# Notes

- Assignment 2 instability - don't worry about it right now
- Please read
  - D. Baraff, "Fast contact force computation for nonpenetrating rigid bodies", SIGGRAPH '94
  - D. Baraff, "Linear-time dynamics using Lagrange multipliers", SIGGRAPH '96



# Rigid Collision Algorithms

- Use the same collision response algorithm as with particles
  - Identify colliding points as perhaps the deepest penetrating points, or the first points to collide
  - Make sure they are colliding, not separating!
- Problem: multiple contact points
  - Fixing one at a time can cause rattling.
  - Can fix by being more gentle in resolving contacts - negative coefficient of restitution
- Problem: multiple collisions (stacks)
  - Fixing one penetration causes others
  - Solve either by resolving simultaneously or enforcing order of resolution



# Stacking

- Guendelman et al. "shock propagation"
- After applying contact impulses (but penetrations remain)
  - Form contact graph: "who is resting on whom"
    - Check new position of each object against the other objects' old positions --- any penetrations indicate a directed edge
  - Find "bottom-up" ordering: order fixed objects such as the ground first, then follow edges
    - Union loops into a single group
  - Fix penetrations in order, freezing objects after they are fixed
    - Slight improvement: combine objects into a single composite rigid body rather than simply freezing



# Stacking continued

- Advantages:
  - Simple, fast
- Problems:
  - Overly stable sometimes
  - Doesn't really help with loops
- To resolve problems, need to really solve the global contact problem (not just at single contact points)



# The Contact Problem

- See e.g. Baraff "Fast contact force computation…", SIGGRAPH'94
- Identify all contact points
  - Where bodies are close enough
- For each contact point, find relative velocity as a (linear) function of contact impulses
  - Just as we did for pairs
- Frictionless contact problem:
  - Find normal contact impulses that cause normal relative velocities to be non-negative
  - Subject to constraint: contact impulse is zero if normal relative velocity is positive
  - Called a linear complementary problem (LCP)



# Frictional Contact Problem

- Include tangential contact impulses
  - Either relative velocity is zero (static friction) or tangential impulse is on the friction cone
  - By approximating the friction cone with planar facets, can reduce to LCP again
- Note: modeling issue - the closer to the true friction cone you get, the more variables and equations in the LCP

# Constrained Dynamics



# Constrained Dynamics

- We just dealt with one constraint: rigid motion only
- More general constraints on motion are useful too
  - E.g. articulated rigid bodies, gears, scripting part of the motion, …
- Same basic issue: modeling the constraint forces
  - Principle of virtual work - constraint forces shouldn't influence the unconstrained part of the motion



# Three major approaches

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



# Equality constraints

- Generally, want motion to satisfy C(x,v)=0
  - C is a vector of constraints
- Inequalities also possible - C(x,v)≥0 - but let's ignore for now
  - Generalizes notion of contact forces
  - Also can do things like joint limits, etc.
  - Generally need to solve with heavy-duty optimization, may run into NP-hard problems
  - One approach: figure out or guess which constraints are "active" (equalities) and just do regular equality constraints, maybe iterating



# 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 $E = \frac{1}{2}KC^T C$
  - Just like hyper-elasticity!



# Potential force

- We'll use the gradient of the potential as a force: $F = -\left(\frac{\partial E}{\partial x}\right)^T = -K\left(\frac{\partial C}{\partial x}\right)^T C$
- 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 $F_c \cdot \Delta x$
  - 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:

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

- 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 - this is post-processing to correct drift



# Velocity constraints

- How do we handle C(v)=0?
- Take time derivative as before: $\frac{\partial C}{\partial v}\dot{v} = 0$
- And again apply principle of virtual work just like before: $F_c = \left(\frac{\partial C}{\partial v}\right)^T \lambda$
- And end up solving:

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



# J notation

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

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

  (J is for Jacobian)
- We assume $F_c = J^T \lambda$
- This gives SPD system for λ: $JM^{-1}J^T \lambda = b$



# 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 $\Delta t$ 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:

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

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

- Using principle of virtual work:

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



# 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



# 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