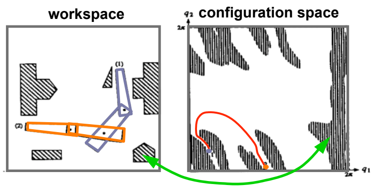
Last lecture

- Configuration space
  Convert moving objects into points, and apply algorithms for point robots.



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?

Collision detection & distance computation

CLEAR(q)

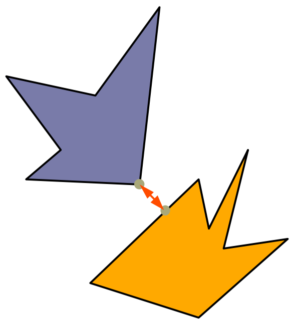
- **Input**: two objects A and B
- **Output**:
  - Distance computation: compute the distance (in the **workspace**) between A and B

    OR

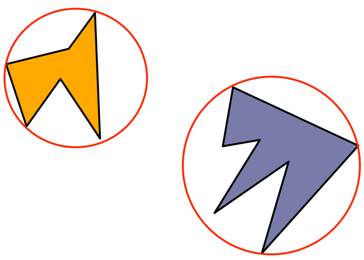
  - Collision detection: determine whether A and B collide or not

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



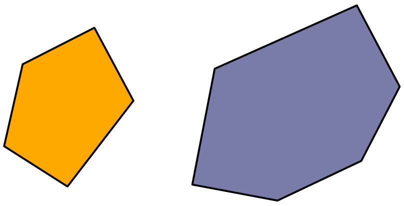
Collision detection may be easier than distance computation



Applications

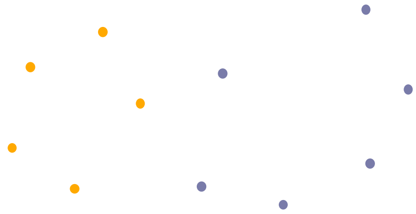
- Robotics
  - Collision avoidance
  - Path planning
- Graphics & virtual environment simulation
- Haptics
  - Collision detection
  - Force proportional to distance

How will you compute the distance?



What is the distance between two convex polygons?

How will you compute the distance?



What is the distance between two sets of points?

Two approaches

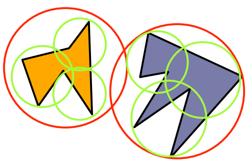
- **CLEAR**(q)
  - **Hierarchical bounding approximation** of objects
    - Spheres
    - Boxes
    - …
  - Tracking closest pairs of features

Spherical bounding hierarchy

- *Efficient Distance Computation Between Non-Convex Objects*, S. Quinlan, 1994

Overview

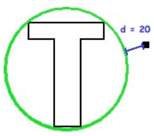
- Create a hierarchy of bounding spheres (bounding sphere tree) to approximate the object



- Recursive depth-first search of the tree to find the minimum distance
- Only search down the tree to required granularity

Simple example

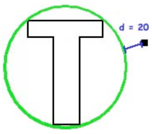
- Set initial distance value to infinity



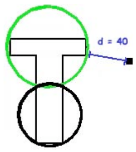
Start at the root node.
20 < infinity, so continue searching.

Simple example

- Set initial distance value to infinity



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

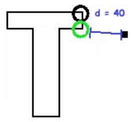


40 < infinity, so continue searching recursively.

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

Simple example

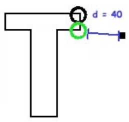
- Eventually reach a leaf node



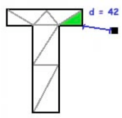
40 < infinity; examine the polygon to which the leaf node is attached.

Simple example

- Eventually reach a leaf node



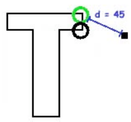
40 < infinity; examine the polygon to which the leaf node is attached.



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

Simple example

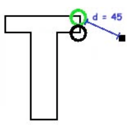
- Continue depth-first search



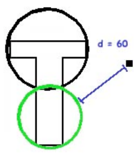
45 > 42; don't search this branch any further

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

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 do not reduce minimum distance
- Once leaf node is reached, examine underlying convex polygon for exact distance

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 and compare 2 trees
  - Start at root of both trees
  - Compare spheres; split the larger sphere
  - First continue the search comparing the unsplit node from the first tree and the closest child node from the other tree. Then compare the unsplit node and the other child.

Extension: relative error

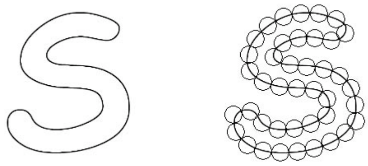
- When updating the minimum distance d' between objects, set $d' = (1-a)d$ (d = actual distance).
  - a is our relative error, why?
  - Guarantee that objects are at least d' apart

$$d_{\min} \ge d' \Rightarrow d_{\min} \ge (1-a)d \Rightarrow (d - d_{\min})/d \le a$$

  - $(1\text{-}a)d = 0$ iff $d = 0$; correctly detects collisions
- Improves performance by pruning search

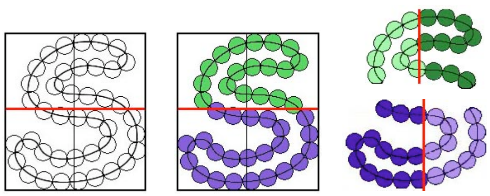
Creating the sphere tree

1. Cover the object surface with tiny spheres (leaf nodes). Radius is user-determined.



Creating the sphere tree

2. Find a rectangular bounding box.
3. Divide the box's major axis in half.
4. Recurse until each set contains only a single leaf node.



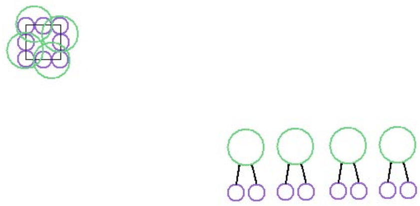
Creating the sphere tree

5. Build the tree from bottom up, creating bounding spheres for each node.

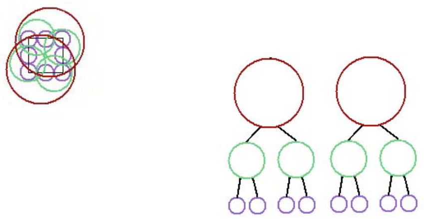
   Two methods:
   - Find the minimal sphere that contains the two spheres of the child nodes.
   - Determine a sphere directly from the leaf nodes descended from this node.

Example

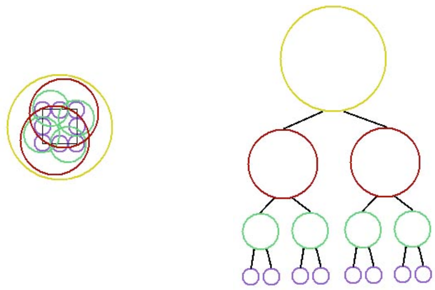
Example



Example



Example



Sphere tree

- Binary tree
  - Root node is the object’s bounding sphere.
  - Leaf nodes are tiny spheres; their union approximates the object’s surface.
- Every node’s sphere contains the spheres of its descendant nodes.

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.
- Worst case time $O(n^2)$
  - If tree is extremely unbalanced
- Tree is built only once and can often be pre-computed.

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

Implementation tricks

- Store polygon comparisons in a hash table to avoid repeat calculations
- For path planning, make the robot one object and the union of all obstacles a single, second object

Key features

- It works for both convex and non-convex objects in 2-D or 3-D environments.
- It computes the exact or approximate distance.
- It uses hierarchical approximation to achieve efficiency.

Simplifying assumptions

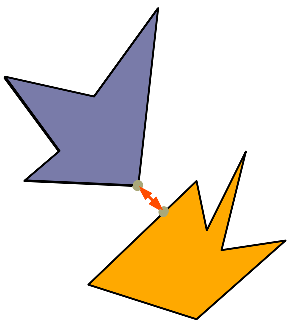
- Surface analysis only
- Decomposition of objects into sets of convex surfaces
  - Easy in graphics; all surfaces are composed of triangles
- Existence of efficient algorithm to determine distance between 2 convex polygons

Summary

- Simple and intuitive way to speed up distance calculations using hierarchical bounding approximation of objects
  - Spheres
  - Boxes
- Other related work
  - OBB-Tree: A hierarchical structure for rapid interference detection. S. Gottschalk, M. Lin, and D. Manocha. In *SIGGRAPH 96 Conference Proceedings*, pp. 171-180, 1996.
- Software libraries (http://www.cs.unc.edu/~geom/collide/packages.shtml)
  - PQP

Two approaches

- **CLEAR**(q)
  - Hierarchical bounding approximation of objects
    - Spheres
    - Boxes
    - …
  - **Tracking closest pairs of features**



Tracking the closest pair

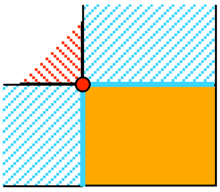
- *V-Clip: Fast and Robust Polyhedral Collision Detection,* B. Mirtich, 1997

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.

Voronoi regions of points and edges

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



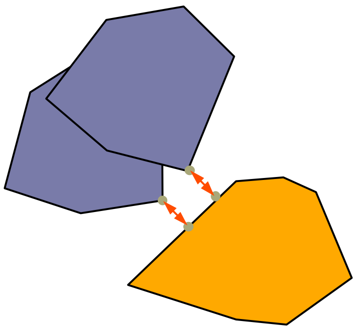
Critical condition

- 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.

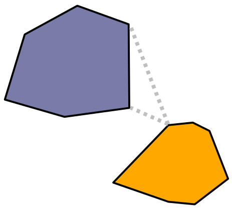
Sketch of the algorithm

```
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).
```


Motion coherence



Iterative improvement

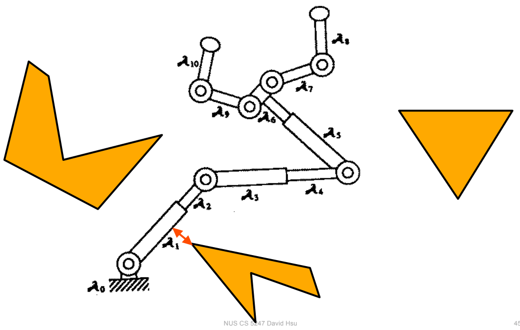


- For **convex** objects, an iterative step always results in a decrease in the candidate "feature" pair.

Key features

- It works for convex objects in 2-D or 3-D environments.
- It computes the exact distance.
- It uses motion coherence to achieve efficiency.

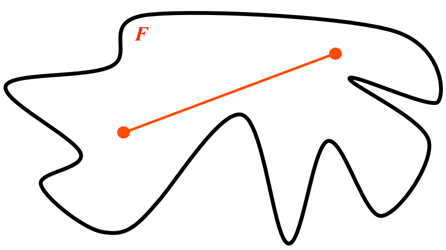
Articulated robot



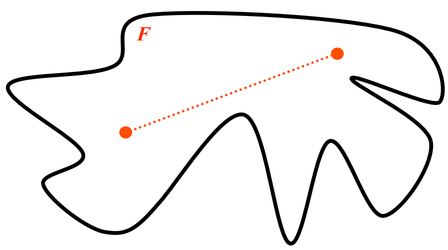
Sketch of the algorithm

```
Set dmin to ∞.
for every pair (A,B) of robot link A and
workspace obstacle B
  Compute the distance d between A and B
  If d = 0 then return collision
  If d < dmin then set dmin to d
return dmin
```

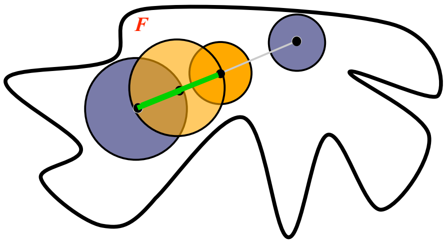
Collision detection does not allow us to test for free path rigorously



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



Use distance to check for free path rigorously



Use distance to check for free path rigorously

```
Link(q0, q1)
1: if q0∈N(q1) or q1∈N(q0)
2: then
3:   return TRUE.
4: else
5:   q' = (q0+q1)/2.
6:   if q' is in collision
7:   then
8:     return FALSE
9:   else
10:    return Link(q0, q') && Link(q1,q').
```