Chapter 10
Collision and Rigid Body Dynamics
asyrani.com
10.1 Do You Want Physics in Your Game?
Things You Can Do with a Physics System

A LOT
Is Physics Fun?

Subspace Fluid Re-Simulation
Simulations (Sims)

- Flight Simulator
- Gran Turismo
- Need For Speed
Physics Puzzle Games

Bridge Builder

Fantastic Contraption

Crayon Physics

The Incredible Machine
Sandbox Games

LittleBigPlanet

GTA 5

Spore
Goal-Based and Story-Driven Games

A goal-based game has rules and specific objectives that the player must accomplish in order to progress; in a story-driven game, telling a story is of paramount importance.

Integrating a physics system into these kinds of games can be tricky. We generally give away control in exchange for a realistic simulation, and this loss of control can inhibit the player’s ability to accomplish goals or the game’s ability to tell the story.
Impact of Physics on a Game

- Predictability
- Tuning and control
- Emergent behaviors
<table>
<thead>
<tr>
<th>Engineering Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tools pipeline</td>
</tr>
<tr>
<td>User interface</td>
</tr>
<tr>
<td>Collision detection</td>
</tr>
<tr>
<td>AI</td>
</tr>
<tr>
<td>Animation and character motion</td>
</tr>
<tr>
<td>Rag doll physics</td>
</tr>
<tr>
<td>Graphics</td>
</tr>
<tr>
<td>Networking and multiplayer</td>
</tr>
<tr>
<td>Record and playback</td>
</tr>
</tbody>
</table>
Art Impacts

- Additional tool and workflow complexity
- More-complex content
- Loss of control
Other Impacts

*Interdisciplinary impacts.* The introduction of a dynamics simulation into your game requires close cooperation between engineering, art, and design.

*Production impacts.* Physics can add to a project’s development costs, technical and organizational complexity, and risk.
10.2 Collision/Physics Middleware
Architecture for Multi-body Collision Detection

- Pruning Multi-body Pairs
  - object transformations
  - overlapping pairs

- Pairwise Exact Collision Detection
  - response parameters

- Analysis/Response
  - colliding pairs
Figure 1. Dobkin-Kirkpatrick Hierarchy Construction
V-Collide

Diagram:

- Active object transformations
- VRML Scene Graph
- Sweep and Prune
- V-COLLIDE Architecture
- Exact Primitive Intersection Test
- Overlapping AABB Test
- OBBTree Overlap Test
- Colliding object pairs
RAPID
ODE stands for “Open Dynamics Engine” (http://www.ode.org). As its name implies, ODE is an open-source collision and rigid body dynamics SDK. Its feature set is similar to a commercial product like Havok.

Its benefits include being free (a big plus for small game studios and school projects!) and the availability of full source code (which makes debugging much easier and opens up the possibility of modifying the physics engine to meet the specific needs of a particular game).
Bullet is an open-source collision detection and physics library used by both the game and film industries. Its collision engine is integrated with its dynamics simulation, but hooks are provided so that the collision system can be used standalone or integrated with other physics engines.

It supports continuous collision detection (CCD)—also known as time of impact (TOI) collision detection—which as we’ll see below can be extremely helpful when a simulation includes small, fast-moving objects.
PhysX started out as a library called Novodex, produced and distributed by Ageia as part of their strategy to market their dedicated physics coprocessor.

It was bought by NVIDIA and is being retooled so that it can run using NVIDIA’s GPUs as a coprocessor.
Havok is the gold standard in commercial physics SDKs, providing one of the richest feature sets available and boasting excellent performance characteristics on all supported platforms. (It’s also the most expensive solution.)

Havok is comprised of a core collision/physics engine, plus a number of optional add-on products including a vehicle physics system, a system for modeling destructible environments, and a fully featured animation SDK with direct integration into Havok’s rag doll physics system.
The Physics Abstraction Layer (PAL) provides a unified interface to a number of different physics engines. This enables the use of multiple physics engines within one application.

http://www.adrianboeing.com/pal/index.html
Pixelux Entertainment S.A., located in Geneva, Switzerland, has produced a unique physics engine that uses finite element methods to simulate the dynamics of deformable and breakable objects, called Digital Molecular Matter (DMM).

The engine has both an offline and a runtime component. It was released in 2008 and can be seen in action in LucasArts’ *Star Wars: The Force Unleashed*.

http://www.pixelux.com/
10.3 The Collision Detection System
The primary purpose of a game engine’s collision detection system is to determine whether any of the objects in the game world have come into contact. To answer this question, each logical object is represented by one or more geometric shapes.

These shapes are usually quite simple, such as spheres, boxes, and capsules. However, more-complex shapes can also be used. The collision system determines whether or not any of the shapes are intersecting (i.e., overlapping) at any given moment in time. So a collision detection system is essentially a glorified geometric intersection tester.
If we want a particular logical object in our game to be capable of colliding with other objects, we need to provide it with a collision representation, describing the object’s shape and its position and orientation in the game world.

This is a distinct data structure, separate from the object’s gameplay representation (the code and data that define its role and behavior in the game) and separate from its visual representation (which might be an instance of a triangle mesh, a subdivision surface, a particle effect, or some other visual representation).
Tunneling
Tunneling
Tunneling
Tunneling
Complex Objects

• $N^2$ problem between triangles of individual models
Approaches

• Too many objects
  • Partition Space
  • Velocity Bounds
Spatial Partition

- Uniform Grid
Interval Projection

- Project bounding volume on each axis
- Maintain sorted lists
- Possible collision when intervals overlap
Tunneling

• Fundamentally a sampling problem
  • If you know the max velocity and minimum thickness, can bound the error
Tunneling

• Fundamentally a sampling problem
  • If you know the max velocity and minimum thickness, can bound the error
Sweep Methods

- Sweep out the volume along the path
  - Different accuracy choices
- Test for collisions
  - False positives
  - Bisect the interval
- Rotations are tough
Time of collision

- Interval Halving
- Conservative Advancement
- Minkowski Difference/ray-cast
Collision for Complex Models

• Models may have millions of primitives
  • Not moving independently, so sweep methods are overkill
  • Spatial partitions are tough to update
  • May be in close proximity for extended periods
    • Need to avoid false positives
Bounding Volumes

- Objects are often not colliding
  - Need fast reject for this case
  - Surround with some bounding object
    - Like a sphere

- Why stop with one layer of rejection testing?
  - Build a bounding volume hierarchy (BVH)
    - A tree
Bounding Volume Hierarchies

• Model Hierarchy:
  • each node has a simple volume that bounds a set of triangles
  • children contain volumes that each bound a different portion of the parent’s triangles
  • The leaves of the hierarchy usually contain individual triangles

• A binary bounding volume hierarchy:
Introduction

• Bounding Volume Hierarchies vs. Spatial Partitioning
  • What are they and how do they compare?

• Motivation: Need for Speed!
  • Demonstration through applications:
    View-frustum culling, ray-tracing, collision detection

• How can hierarchies help?
  • Apply to example applications

• Building bounding volume hierarchies
• Building spatial partitionings
• What’s the best choice?
• Can we do better?
What are they? How do they Compare?

Bounding Volume Hierarchies

• Hierarchical object representation
• Object subdivision
• Hierarchical clustering of objects
• Object levels of detail
• Classifies regions of space around objects

Examples:
• OBB-trees
• AABB-trees
• Sphere-trees
• k-DOPs

Spatial Partitioning

• Hierarchical spatial representation
• Spatial subdivision
• Hierarchical clustering of space
• Spatial levels of detail
• Classifies objects around regions of space

Examples:
– Uniform grids
– Quadtrees & Octrees
– BSP-trees
– kD-trees
Examples

Bounding Volume Hierarchies

• Tightly fits objects
• Redundant spatial representation

Spatial Partitioning

• Tightly fills space
• Redundant object representation
Examples

Bounding Volume Hierarchies

- Tightly fits objects
- Redundant spatial representation

Spatial Partitioning

- Tightly fills space
- Redundant object representation

Volumes overlap multiple objects

Objects overlap multiple volumes
Examples

Bounding Volume Hierarchies

• Tightly fits objects
• Redundant spatial representation

Spatial Partitioning

• Tightly fills space
• Redundant object representation

Volumes overlap multiple objects

Objects overlap multiple volumes
Examples

Bounding Volume Hierarchies

• Tightly fits objects
• Redundant spatial representation

Spatial Partitioning

• Tightly fills space
• Redundant object representation

Volumes overlap multiple objects
Objects overlap multiple volumes
Motivation: Example Applications

View-frustum culling $O(n)$

Ray-tracing $O(n)$ per ray

Collision detection $O(n^2)$

$n$: number of objects in the scene
How do we speed it up?

• More efficient intersection calculations
• Avoid intersection calculations
  • Make a single intersection calculation to decide for an entire cluster of objects or space
  • Cluster hierarchically
How can bounding volume hierarchies help?

- View-frustum culling
- Ray-tracing
- Collision detection
How can bounding volume hierarchies help?

View-frustum culling

Ray-tracing

Collision detection
How can bounding volume hierarchies help?

- View-frustum culling
- Ray-tracing
- Collision detection
How can bounding volume hierarchies help?

- View-frustum culling
- Ray-tracing
- Collision detection
How can bounding volume hierarchies help?

Logarithmic search for intersecting primitives!
How can spatial partitioning help?

- View-frustum culling
- Ray-tracing
- Collision detection

Uniform spatial partitioning
How can spatial partitioning help?

Performance varies for uniform partitioning, but hierarchical approaches also give logarithmic search for intersecting primitives!
What are the potential problems?

• What are the hidden costs?
  • When nothing intersects?
  • When nearly everything intersects?
  • What are the worst cases?

• Is it worth it?

• What applications get the most benefit?

• What about just using my modeling hierarchy?
  • Too shallow (not fine enough level of detail
  • Designed for object manipulation rather than minimizing intersections
  • Insensitive to actual positions of objects
Building Bounding Volume Hierarchies

• Choose a bounding volume type
  • Axis-aligned bounding box (AABB)
  • Oriented bounding box (OBB)
  • Sphere
  • Convex Hull
  • k-DOP (discrete oriented polytope)

• Choose a clustering strategy
  • Top-down:
    how do we partition objects among children?
  • Bottom-up:
    how do we find leaf clusters and merge into parents?
Bounding Volume Type

- Intersection cost vs. tightness of fit vs. storage overhead vs. implementation complexity
- How do we find the best fit for a particular bounding volume?
  - AABBs and convex hulls are clear. What about spheres, k-DOPs, and OBBs?
- How do we compare the quality of fit between different BVs?
  - Min volume, min surface area, etc.
Hierarchical Clustering Strategy

**Top-down:** how do we partition objects among children?
- Choosing splitting axis
  - longest dimension, largest spread of objects, etc.
- Choosing split point
  - mean, median, largest gap, etc.

**Bottom-up:** how do we find leaf clusters and merge into parents?
- Leaf object clusters
  - single primitive, specific minimum size cluster, etc.
- Merging children into parent
  - Nearest neighbors: uniform subdivision, Voronoi diagram
Building Spatial Partitionings

Decide how to recursively subdivide space (top-down)

Uniform subdivision

Quadtrees

kD-tree

BSP-tree

Decide how to classify objects into regions of space with respect to partitioning plane

Store in both regions,
Store with partition, or
Split geometry
What’s the best choice?

• Depends on the application
  • trial and error?
  • “Gut” feeling?
  • Careful analysis based on a cost function?

• Factors:
  • Complexity of implementation
  • Storage overhead
  • Computational overhead
  • Type of geometry: static or dynamic
Can we do better?

• Combining bounding volume hierarchies and spatial partitioning
  • Examples:
    • Occlusion culling: octrees of BSP-trees
    • Radiosity: 3D BSP-trees of 2D BSP-trees

• Hybrid bounding volume hierarchies
  • adaptive nodes
  • adaptive trees
  • performance driven metrics
BVH-Based Collision Detection
Type of Bounding Volumes

- Spheres
- Ellipsoids
- Axis-Aligned Bounding Boxes (AABB)
- Oriented Bounding Boxes (OBBs)
- Convex Hulls
- $k$-Discrete Orientation Polytopes ($k$-dop)
- Spherical Shells
- Swept-Sphere Volumes (SSVs)
  - Point Swept Spheres (PSS)
  - Line Swept Spheres (LSS)
  - Rectangle Swept Spheres (RSS)
  - Triangle Swept Spheres (TSS)
• OBB or AABB
  • OBB slightly better for close proximity
  • AABB better for handling deformations
Building an OBBTree

Recursive top-down construction: partition and refit
Building an OBB Tree

Given some polygons, consider their vertices...
Building an OBB Tree

... and an arbitrary line
Building an OBB Tree

Project onto the line

Consider variance of distribution on the line
Building an OBB Tree

Different line, different variance
Building an OBB Tree

Maximum Variance
Building an OBB Tree

Minimal Variance
Given by eigenvectors of covariance matrix of coordinates of original points
Building an OBB Tree

Choose bounding box oriented this way
Building an OBB Tree

Good Box
Building an OBB Tree

Add points: worse Box
Building an OBB Tree

More points: terrible box
Building an OBB Tree

Compute with extremal points only
Building an OBB Tree

“Even” distribution:
good box
“Uneven” distribution: bad box
Building an OBB Tree

Fix: Compute facets of convex hull...
Better: Integrate over facets

Building an OBB Tree
Building an OBB Tree

... and sample them uniformly
Tree Traversal

Disjoint bounding volumes:
No possible collision
Tree Traversal

Overlapping bounding volumes:
- split one box into children
- test children against other box
Tree Traversal
Tree Traversal

Hierarchy of tests
Separating Axis Theorem

L is a separating axis for OBBs A & B, since A & B become disjoint intervals under projection onto L.
OBB Overlap Test: An Axis Test

$L$ is a separating axis iff: $s > h_a + h_b$
OBB Overlap Test: Axis Test Details

Box centers project to interval midpoints, so midpoint separation is length of vector $T$’s image.

$$s = \left| \left( T^A - T^B \right) \cdot n \right|$$
OBB Overlap Test: Axis Test Details

• Half-length of interval is sum of box axis images.

\[ r_B = b_1 |R_1^B \cdot n| + b_2 |R_2^B \cdot n| + b_3 |R_3^B \cdot n| \]
OBB Overlap Test

- Strengths of this overlap test:
  - 89 to 252 arithmetic operations per box overlap test
  - Simple guard against arithmetic error
  - No special cases for parallel/coincedent faces, edges, or vertices
  - No special cases for degenerate boxes
  - No conditioning problems
  - Good candidate for micro-coding
Talk outline

• What is the GJK algorithm
• Terminology
• “Simplified” version of the algorithm
  • One object is a point at the origin
  • Example illustrating algorithm
• The distance subalgorithm
• GJK for two objects
  • One no longer necessarily a point at the origin
• GJK for moving objects
GJK solves proximity queries

• Given two convex polyhedra
  • Computes distance $d$
  • Can also return closest pair of points $P_A$, $P_B$
GJK solves proximity queries

- Generalized for arbitrary convex objects
  - As long as they can be described in terms of a support mapping function
Terminology 1(3)

Supporting (or extreme) point $P$ for direction $d$ returned by support mapping function $S_C(d)$.
Terminology 2(3)

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

simplex
Terminology 3(3)

Point set $C$

Convex hull, $CH(C)$
The GJK 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 $\text{CH}(Q)$
3. If $P$ is the origin, exit; return 0
4. Reduce $Q$ to the smallest subset $Q'$ of $Q$, such that $P$ in $\text{CH}(Q')$
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 polyhedron $C$ 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)

$Q = \{Q_0, Q_1, Q_2\}$
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 $\text{CH}(Q')$

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

$Q = \{Q_1, Q_2\}$

$V = S_c(-P)$
GJK example 6(10)

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 $\text{CH}(Q)$

$Q = \{Q_1, Q_2, V\}$

$Q_1$ to $Q_2$ connected with polygon, $V$ and $P$ marked on it.
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 \(-P\)

\[ Q = \{Q_2, V\} \]
GJK example 10(10)

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

\[ Q = \{Q_2, V\} \]
Distance subalgorithm 1(2)

• Approach #1: Solve algebraically
  • Used in original GJK paper
  • Johnson’s distance subalgorithm
    • Searches all simplex subsets
    • Solves system of linear equations for each subset
    • Recursive formulation
    • From era when math operations were expensive
    • Robustness problems
  • See e.g. Gino van den Bergen’s book
Distance subalgorithm 2(2)

- Approach #2: Solve geometrically
  - Mathematically equivalent
    - But more intuitive
    - Therefore easier to make robust
  - Use straightforward primitives:
    - ClosestPointOnEdgeToPoint()
    - ClosestPointOnTriangleToPoint()
    - ClosestPointOnTetrahedronToPoint()
  - Second function outlined here
    - The approach generalizes
Closest point on triangle

- `ClosestPointOnTriangleToPoint()`
  - Finds point on triangle closest to a given point
Closest point on triangle

- Separate cases based on which feature Voronoi region point lies in
Closest point on triangle

\[ AX \cdot AB \leq 0 \]
\[ AX \cdot AC \leq 0 \]
Closest point on triangle

\[(BC \times BA) \times BA \cdot BX \geq 0\]

\[AX \cdot AB \geq 0\]

\[BX \cdot BA \geq 0\]
GJK for two objects

• What about two polyhedra, $A$ and $B$?
• Reduce problem into the one solved
  – No change to the algorithm!
  – Relies on the properties of the Minkowski difference of $A$ and $B$

• Not enough time to go into full detail
  – Just a brief description
Minkowski sum & difference

- Minkowski sum
  - The sweeping of one convex object with another

- Defined as:
  - \( A + B = \{a + b : a \in A, b \in B\} \)
End of Game Engine Development I