Introduction to Cloth

Christopher Twigg

15-864 Spring 2004

Outline

- Review of physically based modeling
- A simple cloth system
- Implementation details

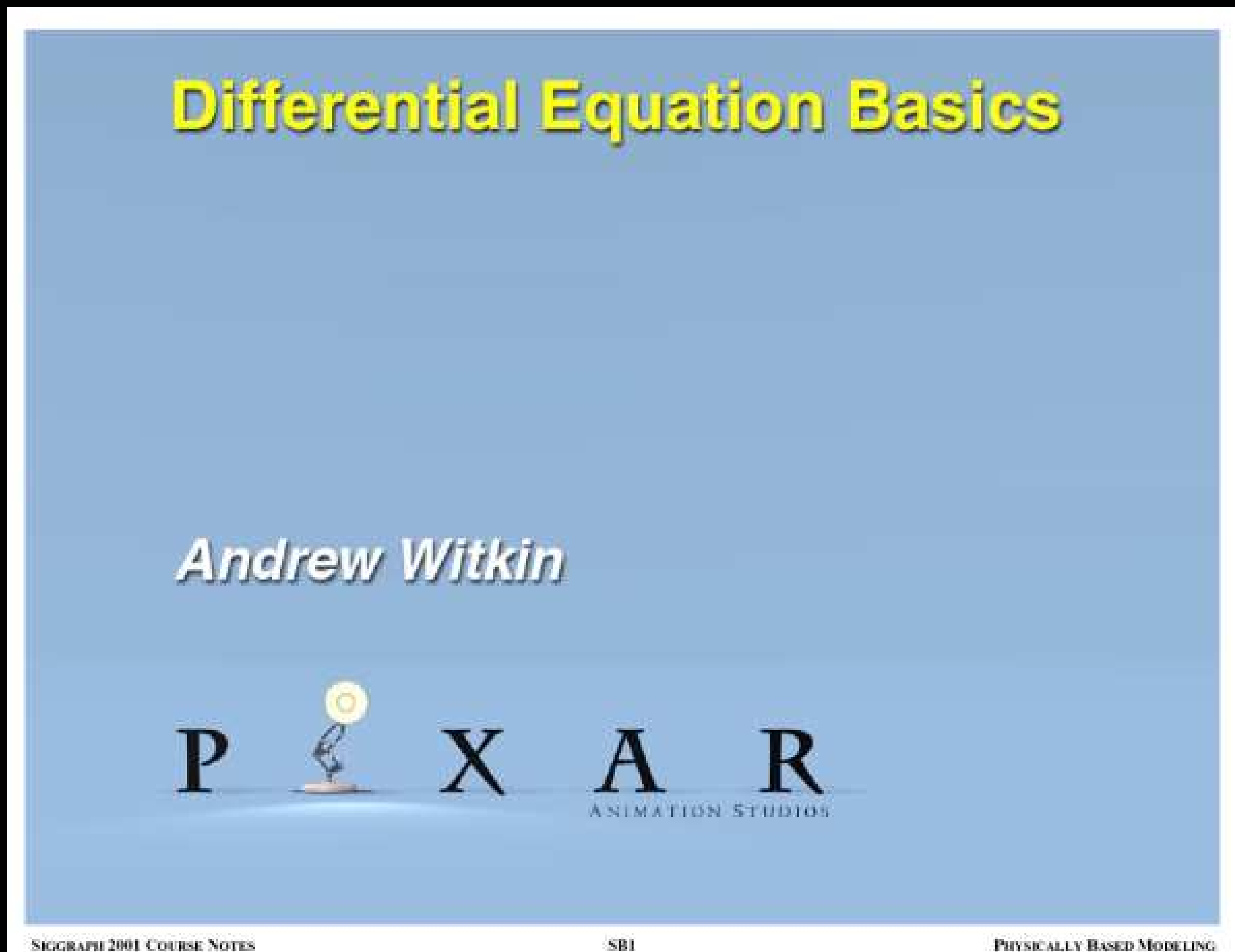
Review of PBM basics

$$F = ma^{*}$$

* The rest is just details

Recall: Particles

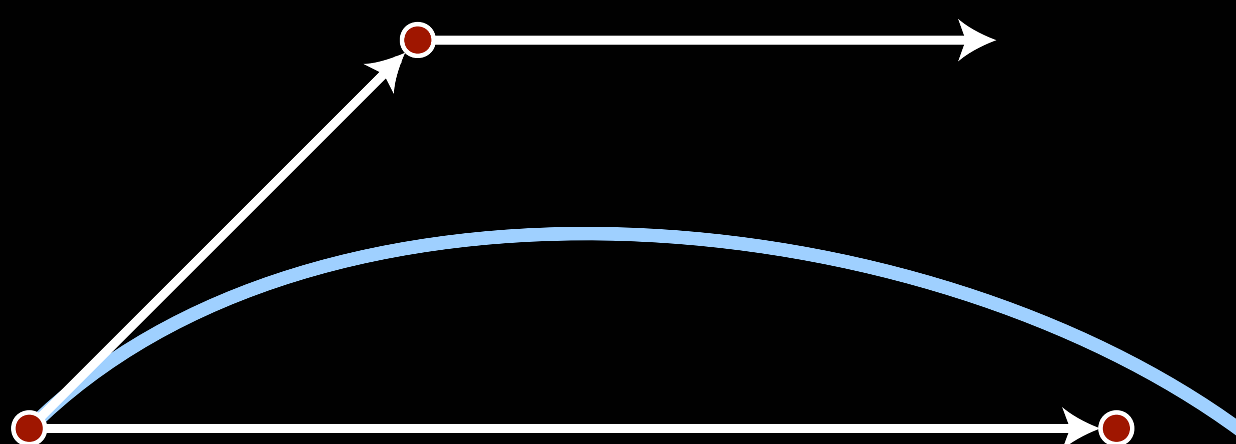
Recall: ODE Basics



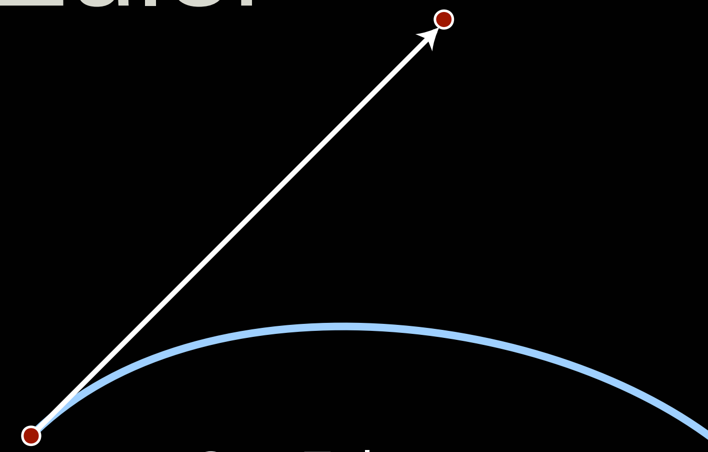
Midpoint method

$$x_{t+\Delta t/2} = x_t + \frac{\Delta t}{2} f(x_t, t)$$

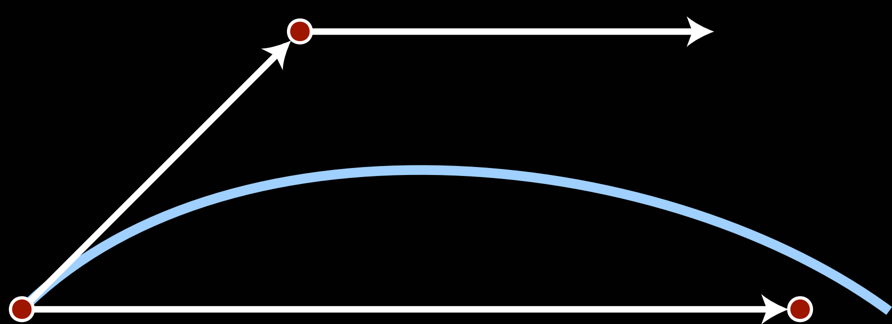
$$x_{t+\Delta t} = x_t + (\Delta t) f(x_{t+\Delta t/2}, t + \Delta t/2)$$



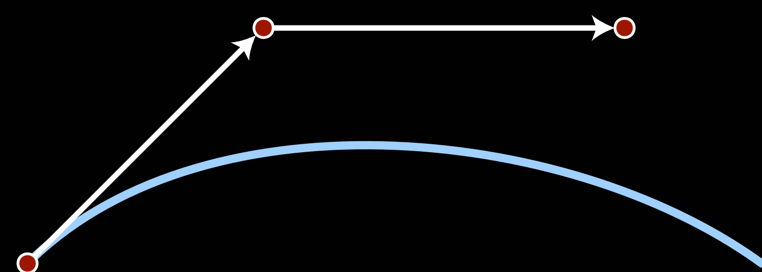
Midpoint vs. Euler



One Euler step

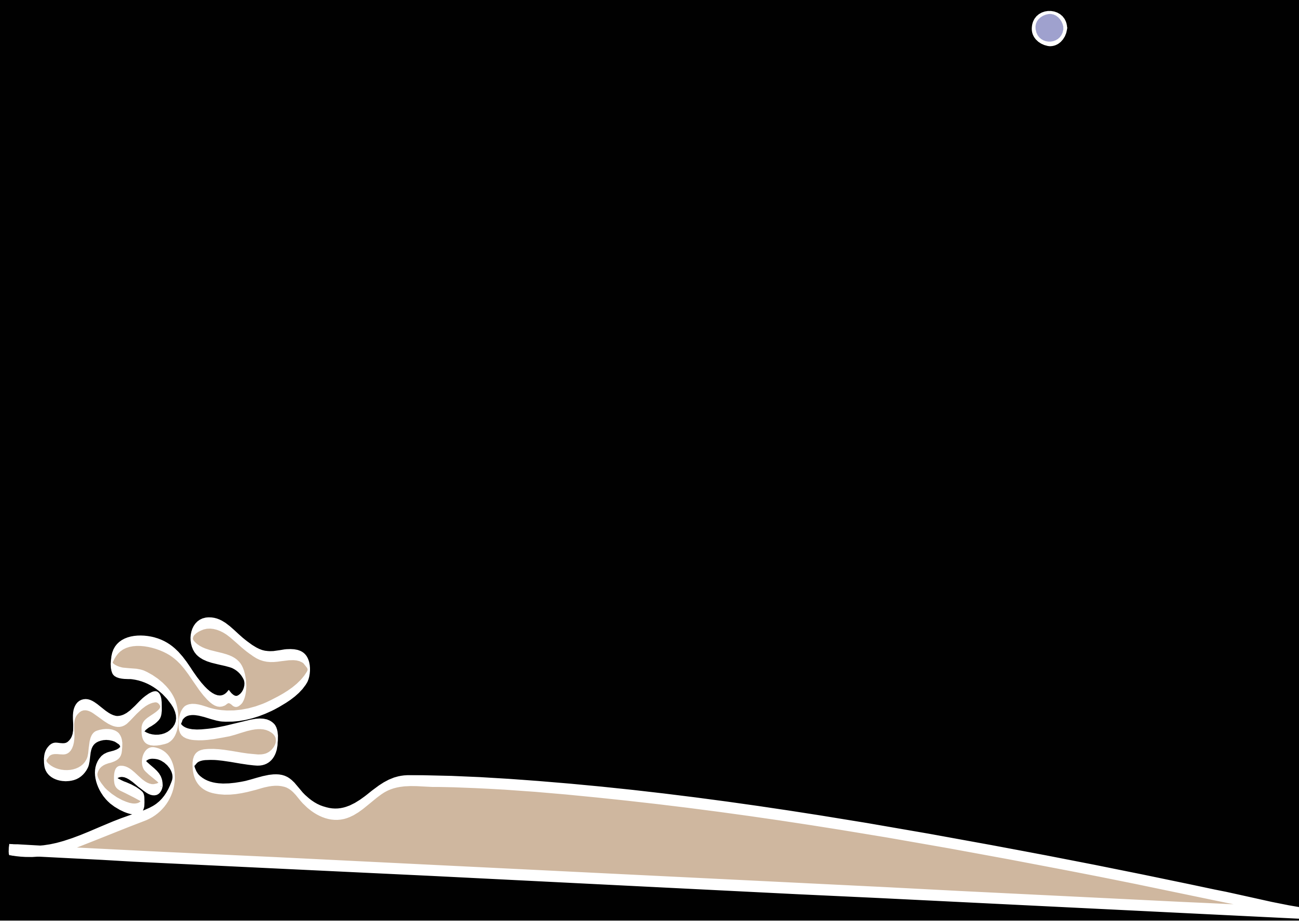


Midpoint method

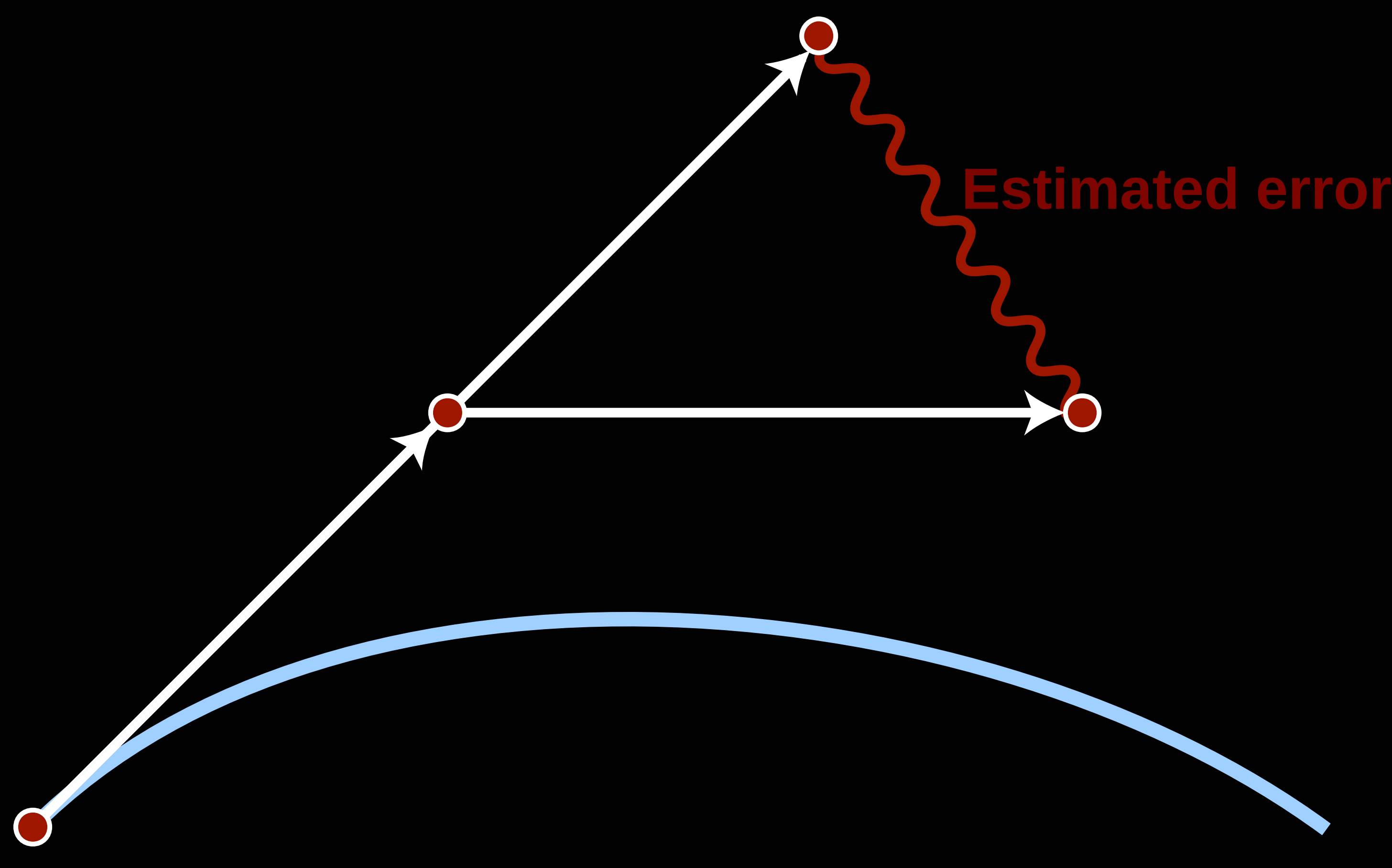


Two Euler steps

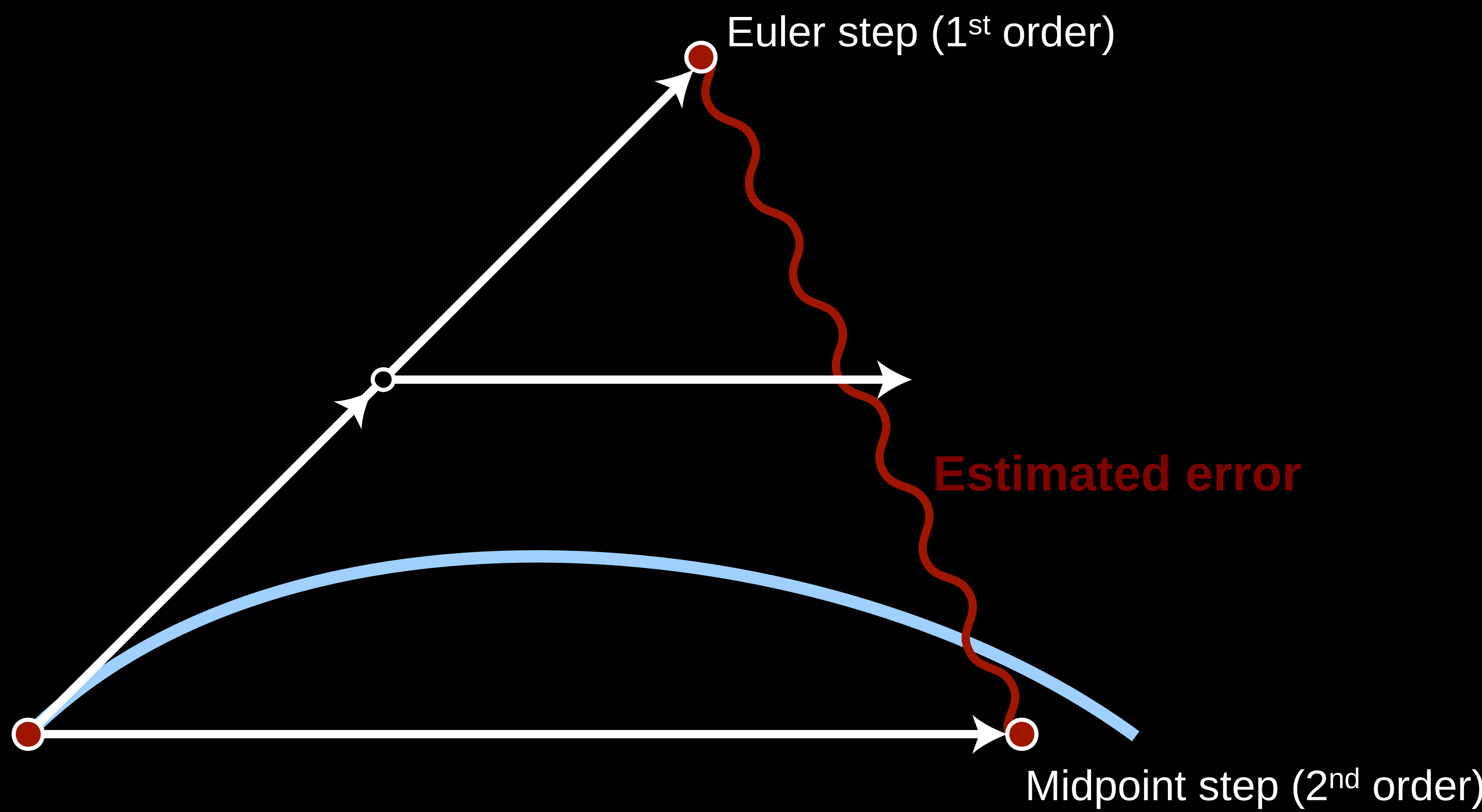
Adaptive stepping



Idea 1: step doubling



Idea 2: Steps of different order



Idea 3: Embedded pairs

- Compare methods of different orders, but:
- Reuse derivatives

Idea 4: Use somebody else's code

- Because who wants to write code like this?

```
static float a2=0.2,a3=0.3,a4=0.6,a5=1.0,a6=0.875,b21=0.2,
    b31=3.0/40.0,b32=9.0/40.0,b41=0.3,b42 = -0.9,b43=1.2,
    b51 = -11.0/54.0, b52=2.5,b53 = -70.0/27.0,b54=35.0/27.0,
    b61=1631.0/55296.0,b62=175.0/512.0,b63=575.0/13824.0,
    b64=44275.0/110592.0,b65=253.0/4096.0,c1=37.0/378.0,
    c3=250.0/621.0,c4=125.0/594.0,c6=512.0/1771.0,
    dc5 = -277.00/14336.0;
float dc1=c1-2825.0/27648.0,dc3=c3-18575.0/48384.0,
    dc4=c4-13525.0/55296.0,dc6=c6-0.25;
float *ak2,*ak3,*ak4,*ak5,*ak6,*ytemp;

ak2=vector(1,n);
ak3=vector(1,n);
ak4=vector(1,n);
ak5=vector(1,n);
ak6=vector(1,n);
ytemp=vector(1,n);
for (i=1;i<=n;i++)                    First step.
    ytemp[i]=y[i]+b21*h*dydx[i];
(*derivs)(x+a2*h,ytemp,ak2);          Second step.
for (i=1;i<=n;i++)
    ytemp[i]=y[i]+h*(b31*dydx[i]+b32*ak2[i]);
(*derivs)(x+a3*h,ytemp,ak3);          Third step.
for (i=1;i<=n;i++)
    ytemp[i]=y[i]+h*(b41*dydx[i]+b42*ak2[i]+b43*ak3[i]);
(*derivs)(x+a4*h,ytemp,ak4);          Fourth step.
for (i=1;i<=n;i++)
    ytemp[i]=y[i]+h*(b51*dydx[i]+b52*ak2[i]+b53*ak3[i]+b54*ak4[i]);
(*derivs)(x+a5*h,ytemp,ak5);          Fifth step.
for (i=1;i<=n;i++)
    ytemp[i]=y[i]+h*(b61*dydx[i]+b62*ak2[i]+b63*ak3[i]+b64*ak4[i]+b65*ak5[i]);
(*derivs)(x+a6*h,ytemp,ak6);          Sixth step.
for (i=1;i<=n;i++)                    Accumulate increments with proper weights.
    yout[i]=y[i]+h*(c1*dydx[i]+c3*ak3[i]+c4*ak4[i]+c6*ak6[i]);
for (i=1;i<=n;i++)
    yerr[i]=h*(dc1*dydx[i]+dc3*ak3[i]+dc4*ak4[i]+dc5*ak5[i]+dc6*ak6[i]);
    Estimate error as difference between fourth and fifth order methods.
```

Review: Second-order systems

Remember this guy?

$$F = ma$$

Review: Second-order systems

To be more precise:

$$F = m\frac{d^2x}{dt^2}$$

Review: Second-order systems

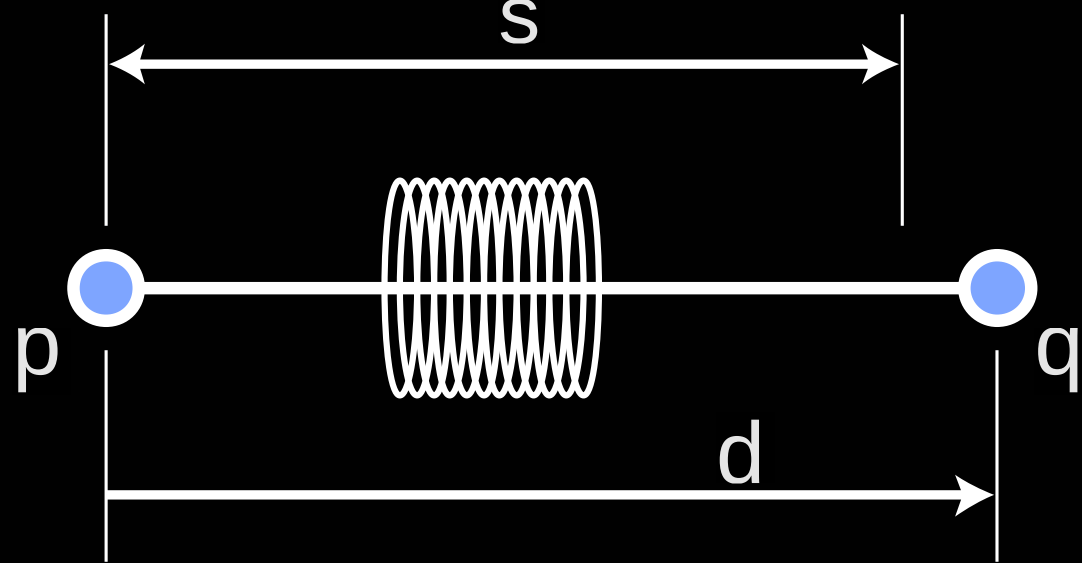
- Fortunately, we can rewrite this as

$$\frac{dx}{dt} = v$$

$$\frac{dv}{dt} = \frac{F}{m}$$

A simple cloth system

Recall: spring-mass systems



$$f_{spring} = -k_s(||d|| - s)\frac{d}{||d||}$$

where

$d = p - q$

s is the *rest length* of the spring

k_s is the *spring constant*

Damped spring

- Add a simple damping force for stability
- Properties:
  - Depends on *relative velocity* in the *direction of force*
  - Acts in direction of spring force

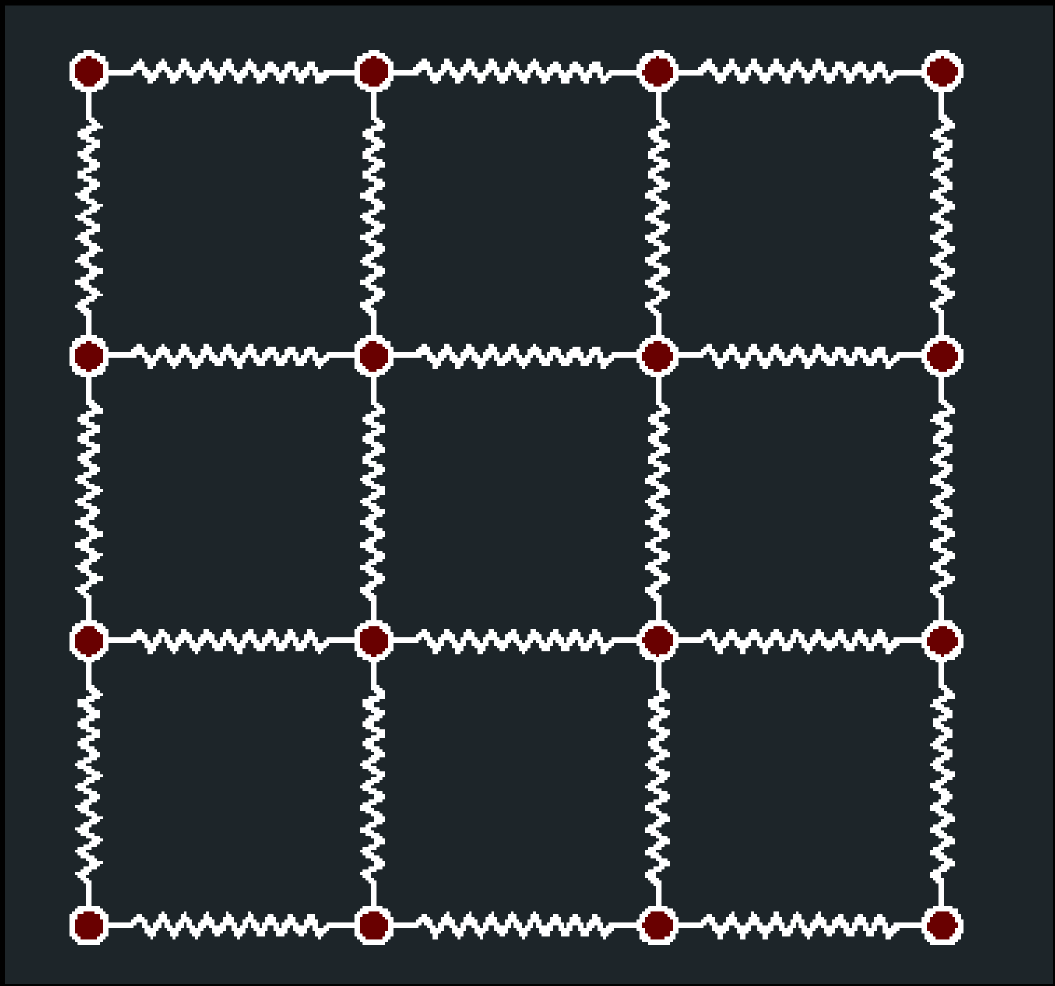
$$\mathbf{f}_{spring} = -\left(k_s(\|\mathbf{d}\| - s) + k_d \frac{\dot{\mathbf{d}} \cdot \mathbf{d}}{\|\mathbf{d}\|}\right) \frac{\mathbf{d}}{\|\mathbf{d}\|}$$

where

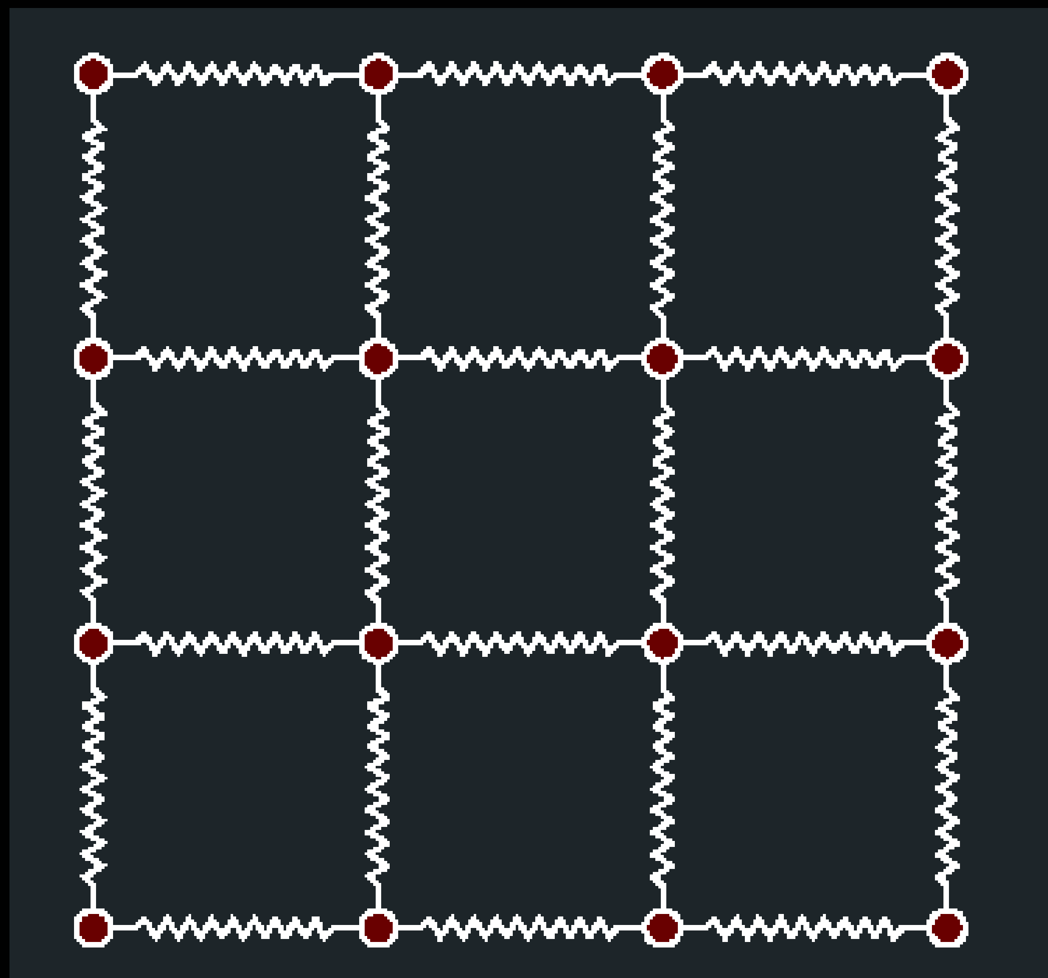
$$\dot{\mathbf{d}} = \frac{d\mathbf{p}}{dt} - \frac{d\mathbf{q}}{dt}$$

Simple spring-based cloth

Now, we just start connecting things up…

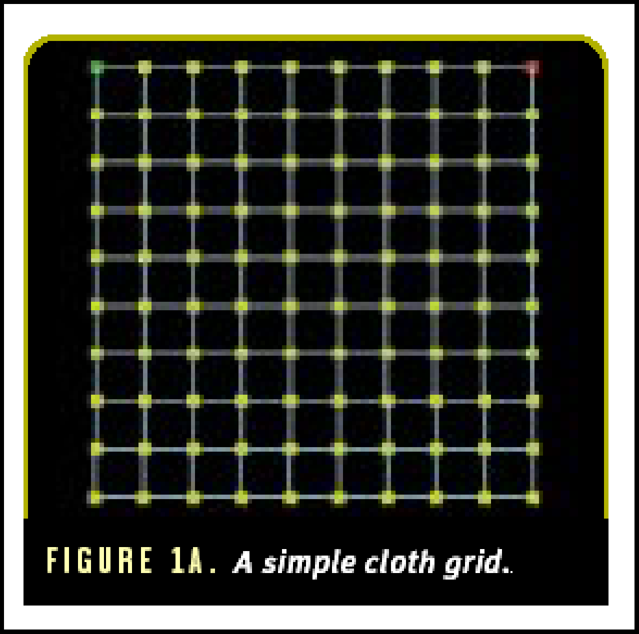


Step 1: Stretch springs



Step 1: Stretch springs

Oops…



Source: [Lander 1999]

Step 2: Shear springs

Better…

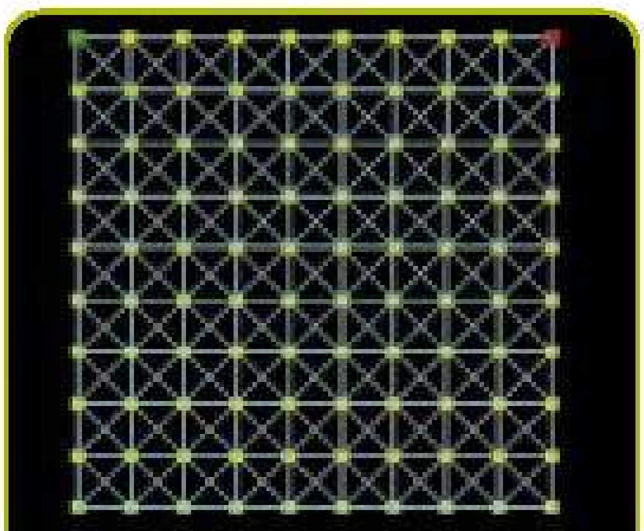


FIGURE 2A. *Added shear springs.*

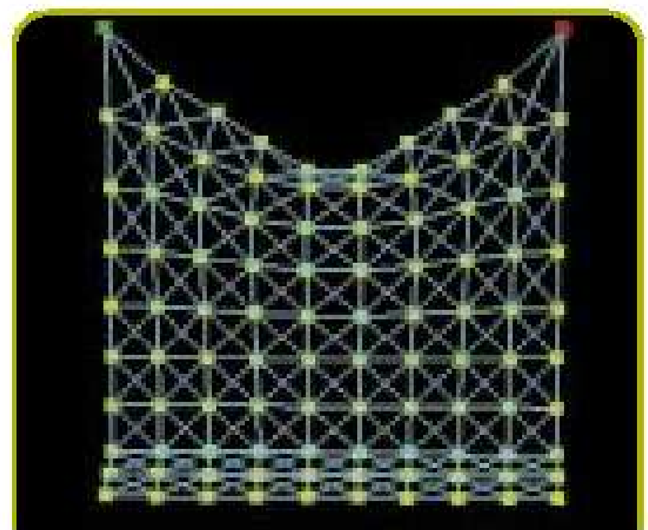
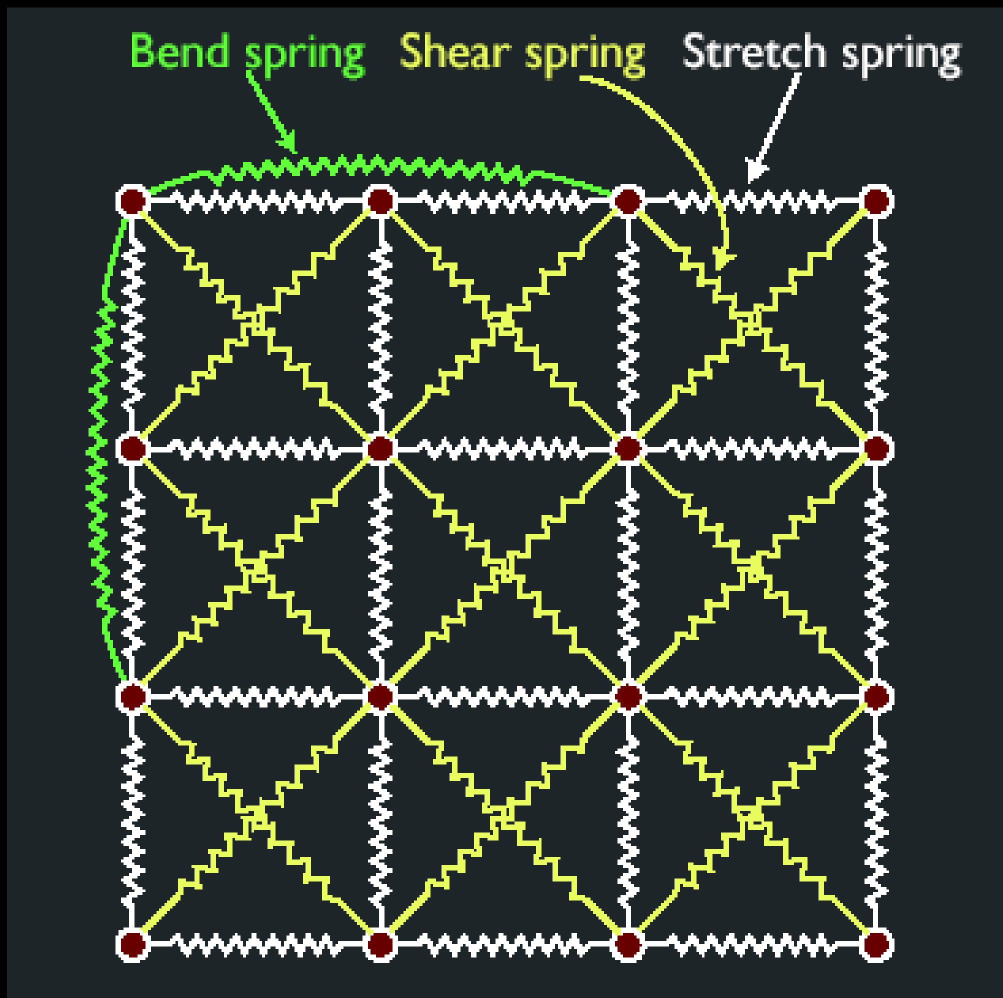


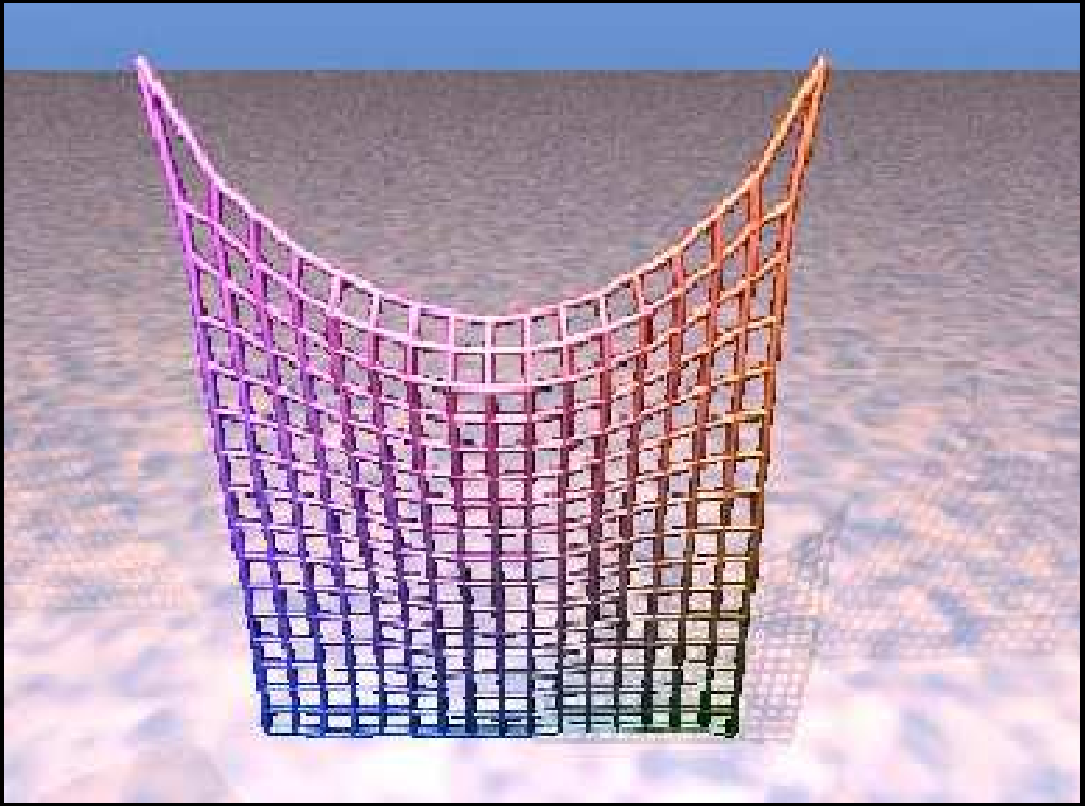
FIGURE 2B. *A better cloth model.*

Source: [Lander 1999]

Step 3: Bend springs



Are we done?

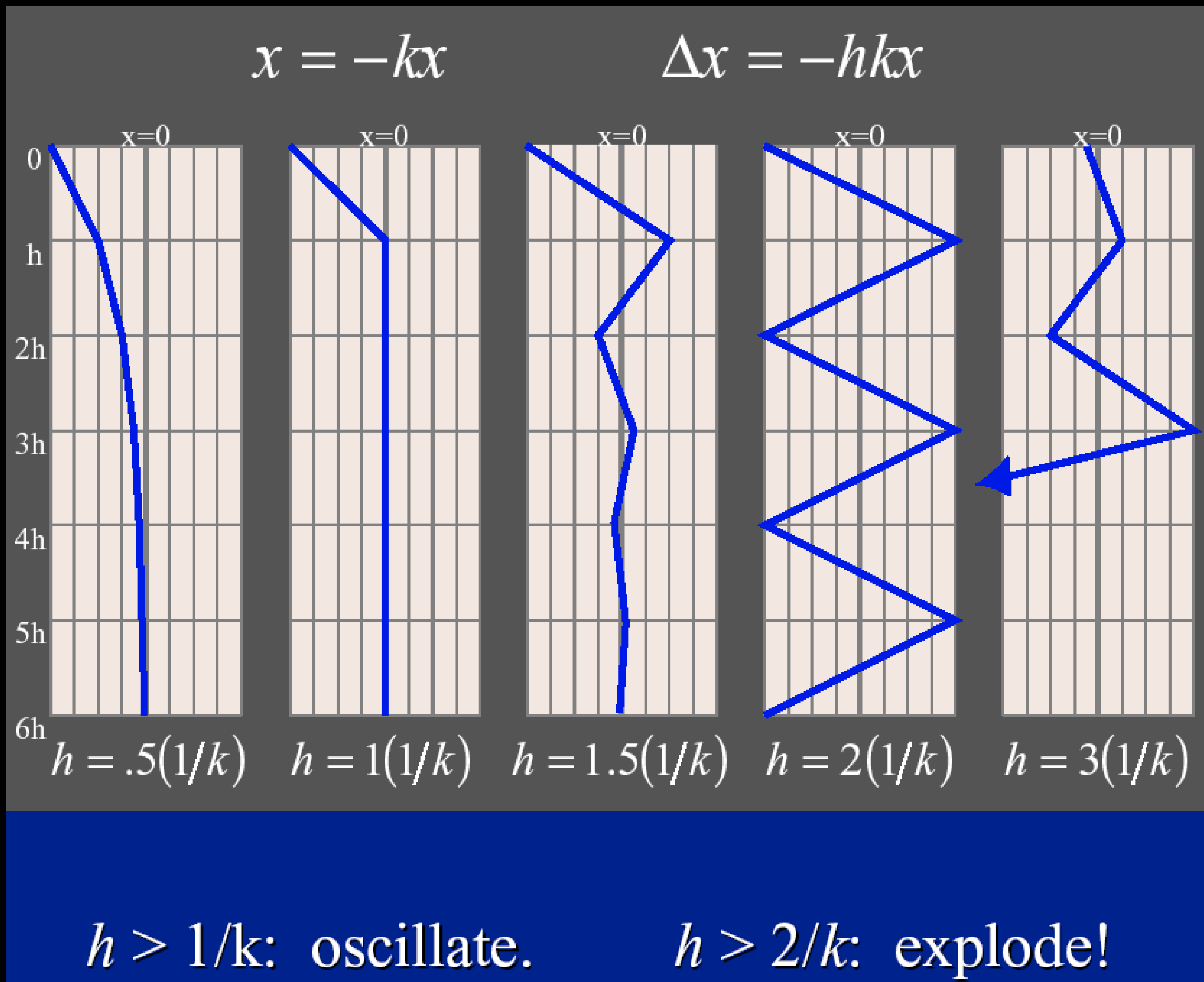


Source: [Provot 1995]

Option 1: increase k_s

- Problem: stiffness (on board)

Stiffness (2)



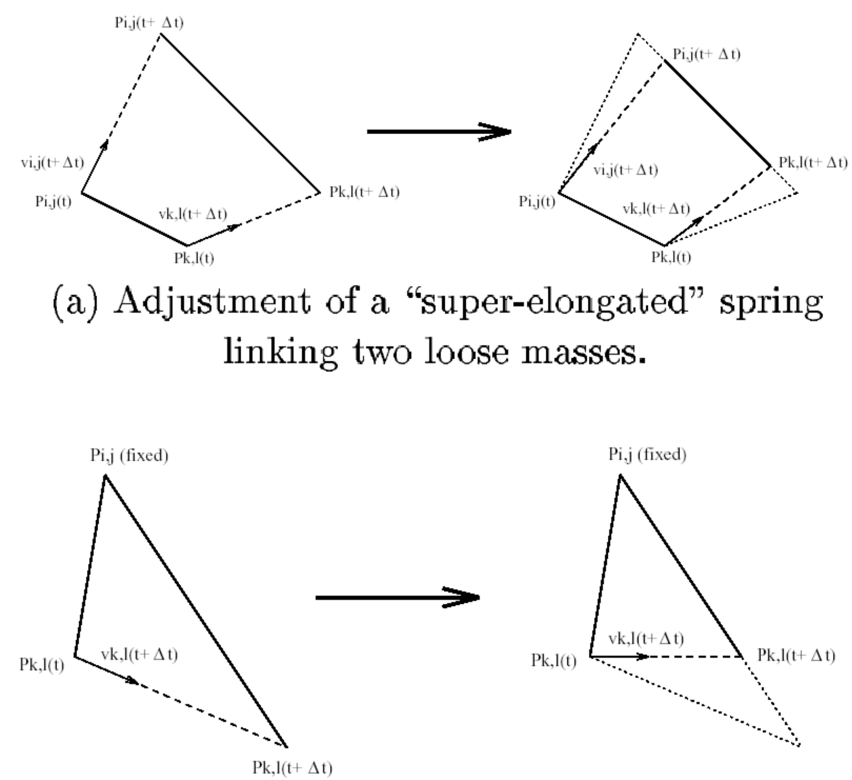
Source: Baraff course notes (2001)

Stiffness (3)

- We'll deal with this specifically on Thursday, but for now…

Other solutions?

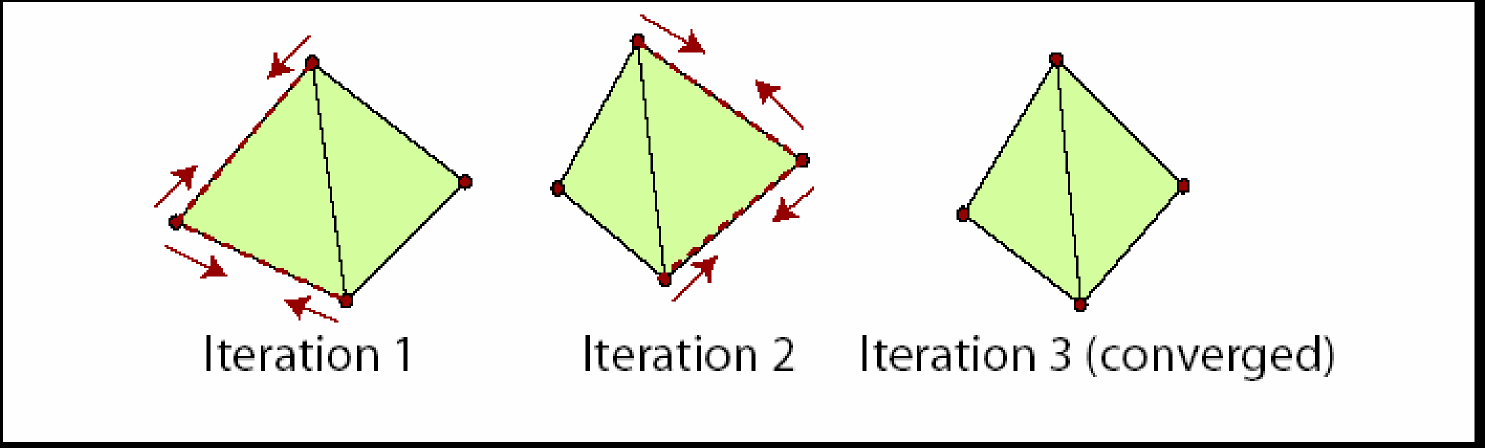
- Simple heuristic due to Provot:
  - After every timestep, adjust spring lengths



(a) Adjustment of a "super-elongated" spring linking two loose masses.

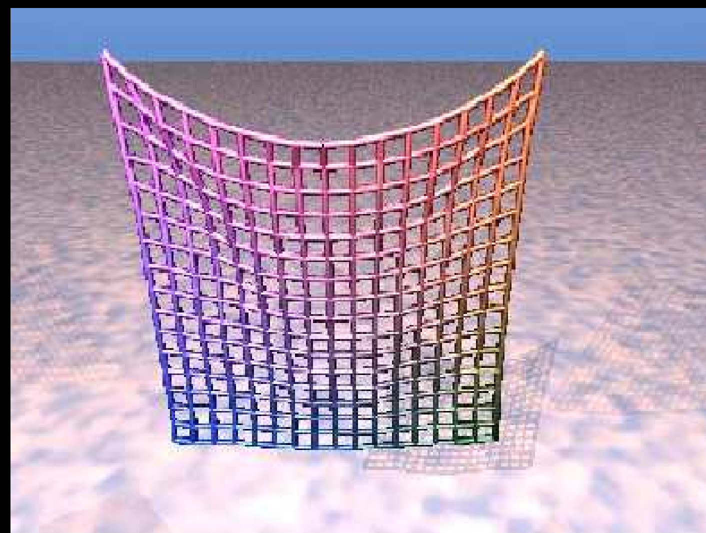
(b) Adjustment of a "super-elongated" spring linking a fixed mass and a loose mass.

Note: must iterate until convergence

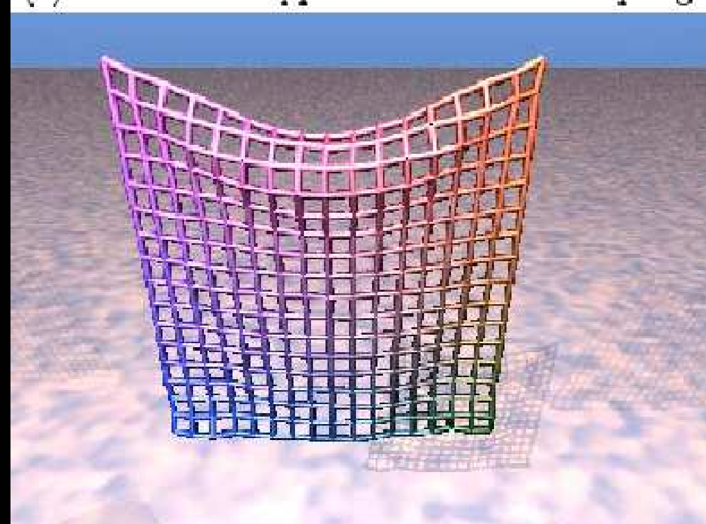


[Provot 1995]

Results:

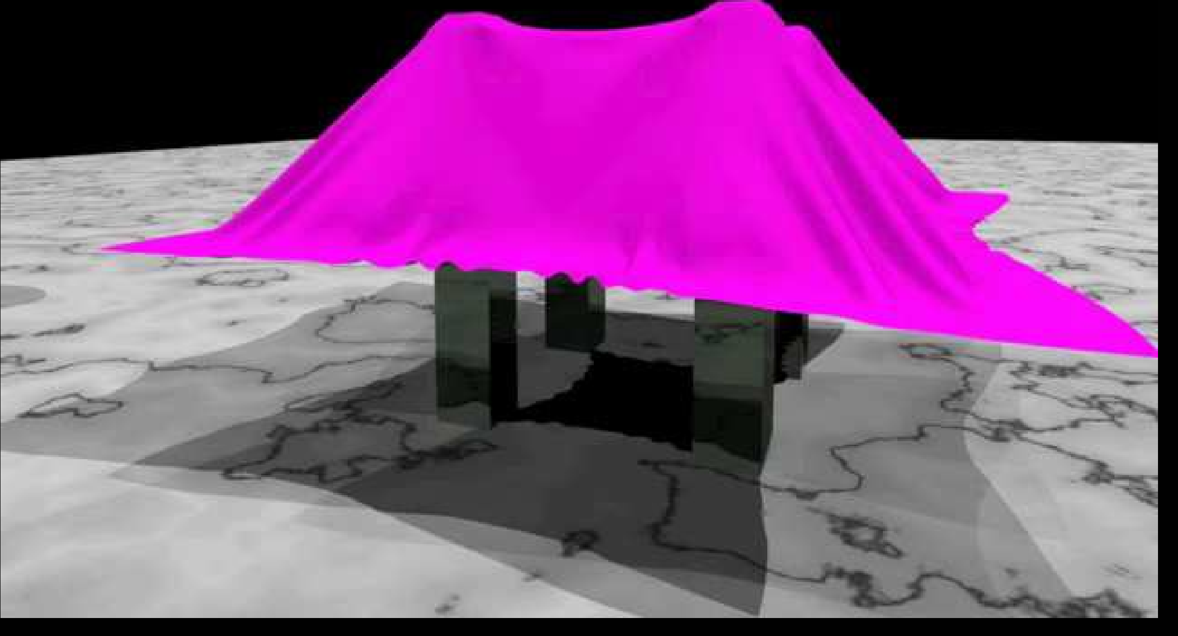


(a) Our method applied to “structural” springs



Heuristic approach…

- Also used by [Bridson et al., 2002] (covered in more depth on Thursday)

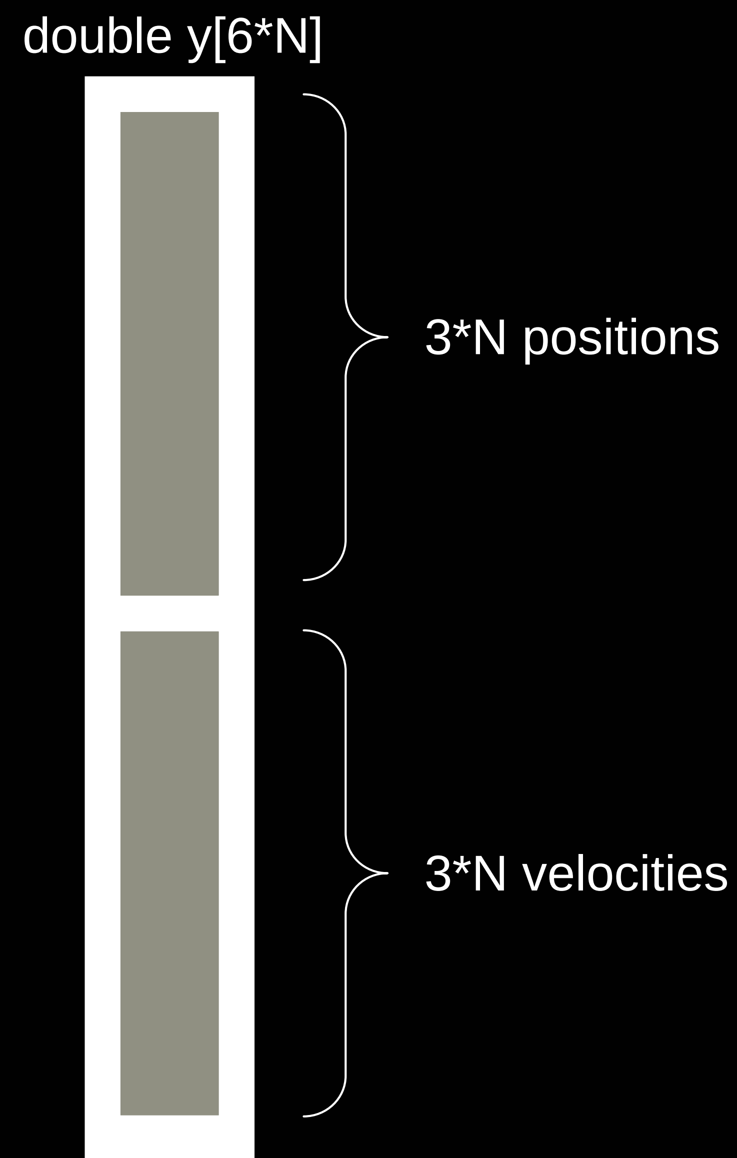


Implementation details

- Use an external solver!
- GSL (Gnu Scientific Library) contains:
  - Embedded 2nd order Runge-Kutta with 3rd order error estimate
  - Embedded 4th order Runge-Kutta-Fehlberg method with 5th order error estimate
  - Embedded 8th order Runge-Kutta Prince-Dormand method with 9th order error estimate (!!)

  …

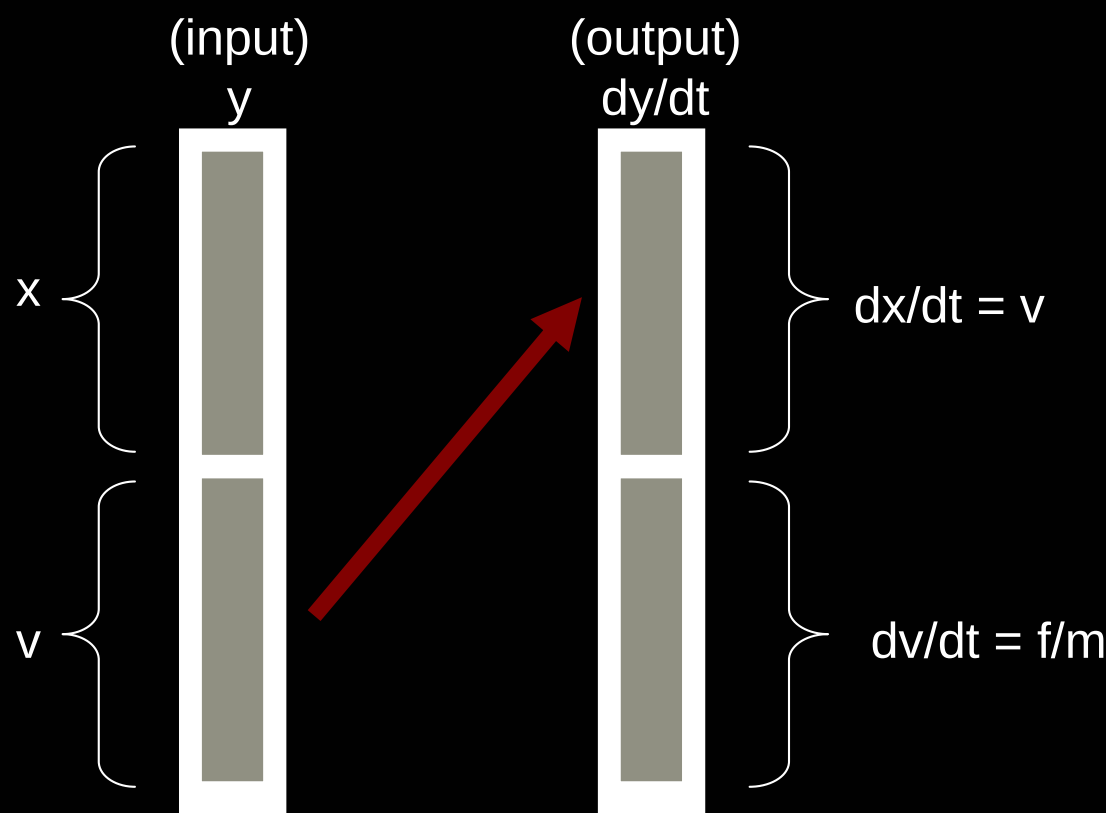
ODE solver (1): storing state



ODE solver (2): the derivs function

```
int derivs (double t, const double y[],
      double dydt[], void * params)
```

Evaluates the state derivatives at (y, t)



ODE solver (3): the derivs function

```
int derivs (double t, const double y[],
        double dydt[], void * params)
```

Pass in anything that is useful; e.g.,
a pointer to the particle system.

ODE solver (4): the derivs function

```
int derivs (double t, const double y[],
      double dydt[], void * params)
{
  ClothModel* model =
      reinterpret_cast< ClothModel *>(params);

  // Copy velocities into dx/dt spot
  std::copy( y + 3*model->N,
      y + 6*model->N, f );

  // Evaluate forces at the current position and timestep
  model->f( t, y, y + 3*model->N, f + 3*model->N );

  // Scale forces by 1/m
  // Note that invMass is 0 for constrained particles
  for( unsigned int i = 0; i < 3*model->N; ++i )
        f[ 3*model->N + i ] *= model->invMass( i/3 );
}
```

ODE setup for GSL

- First, pick a stepping function from this list:
  - **gsl_odeiv_step_rk2**
  - **gsl_odeiv_step_rk4**
  - **gsl_odeiv_step_rkf45**
  - **gsl_odeiv_step_rkck**
  - **gsl_odeiv_step_rk8pd**
- (implicit methods are off-limits for now)

ODE setup for GSL (2)

- Now, do some boilerplate allocation:

```
// At start
gsl_odeiv_step* s_;
gsl_odeiv_control* c_;
gsl_odeiv_evolve* e_;

s_ = gsl_odeiv_step_alloc (stepType, 6*N_);
c_ = gsl_odeiv_control_y_new (error, 0.0);
e_ = gsl_odeiv_evolve_alloc (6*N);

// At exit
gsl_odeiv_evolve_free(e_);
gsl_odeiv_control_free(c_);
gsl_odeiv_step_free(s_);
```

ODE setup for GSL (3)

- Or, if we're clever…

```
class GSLWrapper
{
public:
          GSLWrapper( const gsl_odeiv_step_type* stepType,
                    unsigned int systemSize,
                    double *timestep,
                    Real error)
                    :         timestep_( timestep ),
                              systemSize_(systemSize)
          {
                    s_ = gsl_odeiv_step_alloc (stepType, systemSize_);
                    c_ = gsl_odeiv_control_y_new (error, 0.0);
                    e_ = gsl_odeiv_evolve_alloc (systemSize);
          }

          ~GSLWrapper()
          {
                    gsl_odeiv_evolve_free(e_);
                    gsl_odeiv_control_free(c_);
                    gsl_odeiv_step_free(s_);
          }

          // …
};
```

Now, for the solver loop

```
std::vector<double> state( 6*N );
// copy state from system to state vector

double currentTime = 0.0;
double endTime = 1.0/30.0;  // one animation frame
double timestep = 0.01;     // initial guess
while( currentTime < endTime )
{
        gsl_odeiv_system sys =
                { derivs, 0, 6*N, (*this) };
        int status = gsl_odeiv_evolve_apply(
                        e_, c_, s_,
                        &sys,
                        &currentTime, endTime,
                        &timestep, &y[0]);
        if( status != GSL_SUCCESS )
                // do something!

        // should correct particle positions here.
}

// now copy state back into the particle system
```

And that's it!

- You should have (almost) everything you need to implement your cloth solver
- Collisions on Thursday