Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

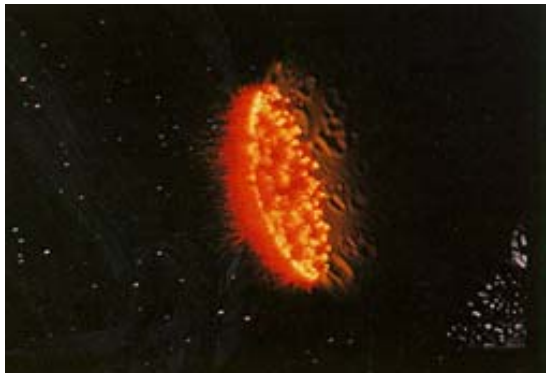
02: PARTICLE SYSTEMS

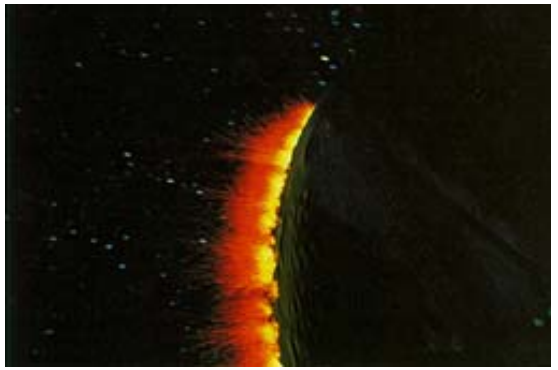
21/01/2016

PARTICLE SYSTEMS

Particle simulation, though fairly simple to implement can create some very interesting visual effects.

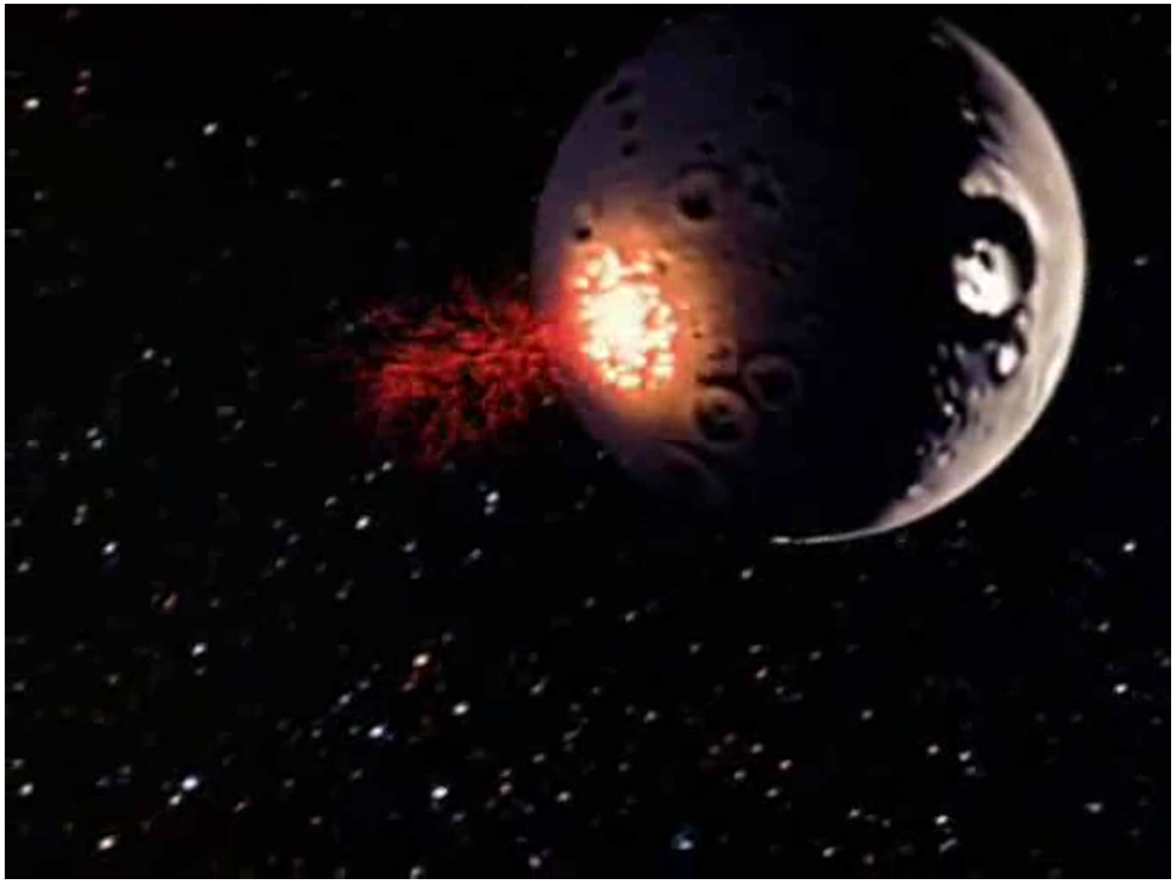
The Genesis Effect - Star Trek II: The Wrath of Khan (1983)





http://www.siggraph.org/education/materials/HyperGraph/animation/movies/genesisp.mpg

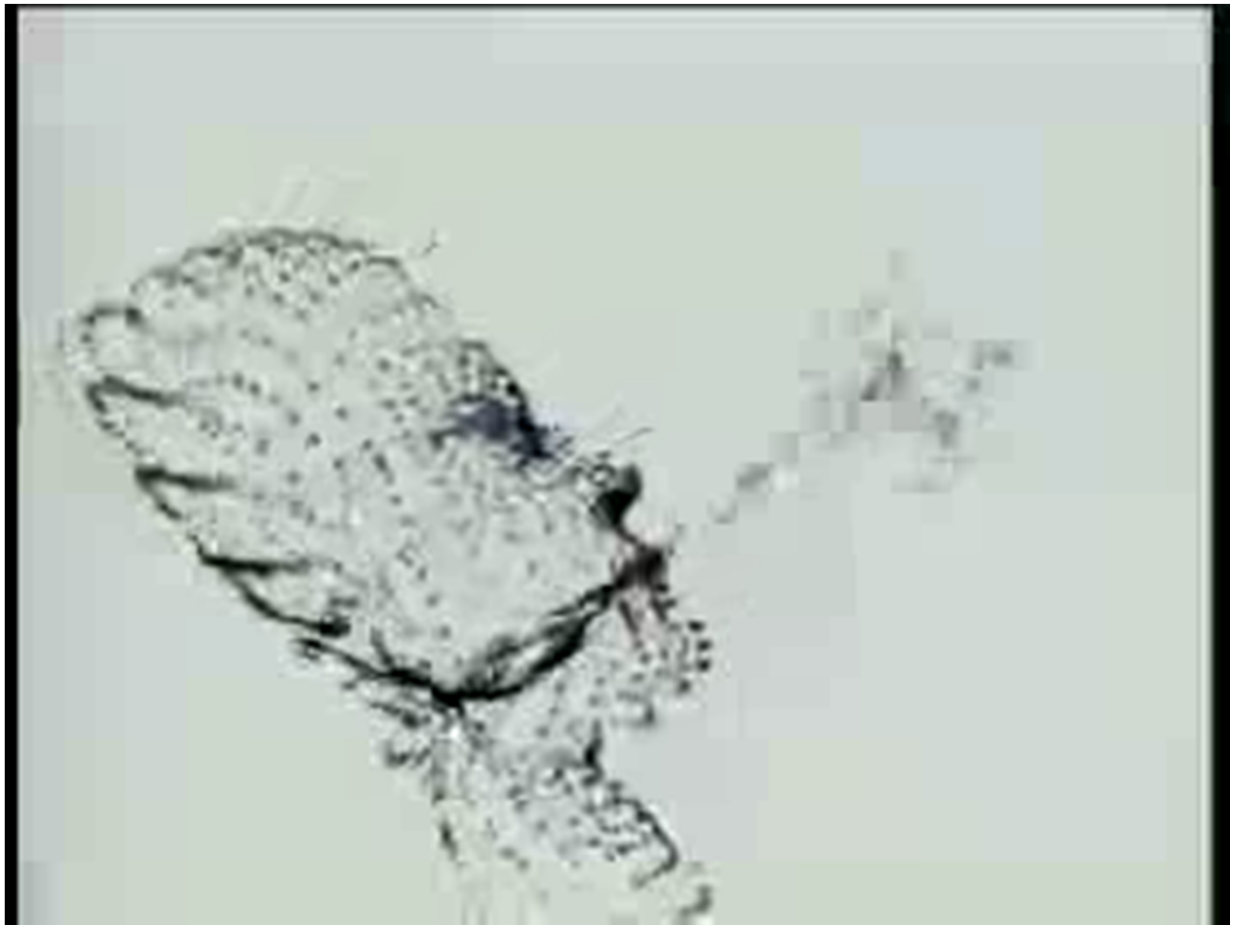
PARTICLE SYSTEMS



The Genesis Effect - Star Trek II: The Wrath of Khan (1983)

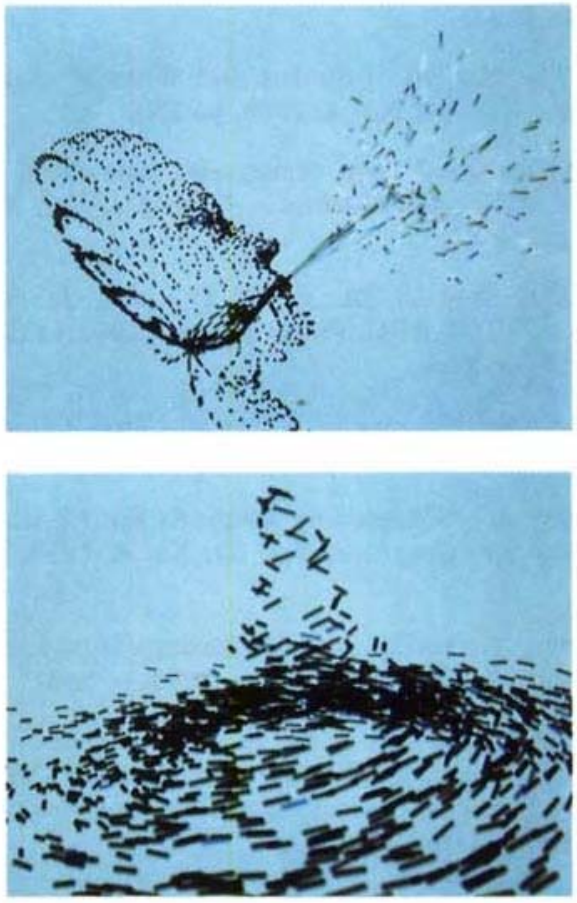
http://www.siggraph.org/education/materials/HyperGraph/animation/movies/genesisp.mpg

PARTICLE SYSTEMS



Karl Sims: Particle Dreams (Siggraph 1988) Karl Sims: http://www.karlsims.com/papers/ParticlesSiggraph90.pdf

PARTICLE SYSTEMS



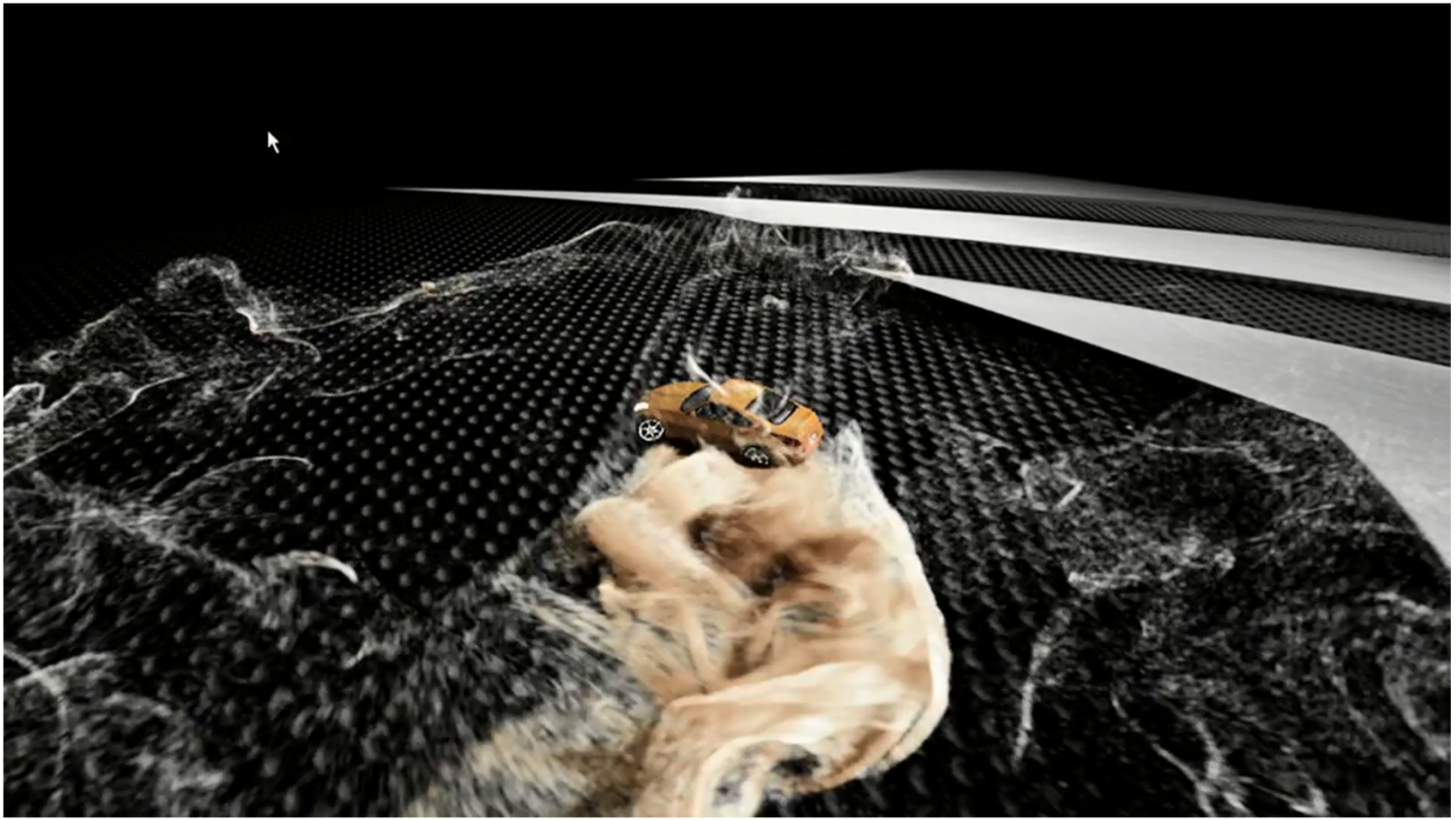
Karl Sims - Particle Dreams (Siggraph 1988) Karl Sims:

http://www.karlsims.com/papers/ParticlesSiggraph90.pdf

PARTICLE SYSTEMS

The emergent behaviour of particles systems can also be used to simulate some very complex high level phenomena such as cloth, hair and fluids.

MORE COMPLEX PARTICLES



NEWTONIAN PARTICLE

Particles don't have shape, orientation, structure, but motions are still dependent on Mass and Forces.

Motion represented by Newton's Second Law

$$\boldsymbol{f} = m\boldsymbol{a}$$

- which can be expressed as $\ddot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t)/m$

For implementation, introduce an extra variable v to create a pair of coupled first order equations

$$\dot{\boldsymbol{x}} = \boldsymbol{v}$$

$$\dot{\boldsymbol{v}} = \boldsymbol{f}/m$$

PHASE SPACE

Position and velocity can be grouped together into a 6-d vector called the phase of the particle

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{v} \end{bmatrix} \qquad \{x_x, x_y, x_z, v_x, v_y, v_z\}$$

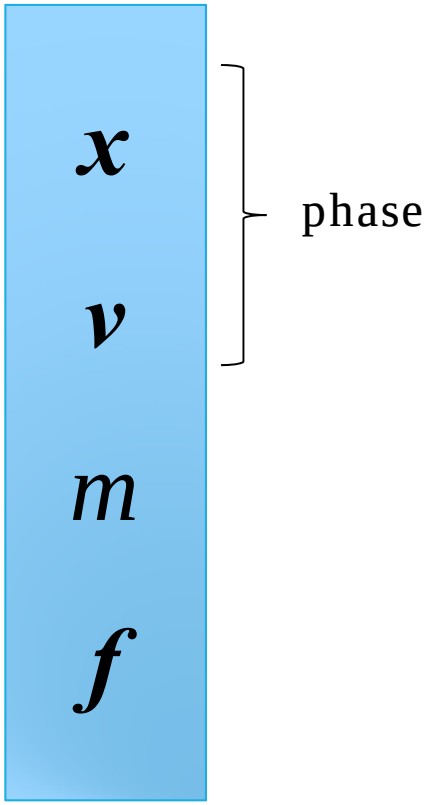
The phase space equation of motion:

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{v}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{f}/m \end{bmatrix}$$

Integrate this to simulate the movement of the particle

$$\{\dot{x}_x, \dot{x}_y, \dot{x}_z, \dot{v}_x, \dot{v}_y, \dot{v}_z\} = \{v_x, v_y, v_z, f_x/m, f_y/m, f_z/m\}$$

PARTICLE STATE REPRESENTATION




```
struct Particle
{
 Vector3  position;
 Vector3  velocity;
 float mass;

 Vector3 force; //force accumulator
};
```

EXAMPLE

For a force take Gravity

$$\boldsymbol{f_{grav} = -mG}$$

Thus equations of motion are

$$\dot{\boldsymbol{x}} = \boldsymbol{v}$$

$$\dot{\boldsymbol{v}} = \frac{\boldsymbol{f_{grav}}}{m} = -\boldsymbol{G}$$

Euler Integration:

$$\Delta \boldsymbol{x} = \boldsymbol{v}\Delta t$$

$$\Delta \boldsymbol{v} = -\boldsymbol{G}\Delta t$$

$$\boldsymbol{x_i = x_{i-1} + v_i \Delta t}$$

$$\boldsymbol{v_i = v_{i-1} - G\Delta t}$$

Pseudocode:

```
x += v*timestep
v += G*timestep
```

ADDING FORCES

Forces on a particle can depend on

- Constant: gravity
- Position/time: force fields
- Velocity: drag, friction
- Links with other particles (n-ary): spring forces

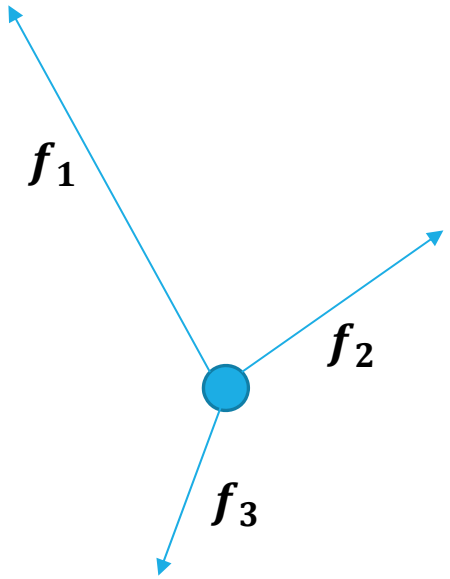
N.B. multiple forces may act on a single particle at any one time

But thankfully this is easy to deal with for particles...

NET FORCE

If multiple forces act on a particle (or through centre of mass of a rigid body), it behaves as if acted on by a single net force

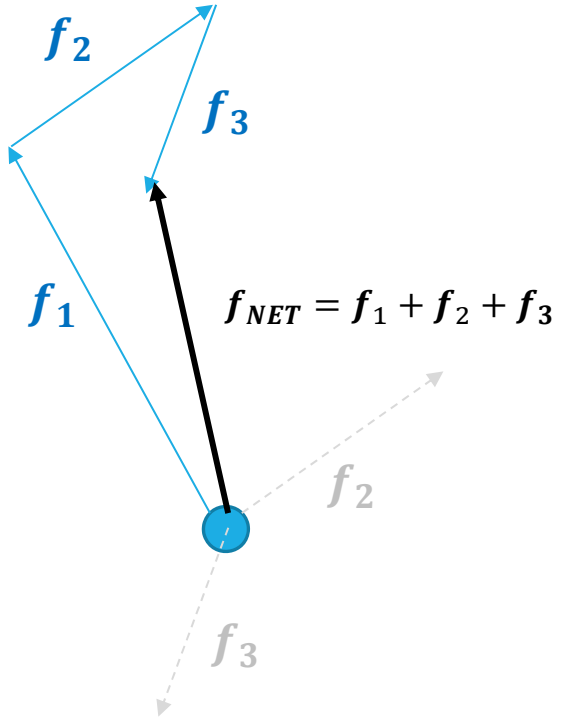
Net force = vector sum of all forces through the centre of mass



NET FORCE

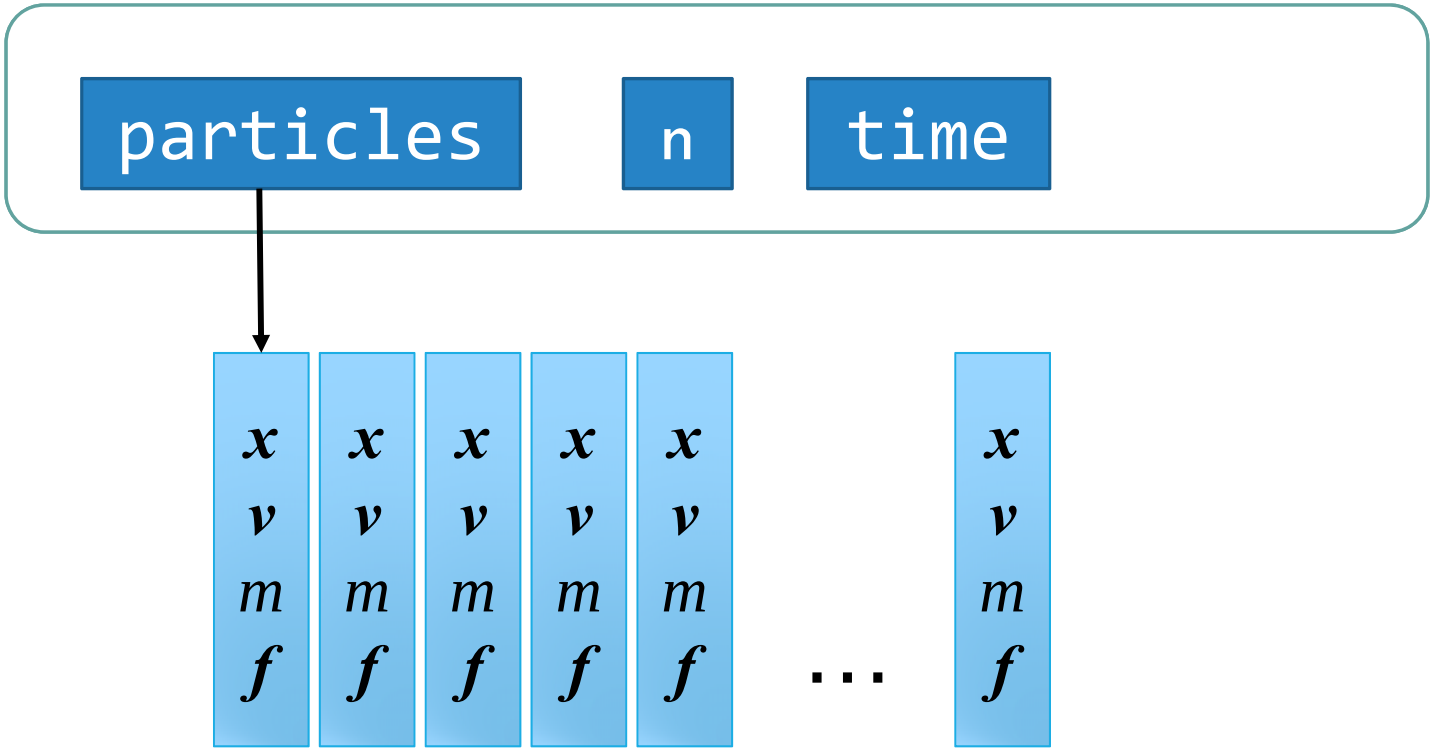
If multiple forces act on a particle (or through centre of mass of a rigid body), it behaves as if acted on by a single net force

Net force = vector sum of all forces through the centre of mass



PARTICLE SYSTEM

List of n Particles and their states at the current timestep



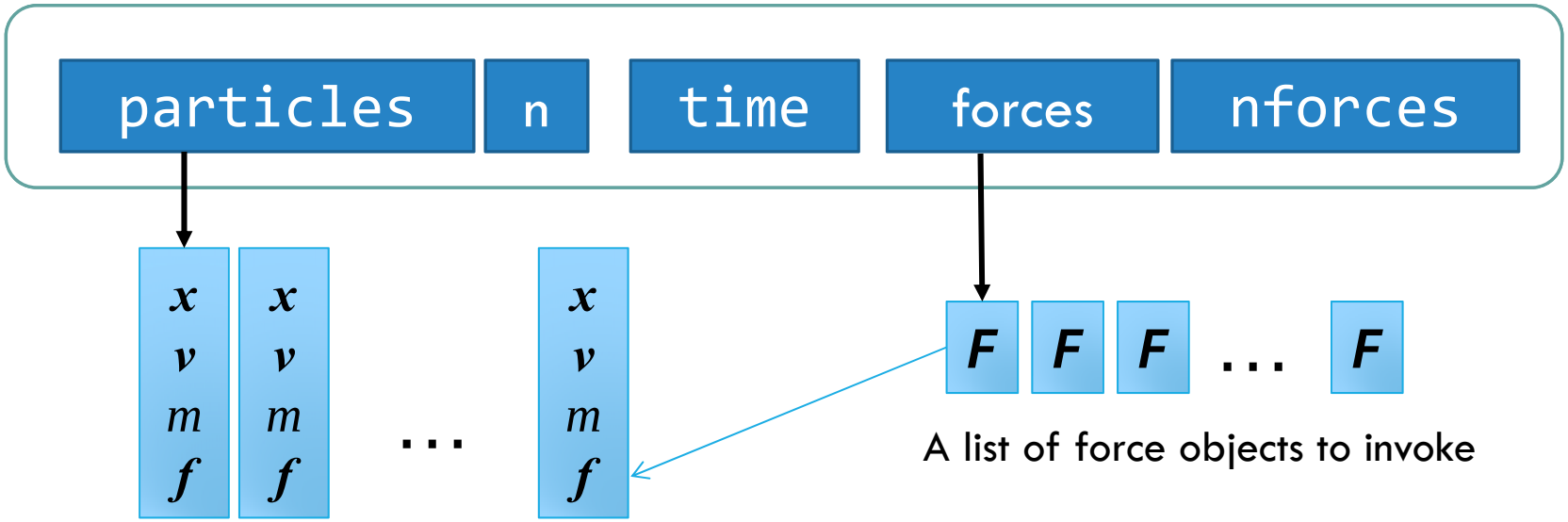
FORCE OBJECTS

List of force objects

- Each points to particles they influence

Global force calculation

- Loop through force list invoking force objects
- Add in their own influences to particles force accumulator



PARTICLE SYSTEM SIMULATION

Particle Update

1. Clear Force from particle $\boldsymbol{f} = 0$
2. Accumulate Forces

   Add forces to particle (loop through force list and invoke each)

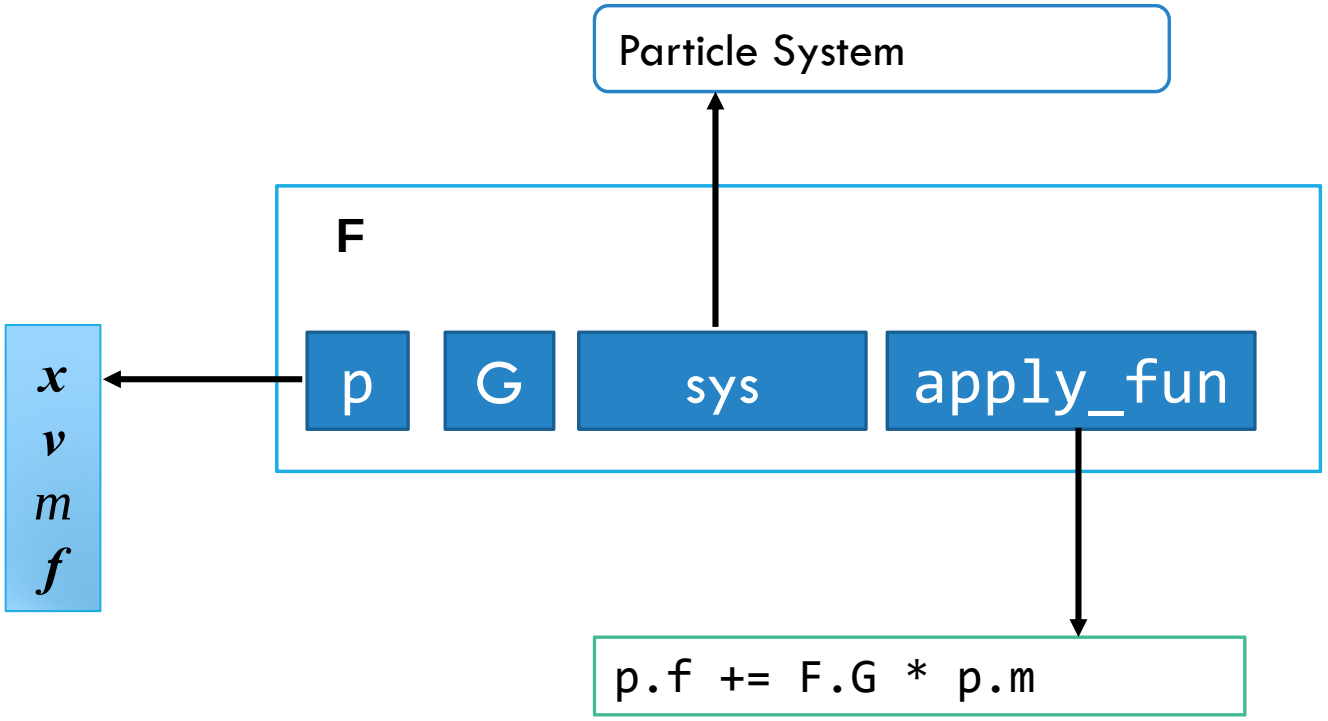
   `pcl[i].f += f[j]`
3. Apply Forces

   Solve for the phase of the particle $\{\boldsymbol{x}, \boldsymbol{v}\}$:

   Integrate $\dot{\boldsymbol{v}} = \frac{\boldsymbol{f}}{m}$, $\dot{\boldsymbol{x}} = \boldsymbol{v}$
4. Do collision handling

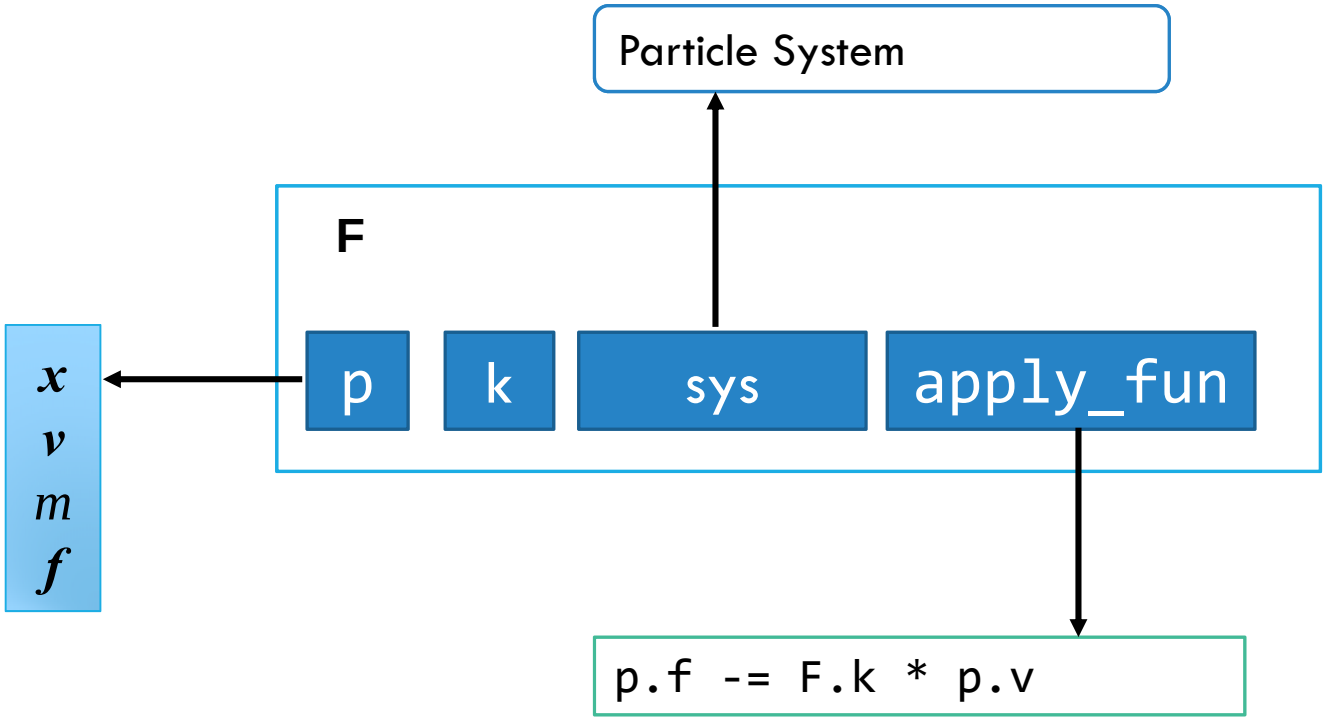
FORCE EXAMPLE: GRAVITY

Force Law: $\boldsymbol{f}_{grav} = m\boldsymbol{G}$



FORCE EXAMPLE: VISCOUS DRAG

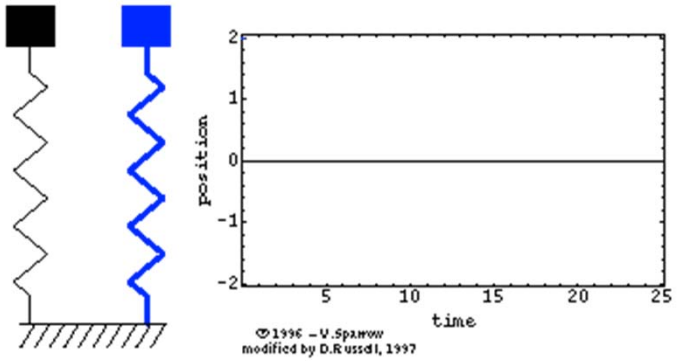
Force Law: $\boldsymbol{f}_{drag} = -k_{drag}\boldsymbol{v}$



FORCE EXAMPLE: SPRING

Basic Spring (Hooke's Law):

- Restoring force
- $\boldsymbol{f}_{spring} = -k_{spring}(\Delta \boldsymbol{x} - \boldsymbol{r})$



Damped Spring

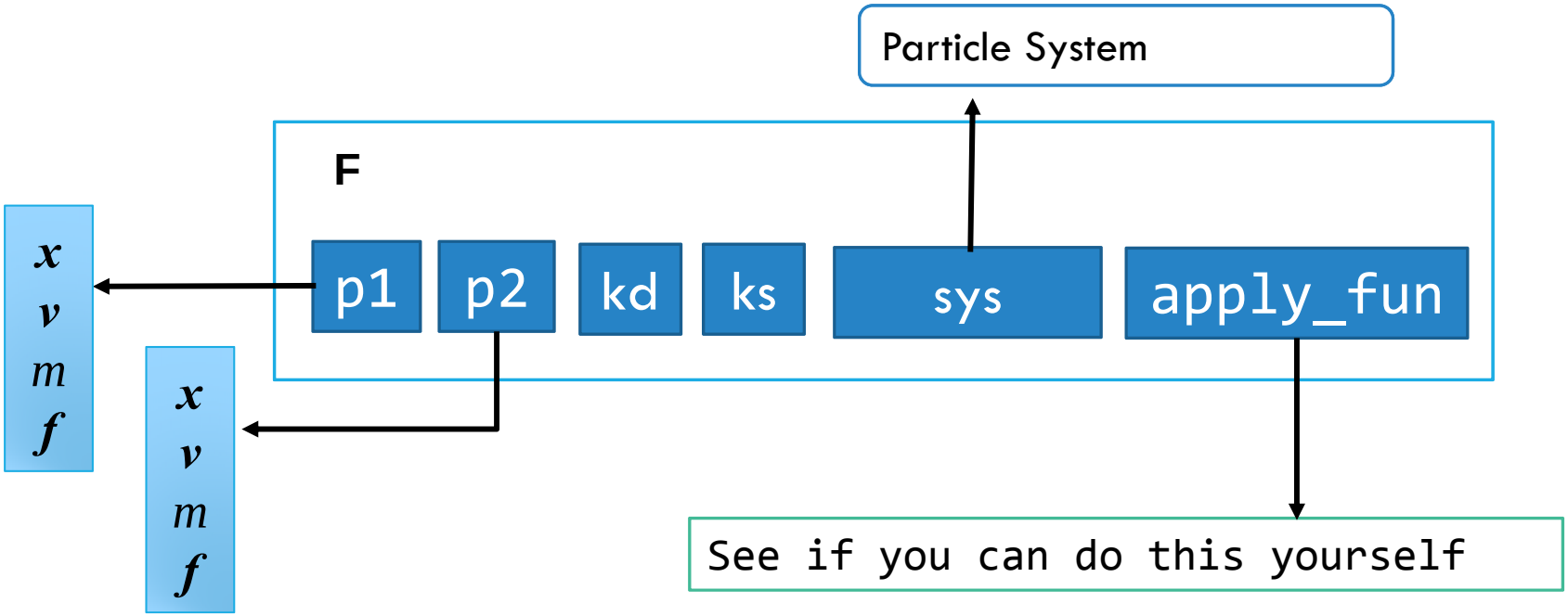
- Damping force (for now, lets just use viscous drag - effectively damps oscillation and other motions)
- $\boldsymbol{f}_{drag} = -k_{drag}\boldsymbol{v}$

FORCE EXAMPLE: DAMPED SPRING

Force Law:

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

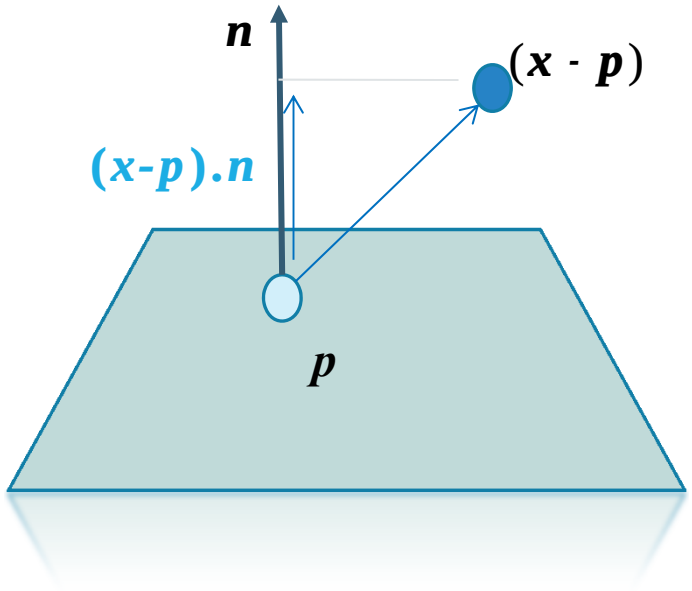
$$\boldsymbol{f}_2 = -\boldsymbol{f}_1$$



Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

COLLISION HANDLING

PARTICLE POSITION



Given normal n and any point p on plane

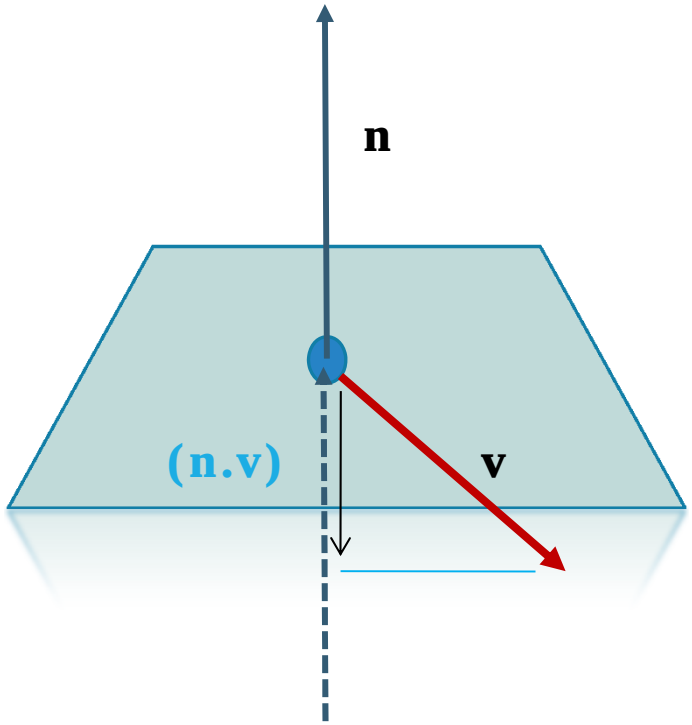
Particle is on the "inside" of the plane (i.e. intersecting)

$$\text{IF} \quad (x\text{-}p).n < 0$$

In practice we take a threshold distance e i.e. Particle is intersecting ...

$$\text{IF} \quad (x\text{-}p).n < \varepsilon$$

PARTICLE VELOCITY

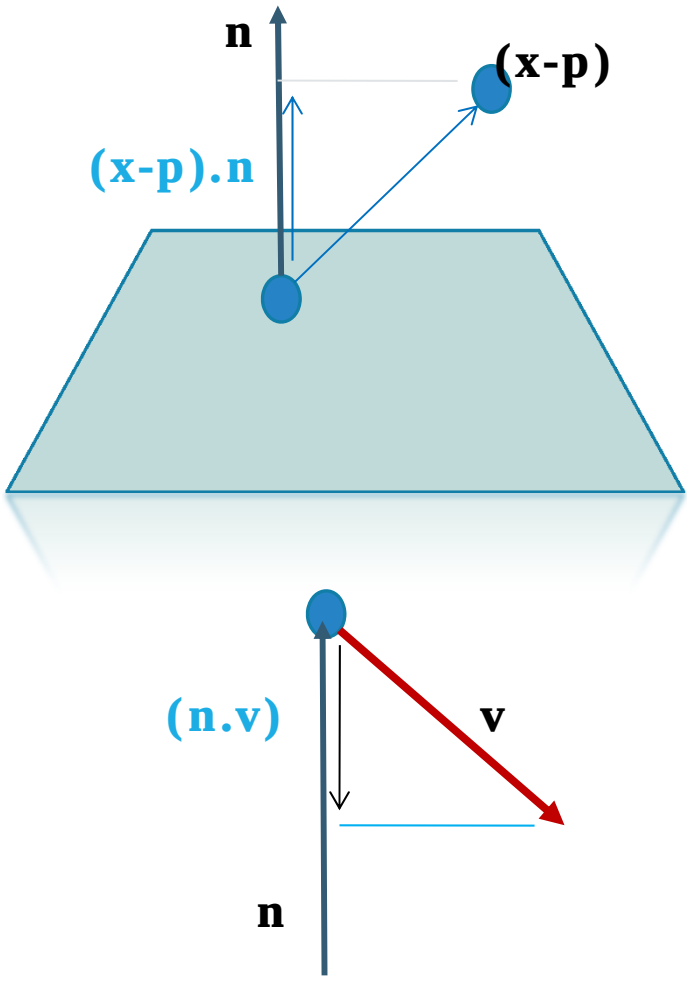


If intersecting we should also check whether the particle is moving further into the plane

- If $(n.v < \varepsilon)$

  Particle is heading deeper into plane

PARTICLE - PLANE COLLISION DETECTION



Given normal n and any point p on plane

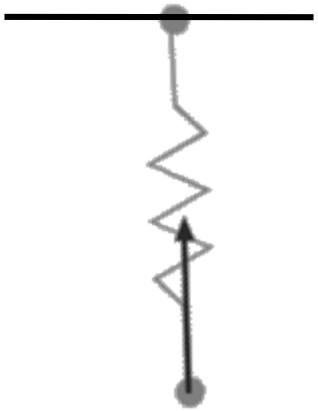
- if $((\boldsymbol{x}-\boldsymbol{p}).\boldsymbol{n} < \varepsilon \quad \&\& \quad (\boldsymbol{n}.\boldsymbol{v} < \varepsilon))$
- i.e. penetrating and going deeper
  - Cancel force into plane
  - Repulse object for physically based "bounce" response

COLLISION RESPONSE: FORCES

Non-interpenetration: Some force opposing the penetrating force should be applied in the opposite direction to cancel out movement into the plane

$$\boldsymbol{f}_{nonpen} = -(\boldsymbol{f}.\boldsymbol{n})\boldsymbol{n}$$

E.g. We could apply a spring force proportional to the interpenetration: (usually called a *penalty method*)



$$\boldsymbol{f}_{nonpen} = -k_{pen}\boldsymbol{n}\Delta q$$

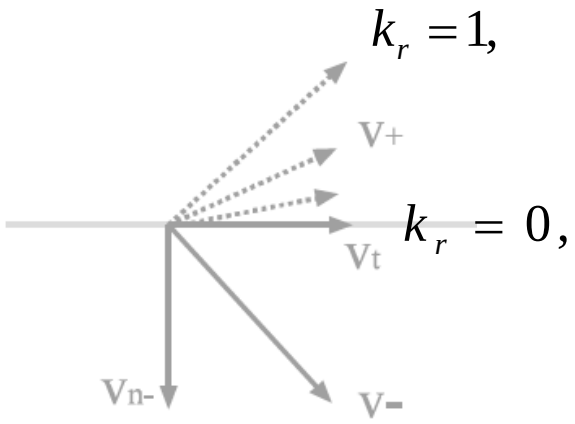
COLLISION RESPONSE: IMPULSES

N.B. THIS IS AN ALTERNATIVE TO THE PREVIOUS SLIDE

Commonly used option is to apply an instantaneous change of velocity: impulse model

For collision response, velocity in the tangential direction is preserved and velocity in normal direction is scaled by $-k_r$, the coefficient of restitution

$$v_t^+ = v_t^-$$
$$v_n^+ = k_r v_n^-$$



$$v^+ = v_t - k_r v_n$$

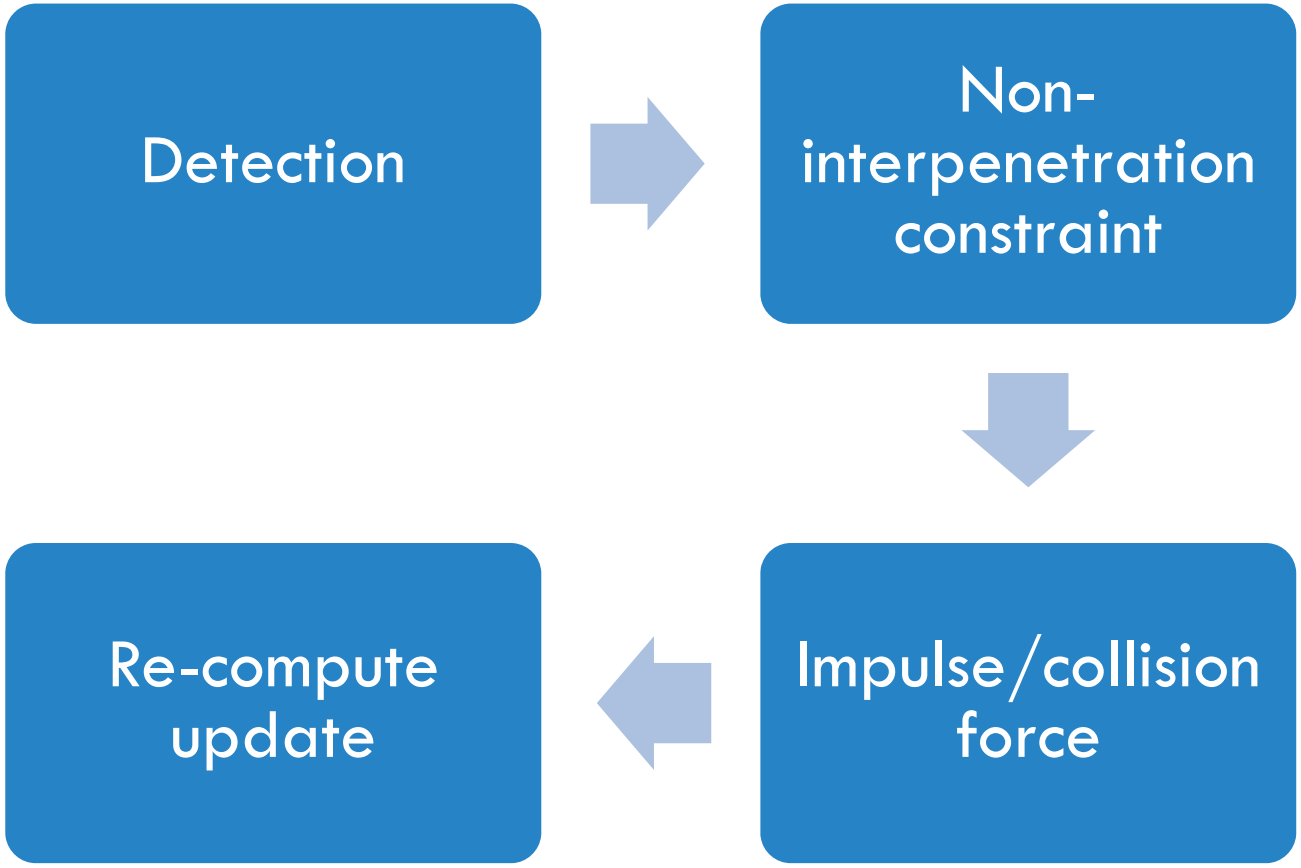
$k_r = 0,$ Fully inelastic

$k_r = 1,$ Fully elastic

v+: post collision velocity
v-: initial velocity
t: tangential to collision plane
n: perpendicular to collision plane

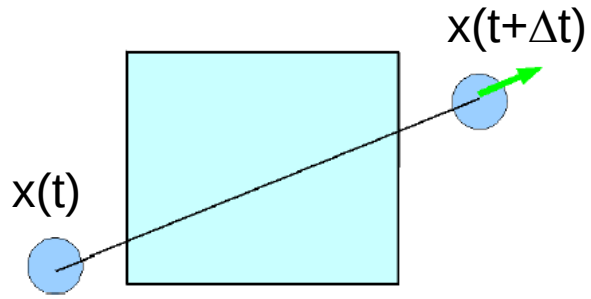
N.B. Impulse or collision force is only created when the object is tending to penetrate further (otherwise you'll add energy into the system)

COLLISION HANDLING



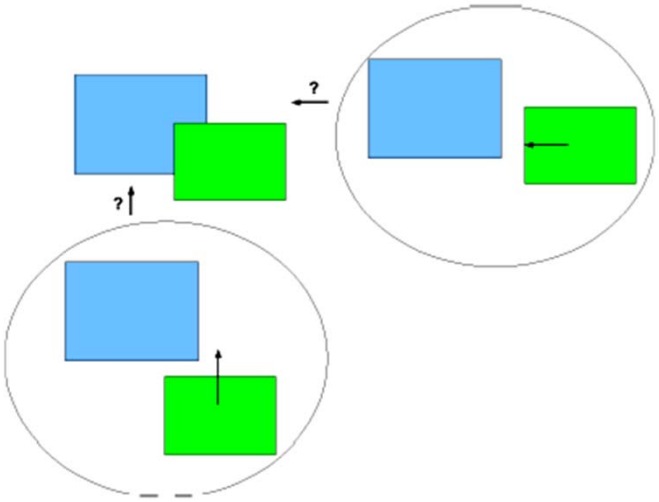
ASIDE: DISCRETE TIME LIMITATIONS

Tunnelling: bullet through paper effect



Undetected traversal through thin objects or high velocities

Penetration-depth ambiguity: what just happened?



BACKTRACKING

N.B. This example is for illustrative purposes and is not a hard and fast solution.

```
if (x-p).n < 0
```

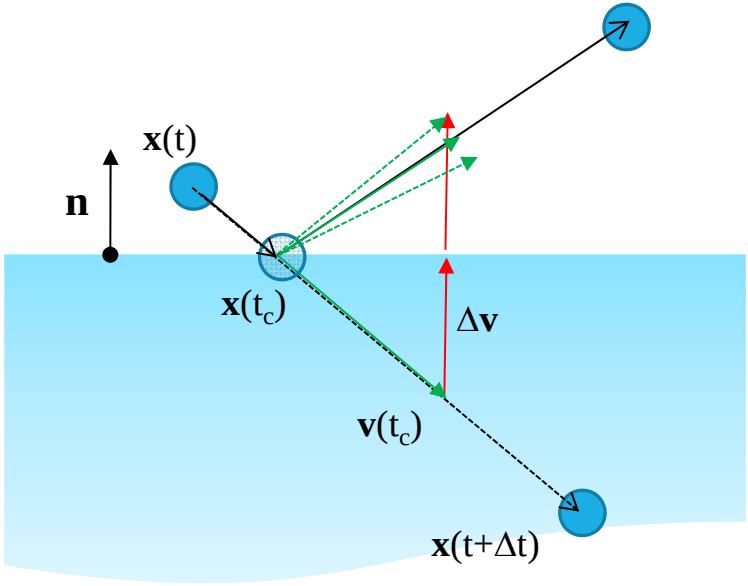
- Backtrack to collision time t_C
  - Assume Linear trajectory within timestep
- Compute change in velocity for elastic collision

$$\Delta v = -(v.n)n$$

- Apply co-efficient of restitution

$$v \mathrel{+}= (1+k_r)\Delta v$$

- Continue simulation



Problem: multiple backtracks may become necessary

POST-PROCESSING

N.B. This example is for illustrative purposes and is not a hard and fast solution.

A.k.a. Projection method

Similar but do not backtrack

```
if (x-p).n < 0
```

- compute positional increment to project the particle back to surface

$$\Delta x = -((x - p).n)n$$

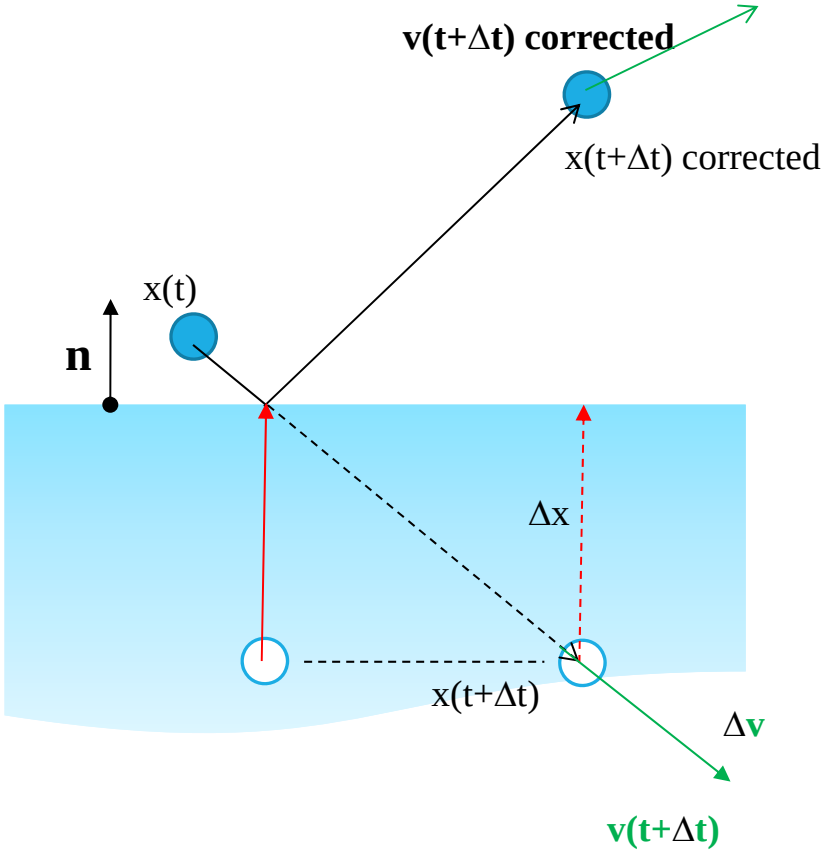
- Apply correction to position

$$x \mathrel{+}= \Delta x$$

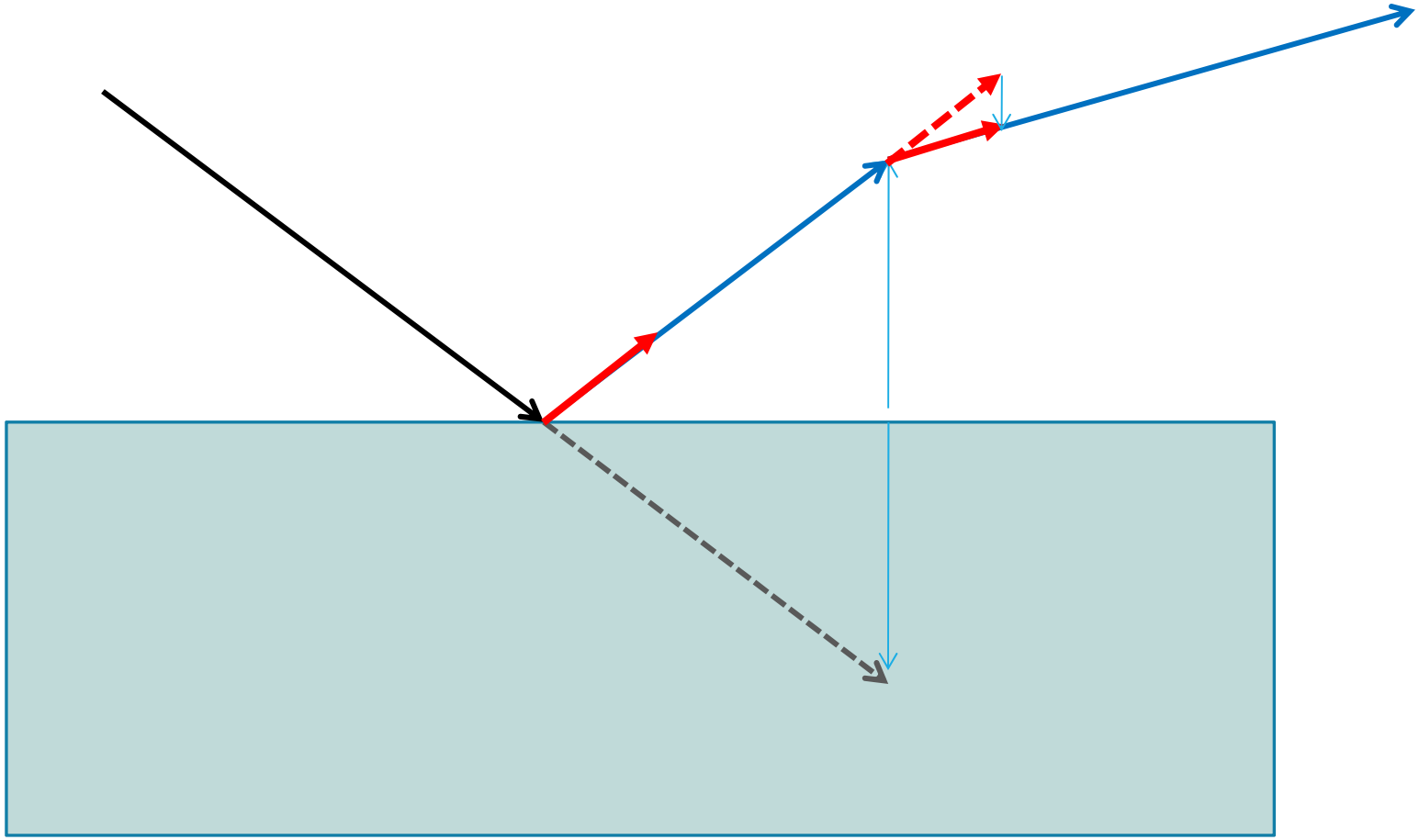
Or even (if big Δt): $\mathbf{x} \mathrel{+}= (1 + k_r)\Delta \mathbf{x}$

- $\Delta \mathbf{v}$ calculated as before

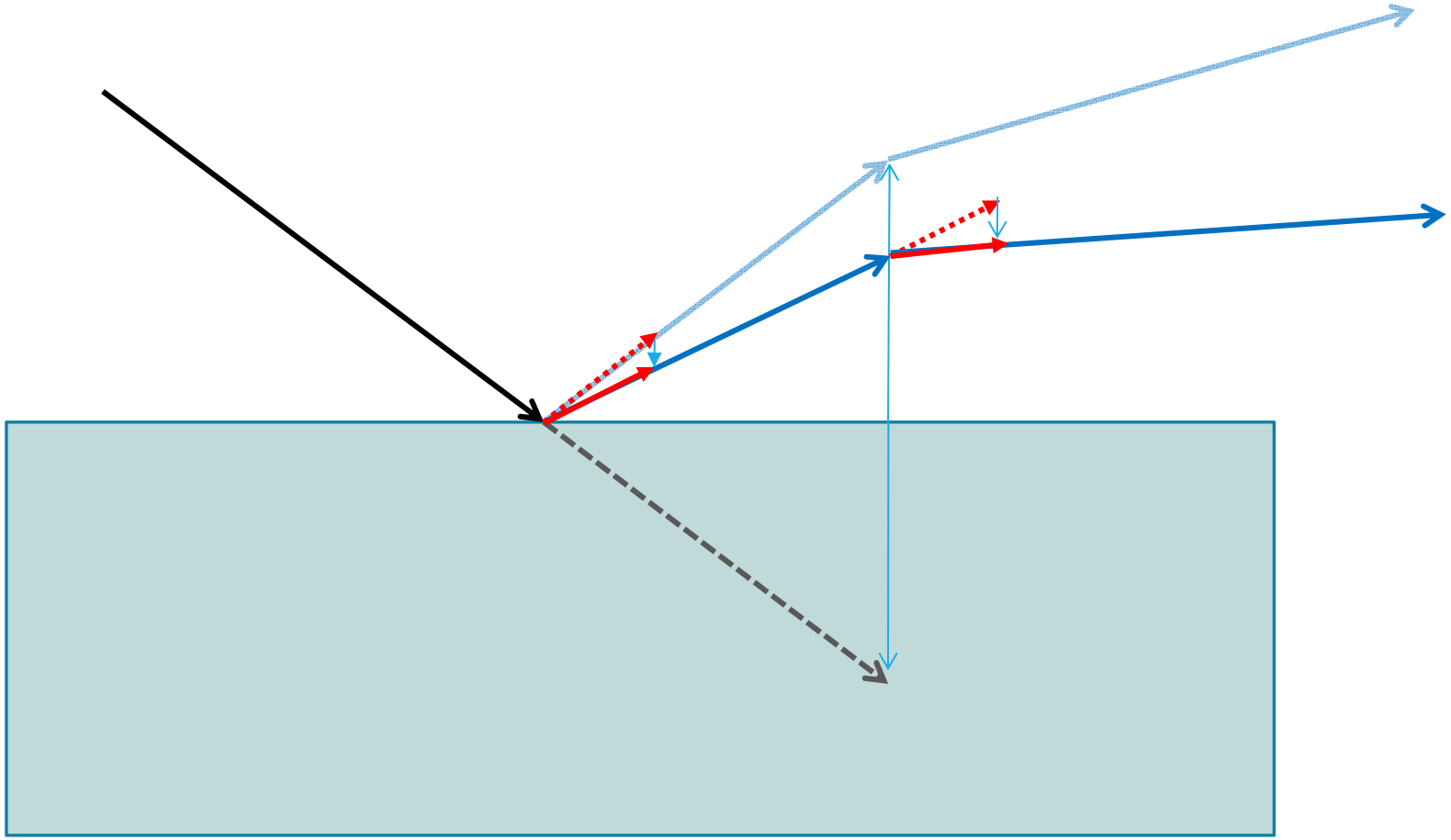
Advantage: all collisions handled at the same time



POST-PROCESSING



BACK-TRACKING



BACKTRACKING VS. POSTPROCESSING

Backtracking

- More accurate solution
- Expensive
  - Particularly with more complex objects + simultaneous interactions
  - Potential for infinite loop?

Post-processing

- Suitable for some simple interactions
- May not be guaranteed a solution

Other techniques:

- Backtrack using Time-step bisection
- Simply move particle to previous time step
- Gradually apply projection over several iterations
- Use penalty method (spring)

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

INTEGRATION

EXAMPLE

An object is moving at 3 meters per second

- After 1 second how much has it moved : 3 meters
- After 2 second ... 6 meters
- After 2.5 seconds ... 7.5 meters

But what if the speed is also changing?

EXAMPLE

We could say:

- Position = Position + Velocity * timestep;
- Velocity = Velocity + Acceleration * timestep;

Time	Acceleration	Position	Velocity
0	2	0	0
1	2	0	2
2	2	2	4
3	2	6	6
4	2	12	8

NOTE this is an approximation

- Dependent on timestep

EXAMPLE 2 (DIFFERENT TIMESTEP)

$\Delta t = 1$

Time	Acceleration	Position	Velocity
0	2	0	0
1	2	0	2
2	2	2	4
3	2	6	6
4	2	12	8

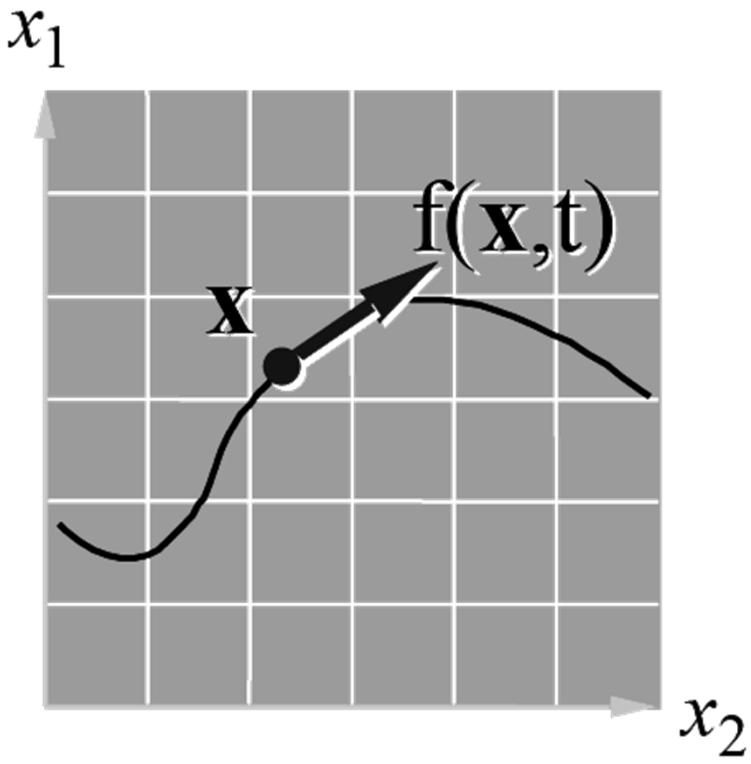
$\Delta t = 0.5$

Time	Acceleration	Position	Velocity
0	2	0	0
0.5	2	0	1
1	2	1	2
1.5	2	1	3
2	2	2.5	4
2.5	2	4.5	5
3	2	7	6
3.5	2	10	7
4	2	13.5	8
4.5	2	17.5	9

Position = Position + Velocity * timestep;

Velocity = Velocity + Acceleration * timestep;

A CANONICAL DIFFERENTIAL EQUATION



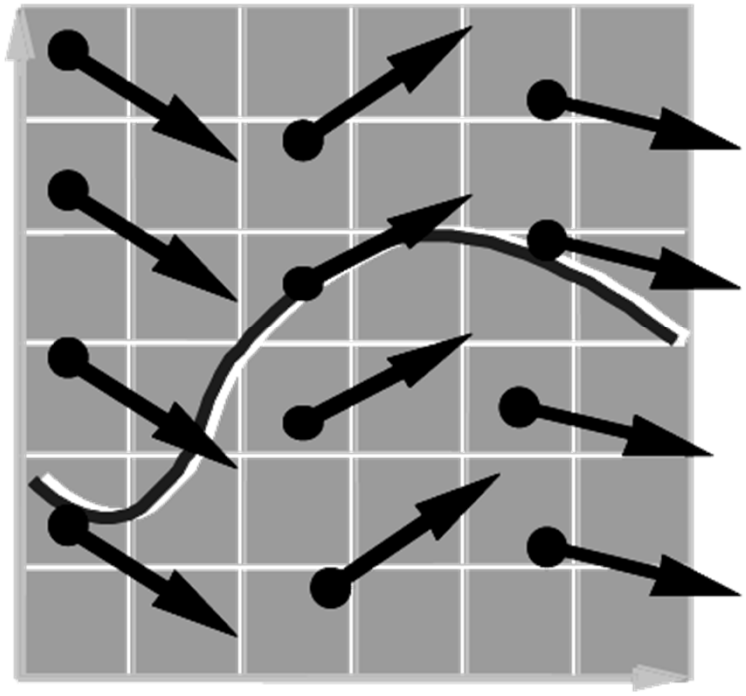
$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t)$$

- **x(t): a moving point.**

$f(\mathbf{x}, t)$: x's velocity.

N.B. f doesn't mean force here

VECTOR FIELD



The differential equation

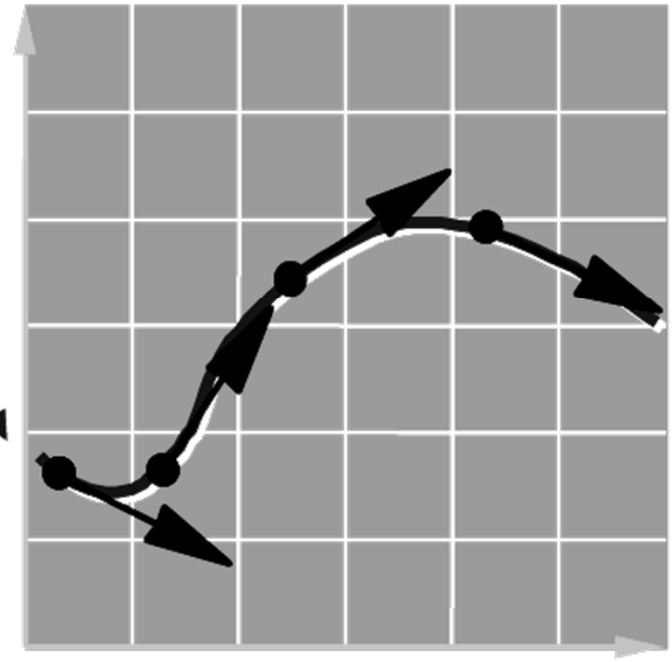
$$\dot{x} = f(x, t)$$

defines a vector field over x.

INTEGRAL CURVE

Start Here

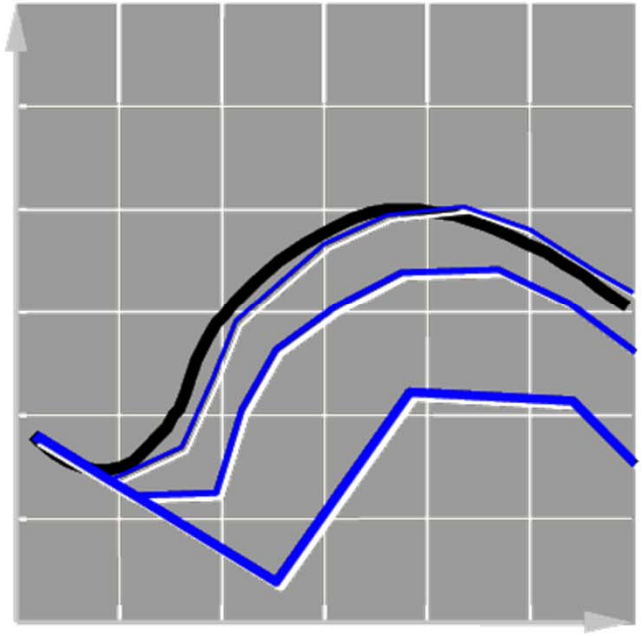
N.B. An Initial Value Problem



Pick any starting point, and follow the vectors.

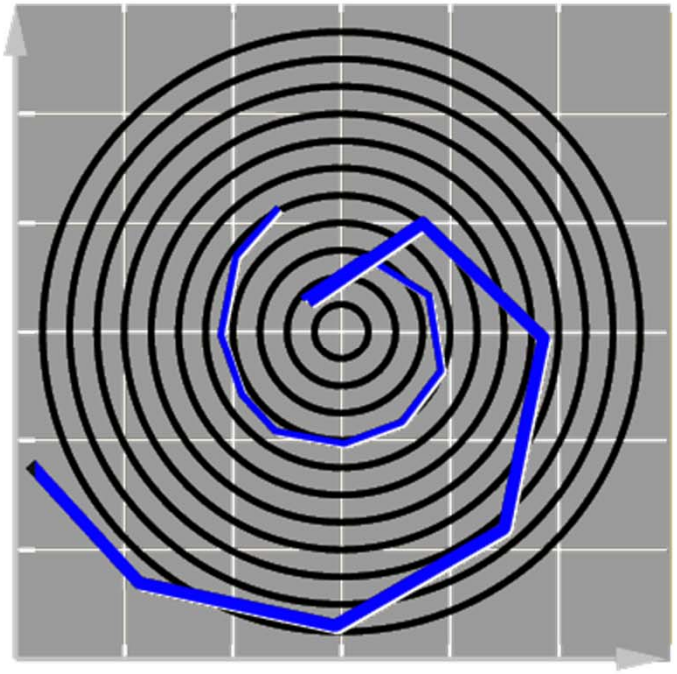
INTEGRATION: EULER'S METHOD

$$\boldsymbol{x}(t + \Delta t) = x(t) + f(\boldsymbol{x}, t)\,\Delta t$$



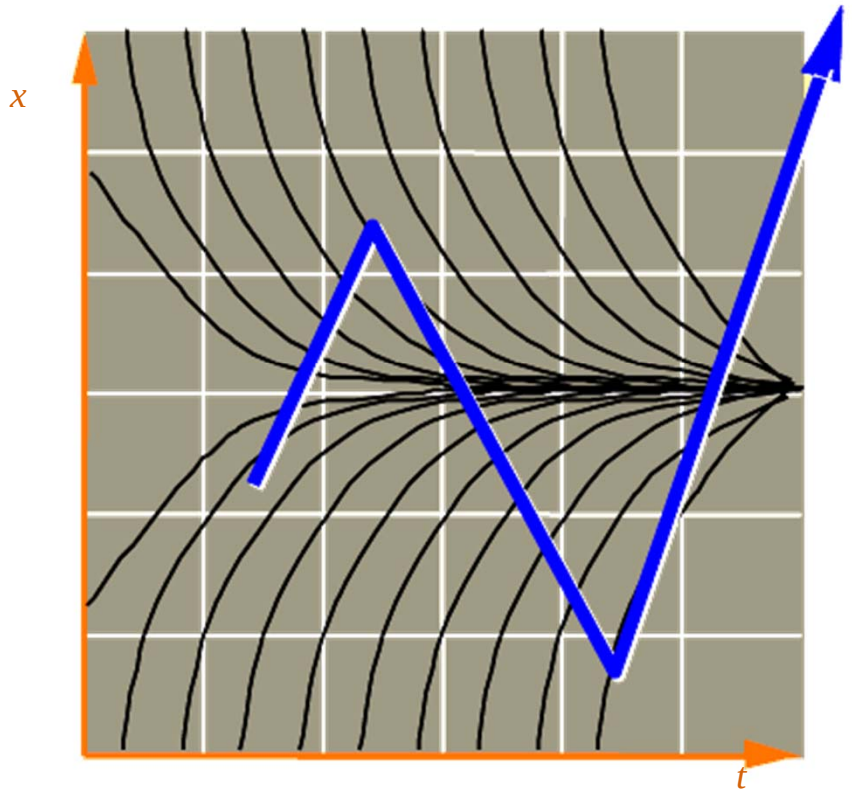
- **Simplest numerical solution method**
- **Discrete time steps**
- **Bigger steps, bigger errors.**

EULER PROBLEMS: INACCURACY

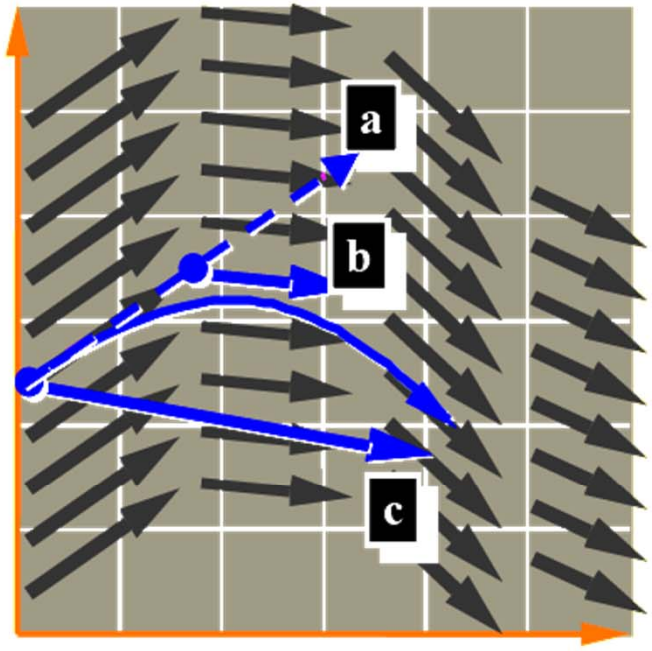


Error turns x(t) from a circle into the spiral of your choice.

EULER PROBLEMS: INSTABILITY



THE MIDPOINT METHOD



a. Compute an Euler step

$$\Delta \boldsymbol{x} = f(\boldsymbol{x}, t)\Delta t$$

b. Evaluate f at the midpoint

$$f_{mid} = f\left(\frac{\boldsymbol{x} + \boldsymbol{\Delta x}}{2}, \frac{t + \Delta t}{2}\right)$$

c. Take a step using the midpoint value

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + f_{mid}\Delta t$$

EULER

A.k.a. Forward Euler

Easy to code (lazy)

Quick for each iteration

$$\boldsymbol{f} = calculateForces(\boldsymbol{x}, \boldsymbol{v}, t_0)$$

$$\boldsymbol{v} = \boldsymbol{v_0} + \frac{\boldsymbol{f}}{m}\Delta t$$

$$\boldsymbol{x} = \boldsymbol{x_0} + \boldsymbol{v}\Delta t$$

Unfortunately

- Unstable except for very small time steps
- Many more steps required

MID-POINT METHOD

Two state updates per iteration

However a lot more stable than Euler for large time steps

- Energy conservative if acceleration constant [1]

In practice more efficient as we have to do less time steps

$$\boldsymbol{f}_1 = calculateForces(\boldsymbol{x}, \boldsymbol{v}, t_0)$$

$$\boldsymbol{v}_{midpnt} = \boldsymbol{v}_0 + \frac{\boldsymbol{f}_1}{m}\frac{\Delta t}{2}$$

$$\boldsymbol{x}_{midpnt} = \boldsymbol{x}_0 + \boldsymbol{v}\frac{\Delta t}{2}$$

$$\boldsymbol{f} = calculateForces\,(\boldsymbol{x}_{midpt}, \boldsymbol{v}_{midpt}, t_0 + \frac{\Delta t}{2})$$

$$\boldsymbol{v} = \boldsymbol{v}_0 + \frac{\boldsymbol{f}}{m}\Delta t$$

$$\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{v}_{midpt}\Delta t$$

[1] Integration in Blender: http://www.blender.org/development/release-logs/blender-246/particles/newtonian-physics/

RUNGE-KUTTA 4 (RK4)

Fourth order method. Slower than midpoint method

More stable:

- Energy conservative even with non-constant acceleration

$$
\begin{aligned}
k_1 &= h f(x_n, y_n) \\
k_2 &= h f(x_n + h/2, y_n + k_1/2) \\
k_3 &= h f(x_n + h/2, y_n + k_2/2) \\
k_4 &= h f(x_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}
\end{aligned}
$$

h = time step

INTEGRATION

See Physically Based Modeling Siggraph Course Notes 2001 by Baraff, Witkin, Kass

- http://www.pixar.com/companyinfo/research/pbm2001/
- See “Differential Equation Basics” Lecture 2 by Andrew Witkin

Other techniques: backward Euler, semi-implicit Euler, Verlet integration, Heun's method, Leapfrog integration

We’ll come back to this later

REQUIRED READING

Must Read: Baraff, Witkin and Kass "Physically based Animation Course notes C. Particle Systems" – Siggraph 2001

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

Also have a brief look at: "Particle Simulation and Rendering Using Data Parallel Computation" - Karl Sims, Siggraph 1990

- http://www.karlsims.com/papers/ParticlesSiggraph90.pdf
- Youtube Video: http://www.youtube.com/watch?v=hwDvna0q3rA

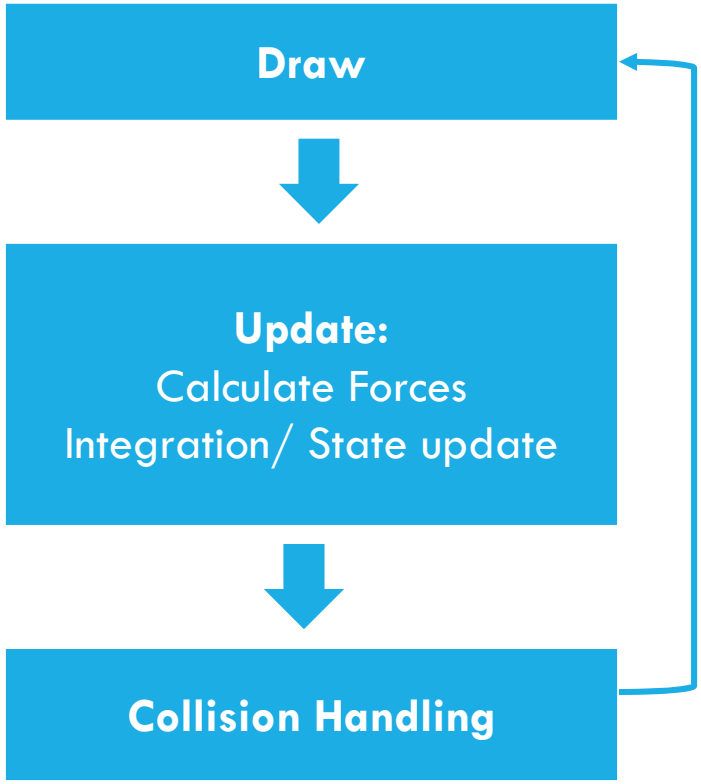
ADDITIONAL REFERENCES

Mueller, James, Theurey, Stam "Real-time Physics" – Siggraph 2008

- http://www.matthiasmueller.info/realtimephysics/index.html

- Ian Millington – "Game Physics Engine Development"
- Kenny Erleben - "Physics based animation"
- David H.Eberley - "Game Physics Engine Development"

PHYSICS PIPELINE FOR PARTICLE SYSTEM



Procedures	Data Structures
RenderParticles	Particles: x, v, f, m
CalculateForces	Particle_System: Particles, Forces
<Force Functions>	Forces: Force Functions Particles
Integration (Euler is sufficient)	
CollisionHandling	

INTEGRATORS

HTTP://WWW.BLENDER.ORG/DEVELOPMENT/RELEASE-LOGS/BLENDER-246/PARTICLES/NEWTONIAN-PHYSICS/

Integrators are a set of mathematical methods available to calculate the movement of particles. The following guidelines will help to choose a proper integrator, according to the behaviour aimed at by the animator.

- **Euler**: Also known as *Forward Euler*. Simplest integrator. Very fast but also very unstable. If no dampening is used, particles get more and more energy over time. For example, bouncing particles will bounce higher and higher each time. Should not be confused with Backward Euler (not implemented) which has the opposite feature, energies decrease over time, even with no dampening. Use this integrator for short simulations or simulations with a lot of dampening where speedy calculations is more important than accuracy.
- **Midpoint**: Also known as *2nd order Runge-Kutta*. Slower than **Euler** but much more stable. If the acceleration is constant (no drag for example), it is energy conservative (also known as symplectic). It should be noted that in example of the bouncing particles, the particles might bounce higher than they started once in a while, but this is not a trend. This integrator is a generally good integrator for use in most **cases**.
- **RK4**: Short for *4th order Runge-Kutta*. Similar to **Midpoint** but slower and in most cases more accurate. It is energy conservative even if the acceleration is not constant. Only needed in complex simulations where **Midpoint** is found not to be accurate enough.

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

ASSIGNMENT 1: PARTICLE SYSTEM

Demo due: Monday 25th January, 2016 | Cumulative Report and Video deadline: Sunday 28rd February 2016 @ 23:59
20% penalty per day late

ASSIGNMENT 1: PARTICLE SYSTEMS

Read Particle Systems notes by Baraff et al (see references)

Implement a particle system (roughly based on Siggraph 2001 course notes by Baraff, Witkin and Kass)

Compulsory requirements:

1. Animate at-least 1000 particles
2. Include at least two forces
3. Implement Particle-Plane Collision Handling
4. Implement evolution* / recycling** OR ensure particles stay in the scene
5. For rendering, you should use **OpenGL**, DirectX or XNA

Optional Requirements:

- Implement a meaningful demo that recreates real-world or aesthetic effect(s)
  - e.g. implement effects based on Karl Sims paper
- Implement an alternative to forward-Euler integration

* Evolution: particles appearance parameters other than physics state changes over time
** Recycling: particles "die" and are destroyed and regenerated

MARKING AND DELIVERABLES

This assignment is worth 10% of the module

- Compulsory Requirements (Quantitative marking): ~50%
- Style / complexity / optional components (Qualitative marking): ~50%

You must do a 5 minute demo in the lab next week (at your own machine)

- Demonstrate the features of your implementation
- Talk about compulsory requirements / optional features implemented
- Mention any underlying technical improvements, personal design decisions beyond the basic spec provided in notes

You must also submit the following in the cumulative submission

- A report (max 1 page): brief description of project, in particular anything not within the compulsory components.
- A link to a youtube video (make this un-listed if you wish). Can be combined with later labs.

REQUIRED READING

Must Read: Baraff, Witkin and Kass "Physically based Animation Course notes C. Particle Systems" – Siggraph 2001

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

Also have a brief look at: "Particle Simulation and Rendering Using Data Parallel Computation" - Karl Sims, Siggraph 1990

- http://www.karlsims.com/papers/ParticlesSiggraph90.pdf
- Youtube Video: http://www.youtube.com/watch?v=hwDvna0q3rA