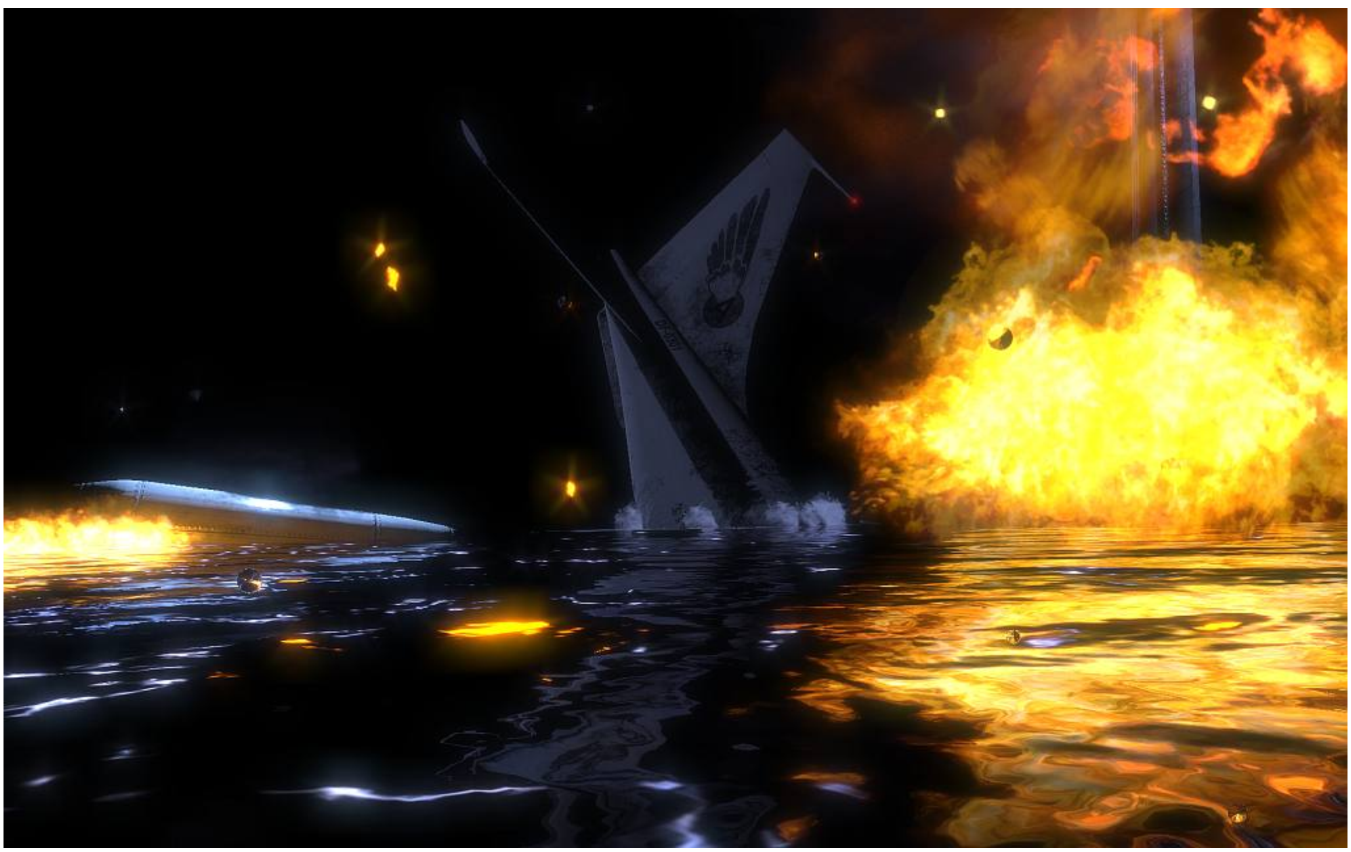
Animation:
Particle Dynamics

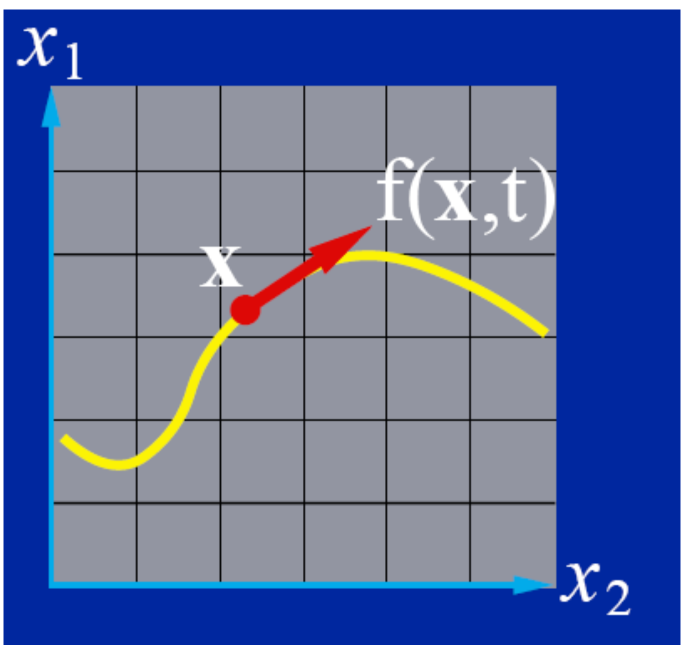
Overview

- One lousy particle (Assignment 1)
- Particle systems
  - this is how we make rain, cloth, fire, smoke, hair, grass…
- Forces: gravity, springs,…
- Implementation and interaction
- Simple collisions



A Newtonian Particle

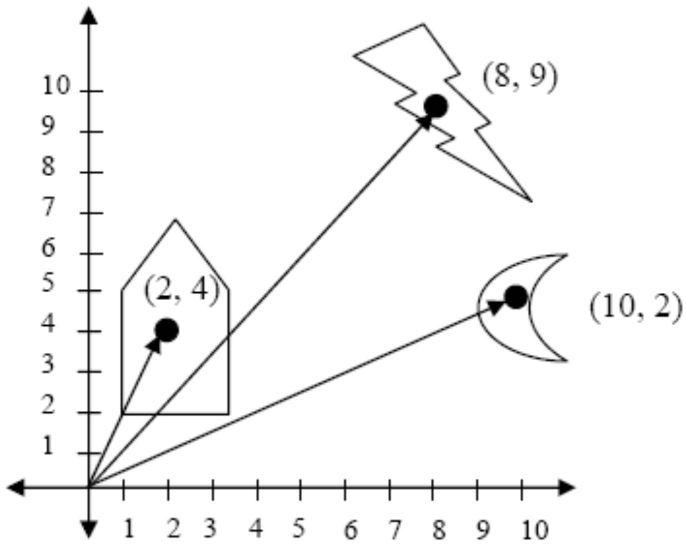
- Particle structure
  - *x position (in 3D, a 3D variable)*
  - *v velocity (a 3D vector)*
  - *a acceleration (a 3D vector) or rather:*
    - *f force accumulator (a 3D vector)*
    - *m mass (a scalar)*




```
typedef struct{
float m; /* mass */
float *x; /* position vector */
float *v; /* velocity vector */
float *f; /* force accumulator */
} *Particle;
```

Vectors & Vector Space

- Consider all locations in relationship to one central reference point, called origin



- A vector tells us which direction to go with respect to the origin, and also the length of the trip
  - A vector does **not** specify where the trip begins, to continue this analogy

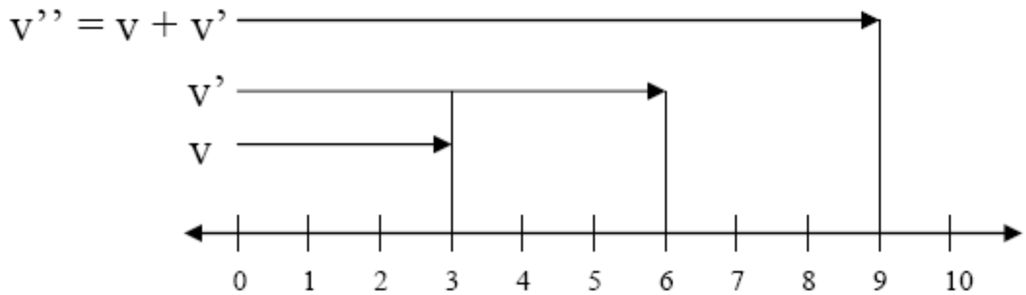
  Notation: **v** = [velocity.x   velocity.y   velocity.z]

  x = [pos.x   pos.y   pos.z]

Vector Addition

Vector addition in $\Re^1$

- Familiar addition of real numbers

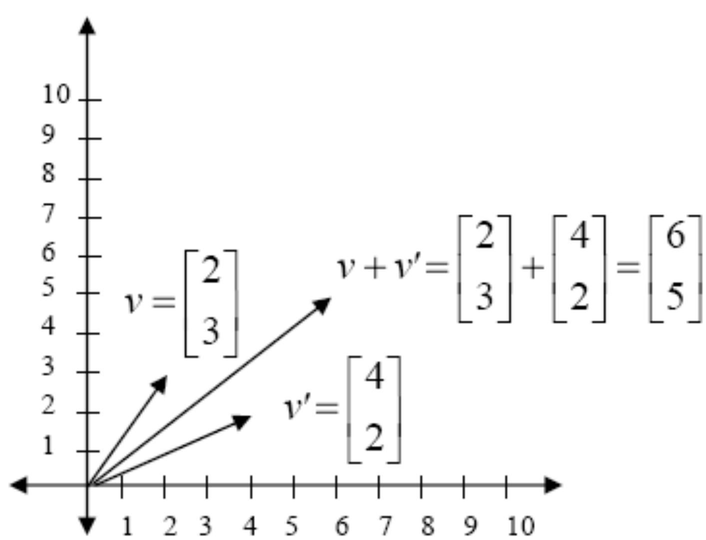


v = [3], v′ = [6], v″ = [9]

Vector addition in $\Re^2$

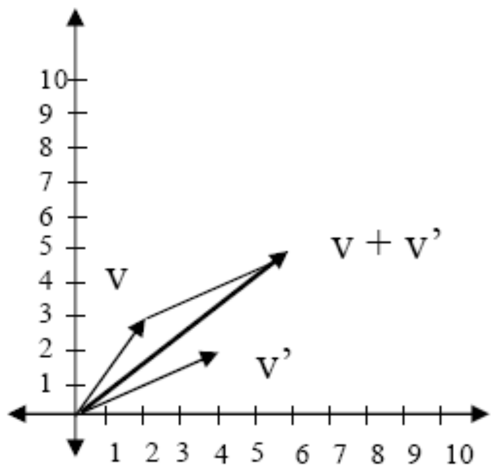
- The x and y parts of vectors can be added using addition of real numbers along each of the axes (component-wise addition)

Result, v + v′, p

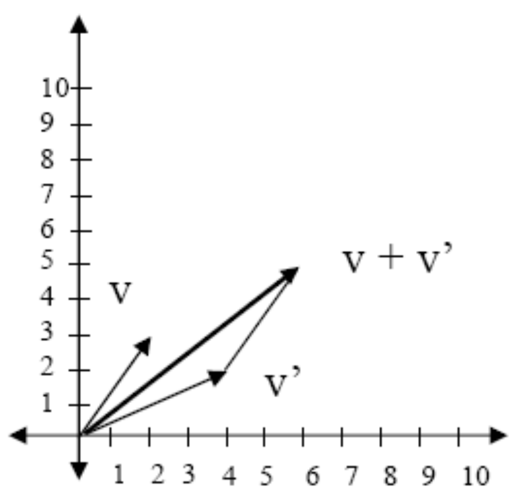


Adding Vectors Visually

- v' added to v, using the parallelogram rule: take vector from the origin to v'; reposition it so that its tail is at the head of vector v; define v+v' as the head of the new vector



- or, equivalently, add v' to v

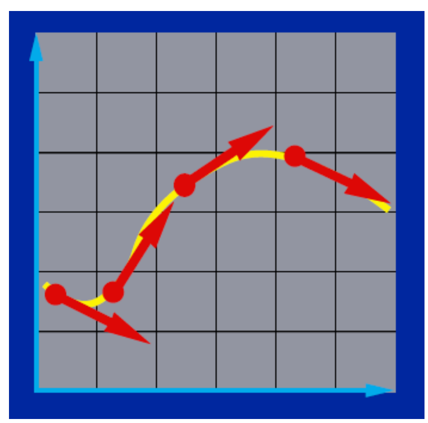


Particle Motion

- Particle motion governed by a differential equation:
  $a = f/m$
  where f depends on: position, velocity, time
  $f(x, v, t)$

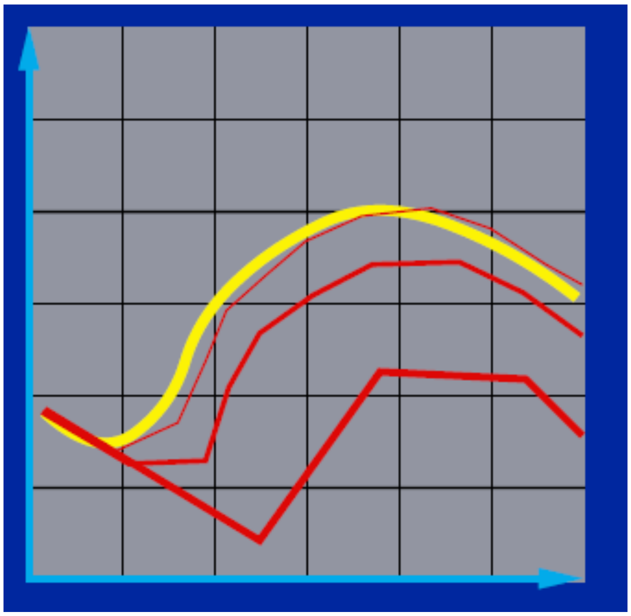
- Why "differential"?
  $a = x''$, $v = x'$ →

  $x'' = f(x, x', t)/m$
  - a vanilla second-order differential equation which we need to solve for x

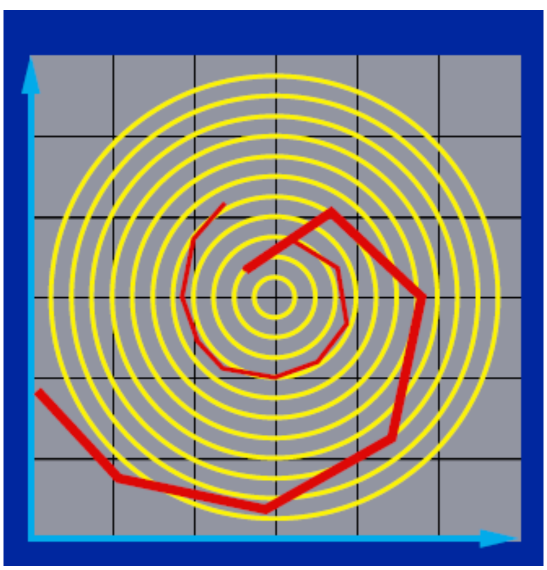


  - for now we'll solve via Euler integration:
    $x(t + dt) = x(t) + dt*f(x,t)$

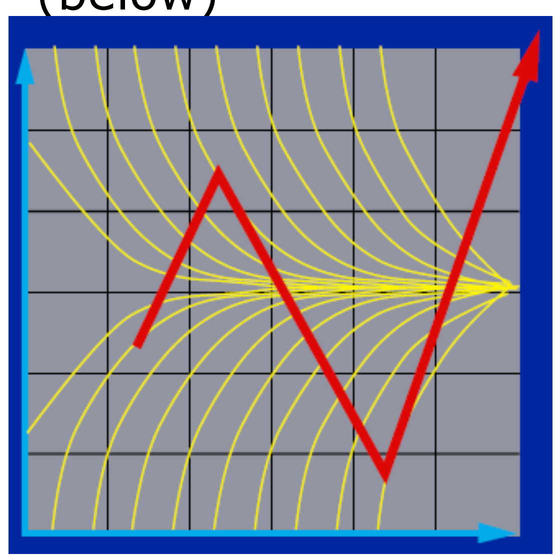
Euler’s Method



- simplest numerical solution method
- discrete time steps
- bigger steps, bigger errors



- problems: inaccuracy (left) and instability (below)



The Equations of Motion

- Update the particle for each time step

  a(t+dt) = f(t)/m

  v(t+dt) = v(t) + a(t)*dt

  x(t+dt) = x(t) + v(t)*dt + a(t) * dt^2/2

- Implementation

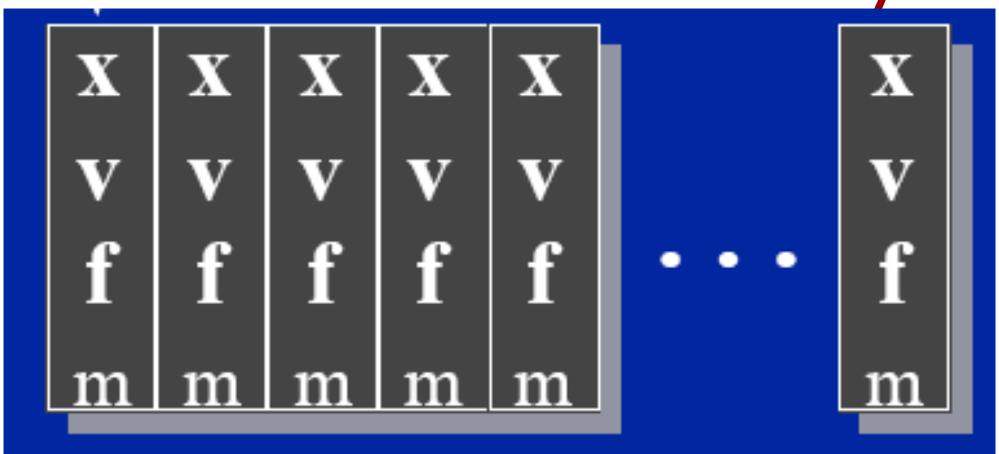
  void timeout() {
    - compute forces
    - update a
    - update v
    - update x

    draw

  }

- Better: smaller timestep than timeout, cycle several times and draw once

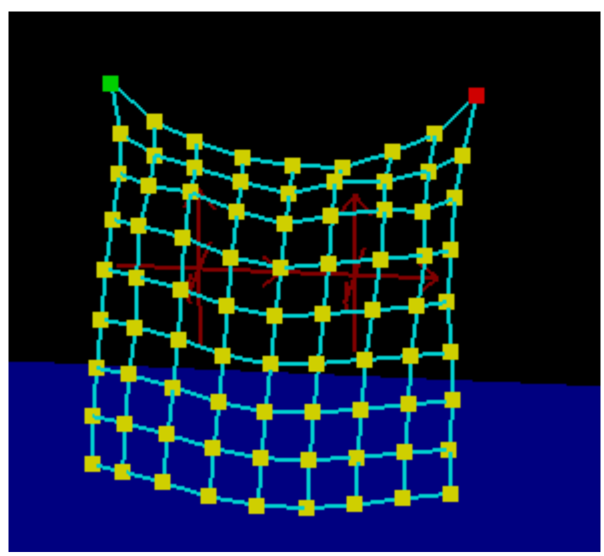
EC: Particle Systems




```
typedef struct{
Particle *p; /* array of pointers
to particles */
int n; /* number of particles */
float t; /* simulation clock */
} *ParticleSystem;
```

- Implementation

```
void timeout() {
    for each particle
    • compute forces
    • update a
    • update v
    • update x
    draw
}
```



Forces

- constant: gravity

Force Law:

$$\mathbf{f}_{grav} = m\mathbf{G}$$

- force fields: wind, pressure,…
- velocity-dependent: viscous drag,…

Force Law:

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

- n-ary: damped springs (r: the rest length)

Force Law:

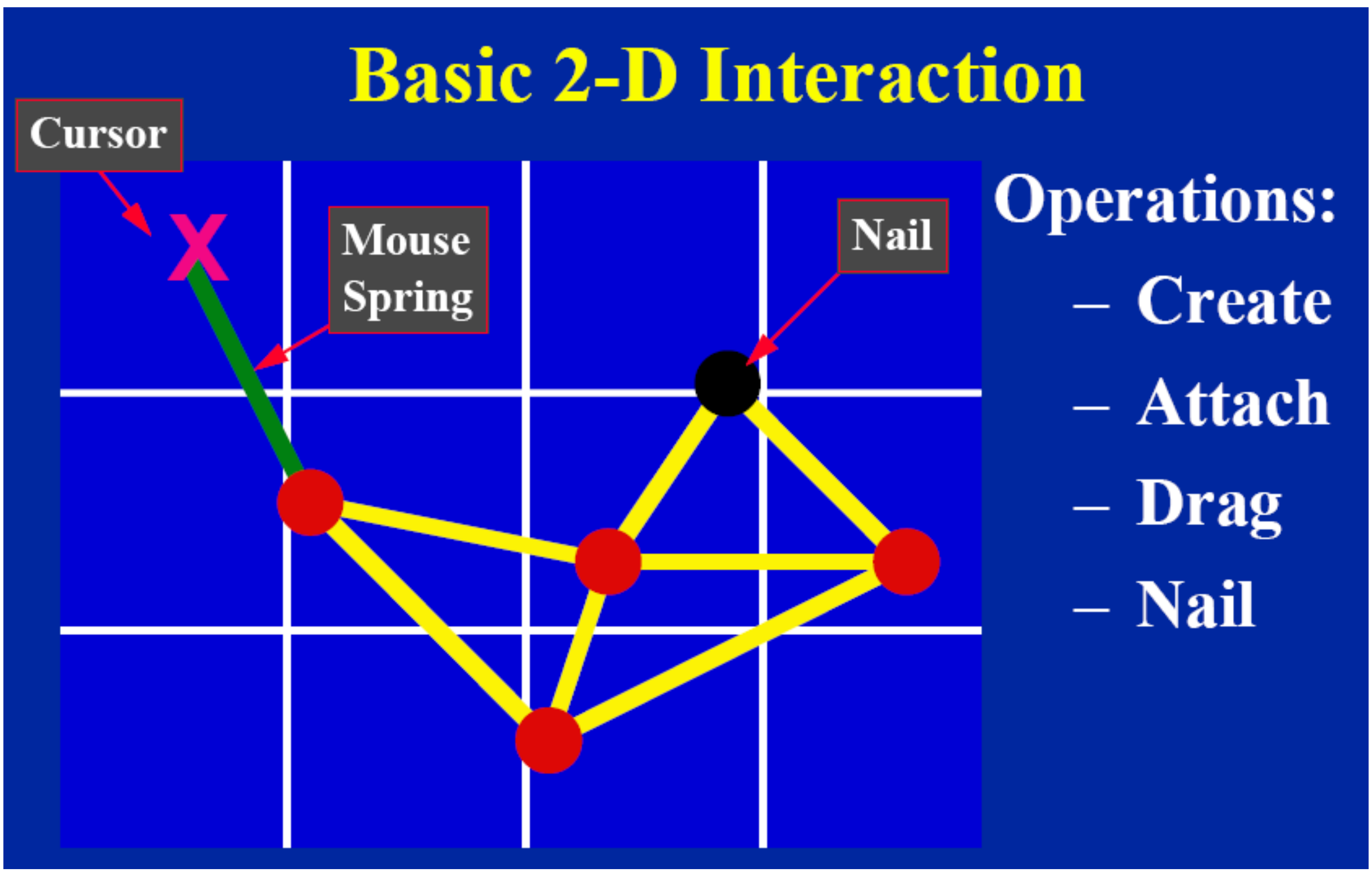
$$\mathbf{f}_1 = -\left[k_s(|\Delta\mathbf{x}| - r) + k_d\left(\frac{\Delta\mathbf{v}\cdot\Delta\mathbf{x}}{|\Delta\mathbf{x}|}\right)\right]\frac{\Delta\mathbf{x}}{|\Delta\mathbf{x}|}$$

$$\mathbf{f}_2 = -\mathbf{f}_1$$

- collisions…

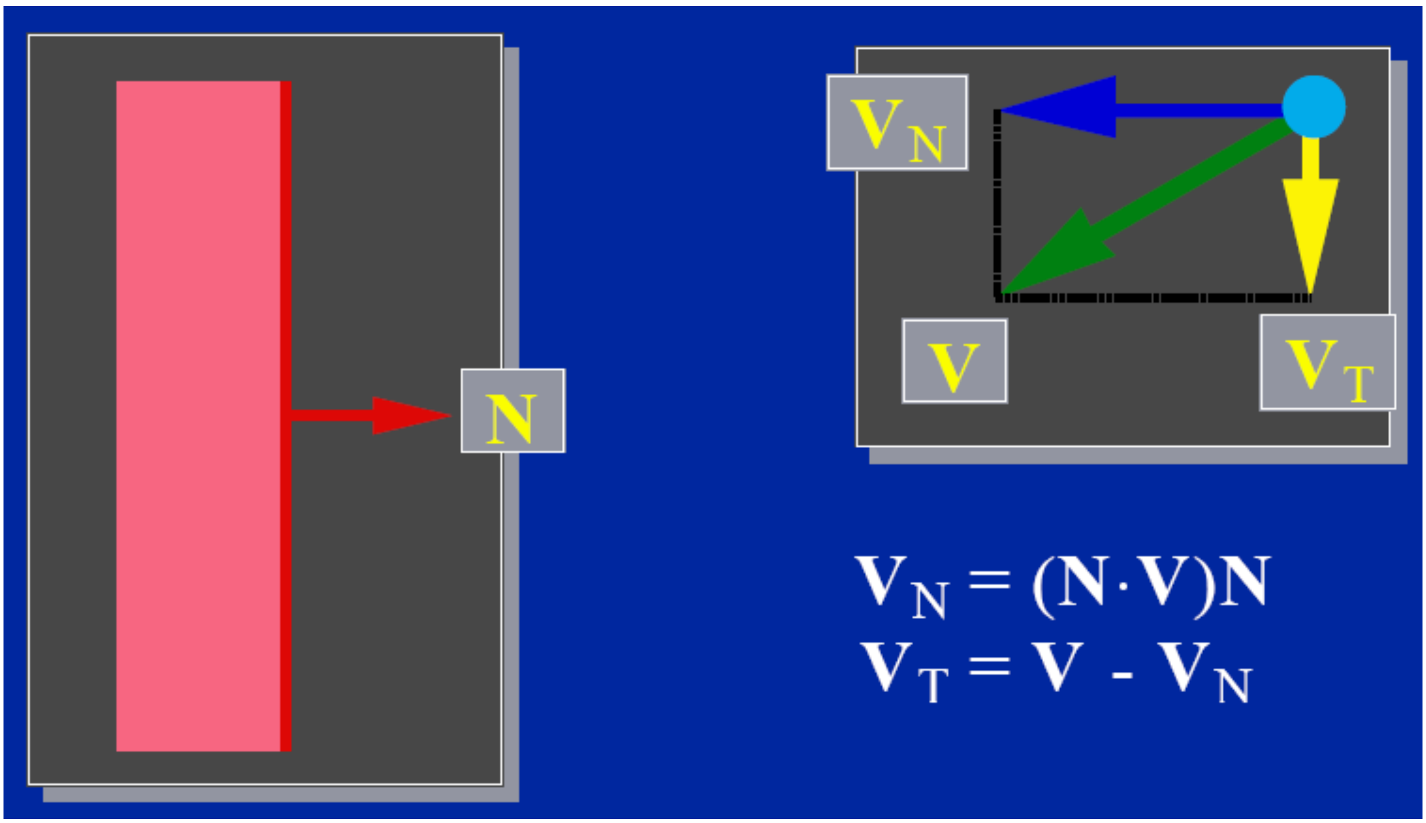
Try This at Home

- the notes give you everything you need to build a basic interactive mass-spring simulator – give it a try!

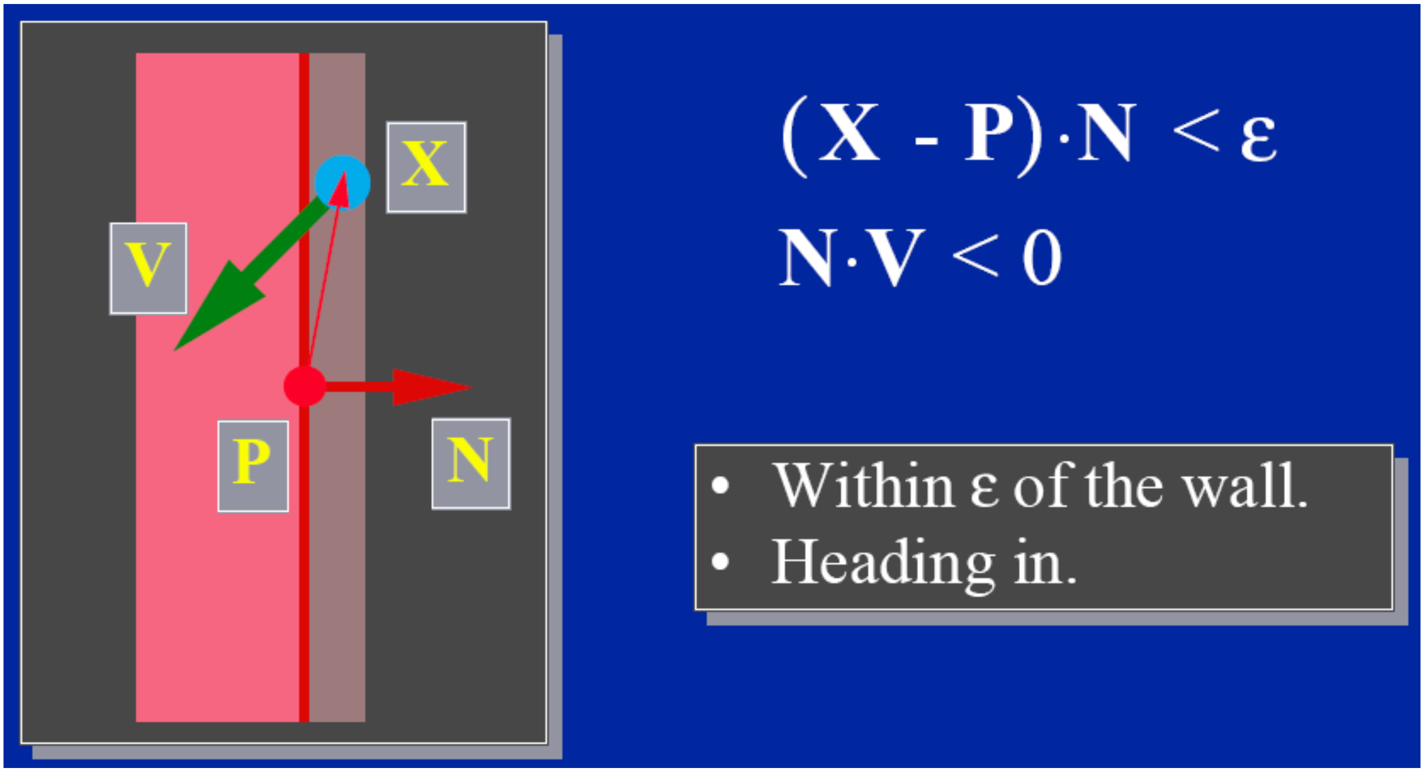


Collisions: Bouncing off the Walls

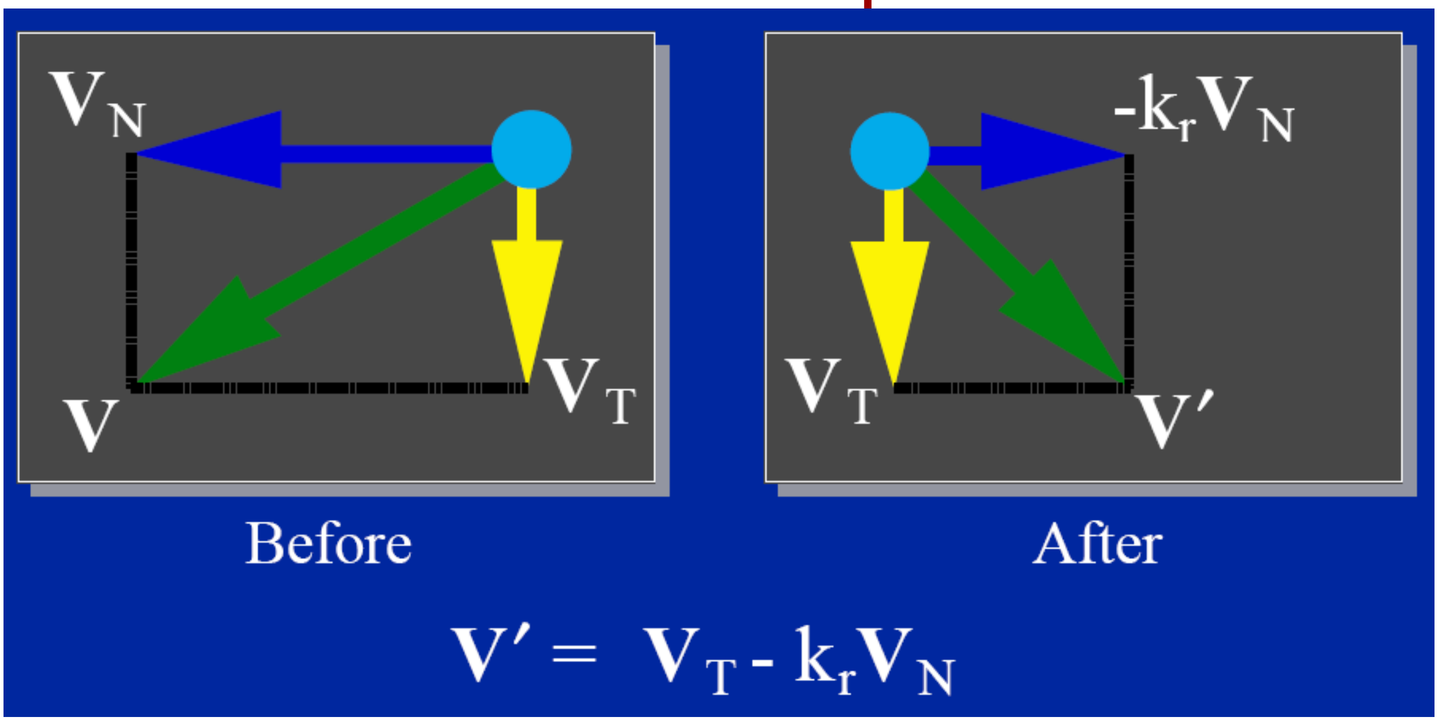
- normal and tangential components
- “normal” is the direction perpendicular to the wall
- “tangential” is the direction along the wall

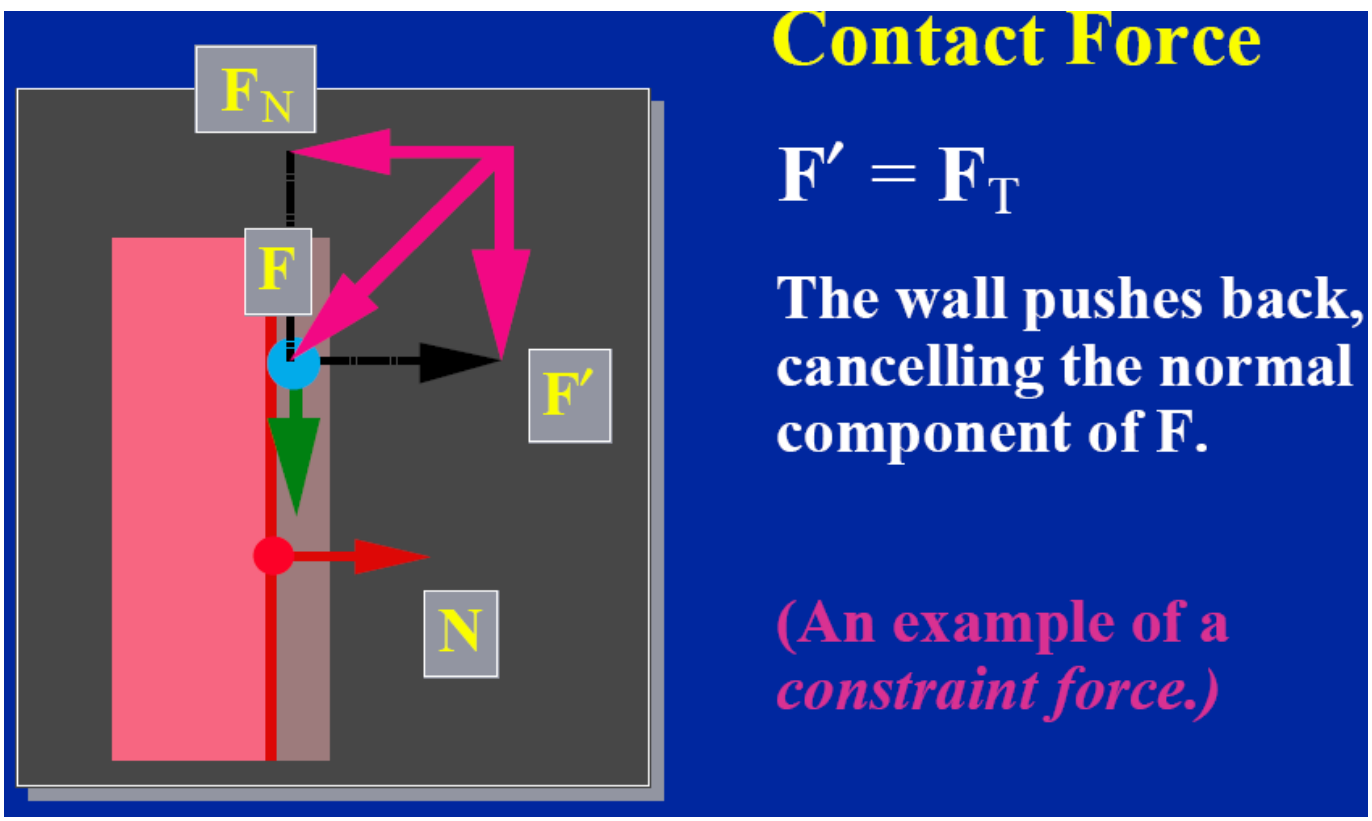


Collision Detection



Collision Response





Give Me Stability or Give Me Death
(D. Baraff motto)

- Do NOT use Euler outside cs1566 (you will use it anyway ☺)
  - if your step size is too big, your simulation blows up. It isn't pretty.
  - stability is all, stability is all, stability is all…
- You can make just about anything out of point masses and springs, *in principle*
- *In practice,* you can make anything you want as long as it's jello
- Rigid links and constraints give us ways to make more interesting contraptions
- Resources: A. Witkin, D.Baraff, "Physically-based Modeling"
  http://www.cs.cmu.edu/~baraff/sigcourse/

Summary

- One lousy particle
- Particle systems (EC)
  - this is how we make rain, cloth, fire, smoke, hair, grass…
- Forces: gravity, (EC) springs,…
- Implementation and interaction
- Simple collisions (EC)