CS 174C Week 5

Aidan Jan

February 7, 2024

Physically Based Animation

What is a Particle

The most basic particle has a position x in space.
There may be other attributes, such as:

- Velocity v
- Acceleration a
- Mass m
- Orientation
- Radius
- Temperature
- averageColor
- Type

The sky is the limit - e.g., AI agent models of behavior

Basic Particle Animation

- Specify a velocity field $v(x, t)$ for any point in space x at any time t
- Break time t into discrete steps Δt
  - E.g., one step per frame - $\Delta t = 1/30$th second
  - Or several steps per frame
- Change the particle's current position $x(t)$ by integrating over the time step
  - Forward Euler Integration:
    - Since velocity $v(t) = \frac{\mathrm{d}x}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{x(t+\Delta t) - x(t)}{\Delta t}$,

$$x(t + \Delta t) \approx x(t) + \Delta t \cdot v(x(t), t)$$

Velocity Fields

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

Seeding Particles

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

Basic Particle 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 they are blurred to cover at least a pixel
- Particle with radius: Kernel function

Motion Blur

Easy for simple particles

- Instead of a dot, draw a line from old position to new position
  - During the "open shutter" time
- May involve decrease in alpha
- More accurately, draw a spline curve
- May also need to take particle radius into account

Physics-Based Motion of a Particle

Second-order dynamic motion of mass particles

- Newton's law $f = ma$
- Need to specify force(s) f
  - gravity, collisions, ...
- Divide net force by particle mass m to obtain acceleration a
- Update velocity v using acceleration
- Update position x using velocity

Integrating the Equations of Motion

- Newtons law:

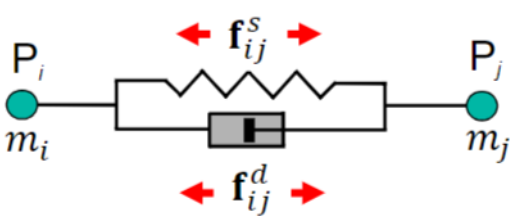
$$a(t) = \frac{1}{m} f(x(t), v(t), t)$$

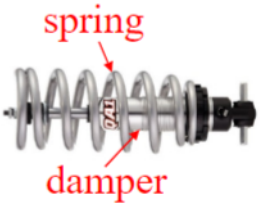
- We solve this by numerical simulation
  - Integrate second-order ODE forward in time
    - Compute $v(t)$, $x(t)$, at $t = t_0, t_0 + \Delta t, t_0 + 2\Delta t, \ldots, t_0 + n\Delta t, \ldots$
    - Symplectic Euler integration step at each time t:

$$v(t + \Delta t) = v(t) + \Delta t a(t)$$
$$x(t + \Delta t) = x(t) + \Delta t v(t + \Delta t)$$

Mass-Spring-Damper Systems

- Particles interconnected by springs and dampers
  - Simplest mass-spring-damper system:



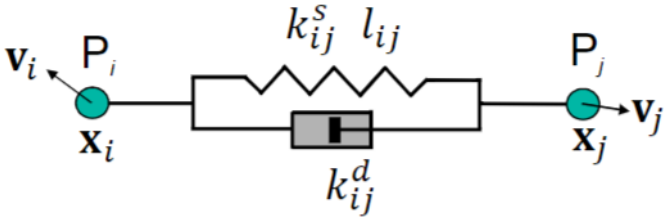


    - Parallel spring and dampler: Voigt visco-elastic (v-e) element

– Equations of motion with $f_{ij}^e = f_{ij}^s + f_{ij}^d$

$$m_i a_i = f_i + f_{ij}^e$$
$$m_j a_j = f_j + f_{ji}^e$$

Viscoelastic Forces



$$d_{ij} = x_j - x_i$$
$$d_{ij_m} = ||d_{ij}||$$
$$\hat{d}_{ij} = d_{ij}/d_{ij_m}$$
$$v_{ij} = v_j - v_i$$

- Spring force (elastic)
  - With spring constant k_{ij}^s and natural length l_{ij}

$$f_{ij}^s = k_{ij}^s(d_{ij_m} - l_{ij})\hat{d}_{ij}; \qquad f_{ji}^s = -f_{ij}^s$$

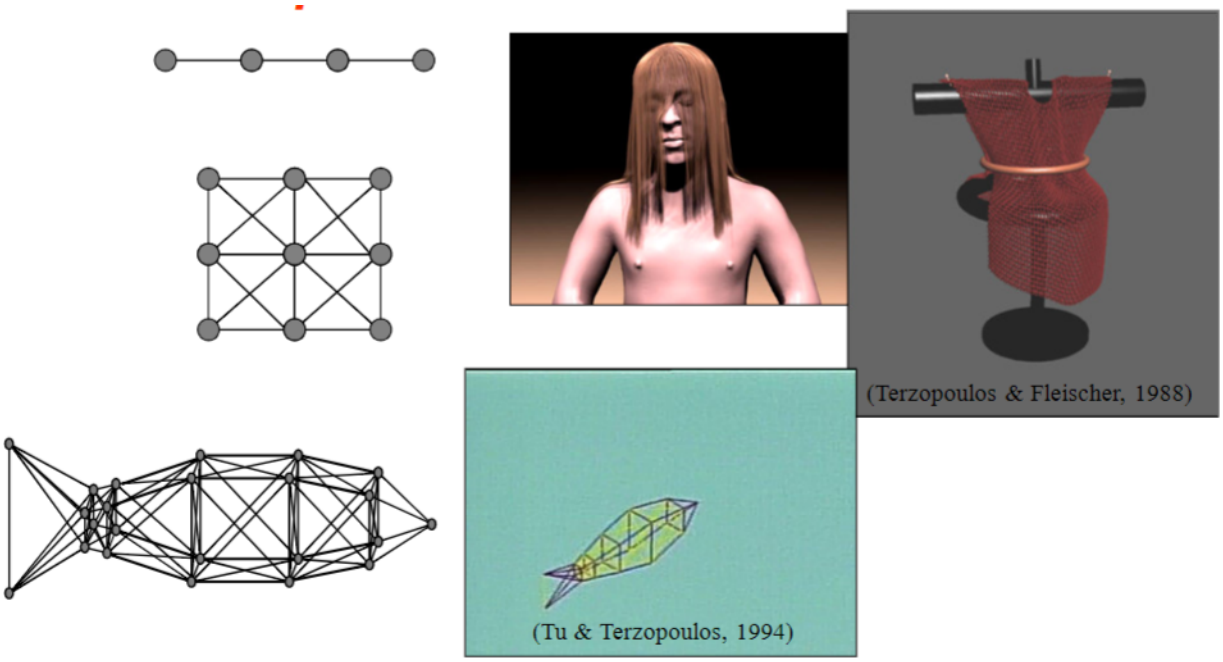
- Damper force (viscous)
  - With damping constant k_{ij}^d

$$f_{ij}^d = -k_{ij}^d(v_{ij} \cdot \hat{d}_{ij})\hat{d}_{ij}; \qquad f_{ji}^d = -f_{ij}^d$$

- Visco-elastic element force: $f_{ij}^e = f_{ij}^s + f_{ij}^d$

Bigger Systems

Each line represents a viscoelastic element



Putting Everything Together

- Particle system parameters
  - An array $P[i]$ of particle data structures, each structure containing
    - particle mass m
    - particle position x
    - particle velocity v
    - net force f acting on particle
  - An array $S[k]$ of v-e element data structures, each structure containing
    - pointer to particle i
    - pointer to particle j
    - spring constant k^s and natural length l_{ij}
    - damper constant k^d
- Emvironmental parameters
  - Perhaps gravity g, other external forces f_i and/or force fields, etc.
- Initial conditions: $P[i].x = x_i(0); P[i].v = v_i(0);$

Simulation loop (starting from $t = 0$ and initial particle positions and velocities)

1. For all particles i, initialize force variables $P[i].f = 0$
2. For all v-e elements k, using particles $P[i]$ and $P[j]$ connected by $S[k]$, compute visco-elastic force f^e_{ij}; accumulate f^e_{ij} on $P[i]$ and $-f^e_{ij}$ on $P[j]$
3. For all particles i, accumulate all external forces acting on particle $P[i]$

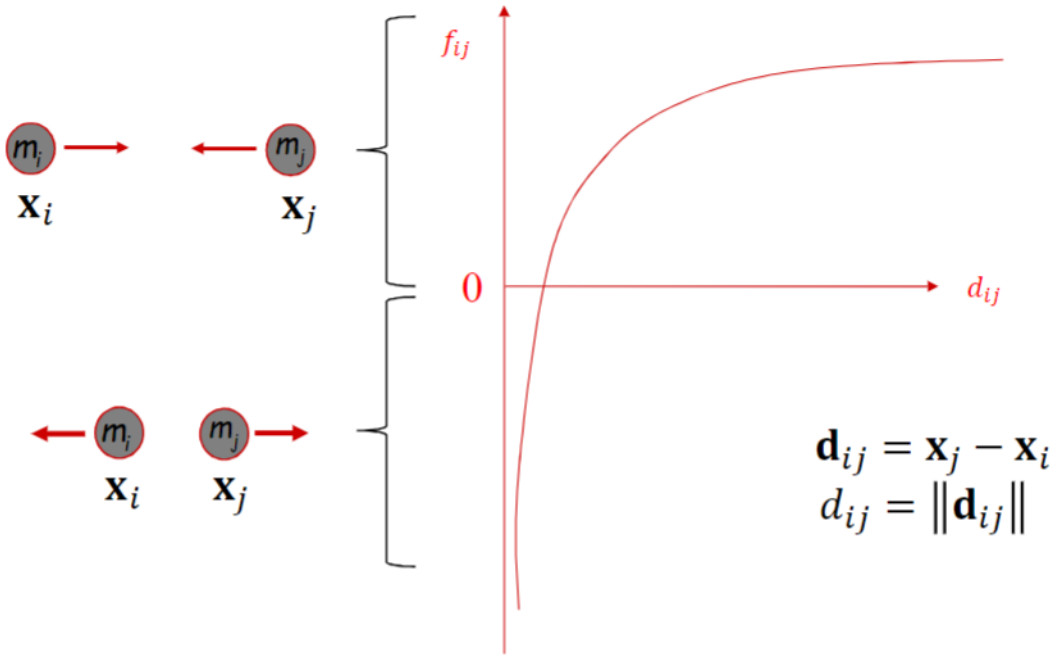
4. For all particles i, update

$$P[i].v := P[i].v + \Delta t \frac{P[i].f}{P[i].m}$$
$$P[i].x := P[i].x + \Delta t P[i].v$$

5. Update $t \leftarrow t + \Delta t$ and go to (1).

Particle-Based Fluid Flow Simulation

Lenard-Jones Force Profile



Discrete Fluid Model

The total force on a particle i due to all other particles:

$$g_i(t) = \sum_{j \neq i} g_{ij}(t)$$

where

$$g_{ij}(t) = m_i m_j \left(\frac{\alpha}{(d_{ij} + \varepsilon)^a} - \frac{\beta}{d_{ij}^b} \right) d_{ij}$$

- Factors α and β scale the strength of the attraction and repulsion ofrces
- Exponents often set to $a = 2$ and $b = 4$
- ε is minimum required separation of particles

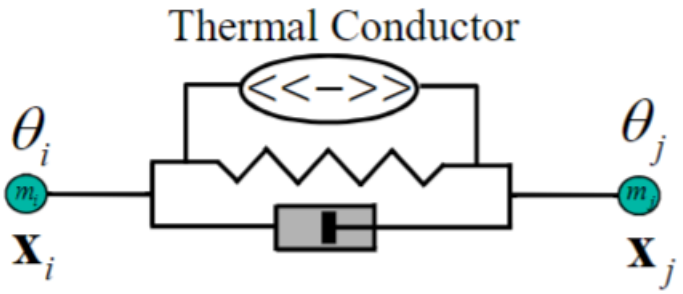
Heat (Diffusion) Equation

Diffusion of heat in materials

$$\frac{\partial}{\partial t}(\mu\sigma\theta) - \nabla \cdot (C\nabla\theta) = q$$

- q: rate of heat generation/loss per unit volume
- μ: mass density, kg/m^3
- σ: specific heat, Joule/(kg $\cdot$ Kelvin), material property
- θ: temperature, Kelvin
- C: thermal conductivity matrix, material property
- $\nabla = \left[\frac{\partial}{\partial u}, \frac{\partial}{\partial v}, \frac{\partial}{\partial w}\right]$

Thermoelasticity



- Spring constant k_{ij} is inversely related to $(\theta_i + \theta_j)/2$
- In the molten state, we can use Lenard-Jones inter-particle forces.

Time Integration Methods

- Issues
  - Stability
  - Accuracy
  - Efficiency
- Two main categories
  - Explicit Methods
  - Implicit Methods

Explicit (Forward) Euler Method

- Simple and fast for integrating First-order ODE

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = v(x(t), t)$$
$$x(t + \Delta t) = x(t) + \Delta t \cdot v(x(t), t)$$

- However,
  - Only modest accuracy
  - Relatively unstable
    - If Δt is too large, accumulating error makes successive approximations spiral away

Midpoint Method

- Better accuracy and stability
  - Use more accurate central approximations

$$v\left(t + \frac{\Delta t}{2}\right) \approx \frac{x(t + \Delta t) - x(t)}{\Delta t}$$

  - Obtain update step

$$x(t + \Delta t) = x(t) + \Delta t \cdot v\left(x\left(t + \frac{\Delta t}{2}\right), t + \frac{\Delta t}{2}\right)$$

  - where from 1^{st}-order Taylor expansion

$$x\left(t + \frac{\Delta t}{2}\right) \approx x(t) + \frac{\Delta t}{2} v(t) = x(t) + \frac{\Delta t}{2} v(x(t), t)$$

Symplectic Euler Method

- For 2^{nd}-order dynamical systems
  - Forward Euler:

$$v(t + \Delta t) = v(t) + \Delta t \frac{1}{m} f(x(t), v(t), t)$$
$$x(t + \Delta t) = x(t) + \Delta t \cdot v(t)$$

  - Better, Symplectic Euler

$$v(t + \Delta t) = v(t) + \Delta t \frac{1}{m} f(x(t), v(t), t)$$
$$x(t + \Delta t) = x(t) + \Delta t \cdot v(t + \Delta t)$$

Verlet Integration Method

- Commonly used for 2^{nd}-order dynamical systems
  - Forward 1^{st} difference
$$v(t) \approx \frac{x(t+\Delta t) - x(t)}{\Delta t}$$
  - Backward 1^{st} difference
$$v(t-\Delta t) \approx \frac{x(t) - x(t-\Delta t)}{\Delta t}$$
  - Centered 2^{nd} difference
$$a(t) \approx \frac{v(t) - v(t-\Delta t)}{\Delta t} = \frac{1}{\Delta t^2}(x(t+\Delta t) - 2x(t) + x(t-\Delta t))$$
- Rearranging terms:
$$x(t+\Delta t) = 2x(t) - x(t-\Delta t) + \frac{\Delta t^2}{m} f(x(t), v(t), t)$$

Velocity Verlet Method

- (see `https://en.wikipedia.org/wiki/Verlet_integration`)
- Rather than two position vectors, need to store position and velocity vectors
  - Assuming $a(t) = \frac{1}{m} f(x(t))$, iterate as follows:
$$x(t+\Delta t) = x(t) + \Delta t v(t) + \frac{\Delta t^2}{2} a(t)$$
$$v(t+\Delta t) = v(t) + \Delta t \frac{a(t) + a(t+\Delta t)}{2}$$