Notes on Last Lecture

• Approximate interior normal may be quite wrong [corner example]
• Lots of potential ways to fix this if it happens
  • Fall back on collision detection (normal at collision point on surface should work)
  • If the object normal doesn’t work, use opposite of particle velocity instead (maybe too inelastic)
  • Use a repulsion impulse (and friction) to get out:
    \[ \Delta v_N = \left(-\frac{v}{\mu}\right)n \] (dangerous: adds energy!)
• Unfortunately, to be robust enough, usually need to throw in a bunch of hacks…

Notes

• Round-off error is also a problem
• In algorithms described before, can get into infinite loops if not careful: \( v_N^{\text{before}} + \Delta v_N \neq v_N^{\text{after}} \)
  • If collision resolution doesn’t seem to ever converge—could just be round-off
  • Simple fix: stop after a fixed number of iterations, keep the old particle position
  • Not so easy with moving objects - really need to update position
  • So use very weak repulsions to push objects just slightly clear of objects

Moving triangles

• Life is a little more complicated
• Assume corners of the triangles move in linear trajectories too
  • Note this is NOT rigid in general…
• At time \( s \), corner is at \( x_j + sv_j \)
  • (assume \( s \) starts at 0 at start of time step)
• Normal is also changing in time
  • So for plane intersection, need to substitute for \( n \) the cross-product formula
  • Thankfully, don’t need to normalize, since that doesn’t change the plane

Moving triangles equation

• A cubic in \( s \) to solve:
\[
\left[(1-s)p + sq - x_i\right] \cdot \left[(x_j(s) - x_i(s)) \times (x_k(s) - x_i(s))\right] = 0
\]
• Only interested in real solutions between 0 and \( \Delta t \)
• Solve iteratively
  • Derivative=quadratic can be solved to tell us if any extrema in interval
  • Values at endpoints and at any extrema in interval tell us the intervals that roots could be in
  • Solve for those roots with secant/bisection search
Acceleration

- Too slow to check every single triangle if mesh is large
- Need acceleration
- Also critical if we have lots of distinct objects (even if implicit)
- Lots of papers written on acceleration structures
  - Prune out unnecessary tests

Bounding Volume Hierarchy

- Surround each triangle (or small group of triangles) with a simple bounding volume
  - E.g. axis-aligned box, sphere, oriented box...
- Surround group of bounding volumes with a parent bounding volume, and so on up
- End up with a tree
- To check a segment against scene, check if it could overlap root of tree
  - If not, we're done
  - If so, recurse on children

Grid Acceleration

- Or, put down a virtual grid in space
  - Each grid cell has a list of which triangles overlap
- To test a segment, only look at triangles in the grid cells the segment crosses
- Can use hash table for memory efficiency
  - Hash on cell indices (i,j,k)
- Note trade-off:
  - The finer the grid, the fewer extraneous triangles
  - But: more grid cells to check, more memory used, and more expensive to build grid
  - Tune for your application!

Rigid Bodies

- Very well studied
- I’ll introduce them from a particle perspective
  - Easy to get lost in abstract notions
  - Particles are fundamental
- Discretize an object into small point masses
  - \( x_i, v_i, m_i \)
- Assume object doesn’t change shape (doesn’t deform)
  - What does that mean for the motion of the particles? How do we describe it, solve for it?
World Space vs. Object Space

- World space: where the particles actually are now
  - This is where we will look at x, v, and almost every other quantity
- Object space: imaginary “reference” place for the particles
  - Label the object space position \( p_i \)
  - Does not change as the object moves - things we compute in object space stay constant
  - We can define it arbitrarily

Mapping

- The map from \( p_i \) to \( x_i(t) \) cannot change the shape
  - The distance between any two particles never changes
  - Thus map has to be \( x_i(t) = R(t)p_i + X(t) \)
  - \( R(t) \) is an orthogonal 3x3 matrix: \( RRT^T = \delta \)
    - The orientation (rotation) of the object
  - \( X(t) \) is a vector
    - The “location” of the object

Rigid Motion

- Differentiate map w.r.t. time (using dot notation):
  \[ v_i = \dot{R}p_i + V \]
- Invert map for \( p_i \):
  \[ p_i = R^T(x_i - X) \]
- Thus:
  \[ v_i = \dot{R}R^T(x_i - X) + V \]
- 1st term: rotation, 2nd term: translation
  - Let’s simplify the rotation

Skew-Symmetry

- Differentiate \( RR^T = \delta \) w.r.t. time:
  \[ \dot{RR}^T + R\dot{R}^T = 0 \quad \Rightarrow \quad \dot{RR}^T = -(\dot{RR}^T)^T \]
- Skew-symmetric! Thus can write as:
  \[ \dot{RR}^T = \begin{pmatrix} 0 & -\omega_2 & \omega_1 \\ \omega_2 & 0 & -\omega_0 \\ -\omega_1 & \omega_0 & 0 \end{pmatrix} \]
- Call this matrix \( \omega^* \) (built from a vector \( \omega \))
  \[ \dot{RR}^T = \omega^* \quad \Rightarrow \quad \dot{R} = \omega^* R \]
The cross-product matrix

- Note that:
  \[ \omega^x = \begin{pmatrix} 0 & -\omega_2 & \omega_1 \\ \omega_2 & 0 & -\omega_0 \\ -\omega_1 & \omega_0 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \omega_1x_2 - \omega_2x_1 \\ \omega_2x_0 - \omega_0x_2 \\ \omega_0x_1 - \omega_1x_0 \end{pmatrix} = \omega \times x \]

- So we have:
  \[ v_i = \omega \times (x_i - X) + V \]

- \( \omega \) is the angular velocity of the object

Angular velocity

- Recall:
  - \( |\omega| \) is the speed of rotation (radians per second)
  - \( \omega \) points along the axis of rotation (which in this case passes through the point \( X \))
  - Convince yourself this makes sense with the properties of the cross-product

Force

- Take another time derivative to get acceleration:
  \[ a_i = \dot{v}_i = \ddot{R}p_i + A \]

- Use \( F = ma \), sum up net force on system:
  \[ \sum_i F_i = \sum_i m_i a_i = \sum_i m_i (\ddot{R}p_i + A) = \ddot{R} \sum_i m_i p_i + A \sum_i m_i \]

- Let the total mass be \( M = \sum_i m_i \)

- How to simplify the other term?

Centre of Mass

- Let’s pick a new object space position:
  \[ p_i^{\text{new}} = p_i - \frac{\sum_j m_j p_j}{M} \]

  - The mass-weighted average of the positions is the centre of mass
  - We translated the centre of mass (in object space) to the point \( 0 \)

  - Now: \[ \sum_i m_i p_i = 0 \]
Force equation

- So now, assuming we’ve set up object space right (centre of mass at 0), $F=MA$
- If there are no external forces, have $F=0$
  - Internal forces must balance out, opposite and equal
  - Thus $A=0$, thus $V=$constant
- If there are external forces, can integrate position of object just like a regular particle!

What about $R$?

- How does orientation change?
- Think about internal forces keeping the particles in the rigid configuration
  - Conceptual model: very stiff spring between every pair of particles, maintaining the rest length
- So $F_i = \sum_j f_{ij}$ where $f_{ij}$ is force on $i$ due to $j$
- Of course $f_{ij}+f_{ji}=0$
- Also: $f_{ij}$ is in the direction of $x_i-x_j$

Net Torque

- Play around: $\left((x_j-X)-(x_j-X)\right) \times f_{ij} = 0$
  
  \[
  (x_i-X) \times f_{ij} = (x_j-X) \times f_{ji} = -(x_j-X) \times f_{ji}
  \]

- Sum both sides (look for net force)
  
  \[
  \sum_{i,j} (x_i-X) \times f_{ij} = \sum_{i,j} (x_j-X) \times f_{ji}
  \]

  \[
  \sum_i (x_i-X) \times F_i = -\sum_j (x_j-X) \times F_j = 0
  \]

- The expression we just computed=0 is the net torque on the object

Torque

- The torque of a force applied to a point is $\tau_i = (x_i-X) \times F_i$
- The net torque due to internal forces is 0
- [geometry of torque: at CM, with opposite equal force elsewhere]
- Torque obviously has something to do with rotation
- How do we get formula for change in angular velocity?
Angular Momentum

- Use \( F = ma \) in definition of torque:
  \[
  \tau_i = (x_i - X) \times m_i a_i
  = \frac{d}{dt} \left[ m_i (x_i - X) \times v_i \right]
  \]
- force=rate of change of linear momentum, torque=rate of change of angular momentum
- The total angular momentum of the object is
  \[
  L = \sum_i m_i (x_i - X) \times v_i
  = \sum_i m_i (x_i - X) \times (v_i - V)
  \]

Inertia Tensor

- \( I(t) \) is the inertia tensor
- Kind of like “angular mass”
- Linear momentum is \( mv \)
- Angular momentum is \( L = I(t) \omega \)
- Or we can go the other way: \( \omega = I(t)^{-1}L \)

Getting to \( \omega \)

- Recall \( v_i - V = \omega \times (x_i - X) \)
- Plug this into angular momentum:
  \[
  L = \sum_i m_i (x_i - X) \times (\omega \times (x_i - X))
  = -\sum_i m_i (x_i - X) \times ((x_i - X) \times \omega)
  = -\sum_i m_i (x_i - X)^\ast (x_i - X)^\ast \omega
  = \left( \sum_i m_i (x_i - X)^\ast (x_i - X)^\ast \right) \omega
  = (I(t)) \omega
  \]

Equations of Motion

\[
\frac{d}{dt} V = \frac{F}{M} \quad \frac{d}{dt} L = T
\]
\[
\frac{d}{dt} X = V \quad \omega = I(t)^{-1} L
\]
\[
\frac{d}{dt} R = \omega \times R
\]

In the absence of external forces \( F=0, T=0 \)
Reminder

• Before going on:
• Remember that this all boils down to particles
  • Mass, position, velocity, (linear) momentum, force are fundamental
  • Inertia tensor, orientation, angular velocity, angular momentum, torque are just abstractions
• Don’t get too puzzled about interpretation of torque for example: it’s just a mathematical convenience

Inertia Tensor Simplified

• Reduce expense of calculating $I(t)$:
  $$I(t) = \sum_i m_i (x_i - X)^T (x_i - X)$$
  $$= \sum_i m_i \left[ (x_i - X)^T (x_i - X) \delta - (x_i - X)(x_i - X)^T \right]$$
  • Now use $x_i - X = Rp_i$ and use $R^T R = \delta$
  $$I(t) = \sum_i m_i \left[ p_i^T R^T Rp_i \delta - Rp_i p_i^T R^T \right]$$
  $$= R \left( \sum_i m_i \left[ p_i^T p_i \delta - p_i p_i^T \right] \right)^T R^T$$

Inertia Tensor Simplified 2

• So just compute inertia tensor once, for object space configuration
  • Then $I(t) = RI_{body}R^T$
  • And $I(t) = R(I_{body})^{-1}R^T$
    • So precompute inverse too
  • In fact, since $I$ is symmetric, know we have an orthogonal eigenbasis $Q$
  • Rotate object-space orientation by $Q$
    • Then $I_{body}$ is just diagonal!

Degenerate Inertia Tensors

• $I$ is just sum of symmetric positive semi-definite matrices
  • Each one has null space: vectors parallel to $x_i - X$
  • If all the points line up (object is a rod) then sum $I$ has the same null space
    • Singular: cannot be inverted
    • We don’t care though, since we can’t track rotation around that axis anyways
    • So diagonalize $I$, and only invert nonzero elements
• Similarly for a single point…
Taking the limit

• Letting our decomposition of the object into point masses go to infinity:
  • Instead of sum over particles, integral over object volume
  • Instead of particle mass, density at that point in space
  \[ \sum m_i \rho_0(x_i) \rightarrow \iiint \rho(x) \rho_0(x) dx \]
• No big deal

Computing Inertia Tensors

• Do the integrals: \[ I_{\text{body}} = \iiint p(p^T p \delta - pp^T) dp \]
• Lots of fun!
• You may want to look them up instead
  • E.g. Eric Weisstein’s World of Science on the web
• Align axis perpendicular to planes of symmetry (of \( \rho \) in object space
  • Guarantees some off-diagonal zeros
• Example: sphere, uniform density, radius \( R \)
  \[
  \begin{pmatrix}
  \frac{2}{5} MR^2 & 0 & 0 \\
  0 & \frac{2}{5} MR^2 & 0 \\
  0 & 0 & \frac{2}{5} MR^2
  \end{pmatrix}
  \]

Approximating Inertia Tensors

• For complicated geometry, don’t really need exact answer
• Instead use numerical quadrature
  • If we can afford to spend a lot of time precomputing, life is simple
  • Simplest approach: Monte-Carlo
    • Obviously stratified sampling etc. helps

Combining Objects

• What if object is union of two simpler objects?
• Integrals are additive
  • But be careful about adding \( I_1(t) + I_2(t) \):
    • World-space formulas \((x-X)\) use the \( X \) for the object: \( X_1 \) and \( X_2 \) may be different
    • Simplified \( I_{\text{body}} \) formula based on having centre of mass at origin
  • Let’s work it out from the integral of \( I(t) \)
• Combined mass: \( M = M_1 + M_2 \)
• Centre of mass of combined object:
  \[
  X = \frac{\int_{\Omega_1 \cup \Omega_2} \rho x \\ 
  \int_{\Omega_1 \cup \Omega_2} \rho}
  = \frac{M_1 X_1 + M_2 X_2}{M}
  \]
Combined Inertia Tensor

\[ I(t) = \int_{\Omega_1 \cup \Omega_2} \rho(x-X)^T (x-X)^T \]
\[ = \int_{\Omega_1} \rho(x-X + X_1 - X)^T (x-X + X_1 - X)^T + \int_{\Omega_2} \rho(x-X)^T (x-X)^T \]
\[ = \int_{\Omega_1} \rho(x-X_1)^T (x-X_1)^* + \int_{\Omega_1} \rho(x-X)^T (x-X)^* \]
\[ + \int_{\Omega_2} \rho(x-X_1)^T (x-X_1)^* + \int_{\Omega_2} \rho(x-X)^T (x-X)^* \]
\[ = I_1(t) + (X_1 - X)^T \underbrace{\int_{\Omega_1} \rho(x-X_1)^*}_0 + \int_{\Omega_1} \rho(x-X)^T (X_1 - X)^* \]
\[ + M_1(X_1 - X)^T (X_1 - X)^* + \int_{\Omega_2} \]
\[ = I_1(t) + M_1(X_1 - X)^T (X_1 - X)^* + I_2(t) + M_2(X_2 - X)^T (X_2 - X)^* \]

Numerical Method

- For advancing V and X, can use any of the second order schemes we discussed before
  - Often only gravity and small amount of wind drag
- For advancing angular stuff:
  - Constraint on R makes life a little more interesting

Advancing angular stuff

- Symplectic Euler-like algorithm simplest choice: \( L_{n+1} = L_n + \Delta tT \)
  \( \omega_{n+1} = I(t_n)^{-1} L_{n+1} \)
  \( R_{n+1} = R_n + \Delta t \omega_{n+1}^* R_n \)
- Note: updated R isn't quite orthogonal
- Need to correct (otherwise objects inflate)
- Simplest choice: Gram-Schmidt
  - But introduces axis-bias, and expensive
- Could also compute rotation matrix for \( \Delta t \omega \)
  - Even more expensive, still have some drift

Stability? Accuracy?

- Note R cannot blow up (we keep making it orthogonal)
- But if \( T=T(R, \omega) \) there is potential for L and \( \omega \) to blow up
  - Rarely the case (usually \( T=0 \), apart from isolated collision impulses)
  - If it is the case, can go implicit
- May want to restrict \( \Delta t=O(\omega^{-1}) \) to properly sample rotations
Improving on R

- Expensive (and maybe biased) to keep R orthogonal
  - 9 numbers for 3 parameters
  - Use a less redundant representation
- Quaternions work better!
  - Still cheap and easy to deal with (unlike Euler angles, for example)
  - Only 4 numbers - still need to normalize
  - But can do it without axis bias
  - and for much cheaper

Review quaternions

- Instead of R, use \( q=(s,x,y,z) \) with \( |q|=1 \)
  - Can think of \( q=s+xi+yj+zk \)
  - \( i^2=j^2=k^2=1, ij=-ji=k, jk=-kj=i, ki=-ik=j \)
  - Don’t commute! \( q_1q_2\neq q_2q_1 \)
- Represents “half” a rotation:
  - \( q=\cos(\theta/2) \)
  - \( lx,y,zl^2=\sin^2(\theta/2) \)
  - Axis of rotation is \( (x,y,z) \)
- Conjugate (inverse for unit norm) is

\[
\bar{q} = (s,-x,-y,-z)
\]

Rotating with quaternions

- Instead of \( Rp \), calculate \( q(0,p)\bar{q} \)
- Composing a rotation of \( \Delta t\omega \) to advance a time step:
  \[
  q_{n+1} = \left( \sqrt{1 - \left( \Delta t \frac{\omega}{2} \right)^2}, \Delta t \frac{\omega}{2} \right) q_n
  \]
- For small \( \Delta t\omega \) approximate:
  \[
  q_{n+1} = \left( 1, \Delta t \frac{\omega}{2} \right) q_n = q_n + \Delta t \frac{\omega}{2} q_n
  \]
- From this get the differential equation:
  \[
  \dot{q} = \frac{1}{2} \omega q
  \]

Converting q to R

- Clearly superior to use quaternions for storing and updating orientation
- But, slightly faster to transform points with rotation matrix
- If you need to transform a lot of points (collision detection….) may want to convert q into R
- Basic idea: columns of R are rotated axes \( R(1,0,0)^T, R(0,1,0)^T \), and \( R(0,0,1)^T \)
- Do the rotation with q instead.
  - Can simplify and optimize for the zeros - look it up