What are particle systems?

A **particle system** is a collection of point masses that obeys some physical laws (e.g., gravity or spring behaviors).

Particle systems can be used to simulate all sorts of physical phenomena:
- Smoke
- Snow
- Fireworks
- Hair
- Cloth
- Snakes
- Fish
Particle in a flow field

We begin with a single particle with:
- Position, \( \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \)
- Velocity, \( \mathbf{v} = \dot{\mathbf{x}} = \frac{dx}{dt} = \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} \)

Suppose the velocity is dictated by some driving function \( g \):
\[
\dot{x} = g(x,t) \quad y
\]

How does our particle move through the vector field?

Vector fields

At any moment in time, the function \( g \) defines a vector field over \( \mathbf{x} \):

Diff eqs and integral curves

The equation \( \dot{x} = g(x,t) \) is actually a first order differential equation.

We can solve for \( \mathbf{x} \) through time by starting at an initial point and stepping along the vector field:

This is called an initial value problem and the solution is called an integral curve.

Euler’s method

One simple approach is to choose a time step, \( \Delta t \), and take linear steps along the flow:
\[
\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t \cdot \dot{\mathbf{x}}(t) = \mathbf{x}(t) + \Delta t \cdot g(\mathbf{x},t)
\]

This approach is called Euler’s method and looks like:

Properties:
- Simplest numerical method
- Bigger steps, bigger errors

Need to take pretty small steps, so not very efficient. Better (more complicated) methods exist, e.g., “Runge-Kutta.”
Particle in a force field

- Now consider a particle in a force field \( f \).
- In this case, the particle has:
  - Mass, \( m \)
  - Acceleration, \( a \equiv \ddot{x} = \frac{d}{dt} \frac{d^2x}{dt^2} \)
- The particle obeys Newton’s law: \( f = ma = m\ddot{x} \)
- The force field \( f \) can in general depend on the position and velocity of the particle as well as time.
- Thus, with some rearrangement, we end up with:
  \[ \ddot{x} = \frac{f(x, \dot{x}, t)}{m} \]

Second order equations

This equation: \( \ddot{x} = \frac{f(x, \dot{x}, t)}{m} \)

is a second order differential equation.

Our solution method, though, worked on first order differential equations.

We can rewrite this as:

\[
\begin{bmatrix}
\dot{x} \\
\dot{v}
\end{bmatrix} = \begin{bmatrix}
v \\
\frac{f(x, v, t)}{m}
\end{bmatrix}
\]

where we have added a new variable \( v \) to get a pair of coupled first order equations.

Phase space

Concatenate \( x \) and \( v \) to make a 6-vector: position in phase space.

Taking the time derivative: another 6-vector.

\[
\begin{bmatrix}
\dot{x} \\
\dot{v}
\end{bmatrix} = \begin{bmatrix}
v \\
\frac{f}{m}
\end{bmatrix}
\]

A vanilla 1st-order differential equation.

Particle structure

Position in phase space

- \( x \) ← position
- \( v \) ← velocity
- \( f \) ← force accumulator
- \( m \) ← mass
Solver interface

particles  n  time

get/setState  getDim  derivEval

[6n]

x  v  x  v  ...  x  v

f  f  ...  f  f

m  m  ...  m  m

Particle systems

particles  n  time

[6n]

x  v  x  v

f  f  ...  f  f

f / m

Forces

- Constant (gravity)
- Position/time dependent (force fields)
- Velocity-dependent (drag)
- N-ary (springs)
Gravity

Force law:
\[ f_{\text{grav}} = mg \]

\[ p->f += p->m \times F->G \]

Viscous drag

Force law:
\[ f_{\text{drag}} = -k_{\text{drag}} v \]

\[ p->f -= F->k \times p->v \]

Damped spring

Force law:
\[ f_i = -k_s \left( |\vec{x}| - r \right) + k_d \left( \frac{\dot{v} \cdot \vec{x}}{|\vec{x}|} \right) \]
\[ f_2 = -f_1 \]

\[ r = \text{rest length} \]

\[ \vec{x} = x_1 - x_2 \]

\[ \dot{v} = v_1 - v_2 \]

Particle systems with forces

\[ \begin{bmatrix} x_1 \\ v_1 \\ m_1 \end{bmatrix}, \ldots, \begin{bmatrix} x_n \\ v_n \\ m_n \end{bmatrix} \]
derivEval loop

1. Clear forces
   - Loop over particles, zero force accumulators
2. Calculate forces
   - Sum all forces into accumulators
3. Gather
   - Loop over particles, copying v and f/m into destination array

drivEval Loop

1. Clear force accumulators
2. Apply forces to particles
3. Return [v,f/m,…] to solver

Solver interface

Differential equation solver

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 \\
\vdots \\
\dot{x}_n \\
\end{bmatrix} =
\begin{bmatrix}
\dot{v}_1 \\
\dot{v}_2 \\
\vdots \\
\dot{v}_n \\
\end{bmatrix} =
\begin{bmatrix}
\ddot{x}_1 \\
\ddot{x}_2 \\
\vdots \\
\ddot{x}_n \\
\end{bmatrix} =
\begin{bmatrix}
\frac{v_1'}{m_1} \\
\frac{v_2'}{m_2} \\
\vdots \\
\frac{v_n'}{m_n} \\
\end{bmatrix}
\]

Euler method:

\[
\begin{bmatrix}
x_1^{i+1} \\
x_2^{i+1} \\
\vdots \\
x_n^{i+1} \\
\end{bmatrix} =
\begin{bmatrix}
x_1^i \\
x_2^i \\
\vdots \\
x_n^i \\
\end{bmatrix} +
\begin{bmatrix}
\frac{v_1^i}{m_1} \\
\frac{v_2^i}{m_2} \\
\vdots \\
\frac{v_n^i}{m_n} \\
\end{bmatrix}
\]
Bouncing off the walls

- Add-on for a particle simulator
- For now, just simple point-plane collisions

Normal and tangential components

\[ V_N = (N \cdot V)N \]
\[ V_T = V - V_N \]

Collision Detection

\[(X - P) \cdot N < \epsilon \quad \text{Within } \epsilon \text{ of the wall} \]
\[ N \cdot V < 0 \quad \text{Heading in} \]

Collision Response

\[ V' = V_T - k_r V_N \]
Artificial Fish

Related Research

- Determining dynamic parameters for cloth simulation

Summary

What you should take away from this lecture:
- The meanings of all the **boldfaced** terms
- Euler method for solving differential equations
- Combining particles into a particle system
- Physics of a particle system
- Various forces acting on a particle
- Simple collision detection