Notes

- ppmtompeg: making animations
- Please read D. Baraff, "Linear time dynamics…", SIGGRAPH'96
- Also see Witkin and Baraff course notes on physics-based modeling
- Homework 1 - I will try to get it back today.

Constrained Dynamics

- We thought of rigid bodies as a multitude of particles, with constraints that interparticle distances remained constant
- How do we apply more general constraints to dynamics?
  - Bead on a wire
  - Articulated rigid bodies
  - Gears
  - Character interaction

Three major approaches

- "Soft" constraint forces
  - Like repulsions
- Add unknown constraint forces (lagrange multipliers)
  - Closely related: projection methods
- Solve in terms of reduced number of degrees of freedom (generalized coordinates)

Before constraints

- We have a (long) vector x of positions (maybe also orientations)
- We have a (long) vector v of velocities (maybe also angular velocities)
- A matrix M with masses down the diagonal (maybe also inertia tensors)
- A (long) vector F of forces (maybe also torques)

$$\dot{v} = M^{-1}F$$

$$\dot{x} = v$$

Equality constraints

- Generally, want motion to satisfy C(x,v)=0
  - C is a vector of constraints
- Want motion to be natural F=ma, except that constraints are also satisfied
  - We don't want to have to model exactly why they are satisfied in reality…
- We will need to add forces to the simulation that cause it to satisfy (approximately) C(x,v)=0

Inequalities

- Not going to cover inequality constraints C(x,v)≥0
  - Gets into heavy-duty optimization: have to figure out which constraints are "active" etc.
  - Can be NP-hard
- These can be used for contact modeling, friction, joint limits, …
  - But we can approximate by
    - apply corrective impulse when inequality violated,
    - iterate to check on other constraints,
    - and other tricks to handle complex stuff

Soft Constraints

- First assume C=C(x)
  - No velocity dependence
- We won't exactly satisfy constraint, but will add some force to not stray too far
  - Just like repulsion forces for contact/collision
- First try:
  - define a potential energy minimized when C(x)=0
  - C(x) might already fit the bill, if not use $E = \frac{1}{2} K C^T C$
- [example: nailed point]

Potential force

- We'll use the gradient of the potential as a force: $F = -\left(\frac{\partial E}{\partial x}\right)^T = -K\left(\frac{\partial C}{\partial x}\right)^T C$
- [example: nailed point]
- This is just a generalized spring pulling the system back to constraint
- But what do undamped springs do?

Rayleigh Damping

- Need to add damping force that doesn't damp valid motion of the system
- Rayleigh damping:
  - Damping force proportional to the negative rate of change of C(x)
    - No damping valid motions that don't change C(x)
  - Damping force parallel to elastic force
    - This is exactly what we want to damp

$$F_d = -D\left(\frac{\partial C}{\partial x}\right)^T \dot{C} = -D\left(\frac{\partial C}{\partial x}\right)^T \frac{\partial C}{\partial x} v$$

Issues

- Need to pick K and D
  - Don't want oscillation - critical damping
  - If K and D are very large, could be expensive (especially if C is nonlinear)
  - If K and D are too small, constraint will be grossly violated
- Big issue: the more the applied forces try to violate constraint, the more it is violated…
  - Ideally want K and D to be a function of the applied forces

Pseudo-time Stepping

- Alternative: simulate all the applied force dynamics for a time step
- Then simulate soft constraints in pseudo-time
  - No other forces at work, just the constraints
  - "Real" time is not advanced
  - Keep going until at equilibrium
  - Non-conflicting constraints will be satisfied
  - Balance found between conflicting constraints
  - Doesn't really matter how big K and D are (adjust the pseudo-time steps accordingly)

Issues

- Still can be slow
  - Particularly if there are lots of adjoining constraints
- Could be improved with implicit time steps
  - Get to equilibrium as fast as possible…
- This will come up again…

Constraint forces

- Idea: constraints will be satisfied because $F_{total}=F_{applied}+F_{constraint}$
- Have to decide on form for $F_{constraint}$
- [example: y=0]
- We have too much freedom…
- Need to specify the problem better

Virtual work

- Assume for now C=C(x)
- Require that all the (real) work done in the system is by the applied forces
  - The constraint forces do no work
- Work is $F_c \bullet \Delta x$
  - So pick the constraint forces to be perpendicular to all valid velocities
  - The valid velocities are along isocontours of C(x)
  - Perpendicular to them is the gradient: $\frac{\partial C^T}{\partial x}$
- So we take $F_c = \left(\frac{\partial C}{\partial x}\right)^T \lambda$

What is λ?

- Say C(x)=0 at start, want it to remain 0
- Take derivative: $\dot{C}(x) = \frac{\partial C}{\partial x}\dot{x} = \frac{\partial C}{\partial x}v = 0$
- Take another to get to accelerations

$$\ddot{C}(x) = \frac{\partial \dot{C}}{\partial x}\dot{x} + \frac{\partial \dot{C}}{\partial v}\dot{v} = \frac{\partial \dot{C}}{\partial x}v + \frac{\partial C}{\partial x}\dot{v} = 0$$

- Plug in F=ma, set equal to 0

$$\frac{\partial \dot{C}}{\partial x}v + \frac{\partial C}{\partial x}\left(M^{-1}\left(F_a + F_c\right)\right) = 0$$

Finding constraint forces

- Rearranging gives:

$$\frac{\partial C}{\partial x}M^{-1}F_c = -\frac{\partial C}{\partial x}M^{-1}F_a - \frac{\partial \dot{C}}{\partial x}v$$

- Plug in the form we chose for constraint force:

$$\left(\frac{\partial C}{\partial x}M^{-1}\frac{\partial C^T}{\partial x}\right)\lambda = -\frac{\partial C}{\partial x}M^{-1}F_a - \frac{\partial \dot{C}}{\partial x}v$$

- Note: SPD matrix!

Modified equations of motion

- So can write down (exact) differential equations of motion with constraint force
- Could run our standard solvers on it
- Problem: drift
  - We make numerical errors, both in the regular dynamics and the constraints!
- We'll just add "stabilization": additional soft constraint forces to keep us from going too far
  - Don't worry about K and D in this context!
  - Don't include them in formula for λ

Generalizing constraints

- How do we handle C(x,v)?
- Principle of virtual work, isocontours of C, etc. gets a little difficult to interpret!
- Instead look for constraint forces that cause us to satisfy constraints AND are the "smallest" of all such possible forces
  - If we were to apply "larger" constraint forces, they must be doing something beyond satisfying the constraint - messing with the real dynamics
- The key question: how to define "smallest"

Energy norm

- The "right" norm to choose is the same as the one used to measure kinetic energy: $|v|_E^2 = \frac{1}{2} v^T M v$
- So look for constraint forces that minimize energy-norm of acceleration:

$$\min \tfrac{1}{2} a^T M a = \min \tfrac{1}{2} F_c^T M^{-1} F_c$$

The constraint

- Take time derivative of C(x,v)=0 once to get accelerations → forces

$$\frac{\partial C}{\partial x} \dot{x} + \frac{\partial C}{\partial v} \dot{v} = 0$$

$$\frac{\partial C}{\partial x} v + \frac{\partial C}{\partial v} M^{-1} F = 0$$

- Rearranging, splitting F into appl/constr:

$$\frac{\partial C}{\partial v} M^{-1} F_c = -\frac{\partial C}{\partial v} M^{-1} F_a - \frac{\partial C}{\partial x} v$$

J notation

- Both from C(x)=0 and two time derivatives, and C(x,v)=0 and one time derivative, get constraint force equation:

$$JM^{-1}F_c = -JM^{-1}F_a - c$$

  (J is for Jacobian)
- Before we used $F_c = J^T\lambda$
- This gives SPD system for λ: $JM^{-1}J^T\lambda = b$
- General family of solutions is $F_c = J^T\lambda + Ms$ where $Js=0$

Which solution?

- So take the energy-norm of the general solution:

$$\begin{aligned}\tfrac{1}{2}F_c^T M^{-1}F_c &= \tfrac{1}{2}\left(J^T\lambda + Ms\right)^T M^{-1}\left(J^T\lambda + Ms\right)\\ &= \tfrac{1}{2}\lambda^T JM^{-1}J^T\lambda + \tfrac{1}{2}s^T MM^{-1}Ms + \lambda^T JM^{-1}Ms\\ &= \tfrac{1}{2}\lambda^T JM^{-1}J^T\lambda + \tfrac{1}{2}s^T Ms + 0\end{aligned}$$

- Clearly we should take s=0, so indeed, $F_c = J^T\lambda$

Discrete projection method

- It's a little ugly to have to add even more stuff for dealing with drift - and still isn't exactly on constraint
- Instead go to discrete view (treat numerical errors as applied forces too)
- After a time step (or a few), calculate constraint impulse to get us back
  - Similar to what we did with collision and contact
- Can still have soft or regular constraint forces for better accuracy…

The algorithm

- Time integration takes us over Δt from (x_n, v_n) to $(x_{n+1}{}^*, v_{n+1}{}^*)$
- We want to add an impulse
  $v_{n+1} = v_{n+1}{}^* + M^{-1}i$
  $x_{n+1} = x_{n+1}{}^* + \Delta t\, M^{-1}i$
  such that new x and v satisfy constraint: $C(x_{n+1}, v_{n+1})=0$
- In general C is nonlinear: difficult to solve
  - But if we're not too far from constraint, can linearize and still be accurate

The constraint impulse

$$0 = C(x_{n+1}, v_{n+1}) \approx C(x^*_{n+1}, v^*_{n+1}) + \left.\frac{\partial C}{\partial x}\right|^*_{n+1} \Delta x + \left.\frac{\partial C}{\partial v}\right|^*_{n+1} \Delta v$$

- Plug in changes in x and v:

$$\Delta t \frac{\partial C}{\partial x} M^{-1} i + \frac{\partial C}{\partial v} M^{-1} i = -C^*_{n+1}$$

$$\left(\Delta t \frac{\partial C}{\partial x} + \frac{\partial C}{\partial v}\right) M^{-1} i = -C^*_{n+1}$$

- As before, minimize energy norm of Δv:

$$i = J^T \lambda \qquad \text{where} \qquad J = \Delta t \frac{\partial C}{\partial x} + \frac{\partial C}{\partial v}$$

Projection

- We're solving $JM^{-1}J^T\lambda = -C$
  - Same matrix again - particularly in limit
- In case where C is linear, we actually are projecting out part of motion that violates the constraint
  - Foreshadowing: incompressibility

Nonlinear C

- We can accept we won't exactly get back to constraint
  - But notice we don't drift too badly: every time step we do try to get back the entire way
- Or we can iterate, just like Newton
  - Keep applying corrective impulses until close enough to satisfying constraint
- This is very much like running soft constraint forces in pseudo-time with implicit steps, except now we know exactly the best parameters

Solving SPD systems

- Before in implicit methods, systems to solve were small
  - Particles didn't interact, so just 3x3…
- Now: constraints may interact, may have a large system to solve
  - But it's SPD
- If constraints have a special form, may be able to invert matrix very efficiently (cf Baraff)
- But in general, Conjugate Gradient method is the natural choice

Conjugate Gradient

- For solving Ax=b with A an SPD matrix
  - Actually, A an SPD operator: CG doesn't care if it's a simple matrix or not, as long as you can calculate Ap for any vector p
  - This is useful when J is sparse, M^{-1} is sparse, but $JM^{-1}J^T$ isn't nearly as sparse,
    or we don't explicitly know J (just the sum over different constraints)
- Basic idea:
  - Start with initial guess
  - Measure residual
  - Add correction to minimize error, repeat

CG algorithm

- r=b-Ax
- $\rho=r^Tr$, check if already solved
- p=r
- Loop:
  - q=Ap
  - $\alpha= \rho/(p^Tq)$
  - x+= αp, r-= αq
  - $\rho_{new}=r^Tr$, check for convergence
  - $\beta= \rho_{new} /\rho$
  - p=r+ βp
  - $\rho=\rho_{new}$

Next class

- The classical approach to (some) constraints:
  - Parameterize the constrained system so you can't even describe invalid states
  - Drift is impossible