1

Notes

- Text:
  - Motion Blur A.3
  - Particle systems 4.5 (and 4.4.1, 6.6.2…)
  - Implicit Surfaces - 4.6
- Classic particle system papers
  - W. Reeves, "Particle Systems…" SIGGRAPH '83 [REQUIRED READING]
  - K. Sims, "Particle Animation and Rendering…", SIGGRAPH '90

2

Velocity fields

- Velocity field could be a combination of pre-designed velocity elements
  - E.g. explosions, vortices, …
- Or from "noise"
  - Smooth random number field
  - See later
- Or from a simulation
  - Interpolate velocity from a computed grid
  - E.g. smoke simulation

3

Second order motion

- Real particles move due to forces
  - Newton's law F=ma
  - Need to specify force F (gravity, collisions, …)
  - Divide by particle mass to get acceleration a
  - Update velocity v by acceleration
  - Update position x by velocity

$$v_i^{new} = v_i + \Delta t \frac{F(x_i, v_i, t)}{m_i}$$

$$x_i^{new} = x_i + \Delta t v_i^{new}$$

4

Time integration

- Really solving ordinary differential equations in time:

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

- Methods presented before are called "Forward Euler" and "Symplectic Euler"
  - There are better numerical methods
  - These are the simplest that can work - but big issue is stability - more on this later

5

Basic rendering

- Draw a dot for each particle
- But what do you do with several particles per pixel?
  - Add: models each point emitting (but not absorbing) light -- good for sparks, fire, …
  - More generally, compute depth order, do alpha-compositing (and worry about shadows etc.)
  - Can fit into Reyes very easily
- Anti-aliasing
  - Blur edges of particle, make sure blurred to cover at least a pixel
- Particle with radius: kernel function

6

Motion blur

- One case where you can actually do exact solution instead of sampling
- Really easy for simple particles
  - Instead of a dot, draw a line (from old position to new position - the shutter time)
  - May involve decrease in alpha
  - More accurately, draw a spline curve
  - May need to take into account radius as well…

7

More detailed particle rendering

- Stick a texture (or even a little movie) on each particle
  - E.g. a noise function
  - E.g. a video of real flames
- Draw a little object for each particle
  - Need to keep track of orientation as well, unless spherical
  - We'll get into full-fledged rigid bodies later
- Draw between particles
  - curve (hair), surface (cloth)
- Implicit surface wrapped around virtual particles (e.g. water)

8

Implicit Surface Rendering

- Idea for water, mud, etc: **implicit surface**
- Write down a function F(x) that implicitly defines surface
  - Where it is above threshhold t we are inside
  - Where it is below, we are outside
  - Where F(x)=t is the surface
- Ray-tracing implicit surface is pretty easy
  - For ray O+sD solve F(O+sD)=t
    - Could use Bisection or Secant search to find s
  - Get surface normal from ∇F
- Other rendering methods trickier…
  - E.g. for Reyes need to turn into a mesh or subdivision surface: "Marching Cubes"

9

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

10

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…

11

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

12

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

13

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

14

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…)

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

15

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

16

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

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