1

Notes

2

Building implicit surfaces

- Simplest examples: a plane, a sphere
- Can do unions and intersections with min and max
- This works great for isolated particles, but we want a smooth liquid mass when we have lots of particles together
  - Not a bumpy union of spheres

3

Blobbies and Metaballs

- Solution is to add kernel functions together
- Typically use a spline or Gaussian kernel around each particle
- Still may look a little bumpy - can process surface geometry to smooth it out afterwards…

4

Marching Cubes

- Going back to blobby/metaball implicit surfaces: often need mesh of surface
- Idea of marching cubes (or marching tets):
  - Split space up into cells
  - Look at implicit surface function at corners of cell
  - If there's a zero crossing, estimate where, put a polygon there
  - Make sure polygons automatically connect up

Acceleration

- Efficiency of neighbour location
  - Rendering implicit surfaces - need to quickly add only the kernel functions that are not zero (avoid O(n) sums!)
  - Also useful later for liquid animation and collisions
- Use an acceleration structure
  - Background grid or hashtable
  - Kd-trees also popular

Back to animation

- The real power of particle systems comes when forces depend on other particles
- Example: connect particles together with springs
  - If particles i and j are connected, spring force is

$$F_i = -k\left(\frac{\|x_i - x_j\|}{L_{ij}} - 1\right)\frac{x_i - x_j}{\|x_i - x_j\|} \qquad F_j = -F_i$$

  - The rest length is L and the spring “stiffness” is k
  - The bigger k is, the faster the particles try to snap back to rest length separation
  - Simplifies for L=0

Damped springs

- Real springs oscillate less and less
  - Motion is “damped”
  - Add damping force:

$$F_i^{damp} = -D\left[\frac{(v_i - v_j)}{L_{ij}} \cdot \frac{x_i - x_j}{\|x_i - x_j\|}\right]\frac{x_i - x_j}{\|x_i - x_j\|}$$

$$F_j^{damp} = -F_i^{damp}$$

  - D is damping parameter
  - Note: could incorporate L into D
- Simplified form (less physical…)

$$F_i^{damp} = -D(v_i - v_j) \qquad \text{or even} \qquad F_i^{damp} = -Dv_i$$

Elastic objects

- Can animate elastic objects by sprinkling particles through them, then connecting them up with a mesh of springs
  - Hair - lines of springs
  - Cloth - 2D mesh of springs
  - Jello - 3D mesh of springs
- With complex models, can be tricky to get the springs laid out right, with the right stiffnesses
  - More sophisticated methods like Finite Element Method (FEM) can solve this

9

Liquids

- Can even animate liquids (water, mud…)
- Instead of fixing which particles are connected, just let nearby particles interact
  - If particles are too close, force pushes them apart
  - If particles a bit further, force pulls them closer
  - If particles even further, no more force
  - Controlled by a smooth kernel function
- Related to numerical technique called SPH: smoothed particle hydrodynamics
- With enough particles (and enough tweaking!) can get a nice liquid look
- Render with implicit surface

10

Noise

- Useful for defining velocity/force fields, particle variations, and much much more (especially shaders)
- Need a smooth random number field
- Several approaches
- Most popular is Perlin noise
  - Put a smooth cubic (Hermite) spline patch in every cell of space
  - Control points have value 0, slope looked up from table by hashing knot coordinates
  - You can decide spatial frequency of noise by rescaling grid

11

Time integration for particles

- Back to the ODE problem, either

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

- 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?

12

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)

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:

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

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

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