Notes

- Please read
  - Hahn, “Realistic animation of rigid bodies”, SIGGRAPH’88
  - Guendelman et al., “Nonconvex rigid bodies with stacking”, SIGGRAPH’03
- Check web site for instructions on printing slides
  - Use pdf2ps, not acroread

Object collision detection

- Before: particles vs. objects
  - Spheres vs. objects for signed distance
- Now: need objects vs. objects
- Interference detection
  - Do the objects intersect?
  - For collision processing: “where?”
- Collision detection
  - At some point in time, did the objects intersect?
  - When and where?

Convex polyhedra

- Lots of research on convex polyhedral objects
- Intersection of a finite set of half-planes
  - $\{x \mid n_i \cdot (x-x_i) \leq 0 \text{ for } i=0,1, \ldots, m\}$
- Very easy to test inside/outside (if m is small)
  - [Aside: signed distance from this?]
- Also want to know the vertices
  - Object is the convex hull of the vertices
  - Can be used for the points x_i defining the half-planes

Acceleration

- BV hierarchy or grid acceleration structure can certainly prune out a lot of tests
- Also: notion of separating plane
  - A plane s.t. object 1 is on one side, object 2 is on the other
  - Convex objects intersect if and only if there is no separating plane
  - For convex polyhedra, if there is one, one example will either be
    - One of the defining half-planes
    - Or parallel to an edge from object 1 and an edge from object 2, going through one of the edge endpoints

More on convex bodies

- M. Lin: maintain closest pair of features
  - Time coherence (for convex bodies) makes this fast!
- Extension to nonconvex objects
  - Maybe unnecessary (if it's close to convex, just use simplified collision geometry which is convex)
  - Can decompose any object into a union of convex parts
  - Decomposition may be inefficient…

More general shapes

- For most every-day objects, forget about convexity
- Look at general triangle meshes and level sets again
- Fairly difficult to avoid all interpenetration (and not crucial usually)
  - So even if we use full collision detection, still look for residual intersections and use repulsions to separate

Where is the geometry?

- For efficiency, in all but the simplest cases, just store geometry in object space
- Two different objects have different object spaces
- Thus we'll need to convert between the two frequently
- General idea
  - Do collision tests in object 1's object space
  - Precompute composition of map from object 2's object space to world space, then from world space to object 1's object space
  - Use rotation matrix R, not quaternion

Where is the acceleration?

- Note that axis aligned bounding boxes won't be axis aligned after transformation
  - Become oriented bounding boxes
- Oriented bounding boxes need to be rotated in transformation
- Spheres are invariant
- Signed distance is invariant
  - But normals need to be rotated!
- Grids need to be rotated
- Or just keep one grid out in world space, and update or rebuild it each time step

Intersecting triangle meshes

- Check if an edge of one intersects a face of the other
  - Also check if one vertex of object 1 is inside object 2, and vice versa
- Naïve way:
  - Loop over faces of one, check each for intersection (accelerate each check)
  - Repeat for the faces of the other
- Better way:
  - Use acceleration structure to prune in BOTH objects simultaneously
  - Don't check any distant faces or edges

BV Hierarchy

- Start with an empty stack
  - Holds pairs: one BV node from object 1, one BV node from object 2
- Put the root BV's on the stack
- While stack isn't empty
  - Remove a pair of BV's from top, check if they intersect
    - If so, and they are at the base level, do the face/edge intersection tests
    - If so, and not at the base level, but all pairs of children on the stack

Grid

- Find all grid cells with primitives from both objects
  - Probably want separate lists for each object, or a count of # primitives from each object, to make this O(1) per grid cell
- Test those primitives for intersection

Colliding triangle meshes

- In general, both meshes moving
  - Need to detect point-face collisions as before with particles
  - Recall:
assume linear trajectories, form cubic, find possible collision times, test geometry at each collision time
- Also need to detect edge-edge collisions
  - Math is almost the same…

Edge-edge collisions

- Say edge endpoints are x_1-x_2 and x_3-x_4
- Parameterize with s=0 at start of time step, s=1 at end
- Positions are x_i+sv_i at time s
- Two edges intersect only if their lines lie in the same plane
- Redraw diagram
- End up with point in plane
- So get exactly the same cubic as before!

Edge-edge collisions 2

- Once we have a possible collision time, need to check if the edges actually overlap at that time
- $(1-a)x_1+ax_2=(1-b)x_3+bx_4$
- Again, over-determined
- Solve least-squares
  - What are the barycentric coordinates (a for edge 1-2, b for edge 3-4) of the closest points on the lines?
- [work out]
- Also note: normal is cross-product of edges

Acceleration

- Again, need to cover whole extent of triangle motion in the acceleration structure
- Can use the same recursive algorithm for efficiently finding colliding trajectories

Level sets

- Open (but low hanging fruit) problem:
  - Directly search signed distance fields for point where both are negative
- Simpler approach: dual representation
  - Use level sets which are great for point queries
  - Sprinkle particles on the surface (to provide the points)
  - Point sampling could come from a mesh, but it doesn't have to!
  - This will be an approximate detection
  - If we do have a mesh, can also check edges against level set (obj1 vs. obj2 and vice versa)…

Collision resolution

- Tricky part, especially with friction
  - Theorists still arguing about validity of Coulomb friction
  - Baraff '94: finding contact forces (polygonal objects, Coulomb friction) is NP-complete
  - Nobody really understands rolling friction yet
- We'll get by with simplified approximations: plausible results

Repulsion forces

- Don't work so well for stacks
  - Things get mushy…
- Work ok for simpler interaction
  - But friction is kind of dodgy…
- Basic premise
  - When objects collide, stick in a virtual damped spring at the collision point, until they separate
- Need a better approach
  - But still useful for separating objects that are still slightly interfering after other algorithms
  - In this case, can alter just positions…

Simplifying to points

- Let's focus on a single point collision
  - If just checking interference, use the deepest point
- Note: only work with non-separating points
  - If all interfering points are separating, we need to use repulsions instead
- This looks a lot like particles now
- But after resolving deepest point, need to check again with updated velocities
  - Are there more points that need resolving? [example]
  - Iterate a few (5?) times

Frictionless impulse

- Object velocities at point:
  - $v_i=\omega_i \times (x-X_i)+V_i$
- Relative velocity $v=v_1-v_2$
  - Normal component $v_n=v \cdot n$
- Want post-collision relative normal velocity to be $v_n^{after}=-\varepsilon v_n$
- Apply an impulse $j=j_n n$ in the normal direction to achieve this
- [work out]

Computing frictionless impulse

$$K_i = \frac{1}{M_i}\delta + \left(x - X_i\right)^{*T} I_i^{-1}\left(x - X_i\right)^*$$

$$j = \frac{-\left(1+\varepsilon\right)v_n}{n^T\left(K_1 + K_2\right)n} n$$

Adding friction

- Static friction valid only in “friction cone”

$$\left|j_T\right| \le \mu\left|j_n\right|$$

- Approach:
  - Calculate static friction impulse (whatever it takes to make relative velocity zero)
  - Check if it’s in the friction cone
  - If so, we’re done
  - If not, try again with sliding
- [work out]

Computing static friction

$$v^{after} = v - \left(1+\varepsilon\right)v_n n$$

$$j = -\left(1+\varepsilon\right)v_n\left(K_1 + K_2\right)^{-1} n$$

Sliding friction

- If computed static friction impulse fails friction cone test
- In general, sliding direction will change during impact!
  - Several papers, even in graphics, actually solve ODE’s for sliding friction during impact
- We’ll assume not: tangential impulse just in the initial relative velocity direction
  - In practice, good enough
- [work out]

Computing sliding friction

$$T = \frac{v - v_n n}{\left| v - v_n n \right|}$$

$$j = j_n n - \mu j_n T$$

$$j_n = \frac{-(1+\varepsilon) v_n}{n^T (K_1 + K_2)(n - \mu T)}$$

New twist: multiple collisions

- Before (particles vs. objects) we ignored particle vs. particle collisions
- Here, many collisions can occur in a time step
- Solving one pair changes velocities, which can cause a different pair to collide
- Solving that pair can cause another pair to collide
- Need to keep iterating
- [Stack example]

Collision resolution pipeline

- We need to combine different collision techniques
- After elastic bounces (fine to just do a few pairwise collisions) do inelastic contact
- Take candidate velocities for contact
- Pass through a pipeline
  - Early stages are fast, accurate, local - handle the simple cases well
  - Later stages may be slower, less accurate, but global and robust - handle the complex cases robustly and plausibly

Shock propagation

- For rigid bodies, first stages of contact pipeline are just doing pairwise inelastic impulses
  - Say 3-5 passes
- We want to finish up with something that handles the rest: stacks
  - Idea of shock propagation
  - Fix bottom object, freeze in place, then fix next object up, freeze in place, and continue

Who’s on who?

- Need to figure out ordering of objects - who’s on the bottom, who’s on top
- One by one, advance position of just one object and see which objects it collides with
- Form a directed graph (edge means “below”)
- Group cycles together to get a DAG
- Topologically sort DAG to get final ordering

Freezing

- Assume the ground is the bottommost object (and is immovable)
  - Note: if not, momentum is not conserved by shock propagation - need to be careful
- Then freezing just means we glue to the ground (kind of like static friction…)
- Implement it by setting K=0
  - Infinite mass, infinite inertia tensor (since we’re combining the object with the ground)
- Looks terrible, is badly inaccurate: except it works great when we already did a decent job at start of pipeline!

Some open problems

- Rolling friction
- Balancing stacks [draw example]