Physics

Overview

- Collision detection
  - Model representation
  - Dynamic vs. static
  - Discrete vs. continuous
  - Efficiency issues
- Collision resolution
  - Events
  - Solvers
- Dynamics
  - Rigid bodies
  - Impulse-based collision solvers
  - LCP solvers

Properties of a Good Physics Engine

- Fast
  - Naive solutions eat up a lot of performance
  - Designers will always push the limits of a physics engine
- Robust
  - Stable and predictable under typical game frame-rates and object interactions
    - Doesn't “blow up” unexpectedly
  - “Enough” accuracy
    - We are not landing a probe on Mars
- Tunable
  - Intuitive controls to change behaviour of objects
    - No 15 Greek letter parameter friction models
- Extensible
  - One-size-fits-all doesn't

Collision Detection

- Find all relevant spatial interactions of objects in the world
- Input: physics world description
  - Generally a simplified AI/rendering representation
- Output: object interactions
  - Per-frame
  - Usually a list of collision pairs
  - Information associated with interactions depends on how collisions are handled by the game
    - Typical: colliding object ID's, points of contact, contact normals, penetration depth

Model Representation

- Triangle mesh
  - Allows for collision detection against arbitrary shape
  - Can be derived directly off rendering mesh
  - Generally requires tuned hierarchical data structure to be efficient
  - Requires well-formed mesh (no cracks, T-junctions)
  - Collision response can be tricky
- Convex hull
  - Several can be combined to define an arbitrary shape
  - Established, efficient collision detection algorithms (GJK)
  - Artists aren't good at manually creating convex hulls
    - See www.qhull.org for an automated toolkit

Model Representations Continued

- Simplified volume
  - Sphere, ellipse, box (OBB), capsule, cylinder, cone
  - Straightforward, closed form, geometric collision detection formulas
  - Efficiently models certain types of curved surfaces
  - Requires artist to wrap meshes
    - or a somewhat tricky automated system
- Height-field
  - Easy to create
  - Very efficient collision detection, ray casting
  - Unsuitable for general purpose use

Detection

- Collision detection is inherently $O(N^2)$
  - For each object, test it for collision with every other object
  - Gets very slow very quickly
  - Might be good enough for a small game though
- Fortunately, very few objects actually are interacting each frame
  - We hope
- We know things about the world that can speed things up

Pair Filtering

- Some objects will never collide, so don't test them against each other
  - Objects that can't move (static world sections)
  - Important enough that this is usually built into the low-level collision system
    - Distinction between static and dynamic objects
- Use semantics of game world to avoid collision tests
  - E.g. objects attached to characters often have their collision against them disabled
  - We represent this with bit masks

Two Phase Detection

- Often, low-level collision detection is done in two steps:
  - Broad phase: rapidly find potential colliders, usually using approximate bounding volumes
    - Spheres, Axis Aligned Bounding Boxes (AABB)
    - Sphere-sphere is much easier than mesh-mesh!
  - Narrow phase: Brute-force compare potential colliders to find actual collisions

Broad Phase Strategies

- Spatial partitioning
  - Grid
  - Octree
  - BSP tree
- Clustering
  - Volume trees
  - Sweep-and-prune
  - Spatial hashing
- Coherence
  - Collision cache
  - Prediction
  - Relied on heavily in spatial sorting
    - Cf. sweep-and-prune

Rays

- Ray-casting is a very frequently performed operation in a game
- Collision detection systems have to consider rays in their underlying design
  - “Tacked-on” ray casting algorithms tend to be inefficient
    - Can overwhelm collision detection times
- Rays are different from other collision primitives
  - Rays can be very long
  - Often only concerned with the “first hit”
- Some suggestions:
  - Model rays as line segments
  - Keep rays as short as possible
  - Bound short rays with AABBs
  - Favour data-structures that return collisions in “ray order”

Discrete Methods

- Collision detection is typically performed once per frame
- Discrete approaches test instantaneous positions of objects, checking for overlap
  - Majority of collision detection systems operate this way
  - Algorithms well covered in literature
- Have to deal with penetration issues
  - Carefully constructed detection routines to give sensible results in moderate penetration cases
  - Very deep penetration will be hard to deal with
- Time-step has to be small enough to catch all collisions
  - Bullet through sheet of paper problem

Continuous Methods

- Find the exact moment of contact
  - Doesn't suffer from pass-through problems
- Easy with rays
- Considerably more work for complex objects with multi-point contacts
- Hard for non-linear motion
  - Ballistic trajectory
  - Tumbling shapes
- Usually approximated by assuming linear motion between frames
  - Sweep-methods, interval arithmetic
- Higher per-frame cost than discrete
  - But perhaps a larger time-step can be used

Resolution

- What a game does with detected collisions is called resolution, or solving
- Many possibilities, some handled by the game logic, others by the physics system itself:
  - Ignore (objects pass through each other)
  - Bounce (perhaps using dynamics engine)
  - Stick
  - Destroy one, or both objects (replace with special effect)
  - Send event (for sounds, damage application, AI triggers)
  - Apply force
  - Deform
  - Change state of one or more objects
  - A combination of the above
- The resolution system must be flexible

Dynamics

- A dynamics system is concerned with object positions and orientations
  - It can be thought of as physics-based animation system
- Lots of uses and approaches
- We'll briefly talk about rigid bodies
- Won't talk about:
  - Articulated objects (constraints)
  - Flexible objects (rope, hair, cloth, skin)
  - Fluids (smoke, water)

Rigid Body Dynamics

- Rigid body
  - Transform specifying position (centre of mass) and orientation
  - Linear and angular velocity vectors
  - Mass and mass matrix (inertia tensor)
  - Collision primitive (box, sphere, capsule, etc)
  - Material properties (discussed later)
- Useful methods:
  - Get/Set velocity (linear, angular), position
  - Get velocity of point
  - Apply force, torque
- Integrator
  - Updates the linear and angular velocity, position and orientation of rigid bodies under the influence of forces at each time step

Collision Solvers

- Two objectives
  - Prevent object interpenetration
  - Providing plausible collision response
- Two major cases
  - Collision (“bouncing”)
  - Contact (resting, rolling, sliding, friction)
- Variety of techniques
  - Impulse-based methods
  - Linear Complimentary Problem (LCP) solver

Impulse Methods

- Pioneered by Mirtich and Canny [1]
- What is an impulse?
  - A force applied over a very small period of time
  - Impulses effect an instantaneous change in velocity
- For each collision point between two objects
  - Examine the relative velocities of the two points
  - If moving apart, do nothing
  - Otherwise, compute and apply a pair of equal but opposite impulses to the pair of rigid bodies at the collision points
  - Requires collision point, collision normals, velocities, body properties (inertia tensor, co-efficient of restitution, friction)

[1] Brian Mirtich and John Canny, "Impulse-based simulation of rigid bodies,"
in Proceedings of Symposium on Interactive 3D Graphics, 1995

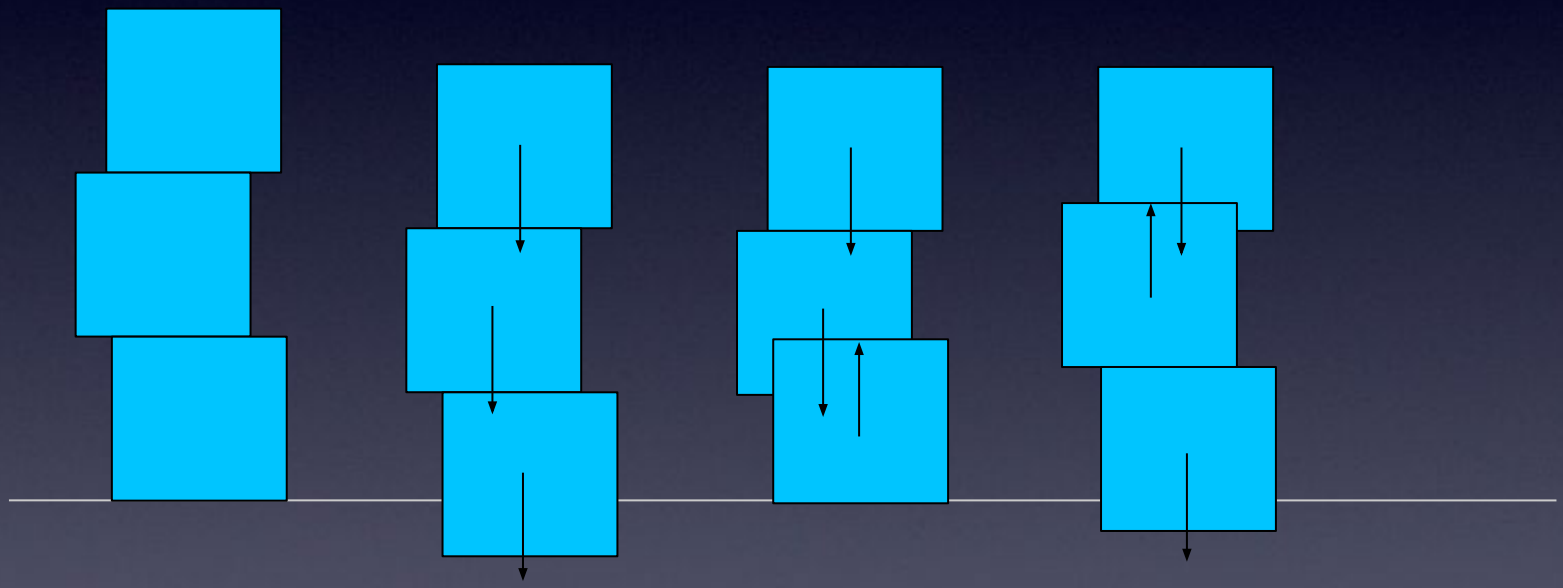
Multi-point Collision

- For the best accuracy, impulses for all collisions should be computed, and applied simultaneously
- This is difficult (see LCP solvers [3]), and could be expensive
- Instead, we can just apply the impulses sequentially, and tolerate the inaccuracy
- For boxes, we can average collision points that lie on the same face, which improves results.

[3] Erin Catto, "Iterative Dynamics with Temporal Coherence," Game Developer Conference, 2005.

Convergence

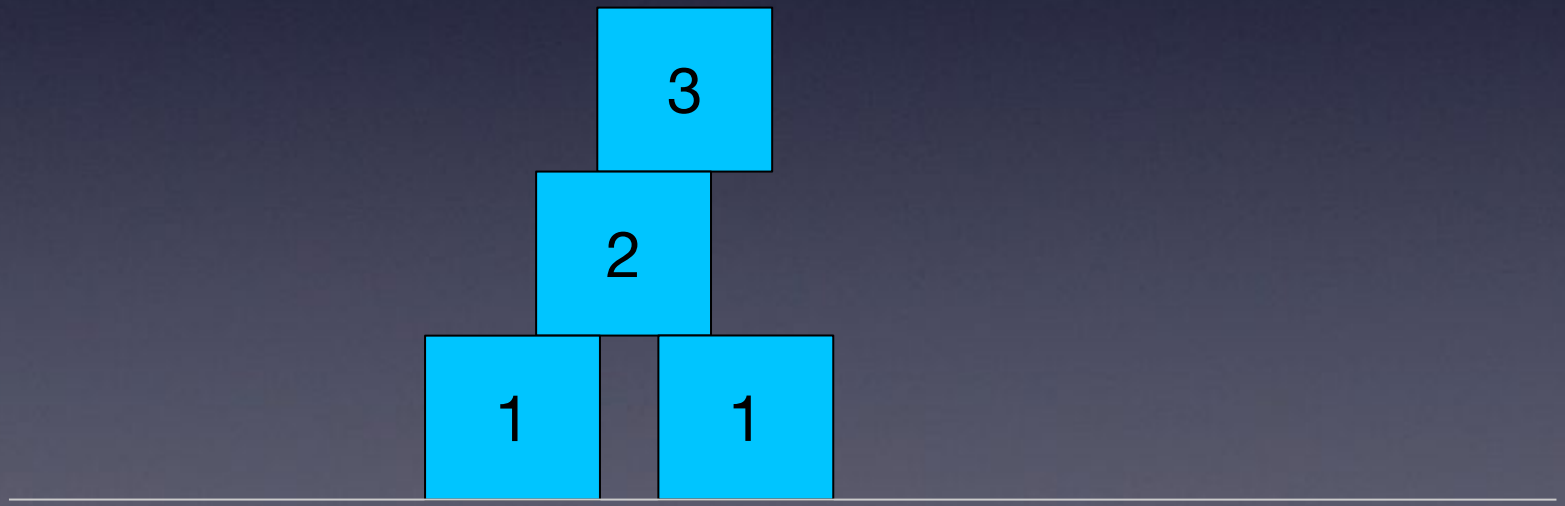
- After all collisions have been processed, some points that were moving apart may be adjusted by other impulses



- Many iterations are needed to solve large stacks

Improving Convergence

- Between two objects, solve deepest contact first
- Generate a “contact graph”, and solve contacts bottom-up with respect to gravity
- Shock propagation [4]
  - Process a level of the contact graph, then freeze the objects for subsequent processing (set mass to ∞ or equivalent)



[4] Eran Guendelman, Robert Bridson, Ronald Fedkiw, “Nonconvex Rigid Bodies with Stacking,” Stanford University.

Summary

- Collision detection and response are quite difference
- Physics engines are complicated and full of interesting design choices
  - Speed vs accuracy tradeoffs
  - Stability tradeoffs
- You will never have to implement one from scratch, but it is useful to have an idea of how they work