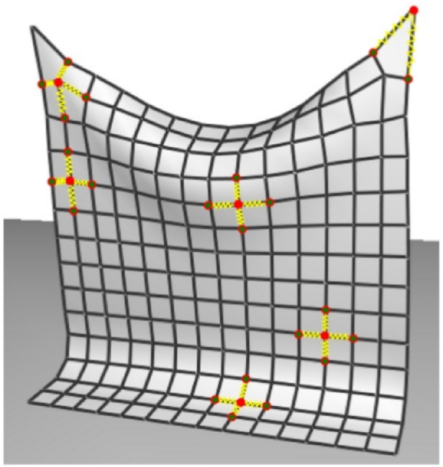
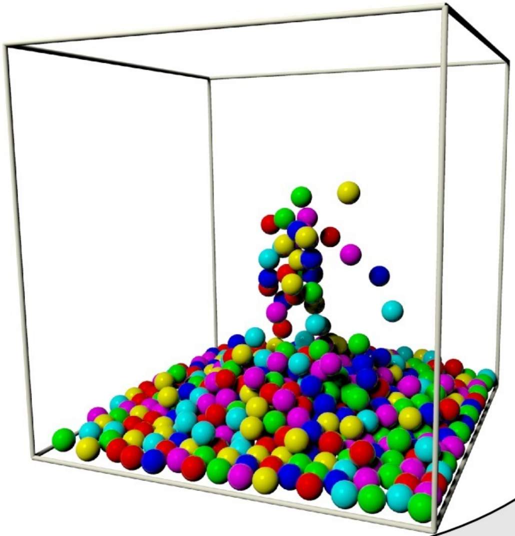
$$\frac{u^{k+1} - u^k}{\Delta t} = \mathrm{A}\, u^{k+1} + \mathrm{b}$$

# Physically based animation

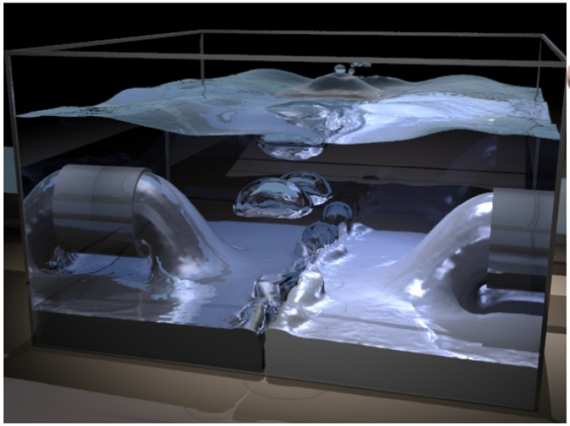


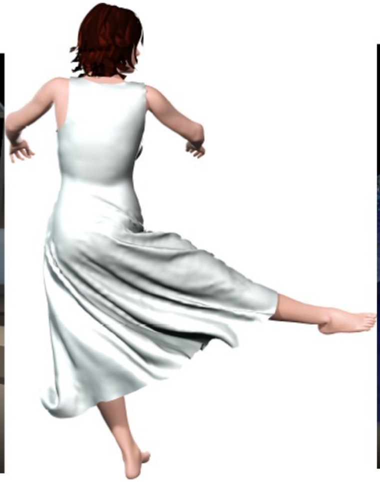


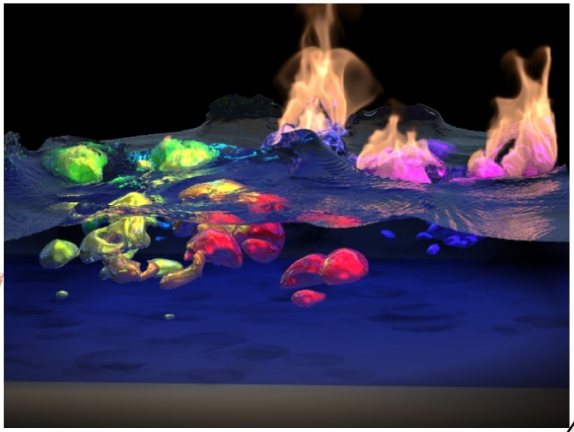
# Physically based animation

When using physically based animation?

- For complex deformation we cannot model "by hand"
- To model physical phenomenon







*[Fedkiw SIGGRAPH 06], [Grinspun SIGGRAPH 07]*

# Principle

1. Define a system $S_0 \text{ at } t=0$

positions, speed, forces

2. Model the temporal evolution of the system using ODE / PDE

$$\mathcal{F}\left(S(t), \frac{\partial^n S}{\partial t^n}, \frac{\partial^m S}{\partial x^m}\right) = 0$$

We usually considers $\dfrac{\partial S}{\partial t} = \mathrm{A}S + \mathcal{H}\left(S(t), \dfrac{\partial^m S}{\partial x^m}\right)$

3. We numericaly solve in moving forward in time by $\Delta t$

# Modelization

Level of precisions

**1. Particles systems**
Everything is reduced to a point (no rotation, EDO)

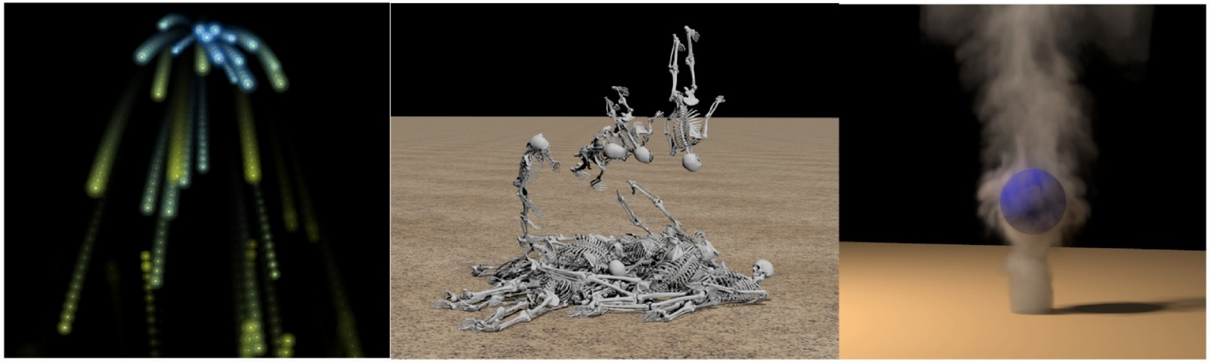
+Simple
- Accuracy

**2. Solid mecanics** (rigid bodies)
Unique set of parameters for the entire object

+Inertia
- Collision

**3. Continuum mechanics**
Varying parameters within the shape (PDE: FEM, FV, ...)

+ Accurate
- Heavy



# Modelization

Level of precisions

**1. Particles systems**

$$ma = \sum_i f_i \Rightarrow x_i'' = F(x, x', t)$$

**2. Solid mecanics** (rigid bodies)

$x(t)$: position
$R(t)$: rotation
$\omega(t)$: angular speed
$L(t)$: angular momentum
$\tau(t)$: torque

$$\frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \omega(t)^* R(t) \\ F(t) \\ \tau(t) \end{pmatrix}$$

**3. Continuum mechanics**

deformation

stress

$$\nabla \cdot \sigma + \mathbf{F}_{\text{ext}} = \eta \frac{\mathrm{d}^2\mathbf{u}}{\mathrm{d}t^2}$$

$$\sigma = E\epsilon$$

strain

$$\varepsilon = \frac{1}{2}(\nabla\mathbf{u} + [\nabla\mathbf{u}]^T + [\nabla\mathbf{u}]^T \nabla\mathbf{u})$$

# Numerical solution of ODE
# (Ordinary Differential Equation)

# What is an ODE ?

Example:

f is an unknown **function**

$$f'(x) = af(x) + b$$

dérivative of f
depends of f

f is 1D function

*(otherwise PDE)*

Can also be written as: $y' = ay + b$

# What is an ODE ?

More examples:

Linear, constant coefficients
$$f'(x) = 4f(x) + 2$$

Linear, variable coefficients
$$f'(x) = 4(x-5)f(x) + 2x^2 - 7$$

Non linear
$$f'(x) = 4x\sin(xf(x)) + 2/f^2(x)$$

General expression
$$\boxed{f'(x) = \mathcal{F}(x, f)}$$

Even more general: Implicit formulation
$$\mathcal{R}(x, f, f') = 0$$

# What is an ODE?

Order of ODE:

Second order,
linear

$$f''(x) = 2f'(x) + 3f(x) - 4$$

Third order,
non linear

$$f^{(3)}(x) = 2x^2 \sin\left(f''(x) + f'(x)\right) - f^2(x)$$

General definition:

$$\boxed{f^{(n)}(x) = \mathcal{F}(x, f, f', \ldots, f^{(n-1)})}$$

Even more general:

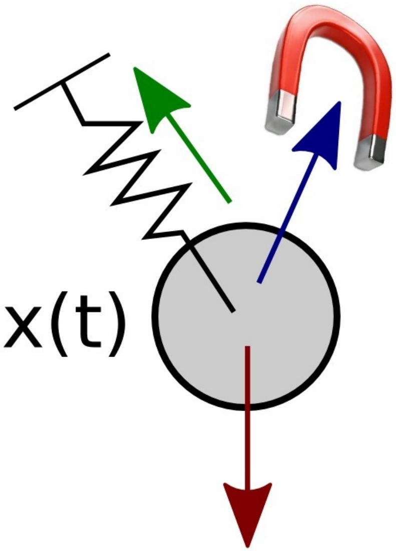
$$\mathcal{R}(x, f, f', \ldots, f^{n-1}, f^{(n)}) = 0$$

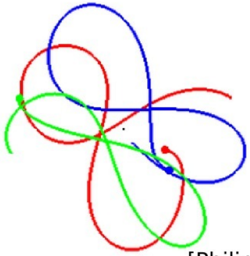
# Why do we need ODE?

Physics:

$$m\,a(t) = \sum F(x(t), t)$$

$$x''(t) = \frac{1}{m}F(x(t))$$



[Philippe Roux]

# Why do we need ODE?

Biology:

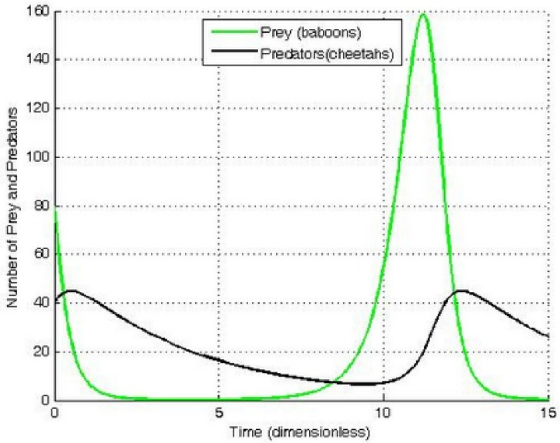
Population grows

f1 : prey
f2 : predator

$$\begin{cases} f_1'(t) = f_1(t)(\alpha - \beta f_2(t)) \\ f_2(t) = -f_2(t)(\gamma - \delta f_1(t)) \end{cases}$$

*Lokta-Volterra equation*

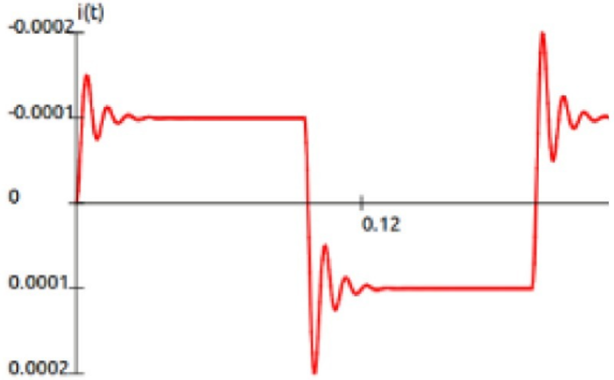
*[Wikipedia]*

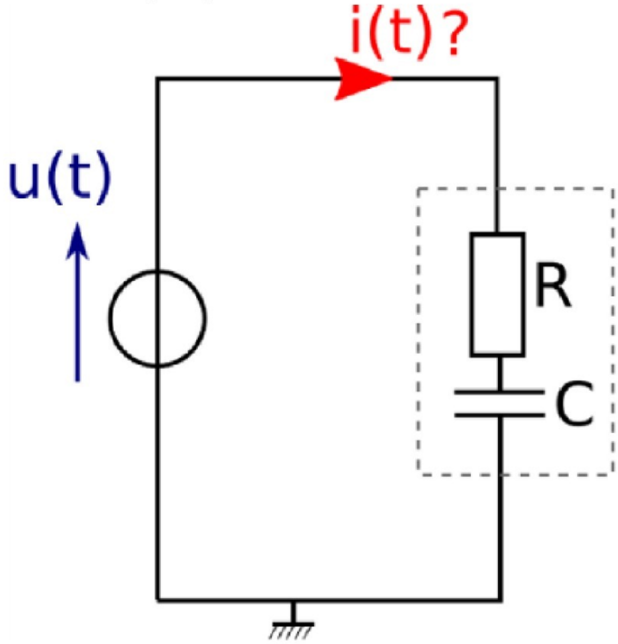


# Why do we need ODE?

Physics:

$$RCi'(t) + i(t) = Cu'(t)$$



# What can we solve analytically?

Linear + constant coefficient

$$a_0 f(x) + a_1 f'(x) + \cdots + a_n f^{(n)}(x) = r(x)$$

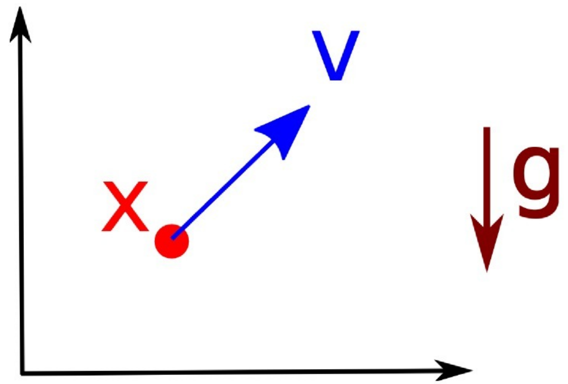
Linear + variable coefficient + low order

$$a_0(x) f(x) + a_1(x) f'(x) + a_2(x) f''(x) = r(x)$$

Non linear: Almost never
Existence and uniqueness ?

# Case study: Free fall under gravity

Motion equations:




Initial conditions:

$$x(t=0)=x_0$$
$$v(t=0)=v_0$$

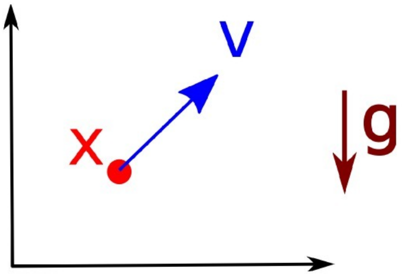
# Numerical approach

Motion equations:

$$x'(t)=v(t)$$
$$v'(t)=g$$

Initial conditions:

$$x(t=0)=x_0$$
$$v(t=0)=v_0$$




Approximation:

$$f'(t) \simeq \frac{f(t+\mathrm{d}t) - f(t)}{\mathrm{d}t}$$

# Numerical approach

Motion equations:

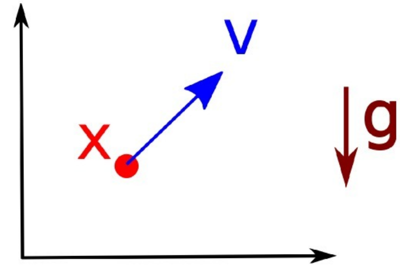
x'(t)=v(t)

v'(t)=g

Initial conditions:

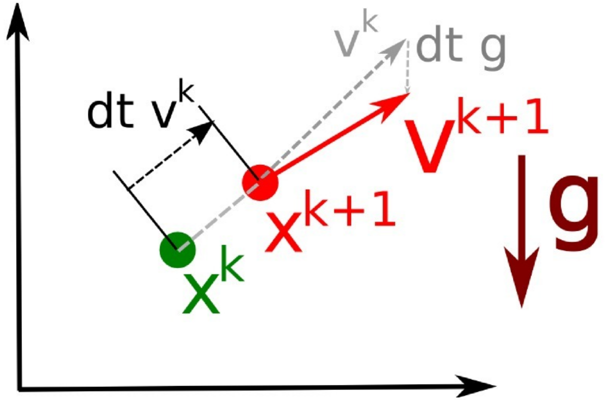
$x(t=0)=x_0$

$v(t=0)=v_0$




## Solution

$$\begin{cases} v^{k+1} = v^k + (\Delta t) g \\ x^{k+1} = x^k + (\Delta t) v^k \end{cases}$$




Code;

```
x=x0;
v=v0
for(k=0;k<N;++k)
{
  x=x+dt*v;
  v=v+dt*g;
}
```

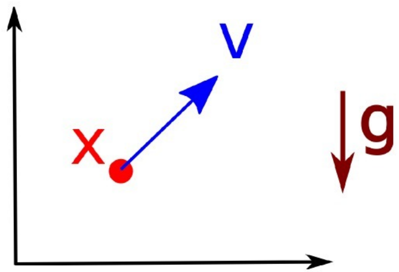
# Numerical approach

Motion equations:

x'(t)=v(t)
v'(t)=g

Initial conditions:

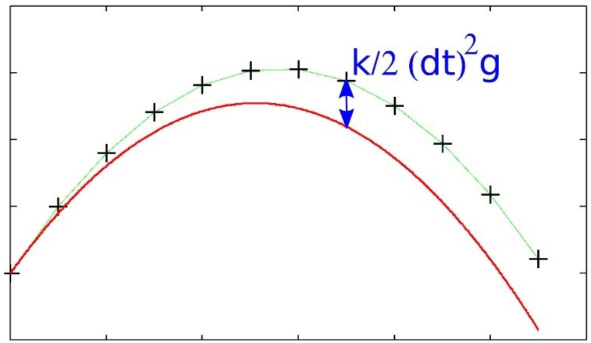
$x(t=0)=x_0$
$v(t=0)=v_0$




Numerical solution:

$$\begin{cases} v^{k+1} = v^k + (\Delta t)g \\ x^{k+1} = x^k + (\Delta t)v^k \end{cases}$$

$$\Rightarrow \begin{cases} x^{k+2} = 2x^{k+1} - x^k + (\Delta t)^2 g \\ x^0 = x_0 \\ x^1 = x_0 + \Delta t v_0 \end{cases}$$




$$\Rightarrow \quad x(t = k\Delta t) = x_0 + (k\Delta t)\, v_0 + \frac{k(k-1)}{2}(\Delta t)^2 g$$

Real solution:

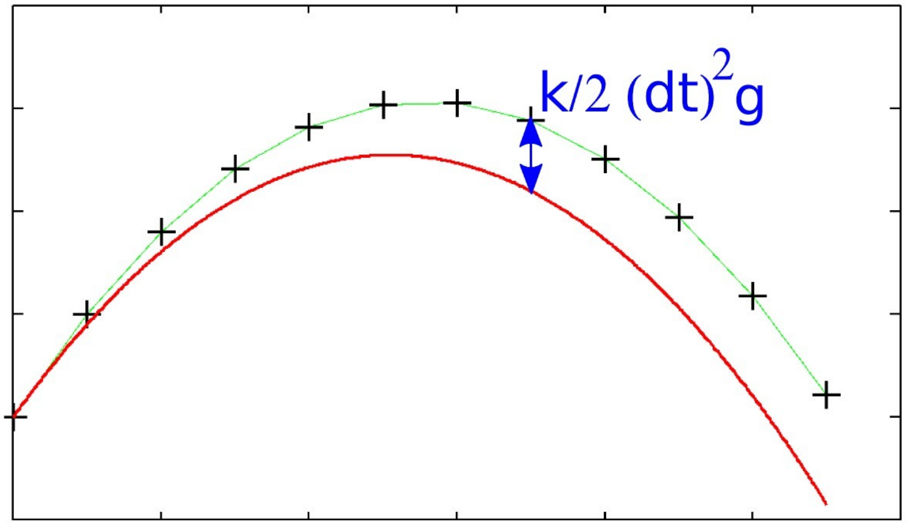
$$\tilde{x}(t = k\Delta t) = x_0 + (k\Delta t)v_0 + \frac{1}{2}(k\Delta t)^2 g$$

# Accuracy

Numerical error: $\|x(k\Delta t) - \tilde{x}(k\Delta t)\| = \frac{k}{2}(\Delta t)^2 g$

Definition of accuracy of order h:

$$\|x(k\Delta t) - \tilde{x}(k\Delta t)\| = \mathcal{O}((\Delta t)^{h+1})$$



# Matrix formulation

Linear ODE of order n
= system of ODE of order 1

$$u(t) = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}$$

$$u'(t) = \mathrm{A}u(t) + \mathrm{b}(t)$$

with $\mathrm{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ $\mathrm{b}(t) = \begin{pmatrix} 0 \\ g \end{pmatrix}$ $u(t) = \begin{pmatrix} x(t) \\ v(t) \end{pmatrix}$

# Matrix formulation

$$u'(t) = \mathrm{A}u(t) + \mathrm{b}(t)$$

$$u(t) = \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} \qquad \mathrm{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$\mathrm{b}(t) = \begin{pmatrix} 0 \\ g \end{pmatrix}$$

Same approach:

$$u^{k+1} = (\mathrm{I} + \Delta t\ \mathrm{A})\, u^k + \Delta t\ \mathrm{b}$$

Same solution ...

# Matrix formulation

$$u'(t) = \mathrm{A}u(t) + \mathrm{b}(t) \qquad u(t) = \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} \qquad \mathrm{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$\mathrm{b}(t) = \begin{pmatrix} 0 \\ g \end{pmatrix}$$

?

Explicit Euler: $$\frac{u^{k+1} - u^k}{\Delta t} = \mathrm{A}\, u^k + \mathrm{b}$$

# Matrix formulation

$$u'(t) = \mathrm{A}u(t) + \mathrm{b}(t) \qquad u(t) = \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} \qquad \mathrm{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$\mathrm{b}(t) = \begin{pmatrix} 0 \\ g \end{pmatrix}$$

?

Explicit Euler: $$\frac{u^{k+1} - u^k}{\Delta t} = \mathrm{A}\, u^k + \mathrm{b}$$

Implicit Euler: $$\frac{u^{k+1} - u^k}{\Delta t} = \mathrm{A}\, u^{k+1} + \mathrm{b}$$

now
unknown

# Implicit Euler

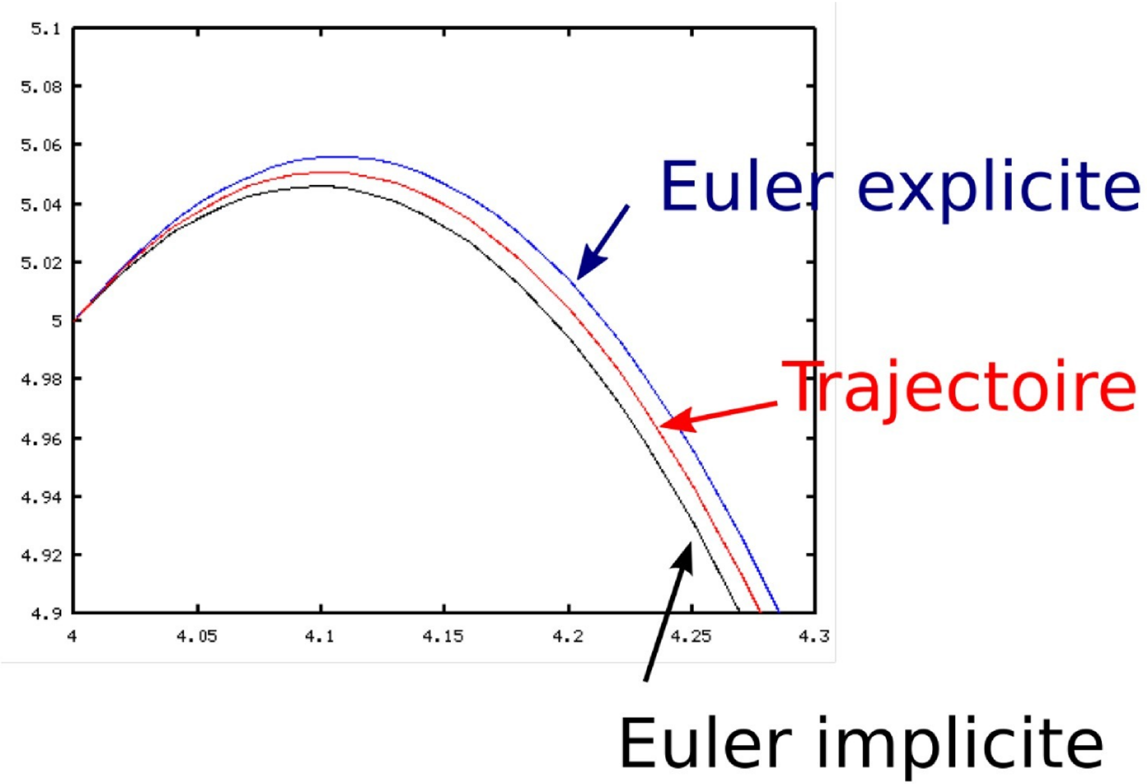
$$u^{k+1} = \left(\mathrm{I} - \Delta t \mathrm{A}\right)^{-1} \left(u^k + \Delta t \, \mathrm{b}\right)$$

$$\begin{cases} x^{k+2} = 2x^{k+1} - x^k + (\Delta t)^2 g \\ x^0 = x_0 \\ x^1 = x_0 + \Delta t \, v_0 + (\Delta t)^2 g \end{cases}$$

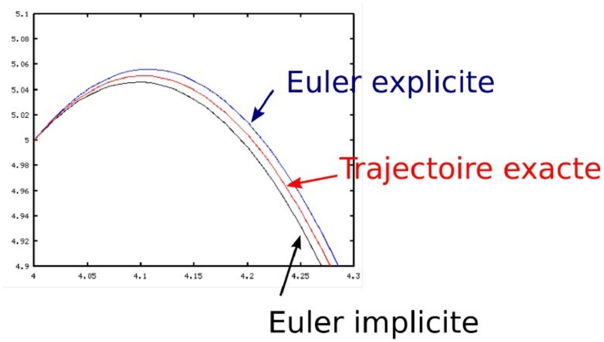
$$x(k\Delta t) = x_0 + (k\Delta t)\, v_0 + \frac{k(k+1)}{2}(\Delta t)^2 g$$

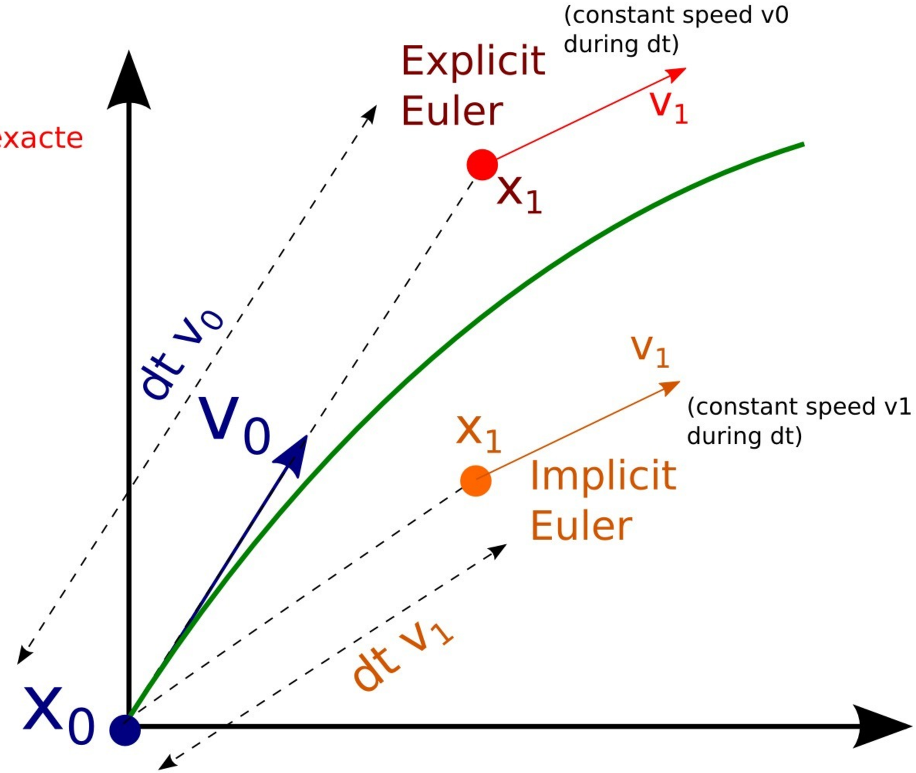
Still not the true solution

# Implicit Euler



# Implicit Euler



# Summary Explicit/Implicit Euler

$$u'(t) = F(t, u)$$

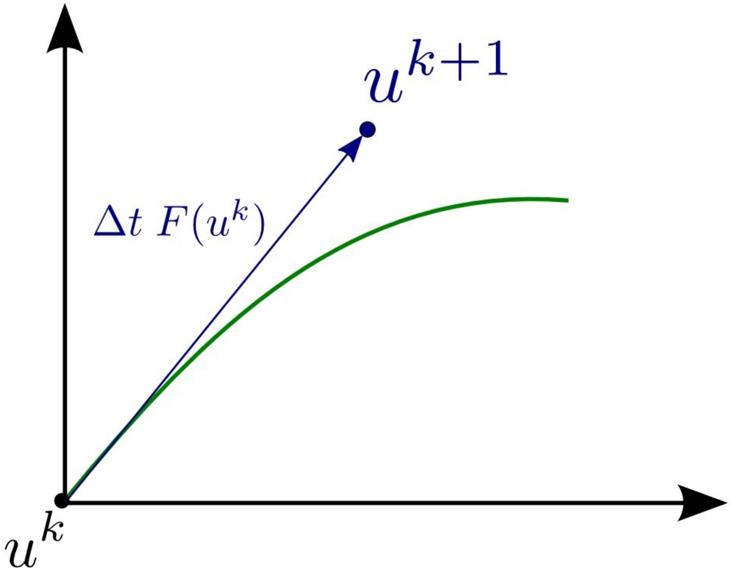
Explicit Euler: $$\frac{u^{k+1} - u^k}{\Delta t} = F(u^k)$$

Implicit Euler: $$\frac{u^{k+1} - u^k}{\Delta t} = F(u^{k+1})$$

*(F must be inverted)*

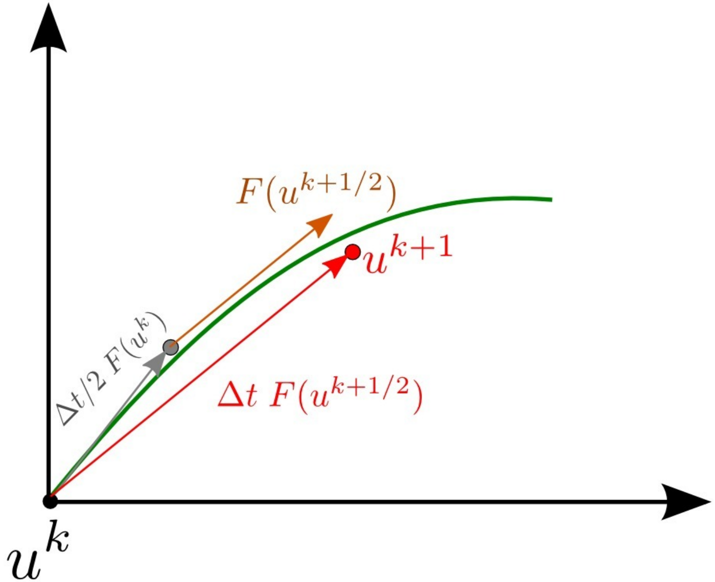
# Runge Kutta

Explicit Euler



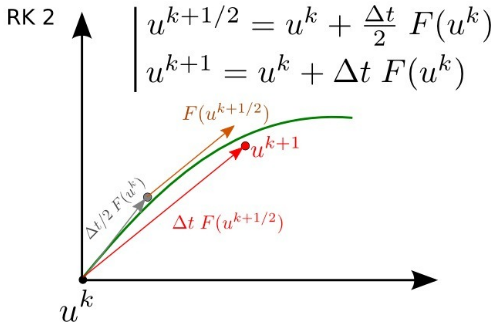

$$u^{k+1} = u^k + \Delta t\ F(u^k)$$

Runge Kutta (order 2)




$$\left| \begin{aligned} u^{k+1/2} &= u^k + \frac{\Delta t}{2}\ F(u^k) \\ u^{k+1} &= u^k + \Delta t\ F(u^{k+1/2}) \end{aligned} \right.$$

# Runge Kutta: Free fall

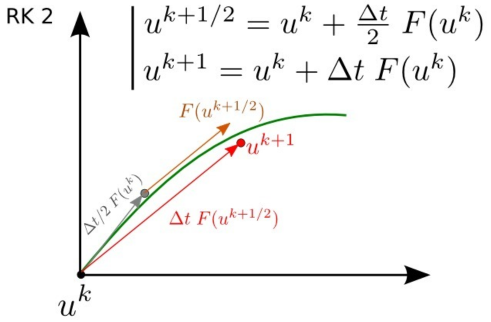



$$F : u^k = (x^k, v^k) \mapsto \begin{pmatrix} v^k \\ g \end{pmatrix}$$

$$u^{k+1/2} = \begin{pmatrix} x^k + \Delta t/2\, v^k \\ v^k + \Delta t/2\, g \end{pmatrix}$$

$$F(u^{k+1/2}) = \begin{pmatrix} v^k + \frac{\Delta t}{2} g \\ g \end{pmatrix}$$

# Runge Kutta: Free fall




$$x^{k+1} = x^k + \Delta t\, v^k + \frac{(\Delta t)^2}{2} g$$
$$v^{k+1} = v^k + \Delta t g$$

$$\begin{cases} x^{k+2} = 2x^{k+1} - x^k + (\Delta t)^2 g \\ x^0 = x_0 \\ x^1 = x_0 + \Delta t\, v_0 + \frac{(\Delta t)^2}{2} g \end{cases} \Rightarrow \boxed{x^k = x_0 + (k\,\Delta t)\, v_0 + \frac{(k\Delta t)^2}{2} g}$$

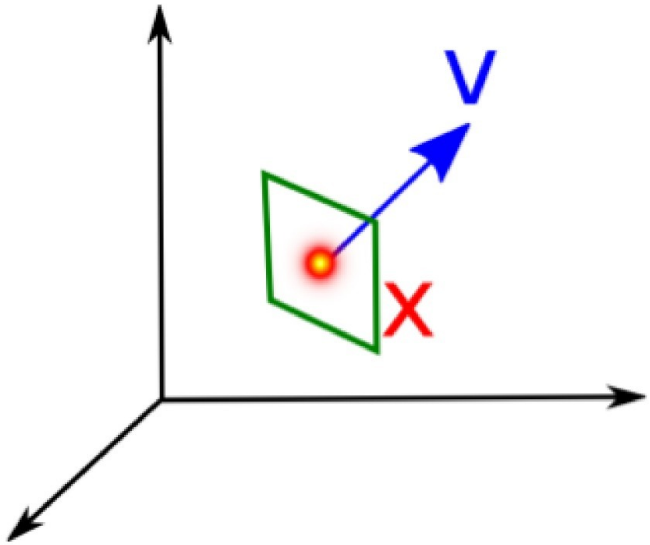
# Runge Kutta: ode45

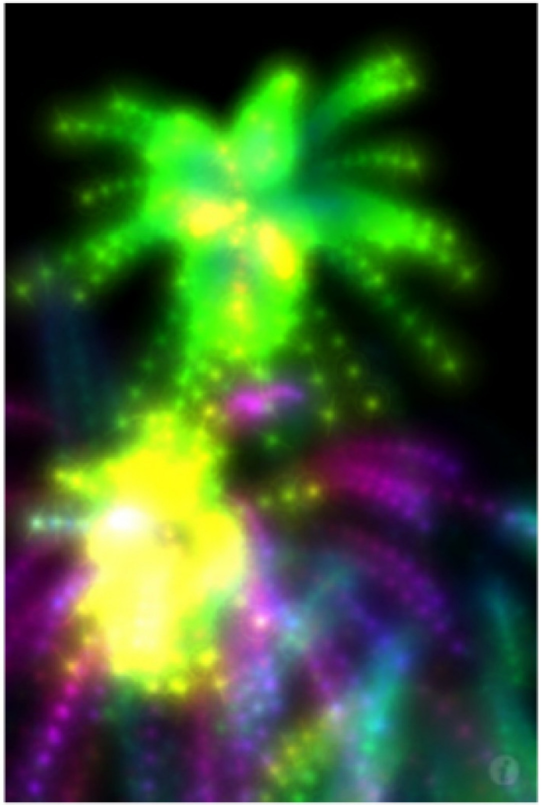
$$
\begin{aligned}
k_1 &= F(t_n, u_n)\\
k_2 &= F\left(t_n + \tfrac{1}{5}\Delta t, u_n + \tfrac{1}{5}\Delta t\ k_1\right)\\
k_3 &= F\left(t_n + \tfrac{3}{10}\Delta t, u_n + \left(\tfrac{3}{40}k_1 + \tfrac{9}{40}k_2\right)\Delta t\right)\\
k_4 &= F\left(t_n + \tfrac{3}{5}\Delta t, u_n + \left(\tfrac{3}{10}k_1 - \tfrac{9}{10}k_2 + \tfrac{6}{5}k_3\right)\Delta t\right)\\
k_5 &= F\left(t_n + \Delta t, u_n + \left(-\tfrac{11}{54}k_1 + \tfrac{5}{2}k_2 - \tfrac{70}{27}k_3 + \tfrac{35}{27}k_4\right)\right)\\
k_6 &= F\Big(t_n + \tfrac{7}{8}\Delta t,\\
&\quad u_n + \left(\tfrac{1631}{55296}k_1 + \tfrac{175}{512}k_2 - \tfrac{575}{13824}k_3 + \tfrac{44275}{110592}k_4 + \tfrac{253}{4096}k_5\right)\Delta t\Big)
\end{aligned}
$$

$$
\begin{aligned}
u^4_{n+1} &= u_n + \Delta t\left(\tfrac{2825}{27648}k_1 + \tfrac{18575}{48384}k_3 + \tfrac{13525}{55296}k_4 + \tfrac{277}{14336}k_5 + \tfrac{1}{4}k_6\right)\\
u^5_{n+1} &= u_n + \Delta t\left(\tfrac{37}{378}k_1 + \tfrac{250}{621}k_3 + \tfrac{125}{594}k_4 + \tfrac{512}{1771}k_6\right)
\end{aligned}
$$

# Application to particles systems

**Sprites**:
Particles falling under gravity
Limited life time
Fill with animated transparent texture

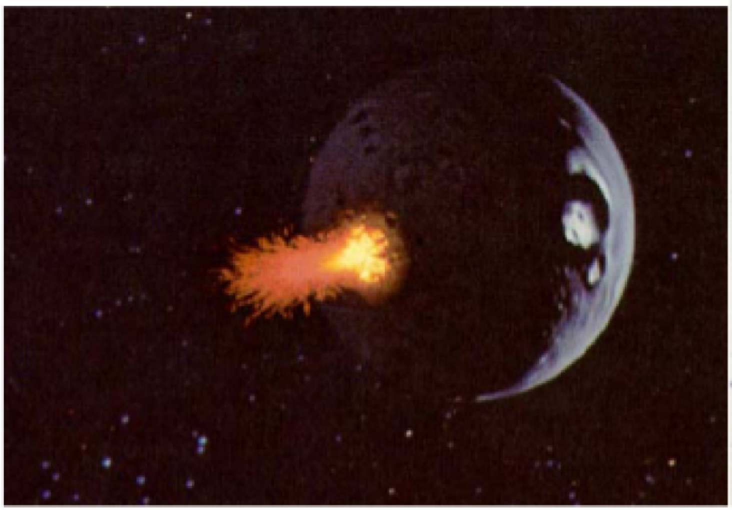


# Application to particles systems

## Drawing trajectories

[William T. Reeves. **Particle Systems. A Technique for Modeling a Class of Fuzzy Objects**. *ACM Transaction on Graphics, 17(3)*. 1983]



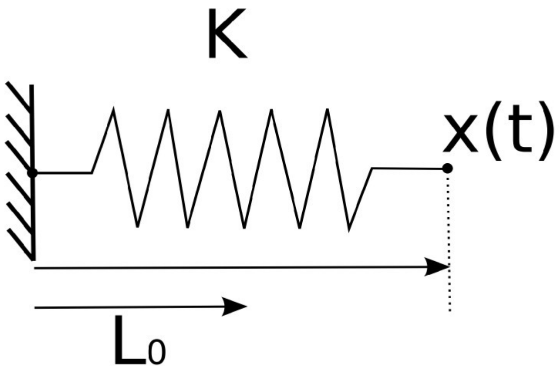
©Lucasfilm, Star Trek II

[Reeves, TOG 83]

# Spring Mass System

Spring force

$$F(t) = K(L_0 - x(t))$$




Equation of motion

$$x''(t) = K/m(L_0 - x(t))$$

ODE formulation

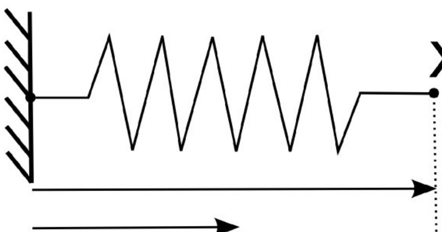
$$u'(t) = \mathrm{A}u(t) + \mathrm{B}$$

$$\mathrm{A} = \begin{pmatrix} 0 & 1 \\ -K/m & 0 \end{pmatrix} \quad \mathrm{b} = \begin{pmatrix} 0 \\ L_0\, K/m \end{pmatrix}$$

# Spring Mass System

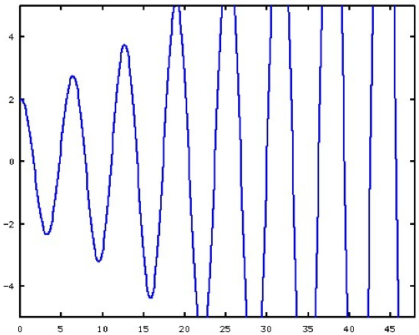
Spring force

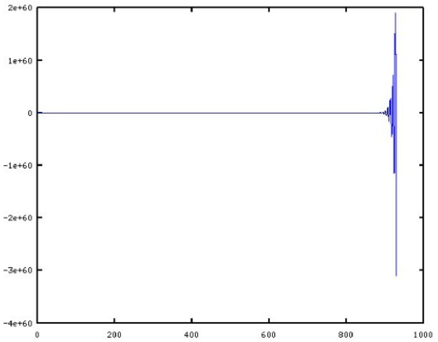
$$F(t) = K(L_0 - x(t))$$




Explicit Euler:

$$x^{k+2} = 2x^{k+1} - \left(1 + (\Delta t)^2 \frac{K}{m}\right) x^k + (\Delta t)^2 \frac{K}{m} L_0$$



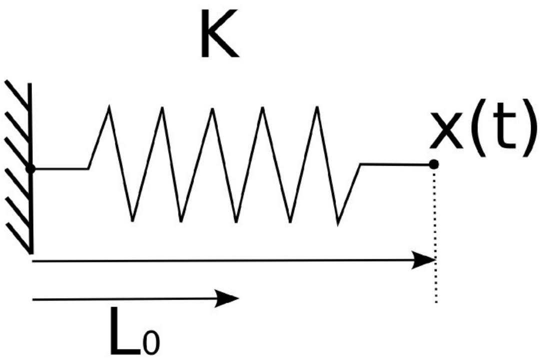

Expect:

$$x(t) = A\sin(\omega t + \varphi)$$

# Spring Mass System

Spring force

$$F(t) = K(L_0 - x(t))$$




Explicit Euler:

$$x^{k+2} = 2x^{k+1} - \left(1 + (\Delta t)^2 \frac{K}{m}\right) x^k + (\Delta t)^2 \frac{K}{m} L_0$$

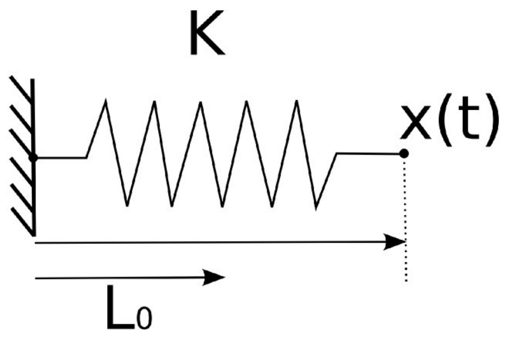
Do not diverge if $1 + \sqrt{(\Delta t)^2 \frac{K}{m})} < 1$

=> Always diverge to infinity !

# Spring Mass System

Spring force

$$F(t) = K(L_0 - x(t))$$




Add fluid damping

$$F_d(t) = -\mu\, v(t)$$

New equation for explicit Euler:

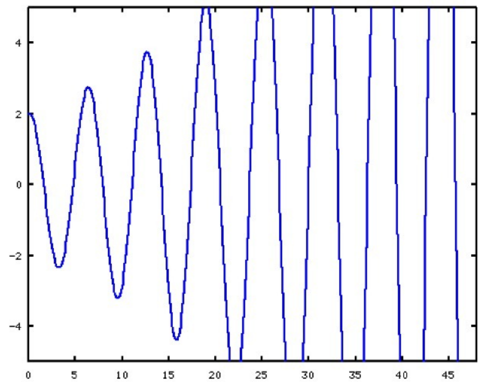
$$x^{k+2} - \left(2 - \frac{\mu}{m}\Delta t\right) x^{k+1} + \left(1 + (\Delta t)^2 \frac{K}{m} - \frac{\mu}{m}\Delta t\right) x^k = (\Delta t)^2 \frac{K}{m} L_0$$

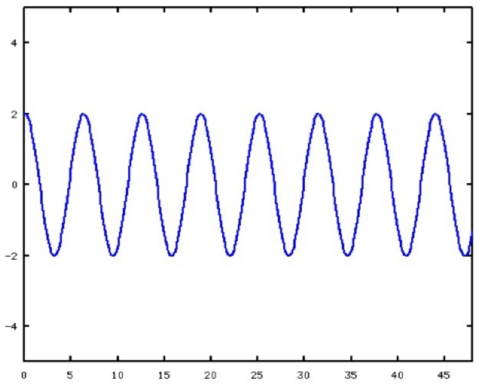
## Conditionnaly stable

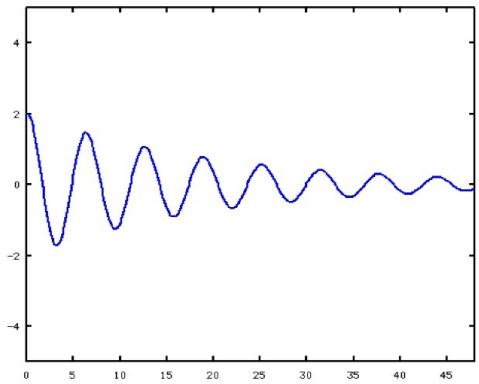
Large K => Stiff springs

**Stiff ODE**

# Accuracy != Stability



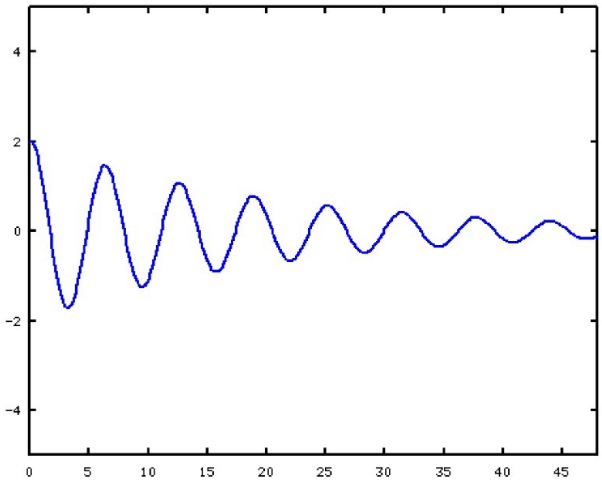



# Implicit Euler

$$\underbrace{\begin{pmatrix} 1 & -\Delta t \\ -\Delta t \frac{K}{m} & 1 \end{pmatrix}}_{\text{M}} u^{k+1} = u^k + \Delta t \begin{pmatrix} 0 \\ K \frac{L_0}{m} \end{pmatrix}$$

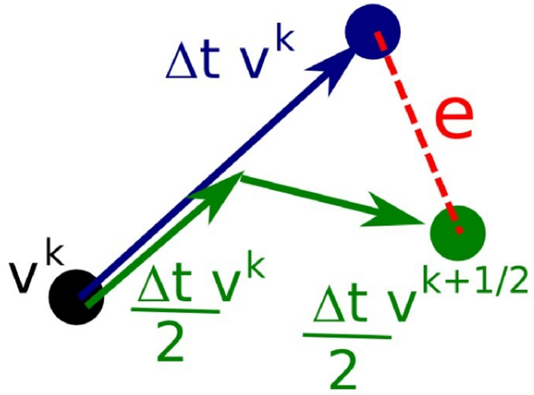
Eigenvalues of $M^{-1}$ $\quad 1 - \Delta t \sqrt{\frac{K}{m}}$ <1

Unconditionally stable



# Automatic step-size

Compute:



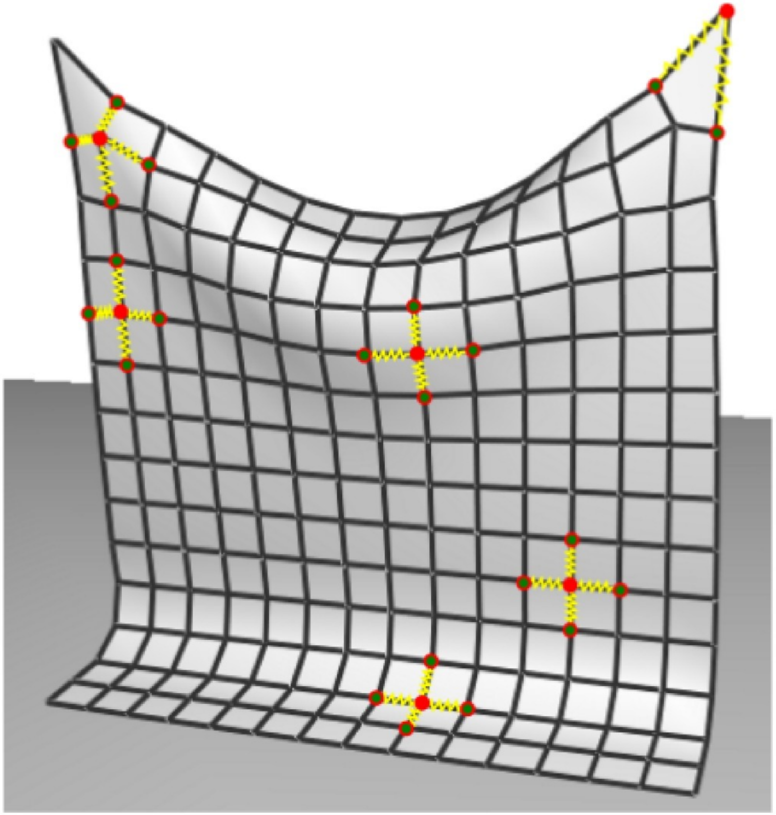

- $u_1^{k+1} = u^k + \Delta t\ F(u^k)$

- $\left|\begin{array}{l} u_2^{k+1/2} = u^k + \Delta t/2\ F(u^k) \\ u_2^{k+1} = u_2^{k+1/2} + \Delta t/2\ F(u_2^{k+1/2}) \end{array}\right.$

- $e = \|u_1^{k+1} - u_2^{k+1}\|$

$$\left|\begin{array}{l} e < K_{\mathrm{max}} \Rightarrow (\Delta t)_{\mathrm{new}} = \Delta t/2 \\ e < K_{\mathrm{min}} \Rightarrow (\Delta t)_{\mathrm{new}} = 2\Delta t \end{array}\right.$$

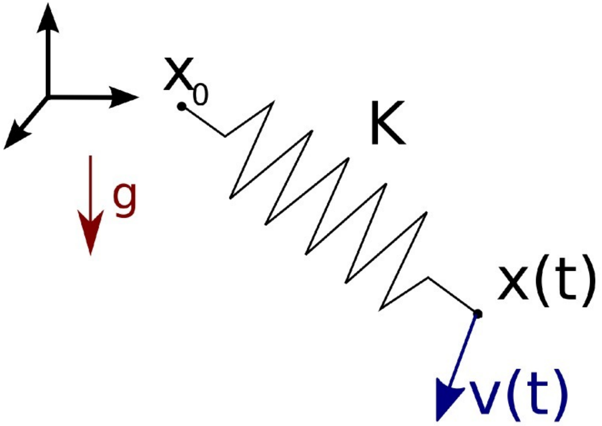
# Cloth Simulation

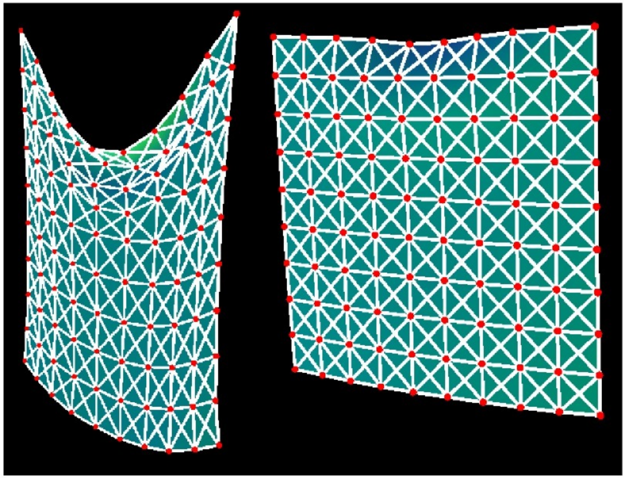


# Cloth Simulation

In 3D:

$$F(t) = K\left(L_0 - \|\mathbf{p} - \mathbf{p}_0\|\right)\frac{\mathbf{p} - \mathbf{p}_0}{\|\mathbf{p} - \mathbf{p}_0\|}$$



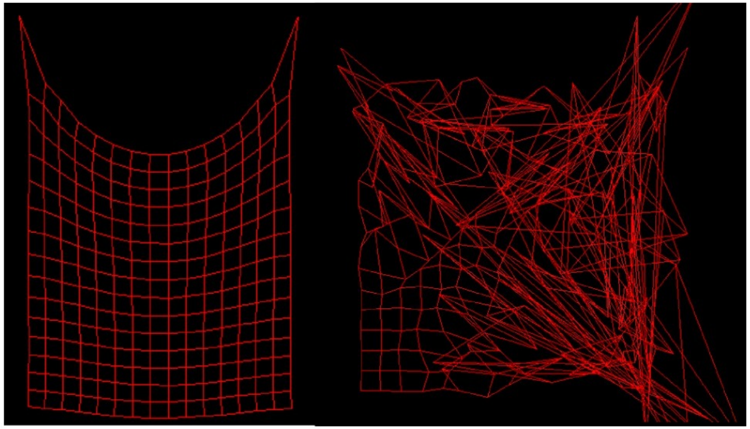
# Cloth Simulation: mass springs

Model cloth using coupled springs
Force on a vertex i with neighbors $\mathcal{V}_i$

$$F(x_i, t) = \sum_{j \in \mathcal{V}_i} K^{ij} \left( L_0^{ij} - \|x_i - x_j\| \right) \frac{x_i - x_j}{\|x_i - x_j\|} + g$$

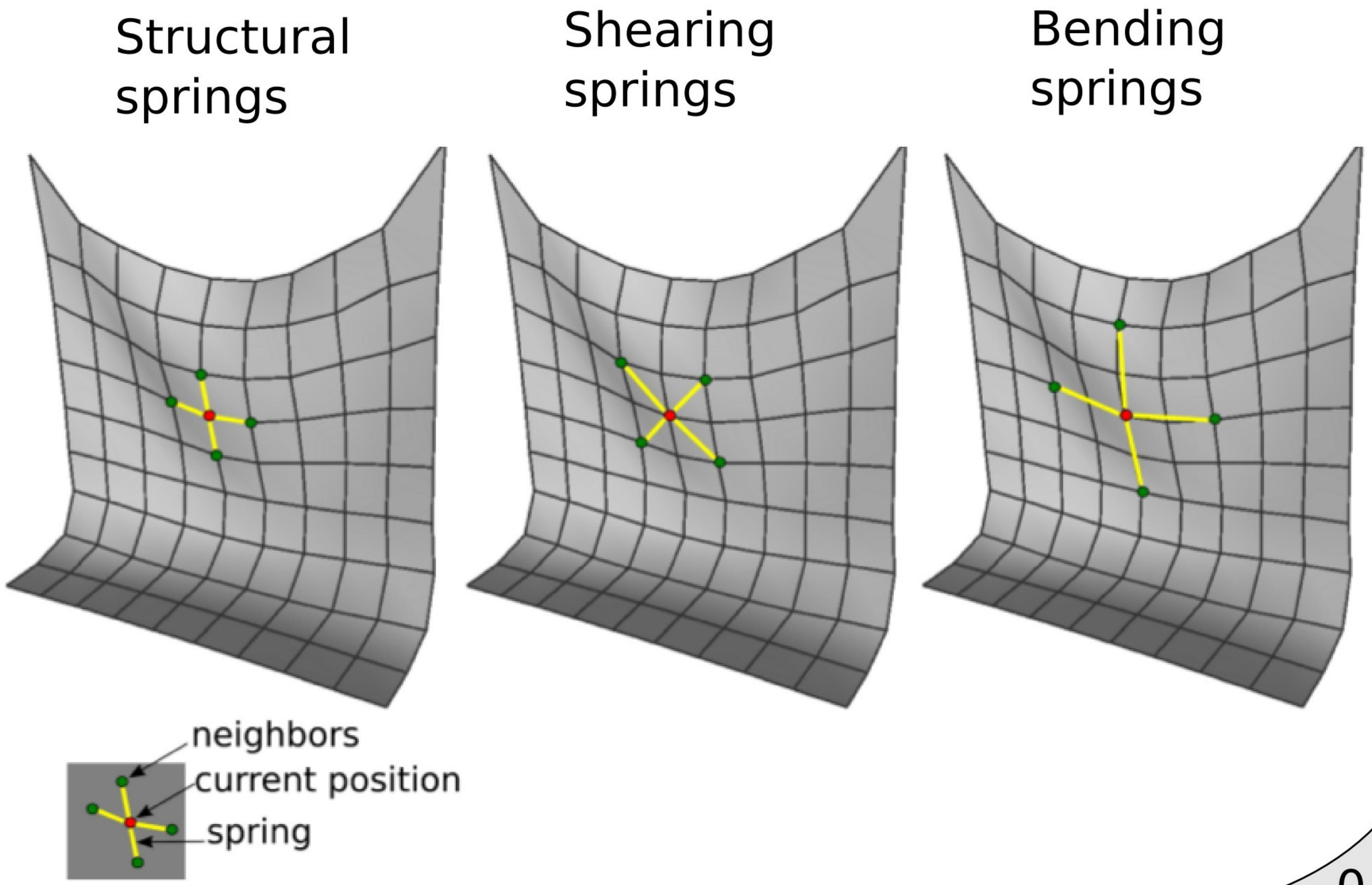
$$\forall i \;, \begin{cases} x_i'(t) = v_i(t) \\ v_i'(t) = \frac{1}{m_i} \sum_j K^{ij} \left( L_0^{ij} - \|x_i - x_j\| \right) \frac{x_i - x_j}{\|x_i - x_j\|} + g \end{cases}$$

Use you best interpolation scheme



[P. Jacobs]

# Cloth Simulation: spring types

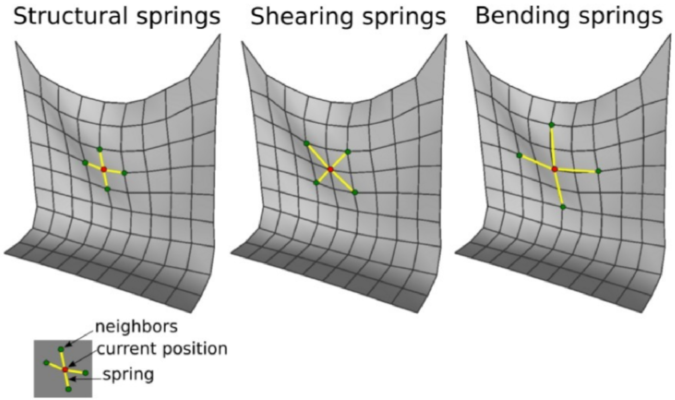


# Cloth Simulation: Resolution

## Explicit Euler

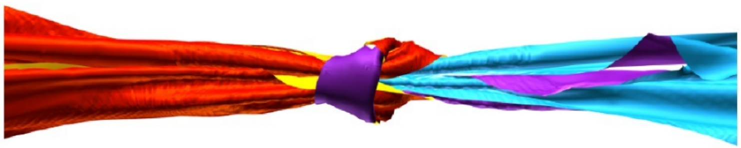
```
//Compute forces
For all i
   For j:  4 direct neighbors (structural): K=K1
           4 diagonal neighbors (shear)    : K=K2
           8 neighbors (bend)              : K=K3
      u=p[i]-p[j]
      F[i] += K (L0-norm(u))*u/norm(u)

For all i
   v[i] += dt*F[i]
   p[i] += dt*v[i]
```



# Complex cloth simulation

Full simulation = Cloth + complex collisions

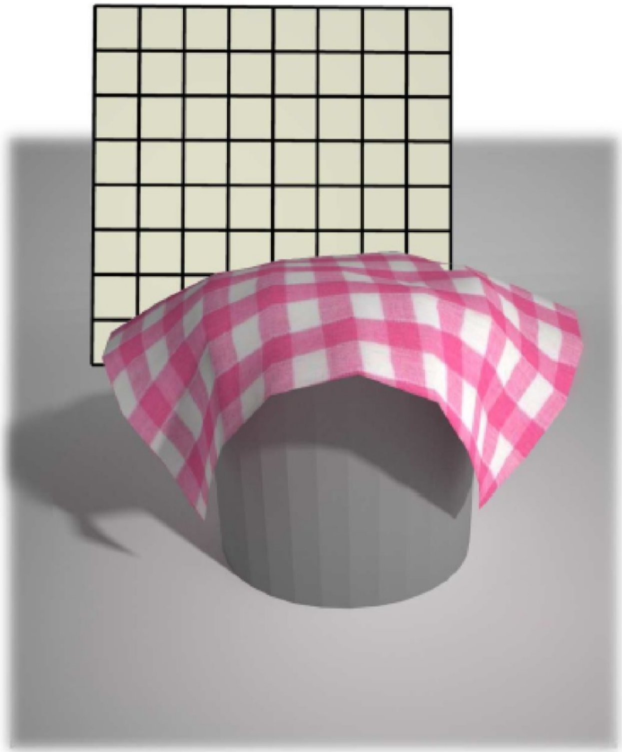


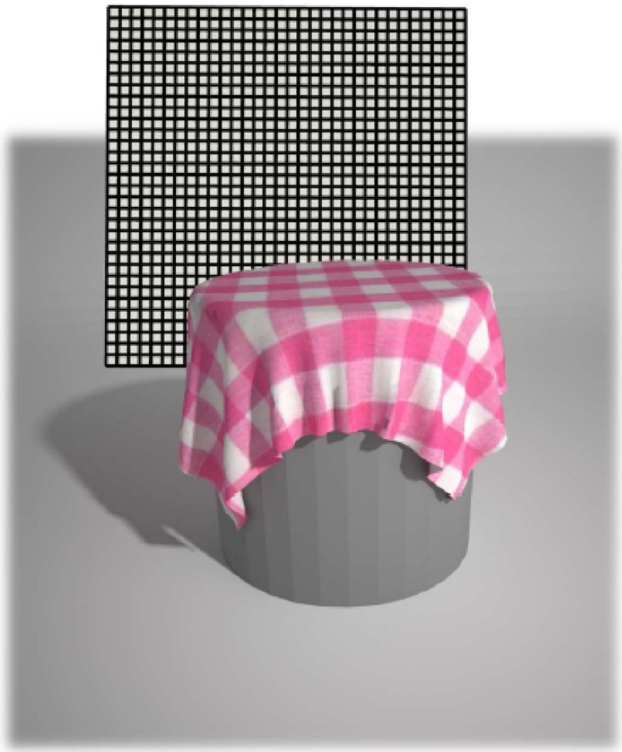
*[Grinspun, SIGGRAPH 09]*

*[DAZ3D, Dynamic Clothing]*

# Cloth simulation: Limitations

Mesh influence





# Implicit scheme

Write the system in a vectorial form

$$u(t) = (p_0(t), p_1(t), \ldots, p_{N-1}(t), v_0(t), \ldots, v_{N-1}(t))$$

$$u'(t) = \mathcal{F}(u(t))$$

Non linear system with N unknown: impossible to invert
Linearize the system in $p(t) - p_0 = \Delta L\, v(t) + \mathcal{O}\left((\Delta t)^2\right)$

Get a linear system.
Loss of the inconditional stability:
In practice, very stable

# Implicit scheme

Linearization:

$$\mathbf{f}(\mathbf{p}_n + \Delta p, \mathbf{v}_n + \Delta v) = \mathbf{f}_n + \frac{\partial \mathbf{f}}{\partial \mathbf{p}} \Delta p + \frac{\partial \mathbf{f}}{\partial \mathbf{v}} \Delta v$$

Solve a linear system at each time step

$$\mathrm{A}\, \Delta v = \mathbf{b}$$

$$\mathrm{A} = \mathrm{I} - \Delta t\, \mathrm{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{v}} - \Delta t^2\, \mathrm{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{p}}$$

$$\mathbf{b} = \Delta t\, \mathrm{M}^{-1} \left( \mathbf{f}_n + \Delta t\, \frac{\partial \mathbf{f}}{\partial \mathbf{p}} \mathbf{v}_n \right)$$

# Collisions

# Particles collisions

Particles can enter in collision with a plane:

$$\mathcal{P} :< \mathbf{x} - \mathbf{p}_0, \mathbf{n} >= 0$$

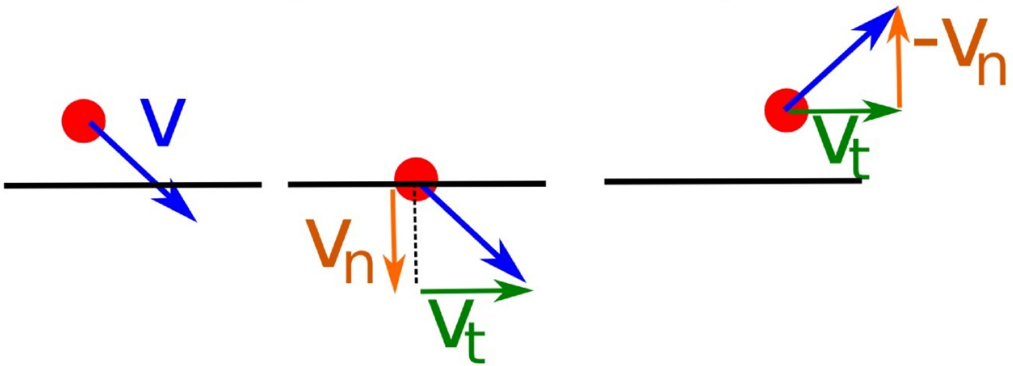
Detection $< \mathbf{x} - \mathbf{p}_0, \mathbf{n} > < 0$

Separate tangential speed $v_t$ and and normal speed $v_n$

$$\begin{cases} v_n =< v, n > n \\ v_t = v- < v, n > n \end{cases}$$

After collision, loss of energy: dumping

$$v^{k+1} = -\mu < v^k, n > n + (v^k - < v^k, n > n)$$



# Particles collisions

Need to take care of the temporal discretization

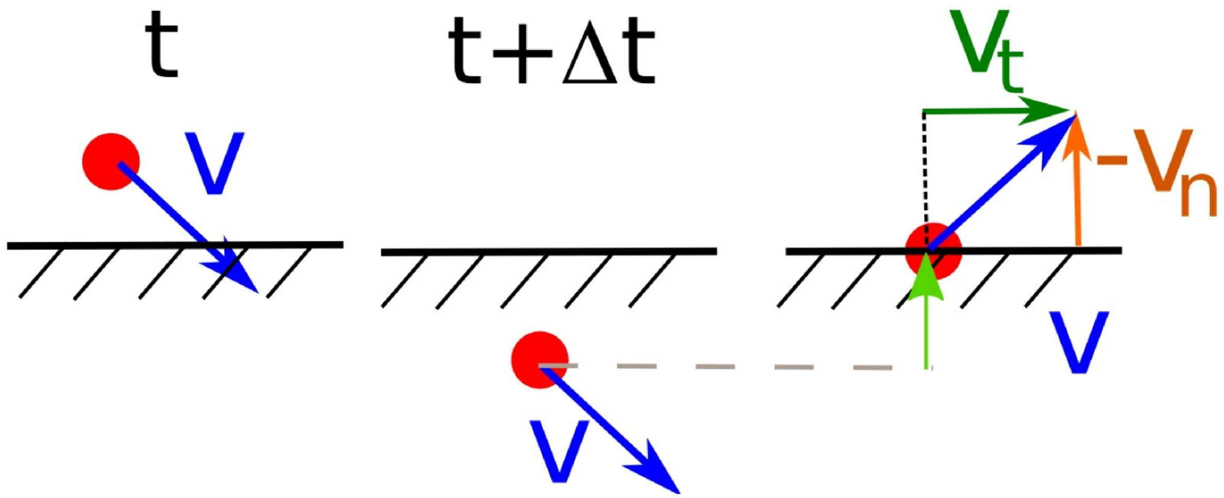
1. Projection of the particle

Easy to compute, physically biased (instabilities)

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} - < \mathbf{x}^{k} - \mathbf{p}_0, \mathbf{n} > \ \mathbf{n}$$

2. Backward search

Less easy, less wrong

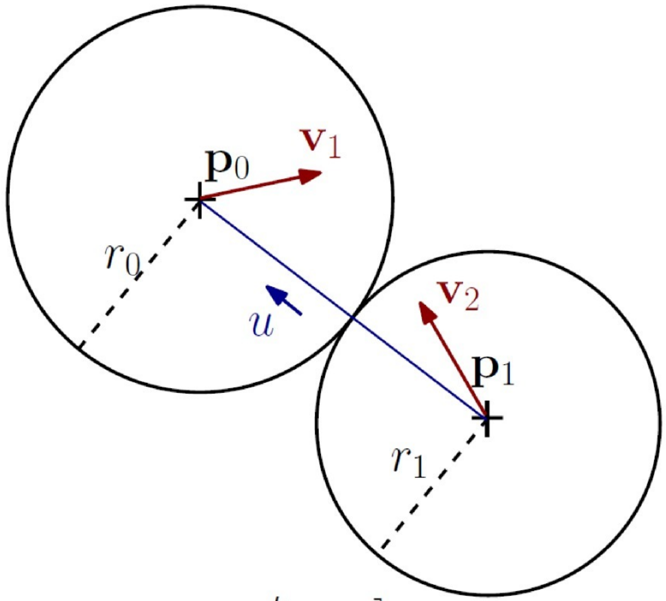


# Model of hard spheres

Particles => Sphere of mass m, center x, radius r

Collision if

$$\|x_1 - x_2\| < r_1 + r_2$$



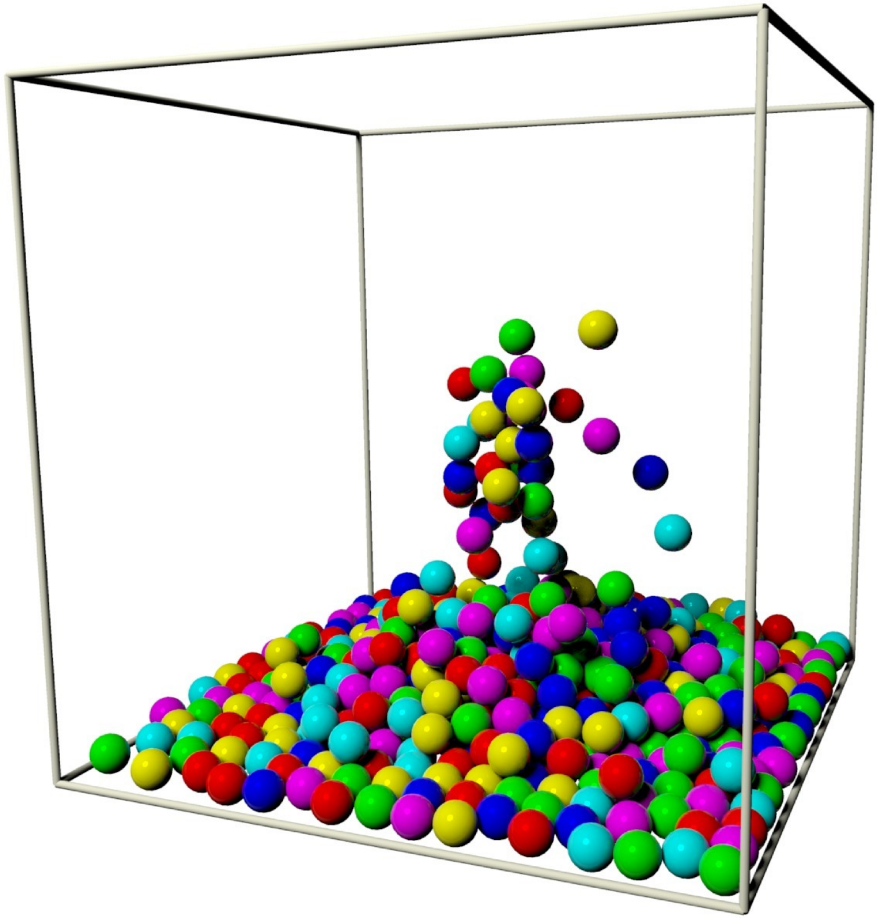

New speed after collision

$$\begin{cases} v_1^{k+1} = v_1^k + \frac{1}{m_1+m_2}\left[m2 < v2^k, u > -\frac{1}{2}(m_1+3m_2) < v1^k, u >\right] u \\ v_2^{k+1} = v_2^k + \frac{1}{m_1+m_2}\left[m1 < v1^k, u > -\frac{1}{2}(m_2+3m_1) < v2^k, u >\right] u \\ u = x1 - x0 \end{cases}$$

*Don't forget to project back to the collision surface*

# Model of hard spheres

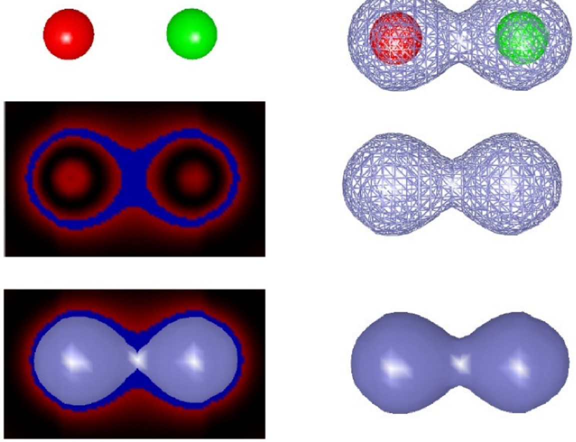
Generate a lot of spheres



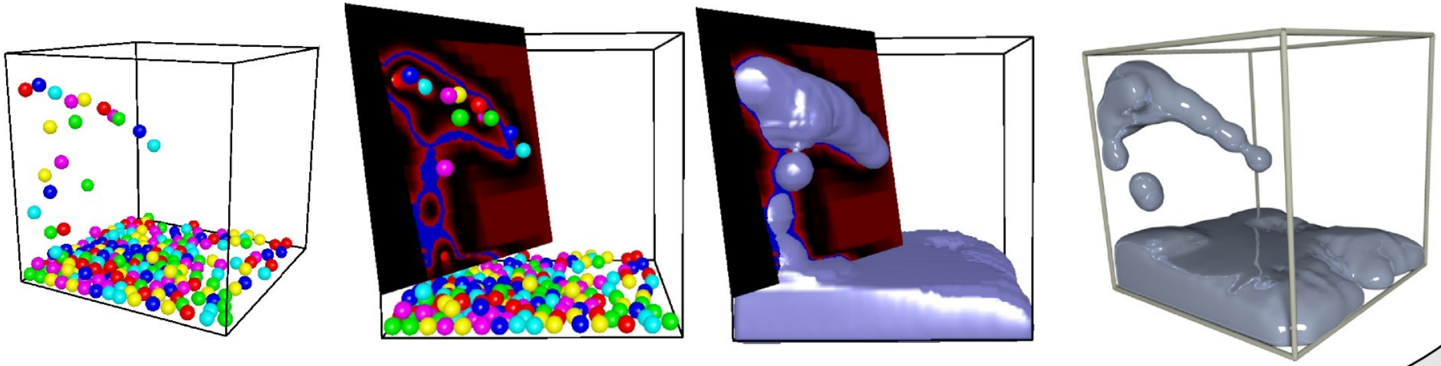
# Sphere: implicit functions

Set a field function to each particle (ex. blobs)

$$\psi_i(\mathbf{p}) = e^{\frac{\|\mathbf{p}-\mathbf{p}_i\|^2}{\sigma^2}}$$



=> Fluid simulation



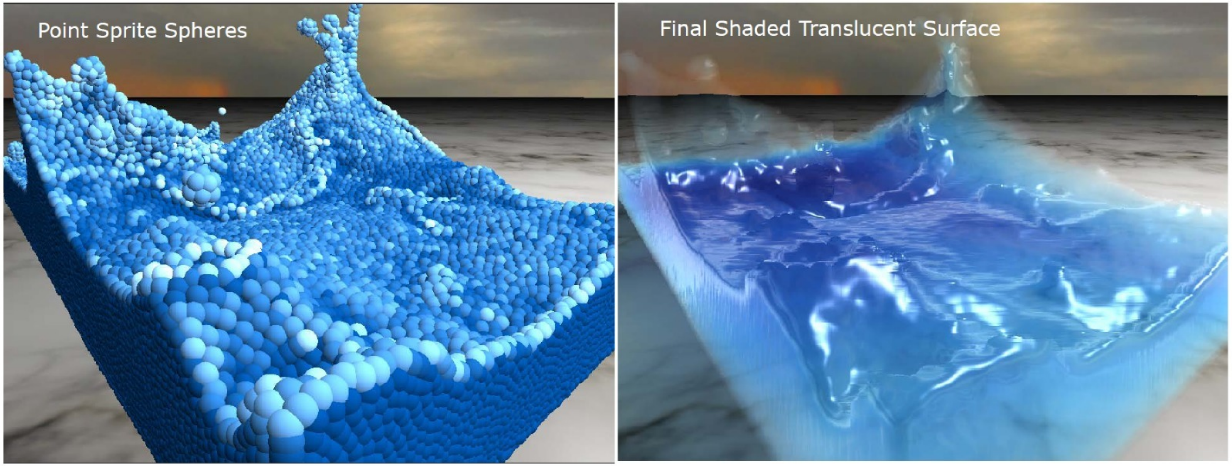
# Efficient collision detection

Brute force algorithm

```
for (i=0;i<N;++i)
  for (j=i+1;j<N;++j)
    if (norm(xi-xj)<r1+r2)
      CollisionResponse(i,j)
```

Complexity in $O(N^2)$

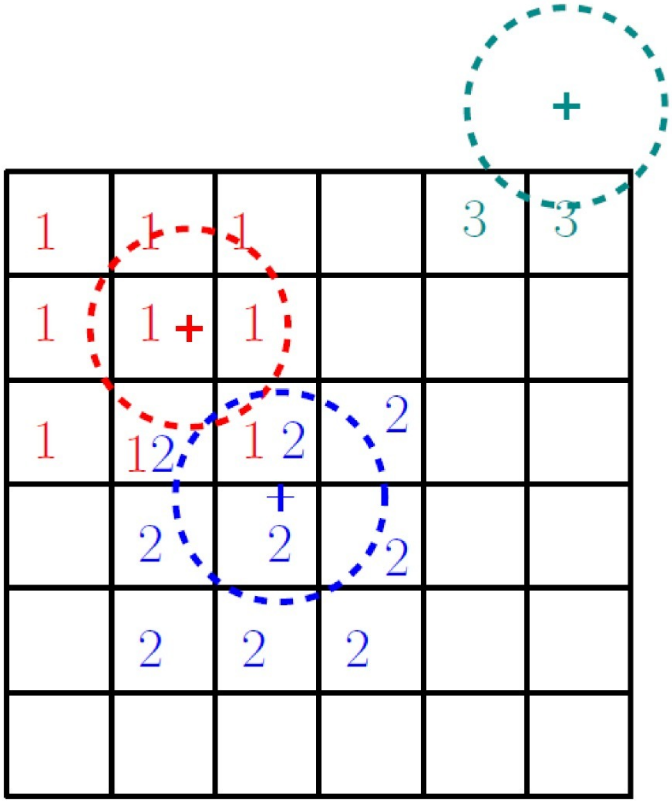
Impossible to simulation $10^{3\text{-}6}$ particules in real time




*[NVIDIA, Green SIGGRAPH 10]*

# Acceleration structure

Regular grid
Research in O(1)

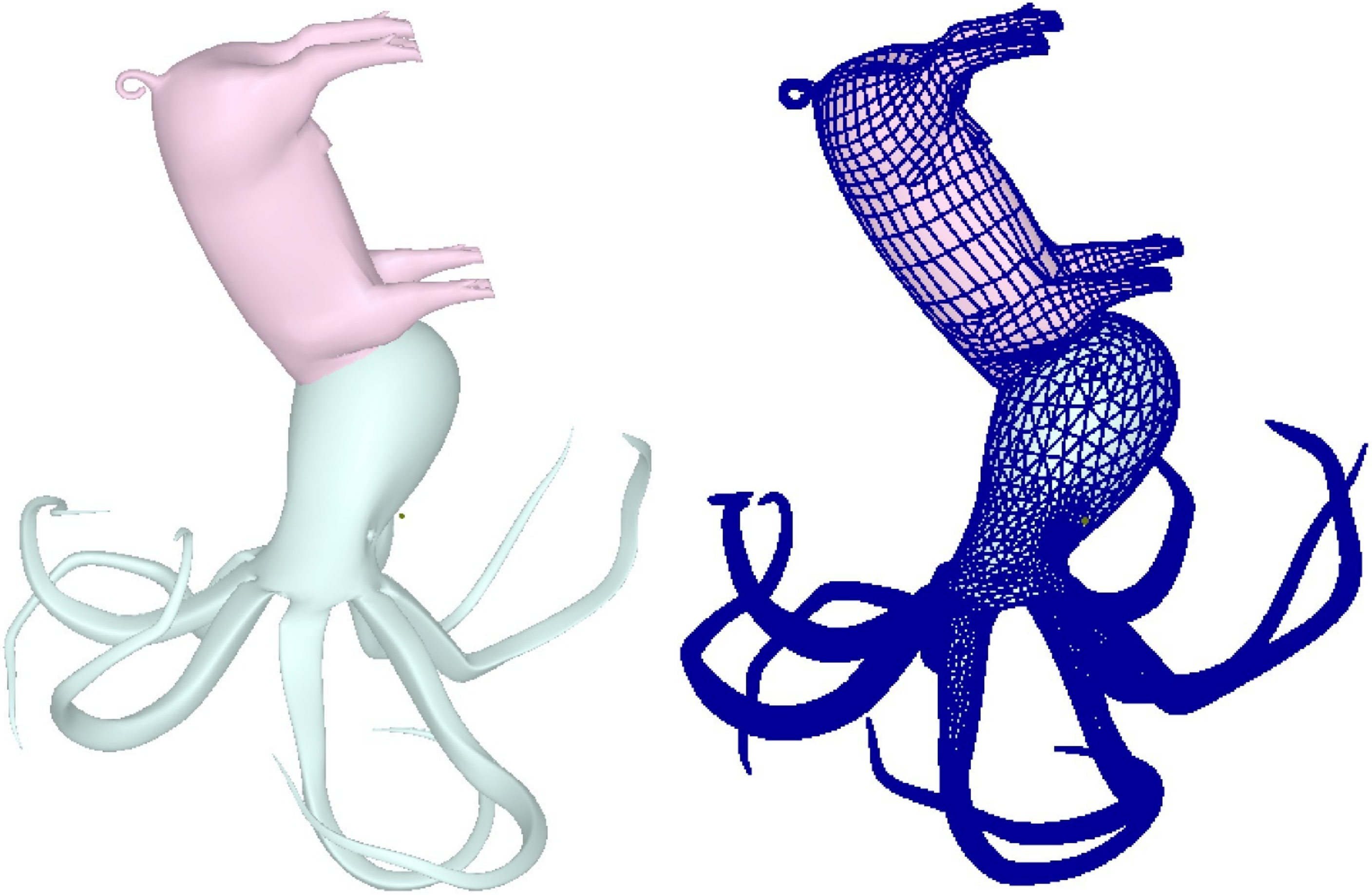



+ Simple
+ Efficient research
+ Good for uniform samples

- Memory consumption
  for empty areas

# Collision detecion between objects

Collision bw triangles/triangles in $O(N_{t1}\ N_{t2})$



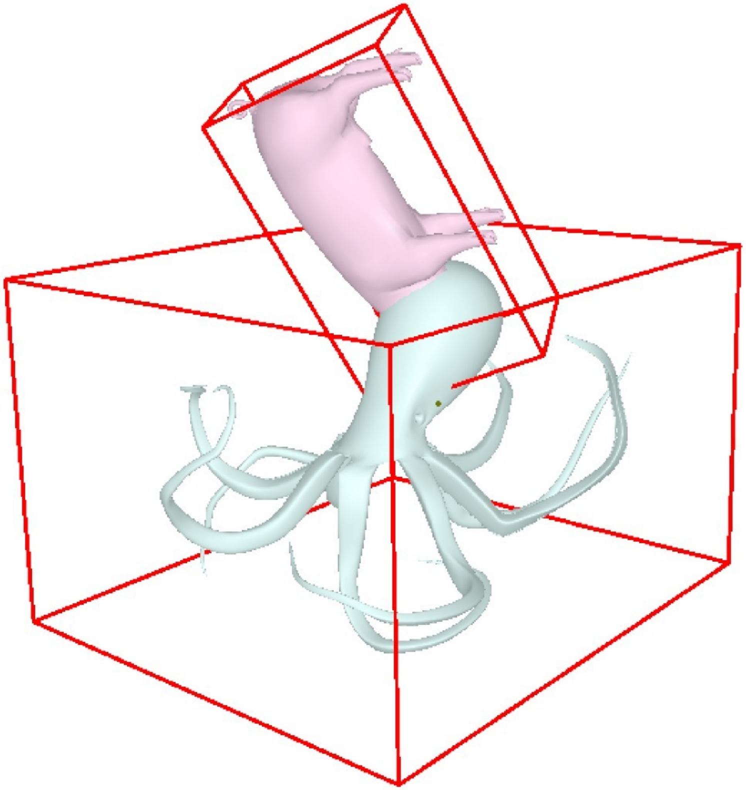
# Bounding boxes

Bounding Box (BB)

The simplest: **AABB**
**A**xis **A**ligned **B**ounding **B**ox

Bounding Spheres

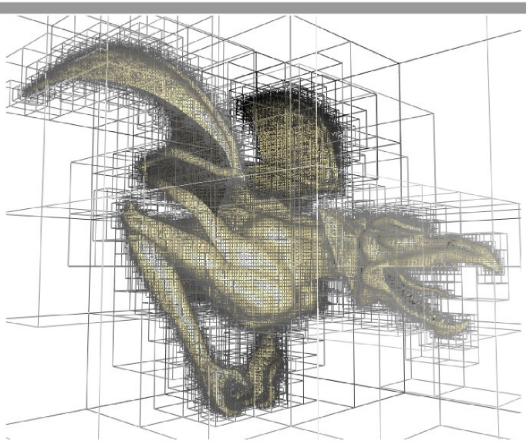
+ Detection of non-collision in $O(N^2_{obj})$



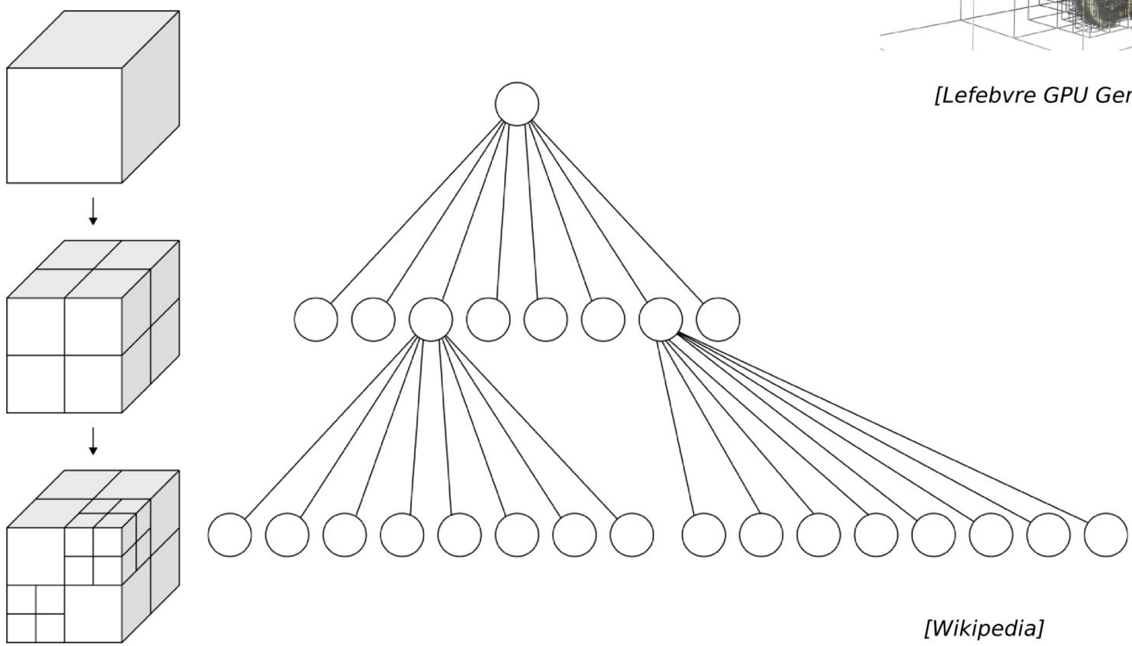
# Octree

Self-adaptable grid structure

+ Can adapt to complex geometry
+ Research in O(log(M)), M=tree depth



*[Lefebvre GPU Gems 2, 2004]*

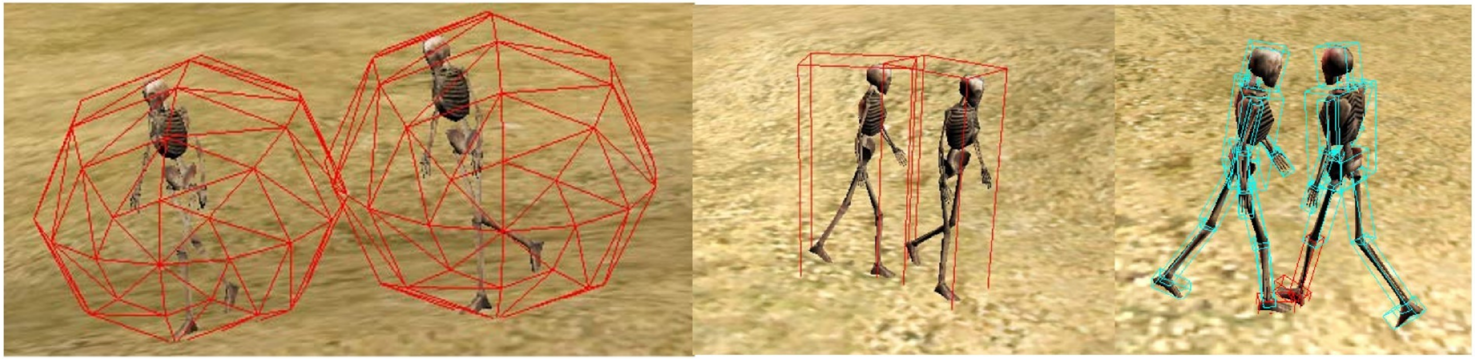


*[Wikipedia]*

# Collisions

## In practice: Several levels of details

ex.
- Spheres within BB within Octree
- OBB within AABB within Spheres



*[Ditchburn]*

Rough test
Less rough test
...
Fine test
...
Eventually: Compute the real intersection (rare)

=> Non collision tests much more efficient than collision