太极图形课

第08讲 Deformable Simulation 01: Spatial and Temporal Discretization

taichi

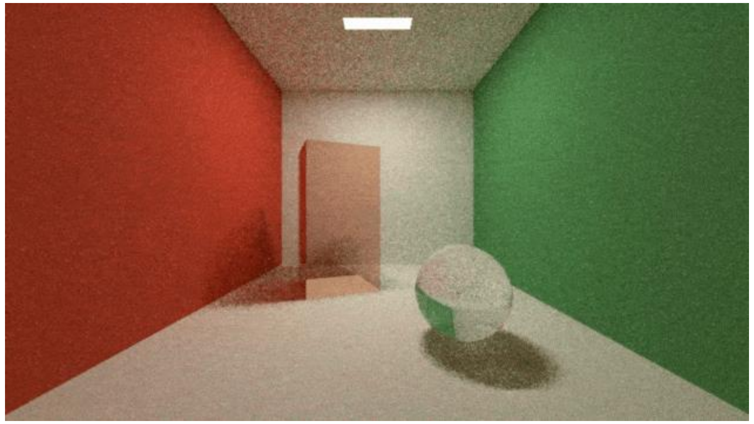
太极图形课

第08讲 Deformable Simulation 01: Spatial and Temporal Discretization

taichi

Previously in this Taichi Graphics Course...

Procedural Animation

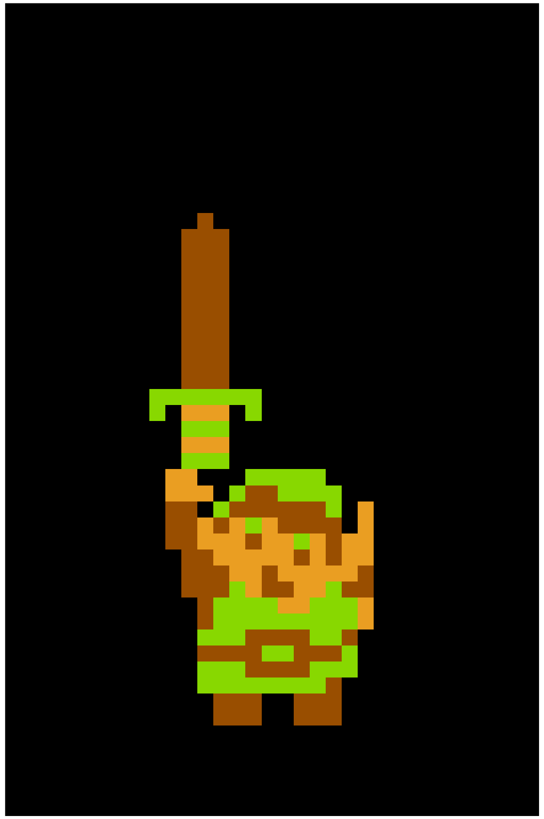


Rendering

Deformable Simulation

Fluid Simulation

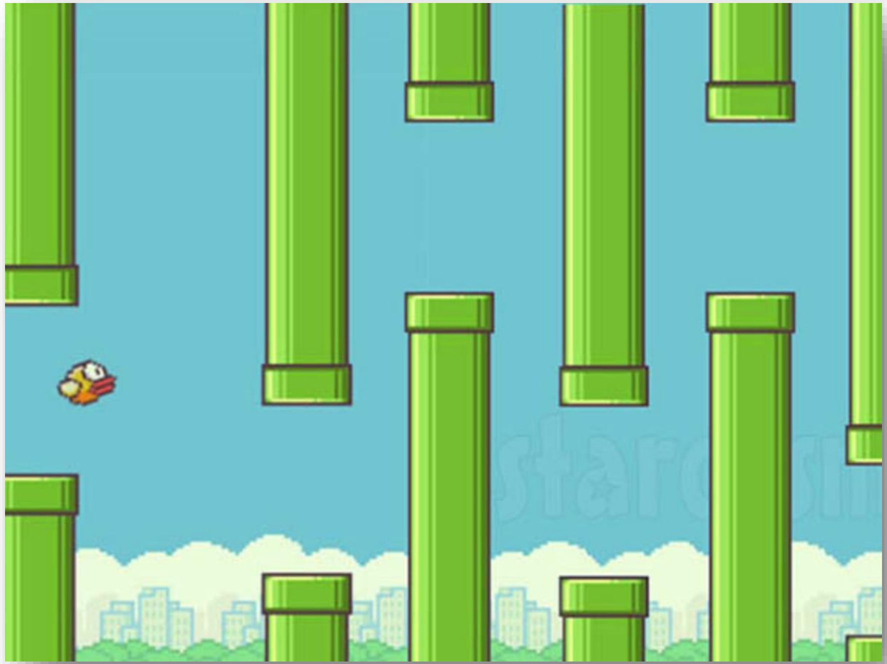
Rendering is a lot of fun...



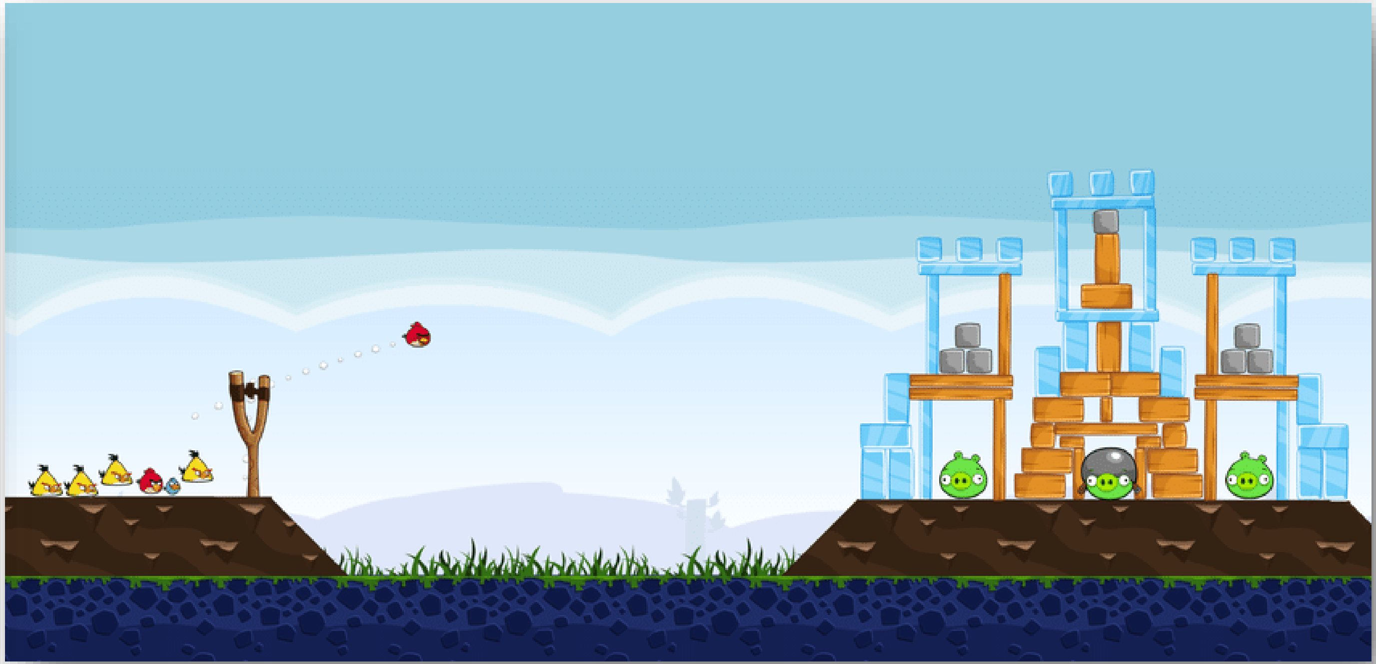
1986

2017

But sometimes we want moving pictures as well



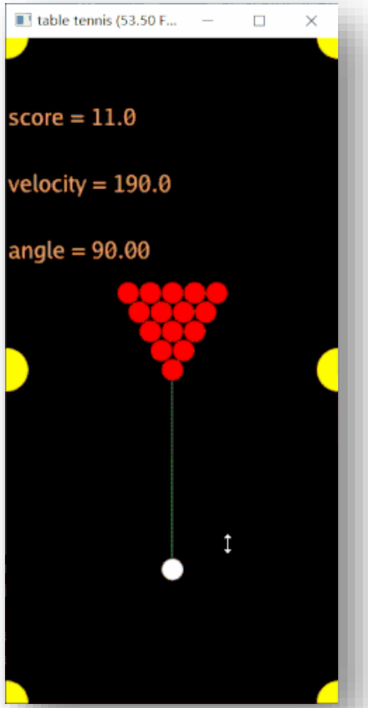
Kinetically-controlled characters



Physically-animated characters

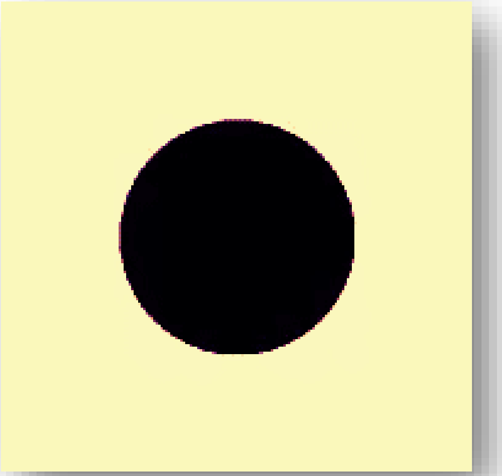
Physically-based animations

- ***Generate*** animated pictures based on ***laws of physics***

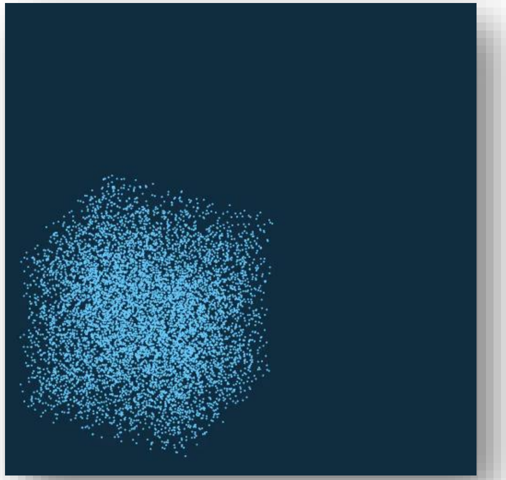


Rigid
@Pierce-qiang

Deformable
@metachow



Compressible Fluid

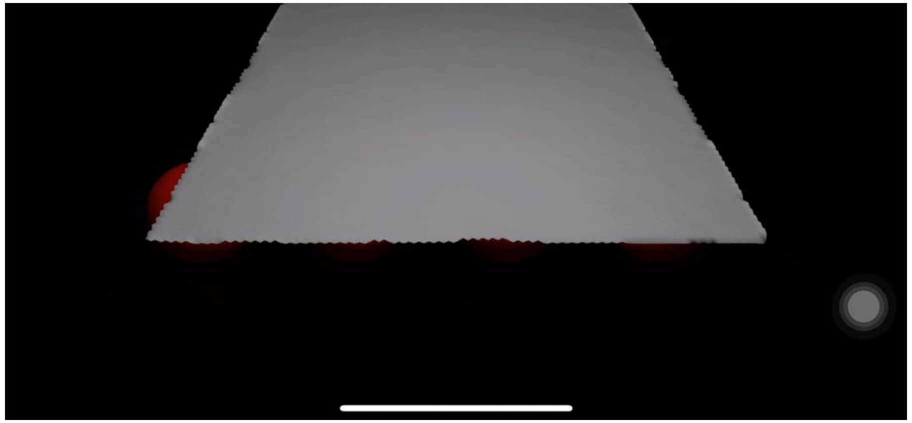


Incompressible Fluid

In the following two classes…

Procedural Animation

Rendering



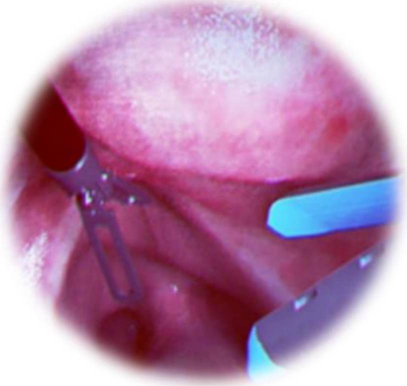
Deformable Simulation

Fluid Simulation

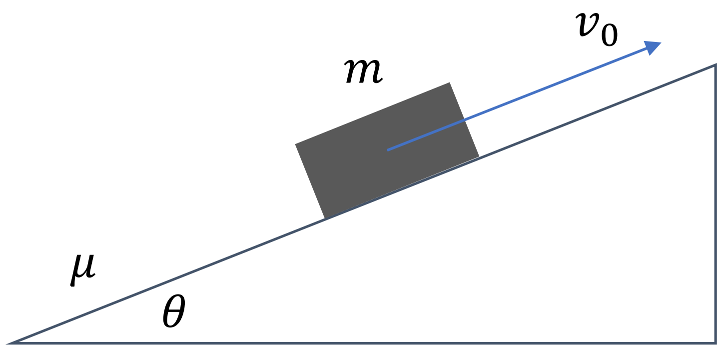
     

Our real lives are surrounded by deformable objects…

… so be our virtual lives

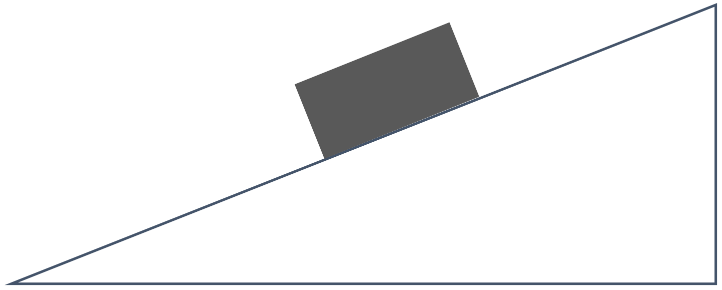
    

Goal of a simulation:
predicting the status of the moving matters at the given time

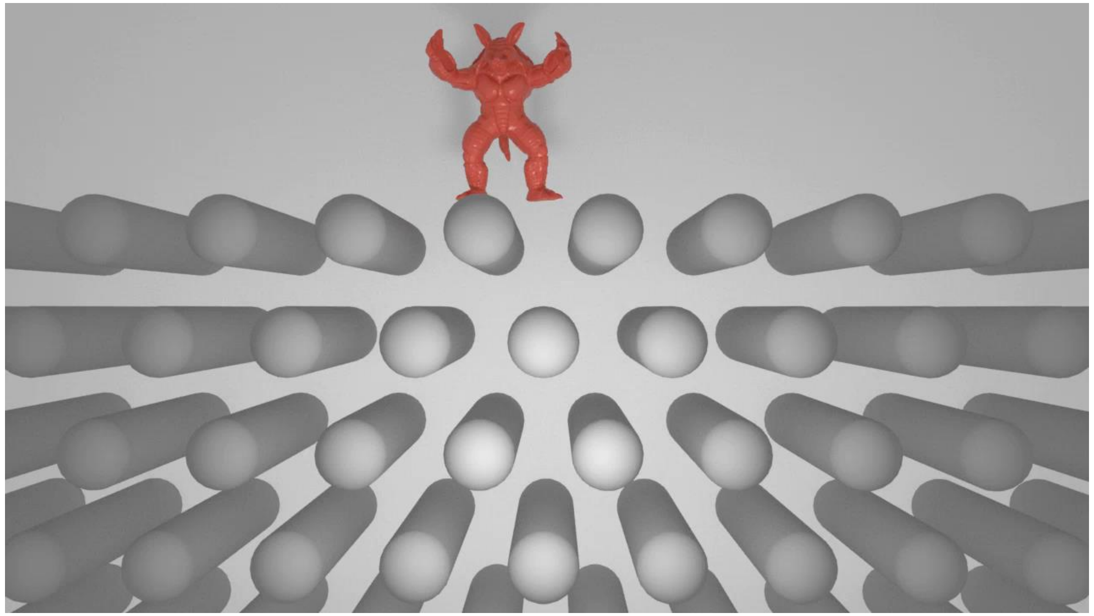


Where is the block at time $t = 1$?
What is the velocity of the block at time $t = 2$?

Outline today:
A practitioner's guide to build your first deformable object simulator

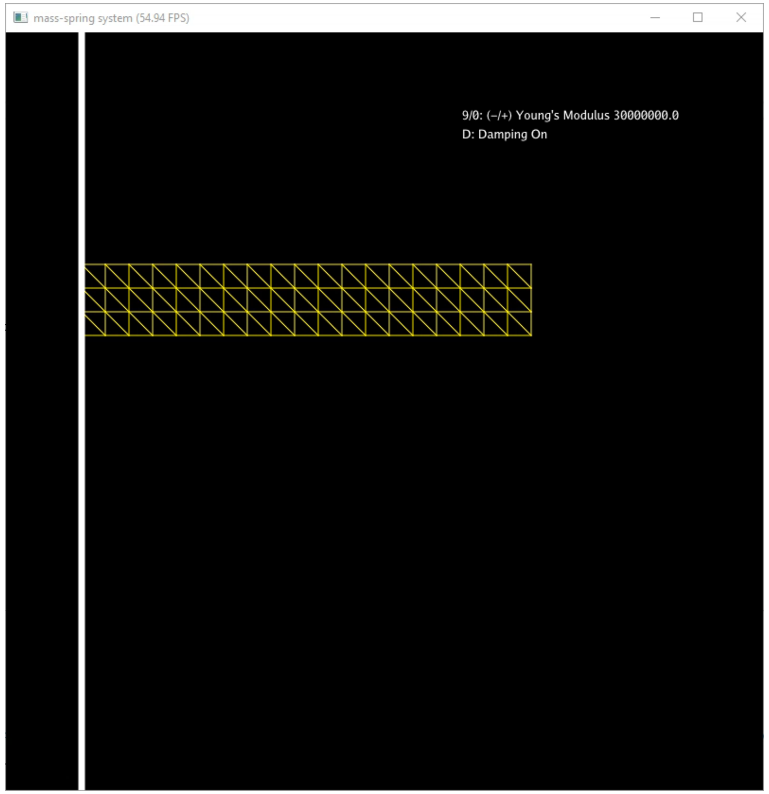


Some details

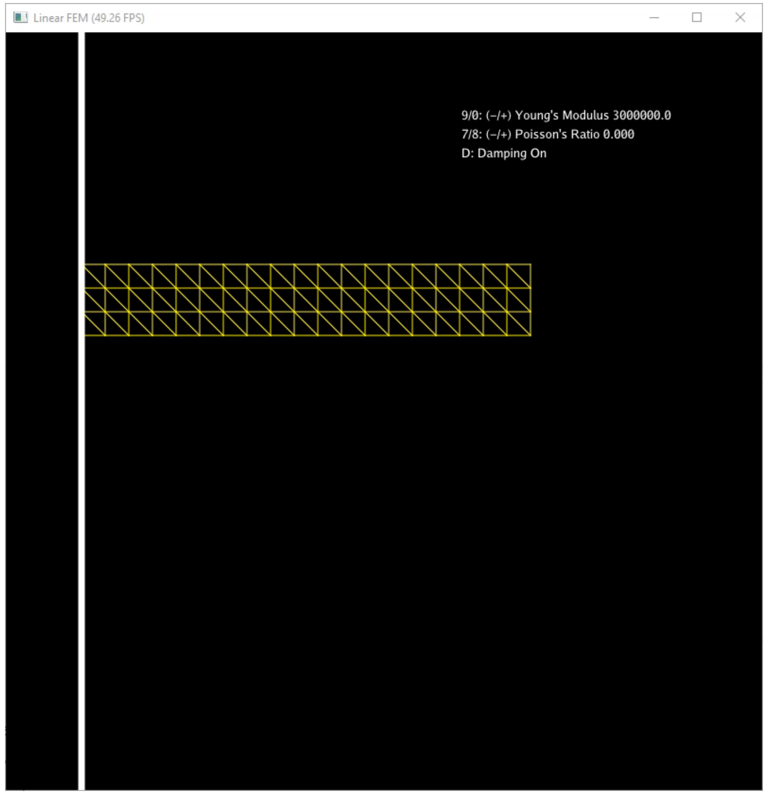


Code of the day

- https://github.com/taichiCourse01/--Deformables



Mass-Spring



Linear FEM

Outline today

- Laws of physics
- Integration in time
- Integration in space
    - A simple (but useful) model: mass-spring system
    - Constitutive models
    - The finite element method

Things NOT covered in today’s class…

- Derivations in continuum mechanics
- Strong form v.s. weak form & basis functions
- Geometric integrators
- Damping / Collisions / Contact

Laws of physics

Equations of motion

- Define $\frac{d}{dt} q \coloneqq \dot{q}$

- We have:
    - $\dot{x} = v$
    - $\dot{v} = a$

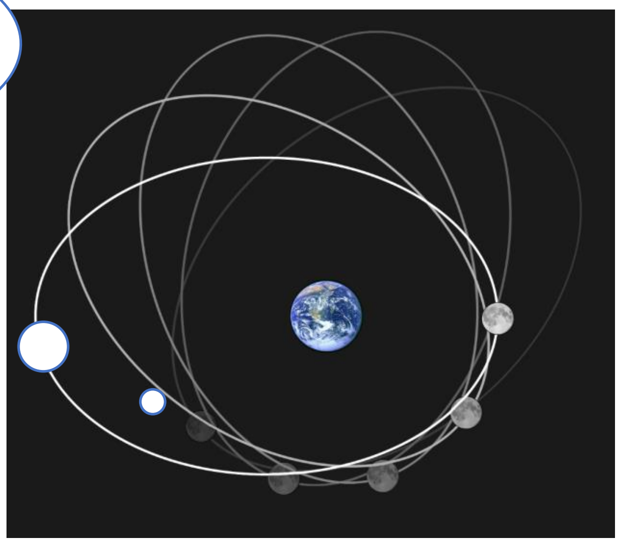
- Or simply:
    - $\ddot{x} = a$

Equations of motion

- Define $\frac{d}{dt}q \coloneqq \dot{q}$

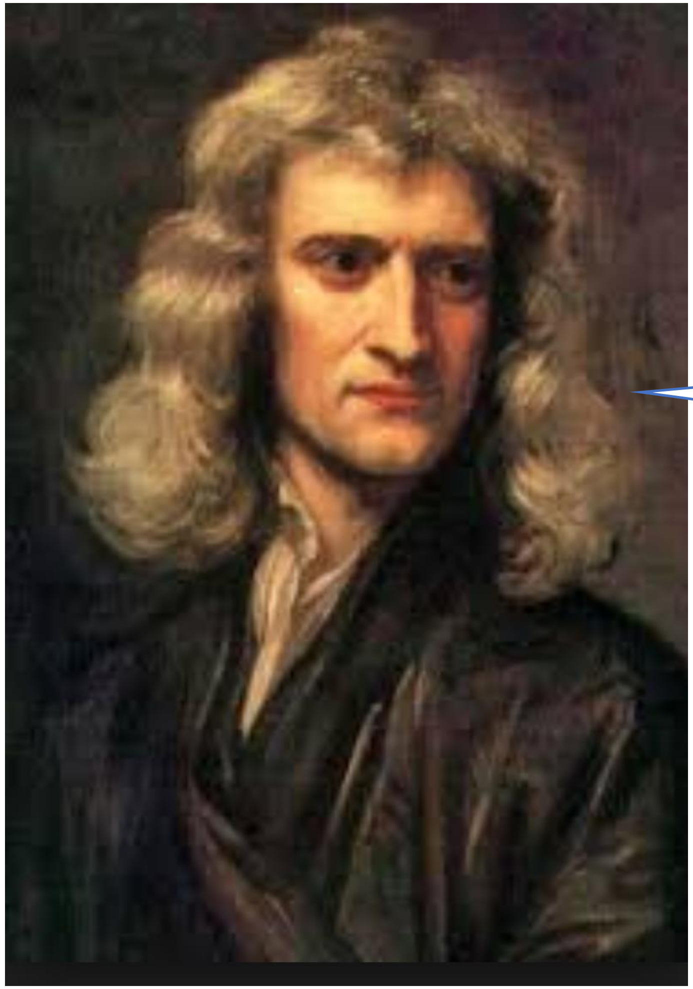
- We have:
  - $\dot{x} = v$
  - $\dot{v} = a$

What is the acceleration of the moon?



- Or simply:
  - $\ddot{x} = a$

Equations of motion

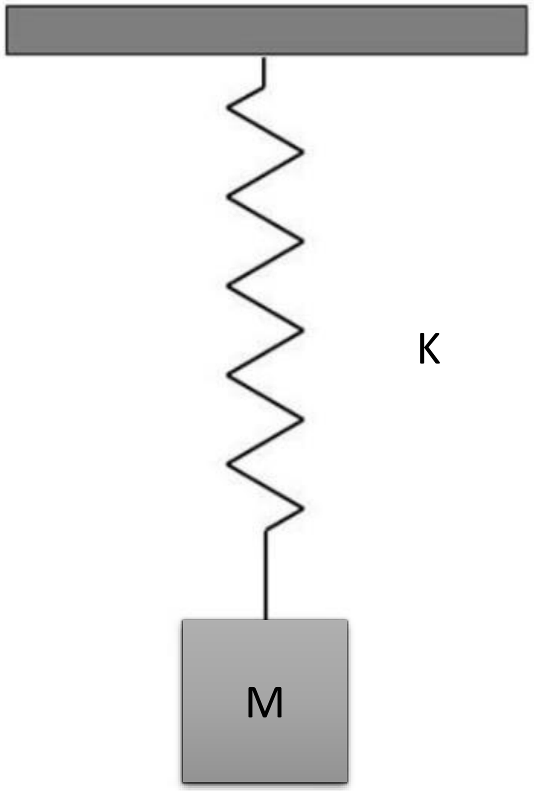


$$\boldsymbol{f} = \mathbf{M}\boldsymbol{a}$$

Equations of motion (linear ODE)

- $M\ddot{x} = f(x)$

- For linear materials, we have $f(x) = -K(x - X)$
  - X: Rest-pose position
  - x: Current-pose position

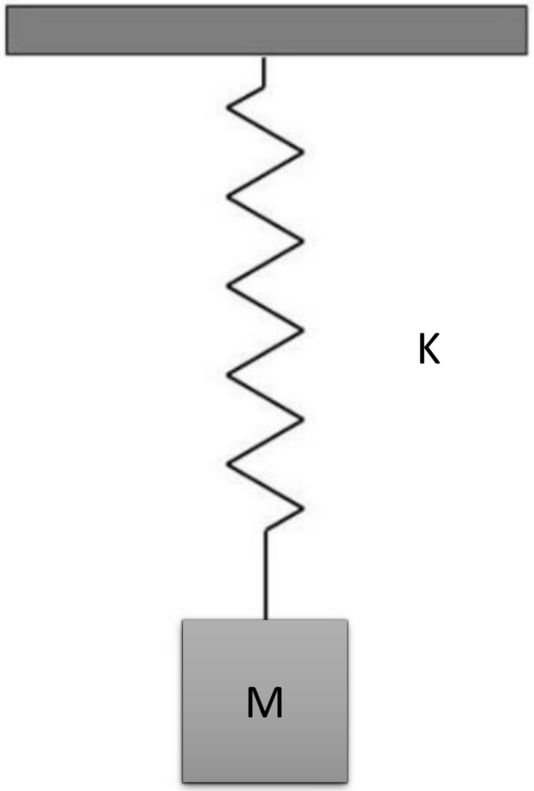


Equations of motion (linear ODE)

- $M\ddot{x} = f(x)$

- For linear materials, we have $f(x) = -K(x - X)$
  - We, therefore, yield a linear differential equation:
    - $M\ddot{x} + K(x - X) = 0$
    - Or sometimes: $M\ddot{u} + Ku = 0$ (define displacement $u \coloneqq x - X$)

Note: linear materials are widely used for small deformations, such as in physically based **sound simulation** (for rigid bodies) and **topology optimization**



Equations of motion (general cases)

- $M\ddot{x} = f(x)$

- $\dot{x} = v$
- $\dot{v} = a = M^{-1}f$

Equations of motion (general cases)

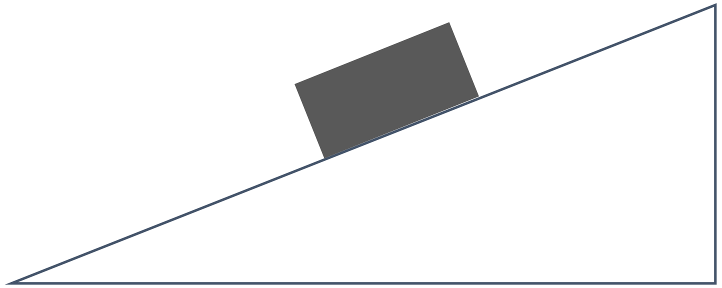
- $M\ddot{x} = f(x)$

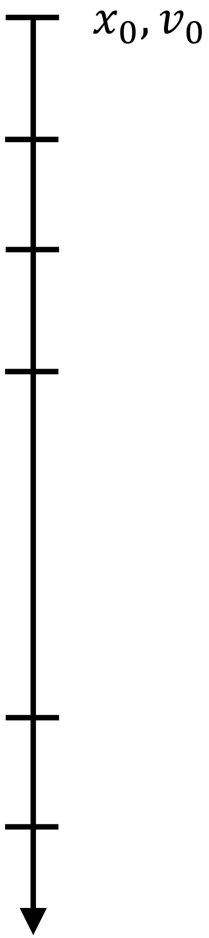
- $\dot{x} = v$
- $\dot{v} = a = M^{-1}f$

```
    for i in range(N):
        #update
        vel[i] += dt*force[i]/m
        pos[i] += dt*vel[i]
```

The temporal integration

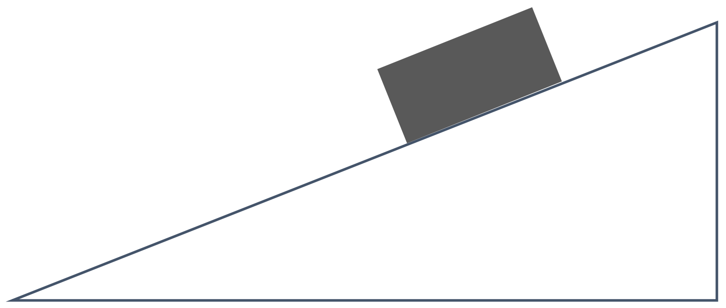
Equations of motion (general cases)



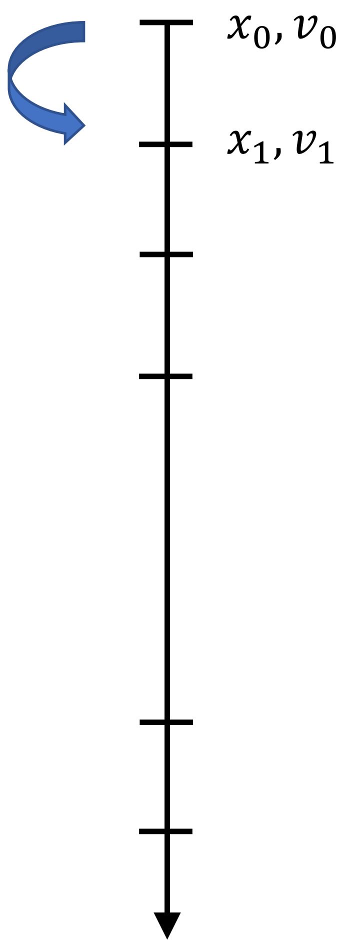



```python
def magic_black_box(x_n, v_n):
    # do something
    return x_np1, v_np1
```

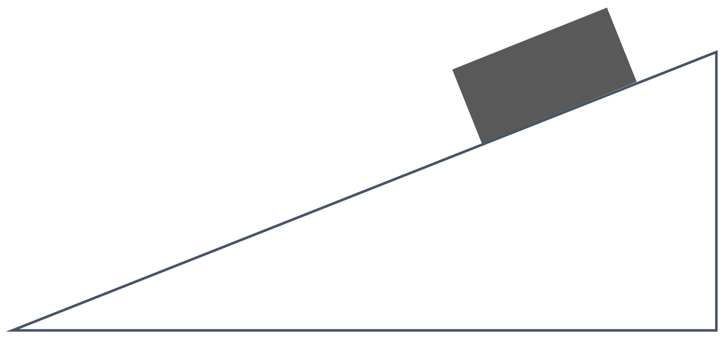
Equations of motion (general cases)

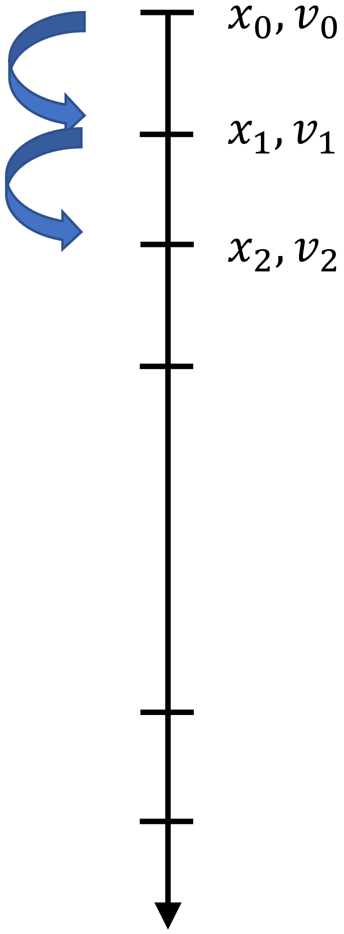


```
def magic_black_box(x_n, v_n):
    # do something
    return x_np1, v_np1
```



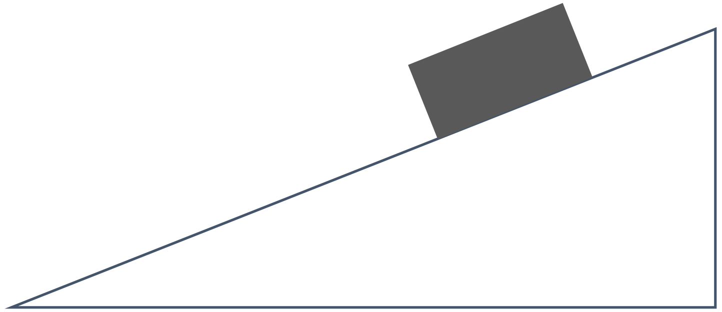
Equations of motion (general cases)



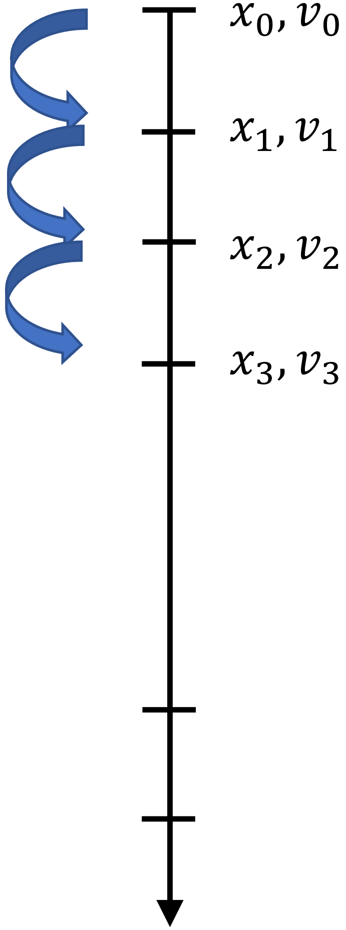



```
def magic_black_box(x_n, v_n):
    # do something
    return x_np1, v_np1
```

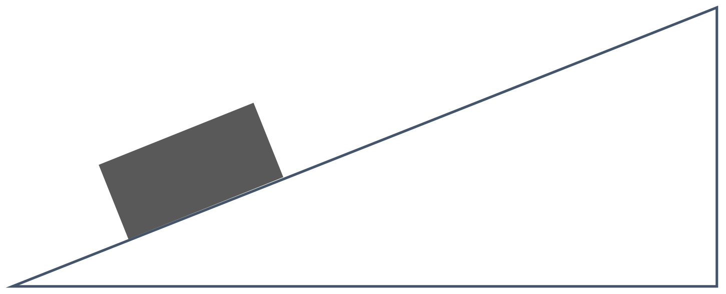
Equations of motion (general cases)



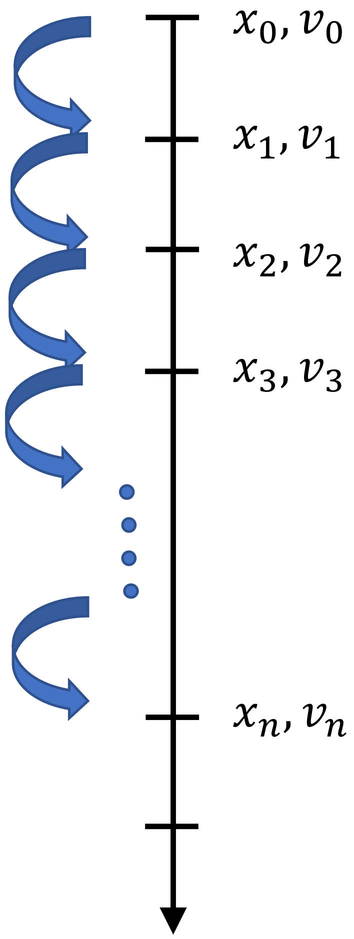
```python
def magic_black_box(x_n, v_n):
    # do something
    return x_np1, v_np1
```



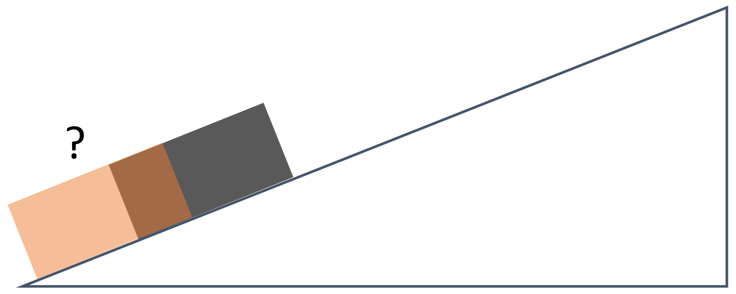
Equations of motion (general cases)



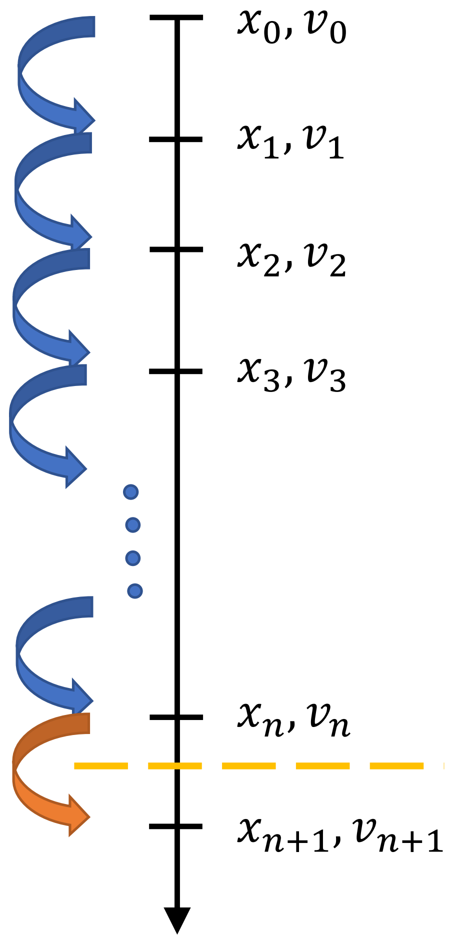
```python
def magic_black_box(x_n, v_n):
    # do something
    return x_np1, v_np1
```



Equations of motion (general cases)




```python
def magic_black_box(x_n, v_n):
    # do something
    return x_np1, v_np1
```



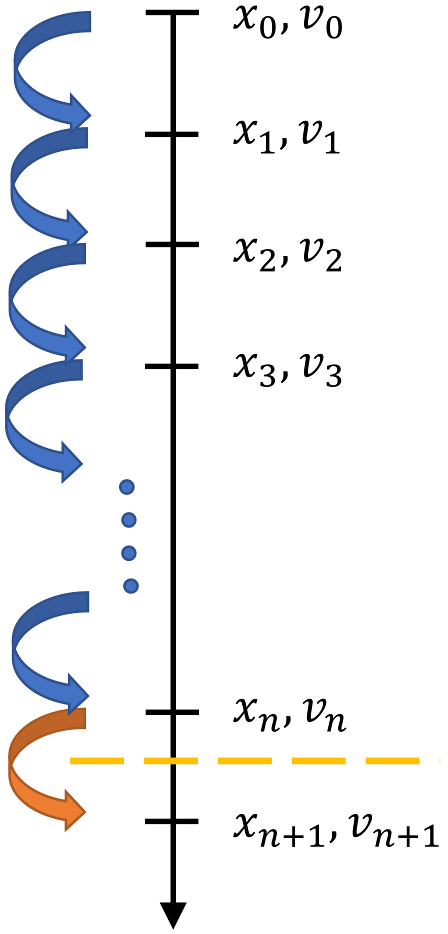
Equations of motion (general cases)

- $M\ddot{x} = f(x)$

- $\dot{x} = v$
- $\dot{v} = a = M^{-1}f$

- $x(t_n + h) = x(t_n) + \int_0^h v(t_n + t)dt$
- $v(t_n + h) = v(t_n) + \int_0^h M^{-1}f(t_n + t)dt$

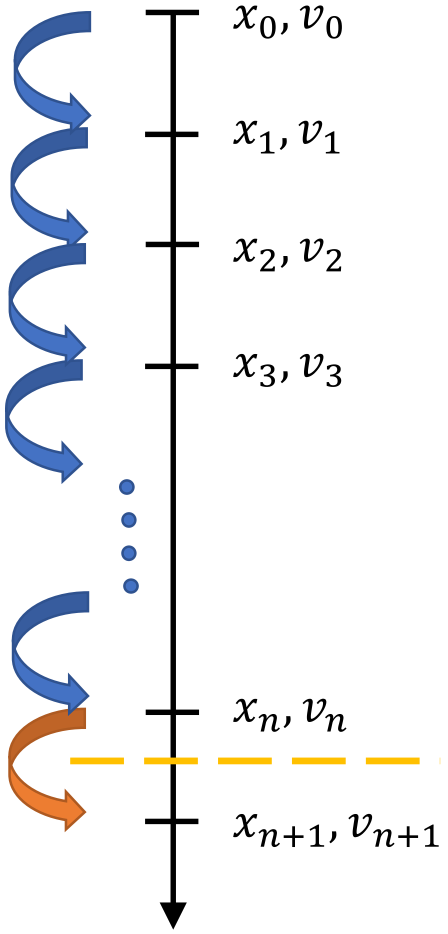
$h = t_{n+1} - t_n$: is the time-step size



Time integration

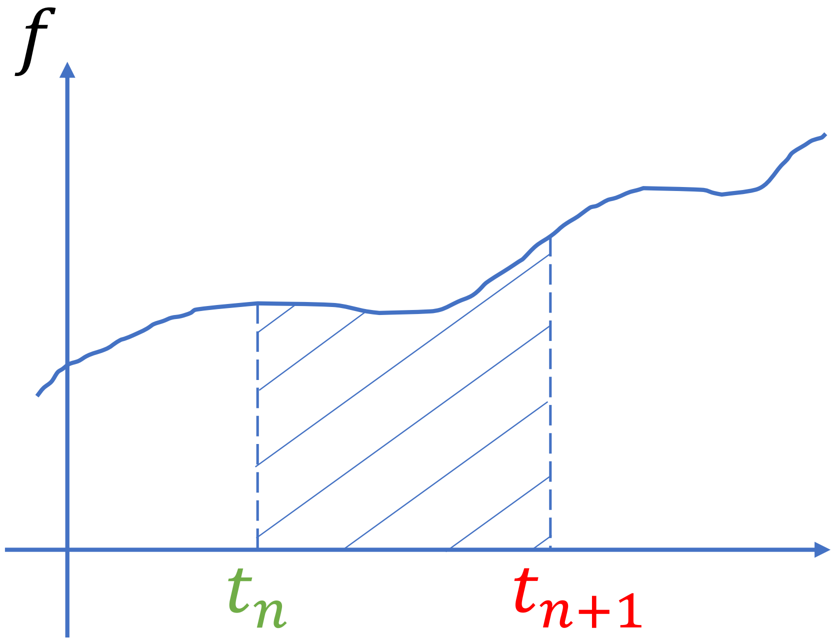
- $x(t_n + h) = x(t_n) + \int_0^h v(t_n + t)dt$
- $v(t_n + h) = v(t_n) + \int_0^h M^{-1} f(t_n + t)dt$

- We don't know how to integrate this quantity
- We don't know anything after t_n



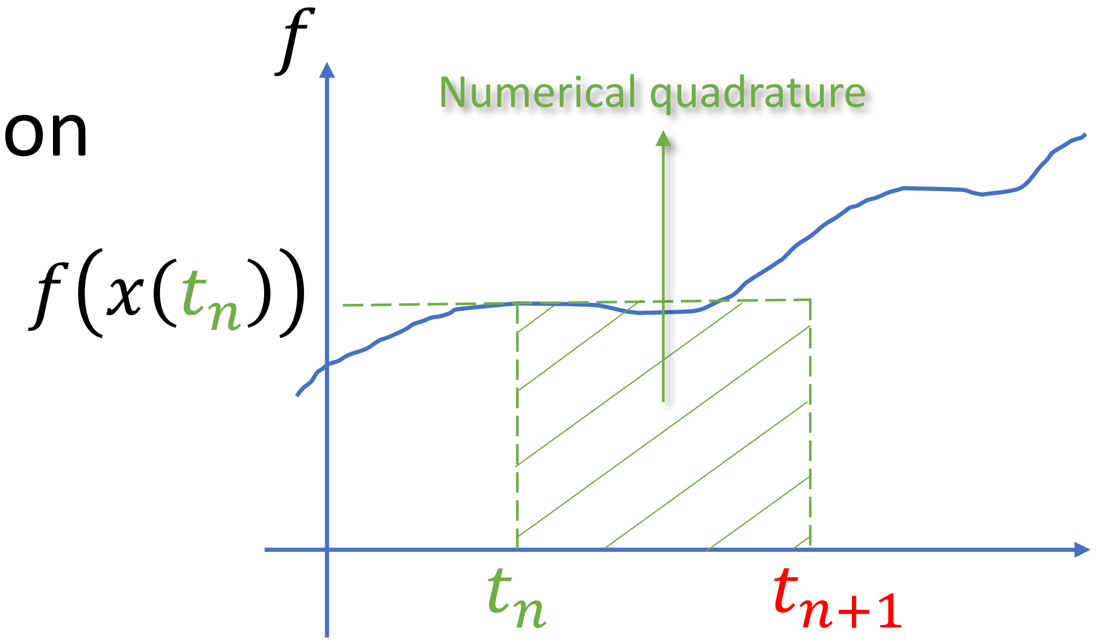
Time integration

- $x(t_n + h) = x(t_n) + \int_0^h v(t_n + t)dt$
- $v(t_n + h) = v(t_n) + \int_0^h M^{-1} f(t_n + t)dt$



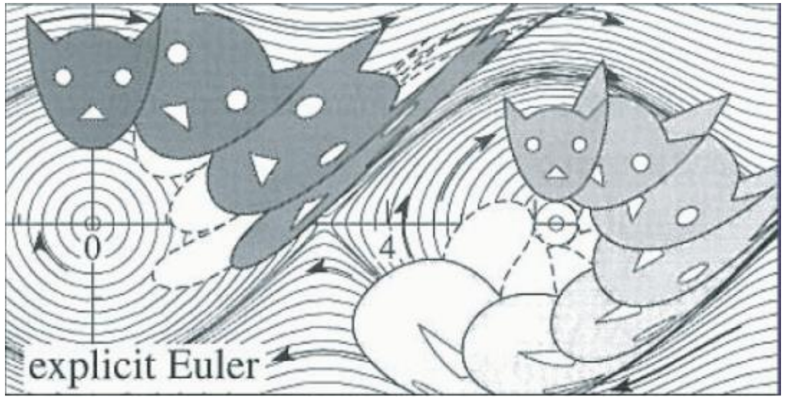
Time integration (explicit)

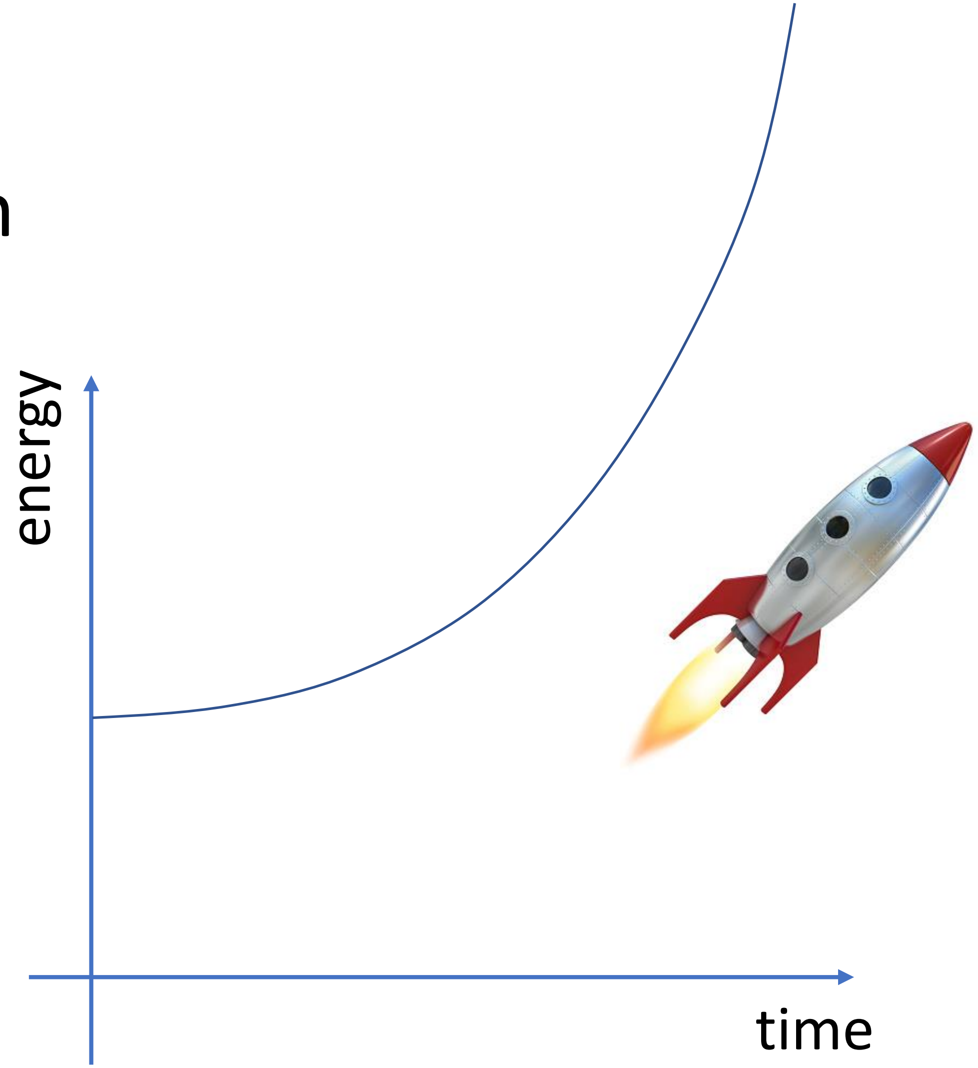
- Explicit(forward) Euler integration
  - $x_{n+1} = x_n + hv_n$
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$



Time integration (explicit)

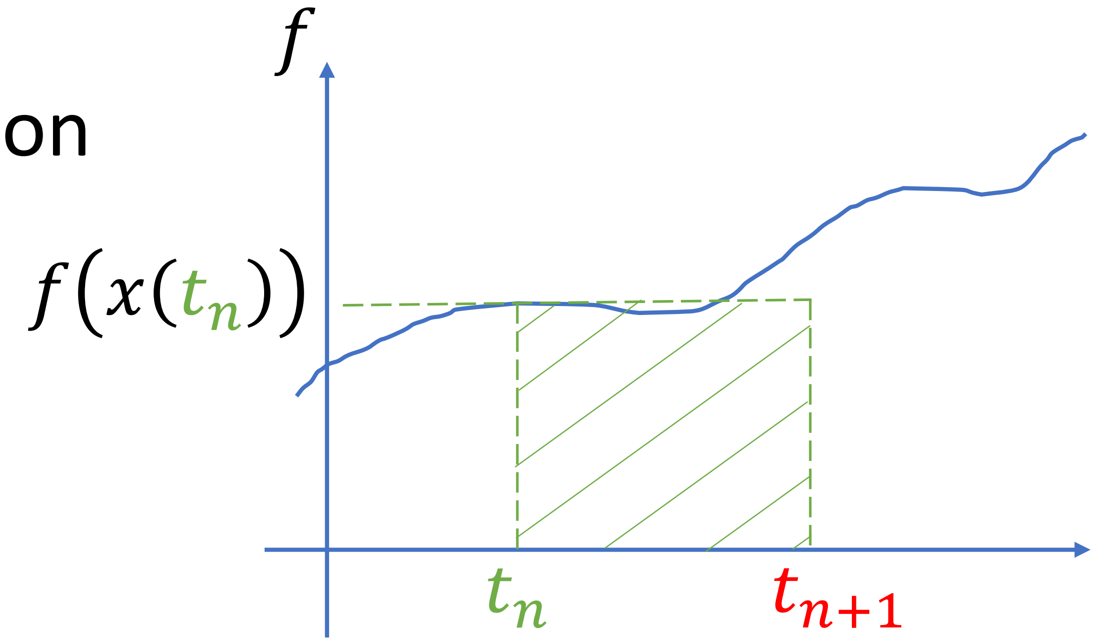
- Explicit(forward) Euler integration
  - $x_{n+1} = x_n + hv_n$
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$





Time integration (explicit)

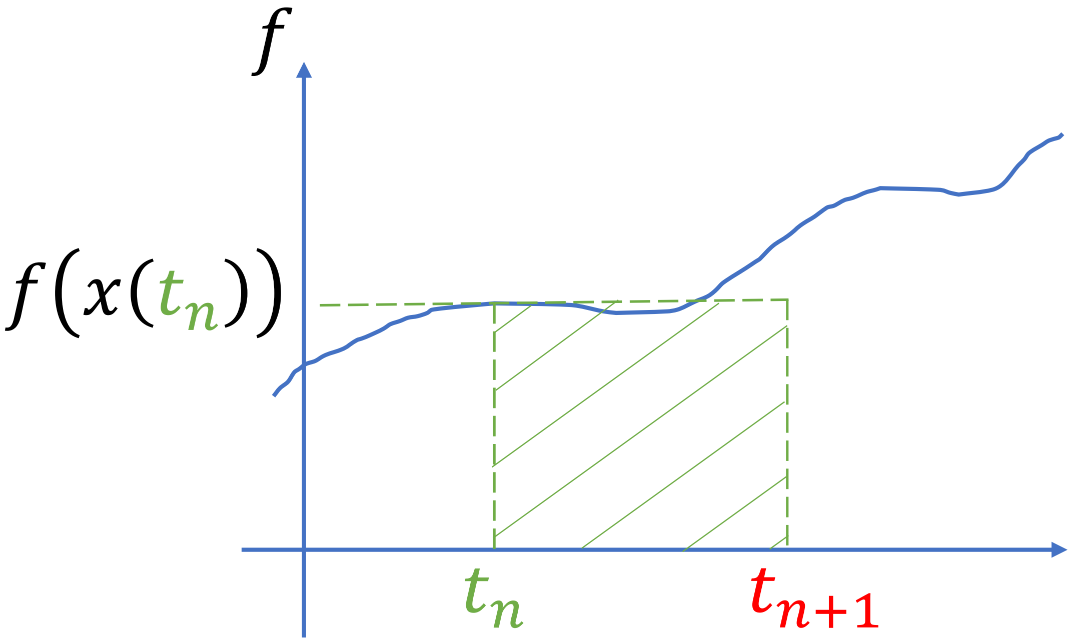
- Explicit(forward) Euler integration
  - $x_{n+1} = x_n + hv_n$
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$



Note: Forward Euler is **extremely fast**, but it will also **increase the system energy** gradually. It is **seldom used** for the existence of symplectic Euler integration.

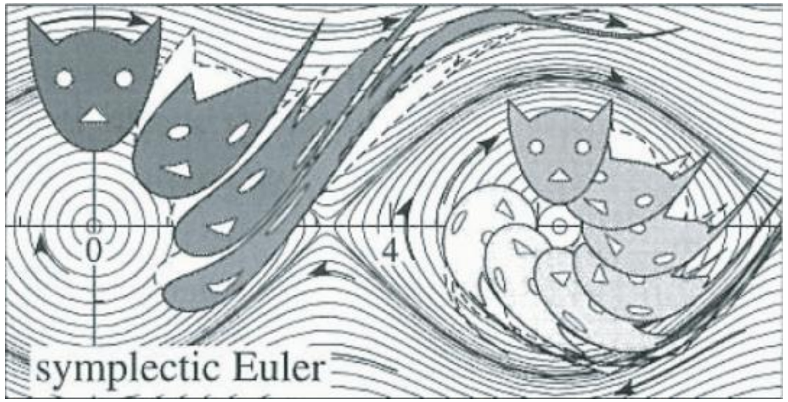
Time integration (explicit)

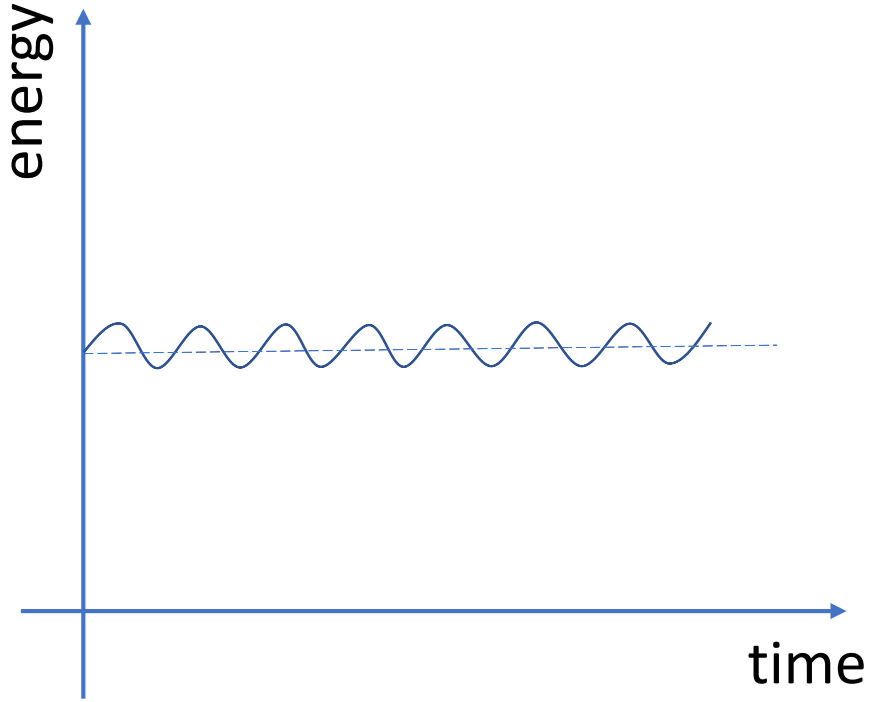
- Symplectic Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$
  - $x_{n+1} = x_n + hv_{n+1}$



Time integration (explicit)

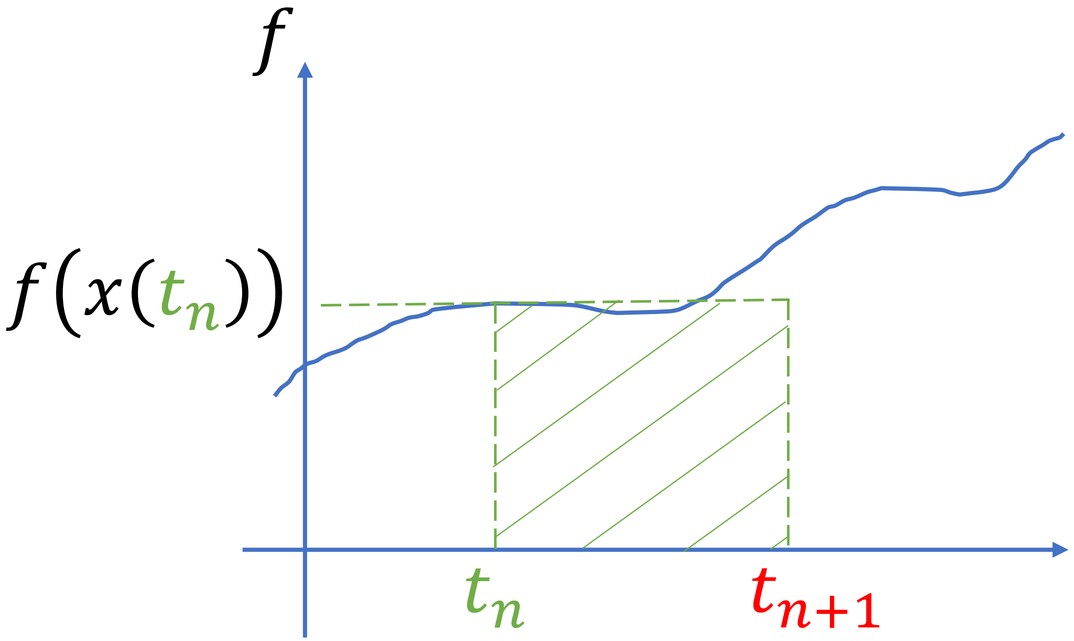
- Symplectic Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$
  - $x_{n+1} = x_n + hv_{n+1}$





Time integration (explicit)

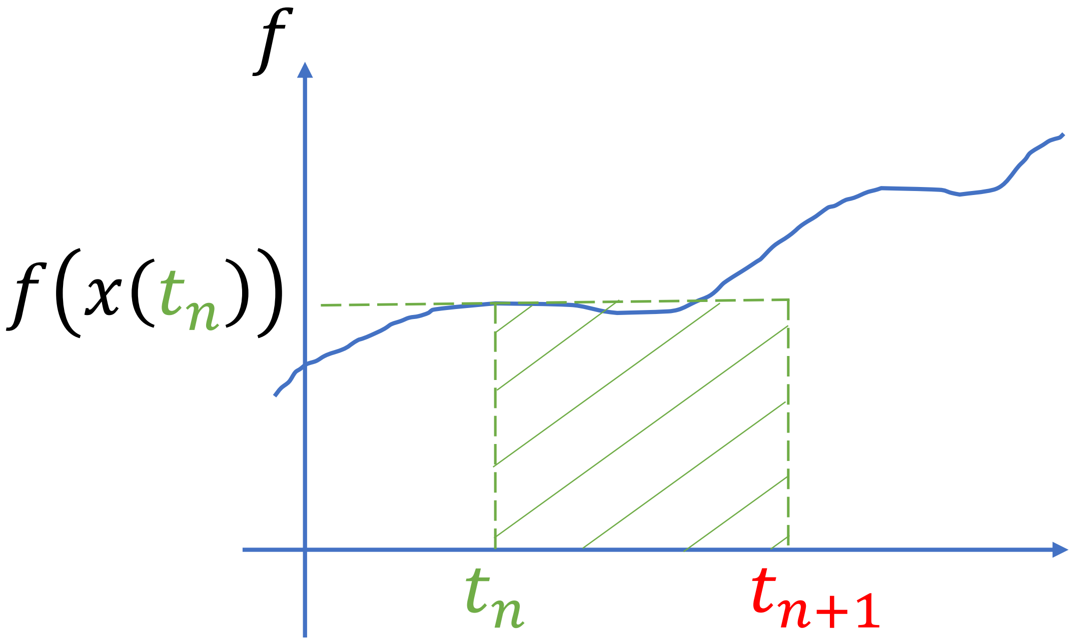
- Symplectic Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$
  - $x_{n+1} = x_n + hv_{n+1}$



Note: Symplectic Euler is as **fast** as forward Euler, it is **momentum preserving**, it has an **oscillating system Hamiltonian**. It is often THE explicit integration method to use. It has been widely used in **accuracy-centric applications** (astronomy simulation / molecular dynamics etc).

Time integration (explicit)

- Symplectic Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$
  - $x_{n+1} = x_n + hv_{n+1}$

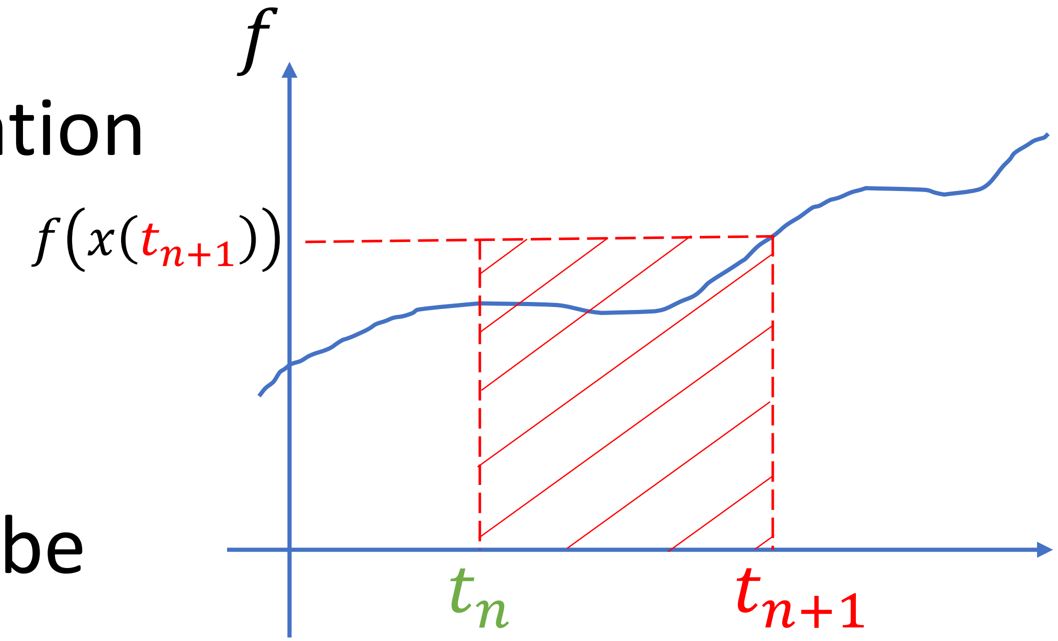


Further Reading: The geometric integrator [Link]

Time integration (implicit)

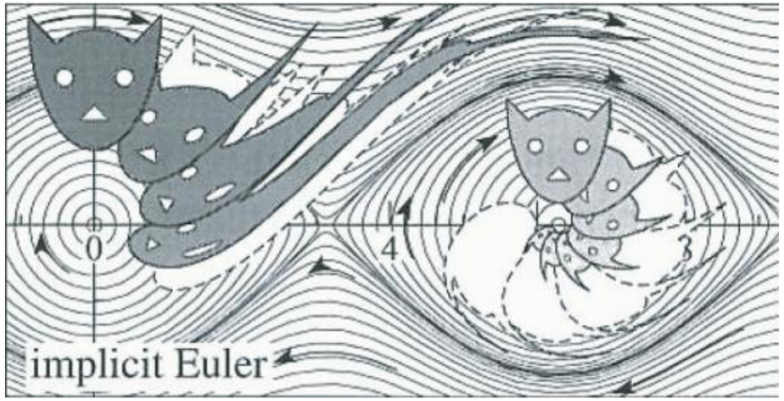
- Implicit (backward) Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_{n+1})$
  - $x_{n+1} = x_n + hv_{n+1}$

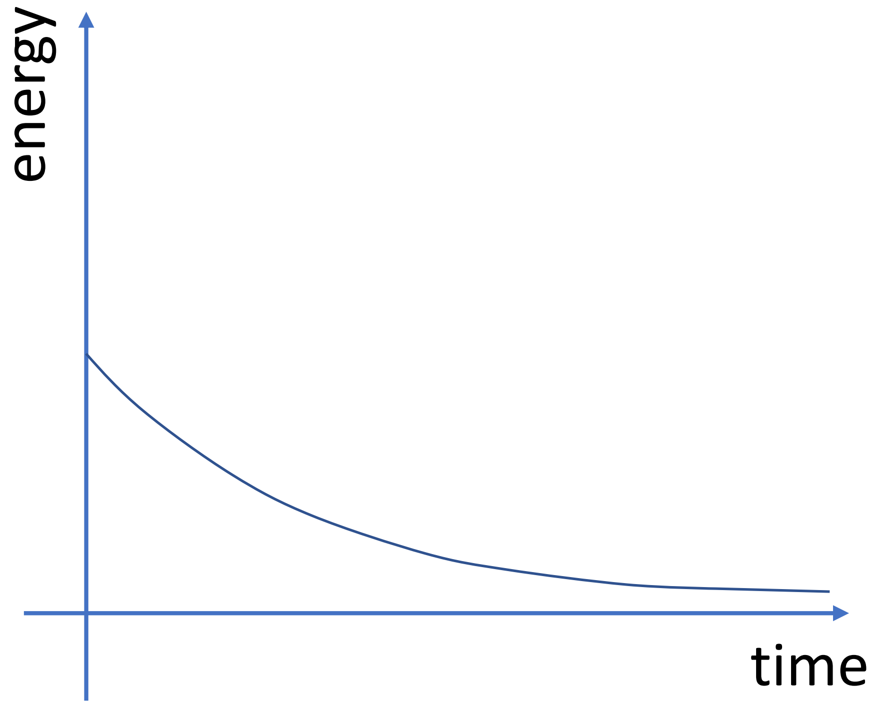
- The *nonlinear system solver* will be covered in the next class.



Time integration (implicit)

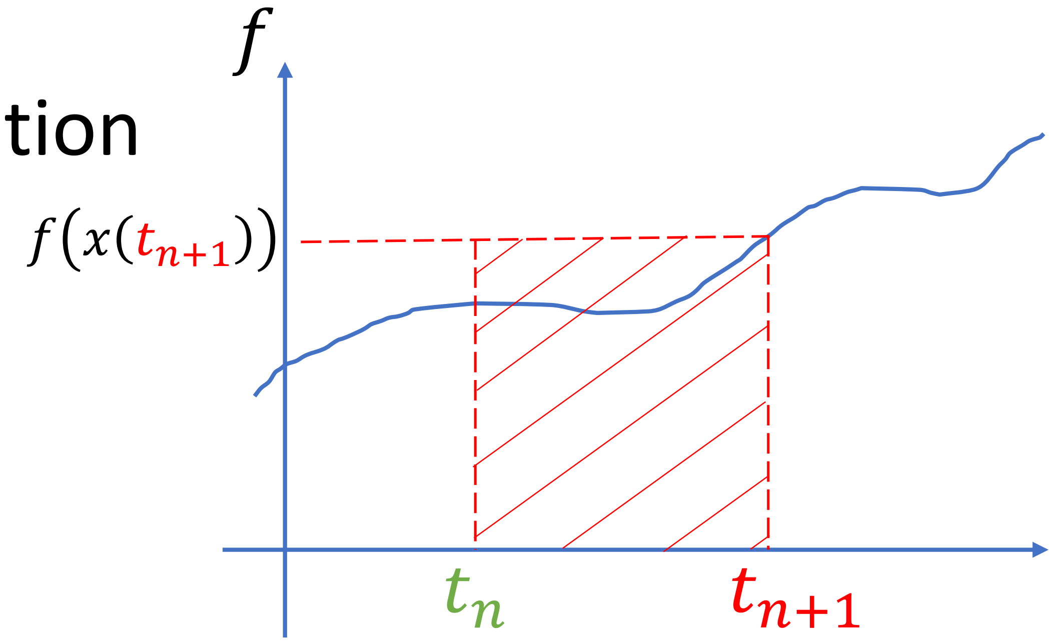
- Implicit (backward) Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_{n+1})$
  - $x_{n+1} = x_n + hv_{n+1}$





Time integration (implicit)

- Implicit (backward) Euler integration
  - $v_{n+1} = v_n + hM^{-1}f(x_{n+1})$
  - $x_{n+1} = x_n + hv_{n+1}$



Note: Implicit Euler is often **expensive** due to the nonlinear optimization, it **damps the Hamiltonian** from the oscillating components, it is often **stable for large time-steps** and is widely used in performance-centric applications. (game / MR / design / animation)

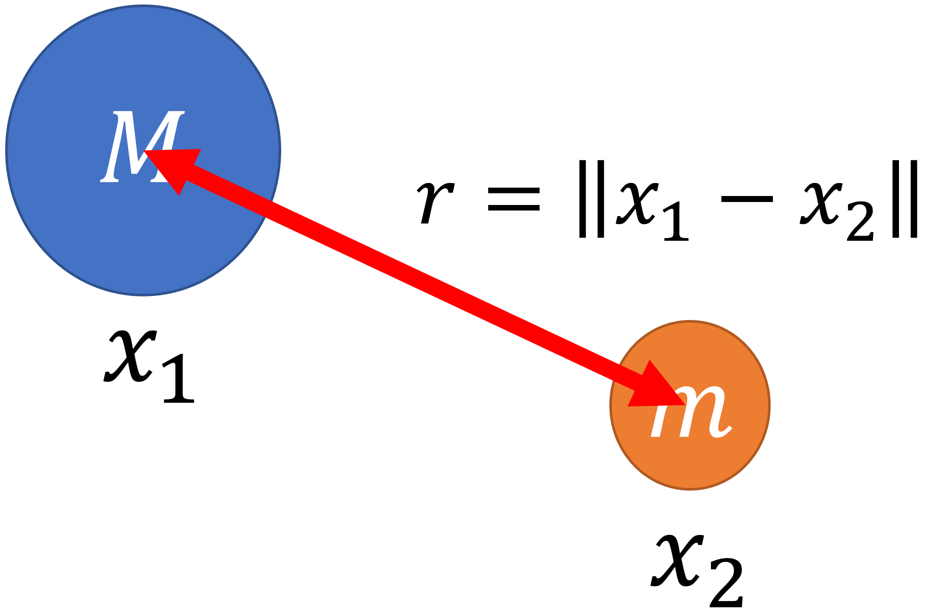
Time integration in practice

- Explicit integration:
  - $v_{n+1} = v_n + hM^{-1}f(x_n)$
  - $x_{n+1} = x_n + hv_{n+1}$

- Time integration steps:
  - Evaluate f at x_n
    - For conservative force: $f(x) = -E(x)$, where E is the potential energy
  - Update v using f (or $M^{-1}f$)
  - Update x using v

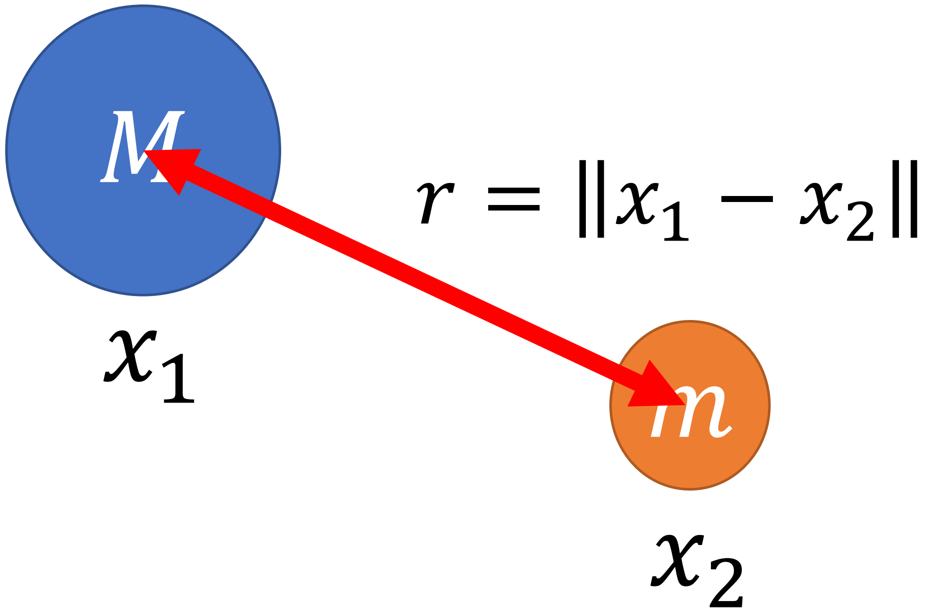
Time integration (an example)

- Gravitational energy:
  - $E = -\frac{GMm}{r(x_1, x_2)}$
- Gradient (gravitational force):
  - $\frac{\partial E}{\partial x_1} = \frac{\partial r}{\partial x_1} \cdot \frac{\partial E}{\partial r} = \frac{x_1 - x_2}{r} * \frac{GMm}{r^2}$
  - $f(x_1) = -\frac{\partial E}{\partial x_1}$
  - $f(x_2) = -\frac{\partial E}{\partial x_2}$
    - or $f(x_2) = -f(x_1)$



Time integration (an example)

- $r = \|x_1 - x_2\| = \sqrt{(x_1 - x_2)^T (x_1 - x_2)}$
- $\frac{\partial r}{\partial x_1} = \left(2(x_1 - x_2)\right) * \frac{1}{2} \frac{1}{\sqrt{(x_1 - x_2)^T (x_1 - x_2)}}$



- Further Readings:
  - *Calculus On Manifolds* [Link]
  - *The Matrix Cookbook* [Link]

The N-body problem [Link]

```python
# compute gravitational force
for i in range(N):
    p = pos[i]
    for j in range(i):
        diff = p-pos[j]
        r = diff.norm(1e-5)

        f = -G * m * m * (1.0/r)**3 * diff

        force[i] += f
        force[j] += -f
```

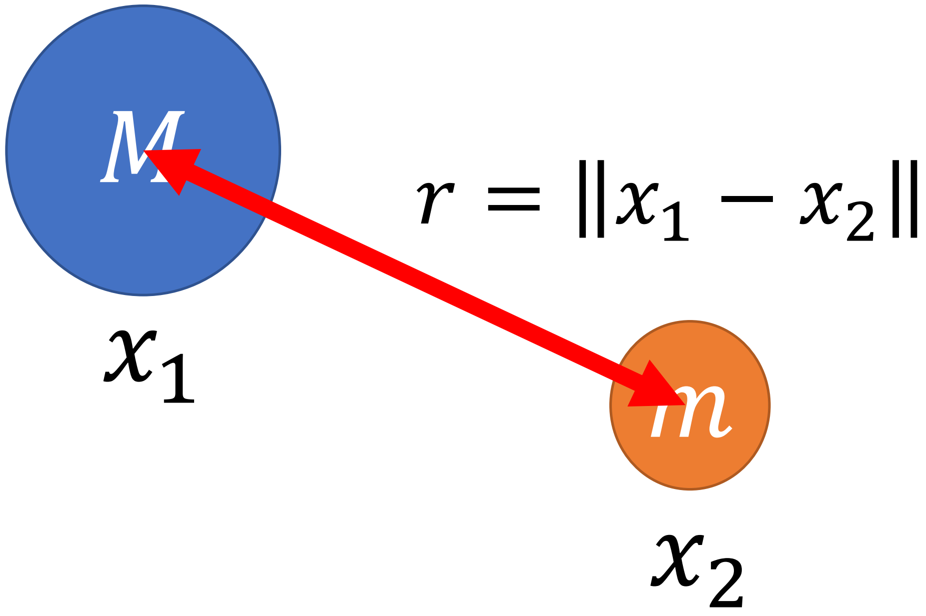
Compute force

```python
for i in range(N):
    #symplectic euler
    vel[i] += dt*force[i]/m
    pos[i] += dt*vel[i]
```

Time integration

The **energy** is all we need

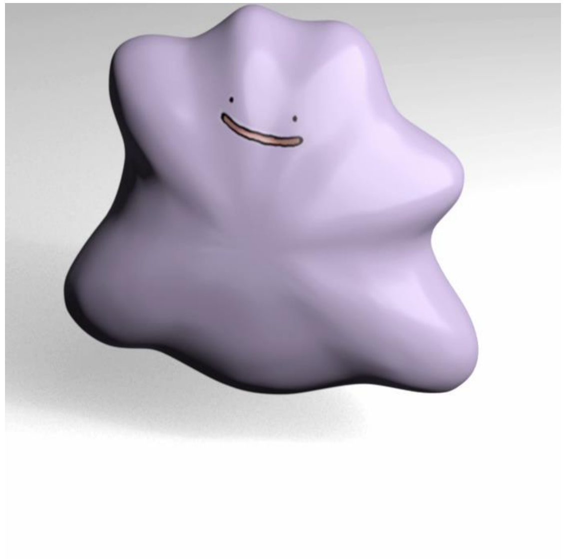
- Gravitational energy:
  - $E = -\frac{GMm}{r(x_1, x_2)}$



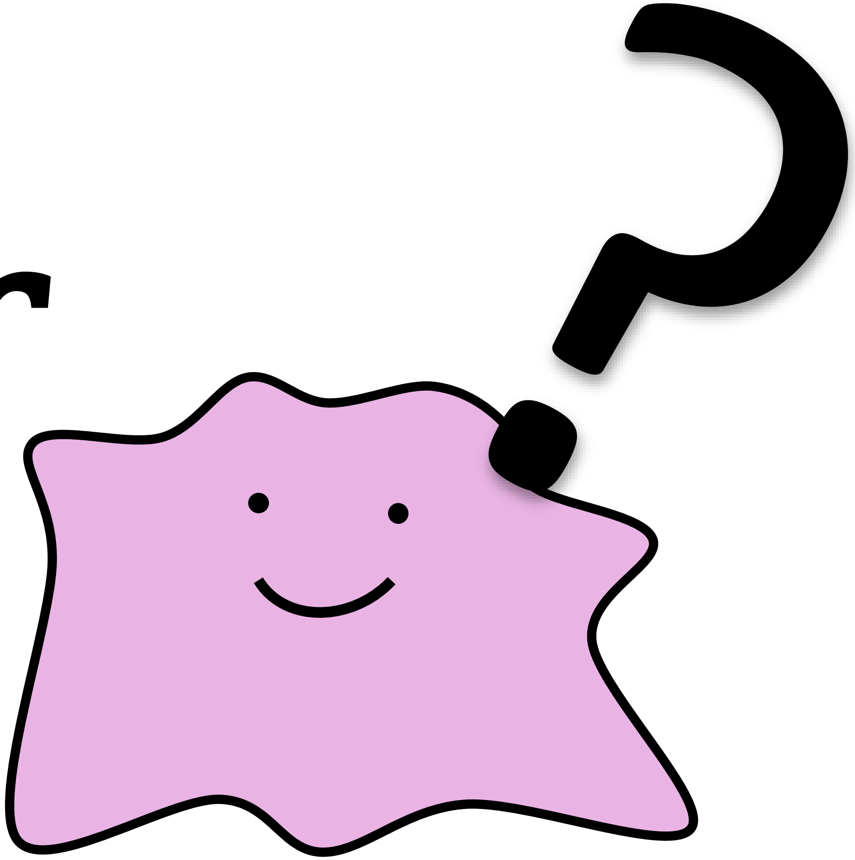
- Take-away:
  - For conservative forces (as most of the elastic forces are), the **energy** definition is all we need for their simulations.

The **energy** is all we need

- A deformable object is a:
  - continuum body



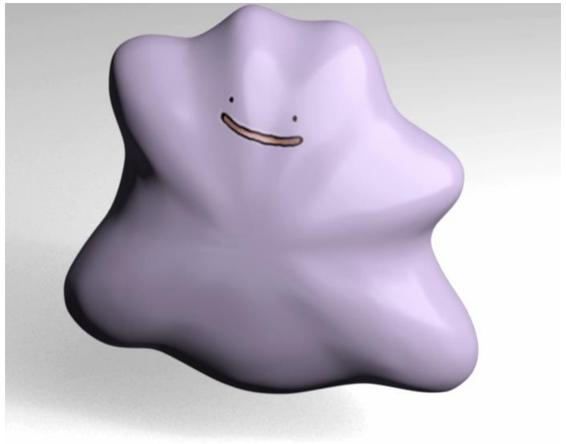
$$E = \int$$



The spatial integration

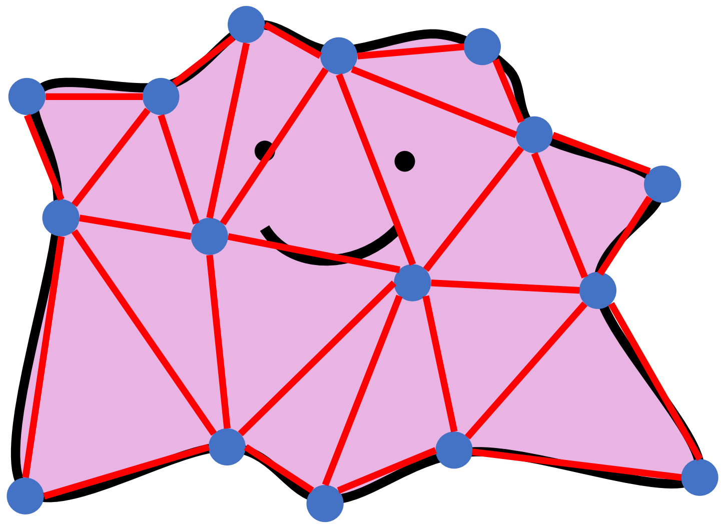
The energy of a deformable continuum body

- Keep these questions in mind…
  - How to describe the deformation?
  - How to describe the elastic energy?
- … when we go through:
  - A mass-spring system
  - The linear finite element method



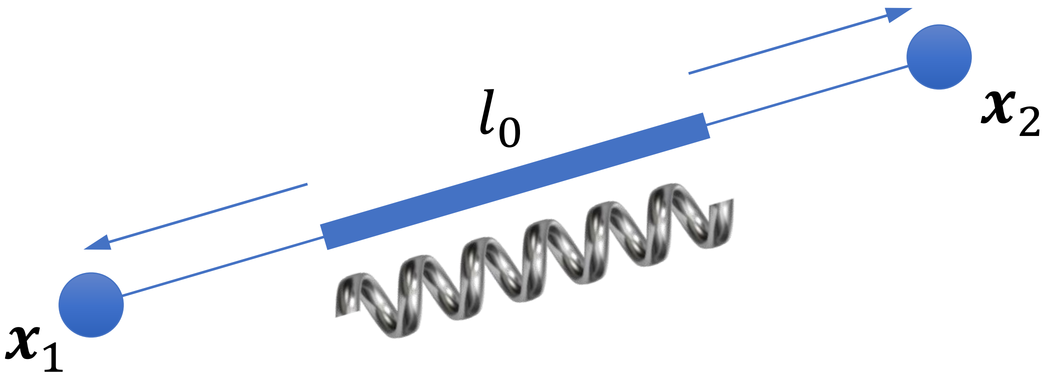
Mass-spring system
-- A simple yet useful discrete deformation model

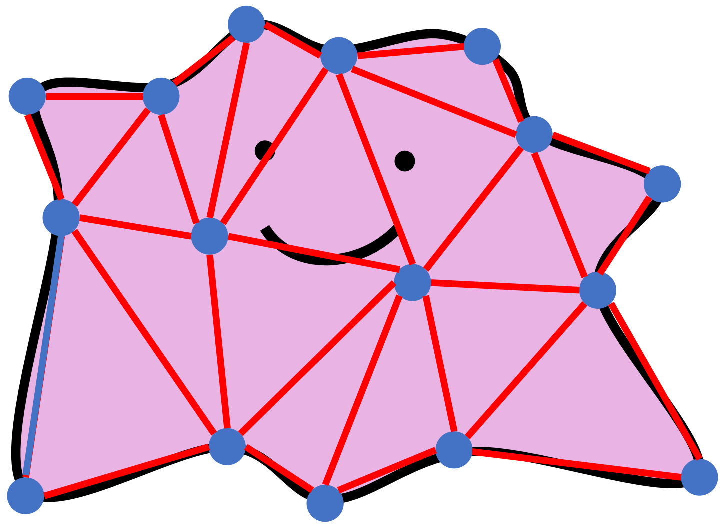
- Tessellate the mesh into a discrete one
- Aggregate the volume mass to the vertices
- Link the mass-vertices with springs



Mass-spring system

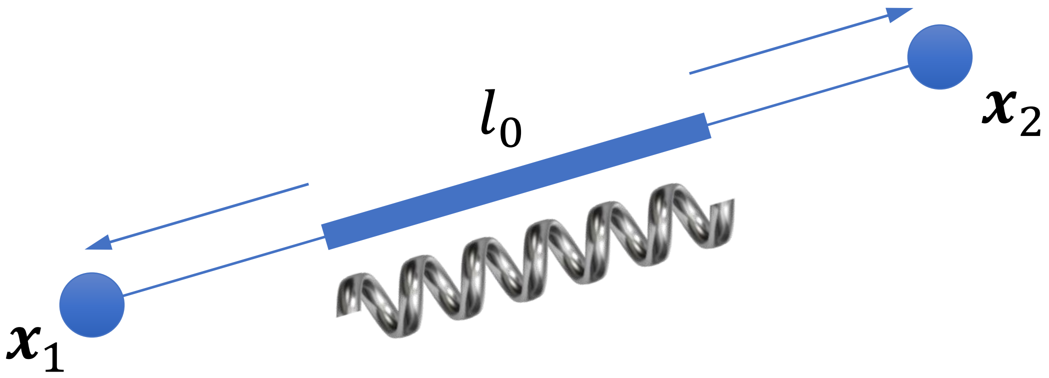
-- A simple yet useful discrete deformation model





Mass-spring system

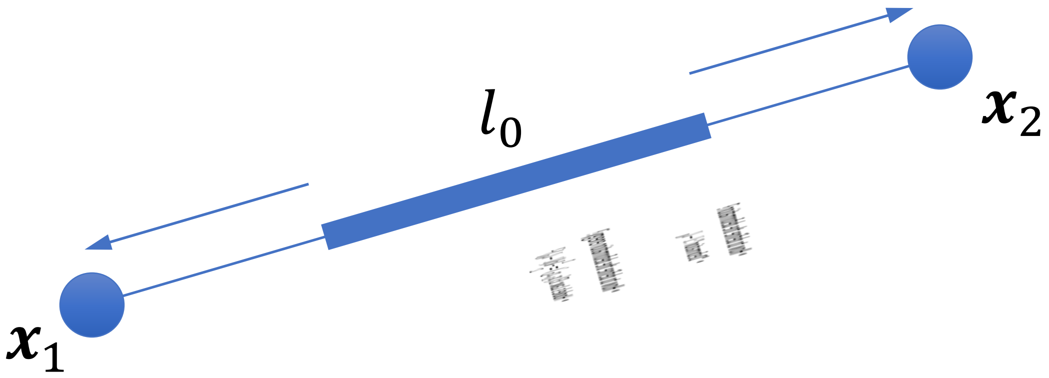
- How to define the deformation?
  - Spring current pose: x_1, x_2
  - Spring current length: $l = \|x_1 - x_2\|$
  - Spring rest-length: l_0
  - "Deformation": $l - l_0$



Mass-spring system

- How to define the deformation?
  - "Deformation": $l - l_0$

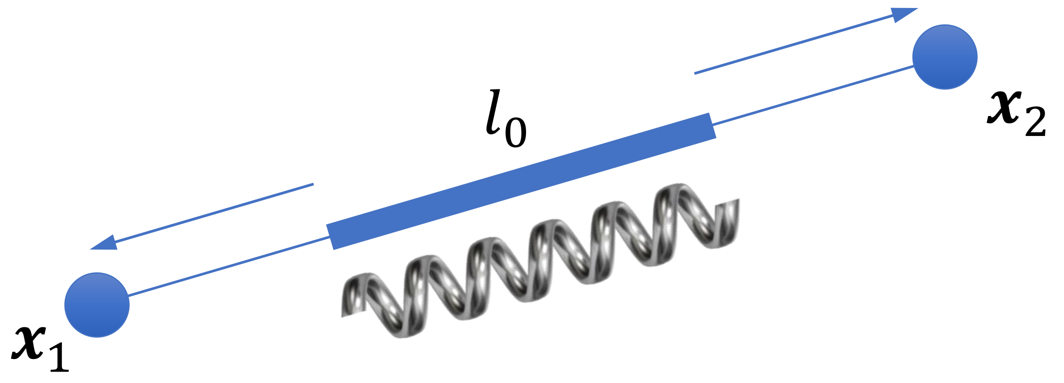
- How to define the deformation energy?
  - Hooke's Law: $E(x_1, x_2) = \frac{1}{2}k(l - l_0)^2$



Mass-spring system

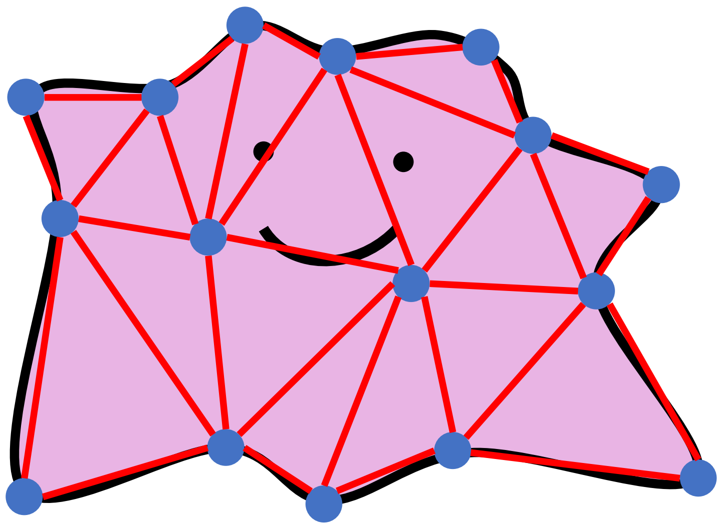
- Elastic energy:
  - $E = \frac{1}{2}k(l - l_0)^2$

- Gradient:
  - $\frac{\partial E}{\partial x_1} = \frac{\partial l}{\partial x_1} \cdot \frac{\partial E}{\partial l} = \frac{x_1 - x_2}{l_0} * k(l - l_0)$
  - $f(x_1) = -\frac{\partial E}{\partial x_1}$
  - $f(x_2) = -f(x_1)$

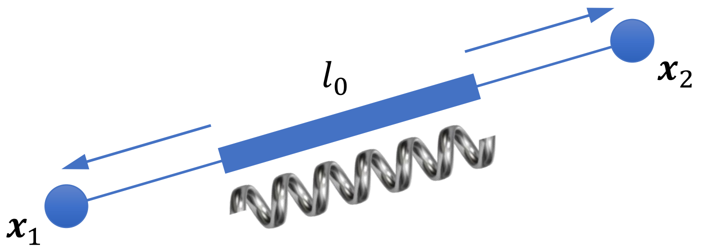


Mass-spring system

$E = \int$



$= \sum$



Mass-spring system (an example)

```
@ti.kernel
def compute_gradient():
    # clear gradient
    for i in range(N_edges):
        grad[i] = ti.Vector([0, 0])

    # gradient of elastic potential
    for i in range(N_edges):
        a, b = edges[i][0], edges[i][1]
        r = x[a]-x[b]
        l = r.norm()
        l0 = spring_length[i]
        k = YoungsModulus[None]*l0
        # stiffness in Hooke's law
        gradient = k*(l-l0)*r/l
        grad[a] += gradient
        grad[b] += -gradient
```

Compute force

```
# symplectic integration
acc = -grad[i]/m - ti.Vector([0.0, g])
v[i] += dh*acc
x[i] += dh*v[i]
```

Time integration

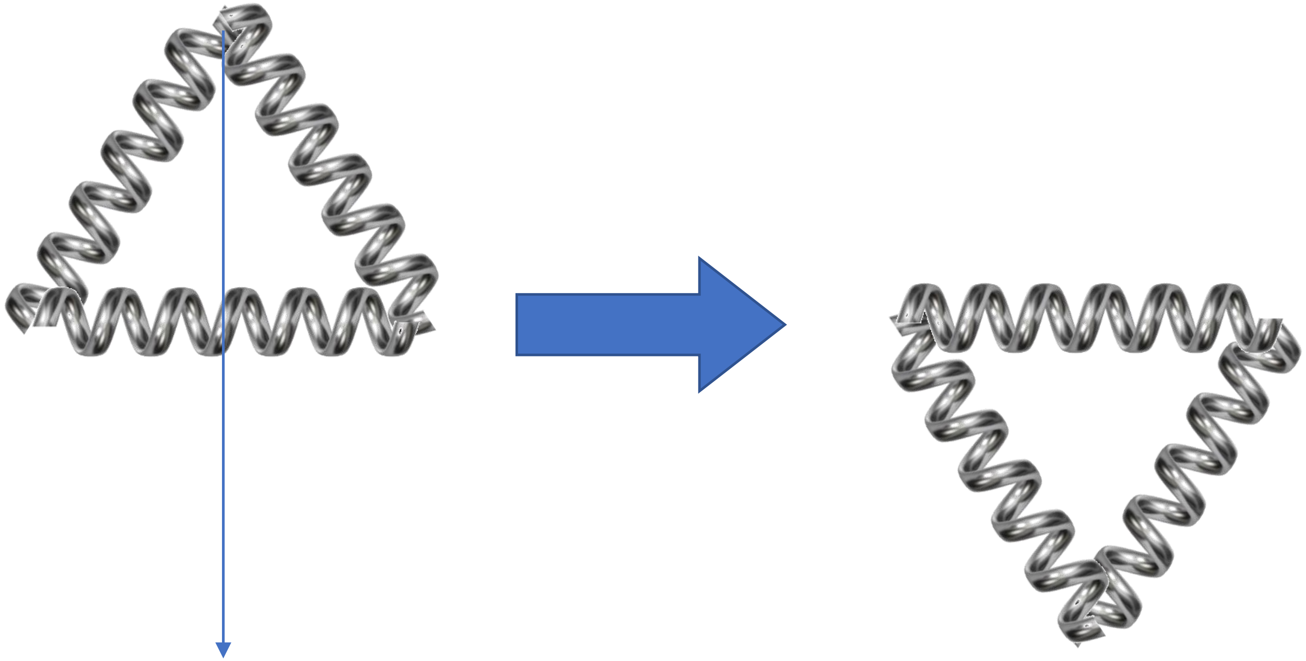
Mass-spring systems are particularly useful in:

Cloth Sim
[Dinev et al. 2018]

Hair Sim
[Selle et al. 2018]

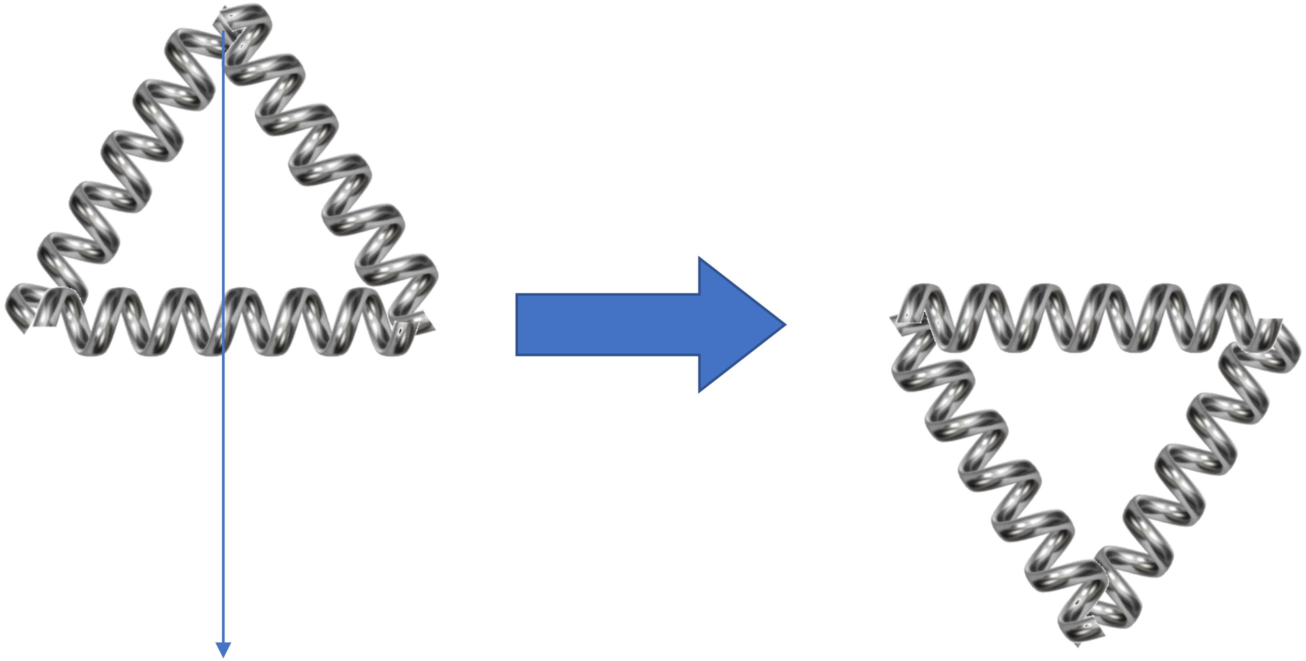
Mass-spring systems are NOT the best choices when simulating **continuum area/volume**

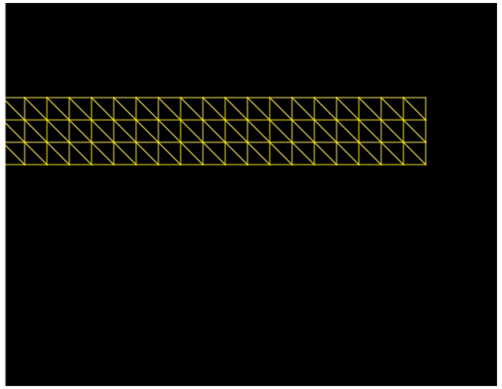
- Area/volume gets inverted without any penalty



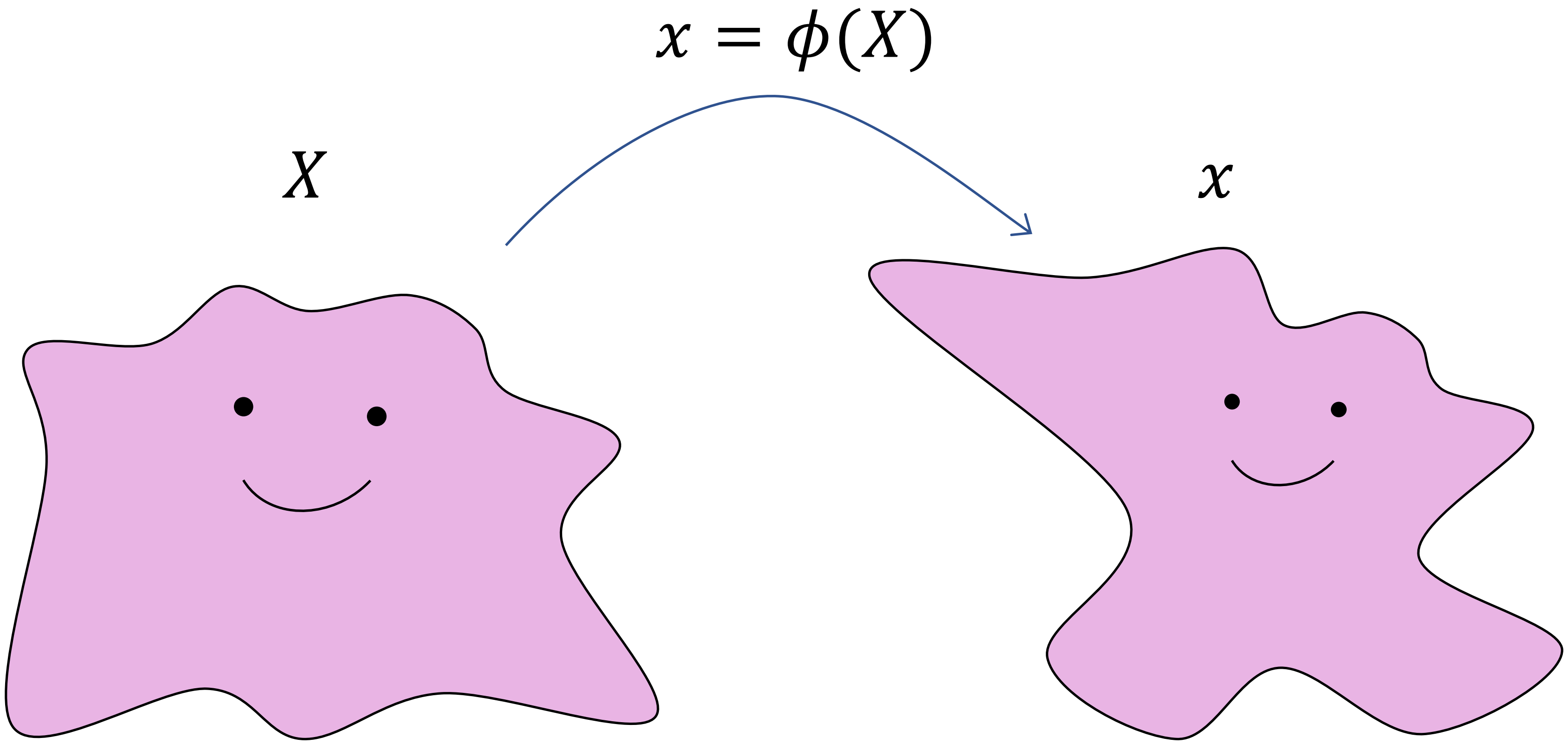
Mass-spring systems are NOT the best choices when simulating **continuum area/volume**

- Area/volume gets inverted without any penalty

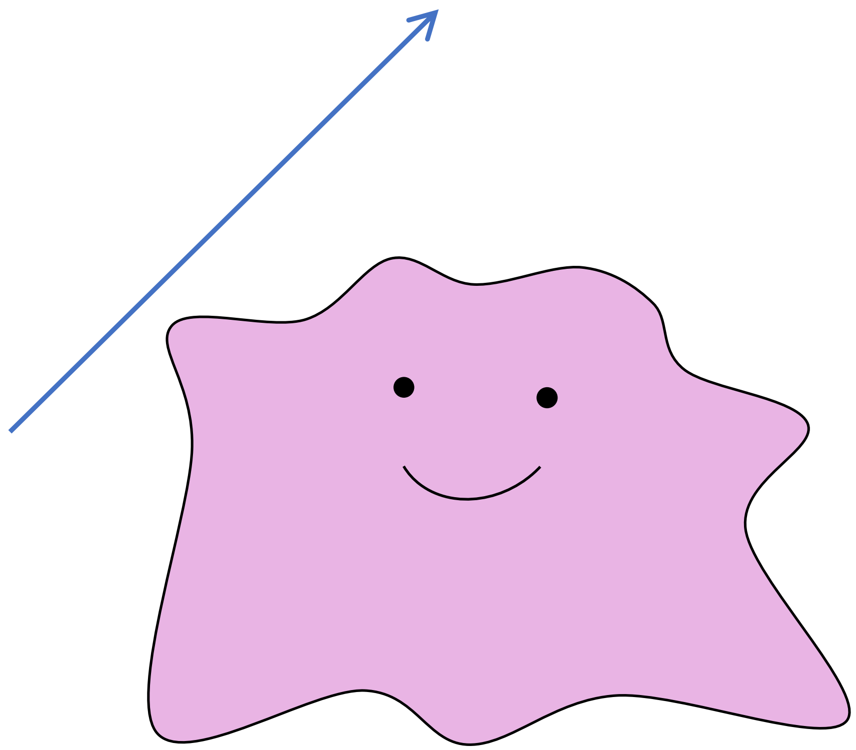




A continuous model to describe deformation



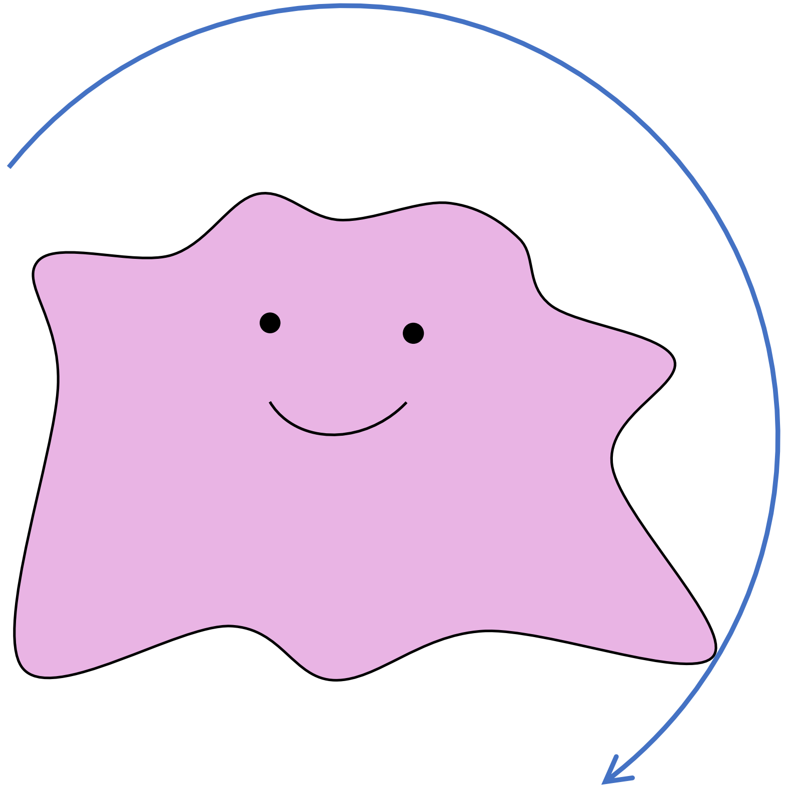
Deformation map



$$\phi(X) = X + t$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

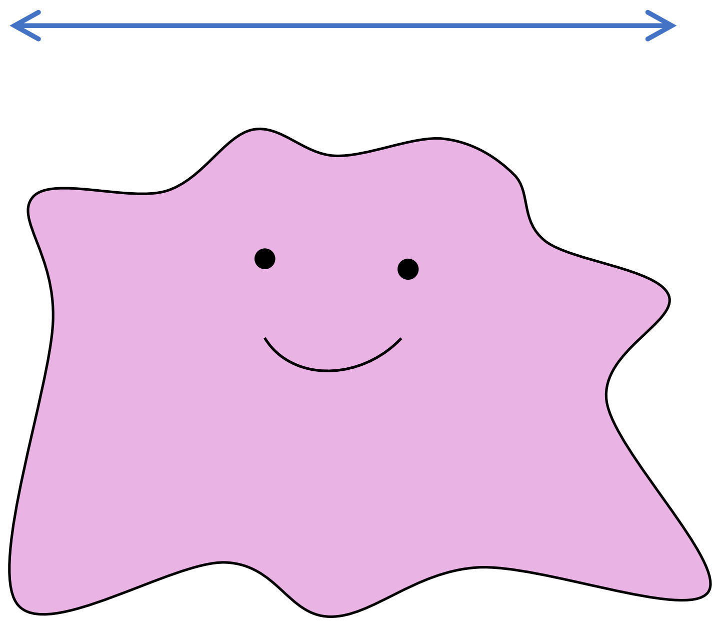
Deformation map



$$\phi(X) = RX$$

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Deformation map

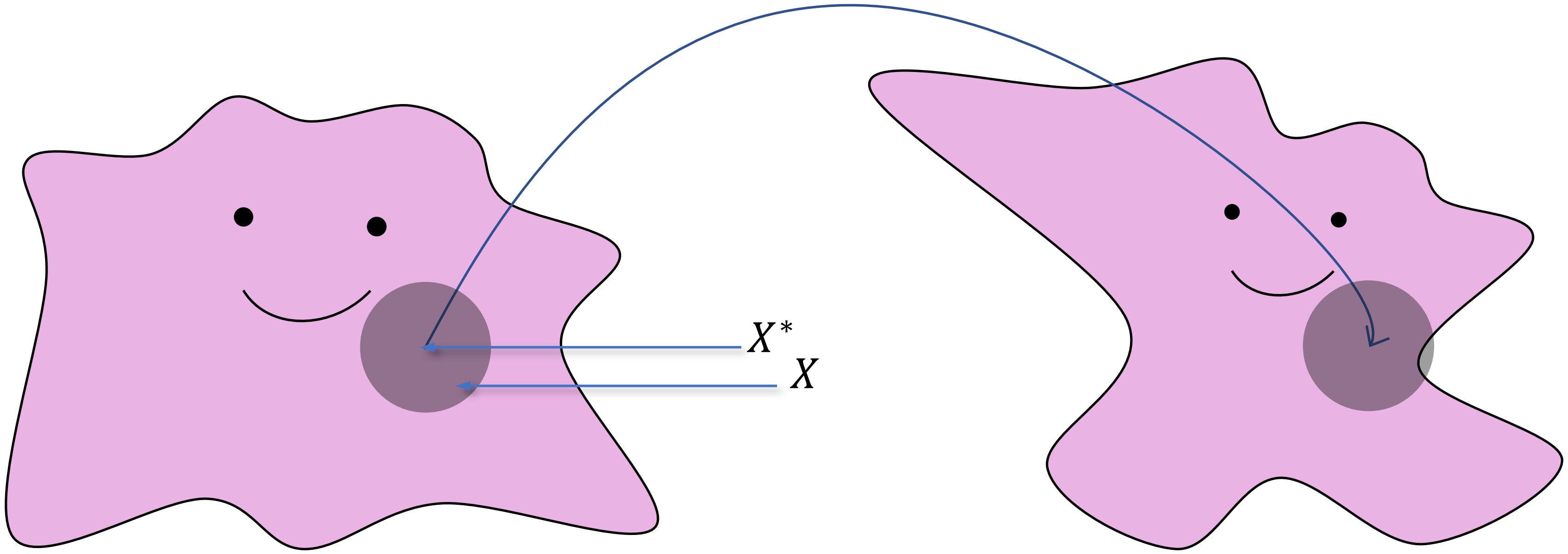


$$\phi(X) = SX$$

$$S = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

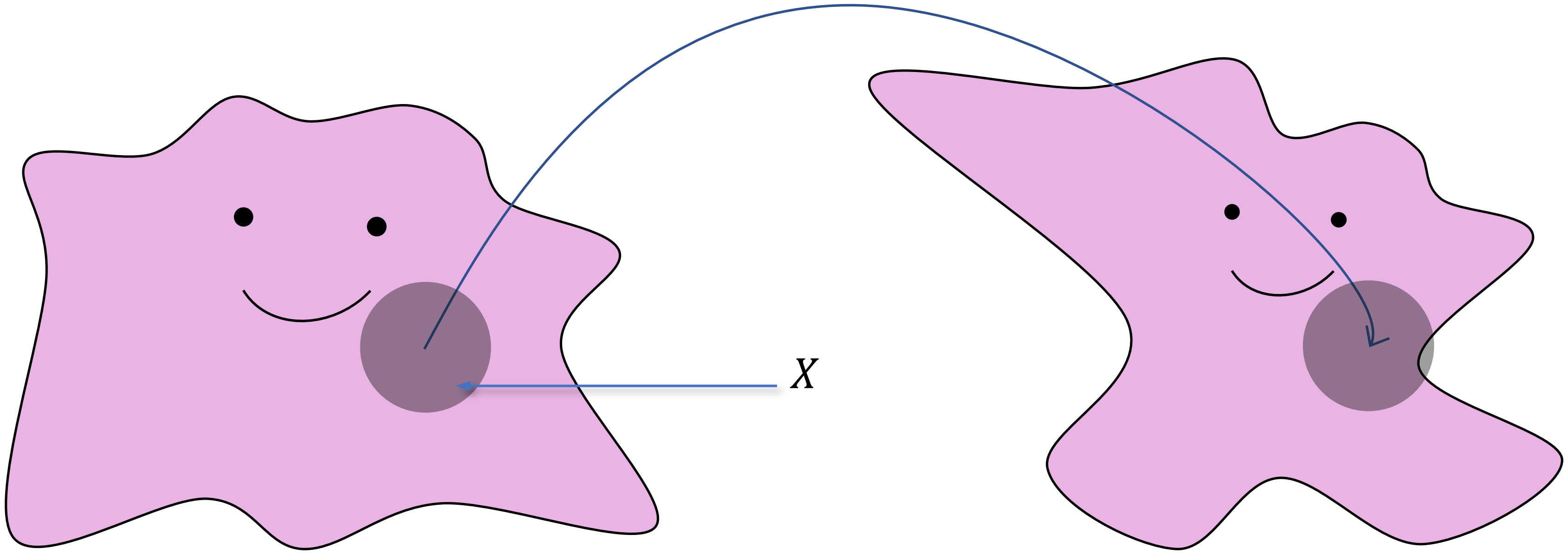
Deformation map

For X near X^*: $\phi(X) \approx \frac{\partial\phi}{\partial X}(X - X^*) + \phi(X^*) = \underbrace{\boxed{\frac{\partial\phi}{\partial X}}}_{F} X + \underbrace{\boxed{\left(\phi(X^*) - \frac{\partial\phi}{\partial X}X^*\right)}}_{t}$

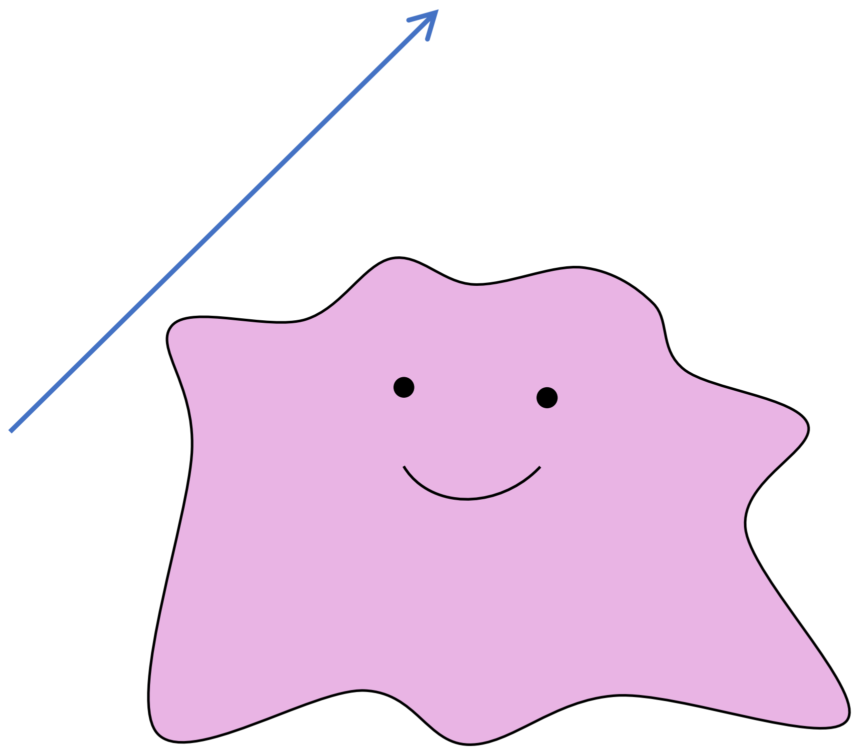


Deformation gradient

$$\phi(X) \approx FX + t$$



Deformation gradient

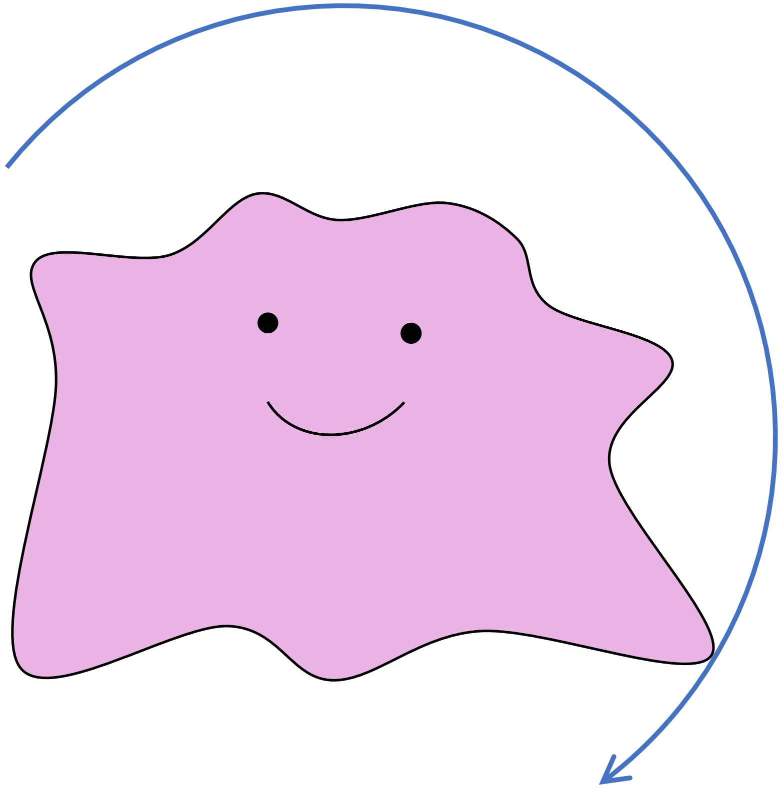


$$\phi(X) = X + t$$

$$F = \frac{\partial \phi}{\partial X} = I$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Deformation gradient

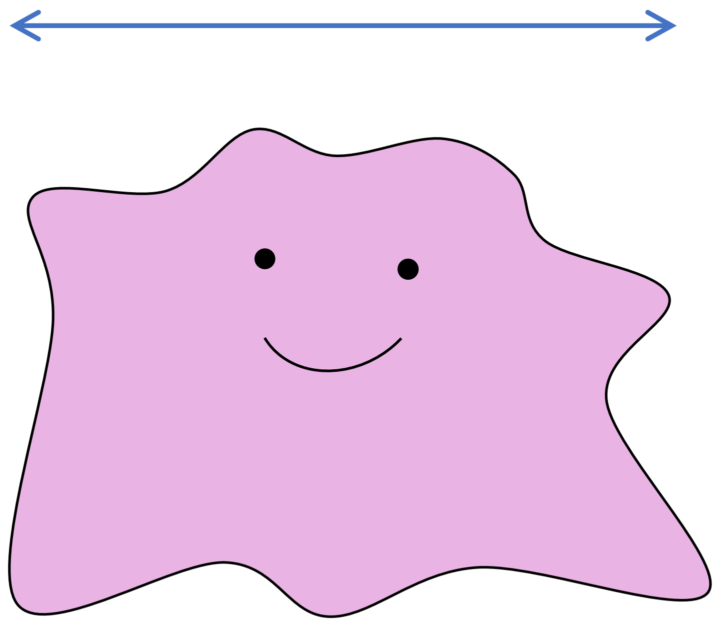


$$\phi(X) = RX$$

$$F = \frac{\partial \phi}{\partial X} = R$$

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Deformation gradient



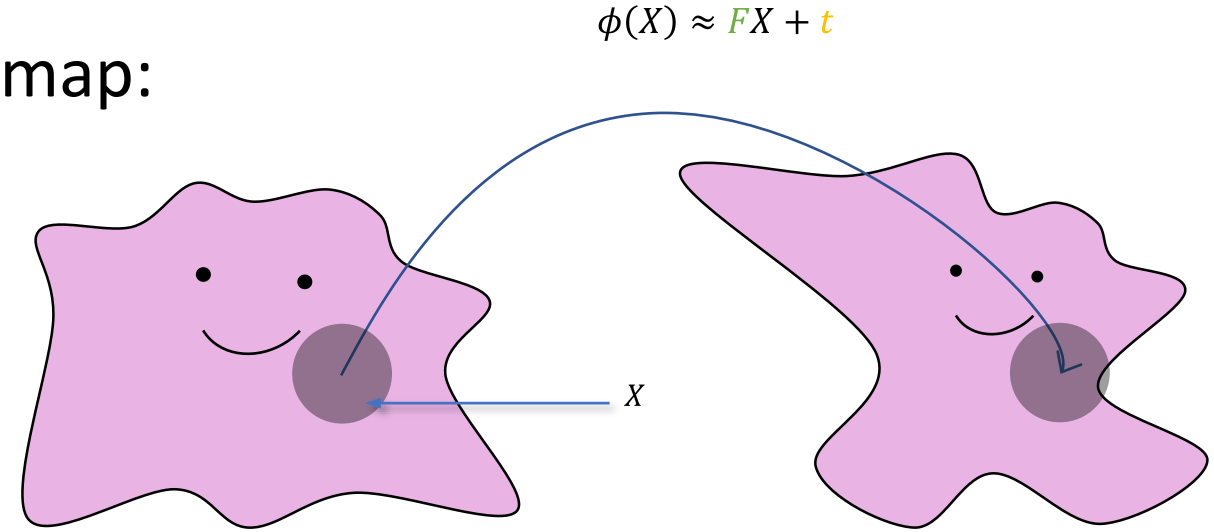
$$\phi(X) = SX$$

$$F = \frac{\partial \phi}{\partial X} = S$$

$$S = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

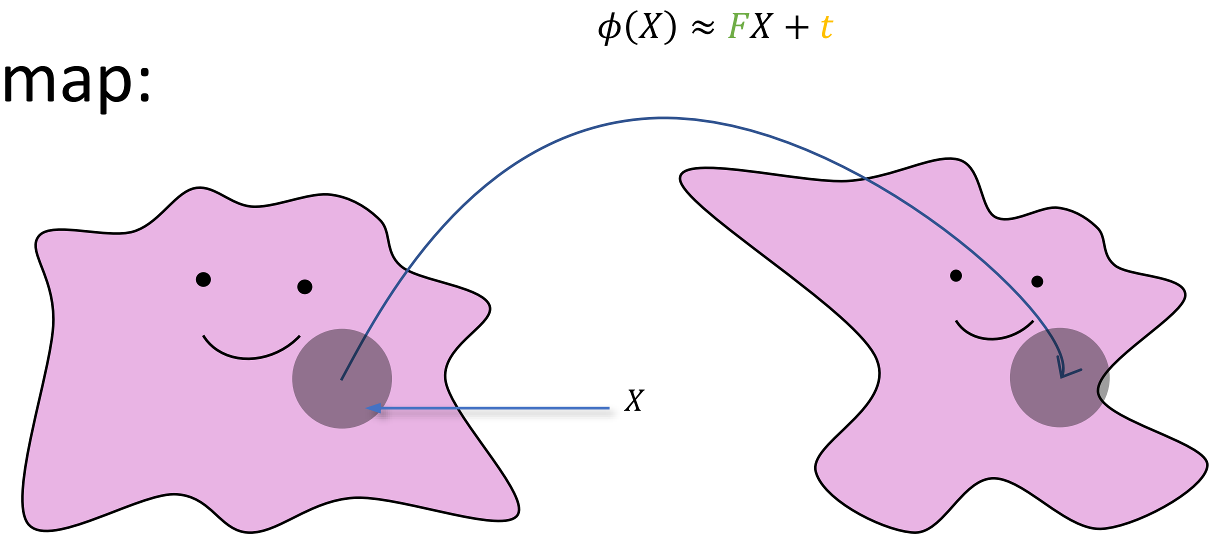
Deformation gradient

- The gradient of the deformation map:
  - $\phi: X \to x$
  - $F = \begin{bmatrix} \partial x_1/\partial X_1 & \partial x_1/\partial X_2 \\ \partial x_2/\partial X_1 & \partial x_2/\partial X_1 \end{bmatrix}$
  - $x \approx FX + t$



Deformation gradient

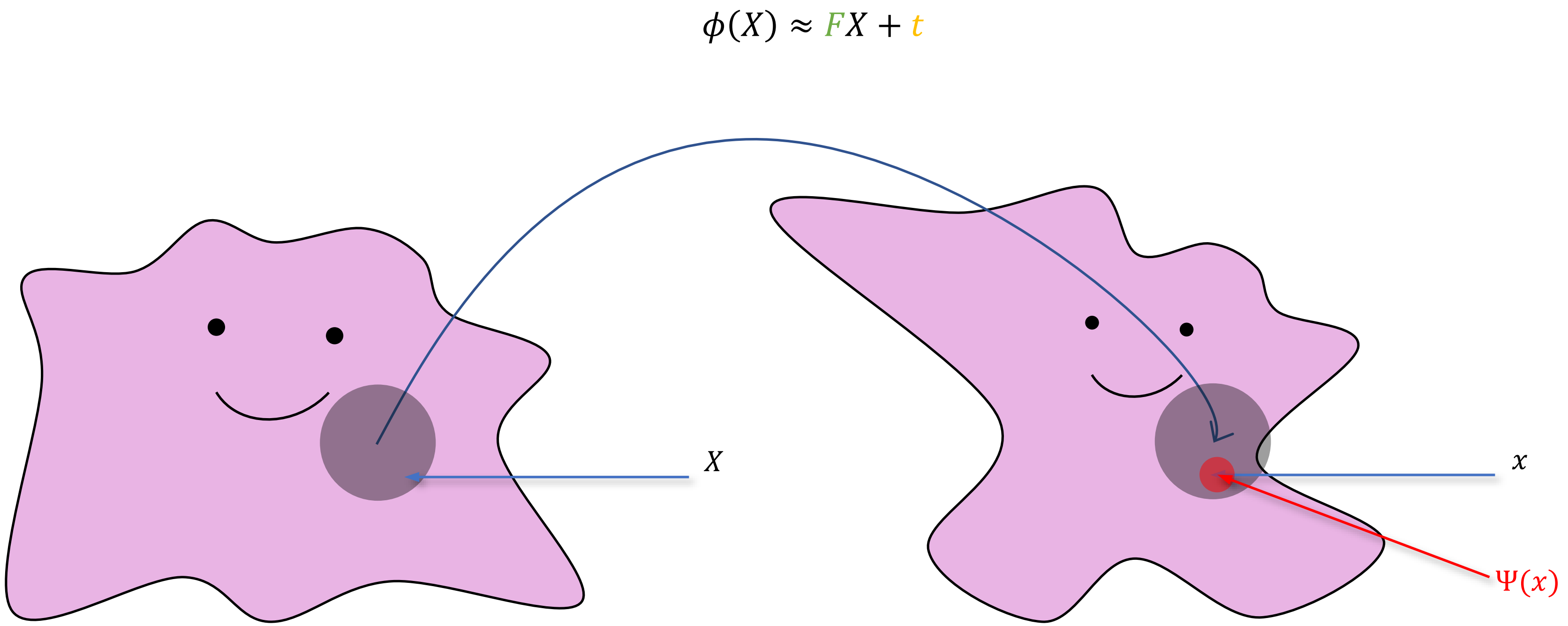
- The gradient of the deformation map:
  - $\phi: X \rightarrow x$
  - $F = \begin{bmatrix} \partial x_1/\partial X_1 & \partial x_1/\partial X_2 \\ \partial x_2/\partial X_1 & \partial x_2/\partial X_1 \end{bmatrix}$
  - $x \approx FX + t$



- A non-rigid deformation gradient shall end up with a non-zero deformation energy.

Energy density: $\Psi(x) = \Psi(\phi(X))$

- Define: $\Psi(x) = \Psi(\phi(X))$ is an energy density function at $x = \phi(X)$



Energy density: $\Psi\big(\phi(X)\big) = \Psi(FX + t)$

- Define: $\Psi(x) = \Psi\big(\phi(X)\big)$ is an energy density function at $x = \phi(X)$
  - Recall that $\phi(X) \approx FX + t$, we have $\Psi(x) \approx \Psi(FX + t)$

Energy density: $\Psi(FX + t) = \Psi(FX)$

- Define: $\Psi(x) = \Psi\big(\phi(X)\big)$ is an energy density function at $x = \phi(X)$
  - Recall that $\phi(X) \approx FX + t$, we have $\Psi(x) \approx \Psi(FX + t)$
  - Since the energy density function should be translational invariant
    - i.e. $\Psi(x) = \Psi(x + t)$

Energy density: $\Psi(FX) = \Psi(F)$

- Define: $\Psi(x) = \Psi\big(\phi(X)\big)$ is an energy density function at $x = \phi(X)$
  - Recall that $\phi(X) \approx FX + t$, we have $\Psi(x) \approx \Psi(FX + t)$

  - Since the energy density function should be translational invariant
    - i.e. $\Psi(x) = \Psi(x + t)$
  - …and X is the state-independent rest-pose (for elastic materials)

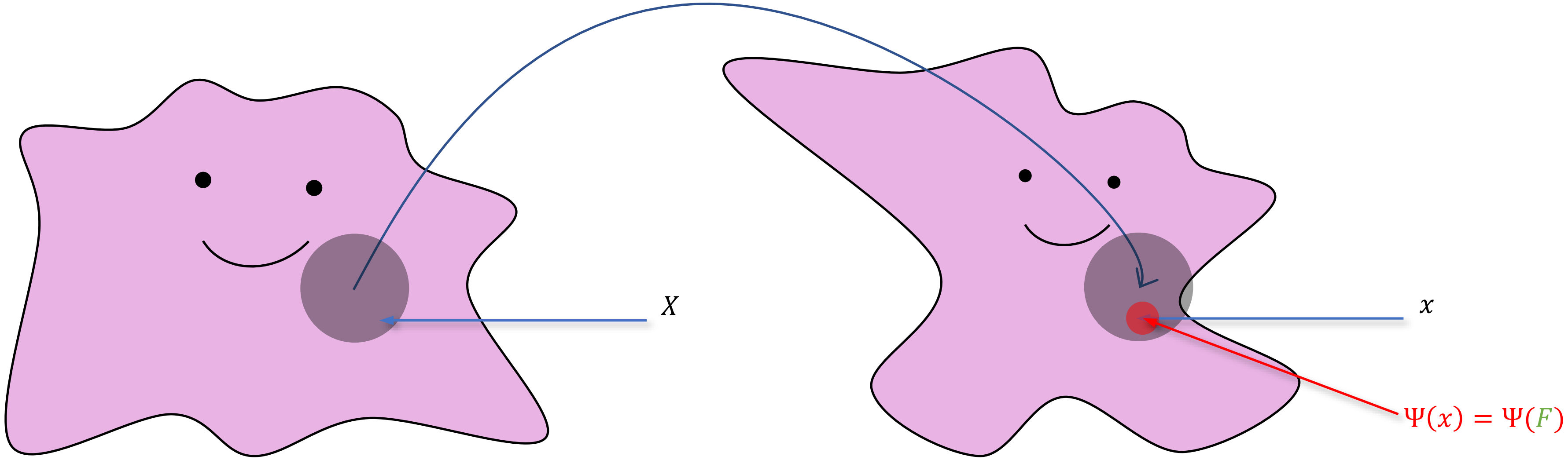
Energy density: $\Psi(x) = \Psi(F)$

- Define: $\Psi(x) = \Psi\big(\phi(X)\big)$ is an energy density function at $x = \phi(X)$
    - Recall that $\phi(X) \approx FX + t$, we have $\Psi(x) \approx \Psi(FX + t)$
    - Since the energy density function should be translational invariant
        - i.e. $\Psi(x) = \Psi(x + t)$
    - …and X is the state-independent rest-pose (for elastic materials)
    - We have $\Psi = \Psi(F)$ being a function of the **local deformation gradient** alone.

Energy density: $\Psi(x) = \Psi(F)$

- Define: $\Psi(x) = \Psi\big(\phi(X)\big)$ is an energy density function at $x = \phi(X)$
  - We have $\Psi = \Psi(F)$ being a function of the **local deformation gradient** alone.

$$\phi(X) \approx FX + t$$



Energy density: $\Psi(x) = \Psi(F)$

- Define: $\Psi(x) = \Psi\big(\phi(X)\big)$ is an energy density function at $x = \phi(X)$
  - We have $\Psi = \Psi(F)$ being a function of the **local deformation gradient** alone.

- What should Ψ look like?

What should Ψ Look like?

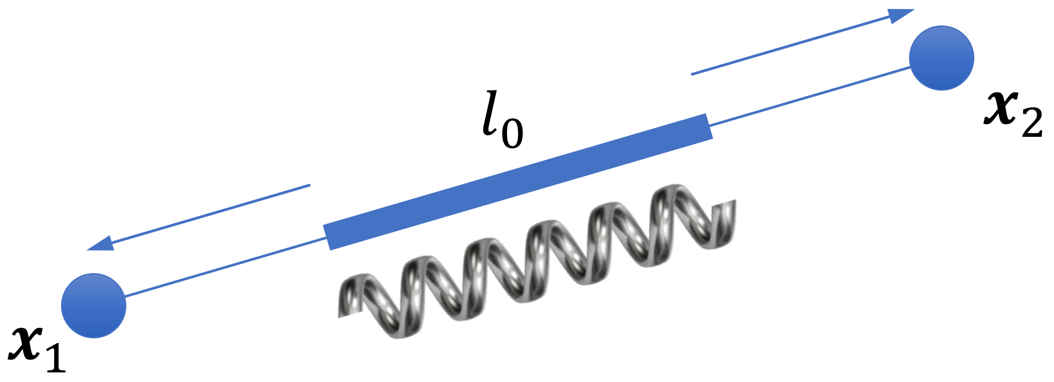
$$\Psi(F) = \frac{1}{2}k\|\mathrm{F} - I\|_F{}^2 \text{ ?}$$

$$\Psi(F) = \frac{1}{2}k\|\mathrm{F}\|_F{}^2 \text{ ?}$$

Note: $\|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2} = \sqrt{tr(A^T A)}$

Deformation gradient is NOT the best quantity to describe **deformation**

- Using the mass-spring system as an analogy:
  - The "deformation gradient" of a spring:
    - $\frac{x_1 - x_2}{l_0}$
  - The "deformation" of a spring:
    - $\left\| \frac{x_1 - x_2}{l_0} \right\| - 1$
    - Translational invariant
    - Rotational invariant
    - Being zero means "no deformation"



We want a descriptor to describe **deformation**

- Strain (tensor): $\epsilon(F)$
  - Descriptor of severity of deformation
  - $\epsilon(I) = 0$
  - $\epsilon(F) = \epsilon(RF)\ for\ \forall\ R \in SO(dim)$

We want a descriptor to describe **deformation**

- Strain (tensor): $\epsilon(F)$
  - Descriptor of severity of deformation
  - $\epsilon(I) = 0$
  - $\epsilon(F) = \epsilon(RF)\ for\ \forall\ R \in SO(dim)$

- Sample strain tensors in different **constitutive models**:
  - St. Venant-Kirchhoff model: $\epsilon(F) = \frac{1}{2}(F^T F - I)$
  - Co-rotated linear model: $\epsilon(F) = S - I, where\ F = RS$
    - Further Reading: (Signed) Polar Decomposition [Link]

What should $\boldsymbol{\Psi}$ Look like?

$$\Psi(\epsilon) = \frac{1}{2}k\|\epsilon\|_F{}^2$$

$$\Psi(\epsilon) = \mu\|\epsilon\|_F{}^2 + \frac{\lambda}{2}tr(\epsilon)^2$$

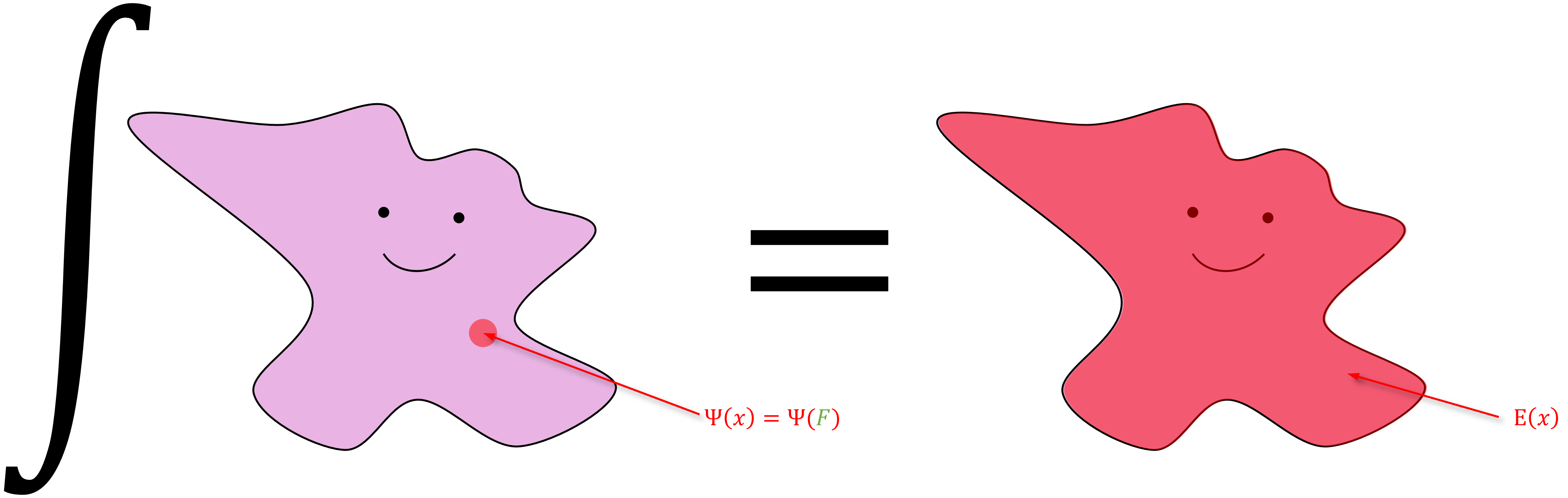
Note: $\|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2} = \sqrt{tr(A^T A)}$

From energy density to energy

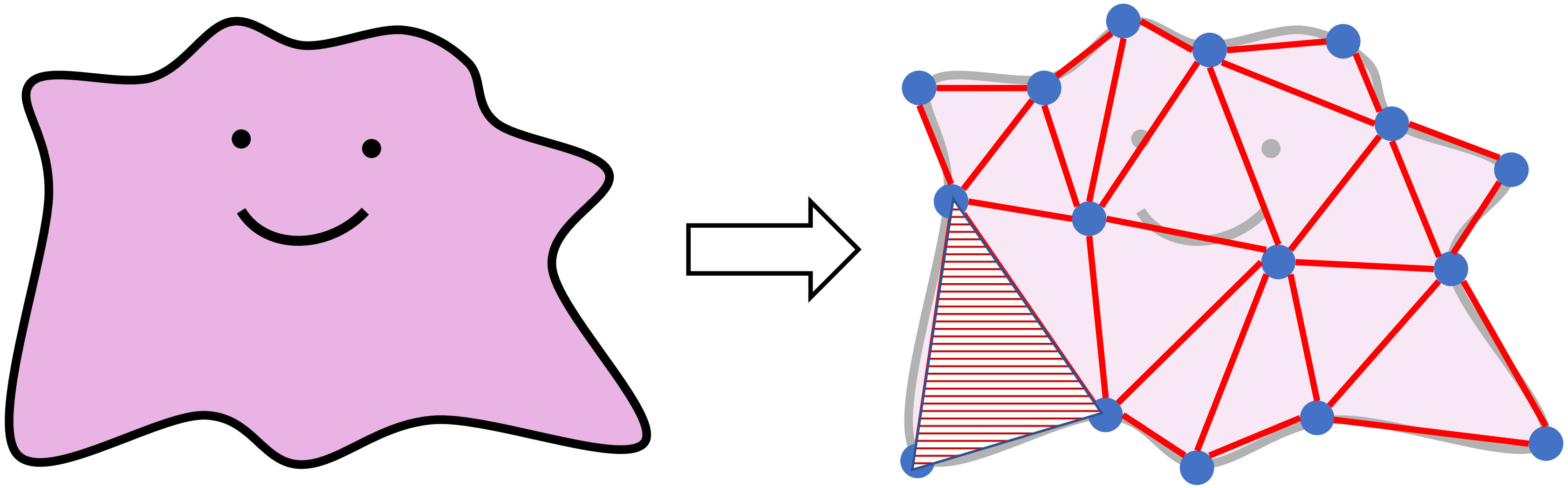
- $E(x) = \int_{\Omega} \Psi(F) dX$
- Spatial Discretization is needed!

From energy density to energy

- $E(x) = \int_{\Omega} \Psi(F) dX$



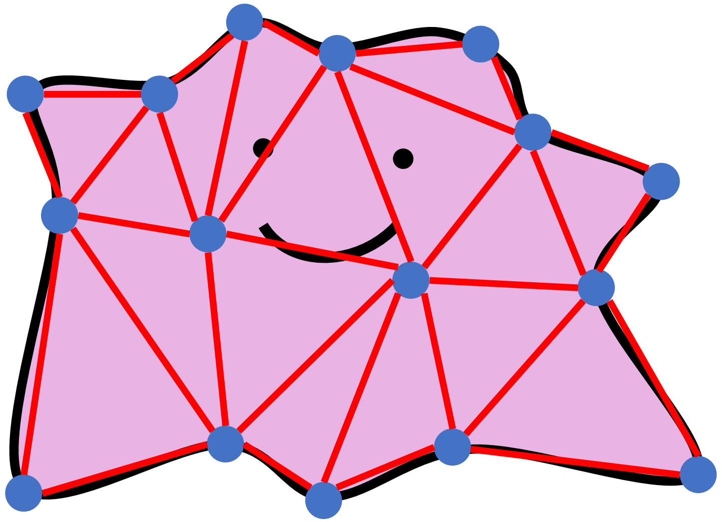
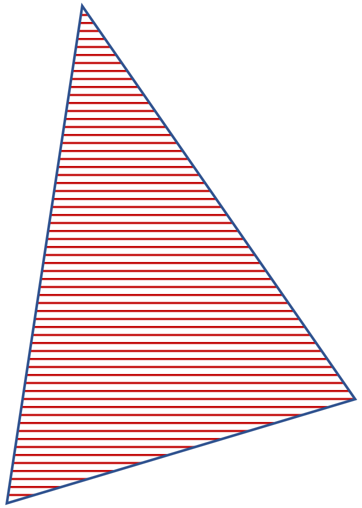
Linear finite element method (FEM)



Linear Element

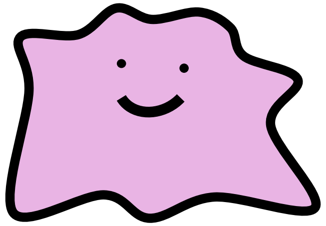
$$\phi(X) = FX + t$$

Linear finite element method (FEM)

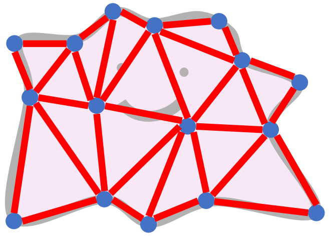
$E = \int$  $= \sum$ 

Linear FEM energy

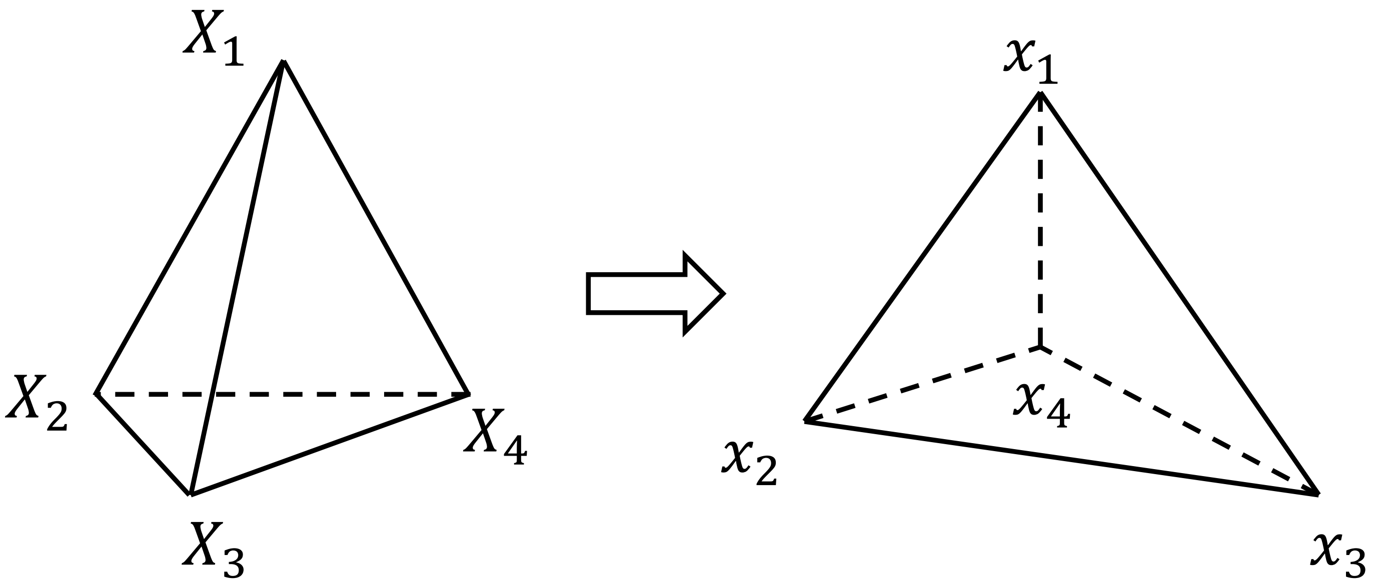
- Continuous Space:
  - $E(x) = \int_{\Omega} \Psi\big(F(x)\big) dX$



- Discretized Space:
  - $E(x) = \sum_{e_i} \int_{\Omega_{\mathrm{e_i}}} \Psi\big(F_i(x)\big) dX = \sum_{e_i} w_i \, \Psi\big(F_i(x)\big)$
  - $w_i = \int_{\Omega_{\mathrm{e_i}}} dX$: size (area/volume) of the i-th element



Linear element: $\phi(X) = FX + t$



$$x_1 = FX_1 + t$$
$$x_2 = FX_2 + t$$
$$x_3 = FX_3 + t$$
$$x_4 = FX_4 + t$$

$$\underbrace{[x_1 - x_4 \quad x_2 - x_4 \quad x_3 - x_4]}_{D_s} = F\underbrace{[X_1 - X_4 \quad X_2 - X_4 \quad X_3 - X_4]}_{D_m}$$

$$F = D_s D_m^{-1}$$

The gradient of $\Psi(F(x))$

- Eventually we will need the gradient of Ψ to run simulations…

- Chain rule: $\frac{\partial \Psi}{\partial x} = \frac{\partial F}{\partial x} : \frac{\partial \Psi}{\partial F}$

In 2D: A $(2n \times 1) \times (2 \times 2)$ tensor A (2×2) tensor (matrix)

A $(2n \times 1)$ tensor (vector)

Note (matrix contraction): $B : A = A : B = \sum_{i,j} A_{ij} B_{ij} = \sqrt{tr(A^T B)}$

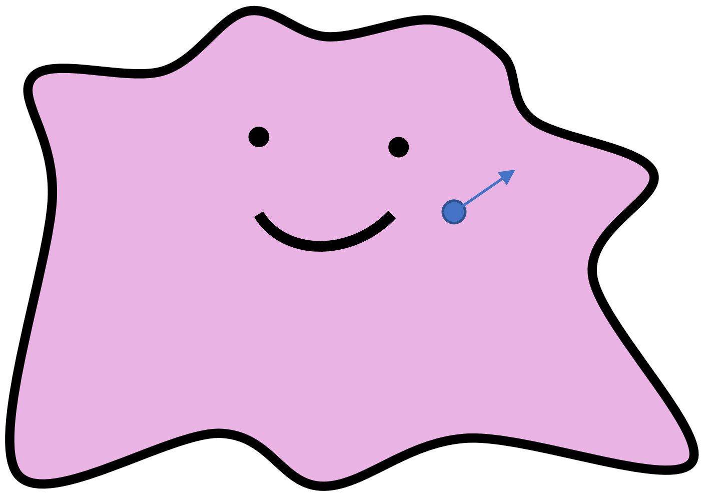
The gradient of $\Psi\big(F(x)\big)$

- Eventually we will need the gradient of Ψ to run simulations…

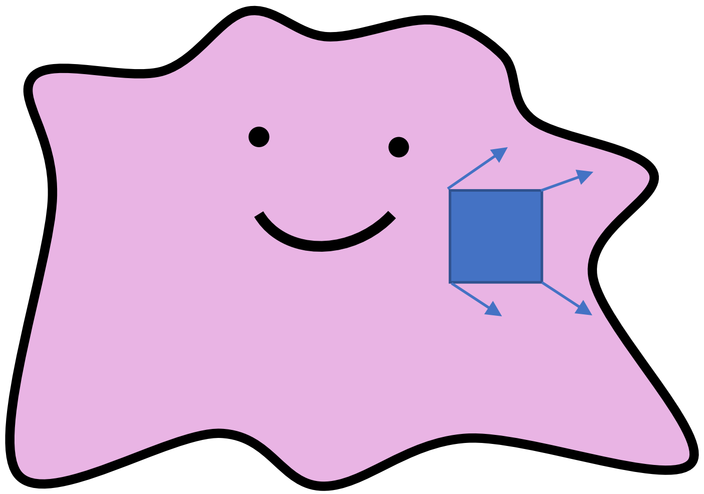
- Chain rule: $\frac{\partial \Psi}{\partial x} = \frac{\partial F}{\partial x} : \frac{\partial \Psi}{\partial F}$

- For hyperelastic materials, the 1st Piola-Kirchhoff stress tensor:
    - $P = \frac{\partial \Psi}{\partial F}$

The 1st Piola-Kirchhoff stress tensor: $P = \frac{\partial \Psi}{\partial F}$



$\frac{\partial \Psi}{\partial x}$



$\frac{\partial \Psi}{\partial F}$

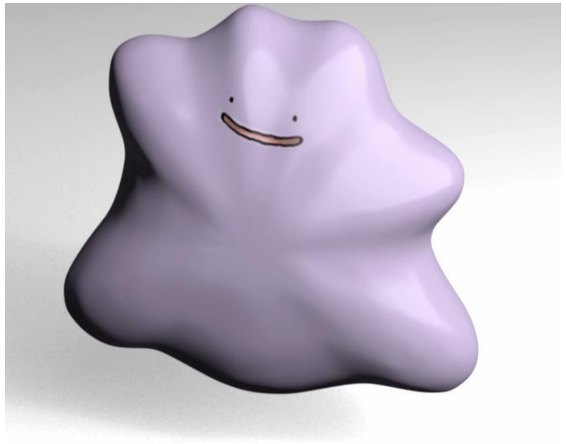
Some 1st Piola-Kirchhoff stress tensors

- St. Venant-Kirchhoff model (StVK):
  - Strain: $\epsilon_{stvk}(F) = \frac{1}{2}(F^T F - I)$
  - Energy density: $\Psi(F) = \mu \left\| \frac{1}{2}(F^T F - I) \right\|_F^{\;2} + \frac{\lambda}{2} tr\left(\frac{1}{2}(F^T F - I)\right)^2$
  - $P = \frac{\partial \Psi}{\partial F} = F\,[2\mu\epsilon_{stvk} + \lambda tr(\epsilon_{stvk})I]$
- Co-rotated linear model:
  - Strain: $\epsilon_c(F) = \mathrm{S} - I, where\ F = RS$
  - Energy density: $\Psi(F) = \mu \| R^T F - I \|_F^{\;2} + \frac{\lambda}{2} tr(R^T F - I)^2$
  - $P = \frac{\partial \Psi}{\partial F} = R[2\mu\epsilon_c + \lambda tr(\epsilon_c)I] = 2\mu(F - R) + \lambda tr(R^T F - I)R$

Linear FEM

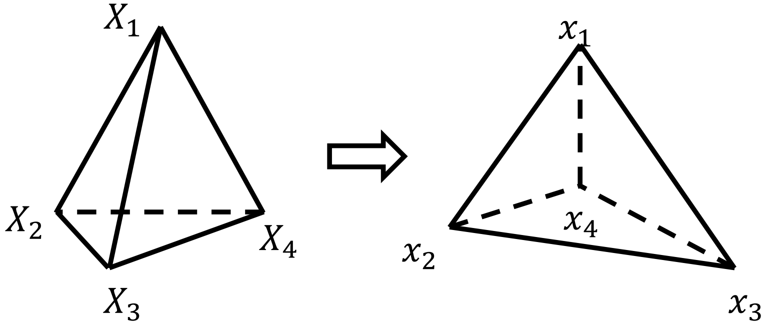
- Elastic energy:
  - $E_i(x) = w_i \Psi(F_i(x))$

- Gradient:
  - $\frac{\partial E_i}{\partial x} = w_i \frac{\partial F_i}{\partial x} : P_i$



Chain rule in detail: $\frac{\partial \Psi}{\partial x_j^{(k)}} = \frac{\partial F}{\partial x_j^{(k)}} : P$

- Let's compute $\frac{\partial F}{\partial x_j^{(k)}}$ first:
  - $j = 1,2,3,4$, stands for the vertex #
  - $k = 1,2,3$, stands for the dimension



Chain rule in detail: $\frac{\partial \Psi}{\partial x_j^{(k)}} = \frac{\partial F}{\partial x_j^{(k)}} : P$

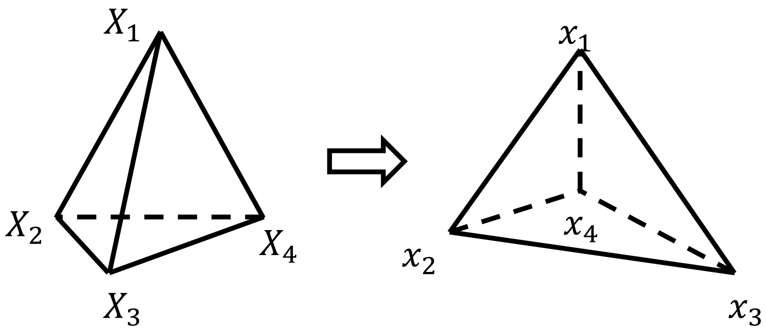
- Let's compute $\frac{\partial F}{\partial x_j^{(k)}}$ first:
  - Since $F = D_s D_m^{-1}$
  - $\frac{\partial F}{\partial x_j^{(k)}} = \frac{\partial D_s}{\partial x_j^{(k)}} D_m^{-1}$
    - Where $\frac{\partial D_s}{\partial x_j^{(k)}} = \delta_k \delta_j^T$, for $j = 1,2,3$
  - Thus: $\frac{\partial F}{\partial x_j^{(k)}} = \delta_k \delta_j^T D_m^{-1}$

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\frac{\partial D_s}{\partial x_1^{(2)}}} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}^T$$

$$\underbrace{[x_1 - x_4 \quad x_2 - x_4 \quad x_3 - x_4]}_{D_s} = F \underbrace{[X_1 - X_4 \quad X_2 - X_4 \quad X_3 - X_4]}_{D_m}$$

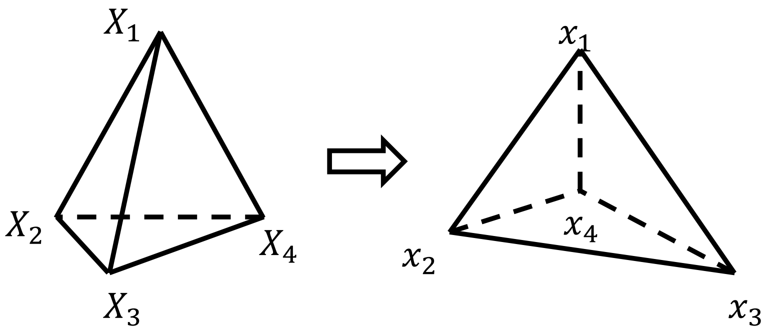
Chain rule in detail: $\frac{\partial \Psi}{\partial x_j^{(k)}} = \frac{\partial F}{\partial x_j^{(k)}} : P$

- $\frac{\partial F}{\partial x_j^{(k)}} : P = \delta_k \delta_j^T D_m^{-1} : P$
- $\frac{\partial F}{\partial x_j^{(k)}} : P = tr\left(D_m^{-T} \delta_j \delta_k^T P\right)$
- $\frac{\partial F}{\partial x_j^{(k)}} : P = tr\left(\delta_k^T P D_m^{-T} \delta_j\right)$
- $\frac{\partial F}{\partial x_j^{(k)}} : P = \delta_k^T P D_m^{-T} \delta_j = [P D_m^{-T}]_{kj}$



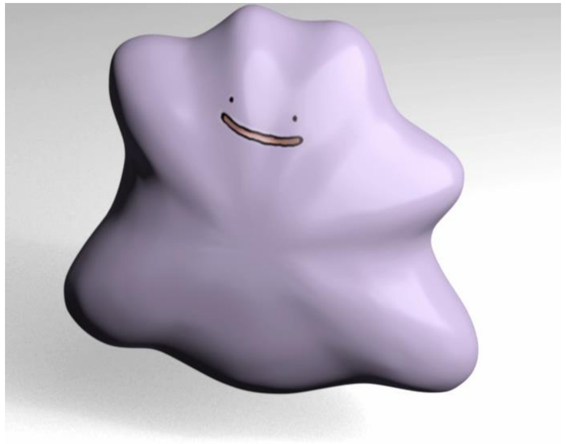
Chain rule in detail: $\frac{\partial \Psi}{\partial x_j^{(k)}} = \frac{\partial F}{\partial x_j^{(k)}} : P$

- $\frac{\partial \Psi}{\partial x_j^{(k)}} = [P D_m^{-T}]_{kj}$
- Thus: $\frac{\partial \Psi}{\partial x_j} =$ the j-th col of $[P D_m^{-T}]$ for j=1,2,3
- $\frac{\partial \Psi}{\partial x_4} = -\sum_{j=1}^{3} \frac{\partial \Psi}{\partial x_j}$
- $\frac{\partial E_i}{\partial x_j} = w_i \frac{\partial E_i}{\partial x_j}$



Linear FEM

- Elastic energy:
  - $E_i(x) = w_i \Psi(F_i(x))$
- Gradient:
  - $\frac{\partial E_i}{\partial x} = w_i \frac{\partial F_i}{\partial x} : P_i$
- Further Readings:
  - *Finite Element Method, Part I* [Link]
  - Or using auto-diff in Taichi [Link]



Linear FEM (an example)

```
# gradient of elastic potential
for i in range(N_triangles):
    Ds = compute_D(i)
    F = Ds@elements_Dm_inv[i]
    # co-rotated linear elasticity
    R = compute_R_2D(F)
    Eye = ti.Matrix.cols([[1.0, 0.0], [0.0, 
1.0]])
    # first Piola-Kirchhoff tensor
    P = 2*LameMu[None]*(F-R) + 
LameLa[None]*((R.transpose())@F-Eye).trace()*R
    #assemble to gradient
    H = elements_V0[i] * P @ 
(elements_Dm_inv[i].transpose())
    a,b,c = 
triangles[i][0],triangles[i][1],triangles[i][2]
    gb = ti.Vector([H[0,0], H[1, 0]])
    gc = ti.Vector([H[0,1], H[1, 1]])
    ga = -gb-gc
    grad[a] += ga
    grad[b] += gb
    grad[c] += gc
```

Compute gradient

```
# symplectic integration
acc = -grad[i]/m - ti.Vector([0.0, g])
v[i] += dh*acc
x[i] += dh*v[i]
```

Time integration

Linear FEM using autodiff (an example)

```
@ti.kernel
def compute_total_energy():
    for i in range(N_triangles):
        Ds = compute_D(i)
        F = Ds @ elements_Dm_inv[i]
        # co-rotated linear elasticity
        R = compute_R_2D(F)
        Eye = ti.Matrix.cols([[1.0, 0.0], [0.0, 1.0]])
        element_energy_density = LameMu[None]*((F-
R)@(F-R).transpose()).trace() +
0.5*LameLa[None]*(R.transpose()@F-Eye).trace()**2

        total_energy[None] += element_energy_density *
elements_V0[i]
```

Compute energy

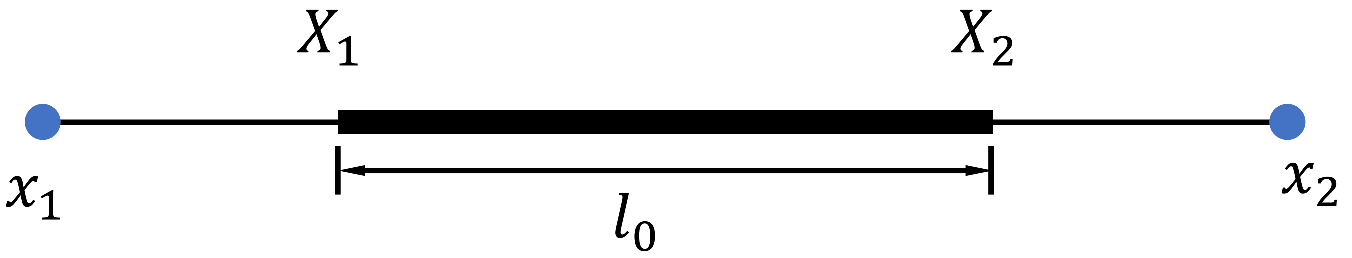
```
if using_auto_diff:
    total_energy[None]=0
    with ti.Tape(total_energy):
        compute_total_energy()
else:
    compute_gradient()
```

Compute gradient

```
# symplectic integration
acc = -x.grad[i]/m - ti.Vector([0.0, g])
v[i] += dh*acc
x[i] += dh*v[i]
```

Time integration

Revisit the mass-spring system



Deformation gradient: $F = D_s D_m^{-1} = \frac{x_1 - x_2}{X_1 - X_2} = \frac{x_1 - x_2}{l_0}$

Deformation strain: $\epsilon = \|F\| - 1$

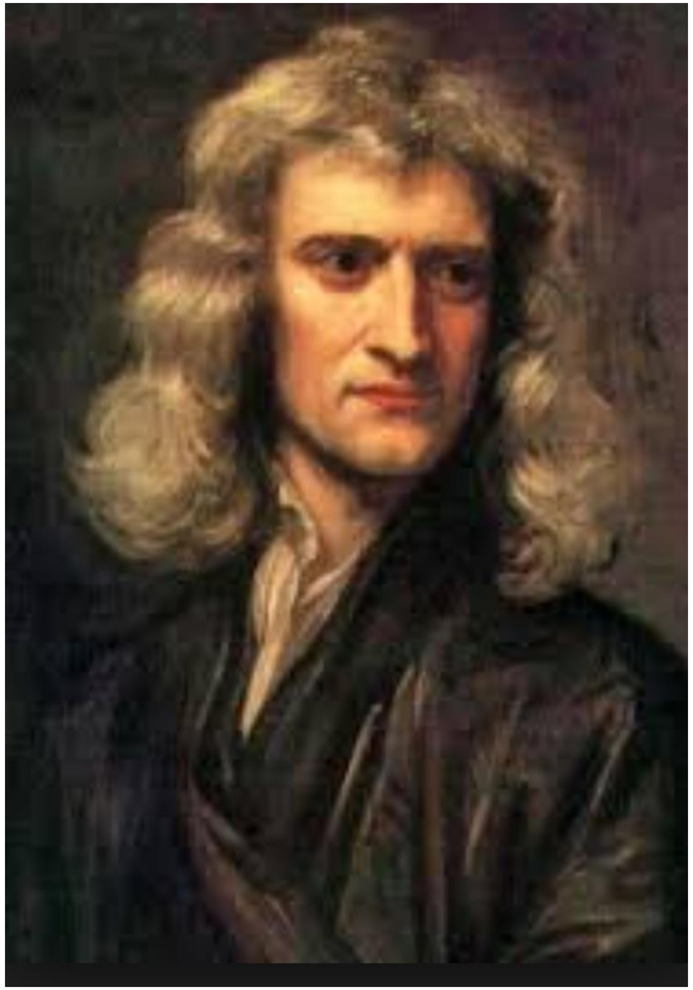
Energy density: $\Psi = \mu\epsilon^2 = \mu\left(\left\|\frac{x_1 - x_2}{l_0}\right\| - 1\right)^2$

Energy: $\mathrm{E} = l_0\Psi = \frac{1}{2}\frac{2\mu}{l_0}l_0^2\epsilon^2 = \frac{1}{2}k(\|x_1 - x_2\| - l_0)^2$

Remark

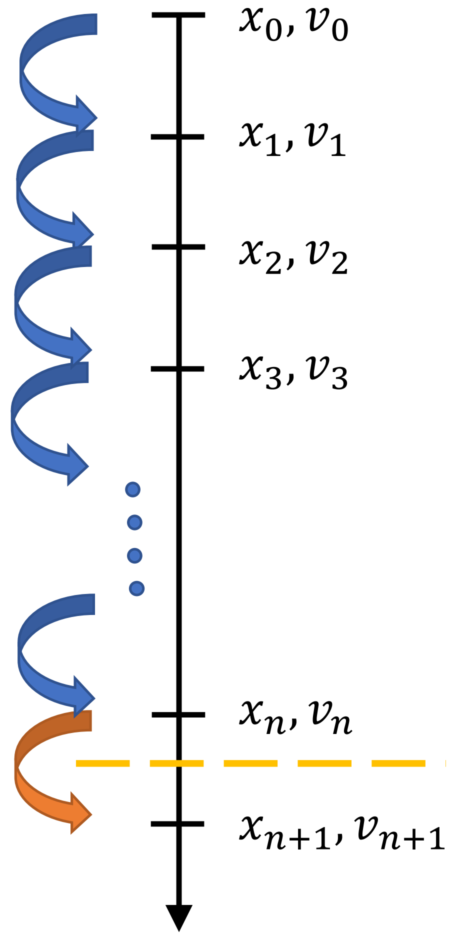
Remark

- Laws of physics
  - Equations of motion
- Integration in time
- Integration in space
  - A simple (but useful) model: mass-spring system
  - Constitutive models
  - The finite element method



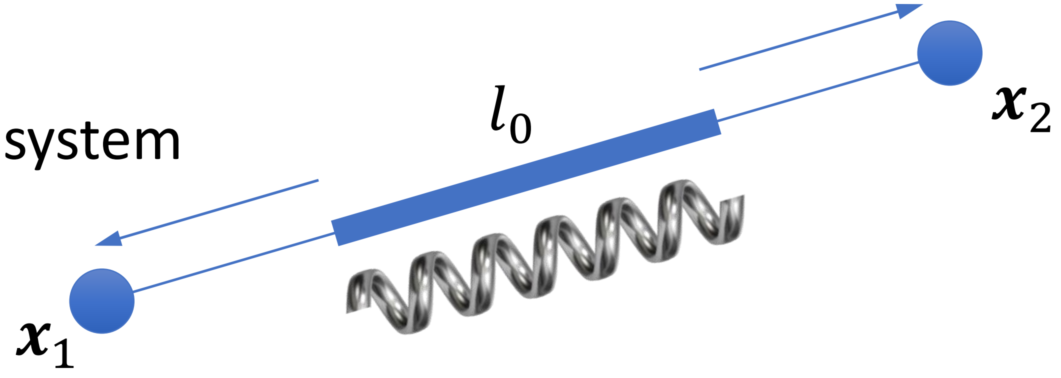
Remark

- Laws of physics
    - Equations of motion
- Integration in time
    - Numerical quadrature
- Integration in space
    - A simple (but useful) model: mass-spring system
    - Constitutive models
    - The finite element method



Remark

- Laws of physics
  - Equations of motion
- Integration in time
  - Numerical quadrature
- Integration in space
  - A simple (but useful) model: mass-spring system
  - Constitutive models
  - The finite element method



Remark

- Laws of physics
  - Equations of motion
- Integration in time
  - Numerical quadrature
- Integration in space
  - A simple (but useful) model: mass-spring system
  - Constitutive models
  - The finite element method

$$\phi$$

$$F$$

$$\epsilon$$

$$\Psi\big(\epsilon(F)\big)$$

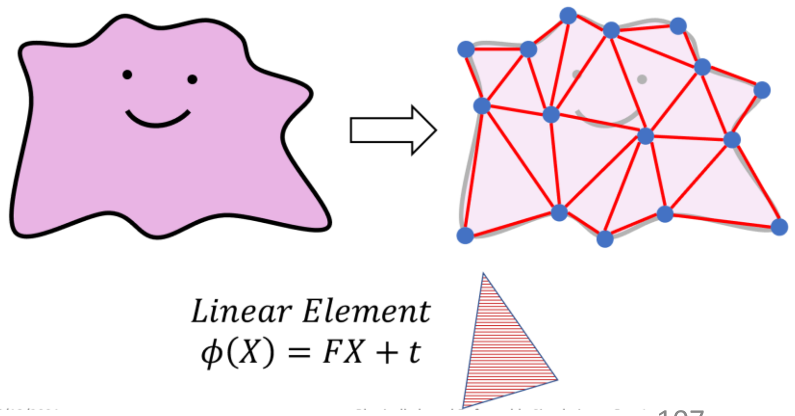
$$P$$

$$E = \int \Psi$$

$$f = -\frac{\partial E}{\partial x}$$

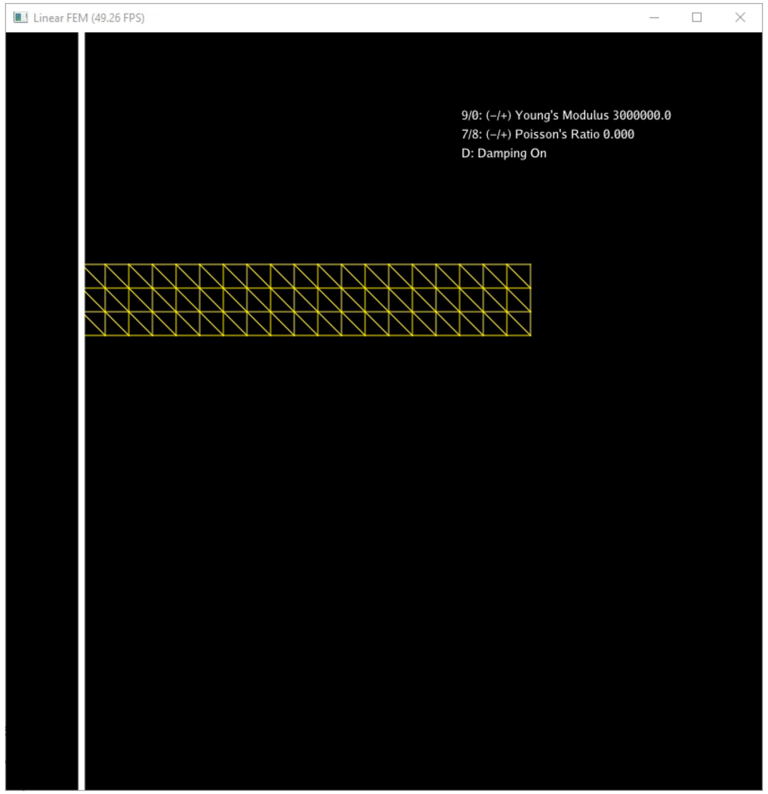
Remark

- Laws of physics
  - Equations of motion
- Integration in time
  - Numerical quadrature
- Integration in space
  - A simple (but useful) model: mass-spring system
  - Constitutive models
  - The finite element method



Remark

- Laws of physics
  - Equations of motion
- Integration in time
  - Numerical quadrature
- Integration in space
  - A simple (but useful) model: mass-spring system
  - Constitutive models
  - The finite element method



Further readings

- *Real Time Physics, Chapter 3,4* [SIGGRAPH 2008 Course] [Link]
- *Finite Element Method, Part I* [SIGGRAPH 2012 Course] [Link]
- *Dynamic Deformables: Implementation and Production Practicalities* [SIGGRAPH 2020 Course] [Link]

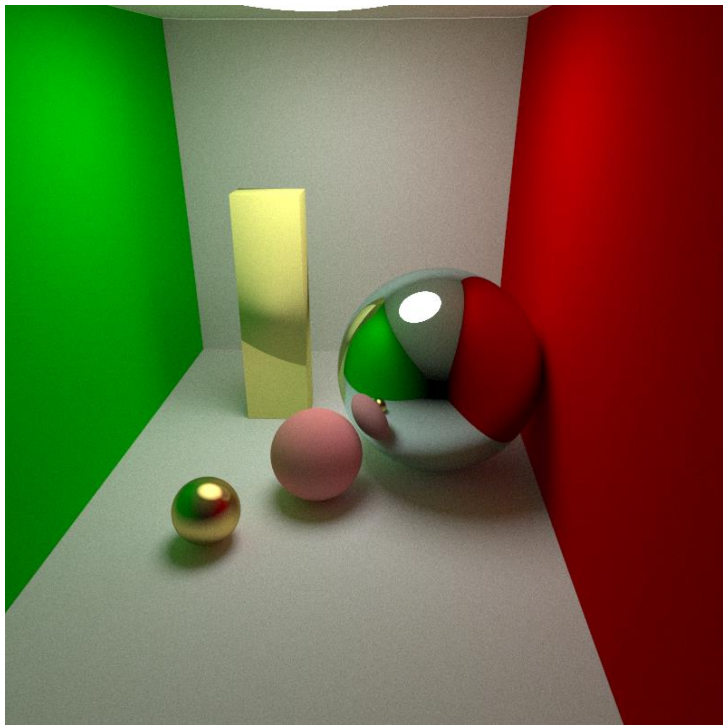
Homework

Homework Today

- Download the repo (--Deformables):
  - https://github.com/taichiCourse01/--Deformables

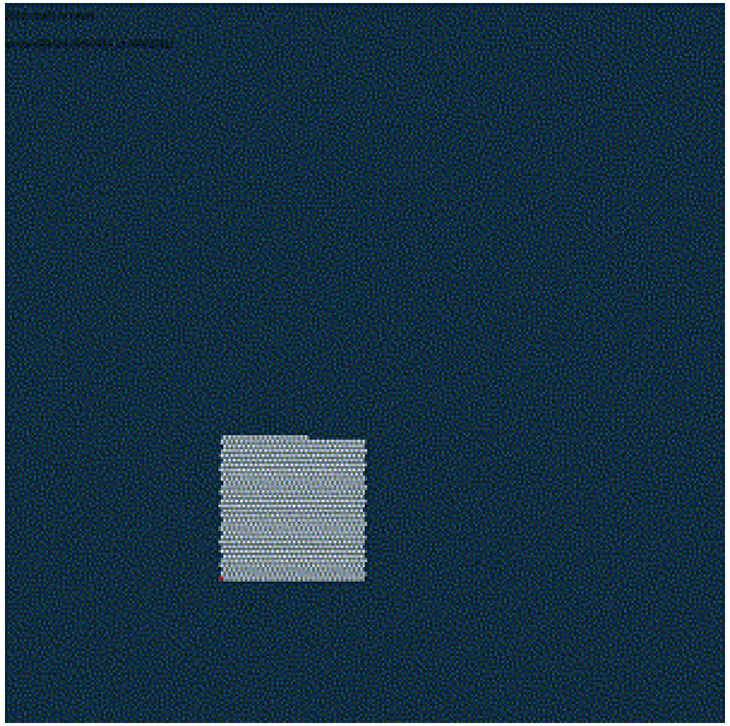
- Try:
  - Changing your time integration scheme from explicit Symplectic Euler to forward Euler (in both --Galaxy and --Deformables)
  - Changing your material model from the corotated Linear model to the StVK model
  - Weave a different 2D/3D structure (other than a cantilever) and simulate it using either the mass-spring model or the linear FEM model

Excellent homework assignments



[@Huanghongru]

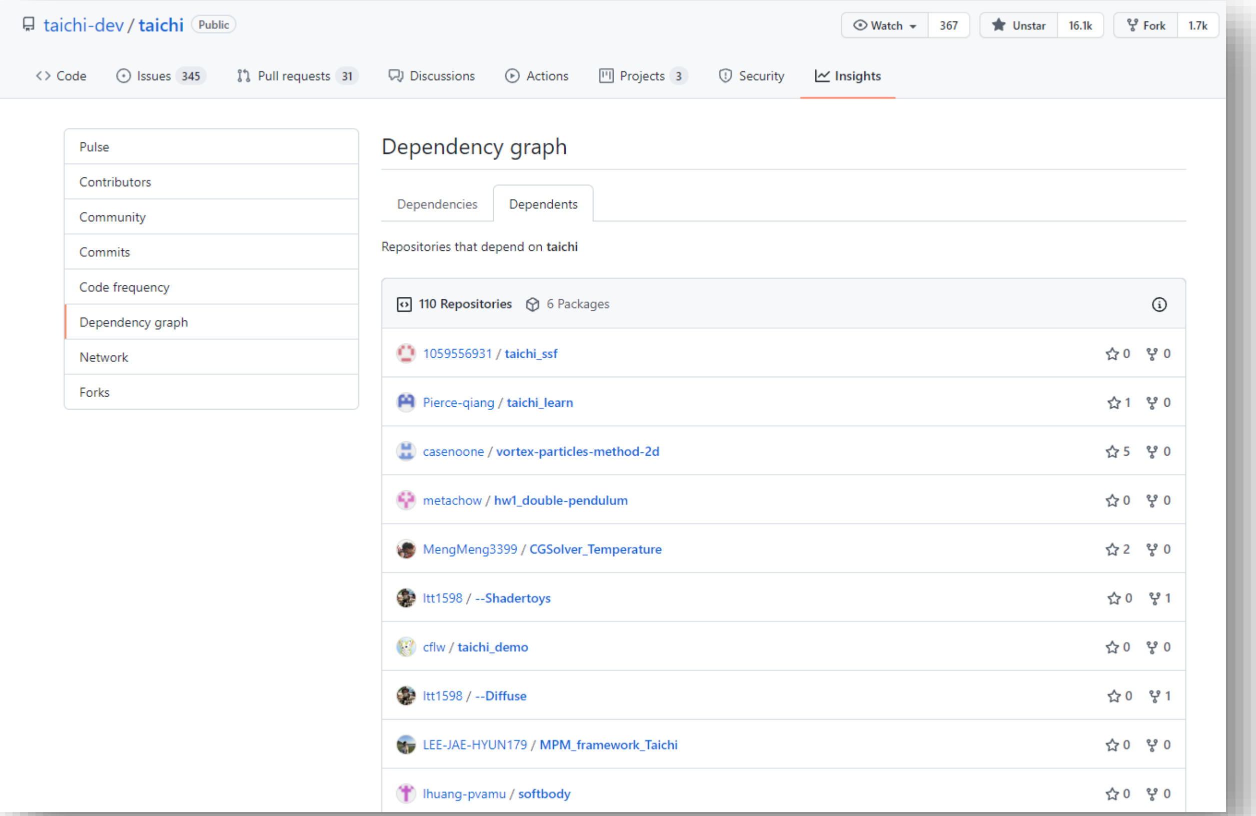
[@cflw]

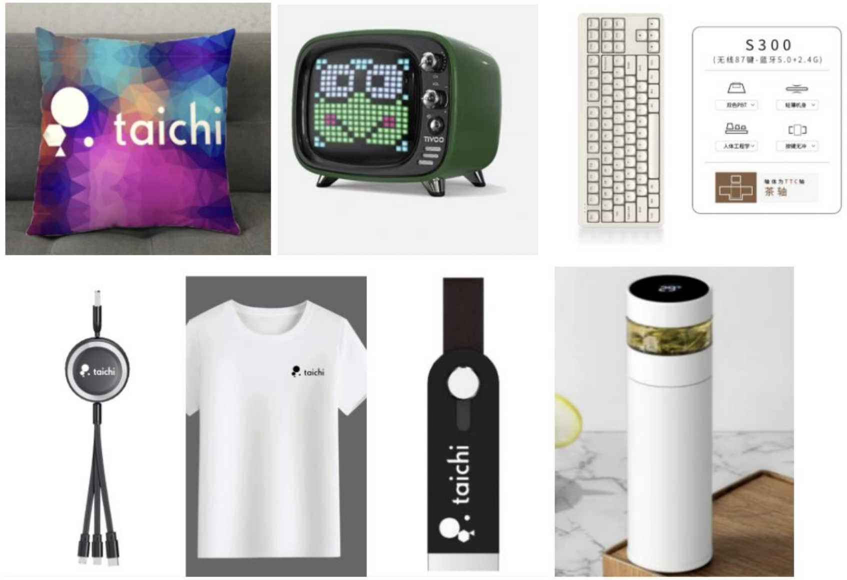


[@chunleili]

Gifts for the gifted

- Use Template for your homework
- Next check Dec. 14, 2021





Questions?

本次答疑：11/18 🡐作业分享也在这里

下次直播：11/23

直播回放：Bilibili 搜索「太极图形」

主页&课件：https://github.com/taichiCourse01

主页&课件(backup)：https://docs.taichi.graphics/tgc01