Motion and Manipulation

Collision Detection
Collision Detection

• Sampling-based path planning: check if a simple path between two neighboring configurations is collision-free

• Detect collision of objects that
  – move under the influence of physics laws
  – are controlled by a user

Collisions often require actions: compute deformation, change direction and magnitude of speed
Standard Approach

t := 0
while not ready do
  1. compute placements of objects at t + ∆t
  2. check for intersections of objects at these placements, and perform action if necessary
  3. t := t + ∆t
Collision or Interference Checking

t := 0

while not ready do

1. compute placements of objects at t + \( \Delta t \)
2. check for intersections of objects at these placements, and perform action if necessary
3. t := t + \( \Delta t \)

Choice of \( \Delta t \) is crucial!
Time Step

- $\Delta t$ too large: danger of undetected collisions

- $\Delta t$ too small: excessive computation time
Expensive Alternative

Use sweep volumes, or add time dimension

Disadvantages
- complex shapes, expensive computation
- motion must be known beforehand
Interference Checking

Given: set $P$ of $n$ primitives (e.g. triangles, lines) or objects (e.g. convex polygons or polytopes)

Approach

- Filtering step, or broad phase: smart selection (using a data structure) of pairs of primitives/objects that may intersect
- Refinement step, or narrow phase: actual intersection check of candidate pairs resulting from filtering step
Refinement Step

- Primitive-primitive collisions
- Object-object collisions
Refinement Step: Primitives

(Fast) intersection tests for primitives are non-trivial

• Test: does tetrahedron A intersect tetrahedron B?
Refinement Step

• Test: does tetrahedron A intersect tetrahedron B?

• Solution:
  - edge of A intersects triangular facet of B, or
  - edge of B intersects triangular facet of A, or
  - vertex of A lies inside B, or
  - vertex of B lies inside A
Refinement Step

- Test: does point $p$ lie inside simple polyhedron $A$?
Refinement Step

• Test: does point $p$ lie inside simple polyhedron $A$?

• Solution:
  - count number of intersections of a half-line emanating from $p$ with boundary facets of $A$
   • if odd: $p$ lies inside $A$
   • if even: $p$ is not in $A$
  - beware: degenerate cases!
Refinement Step

- Test: does segment pq intersect triangle abc?
Refinement Step

• Test: does segment pq intersect triangle abc?

• Solution:
  - p and q lie on opposite sides of plane through a, b, and c and
  - q and c lie on same side of plane through a, b, and p and
  - q and b lie on same side of plane through a, c, and p and
  - q and a lie on same side of plane through b, c, and p
Refinement Step: Objects

- The distance between two non-intersecting objects is the minimum distance between a point in the first object and a point in the second object (or the minimum distance these objects need to translate to become intersecting).

- The penetration depth of two intersecting objects is the minimum distance these objects need to translate to become disjoint.
Refinement Step: Objects

- The Gilbert-Johnson-Kheerti (GJK) algorithm computes the distance or penetration depth for two convex objects $A$ and $B$, which equals the (shortest) distance from the origin to the Minkowski sum of $A$ and $-B$ (or of $B$ and $-A$)

- The Chung-Wang (CW) algorithm is an improvement of GJK for a specific class of convex objects
Separating Axis Theorem

- A separating axis of two disjoint objects A and B is a vector $v$ such that the projections of A and B onto $v$ do not overlap.

- For any pair of nonintersecting polytopes, there exists a separating axis that is orthogonal to a facet of either polytope, or orthogonal to an edge from each polytope.
Separating Axis Tests

• This means that for a pair of polytopes with $f_1$ and $f_2$ facet orientations respectively and $e_1$ and $e_2$ edge orientations respectively, we need to test $f_1 + f_2 + e_1 e_2$ axes for separation

• Number of tests is high; examples (in 3D):
  - triangle – triangle $\rightarrow$ 11 tests
  - triangle – box $\rightarrow$ 13 tests
  - box – box $\rightarrow$ 15 tests
Terminology: Separating Plane

• A strongly separating plane strictly leaves two objects on opposite sides; strongly separating planes are very hard to find

• A weakly separating plane leaves two objects on opposite sides (probably touching them)
Terminology: Support Mapping

- Gives (an) extreme point $S_C(d)$ in $C$ in any direction $d$
Terminology: Simplex

0-simplex  1-simplex  2-simplex  3-simplex

simplex
Terminology: Convex Hull

Point set C

Convex hull, CH(C)
GJ K Algorithm

1. Initialize the simplex set $Q$ with up to $d+1$ points from $C$ (in $d$ dimensions)
2. Compute point $P$ of minimum norm in $CH(Q)$
3. If $P$ is the origin, exit; return 0
4. Reduce $Q$ to the smallest subset $Q'$ of $Q$, such that $P$ in $CH(Q')$
5. Let $V = S_C(-P)$ be a supporting point in direction $-P$
6. If $V$ no more extreme in direction $-P$ than $P$ itself, exit; return $||P||$
7. Add $V$ to $Q$. Go to step 2
INPUT: Convex polytope $C$ given as the convex hull of a set of points
1. Initialize the simplex set $Q$ with up to $d+1$ points from $C$ (in $d$ dimensions)

$$Q = \{Q_0, Q_1, Q_2\}$$
GJK example 3(10)

2. Compute point $P$ of minimum norm in $\text{CH}(Q)$

$Q = \{ Q_0, Q_1, Q_2 \}$
3. If $P$ is the origin, exit; return 0
4. Reduce $Q$ to the smallest subset $Q'$ of $Q$, such that $P$ in $\text{CH}(Q')$

$Q = \left\{ Q_1, Q_2 \right\}$
5. Let \( V = S_C(-P) \) be a supporting point in direction \(-P\)

\[ Q = \{Q_1, Q_2\} \]

\[ V = S_C(-P) \]
6. If V no more extreme in direction \(-P\) than P itself, exit; return \(|P|\)
7. Add V to Q. Go to step 2

\[ Q = \{Q_1, Q_2, V\} \]
2. Compute point P of minimum norm in CH(Q)

$$Q = \{Q_1, Q_2, V\}$$
GJK example 8(10)

3. If P is the origin, exit; return 0
4. Reduce Q to the smallest subset Q' of Q, such that P in CH(Q')

\[ Q = \{ Q_2, V \} \]
5. Let $V = S_C(-P)$ be a supporting point in direction $-P$

$Q = \{Q_2, V\}$

$V' = S_C(-P) = Q_2$
GJK example 10(10)

6. If \( V \) no more extreme in direction \(-P\) than \( P \) itself, exit; return \( ||P|| \)

\[ Q = \{Q_2, V\} \]
Filtering Step

Prevent testing all pairs of primitives for intersection, but use data structure to select a (preferably small) set of candidates

Approaches

• Space partitioning
• Model partitioning
Space Partitioning

- Decompose space into cells
- Store references to intersecting primitives with each cell
Space Partitioning

- Idea: only test pairs of primitives that intersect the same cell (and at least one of the primitives is moving)

- Issue: store moving objects in data structure or not (and instead just query with them)? Storing moving objects requires updates of the data structure
Query with Moving Object
Voxel Grid

- Use regular grid for subdivision of space

- Subdivision is stored implicitly
- Fast access of cells
Voxel Grid

Cluttered scenes

- Distribution is not taken into account
Differently-sized objects

Voxel grids work well for roughly uniformly-distributed and equally-sized objects
Quadtree (or Octree)

- Take distribution into account by appropriately varying cell size
- Square/cubic cells only
- Top-down decomposition: subdivide a cell only if the number of objects intersecting it is (too) large
Quadtree

Build Quadtree (set $P$, square $\sigma$):

if $P$ and $\sigma$ satisfy stop criterion
then create leaf for $\sigma$ and store references to elements of $P$
else
1. split $\sigma$ into four quadrants $\sigma_1$...$\sigma_4$
2. determine subsets $P_i$ of $P$ of objects intersecting $\sigma_i$
3. create internal node $\nu$, with the trees resulting from recursively running Build Quadtree ($P_i$, $\sigma_i$) for $1 \leq i \leq 4$ as its children
Quadtree
Quadtree

- More adaptive than voxel grids
- Extra overhead: tree structure
- Slower access of cells

Still many empty cells in case of uneven object distribution
Kd Tree

More freedom in subdivision: splits by means of axis-parallel line

• Horizontal or vertical:
  – alternatingly
  – longest side first

• Location:
  – bisecting vertices
  – bisecting centroids
Kd Tree

![Diagram of Kd Tree](image)
Kd Tree

Problematic scene
BSP Tree

Even more freedom in subdivision: splits by means of any line
Space Partitioning

General disadvantage: multiple references to the same object (due to that object intersecting multiple cells):

- requiring extra memory
- leading to multiple intersection tests for the same pair during an interference query
Model Partitioning

- Strategies use **bounding volumes**

- Bounding volume of an object is a primitive shape that encloses the object

- Idea: if a query object does not intersect a bounding volume it does also not intersect the enclosed object
Bounding Volume

Desired properties:
• good fit: not much larger than enclosed object
• fast intersection test for pairs of bounding volumes
• small storage requirement

Examples of bounding volumes
• sphere
• axis-aligned bounding box (AABB)
• oriented bounding box (OBB)
• discrete oriented polygon/polytope (k-DOP)
Bounding-volume hierarchy is a tree structure with
• leafs storing a primitive and its bounding volume
• internal nodes storing the bounding volume of all the primitives stored in the subtree rooted at the node

Grouping of objects at internal nodes is crucial to the performance of interference queries (as a query once again accesses all nodes whose associated bounding volume is intersected by the query object)
Bounding-Volume Hierarchy
Bounding-Volume Hierarchy

Bad clustering of objects leads to recursive calls on too many internal nodes
Bounding-Volume Hierarchy

Comparison with original hierarchy

● = query object