Particle Systems

CPSC 414
Robert Bridson

Particle Systems

- A large collection of points that move as a group
- For phenomena that look like big groups of points
  - Dust, water spray, rain, stars, sparks, …
- For "fuzzy" phenomena that are really hard to model
  - Smoke, fire, clouds, grass, fur, water, …

Questions

- When and where do particles start?
  - (and when do they disappear)
- How do they move?
- What do they look like?
- Let's take a quick look at a few movies to see what the answers could be…

What is a particle?

- Most basic particle only has a position x
- Usually add other attributes, such as:
  - Age
  - Colour
  - Radius
  - Orientation
  - Velocity v
  - Mass m
  - Temperature
  - Type

Seeding

- Need to add (or seed) particles to the scene
- Where?
  - Randomly within a shaped volume or on a surface
  - At a point
  - Where there aren't many particles currently
- When?
  - At the start
  - Several per frame
  - When there aren't enough particles somewhere
- Need to figure out other attributes, not just position
  - E.g. velocity pointing outwards in an explosion

Basic animation

- Specify a velocity field v(x,t) for any point in space x, any time t
- Break time into steps
  - E.g. per frame - Δt=1/30th of a second
  - Or several steps per frame
- Change each particle's position x_i by "integrating" over the time step

$$x_i^{new} = x_i + \Delta t v(x_i, t)$$

Velocity fields

- Velocity field could be a combination of pre-designed velocity elements
  - [examples]
- Or from "noise"
  - Smooth random number field
  - [show it]
- Or from a simulation
  - Interpolate velocity from a computed grid

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

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

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

Time integration

- Really solving ordinary differential equations in time:

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

- Methods presented before are called "Forward Euler" and "Symplectic Euler"
  - There are better numerical methods…

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.)
- Anti-aliasing
  - Blur edges of particle, make sure blurred to cover at least a pixel
- Particle with radius: kernel function

Motion blur

- Temporal anti-aliasing
- Really easy for simple particles
  - Instead of a dot, draw a line (from old position to new position)
- More accurately, draw a spline curve
- May need to take into account radius as well…

More detailed 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

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

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

  - The rest length is L and the spring "stiffness" is k

Damped springs

- Real springs slowly oscillate less and less
  - Motion is "damped"
  - Add damping force:

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

  - D is damping parameter

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
- Rendering done differently though
  - Hair - draw curve through particles
  - Cloth/Jello - draw surface triangles between particles

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
- With enough particles, can get a nice liquid look
- But how do we render?
  - There is no fixed surface mesh of triangles!

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…

Building implicit surfaces

- Simplest: a sphere
  - [what is it?]
- How about two or more spheres?
  - [unions]
- 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

Blobbies and Metaballs

- Solution is to add kernel functions together
- Typically use a spline or Gaussian kernel around each particle
- [draw in 1D]

Acceleration

- One last issue for animating and rendering liquids: efficiency
  - Forces - need to quickly find only the nearby particles (avoid O(n) checks!)
  - Rendering - need to quickly add only the kernel functions that are not zero (avoid O(n) sums!)
- Use an acceleration structure
  - Background grid or hashtable