Notes

---

Heightfields

- Especially good for terrain - just need a 2d array of heights (maybe stored as an image)
  - Displacement map from a plane
- Split up plane into triangles
- Particle inside:
  - Figure out which triangle (x,y) belongs to, check z against equation of triangle's plane
- Trajectory cross (stationary heightfield):
  - Check all triangles along path (use 2d line-drawing algorithm to figure out which cells to check)
- Object normal: get from triangle
- Distance etc.: not so easy, but vertical distance easy for shallow heightfields

Triangle mesh

- For any decent size, need to use an acceleration structure
  - Could use background (hash-)grid, octree, kd-tree
  - Also can use bounding volume (BV) hierarchy
    - Spheres, axis-aligned bounding boxes, oriented bounding boxes, polytopes, ...
  - More exotic structures exist...
- Particle inside (closed mesh):
  - Shoot a ray out to infinity, count the number of crossings
- Trajectory cross (stationary mesh):
  - For each candidate triangle (from acceleration) check a sequence of determinants

Triangle intersection

- Many, many ways to do this
- Most robust (and one of the fastest) is to do it based on determinants
  - For vectors a,b,c define $\det(a,b,c) = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix}$ ($= a \times b \cdot c$)
  - $\det(a,b,c) = \pm 6$ volume($\text{tet}(a,b,c)$), the signed volume of the tetrahedron spanned by edges a,b,c from a common point
  - Sign flips when tetrahedron reflected, or alternatively from right-hand-rule on a$\times$b$\cdot$c
- Triangle intersection boils down to
  - 2 sign checks: segment crosses plane
  - 3 sign checks: line goes through triangle
**Triangle Mesh (more)**

- **Object normal**
  - Normalize cross-product of two sides of the triangle
- **Distance from single triangle**
  - Find barycentric coordinates — solve a least-squares problem
  - Need to clip to sides of triangle
  - Compute distance from that point
  - Note: also gives direction to closest point
- **Distance (and direction) from mesh**
  - Compute for all possible triangles, take minimum
  - Trick is to find small list of possible triangles with acceleration structure

**Implicit Surface**

- **Simple function, metaballs, or interpolated from 3d grid ("level set")**
  - Recall - for metaballs need acceleration
- **Particle inside:** \( f(x) < 0 \)
- **Trajectory cross:**
  - Just like ray-tracing - use secant method
- **Object normal:** \( \nabla f / |\nabla f| \)
- **Distance from surface:**
  - If \( f() \) is signed distance, then trivial
  - Otherwise, painful, but \( f() \) might be good enough for application

**Back to particle collisions**

- So now we can represent other geometry, how do we do a repulsion velocity field?
  - \( v(x) = f(\text{distance}(x)) \times n(x) \)
  - \( n(x) \) is the outward direction (normal on surface)
  - \( f \) is some decreasing function that drops towards zero far away
    - Exponential: \( f(d) = e^{-k \cdot d} \)
    - Or linear drop, truncated to zero: \( f(d) = \max(0, m - k \cdot d) \)
    - Or more complicated
  - Outward direction is plus/minus direction to closest point
  - Aside: useful for more than just collisions - e.g. fire particles streaming out of an object

**Force-based repulsions**

- Can do exactly the same trick for force-based motion
  - Add repulsion field to \( F(x) \)
- Simple, often works, but there are sometimes problems
  - What are you trying to model?
  - Robustness - high velocity impacts can penetrate arbitrarily far
    - High velocity impacts may go straight through thin objects
  - How much of a rebound do you want?
Damped repulsions

- Think of repulsion force as a generalized spring
- Add spring damping:
  \[ F_{\text{damp}} = -D(v \cdot n(x))n(x) \]
  - \( D \) is some parameter you set
  - \( n(x) \) is the outward direction again

Aside: springs and damping

- How do you come up with reasonable values for spring constants and damping constants?
  - And how do you pick good step sizes for differential equation solver (Forward Euler etc.)
- Look at 1D simplified model
  \[ Ma = F = -Kx - Dv \]
  - \( M \) is the mass, \( K \) is like a spring stiffness, \( D \) is the damping parameter
- Solve it analytically

Critical Damping

- Three cases:
  - Underdamped \( (D^2 - 4MK < 0) \)
    - Oscillation with frequency \( \omega \sim \frac{1}{2} \sqrt{K/M} \)
    - Characteristic time: \( t \sim 2\pi \sqrt{M/K} \)
    - Exponentially decays at rate \( r = -D/(2M) \)
    - Characteristic time: \( t \sim 2M/D \)
  - Overdamped \( (D^2 - 4MK > 0) \)
    - No continued oscillation
    - Exponentially decays at rates \( r \sim -K/D, -D/M \)
    - Characteristic times: \( t \sim D/K, M/D \)
  - Critically damped \( (D^2 - 4MK = 0) \)
    \( D = 2\sqrt{MK} \)
    - No continued oscillation
    - Fastest decay possible at rate \( r = -D/(2M) \)
    - Characteristic time: \( t \sim 2M/D \)

Numerical time steps

- Should be proportional to minimum characteristic time
  - Implicit methods like Backwards Euler actually let you take larger steps with stability, but wipe out all hope of accuracy for things with small characteristic time
- For nonlinear multi-dimensional forces, what are \( K \) and \( D \)?
  - Estimate them by figuring out what is the fastest \(|F|\) can change if you modify \( x \) or \( v \) respectively
  - This is all very approximate, so don’t get hung up on getting the “right” answer
  - Will ultimately need a fudge factor anyhow (from experiments)
**True Collisions**

- Turn attention from repulsions for a while
- Model collision as a discrete event - a bounce
  - Input: incoming velocity, object normal
  - Output: outgoing velocity
- Need some idea of how “elastic” the collision
  - Fully elastic - reflection
  - Fully inelastic - sticks (or slides)
- Let’s ignore friction for now
- Let’s also ignore how to incorporate it into algorithm for moving particles for now

**Newtonian Collisions**

- Say object is stationary, normal at point of impact is \( \mathbf{n} \)
- Incoming particle velocity is \( \mathbf{v} \)
- Split \( \mathbf{v} \) into normal and tangential components:
  \[
  v_N = \mathbf{v} \cdot \mathbf{n} \\
  v_T = \mathbf{v} - v_N \mathbf{n}
  \]
- Newtonian model for outgoing velocity
  - Unchanged tangential component \( v_T \)
  - New normal component is \( v_N^{\text{new}} = -\varepsilon v_N^{\text{old}} \)
  - The “coefficient of restitution” is \( \varepsilon \), ranging from 0 (inelastic) to 1 (perfectly elastic)
- The final outgoing velocity is
  \[
  v^{\text{new}} = v_T - \varepsilon v_N^{\text{old}} \mathbf{n}
  \]

**Relative velocity in collisions**

- What if particle hits a moving object?
- Now process collision in terms of relative velocity
  - \( v_{\text{rel}} = v_{\text{particle}} - v_{\text{object}} \)
  - Take normal and tangential components of relative velocity
  - Reflect normal part appropriately to get new \( v_{\text{rel}} \)
  - Then new \( v_{\text{particle}} = v_{\text{object}} + (\text{new } v_{\text{rel}}) \)