Physics Simulation in Visual Computing

Lecture Notes

Additional Resources

- Comprehensive tutorial about everything (and more) in the fluid part: “Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids”
- Course about continuous model and FEM in deformable solids: “FEM Simulation of 3D Deformable Solids: A practitioner’s guide to theory, discretization and model reduction”

1. Introduction

▼ *Notations and Definitions*

- $\mathbf{a} = \dot{\mathbf{v}} = \ddot{\mathbf{x}}$
- *Ordinary Differential Equation (ODE):* describes relationship between the rate of change of a quantity and the quantity itself

$$y^{(n)}(x) = f(x, y(x), y'(x), \dots, y^{(n-1)}(x))$$

- Spring force with spring stiffness k (*Hooke's Law*):

$$F(t) = -kx(t)$$

We differentiate between *kinematics* and *dynamics* as

- **kinematics** (study of *how* objects move) focuses on describing the motion of objects *without* considering the forces causing the motion, while
- **dynamics** (study of *why* objects move) involves the study of the forces and their effects on the motion,

providing a complete understanding of both the *cause* and *effect* in physical systems.

1.1. Particle Dynamics

In a *dynamics* simulation the motion is determined by forces acting on the bodies. The relationship between the motion and the forces are described using the three *laws of motion*.

Newton's Laws of Motion

1st Law: *"A body changes its state of motion only if external forces are acting on the body."*

$$\mathbf{F} = \sum_i \mathbf{F}_i \stackrel{!}{=} \mathbf{0}$$
$$\Leftrightarrow \quad \frac{D\mathbf{v}}{Dt} = \mathbf{0}$$

2nd Law: *"The change of the state of motion (acceleration) is proportional to the acting force and has the same direction."*

$$\mathbf{F} = m\mathbf{a} = m\dot{\mathbf{v}} = m\ddot{\mathbf{x}}$$

3rd Law: *"For every action (force) in nature there is an equal and opposite reaction." (actio = reactio)*

Particle

A *particle* is a simple body defined as a point in space with mass, position, velocity and *no rotation* (→ 3-DOF). The motion of a particle can be written as a second-order or two first-order **ODEs**:

$$\ddot{\mathbf{x}} = \frac{1}{m}F(\mathbf{x}, \dot{\mathbf{x}}, t) \quad \Leftrightarrow \quad \dot{\mathbf{v}} = \frac{1}{m}F(\mathbf{x}, \mathbf{v}, t),\ \dot{\mathbf{x}} = \mathbf{v}$$

If we assume that we know the initial position $\mathbf{x}(t_0)$ and velocity $\mathbf{v}(t_0)$, we want to solve a so-called **Initial Value Problem** to get a new position and velocity:

$$\dot{\phi}(t) = \begin{pmatrix} \dot{\mathbf{x}}(t) \\ \dot{\mathbf{v}}(t) \end{pmatrix},\ \phi(t_0) = \begin{pmatrix} \mathbf{x}(t_0) \\ \mathbf{v}(t_0) \end{pmatrix} \overset{solution}{\rightarrow} \phi(t) = \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{pmatrix}$$

Usually, we cannot find analytical solutions, so we solve these problems numerically.

1.2. Discrete Model

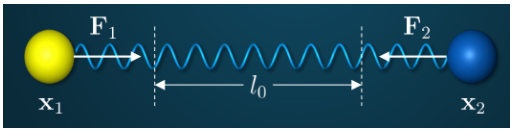
In a *discrete* model, we assume a *particle system* of *finitely many* particles.

- **Elasticity** is the property of a material to deform if a force is acting on the material and to return to its original size and shape if the force is removed.
- *External forces* act from the outside on the simulated system. *Internal forces* act between the single particles in the system. Since the momentum conservation must be guaranteed, the sum of all internal forces must be zero 0.

Spring Force with $\mathbf{F}_1 + \mathbf{F}_2 = 0$:

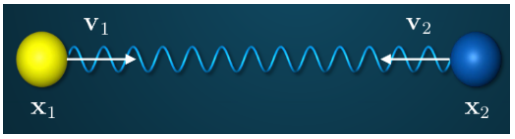
$$\mathbf{F}_{1/2} = \pm \underbrace{k(||\mathbf{x}_2 - \mathbf{x}_1|| - l_0)}_{\Delta l} \underbrace{\frac{\mathbf{x}_2 - \mathbf{x}_1}{||\mathbf{x}_2 - \mathbf{x}_1||}}_{\text{direction } \mathbf{d}}$$

$$\text{magnitude } ||F||$$



Damping Force with $\mathbf{F}_1 + \mathbf{F}_2 = 0$:

$$v_{rel} = (\mathbf{v_2} - \mathbf{v_1})^T \frac{\mathbf{x}_2 - \mathbf{x}_1}{||\mathbf{x}_2 - \mathbf{x}_1||}$$
$$\mathbf{F}_{1/2}^{\text{damping}} = \pm \mu v_{rel} \frac{\mathbf{x}_2 - \mathbf{x}_1}{||\mathbf{x}_2 - \mathbf{x}_1||}$$



(damping force should only damp the *relative* velocity *in direction of the spring*)

Mass-Spring Model

An elastic body given by a mesh (interior also included) is simulated by creating a *particle at each vertex* and a *spring on each edge*. The model is discrete, since the body is represented by a discrete set of particles and springs.

Simulation Step

For each particle i:

Determine sum of all external forces $\mathbf{F}_i^{\text{ext}}$

Determine sum of all spring forces acting on the particle $\mathbf{F}_i^{\text{int}}$

Compute new state for $t \leftarrow t + \Delta t$ by numerically solving the *equation of motion*

$$\dot{\mathbf{x}}_i = \mathbf{v}_i \rightarrow \mathbf{x}(t)$$
$$\dot{\mathbf{v}}_i = \frac{1}{m_i}(\mathbf{F}_i^{\text{int}} + \mathbf{F}_i^{\text{ext}}) \rightarrow \mathbf{v}(t)$$

Problems:

- Mass is concentrated at the particles, there is no mass between them.

- Elastic forces can only act on the edges and in the direction of the edges (reaction might not be as expected).
- Material behavior (e.g. stiffness parameters, masses) depends on the mesh (-resolution).

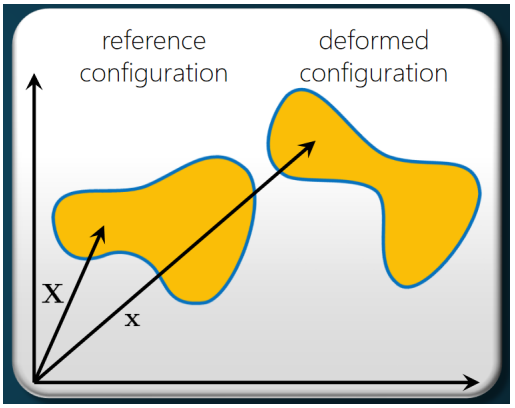
1.3. Continuous Model

Using the continuum theory, we can represent an object continuously by an *infinite* number of particles with *infinitesimal* volume. If we solving the spatial integrals we still require to discretize the domain. But, the discretization of a continuous model carries the field quantities (e.g. velocities), *such that the material behavior isn't influenced by the discretization*, unlike the discretization of a discrete model.

Well-known discretization techniques are the ***Finite Element Method (FEM)*** and ***Smoothed Particle Hydrodynamics (SPH)***.

Quantities:

$\mathbf{X}$	positions at time t_0 defining reference configuration
$\mathbf{x}(\mathbf{X}, t)$	deformed position at time t
$\rho(\mathbf{X})$	mass density [mass/V] - *constant* over time - $m = \int_\Omega \rho(\mathbf{X}) d\mathbf{X}$ for body Ω
$\mathbf{f}(\mathbf{X})$	force density [force/V] - $\mathbf{F}(\mathbf{X}) = \int_\Omega \mathbf{f}(\mathbf{X}) d\mathbf{X}$ for body Ω



Conservation Law of Linear Momentum

(*Linear momentum*: we describe the change in momentum via time-derivative of momentum m*v)

The *conservation law of linear momentum* can be interpreted as a generalization of *Newton's second law of motion* for continua and is also often called the **equation of motion**. It states that the rate of change of momentum of a material particle is equal to the sum of all internal and external volume forces acting on the particle, i.e

$$\rho \frac{D\mathbf{v}}{Dt} = \underbrace{\nabla \cdot \mathbf{T}}_{\mathbf{f}^{\text{int}}} + \mathbf{f}^{\text{ext}}$$

- $\mathbf{T}$: *stress tensor*, encoding material behavior in form of internal forces as a result of stress
- $\frac{D(\cdot)}{Dt}$: *material derivative*, time rate of change of a field quantity at a material point; explicit form depends in the type of coordinates used to describe the system (Eulerian vs. *Lagrangian)*; we use the Lagrangian formulation in the following, so $\frac{D(\cdot)}{Dt} = \frac{\partial(\cdot)}{\partial t}$

This is the equation we want to solve to get the state of our simulated objects for the next time step.

2. Mathematical Foundations

2.0. Notations

- **Einstein's Summation Convention:** The sum over each index appearing exactly twice (*dummy index*) can be neglected. *Free indices* appear only once.

$$s = a_{ij}x_i y_j = \sum_i \sum_j a_{ij}x_i y_j, \qquad y_i = a_{ij}x_j$$

- **Kronecker Delta:** Convenience function to define the identity matrix.

$$[\delta_{ij}] = I_n, \qquad \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

2.1. Tensors

Remember:

- **Trace**: $\mathrm{tr}\mathbf{T} = T_{ii}$
- **Dyadic Product** (= outer product $\otimes$): $[\mathbf{a}][\mathbf{b}]^T$ with $(\mathbf{ab})_{ij} = a_i b_j$

A **tensor** can be generally be seen as an arbitrary dimensional "array", where the *second-oder tensor* (or simply tensor) behaves like a matrix.

- Tensor entries are always defined with respect to a set of *orthorgonal* (= orthonormal) base vectors.
- In particular, a tensor can be seen as a *rotation matrix*, since it is orthogonal.

Coordinate Systems

To transform a vector $[\mathbf{a}]$ to $[\mathbf{a}]$' we need to apply the *transpose* of the tensor $[\mathbf{Q}]$:

$$[\mathbf{a}]' = [\mathbf{Q}]^T[\mathbf{a}], \qquad a'_i = Q_{mi}a_m$$

To transform a tensor $[\mathbf{T}]$, we need to apply the tensor $[\mathbf{Q}]$ on both sides:

$$[\mathbf{T}]' = [\mathbf{Q}]^T[\mathbf{T}][\mathbf{Q}], \qquad T'_{ij} = Q_{mi}Q_{nj}T_{mn}$$

Antisymmetry

Antisymmetry Definition: $\mathbf{T} = -\mathbf{T}^T$, $T_{ij} = -T_{ji}$

A tensor can be uniquely decomposed into a symmetric and an antisymmetric tensor:

$$\mathbf{T} = \mathbf{T}^S + \mathbf{T}^A, \qquad \mathbf{T}^S = \frac{\mathbf{T} + \mathbf{T}^T}{2}, \qquad \mathbf{T}^A = \frac{\mathbf{T} - \mathbf{T}^T}{2}$$

Usually, the diagonal elements of an antisymmetric tensor are zero, so we only have to keep track of 3 values. For each antisymmetric tensor $\mathbf{T}$ there exists a vector $\mathbf{t}^A$ (**dual vector**) s.t.

$$\mathbf{Ta} = \mathbf{t}^A \times \mathbf{a}$$

Eigendecomposition

λ and $\mathbf{a}$ are eigenvalue and the corresponding eigenvector *(unit length)* of tensor $\mathbf{T}$ if $\mathbf{T}\mathbf{a} = \lambda\mathbf{a}$.

The eigenvalues are determined as the *roots* of the *characteristic equation* resulting from $|\mathbf{T} - \lambda\mathbf{I}| = 0$.

The eigenvector is determined by using the corresponding eigenvalue λ to solve $(\mathbf{T} - \lambda\mathbf{I})\mathbf{a} = \mathbf{0}$ for non-trivial solutions.

Theorem: *All eigenvalues of a real symmetric tensor are real*

Hence, a real symmetric tensor has at least three real eigenvectors (called **principal directions**) with corresponding eigenvalues (called **principal values**). In particular, all principal directions are *perpendicular*.

We can use the base spanned by the principal directions $\mathbf{n}_i$, s.t. the tensor is expressed as a simple diagonal matrix of the eigenvalues:

$$[\mathbf{T}] = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}_{\mathbf{n}_i}$$

This is in particular useful, if we e.g. take the tensor to a high power.

Tensor Functions

$$\frac{d}{dt}(\mathbf{T}\mathbf{a}) = \frac{d\mathbf{T}}{dt}\mathbf{a} + \mathbf{T}\frac{d\mathbf{a}}{dt}$$
$$\frac{d}{dt}(\mathbf{T}^T) = \left(\frac{d\mathbf{T}}{dt}\right)^T$$

Scalar Field

The change in a scalar field $\phi(\mathbf{r})$ is defined as the *scalar*

$$d\phi = \phi(\mathbf{r} + d\mathbf{r}) - \phi(\mathbf{r}) = \nabla\phi \cdot d\mathbf{r}$$

Vector Field

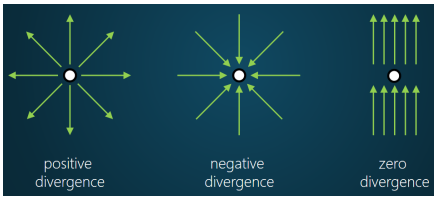
The change on a vector field $\mathbf{v}(\mathbf{r})$ is defined as the *tensor*

$$d\mathbf{v} = \mathbf{v}(\mathbf{r} + d\mathbf{r}) - \mathbf{v}(\mathbf{r}) = (\nabla\mathbf{v})d\mathbf{r}$$

Operators

Divergence of vector field:

$$\text{div } \mathbf{v} \equiv \text{tr}(\nabla\mathbf{v}), \qquad \text{div } \mathbf{v} = \frac{\partial v_i}{\partial x_i}$$



Divergence of tensor field:

$$(\text{div } \mathbf{T}) \cdot \mathbf{a} \equiv \text{div } (\mathbf{T}^T\mathbf{a}) - \text{tr}(\mathbf{T}^T\nabla\mathbf{a}), \qquad \text{div } \mathbf{T} = \left(\frac{\partial \mathbf{T}_{ij}}{\partial x_j}\right)\mathbf{e}_i$$

Curl of vector field describing the inifitesimal rotation of a 3D vector field, with $\mathbf{t}^A$ being the dual vector of $(\nabla \mathbf{v})^A$:

$$\operatorname{curl} \mathbf{v} \equiv 2\mathbf{t}^A$$

Laplacian of a scalar field:

$$\nabla^2 f = \operatorname{div}(\nabla f) = \operatorname{tr}(\nabla(\nabla f)), \qquad \nabla^2 f = \frac{\partial^2 f}{\partial x_i \partial x_i}$$

If we define the ∇-operator as $\nabla = (\frac{\partial}{\partial x_1} \cdots \frac{\partial}{\partial x_n})^T$, we can easily derive *gradient* ∇f, *divergence* $\nabla \cdot f$ and *curl* $\nabla \times f$ of a function f.

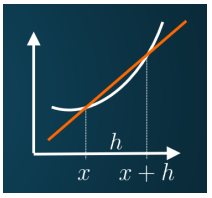
2.2. Numerical Differentiation

Goal: Approximate the derivative of a mathematical function.

Finite Differences

For Newton's difference quotient:

$$\frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



For *symmetric* difference quotient (small h):

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

(The symmetric difference quotient is *unbiased*, unlike the difference quotient, since there we check the function value in one direction only.)

2.3. Numerical Integration

Our goal is to solve *first-order ODEs*. Generally, we want to use small time sizes, since the accuracy directly depends on it. Especially for *stiff* equations (small difference in initial values have big impact) these methods tend to get numerically unstable unless using a very small step size.

In general, we want to update position and velocity by a (small) magnitude of the corresponding gradient:

$$\begin{aligned} \mathbf{x}(t_0 + \Delta t) &= \mathbf{x}(t_0) + \Delta t \dot{\mathbf{x}} = \mathbf{x}(t_0) + \Delta t \mathbf{v} \\ \mathbf{v}(t_0 + \Delta t) &= \mathbf{v}(t_0) + \Delta t \dot{\mathbf{v}} = \mathbf{v}(t_0) + \Delta t \mathbf{a} \end{aligned}$$

Explicit Methods

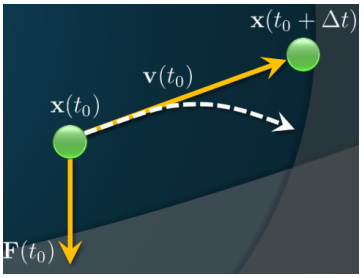
Explicit methods directly calculate new positions and velocity by evaluating two equations.

They are all simple and fast to compute, but only *conditionally stable* (system is only stable under the assumption of a sufficiently small step size), which makes them unsuitable for large time-steps and stiffness parameters (since they introduce large forces).

Explicit Euler

Taylor approximation of first-order:

$$\begin{aligned}\mathbf{x}(t_0+\Delta t) &= \mathbf{x}(t_0) + \Delta t \mathbf{v}(t_0)\\ \mathbf{v}(t_0+\Delta t) &= \mathbf{v}(t_0) + \Delta t \cdot \underbrace{\frac{1}{m}\mathbf{F}(t_0)}_{F=ma \leftrightarrow a=F/m}\end{aligned}$$



- *Error:* $\mathcal{O}(\Delta t^2)$ (inherent error from Taylor expansion, since we consider constant + linear part only)

$$f(t_0+\Delta t) = f(t_0) + \Delta t f'(t_0) + \mathcal{O}(\Delta t^2)$$

Symplectic / Semi-Implicit Euler

Update positions using updated velocities:

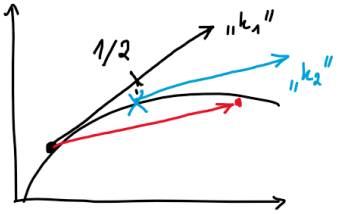
$$\begin{aligned}\mathbf{v}(t_0+\Delta t) &= \mathbf{v}(t_0) + \frac{\Delta t}{m}\mathbf{F}(t_0)\\ \mathbf{x}(t_0+\Delta t) &= \mathbf{x}(t_0) + \Delta t \mathbf{v}(t_0+\Delta t)\end{aligned}$$

- *Error:* $\mathcal{O}(\Delta t^2)$
- more stable than explicit Euler

Second-Order Runge-Kutta (RK2)

For $\mathbf{f}(t,\mathbf{s}) = (\dot{\mathbf{x}}, \dot{\mathbf{v}})^T$ with *state* $\mathbf{s}(t) = (\mathbf{x}(t), \mathbf{v}(t))$, the RK2 method has two stages:

$$\begin{aligned}\mathbf{k}_1 &= \Delta t \mathbf{f}(t_0, \mathbf{s}_0)\\ \mathbf{k}_2 &= \Delta t \mathbf{f}(t_0 + \frac{1}{2}\Delta t, \mathbf{s}_0 + \frac{1}{2}\mathbf{k}_1)\\ \mathbf{s}(t_0+\Delta t) &= \mathbf{s}_0 + \mathbf{k}_2\end{aligned}$$



- *Error:* $\mathcal{O}(\Delta t^3)$
- more accurate results, but at each stage $\mathbf{f}$ has to be evaluated
- higher order RK methods also exist; one can show, that an n-th-order RK method has an error of $\mathcal{O}(\Delta t^{n+1})$

Implicit Methods

In implicit methods, the updates are only computed implicitly by solving a system of equations, that may be hard to solve if the behavior of deformation gradient $\mathbf{F}$ is highly non-linear. Usually, they are solved iteratively using the *Newton method* (by reformulating the equation to be a root-finding problem).

Implicit methods typically have better stability properties than explicit ones.

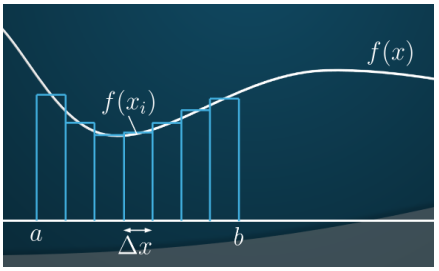
Implicit Euler

$$\mathbf{x}(t_0 + \Delta t) = \mathbf{x}(t_0) + \Delta t \mathbf{v}(t_0 + \Delta t)$$
$$\mathbf{v}(t_0 + \Delta t) = \mathbf{v}(t_0) + \frac{\Delta t}{m}\mathbf{F}(\mathbf{x}(t_0 + \Delta t), \mathbf{v}(t_0 + \Delta t))$$

- *Error:* $\mathcal{O}(\Delta t^2)$
- one step with the implicit Euler is far more expensive than an explicit step and it introduces numerical damping, but it is *unconditionally stable*

Numerical Integration of a Function

$$\int_a^b f(x)dx \approx \sum_i f(x_i)\Delta x$$
$$\iint_\Omega f(x,y)dxdy \approx \sum_i f(x_i, y_i)\Delta x \cdot \Delta y$$



▼ Numerical integration relates to numerical time integration.

$$\frac{dx}{dt} = v$$
$$dx = vdt$$
$$\int_{x(t_0)}^{x(t_1)} dx = \int_{t_0}^{t_1} vdt$$
$$x(t_1) - x(t_0) = \int_{t_0}^{t_1} vdt$$
$$x(t_1) = x(t_0) + \int_{t_0}^{t_1} vdt$$
$$x(t_0 + \Delta t) \approx x(t_0) + \Delta t v(t_0)$$

3.1. Deformable Solids: Continuous Model

▼ *Notations*

- $\mathbf{A} : \mathbf{B} = \text{tr}[\mathbf{A}^T\mathbf{B}]$
- *Frobenius Norm:* $||\mathbf{A}||_F := \sqrt{\sum_i \sum_j |a_{ij}|^2}$

Here, we describe the simulation of deformable solids using continuum mechanical approaches under the assumption of a *continuous model*.

Terminology:

- **Deformation**: transformation of a body from a reference or material configuration to a current deformed configuration
- **Elasticity:** property of a material to deform if a force is acting on the material and to return to its original size and shape if the force is removed
- **Plasticity:** property of a material to deform irreversibly in response to applied forces when the deformation exceeds a *yield point* (below the yield point the material deforms *elastically*)

Summary

1. Describe the deformation by the **Kinematic Equation**.
2. Define a **Strain Measure** *(Dehnung)* for this deformation.
3. Define the relationship between *strain* and *stress* *(Spannung)* by a **Constitutive Law**.
4. The resulting stress tensor in combination with the **Momentum Equation** gives us the equation of motion.

💡 ***Recipe***

1. Determine current *displacement* $\mathbf{u}(\mathbf{X}, t)$.
2. Determine *deformation gradient* $\mathbf{F}$.
3. Choose a *constitutive law* defining strain energy density $\Phi(\mathbf{F})$, from which we determine the *stress tensor* $\mathbf{P}(\mathbf{F})$.
4. Determine (internal) *forces* $\mathbf{f}$ as negative gradient of potential energy $E(\mathbf{x})$, where the potential energy is zero if material is in equilibrium.
5. Use forces to (numerically) update velocities and positions.

3.1.1. Kinematic Equation

From the continuous model, we can describe the motion of a particle in a continuum using deformed positions $\mathbf{x} = \mathbf{x}(\mathbf{X}, t)$ with initial configuration $\mathbf{X} = \mathbf{x}(\mathbf{X}, t_0)$, where vector $\mathbf{X}$ are also known as the *material coordinates*.

We can also describe some quantity $\mathbf{q}$ of a continuum in motion, that changes with time:

- **Material / Reference / Lagrangian Description:** $\mathbf{q} = \hat{\mathbf{q}}(\mathbf{X}, t)$

- [Spatial / Eulerian Description: $\mathbf{q} = \tilde{\mathbf{q}}(\mathbf{x}, t)$]

> *Intuitively, the Lagrangian description follows the initial discretization through its deformation process, whereas the Eulerian description, the quantity depends on the discretization of the already deformed shape.*

Given reference configuration $\mathbf{X}$ and deformed configuration $\mathbf{x}$, we can define the *displacement field* as the **kinematic equation**

$$\mathbf{u}(\mathbf{X}, t) = \mathbf{x}(\mathbf{X}, t) - \mathbf{X} \quad \Leftrightarrow \quad \mathbf{x}(\mathbf{X}, t) = \mathbf{u}(\mathbf{X}, t) + \mathbf{X}$$

The **deformation gradient** is later used to define *strain measures*:

$$\boxed{\mathbf{F} = \frac{\partial \mathbf{x}(\mathbf{X}, t)}{\partial \mathbf{X}} = \mathbf{I} + \nabla \mathbf{u} = \mathbf{I} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}}}$$

Per definition, $\mathbf{F}$ neglects *translation* in the deformed positions.

3.1.2. Strain Measure

Strain *(Dehnung)* describes the deformation in terms of the *relative* displacement of the particles in a body where *rigid-body motions* (rotation, translation) are excluded.

Generally, we don't use the deformation gradient $\mathbf{F}$ as the strain tensor itself, since we want "zero deformation" for rotation as well.

Green Strain Tensor

The *right Cauchy-Green deformation tensor* is defined on the left, which yields the **Green strain tensor** on the right:

$$\mathbf{C} = \mathbf{F}^T \mathbf{F} \quad \Rightarrow \quad \boxed{\mathbf{E} = \frac{1}{2}(\mathbf{F}^T \mathbf{F} - \mathbf{I})}$$

- For reference configuration and rigid body motions we get $\mathbf{C} = \mathbf{F}^T \mathbf{F} = \mathbf{I}$ and $\mathbf{E} = \mathbf{0}$ *(since translation vanishes during differentiation and for rotation $\mathbf{F}$ is orthogonal)*.

We can therefore decompose $\mathbf{F} = \mathbf{RS}$ into an *orthogonal* $(\mathbf{R})$ and a *symmetric*$(\mathbf{S})$ matrix via *polar decomposition*, which yields

$$\mathbf{E} = \frac{1}{2}(\mathbf{S}^2 - \mathbf{I})$$

- $(\mathbf{RS})^T \mathbf{RS} = \mathbf{S}^T \mathbf{R}^T \mathbf{RS} = \mathbf{S}^T \mathbf{IS} = \mathbf{S}^2$ *(since $\mathbf{R}$ orthogonal and $\mathbf{S}$ symmetric)*
- *Stretch* and *shear* information are retained in $\mathbf{S}$.

Cauchy Strain Tensor

The Green strain tensor is *quadratic*, which makes it more expensive to use. For *infinitesimal* deformations, we can approximate it as a *linear* strain tensor (removing quadratic $(\nabla \mathbf{u})^T (\nabla \mathbf{u})$ part):

$$\mathbf{E} = \frac{1}{2}(\mathbf{F}^T \mathbf{F} - \mathbf{I}) = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T + (\nabla \mathbf{u})^T (\nabla \mathbf{u})) \approx \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)$$

This linear strain tensor is known as the **Cauchy strain tensor**:

$$\boxed{\boldsymbol{\epsilon} = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T) = \frac{1}{2}(\mathbf{F} + \mathbf{F}^T) - \mathbf{I}}$$

- Can only be used for *small* deformations, and nonlinear deformations like rotations cause undesired results, since ϵ is generally not zero for rotation (unlike Green strain).

3.1.3. Constitutive Law

While strain measures the deformation of a material, **stress** *(Spannung)* is a physical quantity that describes the internal forces between neighboring particles in a continuous material. These internal forces are modeled as *counteracting* forces reacting to strain using **constitutive models**, that give us a relationship between strain and stress, e.g. using stress tensors.

The **stress tensor** is a second-order tensor which describes the state of stress at a certain point inside a material.

In the following, we define:

- The ***Piola-Kirchhoff Stress Tensor*** $\mathbf{P}(\mathbf{F})$, that is dependent on the ***strain energy density*** $\Psi(\mathbf{F})$.
- The *constitutive models*, that determine the required strain energy density $\boldsymbol{\Psi}(\mathbf{F})$:
  - ***Linear Elasticity Model***
  - ***Corotated Linear Elasticity Model***: Redefine Cauchy strain to remove the rotation in the deformation gradient and plug the new strain measure in *linear elasticity* model.
  - ***St. Venant-Kirchhoff Model***: Use the Green strain in the linear elasticity model instead of the Cauchy strain.
  - ***Neohookean Model***: Approximate singular value matrix of $\mathbf{F}$ (= pure scaling) using an *isotropic invariant*.
  - ***Hooke's Generalized Law***: Material model for anisotropic material behavior.

- Strain Energy -

Strain energy is a scalar potential energy in the deformed body due to elastic deformation. The strain energy of a *hyperelastic material* for a given deformation is determined by integrating the *scalar* **strain energy density** Ψ over the entire body Ω:

$$\boxed{E(\mathbf{x}) = \int_{\Omega} \Psi(\mathbf{F}) d\mathbf{X}}$$

- We only address *hyperelastic* materials where the strain energy is *independent of the prior deformation history*.

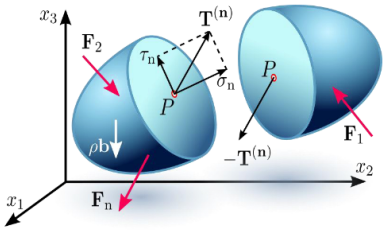
A *force* that *counteracts* the current deformation in order to simulate elasticity is determined by the negative gradient of the strain energy, pointing in the direction of steepest descent in energy:

$$\boxed{\mathbf{f}(\mathbf{x}) = -\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}}}$$

▼ **- Stress Vector -**

We can directly get a per point (vector) force acting on a internal surface by the product of the point-specific stress tensor $\mathbf{P}$ with the normal vector:

$$\mathbf{T}^{(\mathbf{n})} = \mathbf{P} \cdot \mathbf{n}$$



The resulting **traction** (pulling on part of the body) stress vector can be separated into the two components: *normal* (σ_n) and *shearing* (τ_n) *stress*.

The difference of traction to force is that tranction occurs, if we cut open a body and measure the force there.

Stress Tensor: Piola-Kirchhoff Stress Tensor

The *Piola-Kirchhoff stress tensors* describe the stress with respect to the reference configuration, which is the *force response* along different orientation. The first Piola-Kirchhoff stress tensor defines a relationship between forces in the current (deformed) configuration to areas in the reference configuration.

For hyperelastic materials the first **Piola-Kirchhoff stress tensor** is a function which only depends on the deformation gradient $\mathbf{F}$:

$$\boxed{\mathbf{P}(\mathbf{F}) = \frac{\partial \Psi(\mathbf{F})}{\partial \mathbf{F}}}$$

In the following, we define different constitutive models for the *strain energy density function* Ψ.

Constitutive Models: Linear Elasticity Model

The **linear elasticity model** is given as:

$$\boxed{\Psi(\mathbf{F}) = \mu \boldsymbol{\epsilon} : \boldsymbol{\epsilon} + \frac{\lambda}{2} \text{tr}^2(\boldsymbol{\epsilon})}$$

- $\boldsymbol{\epsilon}$: Cauchy strain tensor
- μ, λ: material specific **Lamé coefficients**

The *first Piola-Kirchhoff stress tensor* of the linear elasticity model is determined by:

$$\boxed{\mathbf{P}(\mathbf{F}) = 2\mu\boldsymbol{\epsilon} + \lambda \text{tr}(\boldsymbol{\epsilon})\mathbf{I}}$$

(since $\boldsymbol{\epsilon} : \boldsymbol{\epsilon}$ can be seen as elementwise $\boldsymbol{\epsilon}^2$, and $\frac{\partial \boldsymbol{\epsilon}}{\partial \mathbf{F}} = \frac{1}{2}(2\mathbf{I}) - \mathbf{I} = \mathbf{I}$)

— Lamé Coefficients —

The *Lamé coefficients* are related to the Young's modulus k (measure of *stretch resistance*) and the Poisson's ratio ν (measure of *incompressibility*) as:

$$\mu = \frac{k}{2(1+\nu)}, \quad \lambda = \frac{k\nu}{(1+\nu)(1-2\nu)}$$

Material is *incompressible* (volume preserving) with $\nu = 0.5$, but then λ is not solvable.

Properties:

- Resulting stress tensor is a *linear* function of deformation gradient $\mathbf{F}$, resulting in lower computational effort.
- Model can only be used for *small* deformations, since we use the Cauchy strain.

Constitutive Models: Corotated Linear Elasticity Model

We define a modified strain measure by replacing $\mathbf{F} = \mathbf{RS}$ (*polar decomposition*) with $\mathbf{R}^T\mathbf{F}$ ($= \mathbf{S}$, the *unrotated* version of the deformation gradient):

$$\boldsymbol{\epsilon}_C = \frac{1}{2}(\mathbf{R}^T\mathbf{F} + (\mathbf{R}^T\mathbf{F})^T) - \mathbf{I} = \mathbf{S} - \mathbf{I}$$

Plugging this strain tensor into the definition of the *linear elasticity model*, we get the **corotated linear elasticity model** as

$$\Psi(\mathbf{F}) = \mu||\mathbf{S} - \mathbf{I}||_F^2 + \frac{\lambda}{2}\text{tr}^2(\mathbf{S} - \mathbf{I})$$

which can be rewritten as

$$\boxed{\Psi(\mathbf{F}) = \mu||\mathbf{F} - \mathbf{R}||_F^2 + \frac{\lambda}{2}\text{tr}^2(\mathbf{R}^T\mathbf{F} - \mathbf{I})}$$

(since $\mathbf{S} = \mathbf{R}^T\mathbf{F}$, $\mathbf{F} - \mathbf{R} = \mathbf{R}(\mathbf{S} - \mathbf{I})$, and $||\mathbf{XY}||_F = ||\mathbf{Y}||_F$ if $\mathbf{X}$ orthogonal)

The corresponding *first Piola-Kirchhoff stress tensor* is:

$$\boxed{\mathbf{P}(\mathbf{F}) = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda\text{tr}(\mathbf{R}^T\mathbf{F} - \mathbf{I})\mathbf{R}}$$

Properties:

- With the corotated model we achieve a *linear* model (like the linear elasticity model), that also *mitigates the effects rotations have on the strain tensor*.
- increased computational cost due to polar decomposition for each finite element

---

Constitutive Models: St. Venant-Kirchhoff Model

Previously in the linear elasticity model, we used the Cauchy strain tensor as a linear approximation of the nonlinear rotationally invariant Green strain tensor $\mathbf{E}$.

The **St. Venant-Kirchhoff model** improves the linear elasticity model by replacing the Cauchy strain by the *Green strain*, which yields a *nonlinear* isotropic material model:

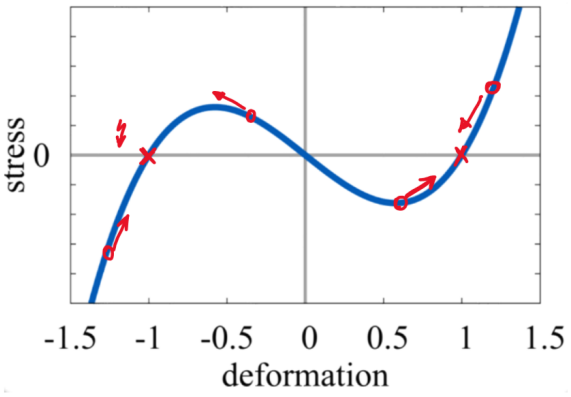
$$\boxed{\Psi(\mathbf{F}) = \mu\mathbf{E} : \mathbf{E} + \frac{\lambda}{2}\text{tr}^2(\mathbf{E})}$$

The *first Piola-Kirchhoff stress tensor* of the St. Venant-Kirchhoff material is determined by:

$$\boxed{\mathbf{P}(\mathbf{F}) = \mathbf{F}(2\mu\mathbf{E} + \lambda \mathrm{tr}(\mathbf{E})\mathbf{I})}$$

which is a function cubic in $\mathbf{F}$. Therefore, the relationship between $\mathbf{P}$ and the deformation gradient $\mathbf{F}$ is given as on the right. In particular, deformations on the rhs want to return to the right root of the function. Negative deformation *(compression so high, that the element turns itself inside out)* want to return to the lhs root, which is problematic, since this case is not found in nature.

This makes the model *poorly resistent to large compression*.



- Rotationally Invariance -

A hyperelastic material model is **rotationally invariant** iff for arbitrary deformation gradient $\mathbf{F}$ and rotation $\mathbf{R}$:

$$\boxed{\Psi(\mathbf{F}) = \Psi(\mathbf{RF})}$$

- The linear elasticity model is not rotationally invariant due to the use of the Cauchy strain tensor, whereas the corotated linear elasticity model and St. Venant-Kirchhof model are constructed to be rotationally invariant.
- uses only 6 of 9 DOFs in $\mathbf{F}$, as defined by symmetric part $\mathbf{S}$

- Isotropy -

All the constitutive models discussed so far are *isotropic*, meaning, that the resistance of the material to deformation is *the same* in all possible directions.

A hyperelastic material model is **isotropic** iff for arbitrary deformation gradient $\mathbf{F}$ and rotation $\mathbf{Q}$:

$$\boxed{\Psi(\mathbf{F}) = \Psi(\mathbf{FQ})}$$

(Intuitively: first rotating an infinitesimal volume and then applying the deformation gradient $\mathbf{F}$ should result in the same strain energy density, if the material is isotropic)

Therefore, for an *isotropic* and *rotationally invariant* material with $\mathbf{F} = \mathbf{U\Sigma V}^T$ SVD of the deformation gradient:

$$\Psi(\mathbf{F}) = \Psi(\mathbf{U\Sigma V}^T) = \Psi(\mathbf{\Sigma})$$

(Intuitively: energy density Ψ of $\mathbf{F}$ is equivalent of those of a scaling transformation)

- uses only 3 DOF, as captured by $\mathbf{\Sigma}$

Constitutive Models: Neohookean Model

The idea now is to define a model, that describes an isotropic material using the singular values $\mathbf{\Sigma}$. However, SVD in each evaluation of the model is expensive. To avoid computational overhead, we instead define three **isotropic invariants**, which are cheap to compute and equally expressive as the singular values:

$$I_1(\mathbf{F}) = \text{tr}(\mathbf{F}^T\mathbf{F})$$
$$I_2(\mathbf{F}) = \text{tr}[(\mathbf{F}^T\mathbf{F})^2]$$
$$I_3(\mathbf{F}) = (\det \mathbf{F})^2$$

The isotropic *nonlinear* **Neohookean material** is defined by the following strain energy density:

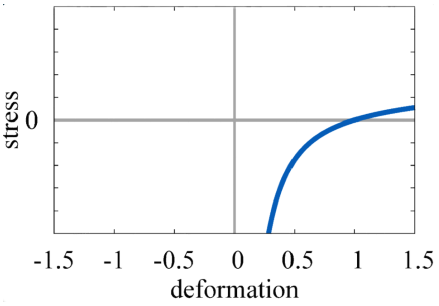
$$\boxed{\Psi(\mathbf{F}) = \frac{\mu}{2}(I_1 - \log(I_3) - 3) + \frac{\lambda}{8}\log^2(I_3)}$$

The *first Piola-Kirchhoff stress tensor* of the Neohookean material is determined by:

$$\mathbf{P}(\mathbf{F}) = \mu\mathbf{F} - \mu\mathbf{F}^{-T} + \frac{\lambda}{2}\log(I_3)\mathbf{F}^{-T}$$

The relationship between $\mathbf{P}$ and $\mathbf{F}$ is shown on the right. Note, that unlike in the St. Venant-Kirchhoff model, the function is not defined for negative deformations.

This means, that the Neohookean model exhibits a strong reaction to large compression.



Constitutive Models: Hooke's Generalized Law

Hooke's generalized law is defined by the following strain energy density:

$$\boxed{\Psi(\mathbf{F}) = \underbrace{\frac{1}{2}\mathbf{E} : \mathbf{C}\mathbf{E}}_{\text{“}\frac{1}{2}k\Delta x^2\text{”}}}$$

where $\mathbf{C}$ is a constant tensor describing (possibly *anisotropic*) material properties.

The *first Piola-Kirchhoff stress tensor* using Hooke's generalized law is determined by:

$$\mathbf{P}(\mathbf{F}) = \mathbf{F}\mathbf{C}\mathbf{E}$$

3.1.4. Momentum Equation

Substitution a stress tensor $\mathbf{P}$ for a specific material, in the (Cauchy-) **momentum equation** gives us the **equation of motion**:

$$\rho_0\ddot{\mathbf{u}} = \nabla \cdot \mathbf{P} + \mathbf{f} \quad \rightarrow \quad \boxed{\rho_0\ddot{\mathbf{u}} = \nabla \cdot \mathbf{P}(\mathbf{F}) + \mathbf{f}}$$

- ρ_0: density
- $\nabla \cdot \mathbf{P}(\mathbf{F})$: internal / elasticity forces
- $\mathbf{u} = \mathbf{x} - \mathbf{X}$: displacement
- $\mathbf{f}$: external body force per unit volume

A simulation can be performed by integrating the resulting equation of motion. To compute the spatial derivative in the momentum equation, the domain has to be discretized.

3.2. Deformable Solids: FEM

Recipe

1. Compute the *deformation gradient* for each element, depending on the initial and deformed positions of its vertices.
2. Depending on the chosen constitutive law, compute the *stress tensor* for each element.
3. Given the stress tensor, we can directly compute the *elastic / internal forces* for each vertex in each element.
4. Use the forces to perform a *time integration* step.

3.2.1. Finite Elements

In the previous chapter, we derived equations for continuous models, which usually have no closed-form solutions and need to be solved using body *discretization*.

The deformable solid is therefore subdivided in a *finite* number of parts with a simple geometry (e.g. tetrahedron in 3D, triangle in 2D), the **finite elements**.

Summary:

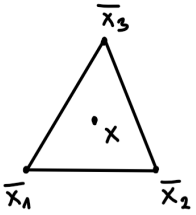
1. For interpolation between node values, choose **shape functions** $N_i(\mathbf{X})$, such that we can compute their gradients w.r.t. the initial configuration $\mathbf{X}$.
2. The **deformation gradient** $\mathbf{F}$ is now only dependent on the gradient of the chosen shape functions, and is in particular *constant* over the entire element for *linear* shape functions.
3. The **strain energy** $E(\mathbf{x})$ simplifies to a linear function of strain energy density Ψ, which is constant over the entire element for a constant deformation gradient.
4. Using the simplified deformation gradient and strain energy, we can easily compute the **force** $\mathbf{f}_n(\mathbf{x})$ for a node n as a sum of neighboring forces, given $\Psi(\mathbf{F})$ from a constitutive model.

Shape Functions

Since all the equations are then evaluated only at the vertex positions, we use *interpolation* to get intermediate values and approximate the underlying continuous function as well. The behavior of interpolation is described using **shape functions**.

To reconstruct the continuous function $\mathbf{x}(\mathbf{X}, t)$ from the values $\bar{\mathbf{x}}_i$ at the vertices, these values are interpolated using *shape functions* N_i:

$$\boxed{\mathbf{x} \approx \sum_i N_i(\mathbf{X}) \cdot \bar{\mathbf{x}}_i}$$



(Note, how now only N_i depends on the initial configuration and not $\bar{\mathbf{x}}_i$)

Shape functions may be of any order, where with increased order the interpolation becomes more accurate, but consequently also more complex.

A simple linear shape functions we will further use are the *barycentric coordinate functions.*

▼ ***Barycentric Coordinate Functions***

The barycetric coordinate functions are defined by 3 polynomials:

$$N_1(\mathbf{X}) = \frac{1}{2A_e} \det \begin{pmatrix} 1 & 1 & 1 \\ X & X_2 & X_3 \\ Y & Y_2 & Y_3 \end{pmatrix}$$
$$N_2(\mathbf{X}) = \frac{1}{2A_e} \det \begin{pmatrix} 1 & 1 & 1 \\ X_1 & X & X_3 \\ Y_1 & Y & Y_3 \end{pmatrix}$$
$$N_3(\mathbf{X}) = \frac{1}{2A_e} \det \begin{pmatrix} 1 & 1 & 1 \\ X_1 & X_2 & X \\ Y_1 & Y_2 & Y \end{pmatrix}$$

where $\mathbf{X} = (X, Y)^T$ and $A_e = \frac{1}{2}||(\mathbf{X}_2 - \mathbf{X}_1) \times (\mathbf{X}_3 - \mathbf{X}_1)||$ the area of a triangle. Stacking and evaluating the derivative results in

$$\frac{\partial N_e}{\partial \mathbf{X}} = \frac{1}{2A_e} \begin{pmatrix} Y_2 - Y_3 & X_3 - X_2 \\ Y_3 - Y_1 & X_1 - X_3 \\ Y_1 - Y_2 & X_2 - X_1 \end{pmatrix}, \quad \mathbf{N}_e = \begin{pmatrix} N_1 \\ N_2 \\ N_3 \end{pmatrix}$$

which is *constant* due to the linear nature of the barycentric coordinate functions and can be therefore computed beforehand.

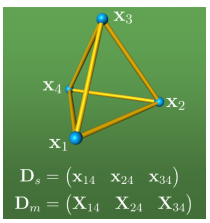
In particular, since we choose the shape functions beforehand, we can make sure, that $\frac{\partial N_i(\mathbf{X})}{\partial \mathbf{X}}$ is *computable*. If the gradient is constant (like it is for the barycentric coordinate functions), we can even precompute them for later use.

Quantities

- Deformation Gradient -

In particular, we now can directly compute the deformation gradient, which was not possible in the continuous case:

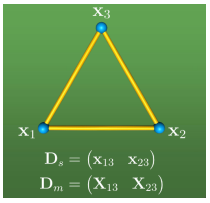
$$\boxed{\mathbf{F} = \frac{\partial \mathbf{x}(\mathbf{X}, t)}{\partial \mathbf{X}} = \sum_i \frac{\partial N_i(\mathbf{X})}{\partial \mathbf{X}} \cdot \bar{\mathbf{x}}_i}$$



(Note, that we assume $\bar{\mathbf{x}}_i$ to be independent of $\mathbf{X}$)

In matrix notation, we write the deformation gradient $\mathbf{F}$ to be

$$\boxed{\mathbf{F} = \mathbf{D}_s \mathbf{D}_m^{-1}}$$



- $\mathbf{D}_s$: *spatial* shape matrix with $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ being the difference in values
- $\mathbf{D}_m$: *material* shape matrix with $\mathbf{X}_{ij} = \mathbf{X}_i - \mathbf{X}_j$ being the difference in position

▼ ***Derivation***

When applying $\mathbf{x}(\mathbf{X}, t) = \frac{\partial \mathbf{x}(\mathbf{X}, t)}{\partial \mathbf{X}} \mathbf{X} + \mathbf{t} = \mathbf{F}\mathbf{X} + \mathbf{t}$ to the separate vertices and subtracting $\mathbf{x}_4 = \mathbf{F}\mathbf{X}_4 + \mathbf{t}$ from all other equations, we get $\mathbf{D}_s = \mathbf{F}\mathbf{D}_m$.

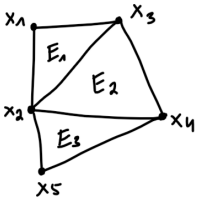
$\mathbf{D}_m$ is always invertible, if the mesh has no collapsed elements, indicated by a zero entry in the matrix.

$\mathbf{F}$ is *constant* for one element, since the shape functions are assumed to the *linear*.

- Strain Energy -

The *strain energy* E of a hyperelastic material for a given deformation is then a sum of the energy contribution of individual elements:

$$\boxed{E(\mathbf{x}) = \int_\Omega \Psi(\mathbf{F}) d\mathbf{X} = \sum_i E_i = \sum_i \int_{T_i} \Psi(\mathbf{F}_i) d\mathbf{X}}$$



Since for linear elements the deformation gradient is constant over the element, the strain energy for a tetrahedral element i is determined by

$$\boxed{E_i = \int_{T_i} \Psi(\mathbf{F}_i) d\mathbf{X} = \Psi(\mathbf{F}_i) \int_{T_i} d\mathbf{X} = V \Psi(\mathbf{F}_i)}$$

where V is the undeformed tet volume and energy density Ψ is constant over one element.

- Forces -

The force of a node n is determined by summing up the contributions of all neighboring elements:

$$\boxed{\mathbf{f}_n = \sum_{i \in N_n} \mathbf{f}_n^i(\mathbf{x})}, \quad \mathbf{f}_n^i = -\frac{\partial E_i(\mathbf{x})}{\partial \mathbf{x}_n}$$

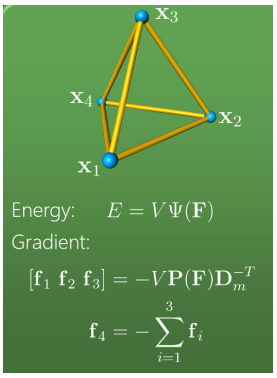
The *elastic forces* for a tet *i* are:

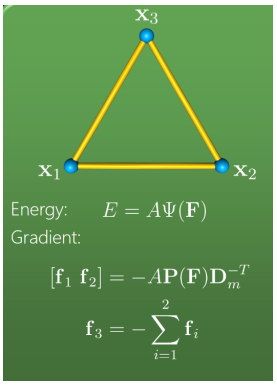
$$\mathbf{H}_i = (\mathbf{f}_1^i, \mathbf{f}_2^i, \mathbf{f}_3^i) = -V \mathbf{P}(\mathbf{F}) \mathbf{D}_m^{-T}, \quad \mathbf{f}_n^i(\mathbf{x}) = -\frac{\partial E_i(\mathbf{x})}{\partial \mathbf{x}_n} = \frac{\partial E_i}{\partial \mathbf{F}} \cdot \frac{\partial \mathbf{F}}{\partial \mathbf{x}_n}$$

$$\mathbf{f}_4^i = -\mathbf{f}_1^i - \mathbf{f}_2^i - \mathbf{f}_3^i \quad \rightarrow \text{internal energy conservation}$$

$\left(\frac{\partial E_i}{\partial \mathbf{F}} = V \frac{\partial \Psi(\mathbf{F})}{\partial \mathbf{F}} = V \mathbf{P}(\mathbf{F}) \right)$

- Combined -





Only difference to 3D is A instead of V.

3.2.2. Equation of Motion

Rearranging the equation of motion we get the *discrete* formulation, which is *not momentum conserving* anymore, since we add an additional (*internal*) damping part $\mathbf{D}\dot{\mathbf{x}}$ to avoid infinite swinging:

$$\boxed{\mathbf{M}\ddot{\mathbf{x}} + \mathbf{D}\dot{\mathbf{x}} + \mathbf{f}_{\text{int}} = \mathbf{f}_{\text{ext}}}$$

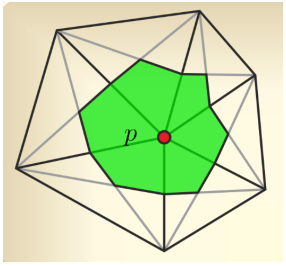
- $\mathbf{M}\ddot{\mathbf{x}}$: *inertial forces* with mass matrix $\mathbf{M}$
- $\mathbf{D}\dot{\mathbf{x}}$: *viscous forces* with damping matrix $\mathbf{D}$
- $\mathbf{f}_{\text{int}}$: *elasticity / internal forces*
- $\mathbf{f}_{\text{ext}}$: *external forces*
- $\mathbf{x}$: positions

With all the necessary values at the discretization points, we now can actually *solve* the equation of motion.

- Mass Matrix -

A simple approximation of the *mass matrix* can be determined by the so-called **mass lumping**, where the mass is concentrated in the nodes of the mesh. Mass of a particle is then specified using a material-specific constant ρ:

$$m_p = A_p\rho, \quad A_p = \frac{1}{3}\sum_{\text{adj. tri.}} A_i$$



- Internal Forces -

The *elastic forces* depend on the constitutive law which is used to simulate a specific material. For a linear material the elastic forces are determined by the *stiffness matrix* $\mathbf{K}$:

$$\mathbf{f}_{\text{int}} = \mathbf{K}(\mathbf{x} - \mathbf{X}) = \mathbf{K}\mathbf{u}$$

- Damping Matrix -

Rayleigh damping describes a simple approximation of the real damping as a combination of *mass* and *stiffness damping*:

$$\mathbf{D} = \alpha\mathbf{M} + \beta\mathbf{K}$$

- The mass damping term also damps rigid body motions. That means that even the motion of a free falling body is damped by this term.
- The stiffness damping term can be interpreted as *inner* damping which only occurs if the body is deformed.

- Linear Materials -

For linear materials, the equation of motion is

$$\boxed{\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}_{\text{ext}}}$$

for $\ddot{\mathbf{u}}$, $\mathbf{u}$, and $\mathbf{f}_{\text{ext}}$ determined at the discretization points.

▼ ***Derivation (Sketch)***

The equation of motion for linear materials is derived by substitution the Cauchy stress tensor (different from *Hooke's generalized law*) $\mathbf{P} = \mathbf{C}\boldsymbol{\epsilon}$ in the *momentum equation* $\rho_0\ddot{\mathbf{u}} = \nabla \cdot \mathbf{P} + \mathbf{f}$:

$$\underbrace{\int_\Omega \rho \mathbf{N}^T\mathbf{N} d\mathbf{X}\ddot{\mathbf{u}}}_{\mathbf{M}} - \underbrace{\int_\Omega \mathbf{N}^T\mathbf{N} d\mathbf{X}\bar{\mathbf{f}}}_{\frac{1}{\rho}\mathbf{M}} + \underbrace{\int_\Omega \mathbf{B}^T\mathbf{C}\mathbf{B} d\mathbf{X}\bar{\mathbf{u}}}_{\mathbf{K}} = 0$$

- $\mathbf{N}$: shape functions
- $\mathbf{B}$: defined by the derivatives of the shape functions to approximate $\boldsymbol{\epsilon} = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T) \approx \mathbf{B}\bar{\mathbf{u}}$
- $\mathbf{C}$: constant tensor describing material properties (see *Hooke's generalized law*)
- $\bar{\cdot}$: values determined at discretization points

With $\bar{\mathbf{f}}^{\text{ext}} = \frac{1}{\rho}\mathbf{M}\bar{\mathbf{f}}$ we get the final equation of motion:

$$\mathbf{M}\ddot{\bar{\mathbf{u}}} + \mathbf{K}\bar{\mathbf{u}} = \bar{\mathbf{f}}^{\text{ext}}$$

For linear elements the stiffness matrix is *constant* over the element:

$$\mathbf{K} = \int_\Omega \mathbf{B}^T\mathbf{C}\mathbf{B} d\mathbf{X} = \mathbf{B}^T\mathbf{C}\mathbf{B}\int_\Omega d\mathbf{X} = V\mathbf{B}^T\mathbf{C}\mathbf{B}$$

- V: undeformed tet volume
- $\mathbf{B}$: defined by the derivatives of the shape functions to approximate $\boldsymbol{\epsilon} = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T) \approx \mathbf{B}\bar{\mathbf{u}}$
- $\mathbf{C}$: constant tensor describing material properties (see *Hooke's generalized law*)

We can express the corresponding linear equations using the linear implicit Euler method:

$$\underbrace{(\mathbf{M} + \Delta t^2\mathbf{K})}_{known}\underbrace{\Delta\mathbf{v}}_{unknown} = \underbrace{-\Delta t \cdot (\mathbf{K}\mathbf{x} + \mathbf{f}_0 - \mathbf{f}_{\text{ext}} + \Delta t\mathbf{K}\mathbf{v})}_{known}$$
$$\underbrace{\Delta\mathbf{x}}_{unknown} = \underbrace{\Delta t \cdot (\mathbf{v} + \Delta\mathbf{v})}_{known}$$
$$\mathbf{f}_0 = \mathbf{f}(t_0)$$

which can be solved using standard approaches for numerically solving systems of linear equations, like the *conjugate gradient method*.

Alternatively, we can plug in the internal forces in the symplectic Euler method and solve it using time integration.

3.3. Deformable Solids: Real-Time Simulation

For all methods in this part *speed* and *stability* are the most important factors while accuracy is only secondary.

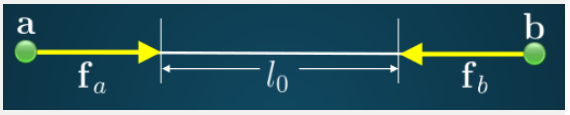
3.3.1. Primal / Dual Methods

Previously in *FEM*, in each simulation step we start with the current state (which is defined by the *DOF* positions, velocities), compute forces and finally perform a time integration. Such methods, which are expressed and solved *in terms of the DOF*, are called **primal methods**.

Dual methods, on the other hand, define *constraints for the DOF*, such that the system is expressed and solved in terms of *Lagrange multipliers*, which gives as an *indirect solution*. Lagrange multiplier represent constraint forces / impulses in the space of the corresponding constraint.

▼ *Example: Distance Constraint*

$$\mathbf{C}(\mathbf{a},\mathbf{b}) = ||\mathbf{a}-\mathbf{b}|| - l_0 \stackrel{!}{=} \mathbf{0}$$



Forces correcting the two particles to the constraint, are determined by the Lagrange multiplier λ:

$$\mathbf{f}_a = \lambda\frac{\partial C}{\partial \mathbf{a}} = \lambda\frac{\mathbf{a}-\mathbf{b}}{||\mathbf{a}-\mathbf{b}||}$$
$$\mathbf{f}_b = \lambda\frac{\partial C}{\partial \mathbf{b}} = -\lambda\frac{\mathbf{a}-\mathbf{b}}{||\mathbf{a}-\mathbf{b}||}$$

Instead of **world space** (WS), the correction via the Lagrange multiplier into the direction of the constrain is performed in **constraint space** (CS), which is defined by the gradient/Jacobian of the constraint: $\frac{\partial C}{\partial \mathbf{x}} = \mathbf{J}$.

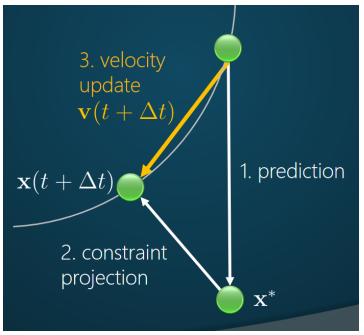
Summary:

- Discuss, how to compute *Lagrange multipliers* for two efficient *dual methods in continuum*:
  - **Position-Based Dynamics (PBD)**
  - **eXtended Position-Based Dynamics (XPBD)**
- Formulate a **Continuous PBD Model**, we we use the constraint solving to minimize the *strain energy*.
- Discuss an optimized primal method based on the FEM approach **(Fast Corotated FEM)**.

3.3.2. Dual Method: Position-Based Dynamics (PBD)

Assume we have a body, which is discretized using particles and their corresponding values. We define a set of constraint functions according to a desired behavior, which must *all* be enforced *in each simulation step*. All constraints we consider are **holonomic**, meaning, they depend on the particle positions only.

The *Position-Based Dynamics (PBD)* method enforces constraints by computing position changes and updating the velocity accordingly.



PBD-Algorithm

1. ***Prediction Step:*** Perform a time integration only considering external forces to get

   candidate positions $\mathbf{x}^*$.
2. ***Constraint Projection:*** Check if constraints are fulfilled, otherwise modify candidate positions to satisfy *one constraint at a time*:

$$\mathbf{x}(t+\Delta t) = constraintProjection\left(C(\mathbf{x}), \mathbf{x}^*\right)$$

3. ***Velocity Update:*** Update the velocities according to the position change *"finite differences style"* to get a consistent system.

1) Prediction Step

Perform *symplectic Euler integration* to get candidate positions and velocities:

$$\begin{aligned}
\mathbf{v}^*(t_0+\Delta t) &= \mathbf{v}(t_0) + \frac{\Delta t}{m}\mathbf{F}^{\text{ext}}(t_0) \\
\mathbf{x}^*(t_0+\Delta t) &= \mathbf{x}(t_0) + \Delta t \mathbf{v}^*(t_0+\Delta t) \\
\Rightarrow \boxed{\mathbf{x}^*(t+\Delta t) &= \mathbf{x}(t) + \Delta t \mathbf{v}(t) + \frac{\Delta t^2}{m}\mathbf{F}^{\text{ext}}(t)}
\end{aligned}$$

2) Constraint Projection

The goal of the constraint projection step is to modify the candidate positions $\mathbf{x}^*$ in order to fulfill the constraints $C(\mathbf{x})$, which results in the final positions $\mathbf{x}(t+\Delta t)$. $C(\mathbf{x})$ may depend on *all* positions in the mesh, and we want to enforce

$$\boxed{\mathbf{C}(\mathbf{x}) \stackrel{!}{=} 0}$$

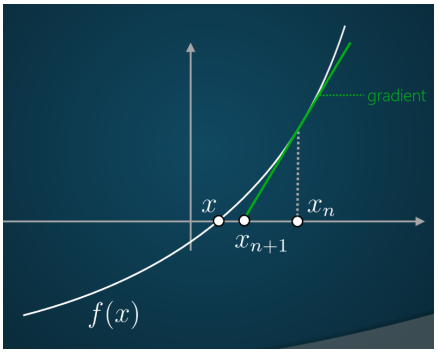
Recipe

1. *Linearize* (possibly non-linear) constraints of the corrected candidate positions using *first-order Taylor expansion*.
2. Perform *constraint projection* in constraint space for *momentum conservation* using **Newton's Method**, which yields a system of linear equations we can solve.

- Newton's Method -

Newton's method is an iterative approach to find the roots of a function f. Given initial guess x_0, the approximation of the solution is improved in each iteration:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



If we have a function $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, we first linearize it by a *first-order Taylor expansion*, and set the approximation to zero, which yields the equivalent vector-valued update rule:

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_0) + \mathbf{J} \cdot (\mathbf{x} - \mathbf{x}_0), \quad \mathbf{J} = \frac{\partial \mathbf{f}(\mathbf{x}_0)}{\partial \mathbf{x}}$$
$$\mathbf{f}(\mathbf{x}_0) + \mathbf{J} \cdot (\mathbf{x} - \mathbf{x}_0) \stackrel{!}{=} \mathbf{0}$$
$$\Rightarrow \quad \mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}^{-1}\mathbf{f}(\mathbf{x}_n)$$

In practice, we don't want to invert the Jacobian, but solve the following system, and update the approximation:

$$\mathbf{J}\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}_n) \quad \Rightarrow \quad \boxed{\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}}$$

- Constraint Projection -

Applying Newton's method to a constraint function $C(\mathbf{x}) = 0$ gives us the **constraint projection**. This means in each iteration step we check if the constraint is fulfilled for the candidate positions $\mathbf{x}^*$. If not, we search for a position correction $\Delta\mathbf{x}$, such that

$$C(\mathbf{x}^* + \Delta x) \overset{Taylor}{\approx} C(\mathbf{x}^*) + \mathbf{J} \cdot \Delta\mathbf{x} = 0$$

with $\mathbf{J} = \frac{\partial C}{\partial \mathbf{x}}$. This is an *underdetermined* linear system.

Intuitively, we want to choose the solution, that doesn't introduce additional energy into the system, i.e., which is *momentum conserving*. This means the constraint projection is performed in *constraint space*, and the *Lagrange multiplier* $\boldsymbol{\lambda}$ define the position correction in *constraint space*:

$$\Delta\mathbf{x} = \mathbf{J}^T\boldsymbol{\lambda}$$

Additionally, we want to weight the corrections with the inverse particle masses to better simulate *inertia* (Trägheit) of high mass particles, and solve the final system of linear equations:

$$\boxed{\Delta\mathbf{x} = \mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}} \quad \wedge \quad C(\mathbf{x}^*) + \mathbf{J} \cdot \Delta\mathbf{x} = 0 \quad \Rightarrow \quad \boxed{\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda} = -C(\mathbf{x}^*)}$$

- $\boldsymbol{\lambda}$: (*unknown*) mass-weighted position correction in CS
- $\mathbf{J}^T\boldsymbol{\lambda}$: mass-weighted position correction in WS
- $\mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}$: position correction in WS
- $\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda}$: position correction in CS
- $-C(\mathbf{x}^*)$: predicted constraint error in CS

→ That means we search for a mass-weighted position correction (in CS) that *counteracts the predicted constraint violation* due to external forces.

The per point position correction to enforce a constraint C is:

$$\lambda_i = -\frac{C(\mathbf{x}^*)}{\sum_j \frac{1}{m_j} \left(\nabla_{\mathbf{x}_j} C(\mathbf{x}^*)\right)^T \nabla_{\mathbf{x}_j} C(\mathbf{x}^*)}$$
$$\Delta \mathbf{x}_i = \frac{1}{m_i} \cdot \lambda_i \cdot \nabla_{\mathbf{x}_i} C(\mathbf{x}^*)$$

▼ ***Explanation***

$$\Delta \mathbf{x}_i = \underbrace{\frac{1}{m_i} \nabla_{\mathbf{x}_i} C}_{\text{"}\mathbf{M}^{-1}\mathbf{J}^T\text{"}} \cdot \underbrace{\underbrace{\frac{1}{\sum_j \frac{1}{m_j} \left(\nabla_{\mathbf{x}_j} C\right)^T \nabla_{\mathbf{x}_j} C}}_{\text{"}(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\text{"}} \cdot \underbrace{(-C)}_{-C(\mathbf{x}^*)}}_{\lambda}$$

- P = contraint.fetch_relevant_particle_data() *(i and j, where j also contains i)*
- C = constraint.evaluate(P)
- ∇C = constraint.evaluate_gradient(P) *(for all j's we get a gradient)*

As PBD approach solves one constraint after the other, the Newton iteration is performed *per constraint* and we iterate over all constraints, instead of performing a Newton iteration for the whole constraint system at once (which would lead to a huge matrix to be solved with).

Possible *constraint solver* are:

- **Modified Jacobi Iteration:** In each iteration solve *all constraints in parallel*, accumulate position corrections, and average corrections after each iteration.
- **Gauss-Seidel Iteration:** In each iteration solve *constraints one by one*.

  Better convergence, but atomic operations required for parallelization.

3) Velocity Update

The velocities are updated according to the position change using a *linear approximation*:

$$\boxed{\mathbf{v}(t + \Delta t) = \frac{1}{\Delta t}(\mathbf{x}(t + \Delta t) - \mathbf{x}(t))}$$

Discussion

- All we need for PBD are the constraints, and their gradients (see $\mathbf{J}$).
- We usually use a low number of iterations to avoid a completely stiff behavior. The remaining error of the approximation gives the "softness" of the body, while #iterations → ∞ would result in a physically correct stiff body.
- PBD cannot accurately generate results for "soft" constraints, but is physically correct for "hard" constraints.
- **Advantages:** *fast, stable*, simple to implement
- **Disadvantages:** Constraints may be a *discrete model*, where material behavior depends on spatial resolution; stiffness depends on #iterations and Δt (not on an actual stiffness parameter), such that *PBD is not physically correct for deformable material behavior*

3.3.3. Continuous PBD Model

Given the *recipe of continuous models*, where our goal is to minimize energy, we can alternatively minimize energy by a position-based formulation:

$$\boxed{C(\mathbf{x}) = E(\mathbf{x}) \overset{!}{=} 0}$$

This means, that we can circumvent solving a system of equations (as given by *equation of motion*) by now minimizing the energy as a optimization problem, just like PBD.

For the PBD approach we need the *constraint function* and *its gradient*.

Ingredients:

- $\phi(\mathbf{X}) = \mathbf{x}$: deformation
- $\mathbf{F} = \frac{\partial \phi(\mathbf{X})}{\partial \mathbf{X}}$: deformation gradient
- Ψ: energy density given by a constitutive law

- Strain Energy -

Continuous

$$E = \int_\Omega \Psi d\mathbf{X}$$

Discrete (with $\mathbf{F} = \mathbf{D}_s \mathbf{D}_m^{-1}$)

$$E = V\Psi(\mathbf{F})$$

- Strain Energy Gradient -

Continuous

$$\frac{\partial E}{\partial \mathbf{x}} = \int_\Omega \frac{\partial \Psi}{\partial \mathbf{x}} d\mathbf{X} = \int_\Omega \mathbf{P}(\mathbf{F}) \frac{\partial \mathbf{F}}{\partial \mathbf{x}} d\mathbf{X}$$

Discrete

Gradient:

$$\begin{bmatrix} \frac{\partial E}{\partial \mathbf{x}_1} & \frac{\partial E}{\partial \mathbf{x}_2} & \frac{\partial E}{\partial \mathbf{x}_3} \end{bmatrix} = V\mathbf{P}(\mathbf{F})\mathbf{D}_m^{-T}$$

$$\frac{\partial E}{\partial \mathbf{x}_4} = -\sum_{i=1}^{3} \frac{\partial E}{\partial \mathbf{x}_i}$$

3.3.4. Dual Method: eXtended Position-Based Dynamics (XPBD)

As opposed to PBD, we want to introduce a proper stiffness parameter.

Summary

1. Express the constraints, we want to enforce, using a *potential energy function* E, which also depends *explicitly* on a *(inverse) material stiffness matrix* $\boldsymbol{\alpha}$.
2. Using $-\nabla E$ as the *internal forces* within Newton's second law and performing *implicit time discretization via finite differences* to enforce an *unconditionally stable* method yields a system of linear equations, that we solve either with
   a. *Gauss-Seidel iteration, or*
   b. mix of *Gauss-Seidel* and *Newton iteration.*

   → The result is an update rule, that is similar to the update rule in PBD, but *which incorporates stiffness*.

Idea: In 1D, for a distance constraint on a spring we enforce $C(x) = x = 0$, so the force is $F(x) = -kx = -kC(x)$. By integrating the force we get a potential energy function $E(x) = -\int_0^x F(x')dx' = \frac{1}{2}kC(x)^2$.

Having this energy formulation, we can deduce an similar equation for a general vector constraint function, which is the *potential energy function* used in **XPBD**:

$$\boxed{E(\mathbf{x}) = \frac{1}{2}\mathbf{C}(\mathbf{x})^T\boldsymbol{\alpha}^{-1}\mathbf{C}(\mathbf{x}) \overset{!}{=} 0}$$

where $\boldsymbol{\alpha}$ is the **compliance matrix** (Nachgiebigkeit), which corresponds to the *inverse stiffness*.

To minimize the energy function, we compute forces as the negative gradient of the energy, Substituting these forces in Newton's second law yields:

$$\underbrace{\mathbf{M}\ddot{\mathbf{x}}}_{\text{“external forces”}} = \underbrace{-\nabla E(\mathbf{x})}_{\text{“internal forces”}}$$

By performing an *implicit time discretization via finite differences* (since we want an *unconditionally stable* method) and reformulation, we get the following equations that have to be solved to get the Lagrange multipliers:

$$\begin{aligned}\mathbf{M}\left(\mathbf{x}^{t+\Delta t} - \tilde{\mathbf{x}}\right) - \nabla\mathbf{C}(\mathbf{x}^{t+\Delta t})^T\boldsymbol{\lambda}^{t+\Delta t} &= \mathbf{0}\,(=\mathbf{g}(\mathbf{x},\boldsymbol{\lambda}))\\ \mathbf{C}(\mathbf{x}^{t+\Delta t}) + \tilde{\alpha}\boldsymbol{\lambda}^{t+\Delta t} &= \mathbf{0}\,(=\mathbf{h}(\mathbf{x},\boldsymbol{\lambda}))\end{aligned}$$

with $\mathbf{x}^{t+\Delta t}$ and $\boldsymbol{\lambda}^{t+\Delta t}$ being unknown, and

- Lagrange multiplier: $\boldsymbol{\lambda} = -\tilde{\boldsymbol{\alpha}}^{-1}\mathbf{C}(\mathbf{x}), \tilde{\boldsymbol{\alpha}} = \frac{\boldsymbol{\alpha}}{\Delta t^2}$
- *Predicted position*: $\tilde{\mathbf{x}} = 2\mathbf{x}^t - \mathbf{x}^{t-\Delta t} = \mathbf{x}^t + \Delta t\mathbf{v}^t$

Since the system is nonlinear, we can use Newton's method via Newton iteration (cf. *Newton's Method in constraint projection*), which yields the following linear system, that we have to solve *in each iteration*:

$$\begin{aligned}\mathbf{J}\Delta\mathbf{z} &= -\mathbf{f}(\mathbf{z}_n)\\ \mathbf{z}_{n+1} &= \mathbf{z}_n + \Delta\mathbf{z}\\ \Rightarrow \begin{pmatrix}\mathbf{K} & -\nabla\mathbf{C}(\mathbf{x}_n)^T\\ \nabla\mathbf{C}(\mathbf{x}_n) & \tilde{\alpha}\end{pmatrix}\begin{pmatrix}\Delta\mathbf{x}\\ \Delta\boldsymbol{\lambda}\end{pmatrix} &= -\begin{pmatrix}\mathbf{g}(\mathbf{x}_n,\boldsymbol{\lambda}_n)\\ \mathbf{h}(\mathbf{x}_n,\boldsymbol{\lambda}_n)\end{pmatrix}\\ \overset{1.,2.}{\Rightarrow} \boxed{\begin{pmatrix}\mathbf{M} & -\nabla\mathbf{C}(\mathbf{x}_n)^T\\ \nabla\mathbf{C}(\mathbf{x}_n) & \tilde{\alpha}\end{pmatrix}\begin{pmatrix}\Delta\mathbf{x}\\ \Delta\boldsymbol{\lambda}\end{pmatrix} = -\begin{pmatrix}\mathbf{0}\\ \mathbf{h}(\mathbf{x}_n,\boldsymbol{\lambda}_n)\end{pmatrix}}\end{aligned}$$

where $\mathbf{K} = \frac{\partial\mathbf{g}}{\partial\mathbf{x}}$.

1. Since $\mathbf{g}$ already contains the gradient of the constraint function, $\mathbf{K}$ would require the computation of the Hessian, which is quite expressive. To improve the runtime, we approximate via $\mathbf{K} \approx \mathbf{M}$, which yields the *Quasi-Newton-Method*.
2. We assume $\mathbf{g}(\mathbf{x}_n,\boldsymbol{\lambda}_n) = \mathbf{0}$, since the values of this functions are typically small, and the assumption simplifies the solver significantly.

Solving the first row and substituting the second row yields the update for row j

$$\begin{aligned}&\boxed{\Delta\mathbf{x} = \mathbf{M}^{-1}\nabla\mathbf{C}(\mathbf{x}_n)^T\Delta\boldsymbol{\lambda}}\\ \Rightarrow\quad &\left(\nabla\mathbf{C}(\mathbf{x}_n)\mathbf{M}^{-1}\nabla\mathbf{C}(\mathbf{x}_n)^T + \tilde{\alpha}\right)\Delta\boldsymbol{\lambda} = -\mathbf{C}(\mathbf{x}_n) - \tilde{\alpha}\boldsymbol{\lambda}_n\\ \Rightarrow\quad &\boxed{\Delta\lambda_j = \frac{-C_j(\mathbf{x}_n) - \tilde{\alpha}_j\lambda_{n,j}}{\nabla C_j\mathbf{M}^{-1}\nabla C_j^T + \tilde{\alpha}_j}}\end{aligned}$$

which is the same formulation as in PBD, except the marked parts. The final system can be solved using *Gauss-Seidel iteration*, where in each iteration we solve the system row by row.

The authors of the original paper propose to mix the *Gauss-Seidel* and the *Newton iterations*.
After computing the change of the Lagrange multiplier for the row
j, they directly update

$$\lambda_{n+1,j} = \lambda_{n,j} + \Delta\lambda_j$$
$$\mathbf{x}_{n+1,j} = \mathbf{x}_{n,j} + \Delta\mathbf{x}_j$$

Discussion

- XPBD solves the problem of PBD, where the stiffness of the material depends on time step and iteration count.
- If we drop the simplifications, and solve the non-linear system to convergence, we get an accurate solution (slow).
- If we are using our energy constraint for the continuous model and using a sufficient number of iteration, we get physically meaningful results.
- However, *XPBD is much slower than PBD*.
- For infinite stiffness ($\boldsymbol{\alpha} = 0$), we get the original PBD.

3.3.5. Primal Method: Fast Corotated FEM

The **fast corotated FEM approach** is based on the *corotated linear elasticity model*, where we can separate the terms according to their function:

$$\Psi(\mathbf{F}) = \underbrace{\mu||\mathbf{F}-\mathbf{R}||_F^2}_{\substack{\text{stretching/compression} \\ \text{resistance}}} + \underbrace{\frac{\lambda}{2}\text{tr}^2(\mathbf{R}^T\mathbf{F}-\mathbf{I})}_{\text{volume conservation}}$$

As the stretching/compression part has a *high stiffness coefficient* μ, which causes a large number of iterations in the linear system solver used in the implicit Euler step (compared to volume conservation part, which doesn't need that).

To speedup the simulation we use two *approximations*:

1. Separate the *stretching part* and *volume part* using **operator splitting** to handle them independently.
2. We assume $\mathbf{R}$ *to be constant during one time-step*, which results in a significant simplification in the computation of the *stretching part*. In the volume part however we cannot simplify anything with this assumption, so that this system is solved using standard *conjugate gradient method*.

Operator Splitting

First, we substitute the equation for $\mathbf{v}^{t+\Delta t}$ in the equation of $\mathbf{x}^{t+\Delta t}$, both given by *implicit Euler*, so that we can solve for new positions:

$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^t + \Delta t\underbrace{\left(\mathbf{v}^t + \Delta t\mathbf{M}^{-1}\left(\mathbf{f}^{\text{ext}} + \mathbf{f}^{\text{stretch}}(\mathbf{x}^{t+\Delta t}) + \mathbf{f}^{\text{volume}}(\mathbf{x}^{t+\Delta t})\right)\right)}_{\mathbf{v}^{t+\Delta t}}$$

Separate by all the difference forces:

$$\mathbf{x}^* = \mathbf{x}^t + \Delta t\mathbf{v}^t + \Delta t^2\mathbf{M}^{-1}\mathbf{f}^{\text{ext}}$$
$$\mathbf{x}^{**} = \mathbf{x}^* + \Delta t^2\mathbf{M}^{-1}\mathbf{f}^{\text{stretch}}(\mathbf{x}^{**})$$
$$\Rightarrow \mathbf{x}^{t+\Delta t} = \mathbf{x}^{**} + \Delta t^2\mathbf{M}^{-1}\mathbf{f}^{\text{volume}}(\mathbf{x}^{t+\Delta t})$$

The operator splitting formulation allows us to use two different linear system solvers for the stretching and volume forces. But, splitting both terms only yields a **weak coupling** between both forces ($\mathbf{f}^{\text{volume}}(\mathbf{x}^{t+\Delta t})$ depends on $\mathbf{f}^{\text{stretch}}(\mathbf{x}^{**})$, but not the other way around), while solving the system with both forces yields a **strong coupling**.

Stretching Part

We use linear tetrahedral elements, so the stretching forces are equivalent to the formulation in *FEM*, but where we only consider the stretching part of the stress tensor:

$$
\begin{aligned}
(\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3) &= -V\mathbf{P}^{\text{stretch}}(\mathbf{F})\mathbf{D}_m^{-T} \\
\mathbf{f}_4 &= -\mathbf{f}_1 - \mathbf{f}_2 - \mathbf{f}_3 \\
\boxed{\mathbf{P}^{\text{stretch}}(\mathbf{F}) &= 2\mu(\mathbf{F} - \mathbf{R})}
\end{aligned}
$$

We approximate force $\mathbf{f}^{\text{stretch}}(\mathbf{x}^{**})$ for *unknown* positions $\mathbf{x}^{**}$ by a Taylor series of first order:

$$
\mathbf{f}^{\text{stretch}}(\mathbf{x}^{**}) \approx \mathbf{f}^{\text{stretch}}(\mathbf{x}^{*}) + \frac{\partial \mathbf{f}^{\text{stretch}}(\mathbf{x}^{*})}{\partial \mathbf{x}} \Delta \mathbf{x}
$$

where $\Delta \mathbf{x} = \mathbf{x}^{**} - \mathbf{x}^{*}$. Substituting this linear approximation in the implict Euler equation and rearranging yields:

$$
\begin{aligned}
\Delta \mathbf{x} &= \Delta t^2 \mathbf{M}^{-1} \left(\mathbf{f}^{\text{stretch}}(\mathbf{x}^{*}) + \frac{\partial \mathbf{f}^{\text{stretch}}(\mathbf{x}^{*})}{\partial \mathbf{x}} \Delta \mathbf{x} \right) \\
\Rightarrow \boxed{\underbrace{\left(\mathbf{M} - \Delta t^2 \frac{\partial \mathbf{f}^{\text{stretch}}(\mathbf{x}^{*})}{\partial \mathbf{x}} \right)}_{\mathbf{A}} \Delta \mathbf{x} &= \Delta t^2 \mathbf{f}^{\text{stretch}}(\mathbf{x}^{*})}
\end{aligned}
$$

$\mathbf{A}$ is *constant*, since

- $\mathbf{R}$ is assumed to be *constant for one time-step*.
- The gradient w.r.t. $\mathbf{x}$ is constant, since $\mathbf{f}^{\text{stretch}}$ depends on $\mathbf{P}^{\text{stretch}}(\mathbf{F})$, whose derivative w.r.t. $\mathbf{x}$ is constant, since $\mathbf{P}^{\text{stretch}}(\mathbf{F})$ linearly depends on $\mathbf{F}$, and $\mathbf{F}$ linearly depends on $\mathbf{x}$.
- Δt is assumed to be constant.

Therefore, $\mathbf{A}$ can be factorized in a preprocessing step (e.g. using *Cholesky factorization*) and be used in the simulation for very efficient system solving in *linear time*.

Due to the precomputed factorization the stretching part can be solved *very efficiently*. Since this is often the most time-consuming part, we achieve a significant speedup.

Volume Part

The volume part of the stress tensor for $\mathbf{f}^{\text{volume}}$ is

$$
\mathbf{P}^{\text{volume}}(\mathbf{F}) = \lambda \text{tr}(\mathbf{R}^T\mathbf{F} - \mathbf{I})\mathbf{R}
$$

Meaning, the derivative of $\mathbf{P}^{\text{volume}}(\mathbf{F})$ w.r.t. $\mathbf{x}$ is *not constant*, since it depends on $\mathbf{R}$, such that the system matrix is also *not constant*. The linear system is therefore solved using a *conjugate gradient method*.

3.3.6. Comparison

- A method is *poorly conditioned*, if the system matrix $\mathbf{A}$ in $\mathbf{Ax} = \mathbf{b}$ is almost *singular* (meaning, determinant is zero), such that iterative solvers like conjugate gradient take a lot of time to converge.
- A block in the system matrix is non-zero:
  - *Primal methods*: if two nodes/particles are linked (e.g. by a spring force or finite element)
  - *Dual methods*: if two constraints have a common node/particle
- *Primal methods* are poorly conditioned *when there are high stiffness ratios* (if stiffness dominates the mass term).
- *Dual methods* are poorly conditioned, *when there are high mass ratios* (within one body).

3.4. Deformable Solids: Extensions

3.4.1. Degenerate & Inverted Elements

Degenerate or *inverted finite elements* can cause artifacts or instabilities in simulations due to inappropriate constitutive laws, and while *remeshing* can solve this, it is impractical for interactive applications like computer games.

We handle these cases by modifying the constitutive laws for such configurations *(stability > accuracy)*.

Assuming

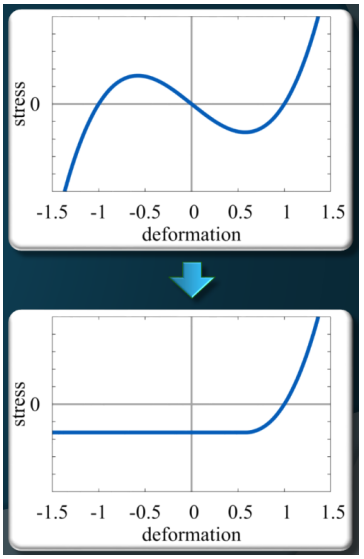
- Deformation gradient is given by SVD: $\mathbf{F} = \mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T$
- Stress tensor $\mathbf{P}$ is invariant under rotations of a deformable object: $\mathbf{P}(\mathbf{UF}) = \mathbf{UP}(\mathbf{F})$
- Stress tensor $\mathbf{P}$ is invariant under rotations of material space (isotropy): $\mathbf{P}(\mathbf{FV}^T) = \mathbf{P}(\mathbf{F})\mathbf{V}^T$

we diagonalize the stress tensor as:

$$\boxed{\mathbf{P}(\mathbf{F}) = \mathbf{U}\mathbf{P}(\hat{\mathbf{F}})\mathbf{V}^T = \mathbf{U}\hat{\mathbf{P}}\mathbf{V}^T}$$

which yields the relationship between the stress and the deformation gradient for the principle directions.

In diagonal $\hat{\mathbf{F}}$, we can directly check, if the diagonal entries are in a degenerate or inverted case (e.g. negative entry, or value < 0.577 for St. Venant-Kirchhoff model). We modify the diagonal deformation gradient to $\hat{\mathbf{F}}'$ (e.g. max(val, 0.577)), which yields the corresponding modified stress tensor $\mathbf{P}'$ with the desired material behavior.



3.4.2. Plastic Deformation

To simulate **plastic deformation** we divide the strain in an *elastic* and a *plastic* part:

$$\mathbf{E} = \mathbf{E}_e + \mathbf{E}_p$$

For each simulation step:

1. Determine current total strain $\mathbf{E}$.
2. Update $\mathbf{E}_p$ via

$$\mathbf{E}_p := \begin{cases} \mathbf{E}_p + \Delta t \gamma_{\text{creep}} \mathbf{E}_e & \gamma_{\text{yield}} < ||\mathbf{E}_e|| < \gamma_{\max} \\ \gamma_{\max} \frac{\mathbf{E}_p}{||\mathbf{E}_p||} & ||\mathbf{E}_e|| \geq \gamma_{\max} \end{cases}$$

   - γ_{yield}: start of plastic deformation
   - γ_{creep}: how fast energy is absorbed by plastic deformation
   - $\gamma_{\max}$: maximal plastic strain
3. Determine elastic forces, which are further used in other equations, as

$$\mathbf{E}_e = \mathbf{E} - \mathbf{E}_p$$

4.1. Fluids: Foundations

▼ *Additions*

- *Dirac-Delta distribution*:

$$\delta(x) = \begin{cases} \infty & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}, \quad \int_{-\infty}^{+\infty} \delta(x)dx = 1$$

We only consider the simulation of **incompressible fluids**, as many real fluids are. This means, that the *fluid density is constant over time* (volume + mass = constant).

In particular, we are interested in **Newtonian fluids**. A Newtonian fluid is one whose viscosity is not affected by shear rate: all else being equal, flow speeds or shear rates do not change the viscosity.

4.1.1. Equations of Motion

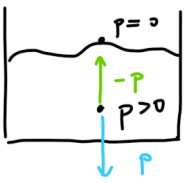
To derive the equations of motion for a *Newtonian fluid* we define a stress tensor for the fluid material and substitute it in the momentum equation.

A **continuity equation** (or transport equation) describes the transport of some quantity, in particular, its conservation. In fluid dynamics, the continuity equation states that the rate at which mass enters a system is equal to the rate at which mass leaves the system plus the accumulation of mass within the system:

$$\frac{D\rho}{Dt} = -\rho\nabla \cdot \mathbf{v}$$

For incompressible fluids the density does not change over time, which simplifies the continuity equation to a *volume* continuity equation:

$$\frac{D\rho}{Dt} = -\rho\nabla \cdot \mathbf{v} \overset{!}{=} 0 \quad \Rightarrow \quad \boxed{\nabla \cdot \mathbf{v} = 0}$$



We define the stress tensor of a Newtonian fluid to be

$$\begin{aligned} \mathbf{P} &= -p\mathbf{I} + \mu 2\dot{\epsilon} \\ &= -p\mathbf{I} + \mu\left(\nabla\mathbf{v} + (\nabla\mathbf{v})^T\right) \end{aligned}$$

- p: pressure, assumed to be equally scaled in all directions (cf. $\mathbf{I}$); $-p$, since particle wants to go away from pressure
- $\mathbf{v}$: particle velocity
- μ: *dynamic viscosity* (inner friction; honey = high, water = low)
- $\dot{\epsilon}$: *strain rate* tensor based on the Cauchy strain

which we plug in the *momentum equation*, rearrange and simplify for $\nabla \cdot \mathbf{v} = 0$, which yields the **Navier-Stokes equations** for incompressible fluids:

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \mu \nabla^2 \mathbf{v} + \mathbf{f}$$

$$\boxed{\frac{D\rho}{Dt} = 0 \Leftrightarrow \nabla \cdot \mathbf{v} = 0 \qquad \frac{D\mathbf{v}}{Dt} = \underbrace{-\frac{1}{\rho}\nabla p}_{A} + \underbrace{\nu \nabla^2 \mathbf{v}}_{B} + \underbrace{\frac{\mathbf{f}}{\rho}}_{C}}$$

- $\nu = \frac{\mu}{\rho}$: *kinematic viscosity*
- A: Acceleration due to pressure differences in the fluid. The pressure force is the most important part, since it conserves the volume and therefore the density of the fluid, and should be solved using an *implicit* solver.
- B: Acceleration due to friction forces between particles with different velocities. Here, we can use an *explicit* solver.
- C: (Constant) External forces (e.g. gravitation) per unit volume

---

4.1.2. Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) is a *mesh-free* Lagrangian method, which is used to simulate the mechanics of *continuum media*, such as deformable solids and fluids. In SPH, we discretize the fluid / body using a finite number of particles *(→ "Particle" in SPH)*. All quantities are approximated using a smooth kernel function on neighboring particles *(→ "Smoothed" in SPH)*.

Note, that since the particles represent the discretized integration elements, particles are actually *cubes* with a side length of d, such that $m = d^3 \rho_0$.

Given a function $f : \Omega \to \mathbb{R}^n$ (e.g. temperature, pressure), it can be rewritten using the Dirac delta identity and approximated using a **kernel function** W with *compact support* h:

$$\begin{aligned} f(\mathbf{x}) &= \int_\Omega f(\mathbf{x}^*)\delta(\mathbf{x} - \mathbf{x}^*)d\mathbf{x}^* \\ &\approx \int_{\mathcal{D}_\mathbf{x}} f(\mathbf{x}^*)W(\mathbf{x} - \mathbf{x}^*, h)d\mathbf{x}^* \end{aligned}$$

$\mathcal{D}_\mathbf{x}$ represent the *spherical* domain around $\mathbf{x}$, where the kernel is not zero.

Kernel Function

A kernel function is a *weighting function*, that assigns weights to value contributions of neighboring particles depending on their distance to particle in question $\mathbf{x}$. It should have a **compact support** (area, in which values are non-zero, but zero everywhere else), must fulfill a number of conditions and must be differentiable:

- **Mathematical Conditions**
  - *Normalization condition*: $\int W(\mathbf{x} - \mathbf{x}^*, h)d\mathbf{x}^* = 1$
  - *Delta function property*: $\lim_{h \to 0} W(\mathbf{x} - \mathbf{x}^*, h) = \delta(\mathbf{x} - \mathbf{x}^*)$
  - *Compact condition* with support radius r: $W(\mathbf{x} - \mathbf{x}^*, h) = 0$ if $||\mathbf{x} - \mathbf{x}^*|| > r$
- **Physical Conditions**
  - *Symmetry condition*: $W(\mathbf{x} - \mathbf{x}^*, h) = W(\mathbf{x}^* - \mathbf{x}, h)$
  - *Non-negative condition*: $W(\mathbf{x} - \mathbf{x}^*, h) \geq 0$

The integral from above is discretized into a *sum over particles* with positions $\mathbf{x}_j$ and volumes $V_j = \frac{m_i}{\rho_j}$:

$$\boxed{f(\mathbf{x}) \approx \sum_j V_j f(\mathbf{x_j}) W(\mathbf{x} - \mathbf{x}_j, h)}$$

In the original SPH formulation a quantity A_i is approximated by the quantities A_j at the neighboring locations $\mathbf{x}_j$:

$$\boxed{A_i \approx \sum_j \frac{m_j}{\rho_j} A_j W_{ij}} \quad W_{ij} = W(\mathbf{x}_i - \mathbf{x}_j, h)$$

Other quantities are

- *Gradient*: $\nabla A_i \approx \sum_j \frac{m_j}{\rho_j} A_j \nabla W_{ij}$
- *Laplace*: $\nabla^2 A_i \approx \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W_{ij}$
- *Curl*: $(\nabla \times \mathbf{A})_i \approx -\sum_j \frac{m_j}{\rho_j} \mathbf{A}_j \times \nabla W_{ij}$

The standard formulations are usually not used in practice. Additionally, we want to avoid the usage of the Laplacian due to the sign change, which causes instability.

▼ **Preferred Variants**

- *Symmetric formula*: preserves linear and angular momentum when used for pressure forces

$$\nabla A_i \approx \rho_i \sum_j m_j \left(\frac{A_i}{\rho_i^2} + \frac{A_j}{\rho_j^2} \right) \nabla W_{ij}$$

- *Difference formula*: estimates results in a more accurate discretization

$$\nabla A_i \approx \sum_j \frac{m_j}{\rho_j} (A_j - A_i) \nabla W_{ij}$$

- *Laplacian*: combine the SPH gradient with a finite difference derivative to get a more robust formulation; for $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$:

$$\nabla^2 A_i \approx 2 \sum_j \frac{m_j}{\rho_j} (A_i - A_j) \frac{\mathbf{x}_{ij} \cdot \nabla W_{ij}}{\mathbf{x}_{ij}^T \mathbf{x}_{ij} + 0.01h^2}$$

  ($0.01h^2$ avoids division by zero, since $h > 0$)

- *Symmetric Curl*: preserves energy

$$(\nabla \times \mathbf{A})_i^{\text{symm}} \approx -\rho_i \sum_j m_j \left(\frac{\mathbf{A}_i}{\rho_i^2} + \frac{\mathbf{A}_j}{\rho_j^2} \right) \times \nabla W_{ij}$$

- *Difference Curl*: better at free surfaces

$$(\nabla \times \mathbf{A})_i^{\text{diff}} \approx \frac{1}{\rho_i} \sum_j m_j (\mathbf{A}_i - \mathbf{A}_j) \times \nabla W_{ij}$$

- Kernel Derivative -

We separate the kernel derivative in a *scalar* $\left(\frac{\partial W(q)}{\partial q} \frac{1}{h||\mathbf{x}_i - \mathbf{x}_j||} \right)$ and a *vector* part $(\mathbf{x}_i - \mathbf{x}_j)$:

$$\nabla W_{ij} = \frac{\partial W(q)}{\partial q} \nabla q = \frac{\partial W(q)}{\partial q} \frac{\mathbf{x}_i - \mathbf{x}_j}{h||\mathbf{x}_i - \mathbf{x}_j||}$$

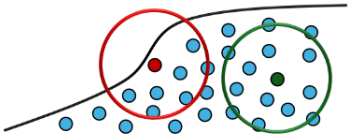
- $q = \frac{||\mathbf{x}_i - \mathbf{x}_j||}{h}$

Then, we can simply create a *lookup table* with regular sampled evaluations within the compact support *for the scalar part* and interpolate, and perform a standard evaluation for the vector part.

This approach can yield very accurate results with a low memory overhead, and achieves a performance gain of ~30%.

Particle Deficiency Problem

Particles at the free surface often do not have enough neighbors to get a good approximation.



- Shepard Filter -

The **Shepard Filter** introduces a *correcting factor* s_i to the kernel computation, that considers the amount of neighborhood volume filled:

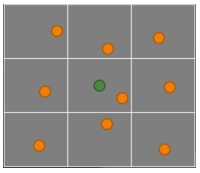
$$\tilde{W}_{ij} = s_i W_{ij} = \frac{1}{\sum_k \frac{m_k}{\rho_k} W_{ik}} W_{ij}$$

- For an ideally filled neighborhood, $\sum_k \frac{m_k}{\rho_k} W_{ik} = 1$.
- We cannot use this correction for density computations due to ρ_k being in the denominator.

4.1.3. Neighborhood Search

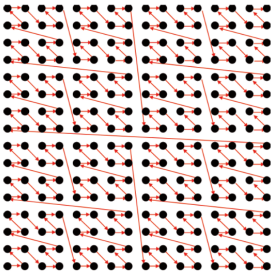
As we need > 30-40 neighbors for the SPH computations for each particle, an efficient neighborhood search is essential.

Determining the bounding box containing all particles, we divide it into a **spatial grid** with each cell size being the *support radius of the kernel* r. Then, we can map point (x, y, z) to the cell index $(i = \lfloor x/r \rfloor, j = \lfloor y/r \rfloor, k = \lfloor z/r \rfloor)$ and get the cells of all potential neighbor particles within 27 cells $(i \pm 1, j \pm 1, k \pm 1)$.



Since the spatial grid is 3D, the memory requirements are in $\mathcal{O}(n^3)$. We therefore map the 3D spatial grid on a 1D **hash table**, which only stores non-empty cells. Cells then can be found with the expected complexity of $\mathcal{O}(1)$. We require a fixed hash table length though.

To better utilize CPU caches, we can additionally perform **spatial sorting** (e.g. *z-index sort*) of the cells to reduce cache misses. We usually resort after a certain amount of simulation steps (100-500 steps).



z-index sort

4.2. Fluids: Pressure Solver & Boundary Handling

Pressure forces counteract volume compression, and maintaining *incompressibility* is crucial for realistic fluid behavior, though *computationally expensive*. We differentiate between explicit and implicit pressure solvers.

- ***Explicit Pressure Solvers*** allow only small time steps to avoid instability.
- ***Implicit Pressure Solvers*** permit larger time steps but are more complex to compute.

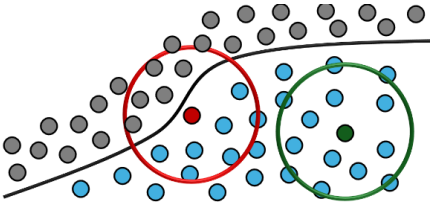
Using the SPH formulation, the *density* is determined as:

$$\rho_i = \sum_j \frac{m_j}{\rho_j} \rho_j W_{ij} = \sum_j m_j W_{ij}$$

We additionally need to deal with the *Particle Deficiency Problem*, where the particles at the free surface do not have enough neighbors to get good approximations, especially for the density. This lead to *negative pressure*, where neighboring particles are "drawn" to the free surface to counteract missing density, resulting in *particle clumping*.

Possible solutions are:

- *Simple*: clamp negative pressures to zero (mostly works)
- *Better, but expensive*: use *ghost particles*



4.2.1. Pressure Solvers

💡 ***Summary:***

- *Explicit Pressure Solvers*:
  - ***Weakly Compressible SPH (WCSPH)***: Locally determine pressure via *Equation of State (EOS)* mimicking an almost incompressible fluid, discretize pressure and viscosity in the Navier-Stokes equation using SPH, and enforce more stability with the *Courant-Friedrichs-Levy (CFL) Condition* on the time-step size.
- *Implicit Pressure Solvers:*
  - ***Implicit Incompressible SPH (IISPH):*** Reformulate and discretize the *continuity equation* to be a system of linear equations of particle pressures p_i, iteratively solve the system row-wise using the *relaxed Jacobi method*.
  - ***Position Based Fluids (PBF):*** Assume *EOS* to be the position-based constraint to enforce incompressibility, that is solved using PBD.
  - ***Divergence-Free SPH (DFSPH):*** Recycle part of the *constant density solver* (*IISPH*) to also enforce a *divergence-free velocity field* afterwards.

Explicit Pressure Solvers

- Weakly Compressible SPH (WCSPH) -

We can locally determine pressure by the density using the **Equation of State (EOS)**:

$$\boxed{p_i = \frac{\kappa \rho_0}{\gamma}\left(\left(\frac{\rho_i}{\rho_0}\right)^{\gamma} - 1\right)}$$

where constants κ, γ define the stiffness. If $\gamma = 1$, we get $p_i = \kappa(\rho_i - \rho_0)$, which is similar to a spring force. The relationship between pressure and density, as defined by the EOS, ensures that for small changes in density, the pressure changes more significantly, mimicking a nearly incompressible flow (hence, *"weakly"* compressible formulation).

With that, we compute the unknown quantities (namely, the *pressure* and *viscosity part*) in the *Navier-Stokes Equation* as a SPH discretization:

$$\frac{D\mathbf{v}}{Dt} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{v} + \frac{\mathbf{f}}{\rho}$$

- **Pressure** (cf. *symmetric formula of SPH gradient*)

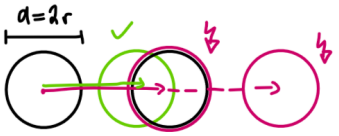
$$\boxed{-\frac{\nabla p_i}{\rho_i} = -\sum_j m_j\left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right)\nabla W_{ij}}$$

- **Viscosity** (cf. *Laplacian SPH formulation*) with $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j, \mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$ and $d = \text{dimension}$:

$$\boxed{\nu\nabla^2\mathbf{v}_i = 2(d+2)\nu\sum_j \frac{m_j}{\rho_j}\frac{\mathbf{v}_{ij}\cdot\mathbf{x}_{ij}}{||\mathbf{x}_{ij}||^2 + 0.01h^2}\nabla W_{ij}}$$

For the explicit pressure solver, we additionally utilize the **Courant-Friedrichs-Levy (CFL) condition**, that is *necessary (but not sufficient)* for the convergence of numerical solvers for differential equations. It proves an upper bound for the time step size:

$$\boxed{\Delta t \leq \lambda\frac{d}{||\mathbf{v}_{max}||}}$$



where λ is a user-defined scaling parameter. Intuitively, we want to limit the path length, such that with the maximal possible path length (as given by $\mathbf{v}_{max} \cdot \Delta t$), a particle cannot interfere or even cross any other particle. It cannot strongly guarantee stability, but leads in practice to stable simulations, if λ is chosen well.

Algorithm:

for all *particle* i **do**
  find neighbors j
for all *particle* i **do**
  $\rho_i = \sum_j m_j W_{ij}$
  compute p_i from ρ_i with a state equation
for all *particle* i **do**
  $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i}\nabla p_i$
  $\mathbf{F}_i^{viscosity} = m_i \nu \nabla^2 \mathbf{v}_i$
  $\mathbf{F}_i^{other} = m_i \mathbf{g}$
  $\mathbf{F}_i(t) = \mathbf{F}_i^{pressure} + \mathbf{F}_i^{viscosity} + \mathbf{F}_i^{other}$
for all *particle* i **do**
  $\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \Delta t \mathbf{F}_i(t)/m_i$
  $\mathbf{x}_i(t+\Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t+\Delta t)$

Implicit Pressure Solvers

To simulate (almost) incompressible fluids using an *equation of state*, we need a high stiffness coefficient. However, for a high stiffness value the explicit pressure solve tends to get unstable. So, we derive a linear system for the unknown pressure values and therefore derive an *implicit pressure solver*.

- Implicit Incompressible SPH (IISPH) -

1) The Linear System

Starting with the *continuity equation*,

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v}$$

we discretize and approximate values of interest to yield the desired linear system of equations:

- For one particle i, discretize left *time-derivative* using a linear approximation (cf. *finite differences* with $h = \Delta t$), and right *space-derivative* using a modified SPH difference formula for divergence on the *implicit* quantity $\mathbf{v}_i(t+\Delta t)$:

$$\frac{\rho_i(t+\Delta t) - \rho_i(t)}{\Delta t} = \sum_j m_j \mathbf{v}_{ij}(t+\Delta t) \cdot \nabla W_{ij},$$

  with $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$.

- Approximate unknown velocities $\mathbf{v}_i(t+\Delta t)$ by an *Euler integration step* (= first-order Taylor) and differentiate between *pressure* ($\mathbf{a}_i^{\mathrm{p}}$) and *non-pressure* ($\mathbf{a}_i^{\mathrm{np}}$) acceleration:

$$\mathbf{v}_i(t+\Delta t) \approx \mathbf{v}_i(t) + \Delta t \left(\mathbf{a}_i^{\mathrm{p}}(t) + \mathbf{a}_i^{\mathrm{np}}(t)\right)$$

- Substituting the new velocities in the rhs yields a known sum and an unknown sum:

$$\frac{\rho_i(t+\Delta t) - \rho_i(t)}{\Delta t} = \underbrace{\sum_j m_j \mathbf{v}_{ij}^*(t) \cdot \nabla W_{ij}}_{known} + \underbrace{\Delta t \sum_j m_j \mathbf{a}_{ij}^P(t) \cdot \nabla W_{ij}}_{unkown}$$

  - $\mathbf{v}_{ij}^* = \mathbf{v}_i^* - \mathbf{v}_j^*$ with $\mathbf{v}_i^* = \mathbf{v}_i(t) + \Delta t \mathbf{a}_i^{\mathrm{np}}$ and $\mathbf{v}_i(t+\Delta t) \approx \mathbf{v}_i^*(t) + \Delta t \mathbf{a}_i^{\mathrm{p}}(t)$

- Move the known part of the rhs to the lhs is equivalent to performing an Euler step on $\rho_i(t)$ to yields the density after applying all non-pressure forces:

$$\rho_i^*(t) = \rho_i(t) + \Delta t \frac{D\rho_i}{Dt} = \rho_i(t) + \Delta t \sum_j m_j \mathbf{v}_{ij}^* \cdot \nabla W_{ij}$$

- Substitute $\rho_i(t+\Delta t) = \rho_0$ due to the *incompressibility constraint* , which yields the linear equation $\mathbf{Ap} = \mathbf{s}$ with *source term* $\mathbf{s}$ (so called *Pressure Poisson Equation*)

$$\boxed{\underbrace{\frac{\rho_0 - \rho_i^*(t)}{\Delta t}}_{\mathbf{s}} = \underbrace{\Delta t \sum_j m_j \mathbf{a}_{ij}^{\mathrm{p}}(t) \cdot \nabla W_{ij}}_{\mathbf{Ap}}}$$

However, if we we allow the pressure accelerations $\mathbf{a}_i^{\mathrm{p}}(t)$ to have all 3 degrees of freedom, the system becomes *underdetermined*. Thus, we express them using the *symmetric gradient formulation*, which results in only one unknown, namely p_i:

$$\mathbf{a}_i^{\mathrm{p}} = -\frac{1}{\rho_i}\nabla p_i = -\sum_j m_i \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right) \nabla W_{ij}$$

2) Solving The System

Instead of directly solving the huge linear system of equations, we want to solve it row-wise via an iterative method.

Since one row of the linear system is defined as $\sum_j a_{ij} p_j = s_i$, we can directly compute the source term s_i using the predicted densities, and the lhs by first computing $\mathbf{a}_i^{\mathrm{p}}$ as above and then $(\mathbf{Ap})_i$:

Having both $\sum_j a_{ij} p_j$ and s_i computed, we can solve the system iteratively using a *relaxed Jacobi method* with relaxation factor $\omega = 0.5$:

$$\boxed{p_i^{(l+1)} = p_i^{(l)} + \frac{\omega}{a_{ii}} \left(s_i - \sum_j a_{ij} p_j^{(l)}\right)}$$

The diagonal elements can be computed by accumulating all coefficients of p_i in $\sum_j a_{ij} p_j = s_j$:

$$\boxed{a_{ii} = -\frac{\Delta t}{\rho_i^2} \left(\left|\left|\sum_j m_j \nabla W_{ij}\right|\right|^2 + \sum_j ||m_j \nabla W_{ij}||^2\right)}$$

Algorithm:

1. Perform neighborhood search to identify neighboring particles.
2. Compute particle densities ρ_i and diagonal matrix elements a_{ii}.
3. Compute predicted velocities $\mathbf{v}_i^*$ considering only computed non-pressure forces.
4. Compute source terms s_i.
5. Loop:
    a. Compute pressure accelerations $\mathbf{a}_i^{\mathrm{p}}$.
    b. Determine $\sum_j a_{ij} p_j$.
    c. Update pressure values using the *relaxed Jacobi method*.
6. Time integration considering pressure accelerations.

- Position Based Fluids (PBF) -

The main idea is to define the *equation of state* as a *position constraint* to enforce incompressibility, and to solve the constraint to get new positions:

$$\boxed{C_i(\mathbf{x}) = \frac{\rho_i}{\rho_0} - 1}$$

We get the same system of linear equations $\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\boldsymbol{\lambda} = -\mathbf{C}(\mathbf{x}^*)$ (cf. *constraint projection in PBD*) and similar Lagrange multiplier for a particle, with the difference of using ϵ to avoid division by zero:

$$\lambda_i = -\frac{C_i(\mathbf{x}^*)}{\sum_j \frac{1}{m_j}||\nabla_{\mathbf{x}_j} C(\mathbf{x}^*)||^2 + \epsilon}$$

With the SPH constraint and its gradients,

$$\begin{aligned} C_i(\mathbf{x}) &= \frac{\rho_i}{\rho_0} - 1 = \frac{1}{\rho_0}\sum_j m_j W_{ij} - 1 \\ \nabla C_i &= \frac{\partial C_i}{\partial \mathbf{x}_i} = \frac{1}{\rho_0}\sum_j m_j \nabla W_{ij} \\ \nabla_{\mathbf{x}_j} C_i &= \frac{\partial C_i}{\partial \mathbf{x}_j} = -\frac{1}{\rho_0} m_j \nabla W_{ij} \end{aligned}$$

we get the following final symmetric position update:

$$\boxed{\Delta\mathbf{x}_i = \frac{1}{m_i \rho_0}\sum_j m_j(\lambda_i + \lambda_j)\nabla W_{ij}}$$

PBF is a modified version of IISPH and could also be implemented in a similar way to IISPH. The main difference is, that in the solver loop the gradient at time $t + h$ is used instead of the gradient at time t since we use the predicted positions in each iteration. This lead to a better convergence but is computationally more expensive.

Algorithm:

1. Time iteration via symplectic Euler to predict candidate particle positions $\mathbf{x}_i^*$ *from external forces only*.
2. Perform *neighborhood search* to find $N_i(\mathbf{x}_i^*)$.
3. Loop:
   a. Compute *Lagrange multipliers* λ_i.
   b. Determine position corrections $\Delta\mathbf{x}_i$ and apply corrections as $\mathbf{x}_i^* = \mathbf{x}_i^* + \Delta\mathbf{x}_i$.
4. Update velocities with $\mathbf{v}_i(t + \Delta t) = \frac{1}{\Delta t}(\mathbf{x}_i(t + \Delta t) - \mathbf{x}_i(t))$ and $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i^*$.

- Divergence-Free SPH (DFSPH) -

Note, that for an incompressible fluid the continuity equation yields:

$$\frac{D\rho}{Dt} = 0 \Leftrightarrow \nabla \cdot \mathbf{v} = 0$$

Although this holds in a continuous setting, in a discrete one it *does not*, since enforcing a divergence-free velocity field also yields numerical errors in the density field that sum up over time (”jittering” of the particles).

Most previous works enforce only a constant density condition ($\rho - \rho_0 = 0$) and neglect the divergence-free condition. **Divergence-free SPH (DFSPH)** combines a *constant density solver* with a *divergence-free solver*. This increases the *stability of the simulation* and therefore allows larger time steps which yields a significant performance gain.

DFSPH first solves the linear system for the *constant density* constraint (as in *IISPH*). Then, it solves a second linear system for the divergence source term *(note the same rhs)*, which enforces a *divergence-free velocity field*:

$$\underbrace{\rho_i \nabla \cdot \mathbf{v}_i}_{\mathbf{s}} = \underbrace{\Delta t \sum_j m_j \mathbf{a}_{ij}^{\mathrm{p}}(t) \cdot \nabla W_{ij}}_{\mathbf{Ap}}$$

Since the rhs is the same as for the first solver, we can reuse the values computed on the rhs (especially the diagonal entries a_{ii}), such that applying a second solver is not that expensive. Additionally, the second solver improves the overall performance since keeping the divergence error small leads to a small density error in the next simulation step. So, less iterations are required.

Algorithm:

For one simulation step:

1. Predict next candidate velocities $\mathbf{v}_i^*$ using non-pressure forces only.
2. Adapt $\mathbf{v}_i^*$ using pressure acceleration $\mathbf{a}_i^{\mathrm{p}}$ from pressure values predicted by the *constant density solver* (*IISPH*).
3. Update all position: $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i^*$.
4. Update all neighbors, densities and diagonal values a_{ii} to match $\mathbf{x}_i(t + \Delta t)$.
5. Adapt $\mathbf{v}_i^*$ using pressure acceleration $\mathbf{a}_i^{\mathrm{p}}$ from pressure values predicted by the *divergence-free solver*.
6. Update all velocities: $\mathbf{v}_i^*(t + \Delta t) = \mathbf{v}_i^*$.

4.2.2. Boundary Handling

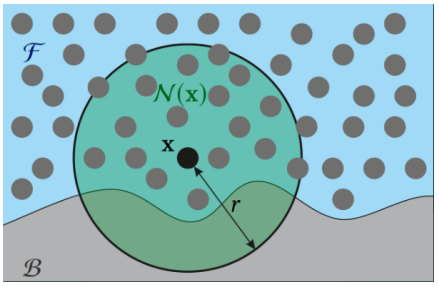
Summary:

- Extend density field into boundary domain and let pressure solver prevent boundary penetration.
- Divide pressure at particle into *pressure in fluid* and *pressure in boundary*. Pressure in boundary is approximated using:
  - ***Particle-Based Boundary Representation:*** Discretize boundary as particles and add the SPH boundary particle density contribution to the density of a fluid particle. The boundary has rest density ρ_0.
  - ***Implicit Boundary Representation***: Express the volume integral of boundary density contribution $\rho_{\mathcal{B}}$ using a *smooth* transition function on a signed distance field, efficiently use it by discretizing the volume integral in a spatial grid and interpolating during simulation.

SPH solvers are often based on the concept of virtually extending the density field into the boundary domain. By extending the density field into the boundary domain, the pressure solver *implicitly* resolves boundary

penetrations as the virtual density contributions lead to local compression that violate the incompressibility constraint.

Extending the density field into the boundary means that we assume that the boundary has the *rest density* ρ_0 of the fluid. Hence, if a particle gets closer to the boundary its current density ρ_i increases and the pressure solver prevents a penetration.



Given the (extended) density field, we split the integral to compute the density at position $\mathbf{x}$ into a fluid ($\mathcal{F}$) and boundary ($\mathcal{B}$) part:

$$\begin{aligned}\rho(\mathbf{x}) &\approx \int_{\mathcal{N}(\mathbf{x})} \rho(\mathbf{x}')W(||\mathbf{x}-\mathbf{x}'||,h)d\mathbf{x}' \\ &= \int_{\mathcal{N}(\mathbf{x})\cap\mathcal{F}} \rho(\mathbf{x}')W(||\mathbf{x}-\mathbf{x}'||,h)d\mathbf{x}' + \int_{\mathcal{N}(\mathbf{x})\cap\mathcal{B}} \rho(\mathbf{x}')W(||\mathbf{x}-\mathbf{x}'||,h)d\mathbf{x}' \\ &= \rho_{\mathcal{F}}(\mathbf{x}) + \rho_{\mathcal{B}}(\mathbf{x})\end{aligned}$$

$\rho_{\mathcal{F}}$ is typically approximated using the standard SPH formulation. $\rho_{\mathcal{B}}$ can be approximated based on a **Particle-Based Boundary Representation** or an **Implicit Boundary Representation**.

Particle-Based Boundary Representation

The core idea of particle-based approaches is to discretize the boundary geometry with particles, which serve as additional sampling points.

We extend the SPH equation for the density by a second sum over the contribution *neighboring boundary* particles k:

$$\rho(\mathbf{x}) = \rho_{\mathcal{F}}(\mathbf{x}) + \rho_{\mathcal{B}}(\mathbf{x}) \approx \sum_j m_j W(||\mathbf{x}-\mathbf{x}_j||,h) + \sum_k \tilde{m}_k W(||\mathbf{x}-\mathbf{x}_j||,h)$$

$\tilde{m}_k$ are *pseudo masses* for each boundary particle, that solely fulfill the purpose of extending the fluid's density field into the boundary and are independent of the boundary material. Similarly, the volume represented by a rigid body particle is independent of mass and density:

$$V_i = \frac{m}{\rho_i} = \frac{m}{\sum_k m W_{ik}} = \frac{1}{\sum_k W_{ik}}$$

(Note, that this formulation is similar to the Shepard filter)

The density computation of a fluid particle is adapted as follows:

$$\boxed{\rho_i = \underbrace{\sum_j m_i W_{ij}}_{\text{fluid}} + \underbrace{\sum_k \rho_0 V_k W_{ik}}_{\text{rigid body}}}$$

The pseudo-mass of a boundary particle is determined by setting its density to ρ_0.

- Rigid-Fluid Coupling -

The rigid-fluid coupling is realized by the corresponding pressure forces between fluids and rigid bodies. *Pressure forces* between a fluid particle i and a rigid body particle j are

$$\mathbf{F}_{i \leftarrow j} = -\mathbf{F}_{j \leftarrow i} = -m_i \underbrace{\rho_0 V_j}_{\tilde{m}_j} \underbrace{\frac{p_i}{\rho_i^2} \nabla W_{ij}}_{\substack{\text{pressure} \\ \text{aceleration}}}$$

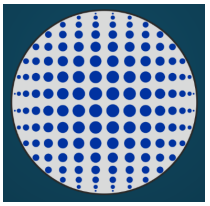
where the sum of both forces is zero. Here, we assume *weakly compressible* fluids with $\rho_i \approx \rho_j$ and $p_i \approx p_j$.

Implicit Boundary Representation

Instead of using the particle-based SPH approximation to determine $\rho_{\mathcal{B}}(\mathbf{x})$ we could alternatively apply *numerical integration*.

We can either use numerical integration, or **Gauss-Legendre quadrature**, which approximates the integral as a sum of weighted function values at specific points in the domain:

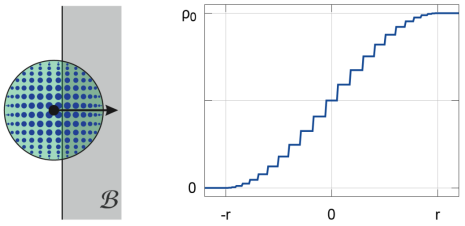
$$\int_{-1}^{1} f(x)dx \approx \sum_i w_i f(x_i)$$



The weights are given by a *fixed pattern*, as depicted in the dots of the circle (right). Solving the integral in the support radius of a particle yields higher weight for close particles.

However, we cannot simply use Gauss-Legendre quadrature to solve the integral in the domain $\mathcal{N}(\mathbf{x}) \cap \mathcal{B}$ when a fluid particle gets close to the boundary:

1. We have to integrate a discontinuous, piece-wise constant function; The density field is zero outside of the boundary and takes on ρ_0 inside the boundary, which leads to staircase artifacts in the resulting function due to the limited accuracy of the fixed pattern quadrature scheme.



2. The numerical integration is expensive and significantly decreases the performance of the simulation.

- First Problem -

Simplify the integral we have to compute as the volume overlap of the spherical domain and the boundary, so that the function we have to integrate is $f(x') = 1$:

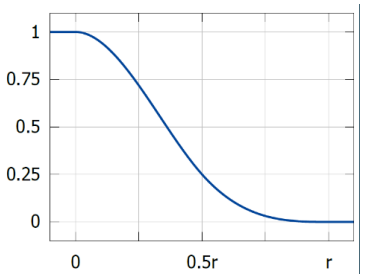
$$\rho_{\mathcal{B}}(\mathbf{x}) = \rho_0 V_{\mathcal{B}}(\mathbf{x}) = \rho_0 \int_{\mathcal{N}(\mathbf{x}) \cap \mathcal{B}} d\mathbf{x}'$$

This integral is not trivial to integrate, so we reformulate the integral to use the *signed distance function / field* Φ *(negative if inside the boundary*) and a *smooth* function f^*:

$$V_{\mathcal{B}}(\mathbf{x}) = \int_{\mathcal{N}(\mathbf{x})} f^*(\Phi(\mathbf{x}'))d\mathbf{x}', \quad f^*(x) = \begin{cases} C(x)/C(0) & \text{if } 0 < x < r \\ 1 & \text{if } x \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

where C is the cubic spline kernel and r the support radius.

For simple shapes we can directly compute the signed distance function. For complex shapes, we define sampling points on a regular grid, from which we precompute the SDF values and interpolate them during runtime.



Function f^* gives us a smooth transition from the area outside of the boundary to the inside.

- Second Problem -

To mitigate the issue of expensive numerical integration, we precompute $V_{\mathcal{B}}(\mathbf{x})$ on a grid, and interpolate the value during runtime. This works for rigid bodies only.

4.3. Fluids: Materials & Physical Phenomena

4.3.1. Viscosity

> 💡 ***Summary***
>
> Explicitly consider *viscosity* $\mu\nabla^2\mathbf{v}$ in the *Navier-Stokes equation* via two possible approaches:
>
> - ***Explicit Viscosity***: Directly express Laplacian using the *SPH finite difference formula for Laplacian's*, or smooth out velocity field to create an artificial viscosity without the need of the Laplacian (*XSPH*).
> - ***Implicit Viscosity***:
>   - *Takahashi et al. 2015*: Express viscous forces in terms of strain rates and solve implicit time integration on $\mathbf{v}$ using these forces.
>   - *Weiler et al. 2018*: Express viscous forces directly using the SPH Laplacian formulation, and solve implicitly.

The **viscosity force** describes the *frictions* between particles with different velocities.

As derived in the *Navier-Stokes equations*, the viscous force for incompressible fluids is

$$\mathbf{f}_{\text{visco}} = \mu\nabla^2\mathbf{v}$$

Recent SPH solvers compute this force by either

- directly determining the Laplacian of $\mathbf{v}$ (***Explicit Viscosity***), or
- computing the divergence of the **strain rate** $\mathbf{E} = \dot{\epsilon} = \frac{1}{2}(\nabla\mathbf{v} + (\nabla\mathbf{v})^T)$ (***Implicit Viscosity***)

Strain rate approaches must additionally enforce a *divergence-free* velocity field to avoid undesired bulking.

Explicit Viscosity

We can directly use the *standard SPH discretization of the Laplacian*, but it has two disadvantages:

- Kernel Laplacian has a sign change within the support radius
- Sensitivity to particle disorder

To avoid stability problem with the Laplacian, there are two popular solutions:

1. Solve one derivative numerically by finite differences (cf. *SPH Laplacian on quantity* $\mathbf{v}_{ij}$)

$$\nabla^2 v_i \approx 2(d+2)\sum_j \frac{m_j}{\rho_j}\frac{\mathbf{v}_{ij}\cdot\mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|^2 + 0.01h^2}\nabla W_{ij}$$
$$\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j,\ \mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j,\ d = \text{dimension}$$

   - *Galilean invariant* *(laws of motion are the same in all inertial frames of reference)*, varnishes for rigid body rotation, conserves linear and angular momentum

2. Smooth the velocity field, which yield an *artificial* viscosity. **Extended SPH (XSPH)** reduces the particle disorder by smoothing the velocity field, which is equivalent to a viscous behavior:

$$\hat{\mathbf{v}}_i = \mathbf{v}_i + \underbrace{\alpha \sum_j \frac{m_j}{\rho_j}(\mathbf{v}_j - \mathbf{v_i})W_{ij}}_{\mathbf{a}_i^{\text{visco}}}$$

- Laplacian not needed, but α is not physically meaningful

Summary: For low viscous flow, explicit methods are cheap and well-suited. The approximation of the Laplacian yields better results while XSPH is slightly faster.

Implicit Viscosity

For highly viscous material coefficients explicit methods (again) tend to become unstable, which forces us to reduce the time step size and therefore increases also the computation time.

- Takahashi et al. 2015 -

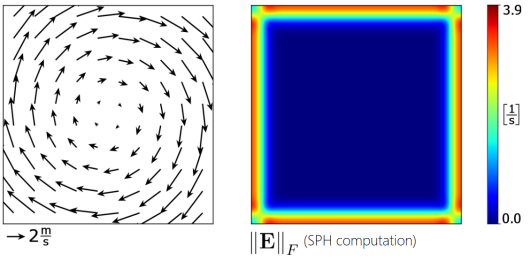
First, compute *strain rates* $\mathbf{E}_i$, express the viscous force as the divergence of the strain rate $\mathbf{E}_i$ via SPH discretization, and integrate *implicitly*:

$$\mathbf{f}_{\text{visco}} = \mu \nabla \cdot \left(\nabla \mathbf{v}_i + (\nabla \mathbf{v}_i)^T\right) = \mu \sum_j m_j \left(\frac{2\mathbf{E}_i}{\rho_i^2} + \frac{2\mathbf{E}_j}{\rho_j^2}\right) \nabla W_{ij}$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}^* + \frac{\Delta t}{\rho} \mathbf{f}_{\text{visco}}(t + \Delta t)$$

The linear system is solved using the *conjugate gradient method*.

However, computing the strain rate using SPH leads to errors at the free surface due to particle deficiency.



- Weiler et al. 2018 -

Authors use a similar implicit integration scheme, but compute the Laplacian as an SPH discretization:

$$\mathbf{v}(t + \Delta t) = \mathbf{v}^* + \frac{\Delta t}{\rho} \mu \nabla^2 \mathbf{v}(t + \Delta t)$$

$$\nabla^2 \mathbf{v}_i = 2(d+2) \sum_j \frac{m_j}{\rho_j} \frac{\mathbf{v}_{ij} \cdot \mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|^2 + 0.01h^2} \nabla W_{ij}$$

The linear system is solved using a *mesh-less* conjugate gradient method.

The direct Laplacian approximation avoids problems at the free surface.

Summary: For highly viscous fluids, implicit methods are recommended to guarantee stability. Strain rate based SPH formulations lead to artifacts at the free surface. Weiler et al. avoid this problem and generate more realistic results.

4.3.2. Surface Tension

Surface tension is especially important for small-scale effects, like simulating droplets, but can generally be neglected in larger scenes. To simulate surface tension, we need two forces:

- **Cohesion Force**: Attraction between two particles of the same kind
- **Surface Area Minimization**: Force creating the surface tension

We compute the *cohesion force* between two particles via

$$\mathbf{F}_{i \leftarrow j} = -\gamma m_i m_j C(|\mathbf{x}_i - \mathbf{x}_j|) \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}$$

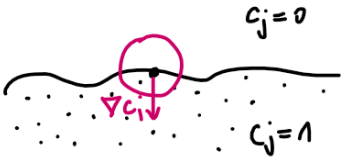
where C is a special splice kernel function.

The force computation for the *surface area minimization* is done by an approximation of the surface curvature as the difference between neighboring surface normals:

$$\mathbf{F}_{i \leftarrow j} = -\gamma m_i (\mathbf{n}_i - \mathbf{n}_j)$$

The normals are computes as *(h = support radius)*

$$\mathbf{n}_i = h \sum_j \frac{m_j}{\rho_j} \nabla W_{ij}$$



Intuitively, we use the gradient field introduced the a placeholder color field $\mathbf{c}_i = \sum_j \frac{m_j}{\rho_j} c_j W_{ij}$ *(which is equivalent to the Shepard filter)*, where we utilize the particle deficiency to get non-zero gradients exactly where the surface lies.

4.3.1. Micropolar Fluids

> 💡 ***Summary***
>
> - For modeling turbulent details, we assume a micropolar fluid model, where particles are *not rotationally invariant* anymore.
> - We extend the Navier-Stokes equation for the velocity update by a *viscosity update for friction transfer between linear and rotational motion*. We additionally solve the *law for conservation of angular momentum* for the *new* angular velocity, which is similar to the Navier-Stokes equation but without the pressure term.

SPH fluid simulations suffer from numerical diffusion *(of momentum)* which leads to a lower **vorticity** *(Wirbelkraft)*, a loss in turbulent details and finally in less realistic results. For the simulation of realistic turbulent fluids a **micropolar fluid model** is introduced which has a special handling of vorticity.

Micropolar Fluids

Micropolar fluids have a *microstructure*, i.e. they consist of rigid, spherical microelements. This results in a *non-symmetric* stress tensor, such that the governing equations also need to consider the *rotational motion* of the fluid particles.

→ Hence, each particle stores its *angular velocity*, which allows a realistic simulation of turbulent flows.

Momentum Equation

In micropolar fluid, we differentiate between the *law for conservation of linear momentum* (left, just as previously) and the **law for conservation of angular momentum** (right).

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \mathbf{P} + \mathbf{f}$$

with:
- density ρ
- particle velocity $\mathbf{v}$
- stress tensor $\mathbf{P}$
- body force per unit mass $\mathbf{f}$

$$\rho\Theta \frac{D\boldsymbol{\omega}}{Dt} = \nabla \cdot \mathbf{C} + \mathbf{P}_\times + \boldsymbol{\tau}$$

with:
- microinertia coefficient Θ
- angular velocity $\boldsymbol{\omega}$
- couple stress tensor $\mathbf{C}$
- external torque $\boldsymbol{\tau}$
- and $\mathbf{P}_\times = \begin{pmatrix} P_{23} - P_{32} \\ P_{31} - P_{13} \\ P_{12} - P_{21} \end{pmatrix}$

Constitutive Model

The stress tensor $\mathbf{P}$ of a micropolar fluid considers the friction between two rotating particles, and we additionally have a couple stress tensor $\mathbf{C}$:

$$\mathbf{P} = -p\mathbf{I} + \mu(\nabla \mathbf{v})^T - \mu_t \nabla \mathbf{v} + (\mu + \mu_t)\boldsymbol{\omega}^\times$$
$$\mathbf{C} = c\nabla(\boldsymbol{\omega})^T$$

with:
- pressure p
- particle velocity $\mathbf{v}$
- dynamic viscosity μ
- transfer coefficient μ_t
- cross product matrix of angular velocity $\boldsymbol{\omega}^\times$

with:
- dynamic rotational viscosity c

Equations of Motion

Substituting the stress tensors in the momentum equations from above:

$$\frac{D\mathbf{v}}{Dt} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{v} + (\nu+\nu_t)\nabla\times\boldsymbol{\omega} + \frac{\mathbf{f}}{\rho}$$
$$\Theta\frac{D\boldsymbol{\omega}}{Dt} = \zeta\nabla^2\boldsymbol{\omega} + (\nu+\nu_t)(\nabla\times\mathbf{v} - 2\boldsymbol{\omega}) + \frac{\boldsymbol{\tau}}{\rho}$$

- Particle acceleration is determined by:
  - Acceleration due to pressure differences in the fluid
  - Viscosity terms for linear friction and angular friction
  - Viscosity term for frictional transfer from rotational into linear motions and vice versa
  - External forces and torques (e.g. gravitation) per unit volume

where $\nu = \frac{\mu}{\rho}$, $\nu_t = \frac{\mu_t}{\rho}$, $\zeta = \frac{c}{\rho}$ are the kinematic viscosity, transfer coefficient and rotational viscosity.

Implementation

The implementation is very easy and only needs a few lines of code with a computational overhead of only ~5%.

In comparison to a simulation with the Navier-Stokes equations, each particle must store an additional vector for the angular velocity. The Navier-Stokes equations are extended by the term $(\nu+\nu_t)\nabla\times\boldsymbol{\omega}$, and one additional equation has to the (explicitly) solved (*lower equation of motion*).

Extras

With the following solvers we want to solve a *linear system of equations*

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

where we usually start with some initial guess for unknown $\mathbf{x}$ or $\mathbf{0}$.

Jacobi Method

Assume a *strictly diagonal dominant* matrix $\mathbf{A}$, meaning

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

The main idea of the *Jacobi Method* is to take each row of matrix $\mathbf{A}$ and assume every other variable in the equation to be fixed except a_{ii}. Then, we update the solution with the following formula:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

The computation of
$x_i^{(k+1)}$ requires each element in $\mathbf{x}^{(k)}$ except itself. Unlike the Gauss–Seidel method, we can't overwrite $x_i^{(k)}$ with $x_i^{(k+1)}$, as that value will be needed by the rest of the computation. The minimum amount of storage is two vectors of size n.

Conjugate Gradient Method

The main idea of the *conjugate gradient method* to view solution of the linear system of equations as an *optimization problem* (similarly to gradient descent). Unlike gradient descent, conjugate gradient takes as most n steps to converge, where n is the size of the matrix of the system. It does so by directly optimizing in each direction in the optimization landscape.

Gauss-Seidel Method

Assume that $\mathbf{A}$ is decomposed into a lower triangular component (+ diagonal) $\mathbf{L}$ and a strictly upper triangular component $\mathbf{U}$, s.t. $\mathbf{A} = \mathbf{L} + \mathbf{U}$. Then,

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ (\mathbf{L} + \mathbf{U})\mathbf{x} &= \mathbf{b} \\ \mathbf{L}\mathbf{x} &= \mathbf{b} - \mathbf{U}\mathbf{x} \end{aligned}$$

resulting in the update rule

$$\mathbf{x}^{(k+1)} = \mathbf{L}^{-1}\left(\mathbf{b} - \mathbf{U}\mathbf{x}^{(k)}\right)$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

The element-wise formula for the Gauss–Seidel method is related to that of the (iterative) Jacobi method, with an important difference: In Gauss-Seidel, the computation of $\mathbf{x}^{(k+1)}$ uses the elements of $\mathbf{x}^{(k+1)}$ that have already been computed, and only the elements of $\mathbf{x}^{(k)}$ that have not been computed in the $(k+1)$-th iteration. This means that, unlike the Jacobi method, only one storage vector is required as elements can be overwritten as they are computed, which can be advantageous for very large problems.

However, unlike the Jacobi method, the computations for each element are generally much harder to implement in parallel.