Notes

Time integration for particles

- Back to the ODE problem, either

$$\frac{dx_i}{dt} = v(x_i, t) \quad \text{or} \quad \begin{cases} \frac{dx_i}{dt} = v_i \\ \frac{dv_i}{dt} = \frac{1}{m_i} F(x_i, v_i, t) \end{cases}$$

- Accuracy, stability, and ease-of-implementation are main issues
  - Obviously Forward Euler and Symplectic Euler are easy to implement - how do they fare in other ways?

Stability

- Do the particles fly off to infinity?
  - Particularly a problem with stiff springs
- Can always be fixed with small enough time steps - but expensive!
- Basically the problem is extrapolation:
  - From time t we take aim and step off to time t+Δt
  - Called “explicit” methods
- Can turn this into interpolation:
  - Solve for future position at t+Δt that points back to time t
  - Called “implicit” methods

Backward Euler

- Simplest implicit method: very stable, not so accurate, can be painful to implement

$$\mathbf{x}^{\mathbf{n+1}} = x^n + \Delta t\, v(\mathbf{x}^{\mathbf{n+1}}, t^{n+1})$$

  - Again, can use for both 1st order and 2nd order systems
  - Solving the system for x^{n+1} often means Newton’s method (linearize as in Gauss-Newton)

5

Simplified Backward Euler

- Take just one step of Newton, i.e. linearize nonlinear velocity field:

$$v\left(x^{n+1}\right) \approx v\left(x^n\right) + \frac{\partial v\left(x^n\right)}{\partial x}\left(x^{n+1} - x^n\right)$$

- Then Backward Euler becomes a linear system:

$$x^{n+1} = x^n + \Delta t\left[v\left(x^n\right) + \frac{\partial v\left(x^n\right)}{\partial x}\left(x^{n+1} - x^n\right)\right]$$

$$\Delta x = \Delta t\left[v\left(x^n\right) + \frac{\partial v\left(x^n\right)}{\partial x}\Delta x\right]$$

$$\left(I - \frac{\partial v}{\partial x}\right)\Delta x = \Delta t\, v\left(x^n\right)$$

6

Particle Collisions

- Usually don't want particles to go through other objects
- In 1st order case (given velocity field) need to change velocity field to avoid collision
  - Can work out special case formulas to make particles stream around object
  - More generally, add a repulsion force field
  - As particle distance decreases, normal force outward from object increases

7

Aside: Object Geometry

- When we talk about particles colliding with objects, need to know how to represent objects, and how to answer:
  - Is particle inside/outside?
  - Does particle trajectory cross?
  - Object normal at some point on surface?
  - Distance/direction to surface in space?
- Standard representations:
  - Special geometry: plane, sphere, cylinder, prism…
  - Heightfield: y=h(x,z)
  - Triangle mesh, closed or open
  - Implicit function
    - Level set (special case)

8

Plane

- Represent with a point p on the plane, and the (outward) normal n of the plane
  - Often simply p=0, n=(0,1,0) --- the ground
- Particle x inside: (x-p)•n<0
- Trajectory cross: does (x-p)•n change sign?
- Object normal: always n
- Distance to surface: if n is unit length, (x-p)•n is "signed distance"
  - |(x-p)•n| is regular distance
- n or -n is direction to closest point on surface (-n if x is outside)

Sphere

- Represent with a centre point p and a radius r
- Particle inside: |x-p|-r<0
- Trajectory cross: complicated!
  - Need to solve quadratic equation for intersection of straight line trajectory…
- Outward object normal: (x-p)/r
- Signed distance: |x-p|-r
- Direction to closest point on surface: ±(x-p)/|x-p|
  - Sign depends on inside/outside
  - Beware of divide by zero at x=p
  - Note: matches up with normal again!

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 $\det(a,b,c) = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix} \quad (= a \times b \cdot c)$
  - Det(a,b,c)=±6 volume(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×b•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