Notes

- References in text:
  - Compositing A.2
  - Motion Blur A.3
  - Matchmove (camera calibration) 6.7.2, B.9
  - 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
  - K. Sims, “Particle Animation and Rendering...”, SIGGRAPH ‘90
- Many many others also

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)

Implicit Surface Rendering

- Idea for water, mud, etc: implicit surface
- Write down a function F(x) that implicitly defines surface
  - Where it is above threshold 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 VF
- Other rendering methods trickier...
  - E.g. for Reyes need to turn into a mesh or subdivision surface: “Marching Cubes”

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
**Bobbies 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...

**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
    \[ F_i = -k \frac{\left[ \frac{x_i - x_j}{L_{ij}} \right]}{\frac{x_i - x_j}{L_{ij}}} \]
    \[ F_j = -F_i \]
  - 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:
    \[ F_{i \text{ damp}} = -D \left( \frac{(v_i - v_j) \cdot (x_i - x_j)}{L_{ij}} \right) \frac{x_i - x_j}{\|x_i - x_j\|} \frac{x_i - x_j}{\|x_i - x_j\|} \]
    \[ F_{j \text{ damp}} = -F_{i \text{ damp}} \]
  - D is damping parameter
  - Note: could incorporate L into D
  - Simplified form (less physical...)
    \[ F_{i \text{ damp}} = -D(v_i - v_j) \quad \text{or even} \quad F_{i \text{ damp}} = -Dv_i \]

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

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
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?