CS-C3100 Computer Graphics
Jaakko Lehtinen

10.3 Implementing ODE Solvers

Picture: A. Selle et al.

Lots of slides from Frédo Durand

In This Video

- It pays off to abstract (as usual)
  - It's easy to design your particle system and ODE solver to be unaware of each other
  - And this comes with many benefits

- Basic idea:
  - Particle system and solver communicate via floating-point vectors and a function that computes $f(\mathbf{X},t)$
    - Solver does not need to know anything else!

Recap: Differential Equations

- Motion of physical systems is modelled by these
  - "If I am in state $\boldsymbol{X}$ at time t, what is my state at time t+dt ?

$$\frac{d\,\mathbf{X}(t)}{dt} = f(\mathbf{X}(t), t)$$

- What is the "state $\boldsymbol{X}$"? A vector that describes all the free parameters of the physical system.
  - For a single point-like mass: position + velocity
  - For a rigid object: position + orientation + linear momentum + angular momentum
  - etc.

Code Two Entities

- A Particle System
  - Knows its own “meaning”, e.g. cloth, smoke, etc.
  - Implements the function $f(\mathbf{X}(t), t)$

- An ODE Solver
  - Does not know the “meaning” of the system being solved
  - Just has access to $\mathbf{X}$ and f
  - Example: To implement the Euler method…

$$\mathbf{X}_1 = \mathbf{X}_0 + h\, f(\mathbf{X}_0, t_0)$$

…we do not need to know anything else!

Division of Labor, Part 1

- Responsibilities of particle system
  - Tells ODE solver how many dimensions (N) the phase space has
  - Has a function to write its state to an N-vector of floating point numbers (communication to solver)
  - Has a function to update its own state from a similar vector (to step forward according to what the solver said)
  - Has a function that evaluates f(**X**,t), given a state vector **X** and time t
    - **Very, very important**: this function has to work for any **X**, not just the current state!
    - Otherwise cannot implement e.g. trapezoid rule

Division of Labor, Part 2

– Responsibilities of ODE solver
  – reads state..
  – ..computes the derivatives using f($\mathbf{X}$,t) where needed..
  – ..outputs a new state vector $\mathbf{X}_{n+1}$ for particle system

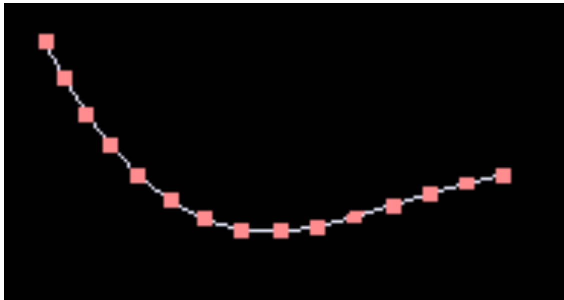
Benefits

- Such a black-box design is nice, because **you can reuse your solver**
  - You could implement a C++ base class “ODE” that would have have get/setState functions and evaluateDerivatives as virtual members
  - Then, your solver would just take in a pointer to ODE and compute new states without knowing anything about what is going on in the system
    - States and derivatives are just numbers

Example (Actual Code from String Demo)

```
// read the current positions + velocities (X) from particles
m_psystem->getState( vecState );

switch( m_integrator )
{
case EULER:
        // derivatives = f(X,t) at current state X
    vecDerivatives = m_psystem->evaluateDerivatives( vecState );
    // state = state + dt*derivatives
    vecState = stepSystem( dt, vecState, vecDerivatives );
    break;



    ....

}

    m_psystem->setState( vecState );
```

`stepSystem`(dt, **X**, **f**) just computes $\mathbf{X}_{new} = \mathbf{X} + dt*\mathbf{f}$



Example (Actual Code from String Demo)

```
// read the current positions + velocities from particles
m_psystem->getState( vecState );

switch( m_integrator )
{
case EULER:
        ...
     break;
case MIDPOINT:
     // evaluate f(X,t) at current state X
     vecDerivatives = m_psystem->evaluateDerivatives( vecState );
     // go half a step forward, intermediatestate = state + 0.5*dt*derivatives
     vecIntermediateState = stepSystem( dt/2.0f, vecState, vecDerivatives );
     // evaluate derivatives after half timestep
     vecDerivatives2 = m_psystem->evaluateDerivatives( vecIntermediateState );
     // full timestep using t+0.5dt derivatives FROM ORIGINAL POSITION
     vecState = stepSystem( dt, vecState, vecDerivatives2 );
     break;
}

    m_psystem->setState( vecState );
```

What does f(**X**,t) compute?

- N point masses
  - Stack all **x**s and **v**s in a big vector of length 6N
  - The $\boldsymbol{F}^i$s are the forces that affect individual particles

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{x}_N \\ \boldsymbol{v}_N \end{pmatrix} \qquad f(\boldsymbol{X}, t) = \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{F}^1(\boldsymbol{X}, t) \\ \vdots \\ \boldsymbol{v}_N \\ \boldsymbol{F}^N(\boldsymbol{X}, t) \end{pmatrix}$$