Physically based modelling

15-462 Computer Graphics I
February 27, 2003

Outline

- Overview
- Particle systems
- Numerical solution of ODEs
- Constraints
- Collisions

Motivation

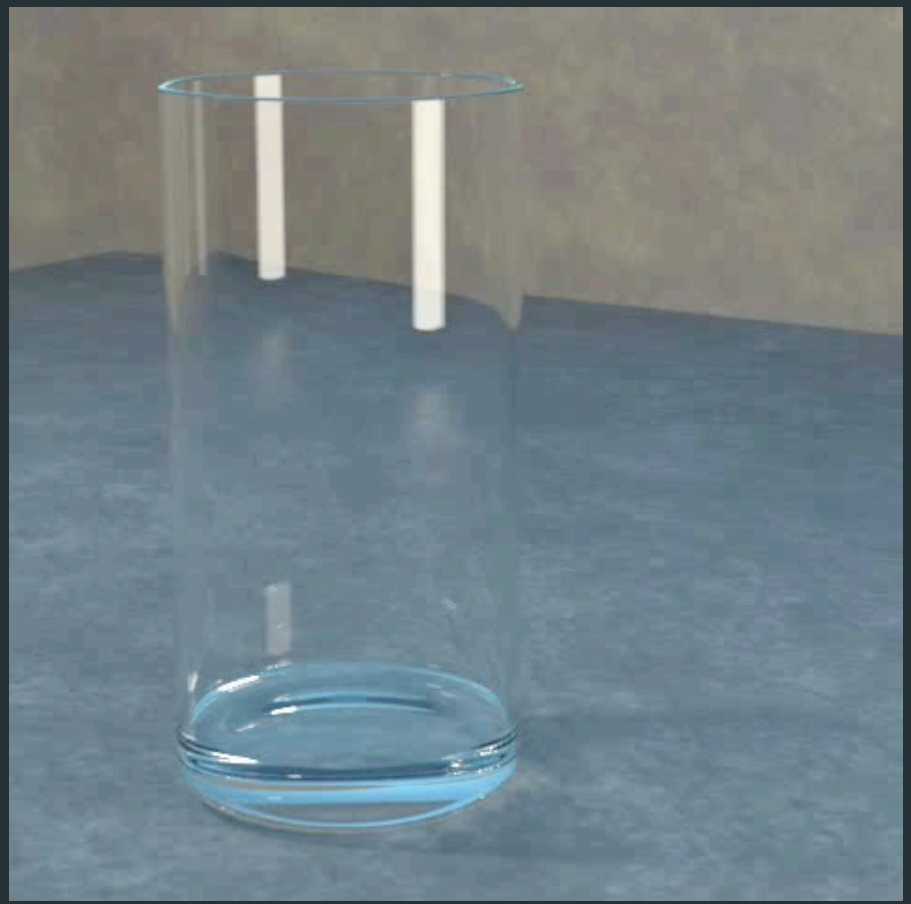
- Animation is hard!
- Secondary motion is important but difficult to keyframe: clothes, hair, etc.

James O'Brien, Jessica Hodgins, Victor Zordan [2000]

Alias | Wavefront cloth simulator (based on work by David Baraff and Andy Witkin [1998])

Motivation (2)

- Physical phenomena are too complicated to animate by hand; e.g., water, fire



Duc Nguyen, Ron Fedkiw, and Henrik Jensen [2002]

Doug Enright, Steve Marschner, and Ron Fedkiw [2002]

Motivation (3)

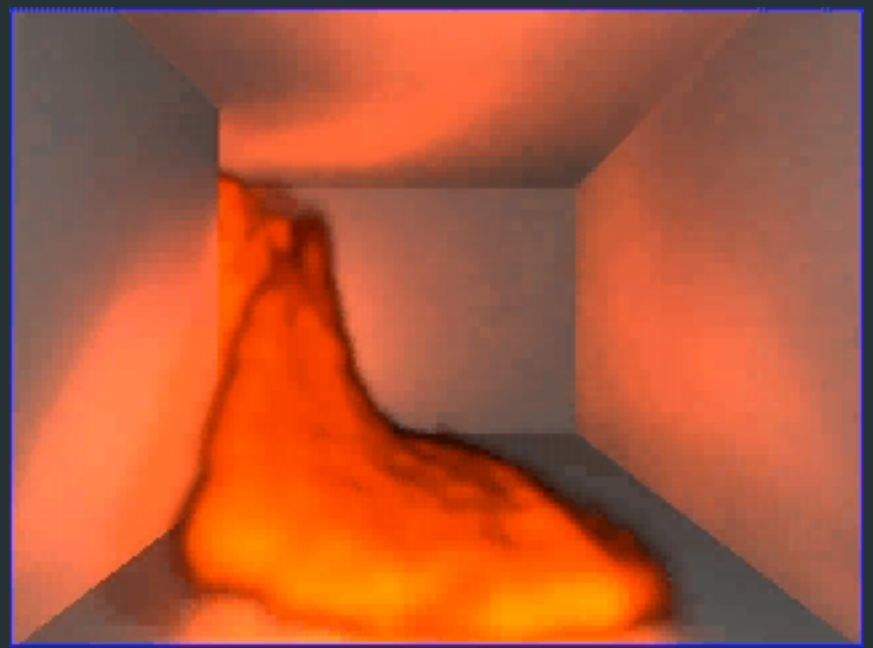
- Interaction is also important
  - Haptics (force-feedback)
  - Interactive simulation

Doug James, Dinesh Pai [2001]

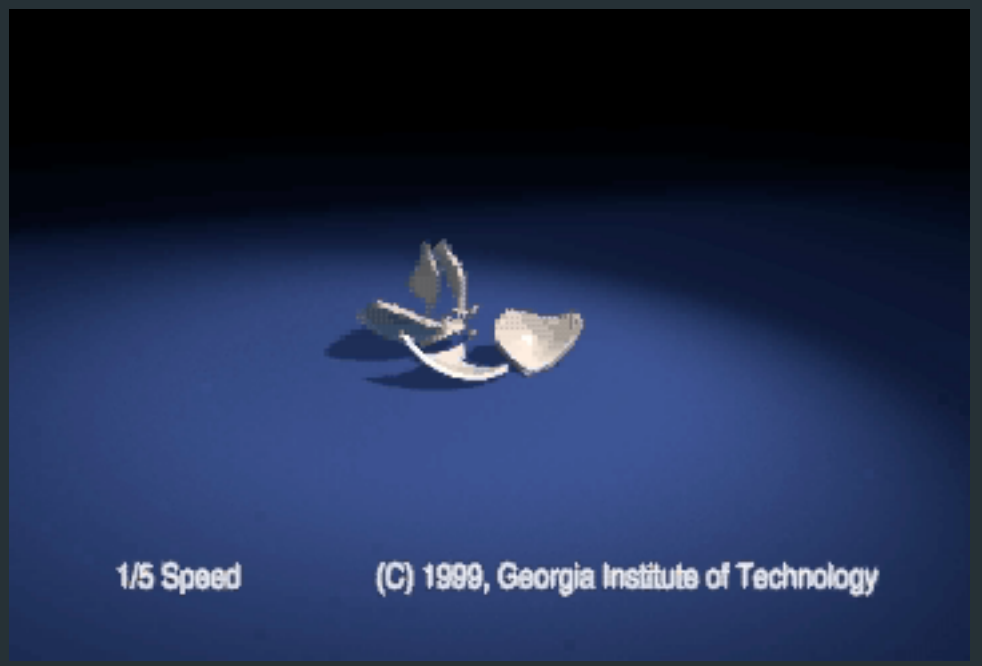
Doug James, Dinesh Pai [2002]

Motivation (4)

- It's cool!



Gary Yngve, James O'Brien, Jessica Hodgins [2000]



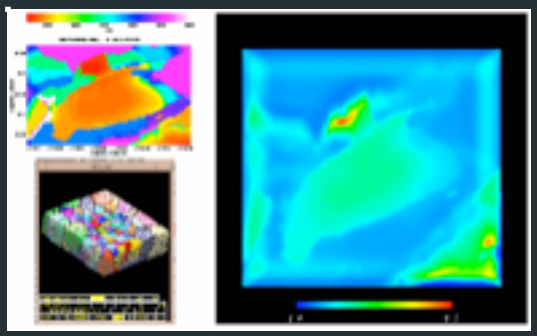
James O'Brien, Jessica Hodgins [1999]

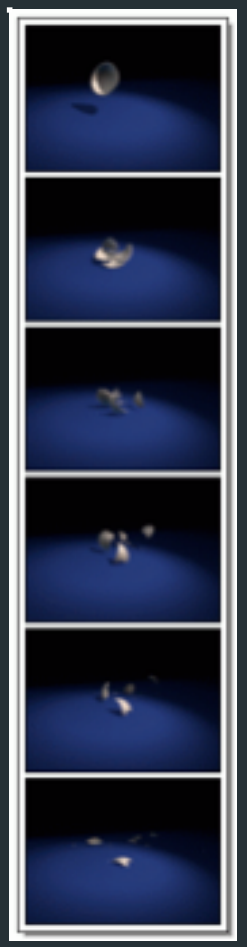
Physics to the rescue!

- Physically realistic motion simulated “for free”
- High-level control
  - Constraints
  - Forces
- Low-level details can be calculated

Scientific computing vs. graphics

- Scientific computation
  - Global error must be bounded
  - Can be really slow (e.g., the Quake Project)
- Graphics
  - Viewer's tolerance dominates
  - Numerical accuracy less important
  - Speed more important (especially for real-time!)
- Cross-fertilization
  - Scientific visualization
  - Physically based modelling

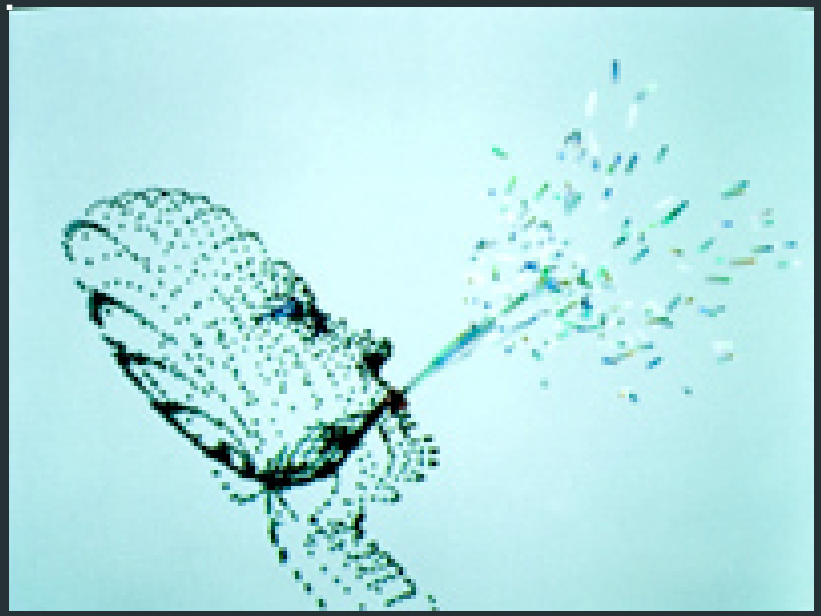




The basics: particle systems

What is a particle system?

- A collection of point masses, driven by forces
- Simplest possible physical model, but it can do a lot!
- Rendering is important



Karl Sims, *Particle Dreams* [1988]

Genesis effect from *Star Trek II: Wrath of Khan* [William Reeves, 1983]

One measly particle

We'll start out with a single point mass in the plane. It has:

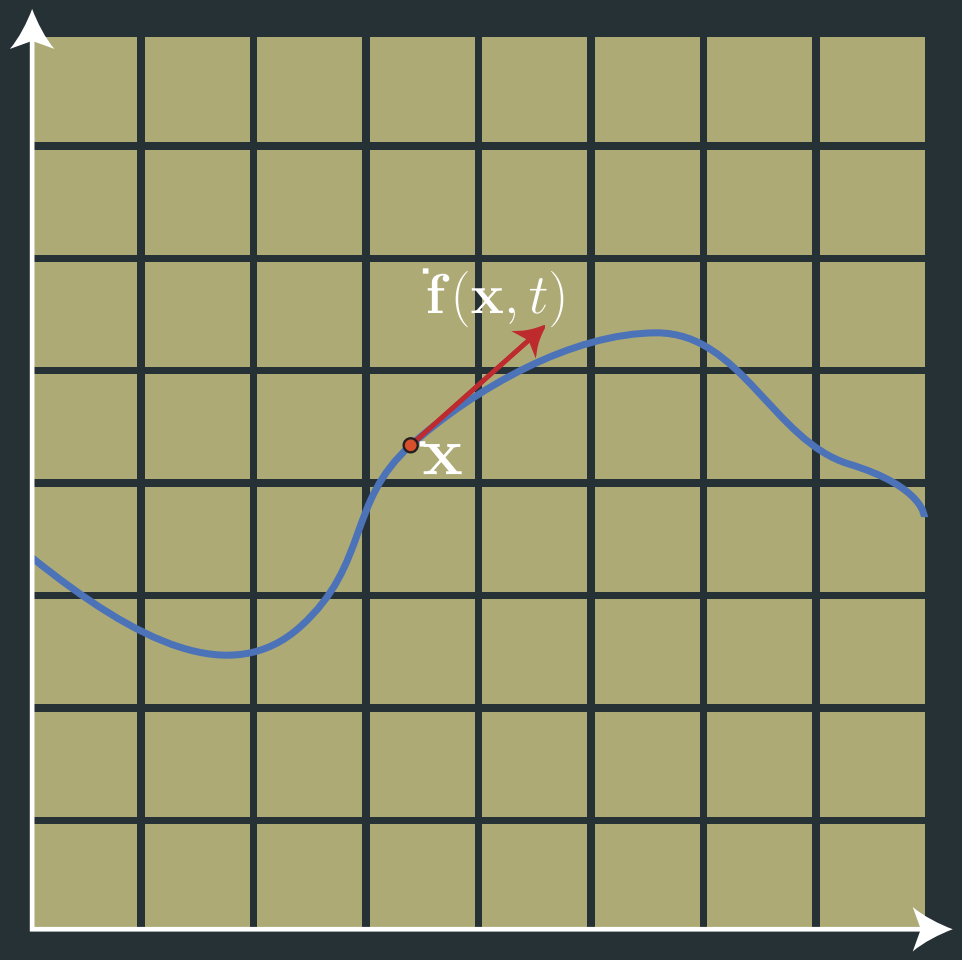
Mass, m

Position,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Velocity,

$$\mathbf{v} = \frac{d\mathbf{x}}{dt} = \begin{bmatrix} dx_1/dt \\ dx_2/dt \end{bmatrix}$$



Suppose velocity is dictated by some vector-valued function, $\mathbf{v} = \mathbf{f}(\mathbf{x}, t)$

Solving the ODE

This is a first-order ordinary differential equation,

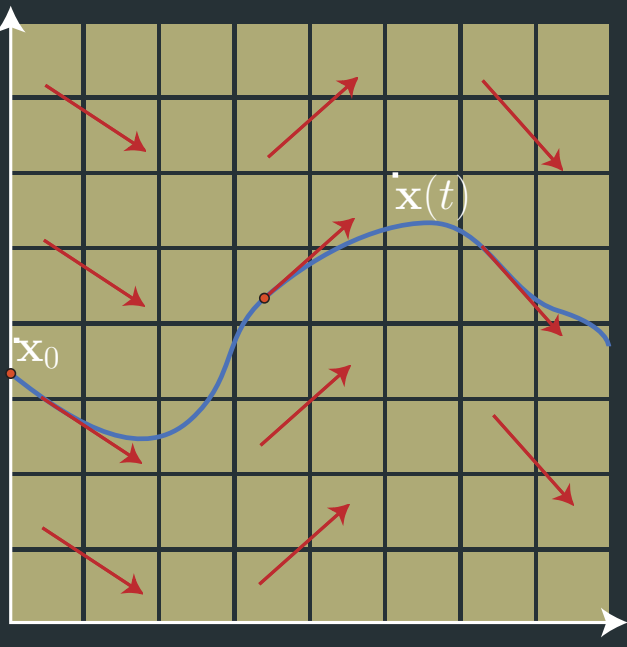
$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t)$$

We need to specify an initial value for x,

$$\mathbf{x}(t_0) = \mathbf{x}_0$$

The solution is a curve that is tangent to f at every point.

How do we find it?



Euler's method

Recall the Taylor expansion of a function,

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + (\Delta t)\frac{d\mathbf{x}}{dt} + \frac{1}{2}(\Delta t)^2\frac{d^2\mathbf{x}}{dt^2} + \cdots$$

We can approximate this with just the first term,

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + (\Delta t)\frac{d\mathbf{x}}{dt} + O(\Delta t^2)$$

Recall that in our case,

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

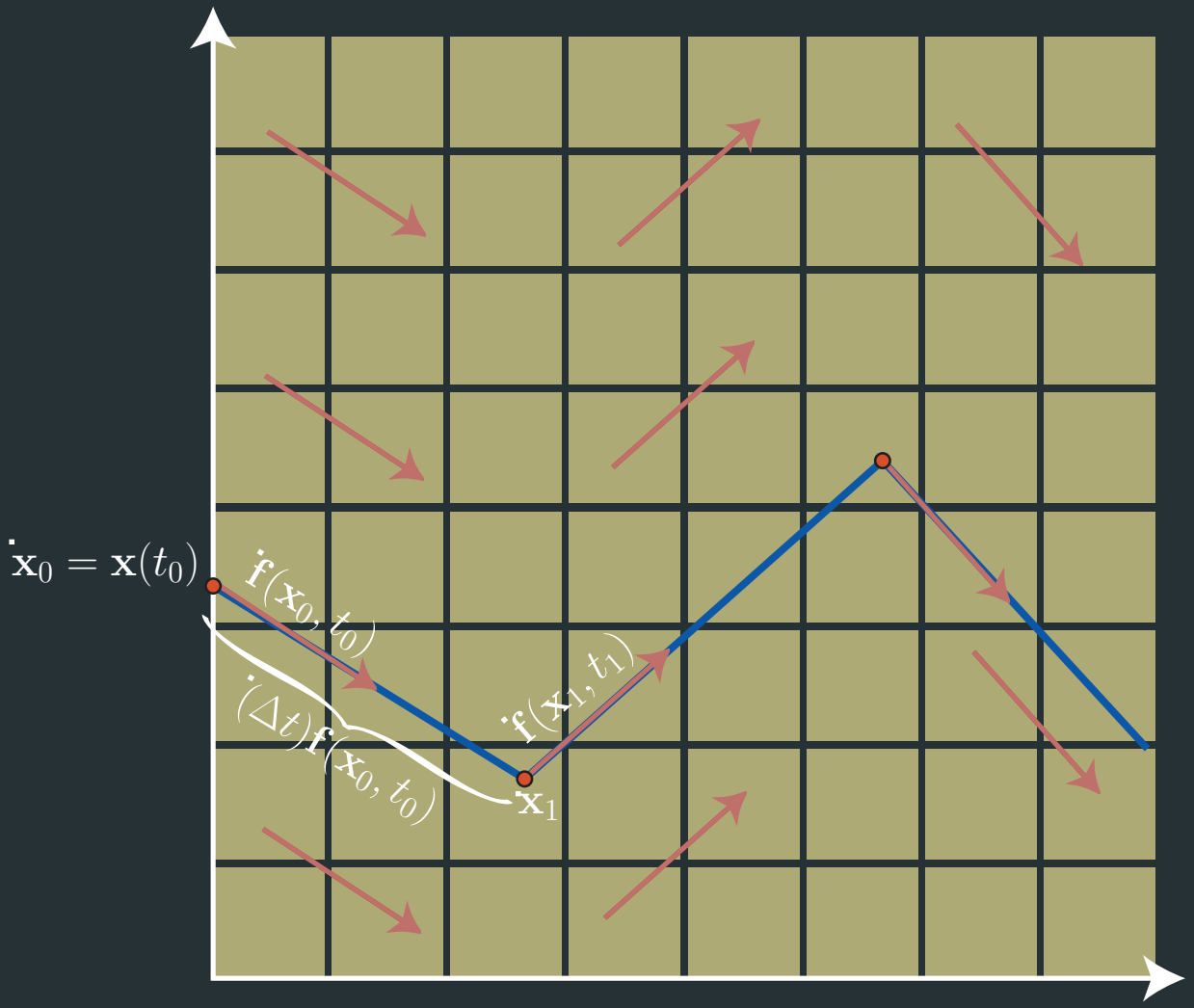
Hence,

$$\mathbf{x}(t+\Delta t) \approx \mathbf{x}(t) + (\Delta t)\mathbf{f}(\mathbf{x}, t)$$

This is known as "Euler's method."

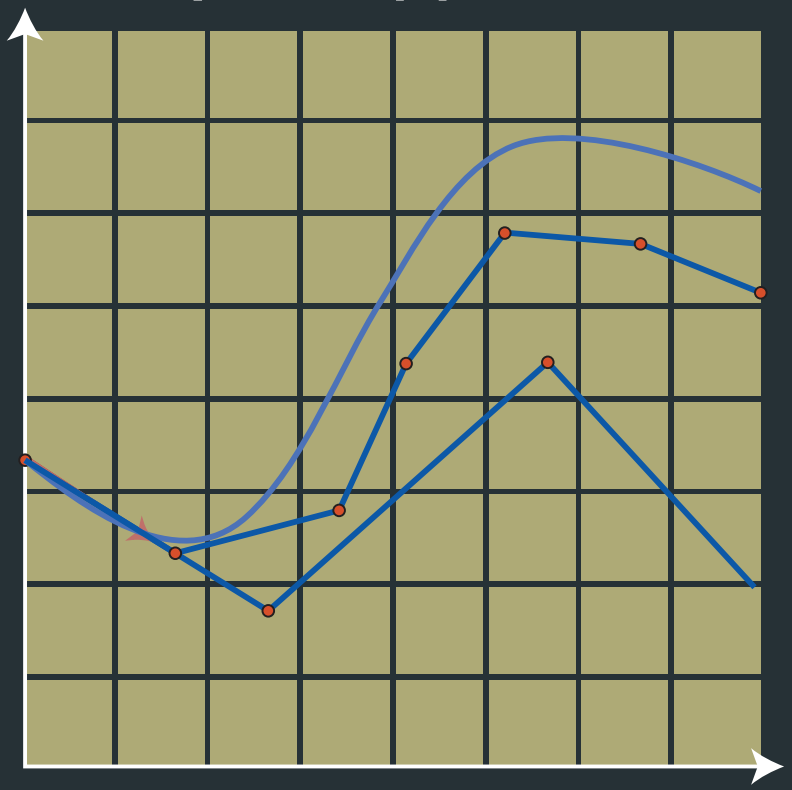
Euler's method (2)

What does this look like?



Euler's method: error

Of course, this is only an approximation.



Error is $O(\Delta t^2)$, so we can get a more accurate answer by reducing the timestep (within limits).

This can be expensive, though.

Higher-order methods

We could get also better accuracy by using more terms from the Taylor expansion,

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + (\Delta t)\frac{d\mathbf{x}}{dt} + \frac{1}{2}(\Delta t)^2\frac{d^2\mathbf{x}}{dt^2} + \cdots$$

This means taking more derivatives, though (and who wants to do that?)

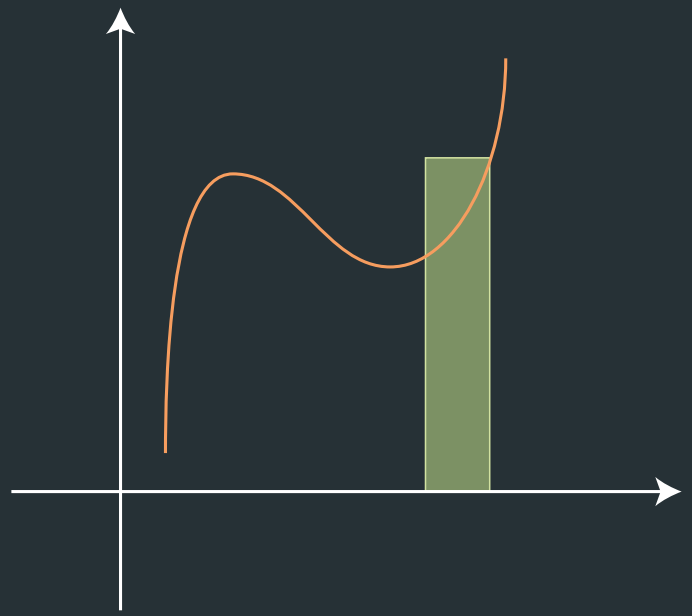
Higher-order methods (2)

We can look at it another way; given x_t, we find $x_{t+\Delta t}$ by

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \int_{\tau=t}^{t+\Delta t} \frac{d\mathbf{x}}{dt}\, d\tau = \mathbf{x}_t + \int_{\tau=t}^{t+\Delta t} \mathbf{f}(\mathbf{x}(\tau), \tau)\, d\tau$$

In the Euler method, we are approximating the integral with the lower sum,

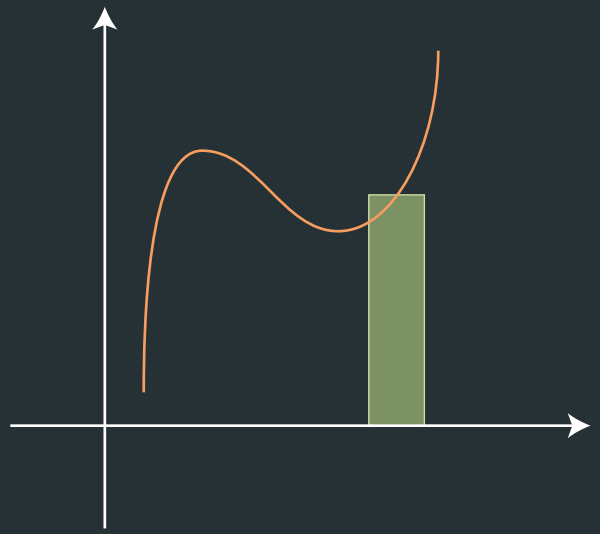
$$\int_{t}^{t+\Delta t} \mathbf{f}(\mathbf{x}(\tau), \tau)\, d\tau \approx \Delta t \mathbf{f}(\mathbf{x}_t, t)$$



Higher-order methods (3)

A more accurate method is to use the midpoint of the interval,

$$\int_{\tau=t}^{t+\Delta t} \mathbf{f}(\mathbf{x}(\tau),\tau)\,d\tau \approx (\Delta t)\mathbf{f}\left(\mathbf{x}_{t+\Delta t/2}, t+\frac{\Delta t}{2}\right)$$



Of course, we don't yet know what $x_{t+\Delta t/2}$ is, so we approximate it with (what else?) the Euler method,

$$\mathbf{x}_{t+\Delta t/2} \approx \mathbf{x}_t + \frac{\Delta t}{2}\mathbf{f}(\mathbf{x}_t, t)$$

Higher-order methods (4)

The resulting method is

$$\mathbf{x}_{t+\Delta t/2} = \mathbf{x}_t + \frac{\Delta t}{2}\mathbf{f}(\mathbf{x}_t, t)$$

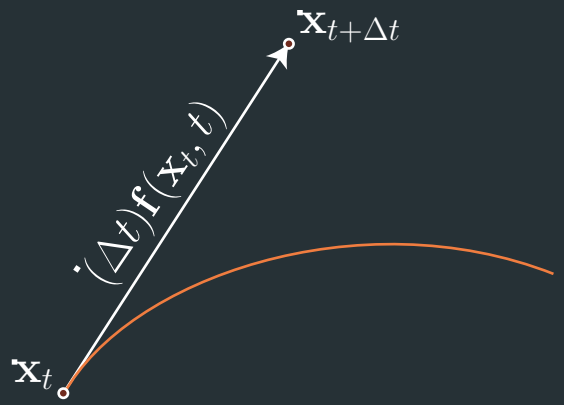
$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + (\Delta t)\mathbf{f}(\mathbf{x}_{t+\Delta t/2}, t + \Delta t/2)$$

This is known as the "midpoint method." The error is $O(\Delta t^3)$, but it requires two evaluations of the function f.

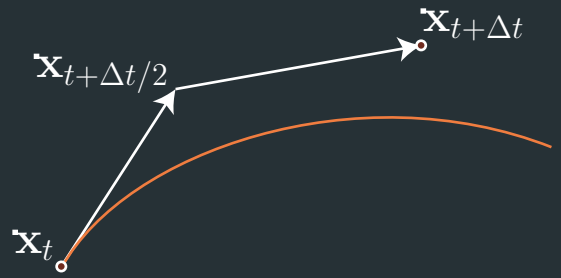
In general, methods of this class are known as "Runge-Kutta" methods.

Higher-order methods (5)

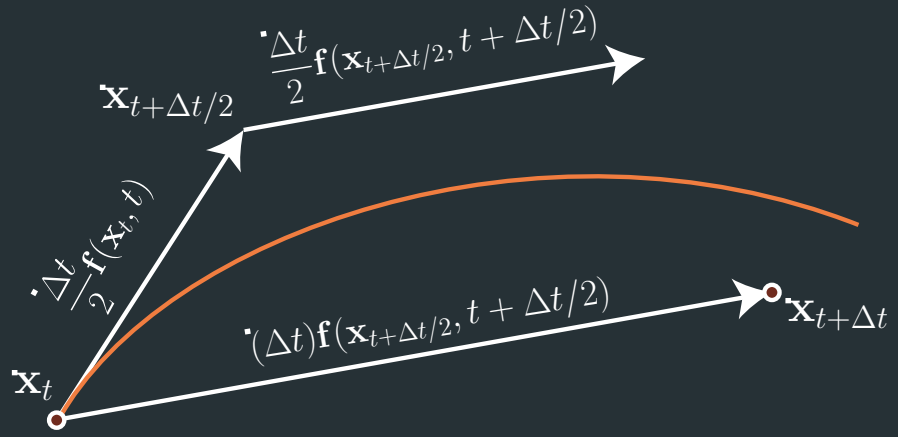
How does this compare to the Euler method?



Single full Euler step
(One function evaluation)



Two half Euler steps
(Two function evaluations)



Midpoint Runge-Kutta step
(Two function evaluations)

Euler's method: stiffness

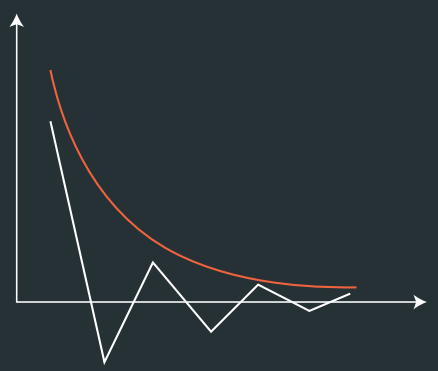
Error isn't the only thing we need to worry about, though. Consider the following (in 1D)

$$\frac{dx}{dt} = -kx, \; k \gg 1$$

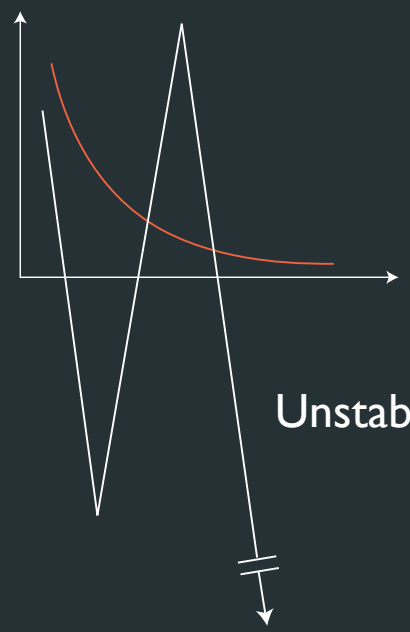
The analytic solution is

$$x(t) = Ce^{-kt}$$

What happens when we use Euler's method?



Barely stable



Unstable

Euler's method: stiffness (2)

The solution is a class of methods called *implicit methods*, but these are beyond the scope of the course.

See for example David Baraff and Andy Witkin's notes at:

http://www.pixar.com/companyinfo/research/pbm2001/

Newton's law

In general, particles are governed by Newton's law,

$$\mathbf{f} = m\mathbf{a} = m\frac{d^2\mathbf{x}}{dt^2}$$

What is the problem here?

Our solution? Introduce a new variable, v:

$$\left[\begin{array}{l} \frac{d\mathbf{x}}{dt} = \mathbf{v} \\ \frac{d\mathbf{v}}{dt} = \frac{\mathbf{f}}{m} \end{array}\right]$$

State space formulation

If concatenate position and velocity, we get a 6-vector in "phase space,"

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix}$$

With this, we rewrite Newton's law as

$$\frac{d}{dt}\begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{f}/m \end{bmatrix}$$

which we already know how to solve.

State space formulation (2)

Conveniently, we can easily extend this to a system of n particles,

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{v}_1 \\ \mathbf{x}_2 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{x}_n \\ \mathbf{v}_n \end{bmatrix}$$

Our ODE is just the obvious extension,

$$\frac{d}{dt} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{v}_1 \\ \mathbf{x}_2 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{x}_n \\ \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{f}_1/m_1 \\ \mathbf{v}_2 \\ \mathbf{f}_2/m_2 \\ \vdots \\ \mathbf{v}_n \\ \mathbf{f}_n/m_n \end{bmatrix}$$

Implementation

We can treat the ODE solver as a black box; this lets us use different solvers if we choose.

We need to provide to the solver

- Dimensionality of the system (6n for n particles)
- Initial state
- Starting and ending times
- How to calculate derivatives of the state variables (we will supply a function callback)

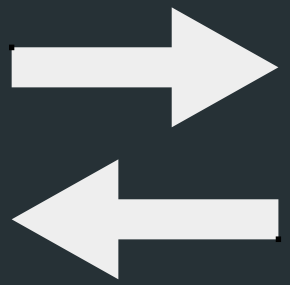
Implementation (2)

Particle system

State

$\mathbf{x}_1$
$\mathbf{v}_1$
$\mathbf{x}_2$
$\mathbf{v}_2$
$\vdots$
$\mathbf{x}_n$
$\mathbf{v}_n$

getState()
setState()
derivEval()
dimension()



ODE solver

odeSolve()

Main loop:

```
InitializeState()
while( simulating )
        odesolve( 6n, t, t+Δt,
          state, derivEval );
        renderParticles();
```

Working with particle systems

Particle system design

We have the machinery now to deal with big systems of particles; it's time to make them do something useful.

First, we'll need some forces...

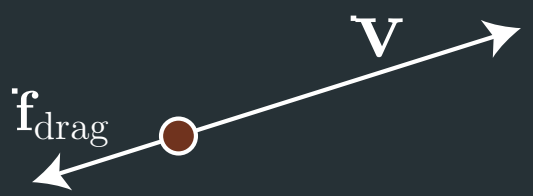
Forces: gravity

$$\mathbf{f}_\mathrm{g} = m\mathbf{g}$$

Forces: viscous drag

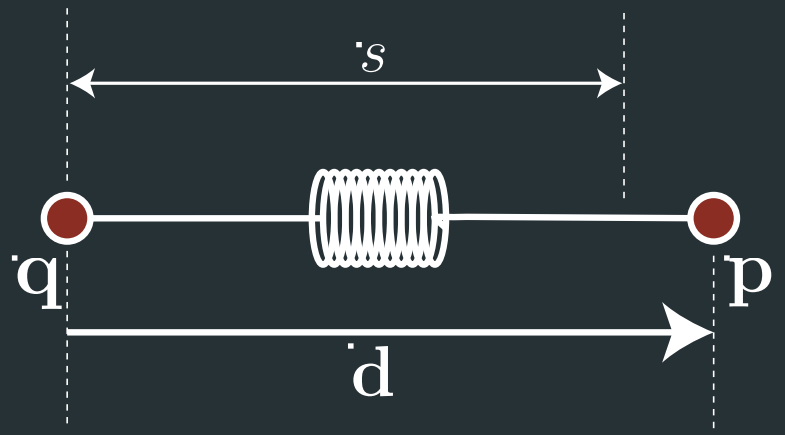
Opposes velocity.

$$\mathbf{f}_{\text{drag}} = -k_{\text{drag}}\mathbf{v}$$



Forces: spring forces

Things start to get interesting when we connect particles together...



Recall Hook's law,

$$\mathbf{f}_{\text{spring}} = -k_s(||\mathbf{d}|| - s)\frac{\mathbf{d}}{||\mathbf{d}||}$$

where

$\mathbf{d} = \mathbf{p} - \mathbf{q}$

s is the *rest length* of the spring

k_s is the spring constant

Damped spring

In general, we will need to add a damping force to get stable, realistic motion. It should

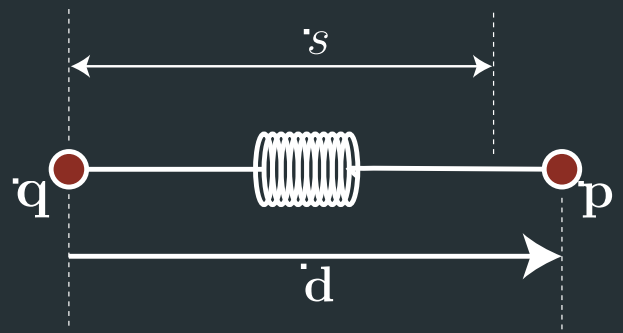
- Depend on velocity
- Act in the same direction of the spring force
- Be proportional to the projection of the velocity vector onto the distance vector

Our force looks like

$$\dot{\mathbf{f}}_{\text{spring}} = -\left(k_s(||\mathbf{d}|| - s) + k_d \frac{\dot{\mathbf{d}} \cdot \mathbf{d}}{||\mathbf{d}||}\right) \frac{\mathbf{d}}{||\mathbf{d}||}.$$

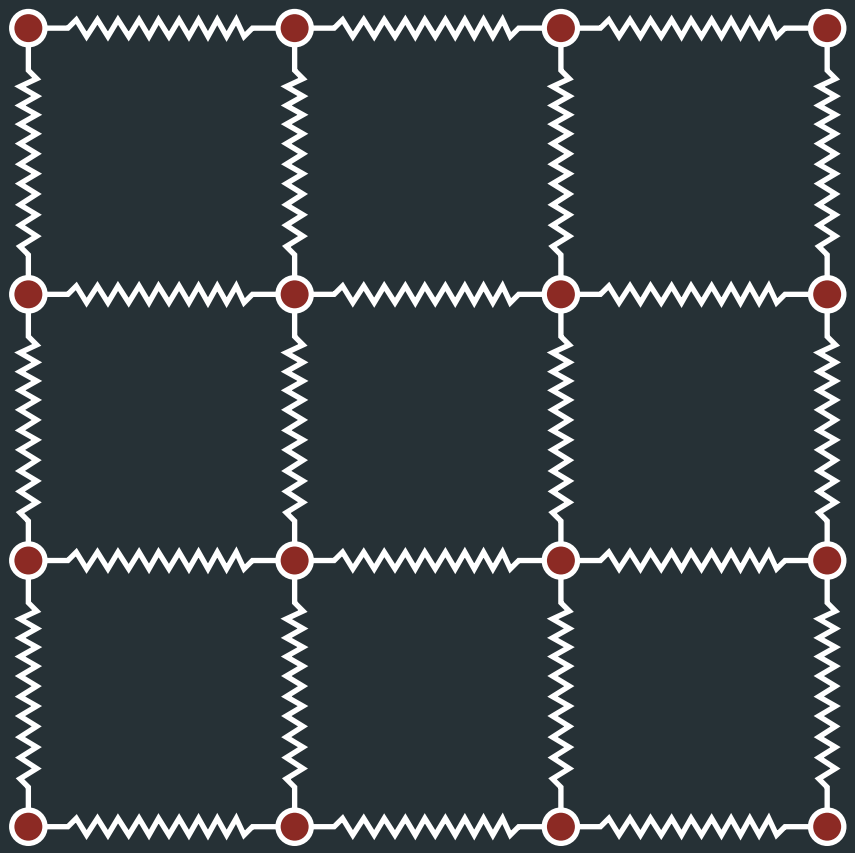
where

$$\dot{\mathbf{d}} = \frac{d\mathbf{d}}{dt} = \frac{d\mathbf{p}}{dt} - \frac{d\mathbf{q}}{dt}$$



Spring-mass systems

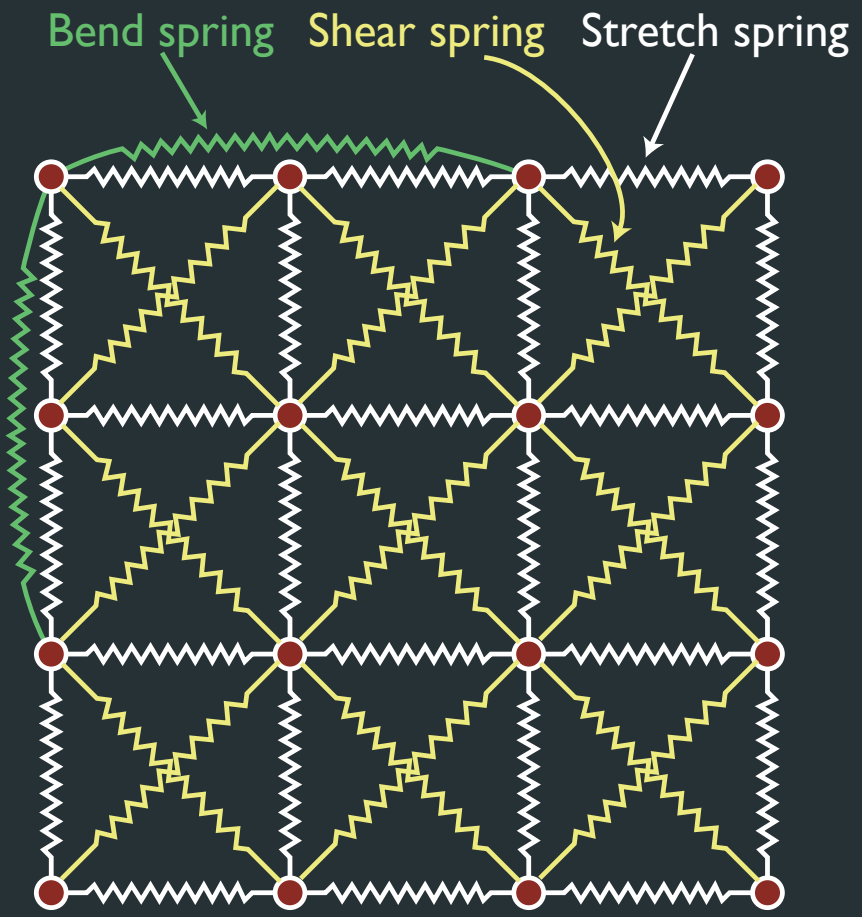
We can connect a bunch of masses with springs to model things like rope (1D), cloth (2D), jello (3D), etc.



Models like these tend to be pretty jiggly unless we make the springs really stiff (and then stability is a problem)

Cloth

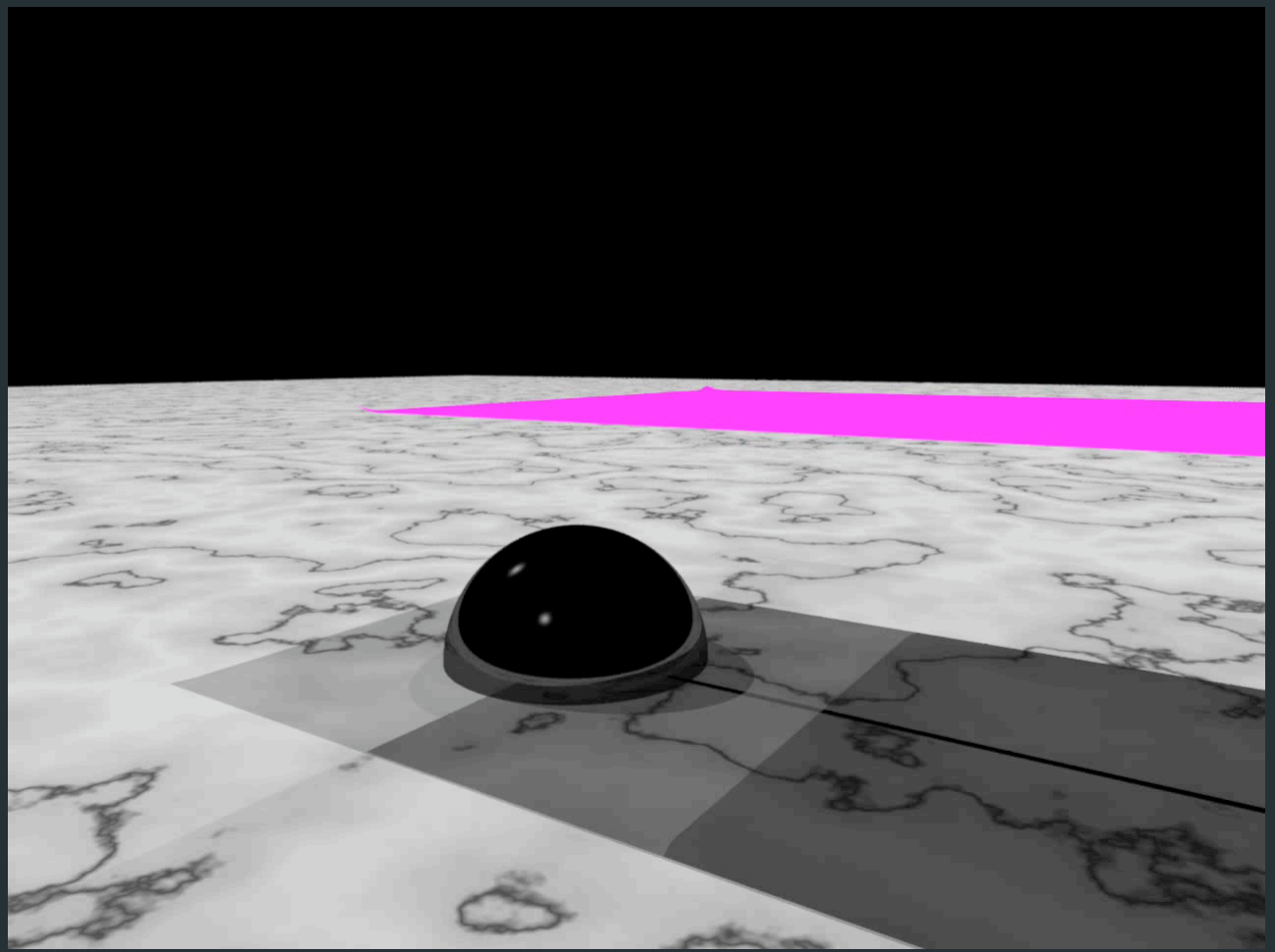
For realistic-looking cloth, we'll need shear springs and bend springs, so it doesn't collapse on itself.



More sophisticated models are available (active area of research).

Cloth (2)

Example of spring-mass cloth. Note that this took 10 hours to calculate.



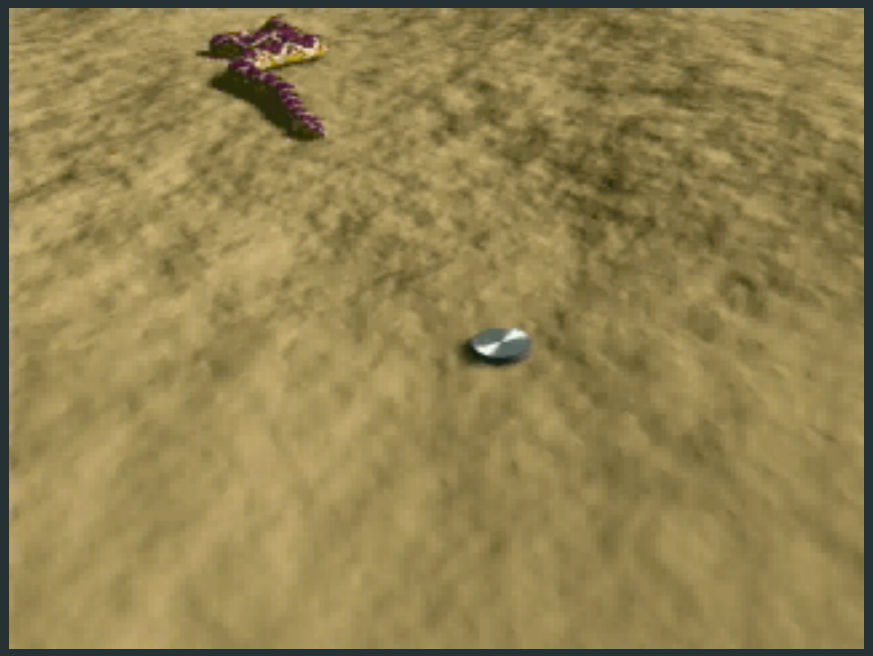
Robert Bridson, Ron Fedkiw, John Anderson [2002]

Other variables

- Creation
  - Number created
  - Initial position and velocity
  - Shape, size, color
  - Lifetime
  - Randomness
- Deletion
- Rendering
  - Alpha blending can be quite effective (especially for fire)
  - Material properties may change with age

Other particle systems

- Clouds
- Smoke
- Fire
- Crowds
- Fish? Snakes? You bet!



Xiaoyuan Tu, *Go Fish* [1993]

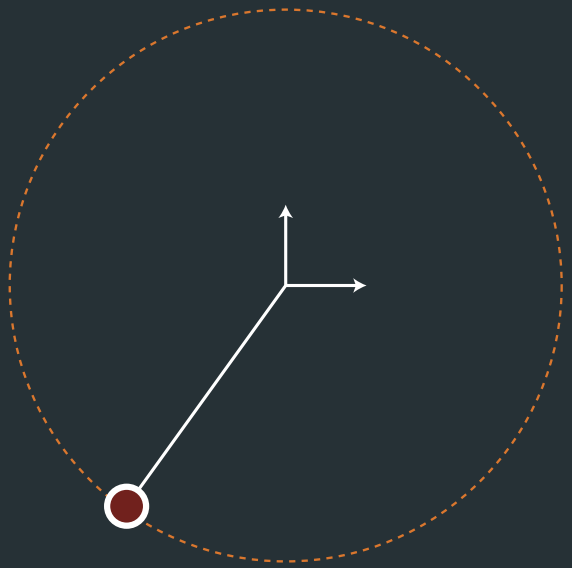
Gavin Miller, *Her Majesty's Secret Serpent* [1989]

Constraints

Constraints: definitions

Often, our particle system will consist of not only forces but *constraints* as well.

e.g., suppose we have a rigid pendulum constrained to lie on a circle about the origin.



In practice, these will come up for example in collision handling.

Constraints: definitions (2)

In general, we can rewrite our ODE as

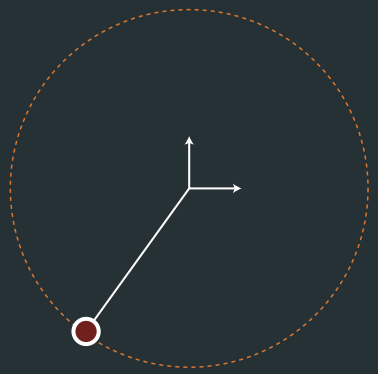
$$\frac{d^2\mathbf{x}}{dt^2} = \mathbf{f}(\mathbf{x}, t)$$
$$\mathbf{g}(\mathbf{x}) = 0$$

Here, the g function represents the constraints that must be satisfied.

This is known as an ODE on a manifold, and is a special case of a class of problems known as *Differential Algebraic Equations (DAEs)*. Unlike ODEs, these are often ill-posed and can be very difficult to solve.

Constraints: reducing dimensionality

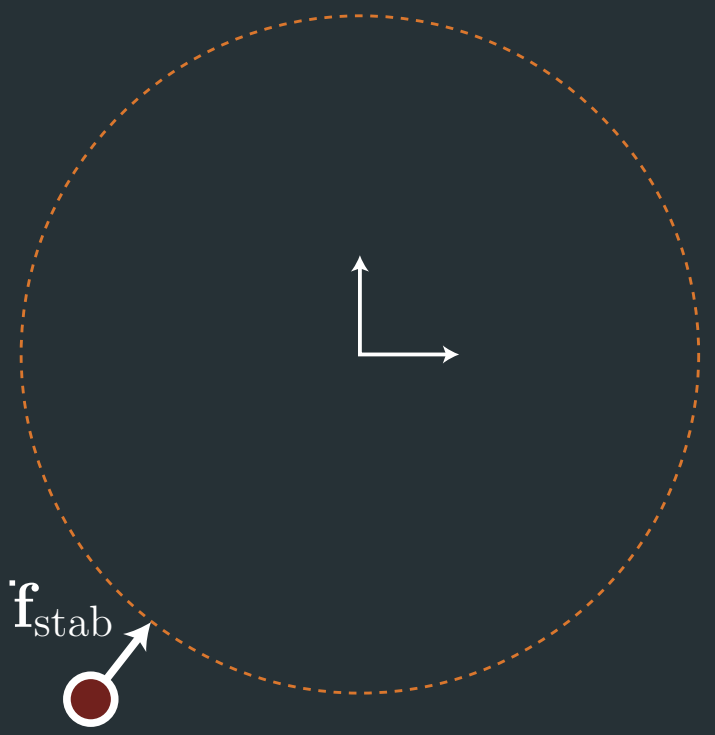
The best solution is often to try reducing the dimensionality of the problem. In our circle example:



Notice that the particle here has only 1 degree of freedom. If we can express the ODE in terms of θ, the constraint will be satisfied "for free."

Constraints: stabilization

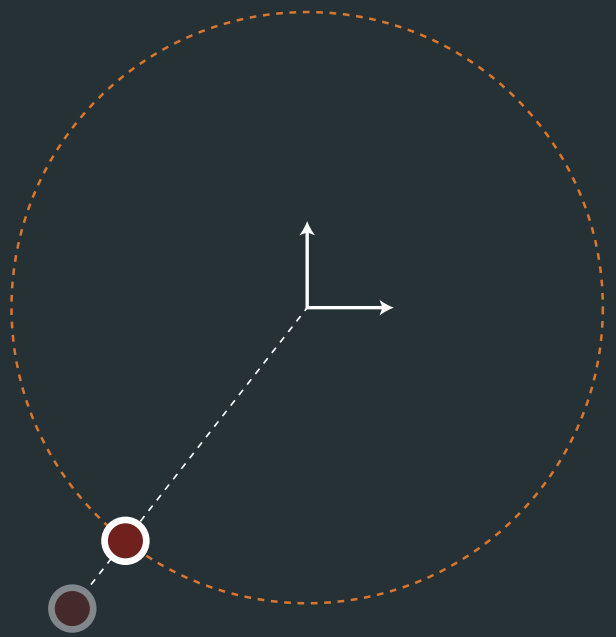
Another common method is to try and *penalize* the particle if it disobeys the constraint, e.g., with a force. This is a simple form of the general method known as "stabilization."



See the Baraff and Witkin notes for all the gory details.

Constraints: poststabilization

Another idea is to solve the ODE as usual, but at the end of the timestep *project* x onto the constraint manifold. This is a form of *poststabilization*.



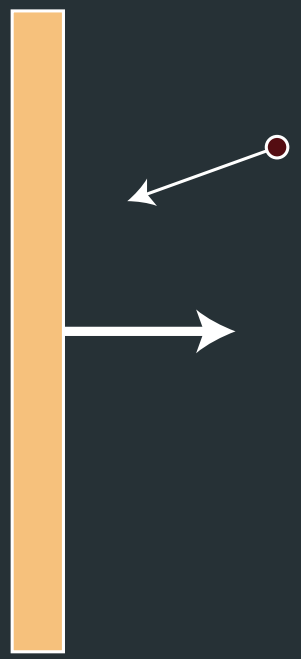
In this case this is easy, but in general it might involve a nonlinear optimization.

Collisions

Collisions: detection

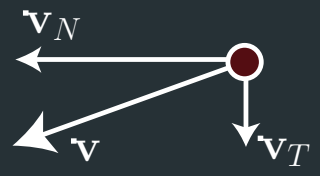
We start with the simple case of a particle bouncing off a wall.

How do we detect when the particle has crossed the plane?

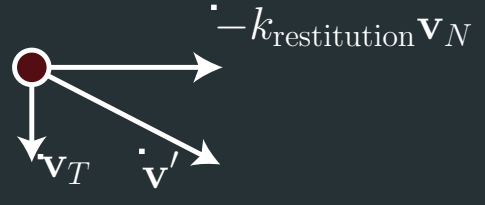


Collisions: response

First decompose velocity into tangent and normal components:

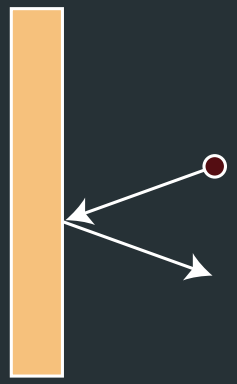


Multiply normal velocity by a restitution coefficient, which determines how much energy is preserved.

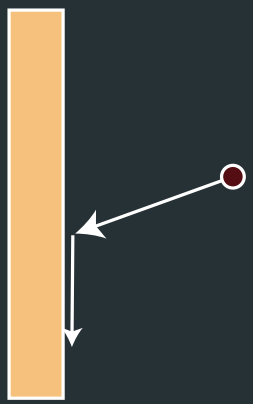


Collisions: response (2)

- Totally elastic collision ($k_{restitution} = 1$)

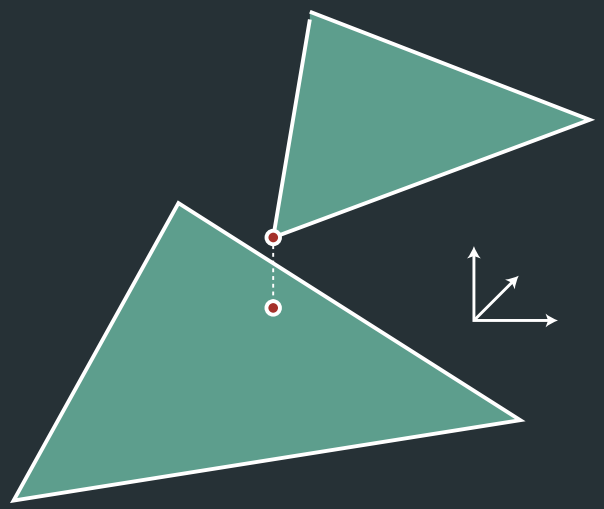


- Totally inelastic collision ($k_{restitution} = 0$)
  - May need to introduce a friction force to oppose v_T

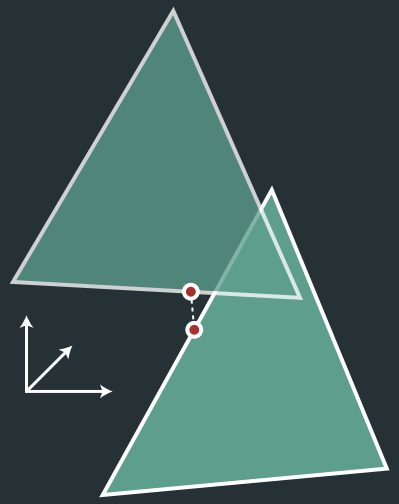


Triangle collisions

In general, we will need to compute collisions between arbitrary pairs of triangles,



point-face collision



edge-edge collision

If we have a lot of triangles, this is slow, but we can accelerate it with bounding box hierarchies.

Summary

- Particle systems
- Numerical solution of differential equations
- Constraints
- Collisions