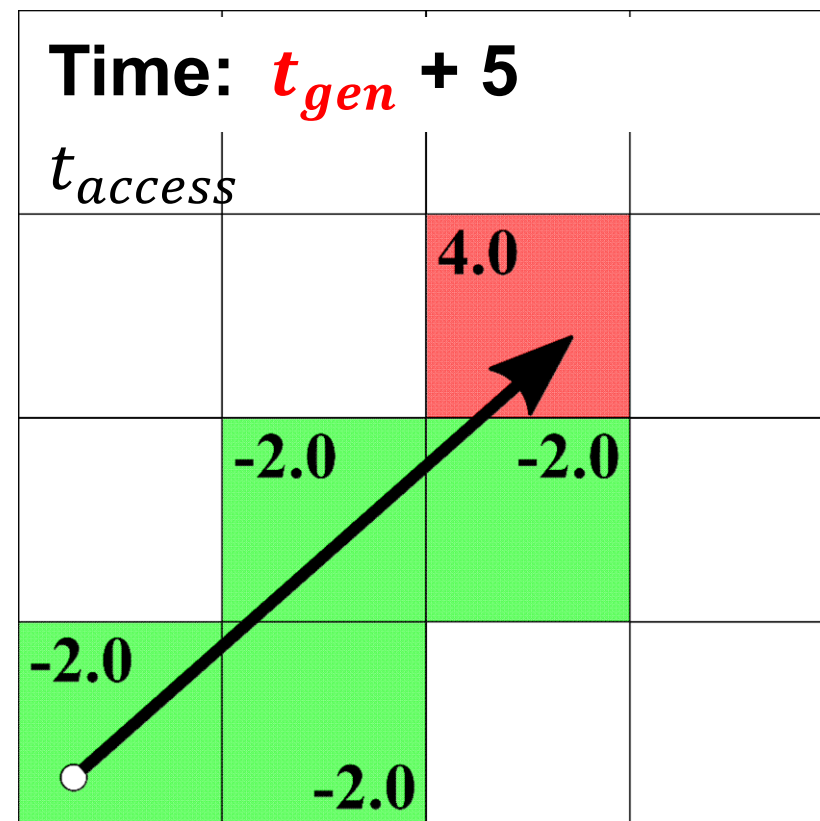
Super Rays [Kwon et al., ICRA16]

- **Benefits of our approach**

- **Faster performance with the same representation accuracy**
- **Codes are available**



State-of-the-art method



Ours

Class Objectives were:

- **Understand collision detection and distance computation**
 - **Bounding volume hierarchies**
- **Handle point clouds**

Next Time...

- **Probabilistic Roadmaps**

Homework

- **Submit summaries of 2 ICRA/IROS/RSS/CoRL/TRO/IJRR papers**
- **Go over the next lecture slides**
- **Come up with 3 questions before the mid-term exam**