Differential Equations & Particle Systems

Thanks to Trueille, Popovic, Baraff, Witkin
Physics-based Animation

http://physbam.stanford.edu/~fedkiw/animations/large_pile.avi
Passive—no muscles or motors

user → initial conditions → model

model → numerical integrator → graphics

Particle systems
Leaves
Water
Smoke
Clothing

Active—internal sources of energy

user → desired behavior → control

control → forces and torques → model

model → numerical integrator → graphics

Running human
Trotting dog
Swimming fish
Dynamics

• Generate motion by specifying mass and force, apply physical laws (e.g., Newton’s laws)
  – particles
  – soft objects
  – rigid bodies
• Simulates physical phenomena
  – gravity
  – momentum (inertia)
  – collisions
  – friction
  – fluid flow (drag, turbulence, ...)
  – solidity, flexibility, elasticity
  – fracture
What variables do we need?

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Radius</td>
<td>• Position</td>
</tr>
<tr>
<td>• Mass</td>
<td>• Velocity</td>
</tr>
<tr>
<td>• Racquet Info</td>
<td>• Rotation?</td>
</tr>
</tbody>
</table>
What Happens Next?

- Position
- Velocity

Discrete Time: $x_{t+1} = f(x_t)$

Continuous Time: $\dot{x} = f(x)$
Differential Equations

\[ \dot{x} = f(x) \]
Differential Equation Basics

Andrew Witkin
A Canonical Differential Equation

\[ \dot{x} = f(x,t) \]

- \( x(t) \): a moving point.
- \( f(x,t) \): x’s velocity.
Vector Field

The differential equation

$$\dot{x} = f(x, t)$$

defines a vector field over $x$. 
Integral Curves

Start Here

Pick any starting point, and follow the vectors.
Given the starting point, follow the integral curve.
Euler’s Method

- Simplest numerical solution method
- Discrete time steps
- Bigger steps, bigger errors.

\[ x(t + \Delta t) = x(t) + \Delta t f(x, t) \]
Two Problems

• Accuracy
• Instability
Accuracy

Consider the equation:

\[ \dot{x} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} x \]

What do the integral curves look like?
Problem I: Inaccuracy

Error turns $x(t)$ from a circle into the spiral of your choice.
Problem 2: Instability

• Consider the following system:

\[
\begin{align*}
\dot{x} &= -x \\
x(0) &= 1
\end{align*}
\]
Problem 2: Instability to Neptune!
Accuracy of Euler Method

\[ \dot{x} = f(x) \]

Consider Taylor Expansion about \( x(t) \)...

\[ x(t + h) = x(t) + hf(x(t)) + O(h^2) \]

Euler’s method has error \( O(h^2) \)... first order.

How can we get to \( O(h^3) \) error?
Euler’s method has a speed limit

\[ x = -kx \quad \text{and} \quad \Delta x = -hkx \]

\[ h = .5(1/k) \quad h = 1(1/k) \quad h = 1.5(1/k) \quad h = 2(1/k) \quad h = 3(1/k) \]

\( h > 1/k: \) oscillate. \quad \( h > 2/k: \) explode!
The Midpoint Method

- Also known as second order Runge-Kutta:

\[ k_1 = h(f(x_0, t_0)) \]

\[ k_2 = hf(x_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}) \]

\[ x(t_0 + h) = x_0 + k_2 + O(h^3) \]
The Midpoint Method

a. Compute an Euler step

\[ \Delta x = \Delta t f(x, t) \]

b. Evaluate \( f \) at the midpoint

\[ f_{\text{mid}} = f \left( \frac{x + \Delta x}{2}, \frac{t + \Delta t}{2} \right) \]

c. Take a step using the midpoint value

\[ x(t + \Delta t) = x(t) + \Delta t f_{\text{mid}} \]
4th-Order Runge-Kutta

\[ k_1 = h f(x_0, t_0) \]

\[ k_2 = h f(x_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}) \]

\[ k_3 = h f(x_0 + \frac{k_2}{2}, t_0 + \frac{h}{2}) \]

\[ k_4 = h f(x_0 + k_3, t_0 + h) \]

\[ x(t_0 + h) = x_0 + \frac{1}{6} k_1 + \frac{1}{3} k_2 + \frac{1}{3} k_3 + \frac{1}{6} k_4 + O(h^5) \]
q-Stage Runge-Kutta

General Form:

\[ x(t_0 + h) = x_0 + h \sum_{i=1}^{q} w_i k_i \]

where:

\[ k_i = f \left( x_0 + h \sum_{j=1}^{i-1} \beta_{ij} k_j \right) \]

Find the constant that ensure accuracy O(h^n).
If your step size is too big, your simulation blows up. It isn’t pretty.

Sometimes you have to make the step size so small that you never get anyplace.

Nasty cases: cloth, constrained systems.
Implicit Euler Method

\[ x(t_0 + h) = x(t_0) + h \dot{x}(t_0) \]

\[ x(t_0 + h) = x(t_0) + h \dot{x}(t_0 + \Delta t) \]
Implicit Euler for $\dot{x} = -kx$

$$x(t + h) = x(t) + h \dot{x}(t + h)$$

$$= x(t) - h k x(t + h)$$

$$= \frac{x(t)}{1 + hk}$$
One Step: Implicit vs. Explicit

\[ \dot{x} = -x, \quad x(0) = 1 \]

Correct Solution:
\[ x(h) = e^{-hk} \]

Implicit Euler Step:
\[ x(h) = \frac{1}{1+ hk} \]

Explicit Euler Step:
\[ x(h) = 1 - hk \]
Modular Implementation

- **Generic operations:**
  - Get \( \text{dim}(x) \)
  - Get/set \( x \) and \( t \)
  - Deriv Eval at current \((x,t)\)

- **Write solvers in terms of these.**
  - Re-usable solver code.
  - Simplifies model implementation.
void eulerStep(Sys sys, float h) {
    float t = getTime(sys);
    vector<float> x0, deltaX;
    
    t = getTime(sys);
    x0 = getState(sys);
    deltaX = derivEval(sys, x0, t);
    setState(sys, x0 + h*deltaX, t+h);
}

Particle Systems
Particle Systems

Clouds
Smoke
Fire
Waterfalls
Fireworks

Reeves ’83, the Wrath of Khan
Batman Returns, using Reynolds’s flocking algorithms
Karl Sims, Particle Dreams
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
Overview

1. One lousy particle
2. Particle systems
3. Forces: gravity, springs
4. Implementation
Particle in a flow field

We begin with a single particle with:

- Position, \( \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \)

- Velocity, \( \mathbf{v} \equiv \dot{\mathbf{x}} = \frac{d\mathbf{x}}{dt} = \begin{bmatrix} dx/dt \\ dy/dt \end{bmatrix} \)

Suppose the velocity is dictated by some driving function \( \mathbf{g} \):

\[ \dot{\mathbf{x}} = \mathbf{g}(\mathbf{x}, t) \]
Particle in a force field

• Now consider a particle in a force field \( \mathbf{f} \).
• In this case, the particle has:
  
  - Mass, \( m \)
  - Acceleration, \( \mathbf{a} \equiv \ddot{\mathbf{x}} = \frac{d\mathbf{v}}{dt} = \frac{d^2\mathbf{x}}{dt^2} \)

• The particle obeys Newton’s law: \( \mathbf{f} = m\mathbf{a} = m\ddot{\mathbf{x}} \)

• The force field \( \mathbf{f} \) can in general depend on the position and velocity of the particle as well as time.
• Thus, with some rearrangement, we end up with:
  
  \[ \ddot{\mathbf{x}} = \frac{\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t)}{m} \]
Second order equations

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

is a second order differential equation.

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

We can rewrite this as:

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

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

\[
\begin{bmatrix}
  x \\
v
\end{bmatrix}
\]  
Concatenate \( x \) and \( v \) to make a 6-vector: position in phase space.

\[
\begin{bmatrix}
  \dot{x} \\
  \dot{v}
\end{bmatrix}
\]
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 1\textsuperscript{st}-order differential equation.
Particle structure

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

- Position in phase space
- position
- velocity
- force accumulator
- mass
Solver interface
Particle systems

\[
\begin{bmatrix}
x_1 \\
v_1 \\
f_1 \\
m_1
\end{bmatrix}
\begin{bmatrix}
x_2 \\
v_2 \\
f_2 \\
m_2
\end{bmatrix}
\begin{bmatrix}
x_3 \\
v_3 \\
f_3 \\
m_3
\end{bmatrix}
\ldots
\begin{bmatrix}
x_n \\
v_n \\
f_n \\
m_n
\end{bmatrix}
\]
Solver interface

particles \ n \ time

get/setState

getDim

drivEval

\[ 6n \]

\begin{array}{ccccccc}
 x_1 & v_1 & x_2 & v_2 & \cdots & x_n & v_n \\
 f_1 & \frac{f_1}{m_1} & f_2 & \frac{f_2}{m_2} & \cdots & f_n & \frac{f_n}{m_n} \\
\end{array} \]
Forces

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

Force law:
\[ f_{grav} = mG \]

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

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

Force law:

\[ f_1 = -k_s (|x| - r) + k_d \left( \frac{v \cdot x}{|x|} \right) \frac{x}{|x|} \]

\[ f_2 = -f_1 \]

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

\[ x = x_1 - x_2 \]

\[ v = v_1 - v_2 \]
Particle systems with forces

\[
\begin{bmatrix}
x_1 \\
v_1 \\
m_1
\end{bmatrix}
\begin{bmatrix}
x_2 \\
v_2 \\
m_2
\end{bmatrix}
\cdots
\begin{bmatrix}
x_n \\
v_n \\
m_n
\end{bmatrix}
\begin{bmatrix}
F \\
F \\
F
\end{bmatrix}
\cdots
\begin{bmatrix}
F
\end{bmatrix}
\]
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
## derivEval Loop

1. Clear force accumulators

2. Apply forces to particles

3. Return $[v, f/m, ...]$ to solver
Solver interface

```
| particles | n | time |
```

get/setState

getDim

derivEval

```
| x_1 | v_1 | x_2 | v_2 | ... | x_n | v_n |
```

```
| v_1 | f_1 | m_1 | v_2 | f_2 | ... | x_n | f_n | m_n |
```
Differential equation solver

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

Euler method:

\[
\begin{bmatrix}
x_1^{i+1} \\
v_1^{i+1} \\
\vdots \\
x_n^{i+1} \\
v_n^{i+1}
\end{bmatrix} =
\begin{bmatrix}
x_1^i \\
v_1^i \\
\vdots \\
x_n^i \\
v_n^i
\end{bmatrix} +
\begin{bmatrix}
v_1^i \\
f_1^i/m_1 \\
\vdots \\
v_n^i \\
f_n^i/m_n
\end{bmatrix} + t
\]
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

\[(X - P) \cdot N < \varepsilon\]  Within \(\varepsilon\) of the wall

\[N \cdot V < 0\]  Heading in
Collision Response

\[ V' = V_T - k_r V_N \]
Summary

- Physics of a particle system
- Various forces acting on a particle
- Combining particles into a particle system
- Euler method for solving differential equations
Example

http://www.youtube.com/watch?v=3_fLO4xjTqg
Spring-Mass Systems

Cloth in 2D
Jello in 3D
Cloth Simulation

Cloth forces:
- Blue (short horizontal & vertical) = stretch springs
- Green (diagonal) = shear springs
- Red (long horizontal & vertical) = bend springs
Many types of cloth
Very different properties
Not a simple elastic surface
Woven fabrics tend to be very stiff
Anisotropic

Breen ‘95
Artificial Fish

Muscle springs

Pectoral fin

2 Swimming Segments

2 Turning Segments

Node 0