



Rigid Body Implementation Basics

Don't worry, people have already
done the hard work for us.



Why model with Rigid Bodies?

- Real materials undergo some deformation when forces are applied
 - However these are often for negligible and unimportant for graphics purposes
- Computational Efficiency
 - Rigidity reduces DoF to 6
 - Modeling internal strains would require discretization of the body volume



Unconstrained RB motion

- Not concerned with collisions, joints, friction, etc.
- We will assume relevant forces have already been computed
- We will show how to use these forces to evolve a body through time

Rigid Body mass properties

- Mass [m] – scalar
- Inertia tensor [I_{body}] – 3x3 matrix
- Can be computed efficiently directly from polygon mesh, assuming uniform density
 - “Fast and Accurate Computation of Polyhedral Mass Properties”
– Brian Mirtich '96 (C code provided)
 - Uses divergence theorem to relate surface integral to volume integral
 - Gives us mass, center of mass, and inertia tensor
- Both constant during simulation



Inertia Tensor

- Describes the distribution of mass
- Relates angular velocity to angular momentum
 - Not as simple as the relationship between linear velocity and linear momentum

Rigid Body State

- Position [x] – 3D Vector
 - Vector from (world) origin to center of mass
- Linear Momentum [P] – 3D Vector
 - $P = m * v$ [v is linear velocity]
- Orientation [q] – (unit) Quaternion
- Angular Momentum [L] – 3D Vector
 - *careful* $L = I * \omega$ [ω is angular velocity]

Why use a quaternion?

- 3x3 Rotation matrix [R] could also be used
- Since R represents a rotation it should be orthogonal, not stretch vectors, etc.
- Unfortunately numerical errors during integration cause R to drift (it acquires a shear component)
 - Solution: renormalize periodically (not ideal)
- Quaternions also drift, but normalizing is easy
- Requires fewer values, 4 vs. 9
 - Not a great memory saver, but means there are fewer ODEs to solve (more on this later)
- Can easily get 3x3 rotation matrix from quaternion

Integration

- ODE software solves systems of 1st order equations
 - Solution: flatten state vectors
- General interface to ODE routine
 - ODE(y_0 , t_0 , t_f , dydt())
 - y_0 - initial state
 - t_0 t_f - initial and final time
 - dydt – function that computes derivatives for a given state vector
 - Usually allow user defined error tolerances
- Wide variety of ODE solvers
 - Explicit vs. Implicit, Adaptive vs. Fixed Step size, etc.
 - Efficiency of each method depends on problem characteristics

Rigid Body Derivatives

- x' and P' are easy
 - simply P / mass and net force respectively
- $q' = 0.5 * (\langle 0, \omega \rangle * q)$
 - Here $\langle 0, \omega \rangle$ is a quaternion
 - Derivation in Baraff notes
 - Haven't said how to get ω from L yet
- $L' = \text{net torque}$

Relating Angular Velocities and Momentum

- Earlier we said that $L = I * \omega$
 - But I changes when body rotates
 - Could recompute I each time....
 - Or simply transform I_{body}
 - $I = R * I_{\text{body}} * R^T$
 - Hence $\omega = R * I_{\text{body}}^{-1} * R^T * L$

Frames of Reference

- Given a particle that is has offset H from the center of mass in local coordinates
 - Current location $x + R * H$
 - Current velocity $v + \omega \times H$

Overview of steps

- Load state from flattened vector
- Compute derived quantities from state
 - $q \rightarrow R$ and $L \rightarrow \omega$
- Compute forces and torques
- Return (flattened) derivatives to integrator



A great reference

- “An Introduction to Physically Based Modeling: Rigid Body Simulation I – Unconstrained Rigid Body Dynamics” – David Baraff
 - Used for SIGGRAPH Courses
 - Provides C pseudo-code
 - Comprehensive and explicit guide



Videos