Notes

Numerical Implementation 1

- Get candidate x(t+Δt)
- Check to see if x(t+Δt) is inside object (interference)
- If so
  - Get normal n at t+Δt
  - Get new velocity v from collision response formulas and average v
  - Replay x(t+Δt)=x(t)+Δtv

Robustness?

- If a particle penetrates an object at end of candidate time step, we fix that
- But new position (after collision processing) could penetrate another object!
- Maybe this is fine-let it go until next time step
- But then collision formulas are on shaky ground…
- Switch to repulsion impulse if x(t) and x(t+Δt) both penetrate
  - Find Δv_N proportional to final penetration depth, apply friction as usual

Making it more robust

- Other alternative:
  - After collision, check if new x(t+Δt) also penetrates
  - If so, assume a 2nd collision happened during the time step: process that one
  - Check again, repeat until no penetration
  - To avoid infinite loop make sure you lose kinetic energy (don't take perfectly elastic bounces, at least not after first time through)
  - Let's write that down:

Numerical Implementation 2

- Get candidate x(t+Δt)
- While x(t+Δt) is inside object (interference)
  - Get normal n at t+Δt
  - Get new velocity v from collision response formulas and average v
  - Replay x(t+Δt)=x(t) + Δt v
- Now can guarantee that if we start outside objects, we end up outside objects

Micro-Collisions

- These are "micro-collision" algorithms
- Contact is modeled as a sequence of small collisions
  - We're replacing a continuous contact force with a sequence of collision impulses
- Is this a good idea?
  - [block on incline example]
- More philosophical question: how can contact possibly begin without fully inelastic collision?

Improving Micro-Collisions

- Really need to treat contact and collision differently, even if we use the same friction formulas
- Idea:
  - Collision occurs at start of time step
  - Contact occurs during whole duration of time step

Numerical Implementation 3

- Start at x(t) with velocity v(t), get candidate position x(t+Δt)
- Check if x(t+Δt) penetrates object
  - If so, process **elastic collision** using v(t) from start of step, **not** average velocity
  - Replay from x(t) with modified v(t)
  - Could add ΔtΔv to x(t+Δt) instead of re-integrating
  - Repeat check a few (e.g. 3) times if you want
- While x(t+Δt) penetrates object
  - Process **inelastic contact** (ε=0) using **average** v
  - Replay x(t+Δt)=x(t)+Δt v

Why does this work?

- If object resting on plane y=0, v(t)=0 though gravity will pull it down by t+Δt
- In the new algorithm, elastic bounce works with pre-gravity velocity v(t)=0
  - So no bounce
- Then contact, which is inelastic, simply adds just enough Δv to get back to v(t+Δt)=0
  - Then x(t+Δt)=0 too
- NOTE: if ε=0 anyways, no point in doing special first step - this algorithm is equivalent to the previous one

Moving objects

- Same algorithms, and almost same formulas:
  - Need to look at relative velocity $v_{particle}-v_{object}$ instead of just particle velocity
  - As before, decompose into normal and tangential parts, process the collision, and reassemble a relative velocity
  - Add object velocity to relative velocity to get final particle velocity
- Be careful when particles collide:
  - Same relative Δv but account for equal and opposite forces/impulses with different masses…

Moving Objects…

- Also, be careful with interference/collision detection
  - Want to check for interference at end of time step, so use object positions there
  - Objects moving during time step mean more complicated trajectory intersection for collisions

Collision Detection

- We have basic time integration for particles in place now
- Assumed we could just do interference detection, but…
- Detecting collisions over particle trajectories can be dropped in for more robustness - algorithms don't change
  - But use the normal at the collision time

Geometry

- The plane is easy
  - Interference: y<0
  - Collision: y became negative
  - Normal: constant (0,1,0)
- Can work out other analytic cases (e.g. sphere)
- More generally: triangle meshes and level sets
  - Heightfields sometimes useful - permit a few simplifications in speeding up tests - but special case
  - Splines and subdivision surfaces generally too complicated, and not worth the effort
  - Blobbies, metaballs, and other implicits are usually not as well behaved as level sets
  - Point-set surfaces: becoming a hot topic

Implicit Surfaces

- Define surface as where some scalar function of x,y,z is zero:
  - {x,y,z | F(x,y,z)=0}
- Interior (can only do closed surfaces!) is where function is negative
  - {x,y,z | F(x,y,z)<0}
- Outside is where it's positive
  - {x,y,z | F(x,y,z)>0}
- Ground is F=y
- Example: $F=x^2+y^2+z^2-1$ is the unit sphere

Testing Implicit Surfaces

- Interference is simple:
  - Is F(x,y,z)<0?
- Collision is a little trickier:
  - Assume constant velocity x(t+h)=x(t)+hv
  - Then solve for h: F(x(t+h))=0
  - This is the same as ray-tracing implicit surfaces…
  - But if moving, then need to solve F(x(t+h), t+h)=0
  - Try to bound when collision can occur (find a sign change in F) then use secant search

Implicit Surface Normals

- Outward normal at surface is just $n = \frac{\nabla F}{|\nabla F|}$
- Most obvious thing to use for normal at a point inside the object (or anywhere in space) is the same formula
  - Gradient is steepest-descent direction, so hopefully points to closest spot on surface: direction to closest surface point is parallel to normal there
  - We really want the implicit function to be monotone as we move towards/away from the surface

Building Implicit Surfaces

- Planes and spheres are useful, but want to be able to represent (approximate) any object
- Obviously can write down any sort of functions, but want better control
  - Exercise: write down functions for some common shapes (e.g. cylinder?)
- Constructive Solid Geometry (CSG)
  - Look at set operations on two objects
    - [Complement, Union, Intersection, …]
  - Using primitive F()'s, build up one massive F()
  - But only sharp edges…

Getting back to particles

- "Metaballs", "blobbies", …
- Take your particle system, and write an implicit function:

$$F(x) = \sum_i \alpha_i f\left(\frac{|x - x_i|}{r_i}\right) - t$$

  - Kernel function f is something smooth like a Gaussian $f(x) = e^{-x^2}$
  - Strength α and radius r of each particle (and its position x) are up to you
  - Threshold t is also up to you (controls how thick the object is)

Problems with these

- They work beautifully for some things!
  - Some machine parts, water droplets, goo, …
- But, the more complex the surface, the more expensive F() is to evaluate
  - Need to get into more complicated data structures to speed up to acceptable
- Hard to directly approximate any given geometry
- Monotonicity - how reliable is the normal?

Signed Distance

- Note infinitely many different F represent the same surface
- What's the nicest F we can pick?
- Obviously want smooth enough for gradient (almost everywhere)
- It would be nice if gradient really did point to closest point on surface
- Really nice (for repulsions etc.) if value indicated how far from surface
- The answer: signed distance

Defining Signed Distance

- Generally use the letter ϕ instead of F
- Magnitude $|\phi(x)|$ is the distance from the surface
  - Note that function is zero only at surface
- Sign of $\phi(x)$ indicates inside (<0) or outside(>0)
- [examples: plane, sphere, 1d]

Closest Point Property

- Gradient is steepest-ascent direction
  - Therefore, in direction of closest point on surface (shortest distance between two points is a straight line)
- The closest point is by definition distance $|\phi|$ away
- So closest point on surface from x is

$$x - \phi(x)\frac{\nabla\phi}{|\nabla\phi|}$$

Unit Gradient Property

- Look along line from closest point on surface to x
- Value is distance along line
- Therefore directional derivative is 1:

$$\nabla\phi \cdot n = 1$$

- But plug in the formula for n [work out]
- So gradient is unit length: $|\nabla\phi| = 1$

Aside: Eikonal equation

- There's a PDE! $|\nabla\phi| = 1$
  - Called the Eikonal equation
  - Important for all sorts of things
  - Later in the course: figure out signed distance function by solving the PDE…
- See Ian Mitchell's course on level sets for a lot more detail

Aside: Spherical particles

- We have been assuming our particles were just points
- With signed distance, can simulate nonzero radius spheres
  - Sphere of radius r intersects object if and only if $\phi(x)<r$
  - i.e. if and only if $\phi(x)-r<0$
  - So looks just like points and an "expanded" version of the original implicit surface - normals are exactly the same, …

Level Sets

- Use a discretized approximation of ϕ
- Instead of carrying around an exact formula store samples of ϕ on a grid (or other structure)
- Interpolate between grid points to get full definition (fast to evaluate!)
  - Almost always use trilinear [work out]
- If the grid is fine enough, can approximate any well-behaved closed surface
  - But if the features of the geometry are the same size as the grid spacing or smaller, expect BAD behaviour
- Note that properties of signed distance only hold approximately!

Building Level Sets

- We'll get into level sets more later on
  - Lots of tools for constructing them from other representations, for sculpting them directly, or simulating them…
- For now: can assume given
- Or CSG: union and intersection with min and max
  [show 1d]
  - Just do it grid point by grid point
  - Note that weird stuff could happen at sub-grid resolution (with trilinear interpolation)
- Or evaluate from analytical formula

Normals

- We do have a function F defined everywhere (with interpolation)
  - Could take its gradient and normalize
  - But (with trilinear) it's not smooth enough
- Instead use numerical approximation for gradient:

$$g_{i,j,k} = \left(\frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2\Delta x}, \frac{\phi_{i,j+1,k} - \phi_{i,j-1,k}}{2\Delta y}, \frac{\phi_{i,j,k+1} - \phi_{i,j,k-1}}{2\Delta z} \right)$$

  - Then, use trilinear interpolation to get (continuous) approximate gradient anywhere
  - Or instead apply finite difference formula to 6 trilinearly interpolated points (mathematically equivalent)
  - Normalize to get unit-length normal

Evaluating outside the grid

- Check if evaluation point x is outside the grid
- If outside - that's enough for interference test
- But repulsion forces etc. may need an actual value
- Most reasonable extrapolation:
  - A = distance to closest point on grid
  - B = ϕ at that point
  - Lower bound on distance, correct asymptotically and continuous (if level set doesn't come to boundary of grid):

$$\operatorname{sign}(B)\sqrt{A^2 + B^2}$$

  - Or upper bound on distance:

$$B + \operatorname{sign}(B)A$$