Physically Based Simulation
Summary HS 2012

Pascal Spörri
pascal@spoerri.io

February 21, 2013

Contents

Part I
Mass-Spring Systems

Steps towards a simulation of a Mass-Spring system:

- *Spatial discretization*: Sample object with mass points
- *Forces*: Define internal (springs!) and external forces
- *Dynamics*: Set up equations of motion
- *Temporal discretization*: Solve equations of motion

1 Spatial Discretization

Sample object with mass points:

$$\text{Total mass of object: } M,$$
$$\text{Number of mass points: } n,$$
$$\text{Mass of each point: } m = \frac{M}{n},$$

with each point holding the properties:

$$\text{Mass: } m_i$$
$$\text{Position: } x_i(t)$$
$$\text{Velocity: } v_i(t)$$

2 Forces

What are the forces that act on particle i?

External Forces Gravity: F_i

Internal Forces

- Elastic spring forces
- Viscous damping forces

Total force $F_i = F_i^{int} + F_i^{ext}$

2.1 Internal Forces

2.1.1 Elastic Springs

Definition (Elasticity). *Ability of a spring to return to its initial length when the deforming force is removed.*

$$F = -k(l - L) \qquad \text{Hooke's Law } (1D)$$
$$F_i = -k(||\vec{x}_i - \vec{x}_j|| - L)\frac{\vec{x}_i - \vec{x}_j}{||\vec{x}_i - \vec{x}_j||} \qquad (3D)$$

L : Initial spring length
l : Current spring length
k : Spring stiffness

For purely elastic springs *the force depends only on the position* and *no energy is lost during the deformation*.

$$W = \int_L^l k(x - L)dx \qquad \text{Work done by forces}$$
$$E = W = \frac{1}{2}k(l - L)^2 \qquad \text{Elastic spring energy}$$
$$\vec{F}_i = -\frac{E}{\delta \vec{x}_i} \qquad \text{Force}$$

Forces at Mass Point internal forces F_0^{int}:

$$\vec{F}_0^{int} = -\sum_{i|i\in\{1,2,3\}} k_i(l_i - L_i)\frac{\vec{x}_i - \vec{x}_0}{l_i}$$

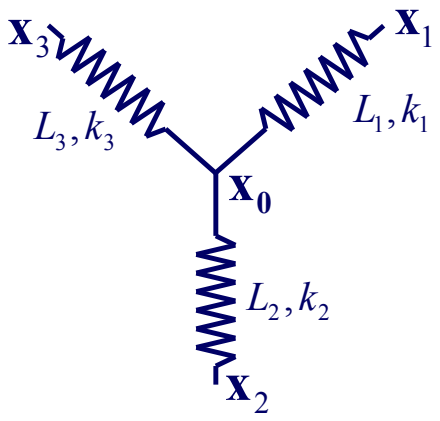


Figure 1: Example

2.2 Dissipative Forces

Real-world mechanical systems dissipate energy over time: *Internal friction* $\Longrightarrow$ *Thermal energy* (irreversible process). Controllable dissipation is useful for physics simulations.

Dissipation for mass-spring systems γ is the damping coefficient

$$\vec{F}^{pd}(t) = -\gamma \cdot \vec{v}(t) \qquad \text{Point damping}$$

Point damping is simple and efficient, but it *damps **all** motion* (i.e. including translations and rotations).

2.3 External Forces

External forces are forces like

- Gravity
- Contact forces
- All forces that are not cause by springs

3 Dynamics

In the dynamics step the equations of motion are setup. The force is known for every particle.

Kinematic relations

$$\textbf{Velocity: } \vec{v}_i(t) = \frac{d\vec{x}_i(t)}{dt}$$

$$\textbf{Acceleration: } \vec{a}_i(t) = \frac{d\vec{v}_i(t)}{dt} = \frac{d^2\vec{x}_i(t)}{dt^2}$$

With **Newton's 2nd Law**

$$\vec{F}_i = m_i \cdot \vec{a}_i$$

we can describe the equations of motion using Newton's second law:

$$\vec{F}_i = m_i \frac{d^2\vec{x}_i(t)}{dt^2} = \vec{F}_i^{int}(t) + \vec{F}_i^{ext}(t)$$

which can be abstracted as

$$\vec{F} = M\frac{d^2\vec{x}(t)}{dt^2} = \vec{F}^{int}(t) + \vec{F}^{ext}(t) \qquad M \in \mathbb{R}^{3n\times 3n}$$

adding damping

$$M\frac{d^2\vec{x}(t)}{dt^2} + D\frac{d\vec{x}(t)}{dt} = \vec{F}^{int}(t) + \vec{F}^{ext}(t)$$

4 Temporal discretization

A *differential equation* describes an unknown function through its derivatives. An ordinary differential equation (ODE) contains only derivatives with respect to a single variable.

Example	Abstract form
$x''(t) = \frac{\vec{F}(t)-\gamma\vec{x}'(t)}{m_i}$	$y''(t) = f(t, y, y')$

The order of the ODE is expressed as function of the highest derivative:

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

Solving an ODE: Given f, determine y. Example:

$$y'(t) = f(t, y) = y(t) \quad \Longrightarrow \quad y(t) = Ce^t$$

The integration constant C is unknown and i determined by the initial value:

$$y(0) = 2 \qquad \Longrightarrow \qquad C = 2$$

ODE + initial value = initial value problem (IVP)

Theorem. *Picard–Lindelöf An IVP has a unique solution if f is Lipschitz continuous.*

4.1 Application to Mass-Spring systems

We have an initial position $\vec{x}_i(t_0)$, an initial velocity $\vec{v}_i(t_0)$ and a governing ODE:

$$\frac{d^2\vec{x}_i(t)}{dt^2} = \frac{\vec{F}_i(t) - \gamma\vec{v}_i(t)}{m_i}.$$

We want the position $\vec{x}_i$ over time.
The first step to rewrite the problem since it's easier to deal with as a first oder ODE. We therefore reduce the 2^{nd} order ODE to two coupled 1^{st} order ODEs.

$$m_i\frac{d^2\vec{x}_i(t)}{dt^2} + \gamma\frac{d\vec{x}_i(t)}{dt} = \vec{F}_i(t)$$

$$\begin{cases} \frac{d\vec{x}_i(t)}{dt} = \vec{v}_i(t) & \text{velocity} \\ \frac{d\vec{v}_i(t)}{dt} = \frac{\vec{F}_i(t)-\gamma\vec{v}_i(t)}{m_i} & \text{acceleration} \end{cases}$$

Write as one system of 1^{st} order ODEs:

$$\vec{y}_i(t) = \begin{pmatrix} \vec{x}_i(t) \\ \vec{v}_i(t) \end{pmatrix}$$

$$\vec{y}_i'(t) = \begin{pmatrix} \vec{v}_i(t) \\ \frac{\vec{F}_i(t)-\gamma\vec{v}_i(t)}{m_i} \end{pmatrix}$$

4.2 Numerical Solution

Compute approximations y_i to true solution for *discrete* time instants t_i. Compute solution at t_{i+1} based on previous solutions at t_{i-1}, t_{i-2}.

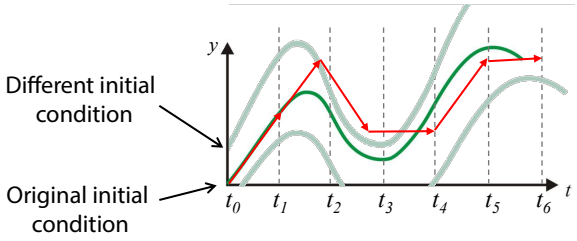


Figure 2: Solutions based on different initial conditions

5 Integration schemes

Explicit Methods

- Explicit Euler
- Heun, Midpoint
- Runge-Kutta methods

Tend to explode with *stiff problems*:

- Explicit methods require very small time steps for stable integration
- Are inefficient since step size is determined by stability not accuracy requirements.

Use implicit methods for stiff problems.

Implicit Methods

- Backward Euler
- Implicit Euler
- BDF methods

Methods for higher order ODEs

- Verlet
- Leapfrog
- Newmark methods

5.1 Computing Approximations

An approximation of a function $y(t)$ can be computed by computing the taylor expansion:

$$y(t+h) = y(t) + \frac{y'(t)}{1!}h + \frac{y''(t)}{2!}h^2 + \dots,$$

which allows us to compute a first order approximation:

$$y(t+h) = y(t) + hy'(t).$$

Which leaves us with *Euler' Method*.

5.1.1 Accuracy criteria

to evaluate an integration scheme:

Convergence Do approximations converge to true solution, i.e.

$$h \mapsto 0 \implies y_i \mapsto y_i(t)$$

Accuracy how fast does the error decrease as $h \mapsto 0$
Local error is $\mathcal{O}\left(h^{p+1}\right) \implies$ Method is accurate of order $\mathbf{p}$.

Stability Is the solution always bounded, i.e. $|y_n| < \infty$

Efficiency Is a given method a good choice for a given problem?

5.2 Integration Schemes

5.2.1 Euler's Method

Start at the initial condition and take a step into the direction of the tangent.

$$y_{n+1} = y_n + hf(t_n, y_n).$$

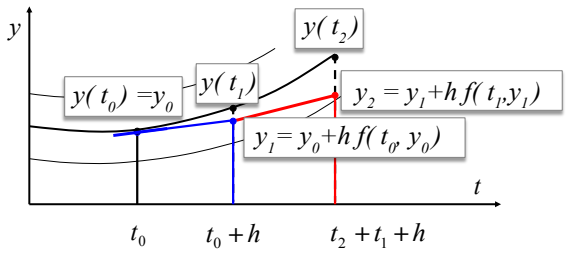


Figure 3: Visualization of the Euler Method

Explicit Euler has an accuracy of order 1: $\mathcal{O}\left(h^2\right)$ error per stop.

5.2.2 Heun's Method

Heun's method is a two step method and provides a **2^{nd} order accuracy**. Idea:

$$y(t+h) \approx y(t) + \frac{h}{2}[y'(t) + y'(t+h)],$$

$y'(t+h)$ is unknown. Use an Euler step to compute it.

$$\tilde{y}_{n+1} = y_n + h \cdot f(t_n, y_n)$$
$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, \tilde{y}_{n+1})\right]$$

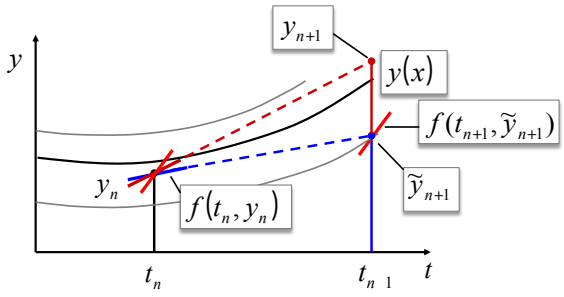


Figure 4: Heun's Method graphically

5.2.3 Explicit Midpoint Method

Heun's method uses $f(t)$ and $f(t+h)$ to achieve 2^{nd} order accuracy. Use $f\left(t + \frac{h}{2}\right)$ instead. This also gives us 2^{nd} order accuracy.

$$\tilde{y}_{n+1} = y_n + \frac{h}{2} \cdot f(t_n, y_n)$$
$$y_{n+1} = y_n + h\left[f(t_n, y_n) + f(t_{n+1}, \tilde{y}_{n+1})\right]$$

This solution provides us also with provides a **2^{nd} order accuracy**.

5.2.4 4^{th}-Order Runge-Kutta

RK4 is one of the most widely used integrators. Four slope evaluations gives us **4^{th} order accuracy**

$$k_1 = f\left(t_n, y_n\right)$$
$$k_2 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right)$$
$$k_3 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right)$$
$$k_4 = f\left(t_n + h, y_n + hk_3\right)$$

y_{n+1} is computed using a weighted average slope:

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$$

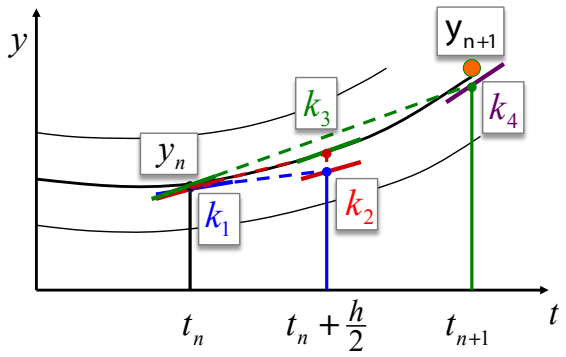


Figure 5: RK4 graphically

5.2.5 Implicit Euler Method

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

Implicit methods have to be solved iteratively. Example with Newton's method:

1. **Rewrite equation**:

$$0 = y_{n+1} - y_n - hf(t_{n+1}, y_{n+1}) = g(y_{n+1})$$

2. **Make initial guess**:

$$\tilde{y}_{n+1} = y_n$$

3. **Taylor expansion**:

$$g(\tilde{y}_{n+1} + \Delta y) = g(\tilde{y}_{n+1}) + g'(\tilde{y}_{n+1}) + \mathcal{O}\left(\Delta y^2\right) = 0$$

4. **Linearize**

$$g'(\tilde{y}_{n+1} \cdot \Delta y = -g(\tilde{y}_{n+1})$$

In order to solve the equation one has to iterate over

$$\begin{aligned} \Delta y &= -\frac{g(\tilde{y}_{n+1})}{g'(\tilde{y}_{n+1})} && \text{Solve,} \\ \tilde{y}_{n+1} &= \tilde{y}_{n+1} + \Delta y && \text{Update,} \\ r &= g(\tilde{y}_{n+1}) && \text{Error,} \end{aligned}$$

until the *Error* is small enough.

Newton's method is powerful, but it has to be used with care:

- Requires the computation of the solution of a linear system in each iteration.
- A robust implementation needs advanced strategies.

5.2.6 Semi-Implicit Euler Method

The *semi-implicit* Euler method can be applied to a pair of *differential equations* of the form:

$$\begin{aligned} \frac{dx}{dt} &= f(t, v) \\ \frac{dv}{dt} &= g(t, x) \end{aligned}$$

The discrete solution can be computed by iterating over:

$$\begin{aligned} v_{n+1} &= v_n + g(t_n, x_n)\Delta t \\ x_{n+1} &= x_n + f(t_n, v_{n+1})\Delta t \end{aligned}$$

5.3 Higher-Order Numerical Integration

Let

$$M\ddot{x} = F^{int} + F^{ext}$$

5.3.1 Verlet Integration

Combine forward and backward expansions of $x(t)$. Which results in a **2nd order method**.

$$\begin{cases} x(t+h) &= x(t) + hx'(t) + \frac{h^2}{2}a(t) + \frac{h^6}{6}x''' \\ x(t+h) &= x(t) - hx'(t) + \frac{h^2}{2}a(t) / \frac{h^6}{6}x''' \end{cases}$$
$$x(t+h) = 2x(t) - x(t-h) + h^2 a(t)$$

- The two-step method makes it problematic for discontinuities.
- *Velocities* have to be *approximated a posteriori*.

5.3.2 Leapfrog

Compute velocities on a *staggered grid*. **2nd order method**.

$$\begin{aligned} v\left(t + \frac{h}{2}\right) &= v\left(t - \frac{h}{2}\right) + h \cdot a(t) \\ x(t+h) &= x(t) + h \cdot v\left(t + \frac{h}{2}\right) \end{aligned}$$

This method is only applicable if $a(t)$ does not depend on velocity: $\implies$ no damping!

5.3.3 Symplectic Euler

2nd order accurate

$$\begin{aligned} v(t+h) &= v(t) + h \cdot a(t) \\ x(t+h) &= x(t) + h \cdot v(t+h) \end{aligned}$$

Has a good stability for oscillatory motion and good conservation properties:

- Exactly conserves momentum
- Nearly conserves energy over long run times.

6 A Basic Mass-Spring System

6.1 Setting up Springs

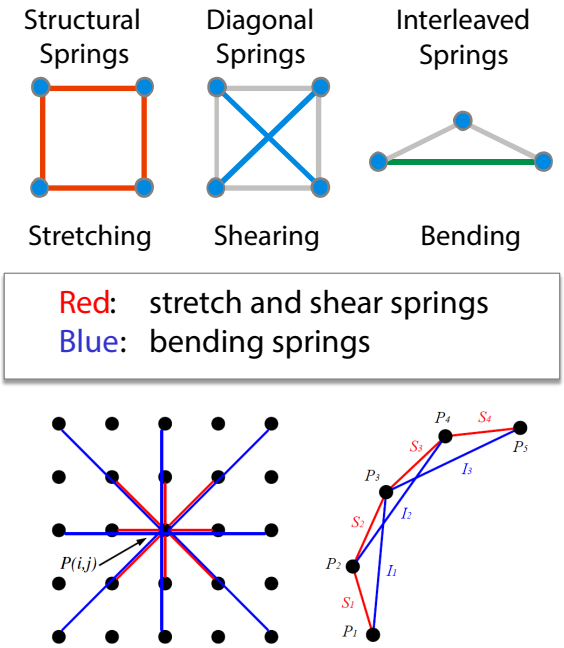
Is the model provided as tetrahedral mesh?

Yes: Model edges as springs

No: Find *stable* topology:

- No control over material properties
- No explicit volume preservation.

Springs for Cloth Types of require springs:



Springs couple different deformation modes:

- Bending and shear springs respond to stretch,
- Which lead to uncontrollable transverse contraction.

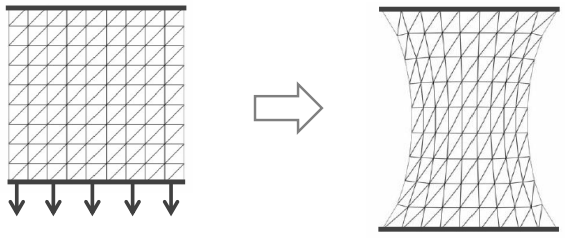


Figure 6: Transverse Contraction due to bending and shear springs

The material behaviour depends strongly on topology.

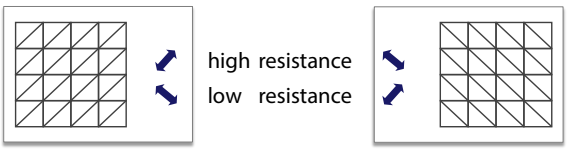


Figure 7: Material behaviour

Limitations

- Strong dependence on discretization (topology)
- Limited control over material behaviour
- No (explicit) volum preservation (for solids)

Verdict Mass-Spring Systems are simple and efficient, but not very accurate.

Alternatives

- Constraints
- Continuum-mechanics with Finite Elements

Part II
Constraints

So far, the motion of particle was defined by forces:

- Elastic springs,
- Damping,
- Gravity

motion is unrestricted using these techniques. Sometimes we want to restrict motion:

- Bead a wire (constant length),
- Incompressible deformations (constant volume).

Constraints can be use to strictly *enforce conditions* (unlike elastic forces) or to *derive general forces* (unlike springs).

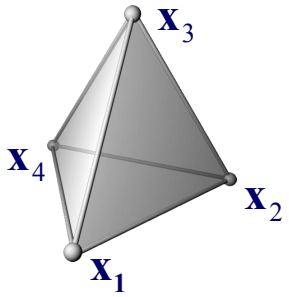
Definition. *Constraint A constraint $C(x_1, x_2, \ldots, x_n)$ is*

- *A scalar-valued function of one or several arguments*
- *An implicit expression for a relation that must hold between its arguments.*

***Convention**: A constraint is satisfied if $\mathbf{C} = \mathbf{0}$.*

Examples of n-ary constraints:

- Constant position $C_1(x_1)$
- Constant length $C_2(x_1, x_2)$
- Constant area $C_3(x_1, x_2, x_3)$
- Constant volume $C_4(x_1, x_2, x_3, x_4)$



7.0.1 Recipes for constraint forces

Area of a triangle (x_0, x_1, x_2):

$$C(x_0, x_1, x_2) = \frac{1}{2} ||(x_1 - x_0) \times (x_2 - x_0)|| - A$$

Volume V of a tetrahedron (x_0, x_1, x_2, x_3):

$$C(x_0, x_1, x_2, x_3) = \frac{1}{6}(x_1 - x_0) \cdot [(x_2 - x_0) \times (x_3 - x_0)] - V$$

7 Soft constraints and the Penalty Method

How can we enforce constraints?

Define potential energy for constraint:

$$E_c(x_1, x_2, \ldots, x_n) = \frac{1}{2} k C(x_1, x_2, \ldots, x_n)^2,$$

where k is a *stiffness* coefficient.

$$\begin{aligned} E_c &= 0 && \text{constraint is met,} \\ E_c &> 0 && \text{otherwise.} \end{aligned}$$

The *constraint force* is a *negative gradient* of potential energy:

$$F_j = -\frac{\delta E_c}{\delta x_j} = -kC(x_1, \ldots x_n)\frac{\delta C(x_1, \ldots, x_n)}{\delta x_j}$$

The *penalty force* pulls the system towards $\mathbf{C} = \mathbf{0}$.

Distance Preservation Preserve distance between two points:

$$\begin{aligned} C(x_0, x_1) &= ||x_1 - x_0|| - L \\ F_1^C(x_0, x_1) &= -kC(x_0, x_1)\frac{\delta C(x_0, x_1)}{\delta x_1} \\ &= \underbrace{-k(||x_1 - x_0|| - L)\frac{x_1 - x_0}{||x_1 - x_0||}}_{\textbf{Spring Force}} \end{aligned}$$

Simulation with Penalty Forces Simulate x_0 and x_1 using Newton's law. Enforce distance constraint with penalty forces:

$$F_i^C(x_0, x_1) = -kC(x_0, x_1)\frac{\delta C(x_0, x_1)}{\delta x_i}$$

Step forward in time (explicit Euler):

$$v_{n+1} = v_n + h\frac{1}{M}\left(F^C(x) + F^{ext}\right).$$

7.1 Summary Soft Constraints

Constraint forces are:

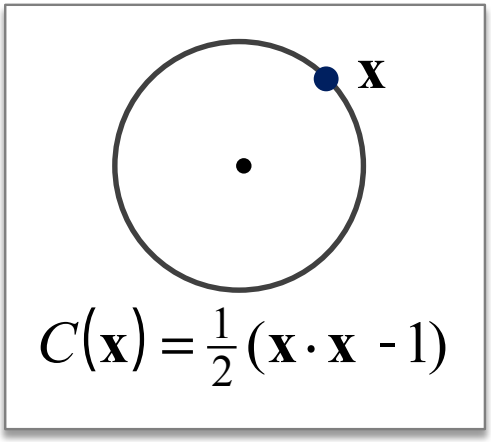
- A powerful mechanism to enforce various conditions,
- Generic: Express condition, forces from a standard scheme,
- n-ary forces as opposed to binary strings.

However, constraints can be:

- Computationally expensive (especially with implicit integration),
- Redundant or conflicting,
- Penalty forces *model **soft** constraints* and are *not sufficient* for strict enforcement (***hard** constraints*).

8 Hard Constraints

How can we model a hard constraint? Consider a 2D particle constrained to move on a unit circle.



First we define *legal* kinematics:

Position: $C(x) = 0$

Velocity: $\dot{C}(x) = 0$

Acceleration: $\ddot{C}(x) = 0$

Idea: Start with a legal position and velocity and ensure that acceleration remains legal (constraint forces).

$$\begin{cases} \underbrace{\ddot{C}(x) = \ddot{x} \cdot x + \dot{x} \cdot \dot{x} = 0}_{\text{Legal acceleration}} \\ \underbrace{\ddot{x} = \dfrac{F + F^C}{m}}_{\text{Newton's Law}} \end{cases}$$

combine:

$$F^C \cdot x = -F \cdot x - m\dot{x} \cdot \dot{x}$$

Require constraint force to act only in gradient direction:

$$F^C = \lambda \frac{\delta C}{\delta x} = \lambda x.$$

Insert into combined formula:

$$\lambda = -\frac{F \cdot x + m\dot{x} \cdot \dot{x}}{x \cdot x},$$

which gives us an *hard constraint force*.

Simulation with Hard Constraints Use hard constraints with an explicit function (since implicit integration is (much) more complicated):

1. Solve (LES) for constraint force magnitudes λ_n.
2. Compute constraint forces:
$$F_n^C = \frac{\delta C(x_n)^t}{\delta x} \lambda_n$$
3. Step forward in time
$$v_{n+1} = v_n + \frac{h}{M}\left(F_n^C + F^{ext}\right)$$

8.1 Summary Hard Constraints

Hard constraint forces

- add just enough force to maintain constraint (exactly),
- require no high stiffness (numerically pleasing).

However

- *Constraints drift* (error in ODE solve), needs correction
- General formulation is more involved (many constraints)
- Requires the solution of systems of equations

Part III
Applied Partial Differential Equations

An Ordinary Differential Equation (ODE) describes an *unknown* function through its derivatives with respect to ***single*** variable:

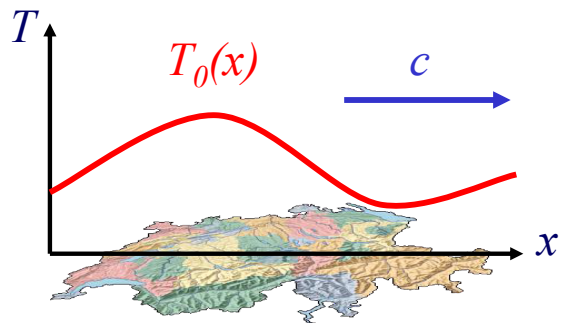
$$m\frac{d^2x(t)}{dt^2} = F(x(t)).$$

A Partial Differential Equation (PDE) describes an unknown function through its partial derivatives with respect to ***multiple*** variables:

$$\frac{\delta u(t,x)}{\delta t^2} = c^2\frac{\delta u(t,x)}{\delta 4x^2}.$$

9 Example:

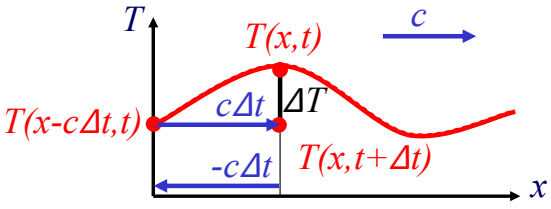
1D Advection. Weather forecast: Simulate temperature evolution.



Given the temperature distribution $T_0(x)$ at time $t = 0$ and wind speed c. Find an expression for the temperature evolution $T(x,t)$!

9.1 Analytical Solution

How does the temperature change over a time interval Δt?



$$T(x,t+\Delta t) = T(x-c\Delta t,t)$$
$$\Delta T = T(x,t+\Delta t) - T(x,t)$$
$$T(x-c\Delta t,t) = T(x,t) - \frac{\partial T}{\partial x}c\Delta t + \mathcal{O}\left(\Delta t^2\right) = T(x,t+\Delta t)$$

From which we can build the advection equation:

$$\frac{\Delta T}{\Delta}t \approx -c\frac{\partial T}{\partial x} \quad \lim_{\Delta\mapsto 0} \implies \quad \frac{\partial T}{\partial t} = -c\frac{\partial T}{\partial x}$$

Any $T(x,t)$ of the form $T(x,t) = f(x-ct)$ solves $\frac{\partial T}{\partial t} = -c\frac{\partial T}{\partial x}$. The solution also needs to satisfy the initial condition:

$$T(x,0) = T_0(x)$$

The solution is thus

$$T(x,t) = T_0(x-ct)$$

9.2 Numerical Solution

Sample temperature $T(x,t)$ on 1D grid $T^t[i] = T(i\cdot h, t\cdot\Delta t)$ with $i \in (1,\dots n)$, $t \in (0,1,2,\dots)$:

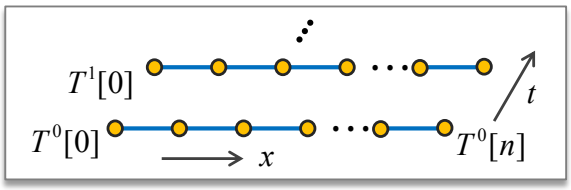


Figure 8: Sampling Grid

10 Overview

Order of PDE The *order* of a PDE is the order of it's highest partial derivative.

Linearity A PDE is *linear* if the unknown function (u) and it's partial derivatives only occure linearly:

Linear example: $u_t + c\cdot u_x = 0$ (Advection eq.)
Nonlinear example: $u_t + u\cdot u = 0$ (Burger's eq.)

Coefficients of linear PDEs can be nonlinear functions:

$$y^2\cdot u_{yy} + x^2\cdot u_{yy} = 0$$

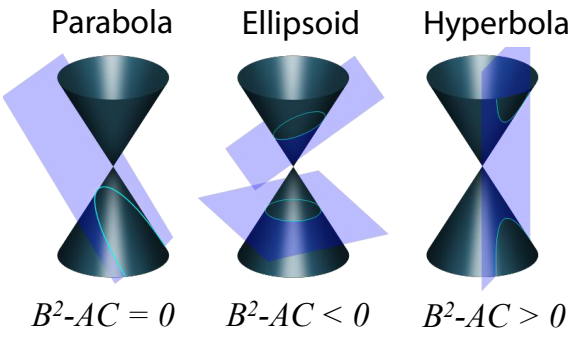
10.1 PDE Classification

2^{nd} order linear PDEs are of highest practical relevance. A 2^{nd} order linear PDE has the form:

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} = F(x,y,u,u_x,u_y).$$

A 2^{nd} order linear PDE is either:

Hyperbolic	$B^2 - AC > 0$	Wave equation
Parabolic	$B^2 - AC = 0$	Heat equation
Elliptic	$B^2 - AC < 0$	Laplace equation



10.2 Hyperbolic PDEs

Hyperbolic PDEs are typically time dependent problems. They retain & propagate disturbances present in initial data.

Prototype: let u be the amplitude and c the propagation speed:

$$\textit{Wave Equation}: \quad \frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} c^2$$

Applications

- Simulate wave propagation for sound, light and water.
- Mechanics: Oscillatory motion, vibrating strings.

10.3 Parabolic PDEs

Parabolic PDEs are typically time dependent problems. Solutions *smooth out* as time increases.

Prototype: let u be the temperature and α the thermal diffusivity:

$$\textit{Heat Equation}: \quad \frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

Application consist of conduction and general diffusion processes.

10.4 Elliptic PDEs

Elliptic PDEs describe static problems (i.e. systems in equilibrium). Solutions are smooth (if the coefficients are).

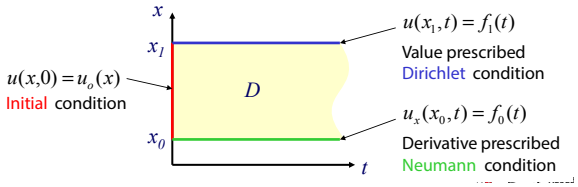
Prototype:

$$\textit{Laplace Equation}: \quad \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Applications consist of steady-state solutions to hyperbolic and parabolic PDEs and of equilibrium problems.

11 Boundary Conditions

Generally there are (infinitely) many u which solve a PDE. So there's additional information required → *Boundary Conditions*. Typically u is defined in a region D and the solution is required to satisfy certain conditions on the boundary δD of D:



12 Solving Poisson's Equation Numerically

12.1 Poisson's Equation

$$-\Delta u(x,y) = f(x,y)$$

The poisson equation is the same as Laplace Equation if $f(x,y) = 0$. The $2D$ Laplace operator:

$$\Delta u(x,y) := \frac{\partial^2}{\partial x^2} u(x,y) + \frac{\partial^2}{\partial y^2} u(x,y)$$

Poisson's equation has practical relevance:

- *Thin membrane* clamped at boundary (drum):
  $f(x,y)$ applies forces, $u(x,y)$ vertical displacement.
- *Electrical potential* due to charge distribution:
  $f(x,y)$ charge distribution, $u(x,y)$ electric potential field.
- Steady state *heat distribution*:
  $f(x,y)$ heat sources, $u(x,y)$ heat distribution.

12.2 Finite Difference Solution

Consider the 1D problem:

$$\begin{aligned} -u_{xx}(x) &= f(x) \qquad x \in \Omega = (0,1) \\ u(0) &= 0 \\ u(1) &= 0 \end{aligned}$$

Use a regular grid to discretize Ω:

$$u[i] = u(i \cdot h) \qquad i \in (0, \ldots, n).$$

Approximate derivatives with finite differences:

$$u_{xx} = \frac{u_x[i] - u_x[i-1]}{h} = \frac{u[i+1] - 2u[i] + u[i-1]}{h}.$$

Which gives us an equation for every grid point $i = 2, \ldots, n-1$:

$$h^2 f[i] = u[i+1] - 2u[i] + u[i+1]$$

2D Finite Differences stencil

$$\Delta_h u[i,j] = \frac{u[i+1,j] + u[i-1,j] + u[i,j-1] + u[i,j+1] - 4u[i,j]}{h^2}$$

Problems and Drawbacks Finite differences

- Are easy to understand and implement
- require a regular grid
- high smoothness requirements on solution
- higher order derivatives require large stencils

12.3 Finite Elements Solution

Consider the 1D problem:

$$\begin{aligned} -u_{xx}(x) &= f(x) \qquad x \in \Omega = (0,1) \\ u(0) &= 0 \\ u(1) &= 0 \end{aligned}$$

Assum the PDE is satisfied point wise then

$$-\int_\Omega u_{xx}(x)dx = \int_\Omega f(x)dx,$$

and also

$$-\int_\Omega u_{xx}(x)\cdot v(x)dx = \int_\Omega f(x)\cdot v(x)dx$$

where $v(x)$ is a testing function. For arbitrary smooth functions $v : \Omega \to \mathbb{R}$.

Weak Form can be computed by using integration by parts:

$$\int_a^b f'\cdot g = [f\cdot g]_a^b - \int_a^b f\cdot g'.$$

$$\begin{aligned} -\int_\Omega u_{xx}\cdot v dx &= \int_\Omega f\cdot v dx, \\ \int_\Omega u_x\cdot v_x dx &= \int_\Omega f\cdot v + [u_x\cdot v]_0^1, \\ &\text{with } v(0)=v(1)=0, \end{aligned}$$

we get the weak form:

$$\int_\Omega u_x\cdot v_x dx = \int_\Omega f\cdot v dx.$$

The *order of the highest derivatives has decreased by one*. This function is called the *weak form* since it imposes *weaker* continuity requirements on the solution. The *weak and strong forms are equivalent*.

12.3.1 FEM-Template Process

1. Weak formulation
2. Ritz-Galerkin approach
3. Choice of Function Space
4. Geometric Decomposition of Domain
5. Setup Matrix Equations
6. Solve Linear System

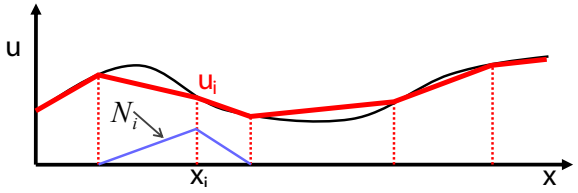
12.3.2 Ritz-Galkerin Approach

So far, $u(x)$ and $f(x)$ were continuous functions. We choose a finite-dimensional subspace for $u(x)$ and $f(x)$ and solve the projected problem (i.e. the weak form).
We discretize u and v on n-dimensional space:

- Sample with nodal positions x_i
- Nodal coefficients u_i
- Basis functions N_i

and get:

$$u(x) = \sum_i u_i N_i(x)$$



We observe that:

$$u(x) = \sum_i u_i N_i(x) \implies \frac{\partial}{\partial x}u(x) = \sum_{i=1}^n u_i \frac{\partial}{\partial x}N_i(x).$$

Insert everything into the weak formulation:

$$\int_\Omega u_x v_x\, dx = \int_\Omega fv\, dx$$

$$\int_\Omega \sum_{i=1}^n u_i \frac{\partial N_i}{\partial x}\cdot \sum_{j=1}^n v_j \frac{\partial N_j}{\partial x}dx - \int_\Omega f\cdot \sum_{j=1}^n v_j N_j(x)\, dx = 0$$

Transform it:

$$\int_\Omega \sum_{i=1}^n u_i \frac{\partial N_i}{\partial x}\cdot \sum_{j=1}^n v_j \frac{\partial N_j}{\partial x}dx - \int_\Omega f\cdot \sum_{j=1}^n v_j N_j(x)\, dx = 0,$$

$$\sum_{j=1}^n v_j \left[\sum_{i=1}^n u_i \int_\Omega \frac{\partial N_i}{\partial x}\cdot\frac{\partial N_j}{\partial x}dx - \int_\Omega f\cdot N_j(x)\, dx\right] = 0.$$

And we get n linear equations for n unknowns:

$$\sum_{i=1}^n u_i \int_\Omega \frac{\partial N_i}{\partial x}\cdot\frac{\partial N_j}{\partial x}dx - \int_\Omega f\cdot N_j(x)\, dx = 0 \quad \forall i = 1,\dots,n$$

Which can be written as a linear system $\mathbf{Ku} = \mathbf{f}$:

$$\underbrace{\begin{pmatrix} K_{11} & \cdots & K_{1n} \\ \vdots & \ddots & \vdots \\ K_{n1} & \cdots & K_{nn} \end{pmatrix}}_{\mathbf{K}} \cdot \underbrace{\begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix}}_{\mathbf{u}} = \underbrace{\begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}}_{\mathbf{f}},$$

where

$$K_{ij} = \int_\Omega \frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x}dx,$$
$$f_i = \int_\Omega f\, N_i(x)dx.$$

This matrix is

- Symmetric
- Positive definite (elliptic PDE)
- Sparse (if N_i has compact support)

12.3.3 Choice of the Function Space

Problem: What basis functions should be used in $u(x) = \sum_i u_i N_i(x)$?
Use piecewise linear basis functions. Basis functions are uniquely defined through *Geometry* and interpolation requirement $N_i(x_j) = \delta_{ij}$.

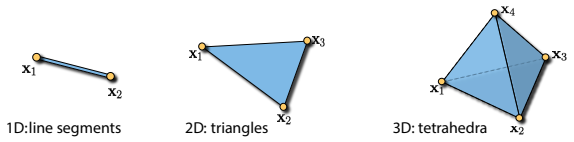
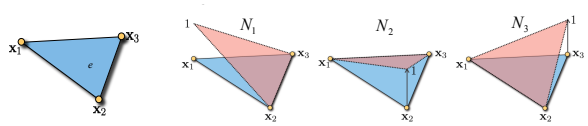


Figure 9: Simplical Elements

Triangle basis



Basis functions:

$$N_i(x, y) = a_i x + b_i y + c.$$

Due to $N_i(x_j) = \delta_{ij}$, we have:

$$\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_i \\ b_i \\ c_i \end{pmatrix} = \begin{pmatrix} \delta_{1i} \\ \delta_{2i} \\ \delta_{3i} \end{pmatrix}.$$

12.4 Finite Elements

A finite element is a triple consisting of

- A closed subset $\Omega_e \subset \mathbb{R}^d$ (in d dimensions)
- A set of n basis functions: $N_i : \Omega_e \to \mathbb{R}$
- A set of n associated nodal variables x_i.

In order to solve a PDE we need to know on how to tessellate a domain into elements:

- 2D: Delaunay triangulator
- 3D: Tetrahedral mesher

12.4.1 Integrating Basis Functions

Goal: Evaluate $\mathbf{f}$

$$f_i = \int_\Omega f N_i dxdy$$

N_i extends over all $n_{e,i}$ elements incident to x_i. N_i is zero outside of $\Omega_i \subset \Omega$. Evaluate integral over Ω_e with *quadrature rule*:

$$\int_{\Omega^e} f\, N_i^e\, dxdy \approx f(x_q, y_q) \cdot N_i^e(x_q, y_q) \cdot A_e$$

$$A_e : \text{ Area of element } e$$
$$(x_q, y_q) : \text{ single quadrature point at bary-center}$$

Goal: Evaluate $\mathbf{K}$

$$K_{ij} = \int_\Omega \frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + \frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} dxdy$$

Observe that $K_{ij} \neq 0 \leftrightarrow \exists \Omega^e$ such that $\int_\Omega^e N_i^e N_j^e dxdy > 0$. Let S_{ij} denote set of all these elements, then:

$$K_{ij} = \sum_{e \in S_{ij}} \int_\Omega \frac{\partial N_i^e}{\partial x}\frac{\partial N_j^e}{\partial x} + \frac{\partial N_i^e}{\partial y}\frac{\partial N_j^e}{\partial y} dxdy$$

This allows for a element-centered implementation: For each element e:

- Compute basis function derivatives:

$$\frac{\partial N_l^e}{\partial x} \qquad l = 1, \ldots, 3$$

- Form products and integrate:

$$\frac{\partial N_l^e}{\partial x}\frac{\partial N_k^e}{\partial x} \qquad l = 1, \ldots, 3\ k = 1, \ldots, 3$$

- Add to global matrix:

$$K_{ij} += A_e \left(\frac{\partial N_i^e}{\partial x}\frac{\partial N_j^e}{\partial x} + \frac{\partial N_i^e}{\partial y}\frac{\partial N_j^e}{\partial y} \right)$$

12.4.2 Advantages of Finite Elements

FEM $\{\ldots\}$

- can easily deal with complex geometries,
- solution in nodes is naturally interpolated inside elements using basis functions
- weaker smoothness requirements on solution
- weak form is natural for many applications
- $\ldots$

Part IV
Rigid Body Simulation

13 Representation of a Rigid Body

Spatial Variables

- *Location of a Particle*: Translation from origin
- *Location of a Rigid Body*: Translation and rotation

Center of Mass representewd by n particles with mass m_i and position r_i. The center of mass x can be computed by:

$$x(t) = \frac{\sum m_i r_i(t)}{\sum m_i} = \frac{\sum m_i r_i(t)}{M}.$$

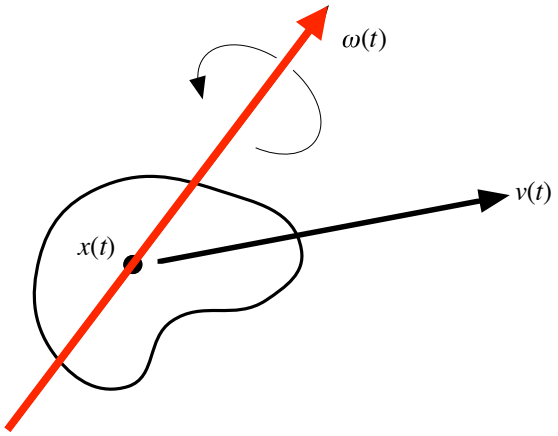
Under translation and rotation:

$$x = \frac{\int x\rho(x)dV}{\int \rho(x)dV}$$

14 Rigid Body Kinematics

Spin The body can spin around an axis:

- *Angular velocity* $\omega(t)$
- *Spin axis* direction of $\omega(t)$



Where $v(t) = \frac{d}{dt}x(t)$.

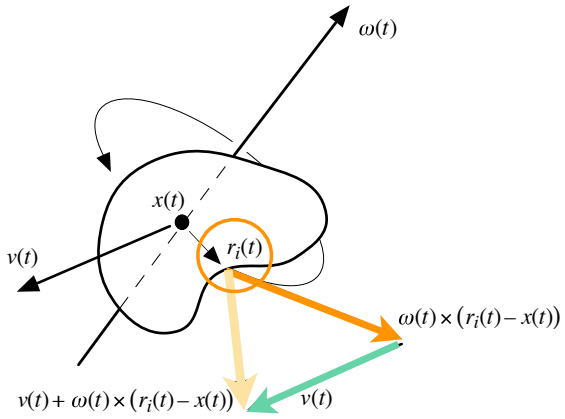
14.1 Total Velocity

The (constant) location of a particle i in body space is r_{0_i}. The location of particle i:

$$r_i(t) = R(t)r_{0_i}$$

The velocity of said particle i consists of

$$\begin{aligned}\dot{r}_i &= w(t)\dot{R}(t)r_{0_i} + v(t) \\ &= \underbrace{w(t) \times (r_i(t) - x(t))}_{\text{angular component}} + \underbrace{v(t)}_{\text{linear component}}\end{aligned}$$



15 Rigid Body Dynamics

- External forces (gravity, wind, ...) chaning:
  - Linear velocity
  - Angular velocity
- Total force acting on a body

$$F(t) = \sum F_i(t) = \sum m_i \dot{v}_i(t) = M\dot{v}_{cm}$$

- Linear velocity change

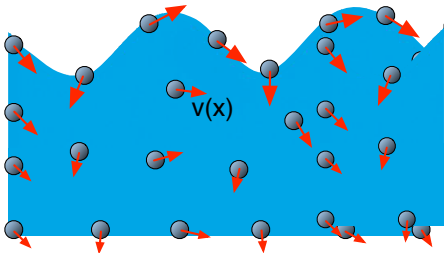
$$a_{cm} = \frac{F}{M}$$

Part V
Fluid Simulation

16 Spatial Discretization

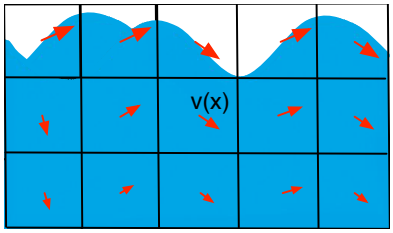
There are two common spatial discretizations:

- *Langragian* viewpoint
  - Particles can move freely in space and carry quantities,
  - Fluid motion is described by moving particles around.



Features
/ *Incompressibility*
+ *Complex interactions*
+ *Splashes, droplets*
+ *Advection*
- *Smooth Surfaces*

- *Eulerian* viewpoint
  - Fixed spatial locations
  - Measure quantities as it flows past



Features

+ *Incompressibility*
/ *Complex interactions*
- *Splashes, droplets*
- *Advection*
+ *Smooth Surfaces*

16.1 Notation

$$\text{Gradient: } \nabla u = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$$

$$\text{Divergence: } \nabla \cdot u = u_x + u_y + u_z$$

$$\Delta u = \nabla^2 u = \nabla \cdot \nabla u = u_{xx} + u_{yy} + u_{zz}$$

16.2 Navier-Stokes Equations

The *Navier-Stokes Equations* are 2 differential equations describing a velocity field over time.

Symbols:

$u =$ velocity
$p =$ pressure
$\rho =$ density
$g =$ gravity
$\nu =$ viscosity (kinematic)

Conservation of momentum

$$\underbrace{\frac{\partial u}{\partial t} + (u \cdot \nabla)u}_{\text{advection}} = \underbrace{-\frac{1}{\rho}\nabla p}_{\text{pressure}} + \underbrace{\nu \Delta u}_{\text{viscosity}} + g$$

Consider a single blob of fluid p. Which has mass m, volume V and velocity u. We start with Netwon's second law $F = ma$ and rewrite it as $F = m\frac{Du}{Dt}$.

Forces acting on the fluid

- *Gravity*: mg
- *Pressure*: The surrounding has to be considered. The particle experiences a negative gradient towards the largest pressure decrease (low pressure areas) Computed with

$$-\nabla p \, V$$

- *Viscosity*: Can be described as "internal friction" and more accurately as *diffusion of velocities*.
- *Diffusion* Strength of dynamic viscosity coefficient:

$$V\mu\nabla \cdot \nabla u = V\mu\Delta u$$

Deduction of formula:

Newton's second law: $F = ma$

$$F = m\frac{Du}{Dt}$$

$$mg - V\nabla p + V\mu\Delta u = m\frac{Du}{Dt} \quad \text{Forces so far}$$

$$\rho g - \nabla p + \mu\Delta u = \rho\frac{Du}{Dt} \quad \text{Divide by } V$$

$$g - \frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\Delta u = \frac{Du}{Dt} \quad \text{Divide by Density}$$

Rearrange and use kinematic viscosity:

$$\frac{Du}{Dt} = -\frac{1}{\rho}\nabla p + \nu\Delta u + g$$

The viscosity term is usually left out on grid based simulations since inaccuracies are introduced. *Particle simulations* need viscosity to stabilise the system!

Material Derivative Change of a quantity $q(t,x)$ during motion:

$$\begin{aligned}\frac{d}{dt}q(t,x) &= \frac{\partial q}{\partial t} + \frac{\partial q}{\partial x}\cdot\frac{dx}{dt} \\ &= \underbrace{\frac{\partial q}{\partial t}}_{\text{change of } q \text{ at fixed point}} + \underbrace{\nabla q \cdot u}_{\text{in-/outflow}} \\ &= \frac{Dq}{Dt}\end{aligned}$$

The material derivative result in two different viewpoints:

- **Eulerian viewpoint**: Use both the change of $q(t,x)$ and the in-/outflow:

$$\frac{Dq}{Dt} = \frac{\partial q}{\partial t} + \nabla q \cdot u$$

- **Lagrangian viewpoint**: Only the change of $q(t,x)$ is used:

$$\frac{Dq}{Dt} = \frac{\partial q}{\partial t} + \nabla q \cdot u$$

Conservation of mass

$$\nabla \cdot u = 0$$

This condition also models *incompressibility*.

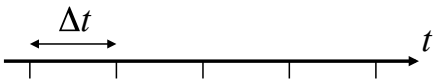
Reformulating the equation

$$\begin{aligned}\text{We have:}\quad \nabla \cdot u &= 0 \\ u'^n &= f(u^n) \\ u^{n+1} &= u'^n - \Delta t \frac{1}{\rho}\nabla p \\ \text{Thus:}\quad \nabla \cdot u^{n+1} &= 0 \\ \text{Insert:}\quad \nabla \cdot u'^n - \Delta t \frac{1}{\rho}\nabla \cdot \nabla p &= 0 \\ \text{Reaarange:}\quad \nabla \cdot \nabla \cdot p &= \frac{\rho}{\Delta t}\nabla \cdot u'^n\end{aligned}$$

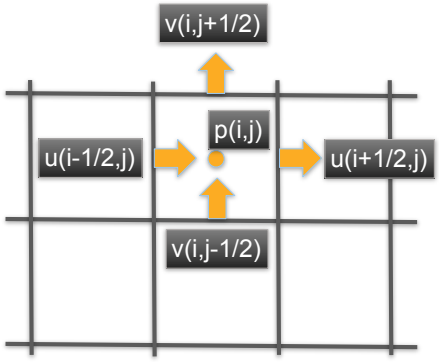
Results in a *Poisson Equation* since the right side of the equation is fully known.

16.3 Numerical Solution

Time discretisation with fixed time step Δt



Space discretisation Use a staggered grid (*MAC grid*) with cubical cells. The velocity components are defined on the faces of the cells (so that the staggering is more stable). The pressure is defined at the center of the grid.



16.3.1 Central Differences

Central differences provide an accuracy of $\mathcal{O}\left((\Delta x)^2\right)$:

$$\frac{\partial q}{\partial x} = \frac{q_{i+1} - q_{i-1}}{2\Delta x}$$

The computation can run into a null space problem since not only the constant function evaluates to zero. Solutions:

- Use forward or backward difference
- Make use of staggered grid and and sample q qt halfway points:

$$\frac{\partial q}{\partial x} = \frac{q_{i+1/2} - q_{i-1/2}}{\Delta x}$$

16.3.2 Algorithm Summary

1. Update grid (mark cells)
2. Advance the velocity field:
   - Advection (semi-Lagrange step for each velocity component)
   - Apply external forces (gravity,...)
   - Calculate and apply pressure (divergence-free)
3. Move marker particles
4. Start over...

17 SPH Solvers

Instead of solving the *Navier-Stokes equations* on a grid, we are using particles instead. SPH solvers are therefore Langrangian based solvers. Each particle represents a volume. This assumes that fluids are continuous: *Continuum assumption*

- Properties such as density, pressure, velocity, etc are well-defined at "infinitely" many points.
- Properties are assumed to *vary continuously* from one point to another.

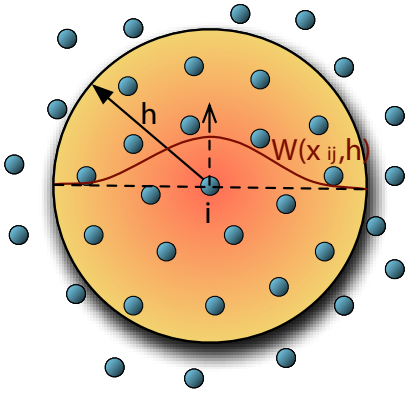
Equations for the particle method:

$$\begin{aligned}\underbrace{\frac{\partial u}{\partial t}}_{\text{advection}} &= \underbrace{-\frac{1}{\rho}\nabla p}_{\text{pressure}} + \underbrace{\nu \Delta u}_{\text{viscosity}} + g \\ \nabla \cdot u &= 0\end{aligned}$$

- The total mass is conserved as long as the particle number is *constant*. This however doesn't imply that the fluid is incompressible.
- The velocity divergence $\neq 0$: Therefore the density varies inside the fluid.

17.1 SPH

Particles concentrate mass at a point. A continuos field is described by smooth particles: A smoothing kernel describes distribution of attributes around particle:



17.1.1 Smoothing Kernels for SPH

Each kernel is described by a integral representation of a field variable:

$$A(x) = \int_\Omega A(x)\delta(x - x')dx$$

The dirac delta function is usually replaced by a smoothing kernel $W(x, h)$:

$$A(x) = \int_\Omega A(x)W(x - x', h)dx$$

Kernel properties

- *Normalisation condition*

$$\int W(x - x', h)dx' = 1$$

- *Dirac delta function*

$$\lim_{h\mapsto 0} W(x - x', h) = \delta(x - x')$$

- *Compact condition*

$$W(x - x', h) = 0 \quad \text{when } ||x - x'|| > h$$

Several Kernels exist [Müller03], [Monaghang92,05]:

$$W_{poly6}(x_{ij}, h) = \frac{315}{64\pi h^9}\begin{cases}(h^2 - x_{ij}^2)^3 & 0 \le x_{ij} \le h\\ 0 & \text{otherwise}\end{cases}$$

$$W_{spiky}(x_{ij}, h) = \frac{15}{\pi h^6}\begin{cases}(h - x_{ij})^3 & 0 \le x_{ij} \le h\\ 0 & \text{otherwise}\end{cases}$$

$$W_{viscous}(x_{ij}, h) = \frac{15}{2\pi h^3}\begin{cases}-\frac{x_{ij}^3}{2h^3} + \frac{x_{ij}^2}{h^2} + \frac{h}{2x_{ij}} - 1 & 0 \le x_{ij} \le h\\ 0 & \text{otherwise}\end{cases}$$

The smoothing can be computed like this for a location $A(x)$:

$$A(x) = \sum_j \frac{m_j}{\rho_j} A_j W(x - x_j, h).$$

This is beneficial for the computation since the differentiation only affects the kernel. The gradient and laplacian can be easily calculated:

$$\nabla A(x) = \sum_j \frac{m_j}{\rho_j} A_j \nabla W(x - x_j, h)$$
$$\nabla^2 A(x) = \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W(x - x_j, h)$$

17.2 Density

Each particle has the same mass m. The density field induced by a particle i is:

$$\rho_i = \sum m_j W(x_{ij}, h).$$

Problem: The *density is underestimated at the boundaries*.

Densities are initially set and evolve over time. This results in inaccuracies in a long-term simulation and thus make said simulation unstable. Rate of change:

$$\frac{d\rho_i}{dt} = \sum_j m_j(v_i - v_j) \cdot \nabla W(x_{ij}, h)$$

17.3 Pressure

The density is used to compute the pressure of each particle. *Equation of state* [Batchelor]:

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

ρ_0 : rest density of water ($1000 kg/m^3$)
k : gas constant
γ : 7

17.4 Pressure Force

The pressure density yields:

$$F_i^{pressure} = -\frac{m_i}{\rho_i}\sum_j \frac{m_j}{\rho_j}\frac{p_i + p_j}{2}\nabla W(x_{ij}, h)$$
$$= -\sum_j m_i m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right)\nabla W(x_{ij}, h)$$

17.5 Time Stepping

The time step size direrectly correlates with efficiency. In order to achieve 25 fps we need to represent $0.04s$ per frame. With a $\Delta t = 0.001$ we therefore need to compute 40 simulation steps per frame.

17.5.1 Courant Condition

If a particle is to be propagated with a maximum velocity v, it must not be integrated with a too large time step, or some grid points will be leaped.

- Each particle must not be passed by

$$\Delta t_c \leq \alpha \frac{h}{c}$$

  where c is for our case considered as the *square root of the speed of sound*.

- High viscosity has to be considered as well (speed of sound is lower)
- *Large forces* lead to

$$\Delta t_f = 0.25 \cdot \min_i \left(\frac{h}{||f_i||} \right)$$

  Which results in a timestep of

$$\Delta t = \min \left(\Delta t_c, \Delta t_f \right)$$

- The less compressible, the smaller Δt

17.6 Algorithm Summary

while animating **do**
  for all i **do**
    find neighbourhoods N_i
  end for
  for all i **do**
    compute density ρ_i
    compute pressure p_i
  end for
  for all i **do**
    compute forces $F^{p,\nu,g,ext}$
  end for
  for all i **do**
    compute new velocity $v_i(t+1)$
    compute new position $x_i(t+1)$
    collision handling
  end for
end while

17.7 Fluid Stiffness

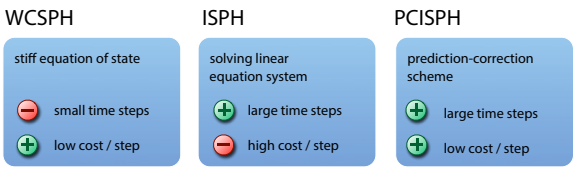
Gas equation:

$$p_i = k\rho_0 \left(\frac{\rho_i}{\rho_0} - 1 \right) = k(\rho_i - \rho_0)$$

k : stiffness parameter

With standard SPH k is in the range of 1000. The liquid is modelled as a compressible matter: Which results in a *High density variation* and a *small speed of sound*. This however makes it difficult to simulate water. The simplest solution would be increasing k (which would require decreasing the time step) or to use *WCSPH*.

17.8 Incompressibility Methods



WCSPH The computation cost increases with decreasing compressibility since the higher stiffness requires smaller time steps.

ISPH A possion equation is used to project velocity onto a divergence-free space. The formulation and solving of the equation system is difficult on a unstructured particle configuration.

PCISPH The pressure is updated efficiently by an iterative prediction-correction scheme.

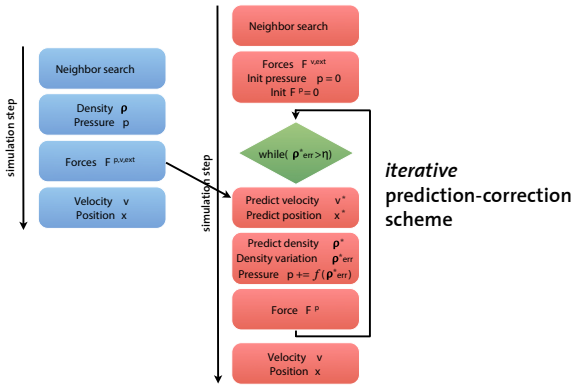


Figure 10: PCISPH