Collision Detection

Many Different Situations
Few moving objects, but complex geometry

Many Different Situations
Thin moving objects, with simple geometry

Many Different Situations
Many simple objects

Many Different Situations
Many simple objects

Many Different Situations
Many contacts, deformable objects, ...
Many Different Situations
Real-time detection

Collision Detection Methods
- Many different methods
- We will focus on two of them:
  - Grid method: good for many simple moving objects of about the same size (e.g., many moving discs with similar radii)
  - Bounding Volume Hierarchy (BVH) method: good for few moving objects with complex geometry

Grid Method
- Subdivide space into a regular grid of cubic or square bins
- Index each object in a bin
- Running time is proportional to number of moving objects

Bounding Volume Hierarchy Method
- Enclose objects into bounding volumes (spheres or boxes)
- Check the bounding volumes first
- Decompose an object into two
Bounding Volume Hierarchy Method
- Enclose objects into bounding volumes (spheres or boxes)
- Check the bounding volumes first
- Decompose an object into two
- Proceed hierarchically

Collision Detection
Two objects described by their precomputed BVHs

BVH in 3D

Collision Detection
Search tree
pruning
Collision Detection

Search tree

If the pieces contained in G and D overlap → collision

Search Strategy

- If there is no collision, all paths must eventually be followed down to pruning or a leaf node
- But if there is collision, it is desirable to detect it as quickly as possible
- Greedy best-first search strategy with $f(N) = d/(r_X + r_Y)$
  [Expand the node XY with largest relative overlap (most likely to contain a collision)]

Performance

Several thousand collision checks per second for 2 three-dimensional objects each described by 500,000 triangles, on a 1-GHz PC

Faster when objects are well separated or have much overlap. Slower when objects barely overlap or are very close. Why?
Bad Case for BVH

Desirable Properties of BVs and BVHs

**BVHs:**
- Tightness
- Separation
- Efficient testing
- Invariance

**Spheres**
- Invariant
- Efficient to test
- But tight?

Axis-Aligned Bounding Box (AABB)
- Not invariant
- Efficient to test
- Not tight

---

**Desirable Properties of BVs and BVHs**

**BVs:**
- Tightness
- Efficient testing
- Invariance
Oriented Bounding Box (OBB)

- Invariant
- Less efficient to test
- Tight

Comparison of BVs

<table>
<thead>
<tr>
<th></th>
<th>Sphere</th>
<th>AABB</th>
<th>OBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tightness</td>
<td>+</td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td>Testing</td>
<td>+</td>
<td>+</td>
<td>o</td>
</tr>
<tr>
<td>Invariance</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

No type of BV is optimal for all situations

Desirable Properties of BVs and BVHs

BVs:
- Tightness
- Efficient testing
- Invariance

BVH:
- Separation
- Balanced tree

Construction of a BVH

- Top-down construction
- At each step, create the two children of a BV
- Example:
  - For OBB, split longest side at midpoint
Static vs. Dynamic Collision Detection

Usual Approach to Dynamic Checking (in PRM Planning)
1) Discretize path at some fine resolution \( \varepsilon \)
2) Test statically each intermediate configuration

\[ < \varepsilon \]

Examples of Difficult Dynamic Collision Checking

Adaptive Bisection

Ideas:
- a) Relate configuration changes to path lengths in workspace
- b) Use distance computation rather than pure collision checking
- c) Bisect adaptively

Greedy Distance Computation
(same recursion as collision detection)

Greedy-Distance\((A,B)\)  
1. If \( \text{dist}(A,B) > 0 \), then return \( \text{dist}(A,B) \)
2. If \( A \) and \( B \) are both leaves, then return distance between their contents
3. Switch \( A \) and \( B \) if \( A \) is a leaf, or if \( B \) is bigger and not a leaf
4. Set \( A_1 \) and \( A_2 \) to be \( A \)'s children
5. \( d_1 \leftarrow \text{Greedy-Distance}(A_1,B) \)
6. If \( d_1 > 0 \) then
   a. \( d_2 \leftarrow \text{Greedy-Distance}(A_2,B) \)
   b. If \( d_2 > 0 \) then return \( \text{Min}(d_1,d_2) \)
7. Return 0
What about deformable objects?