Numerical methods

Selected chapters on astrophysics

Pavel Ševeček

sevecek@sirrah.troja.mff.cuni.cz

Astronomical Institute of Charles University

Outline

1. **Finite difference method**

   Explicit vs. implicit schemes. Leapfrog integrator, predictor-corrector methods, Runge-Kutta, Bulirsch-Stoer. Truncation error and stability. Boundary conditions. 2D and 3D problems. Solution of sparse linear systems.

2. **Finite element method**

   Weak formulation of PDEs. Method of weighted residuals, Galerkin method. Mass and stiffness matrix. (Semi-)linearization, Picard iteration. Dirichlet and Neumann boundary conditions.

3. **Smoothed particle hydrodynamics**

   SPH kernels. Discretization of hydrodynamic equations. Conservation laws vs. discretization error. Smoothing lengths, adaptive spatial resolution. Artificial viscosity. Surface representation and surface forces. Initial and boundary conditions, ghost particles. Efficient neighbor queries.

Disclaimer

Before you start coding something, chances are there already is a library or a code that does what you need, and it does it better that you would have made it.

Astrophysics Source Code Library

`https://ascl.net/`

What is it about?

- Partial differential equations are everywhere in physics, e.g.:

$$\nabla \cdot \boldsymbol{D} = \rho$$
$$\nabla \cdot \boldsymbol{B} = 0$$
$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t}$$
$$\nabla \times \boldsymbol{H} = \boldsymbol{j} + \frac{\partial D}{\partial t}$$

- PDEs generally unsolvable analytically, except for special cases, limits, etc.
- Approximate solutions required — convert derivatives ("infinitelly small difference") into finite differences

Problem definition

For general PDE:

$$\mathcal{L}[f](\boldsymbol{r}, t) = 0, \qquad \boldsymbol{r} \in \Omega, t \in (t_0, \infty)$$

we need to specify:

- Initial conditions

$$f(\boldsymbol{r}, t_0) = f_0 \qquad \forall \boldsymbol{r} \in \Omega$$

Problem definition

For general PDE:

$$\mathcal{L}[f](\boldsymbol{r}, t) = 0, \qquad \boldsymbol{r} \in \Omega, t \in (t_0, \infty)$$

we need to specify:

- Initial conditions

$$f(\boldsymbol{r}, t_0) = f_0 \qquad \forall \boldsymbol{r} \in \Omega$$

- Boundary conditions

$$\mathcal{S}[f](\boldsymbol{r}, t) = 0 \qquad \forall \boldsymbol{r} \in \partial\Omega$$

  e.g.

$$f(\boldsymbol{r}, t) = f_0$$

Finite differences

- Numerically solve one-dimensional equation:

$$y'(t) = f(t, y(t))$$

- Taylor expansion:

$$y(t + \Delta t) = y(t) + y'(t)\Delta t + \frac{1}{2}y''(t)\Delta t^2 + ...$$

Finite differences

- Numerically solve one-dimensional equation:

$$y'(t) = f(t, y(t))$$

- Taylor expansion:

$$y(t + \Delta t) = y(t) + y'(t)\Delta t + \frac{1}{2}y''(t)\Delta t^2 + ...$$

- To linear order:

$$y'(t) = \frac{y(t + \Delta t) - y(t)}{\Delta t}$$

Finite differences

- Numerically solve one-dimensional equation:

$$y'(t) = f(t, y(t))$$

- Taylor expansion:

$$y(t + \Delta t) = y(t) + y'(t)\Delta t + \frac{1}{2}y''(t)\Delta t^2 + \ldots$$

- To linear order:

$$y'(t) = \frac{y(t + \Delta t) - y(t)}{\Delta t}$$

- Centered derivative:

$$y'(t) = \frac{y(t + \Delta t) - y(t - \Delta t)}{2\Delta t}$$

  no offset, but twice the interval, worse locality

Higher-order equations

- Second-order equations can be converted into a set of first-order equations, e.g.

$$\boldsymbol{r}''(t) = -\frac{GM}{r^3}\boldsymbol{r}$$

is equivalent to:

$$r'(t) = v(t)$$
$$v'(t) = -\frac{GM}{r^3}\boldsymbol{r}$$

Higher-order equations

- Second-order equations can be converted into a set of first-order equations, e.g.

$$\boldsymbol{r}''(t) = -\frac{GM}{r^3}\boldsymbol{r}$$

  is equivalent to:

$$r'(t) = v(t)$$
$$v'(t) = -\frac{GM}{r^3}\boldsymbol{r}$$

- Calculate second derivative:

$$y(t + \Delta t) = y(t) + y'(t)\Delta t + \frac{1}{2}y''(t)\Delta t^2$$
$$y(t - \Delta t) = y(t) - y'(t)\Delta t + \frac{1}{2}y''(t)\Delta t^2$$
$$\frac{1}{2}y(t + \Delta t) - y(t) + \frac{1}{2}y(t - \Delta t) = y''(t)\Delta t^2$$

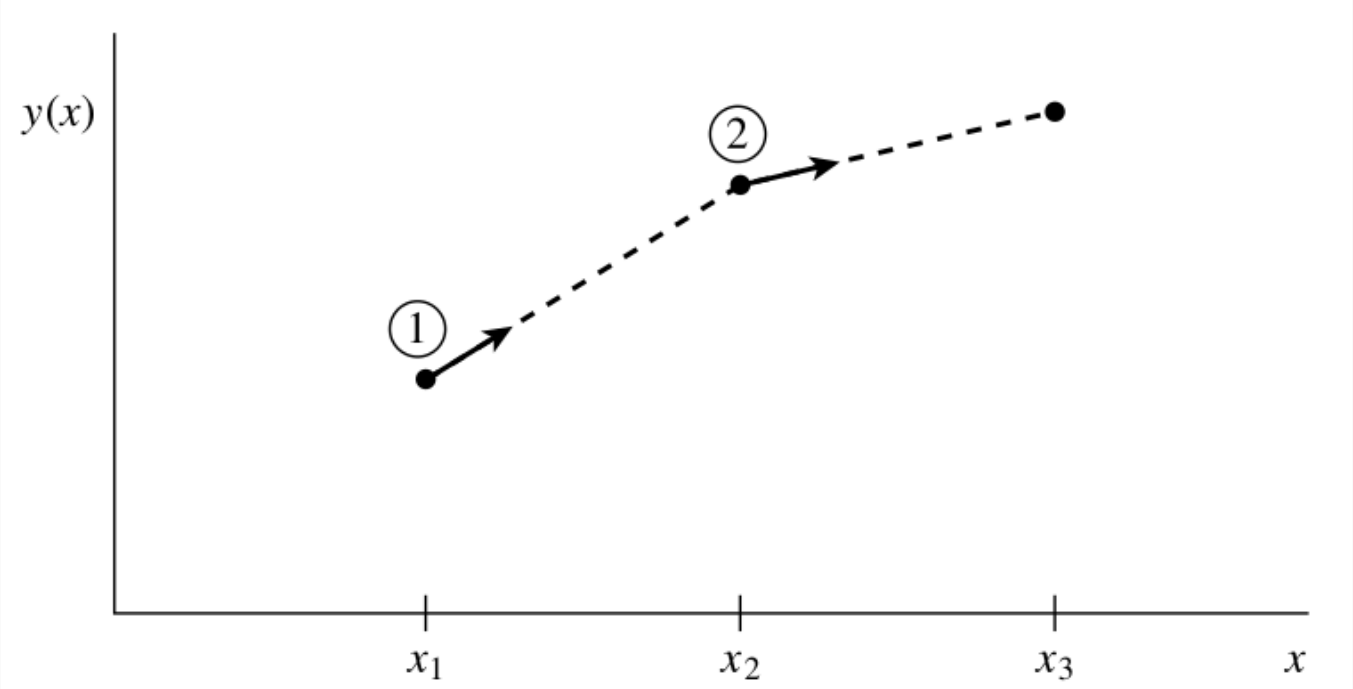
Integrators

- Euler
- Leapfrog
- Runge-Kutta
- Predictor-Corrector
- Bulirsch-Stoer
- ...

Euler method

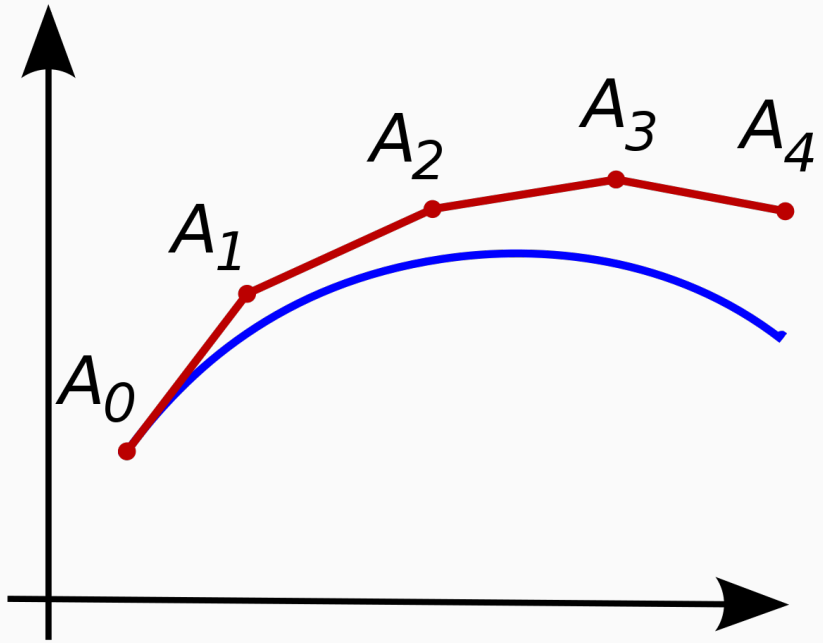
Step computed from derivative at point n:

$$y_{n+1} = y_n + f(t_n, y_n)\Delta t$$



Euler method — properties

+ Fast, simple
– First order accurate :(
– Errors accumulate over time



Euler method

- Explicit method:

$$y_{n+1} = y_n + f(t_n, y_n)\,\Delta t$$

- Implicit method:

$$y_{n+1} = y_n + f(t_{n+1}, y_{n+1})\,\Delta t$$

- May be combined — Crank-Nicolson scheme:

$$y_{n+1} = y_n + ((1-\theta) f(t_n, y_n) + \theta f(t_{n+1}, y_{n+1}))\,\Delta t$$

Explicit (forward) vs. implicit (backward)

Explicit $y_{n+1} = y_n + f(t_n, y_n)\,\Delta t$

- \+ RHS can be computed from known values y_n
- \+ Function f can be non-linear
- – Generally worse stability properties

Explicit (forward) vs. implicit (backward)

Explicit $y_{n+1} = y_n + f(t_n, y_n)\,\Delta t$

- \+ RHS can be computed from known values y_n
- \+ Function f can be non-linear
- – Generally worse stability properties

Implicit methods $y_{n+1} = y_n + f(t_{n+1}, y_{n+1})\,\Delta t$

- \+ More stable, allows larger time steps
- – Both LHS and RHS depend on y_{n+1}
- – Function f must be either linear or approximated by
  e.g. Newton-Rhapson method
  $\longrightarrow$ leads to a (sparse) system of linear equations

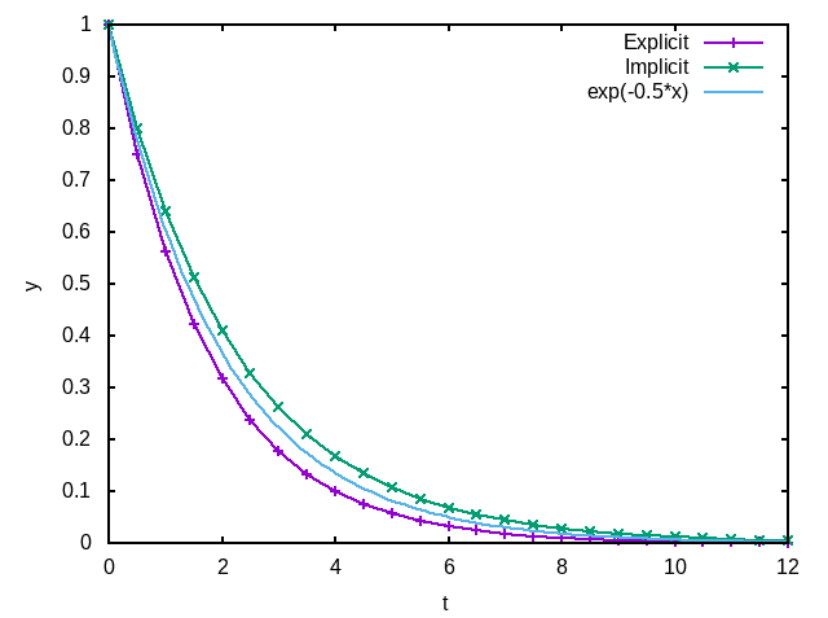
Example: exponential damping

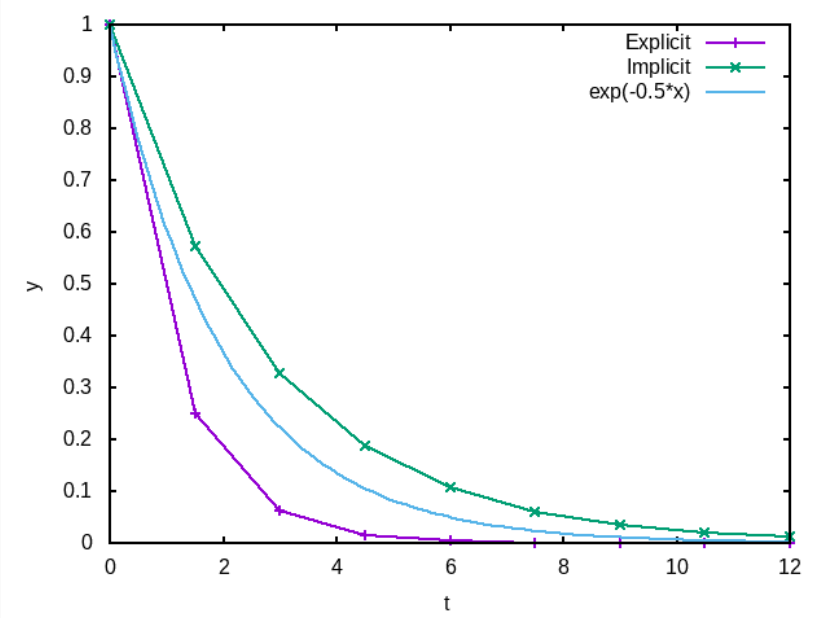
Solve equation:

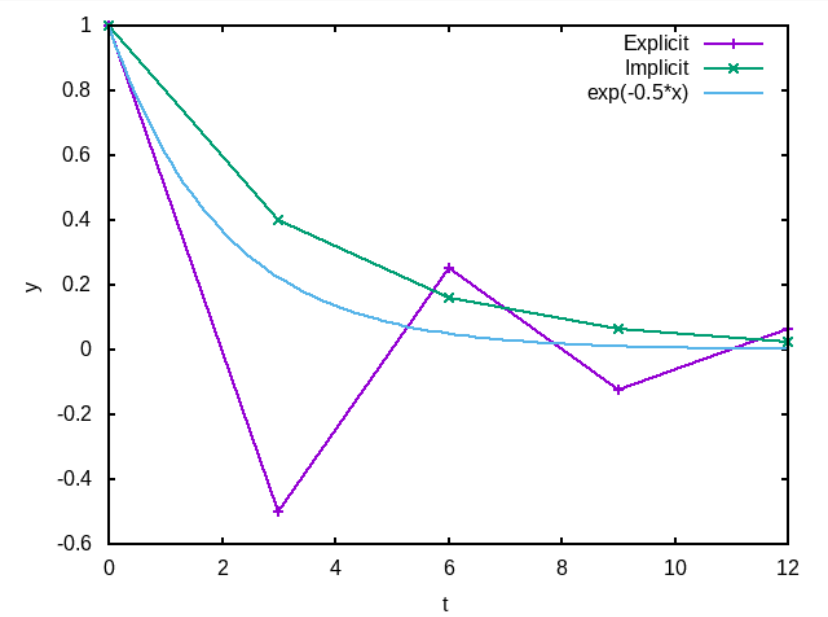
$$\frac{\mathrm{d}y}{\mathrm{d}t} = -ky$$

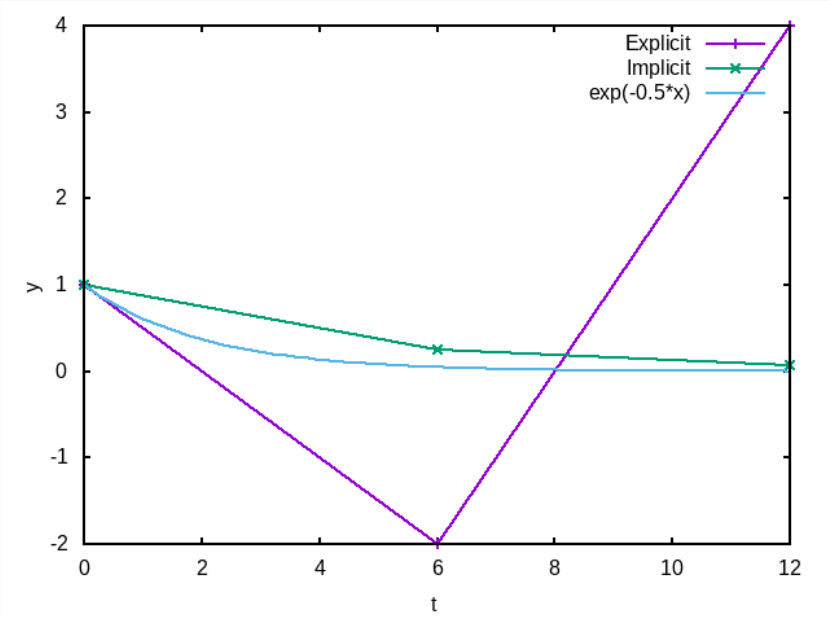
- Explicit Euler: $y_{n+1} = y_n - k y_n \Delta t$
- Implicit Euler: $y_{n+1} = y_n - k y_{n+1} \Delta t = y_n/(1 + k\Delta t)$

Example: exponential damping









Leapfrog

- Half-step difference between positions and velocities
- Drift-Kick-Drift (DKD)

$$
\begin{aligned}
x_{n+\frac{1}{2}} &= x_n + \frac{1}{2} v_n \mathrm{d}t \\
v_{n+1} &= v_n + f(t_{n+\frac{1}{2}}, x_{n+\frac{1}{2}}) \mathrm{d}t \\
x_{n+1} &= x_{n+\frac{1}{2}} + \frac{1}{2} v_{n+1} \mathrm{d}t
\end{aligned}
$$

Leapfrog

- Half-step difference between positions and velocities
- Drift-Kick-Drift (DKD)

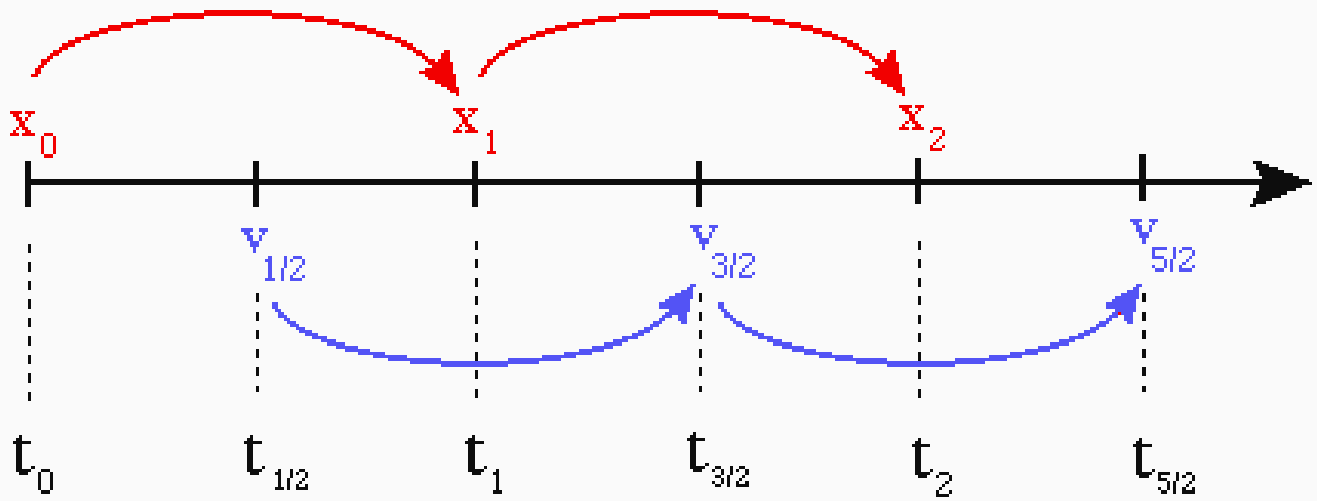
$$
\begin{aligned}
x_{n+\frac{1}{2}} &= x_n + \frac{1}{2} v_n \mathrm{d}t \\
v_{n+1} &= v_n + f(t_{n+\frac{1}{2}}, x_{n+\frac{1}{2}}) \mathrm{d}t \\
x_{n+1} &= x_{n+\frac{1}{2}} + \frac{1}{2} v_{n+1} \mathrm{d}t
\end{aligned}
$$

- Kick-Drift-Kick (KDK)

$$
\begin{aligned}
v_{n+\frac{1}{2}} &= v_n + \frac{1}{2} f(t_n, x_n) \mathrm{d}t \\
x_{n+1} &= x_n + v_{n+\frac{1}{2}} \mathrm{d}t \\
v_{n+1} &= v_{n+\frac{1}{2}} + \frac{1}{2} f(t_{n+1}, x_{n+1}) \mathrm{d}t
\end{aligned}
$$

Leapfrog

- With constant time step — KDKDKDKD ...
- Time-reversible
- Symplectic — conserves total energy



2th order Runge-Kutta

Evaluate y'_n, do a "test" step by $\Delta t/2$, evaluate y' at the center of the interval and use it for the final step

$$\Delta y_1 = f\left(t_n, y_n\right) \Delta t$$

$$\Delta y_2 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta y_1}{2}\right) \Delta t$$

$$y_{n+1} = y_n + \Delta y_2$$

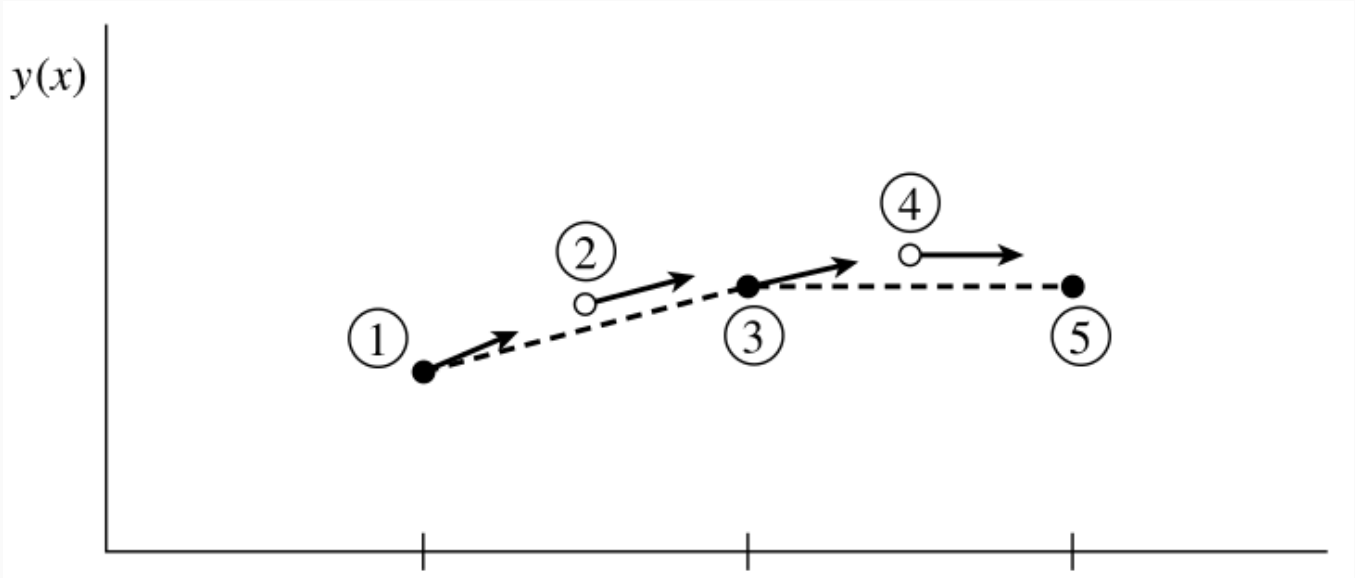
2th order Runge-Kutta

Evaluate y'_n, do a "test" step by $\Delta t/2$, evaluate y' at the center of the interval and use it for the final step

$$\Delta y_1 = f\left(t_n, y_n\right) \Delta t$$

$$\Delta y_2 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta y_1}{2}\right) \Delta t$$

$$y_{n+1} = y_n + \Delta y_2$$



4th order Runge-Kutta

Same idea, except we do 4 steps:

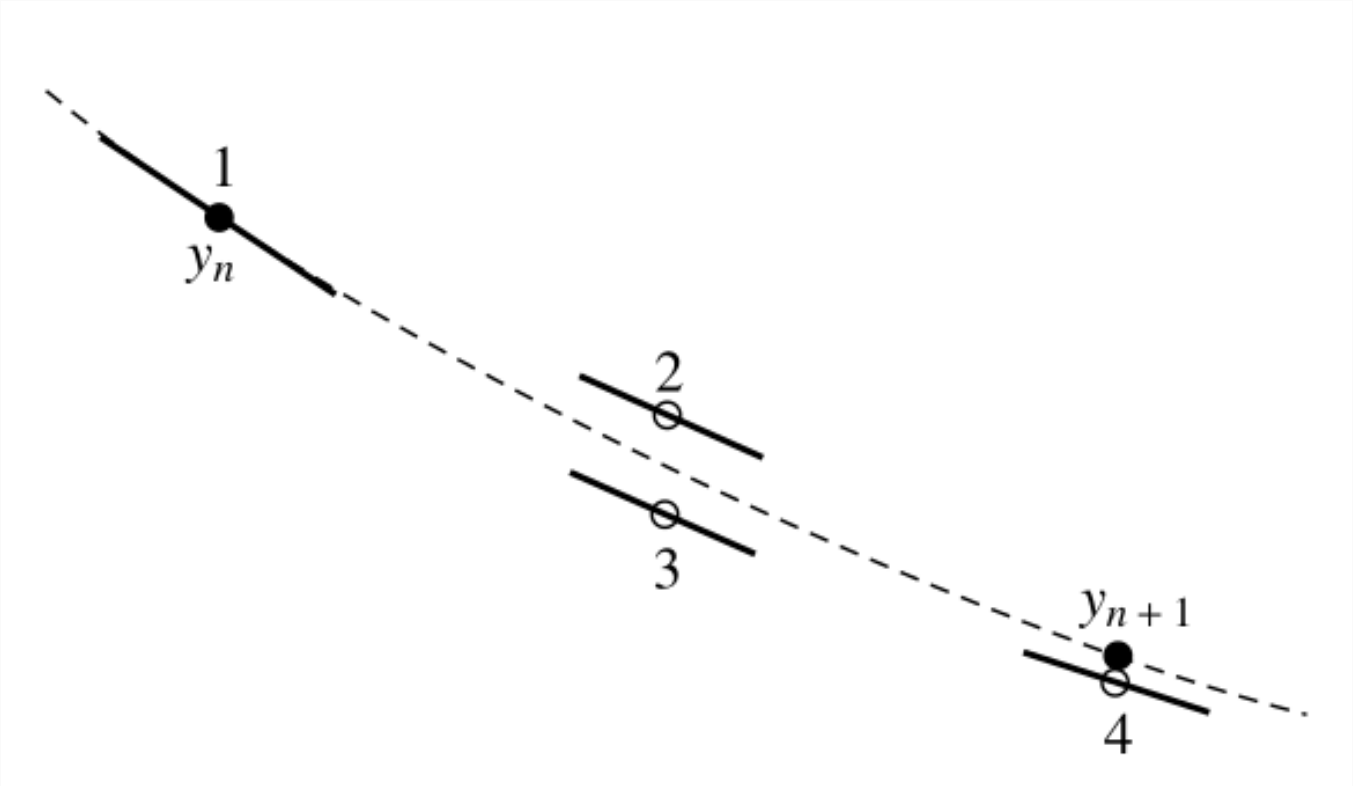
$$y_{n+1} = y_n + \frac{1}{6}(\Delta y_1 + 2\Delta y_2 + 2\Delta y_3 + \Delta y_4)$$

where

$$\begin{aligned}
\Delta y_1 &= f\left(t_n, y_n\right) \Delta t \\
\Delta y_2 &= f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta y_1}{2}\right) \Delta t \\
\Delta y_3 &= f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta y_2}{2}\right) \Delta t \\
\Delta y_4 &= f\left(t_n + \Delta t, y_n + \Delta y_3\right) \Delta t
\end{aligned}$$

Runge-Kutta — properties

- The go-to integrator
- Accuracy $\mathcal{O}(\Delta t^4)$
- Works with discontinuities and "ugly" functions
- Requires four evaluation of f per timestep — good trade-of between performance and accuracy



Predictor-Corrector

- Prediction using $f(t_n, y_n)$:

$$y_{n+1}^{\text{pred}} = y_n + f(t_n, y_n)\Delta t$$

- Correction using $f(t_{n+1}, y_{n+1}^{\text{pred}})$:

$$y_{n+1} = y_n + \frac{1}{2}\left(f(t_n, y_n) + f(t_{n+1}, y_{n+1}^{\text{pred}})\right)$$

- Can be modified to 1 function evaluation per time step
- The difference between prediction and correction — error estimate

Modified midpoint method

- Does m substeps of size $h = \Delta t/m$ within single timestep
- Needs to evaluate function $f(t, y)$ $(n+1)$-times:

$$
\begin{aligned}
z_0 &= y_n \\
z_1 &= z_0 + hf(t_n, z_0) \\
&\dots \\
z_m &= z_{m-2} + 2hf(t_m + (m-1)h, z_{m-1})
\end{aligned}
$$

- Finally

$$y_{n+1} = y_n + \frac{1}{2}\left(z_m + z_{m-1} + hf(x + \Delta t, z_m)\right)$$

- Mainly used as part of Bulirsch-Stoer integrator

Bulirsch-Stoer

- Precise solution y_{n+1} — approximated by estimates with different number of substeps m
- Larger $m \longrightarrow$ lower numerical error
- We can view the true solution as function of m:
  1. compute estimates for various m
  2. fit estimates with a smooth function (polynomial, rational function, ...)
  3. find limit, corresponding to substep size $h \to 0$
- Requires smooth function $f(t, x)$, does not handle discontinuities well

Bulirsch-Stoer

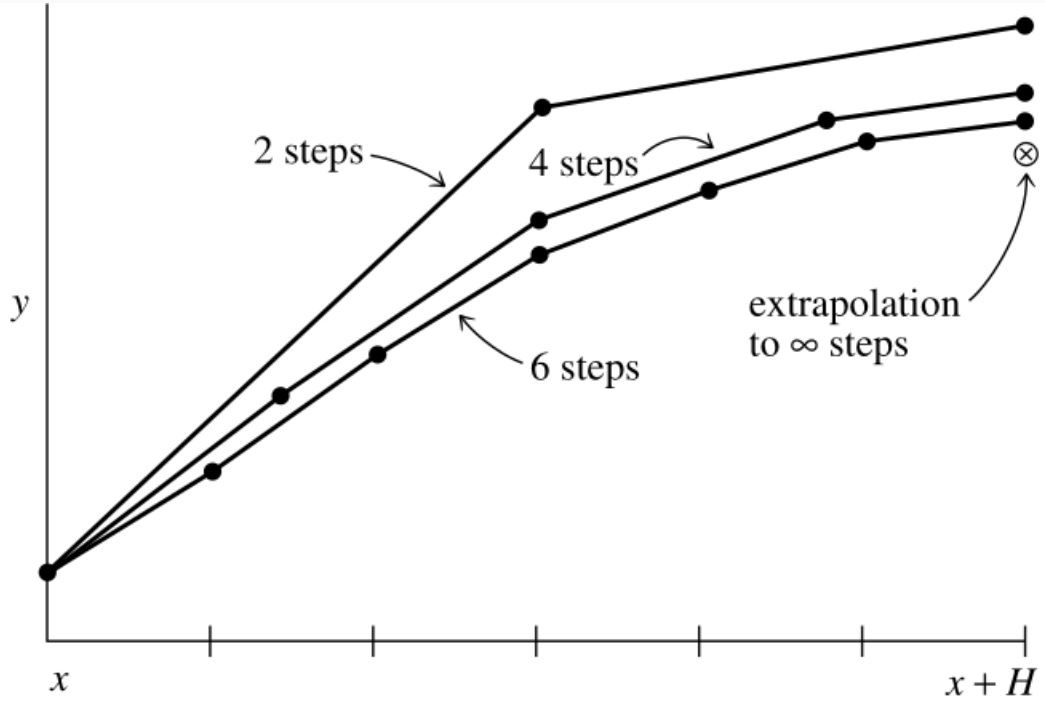


Figure 16.4.1. Richardson extrapolation as used in the Bulirsch-Stoer method. A large interval H is spanned by different sequences of finer and finer substeps. Their results are extrapolated to an answer that is supposed to correspond to infinitely fine substeps. In the Bulirsch-Stoer method, the integrations are done by the modified midpoint method, and the extrapolation technique is rational function or polynomial extrapolation.

Adaptive timestep

- Time step should be computed automatically be the integrator
- Do larger steps when quantities are constant or change linearly
- Do smaller steps when quantities change rapidly
- Time step controls stability and truncation error

$$y(t + \Delta t) = y(t) + y'(t)\Delta t + \overbrace{\frac{1}{2}y''(t)\Delta t^2}^{\text{error estimate}}$$

Step doubling

- Two steps:

$$y_1(t + \Delta t) = y(t) + f\left(t, y(t)\right) \Delta t$$
$$y_1(t + 2\Delta t) = y_1(t + \Delta t) + f(t + \Delta t, y_1(t + \Delta t))\Delta t$$

- One double step:

$$y_2(t + 2\Delta t) = y(t) + 2f(t, y(t))\Delta t$$

- Select Δt so that:

$$\|y_1 - y_2\| < \varepsilon$$

Adaptive timestep

- Value-to-derivative ratio:

$$\Delta t = C \frac{y_n}{y_n'} = C \frac{y_n}{f(t_n, y_n)}$$

  $\longrightarrow$ bounded relative error

- Problem for $y_n \to 0$ — $\Delta t = 0$
- Relative comparison does not work near zero

Adaptive timestep

- Value-to-derivative ratio:

$$\Delta t = C\frac{y_n}{y_n'} = C\frac{y_n}{f(t_n, y_n)}$$

  $\longrightarrow$ bounded relative error

- Problem for $y_n \to 0$ — $\Delta t = 0$
- Relative comparison does not work near zero
- Workaround: Use relative error bound for $|y_n| \gg 0$ and absolute bound for $y_n \simeq 0$:

$$\Delta t = C\frac{|y_n| + y_0}{|y_n'|}$$

- Cons: Necessary to select y_0 — free parameter (for each equation)

Boundary conditions

Matrix problem:

$$A_{ij} u_j = b_i$$

- Dirichlet condition — fixed value on the boundary

$$u_1 = c_1, u_N = c_N$$

$\longrightarrow$ modify the matrix and the right-hand side vector:

$$A_{11} = A_{NN} = 1$$
$$A_{1j} = A_{Nj} = 0 \quad \forall j, 2 \leq j \leq N-1$$
$$b_1 = c_1, b_N = c_N$$

Boundary conditions

Matrix problem:

$$A_{ij} u_j = b_i$$

- Dirichlet condition — fixed value on the boundary

  $$u_1 = c_1, u_N = c_N$$

  $\longrightarrow$ modify the matrix and the right-hand side vector:

  $$A_{11} = A_{NN} = 1$$
  $$A_{1j} = A_{Nj} = 0 \quad \forall j, 2 \leq j \leq N-1$$
  $$b_1 = c_1, b_N = c_N$$

  or alternatively

  $$A_{11} = A_{NN} = L$$
  $$b_1 = Lc_1, b_N = Lc_N$$

  where $L \simeq 10^{30}$

Boundary conditions

- Neumann condition — fixed derivative on the boundary

$$u_1' = d_1, u_N' = d_N$$

$\longrightarrow$ ghost points — u_0, u_{N+1}:

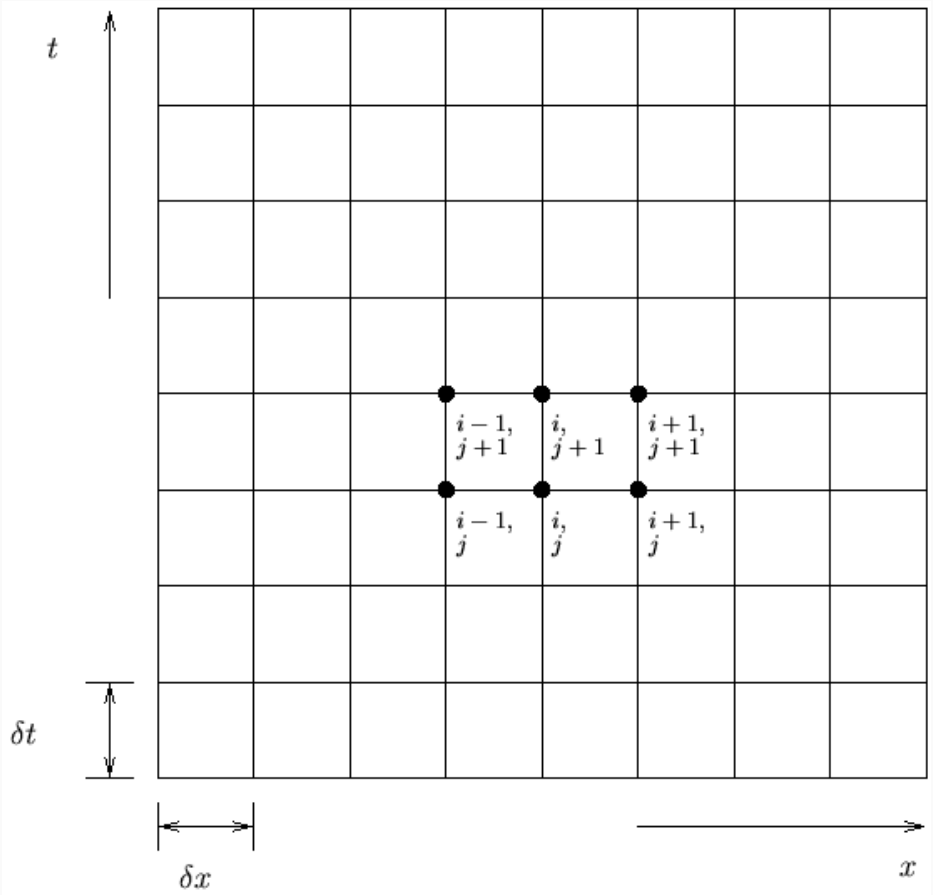
$$u_1' = \frac{u_2 - u_0}{2\mathrm{d}x}, u_N' = \frac{u_{N+1} - u_{N-1}}{2\mathrm{d}x}$$

2D finite differences

Solve

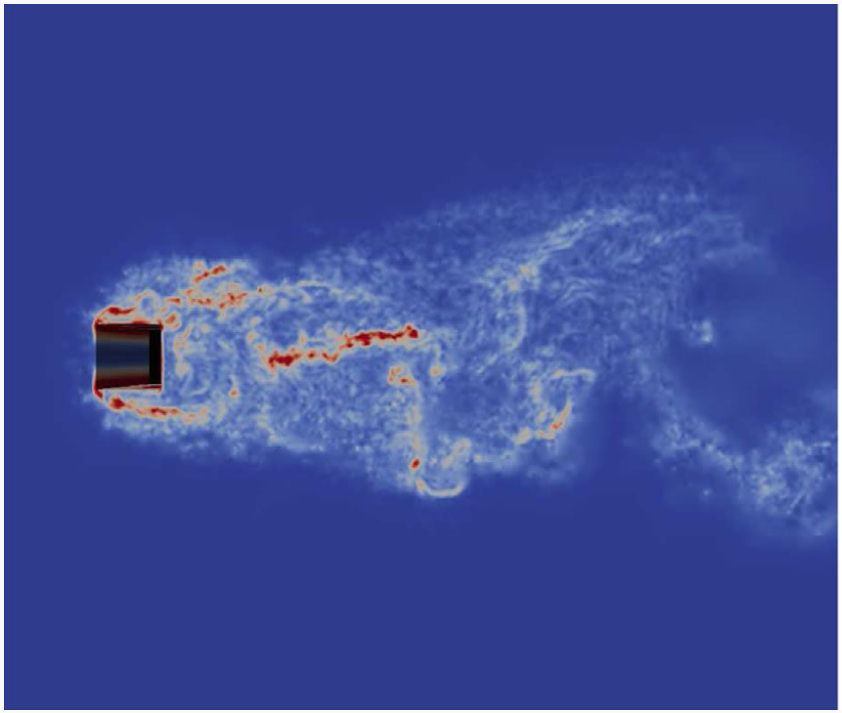
$$y'(x,t) = f(x,t,y(x,t))$$

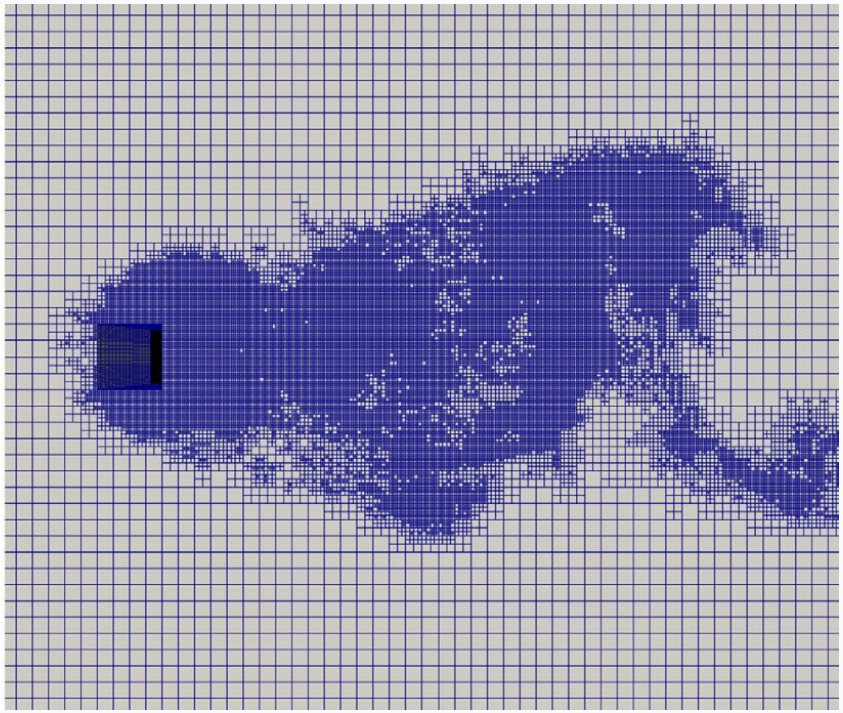
- Create grid (x_i, t_j), $0 \leq i \leq N_x$, $0 \leq j \leq N_t$
- Requires rectangular computational domain



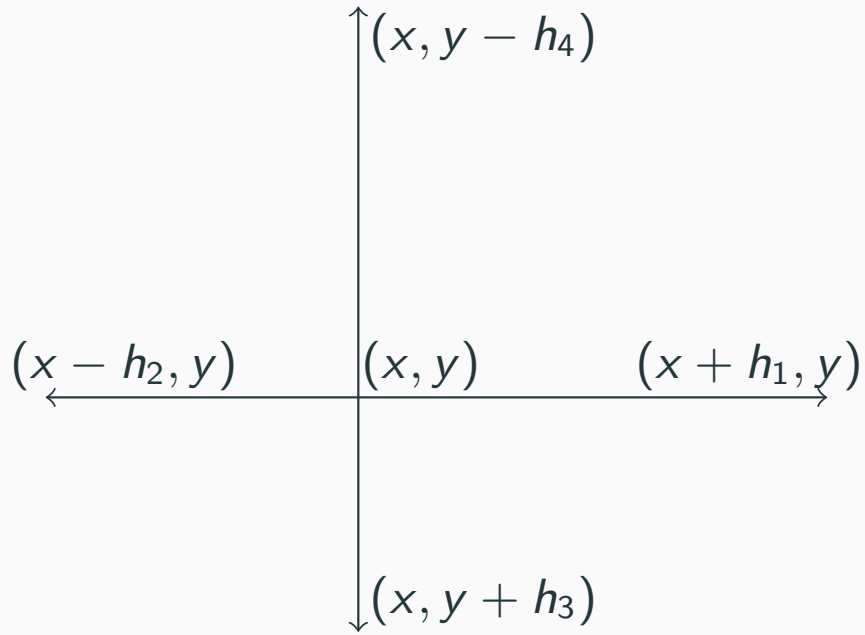
Non-uniform grids

- Constant spatial resolution might be inefficient
- Select fine resolution where the solution changes rapidly,
  coarse resolution where the solution is constant (or changes linearly)





Non-uniform grids

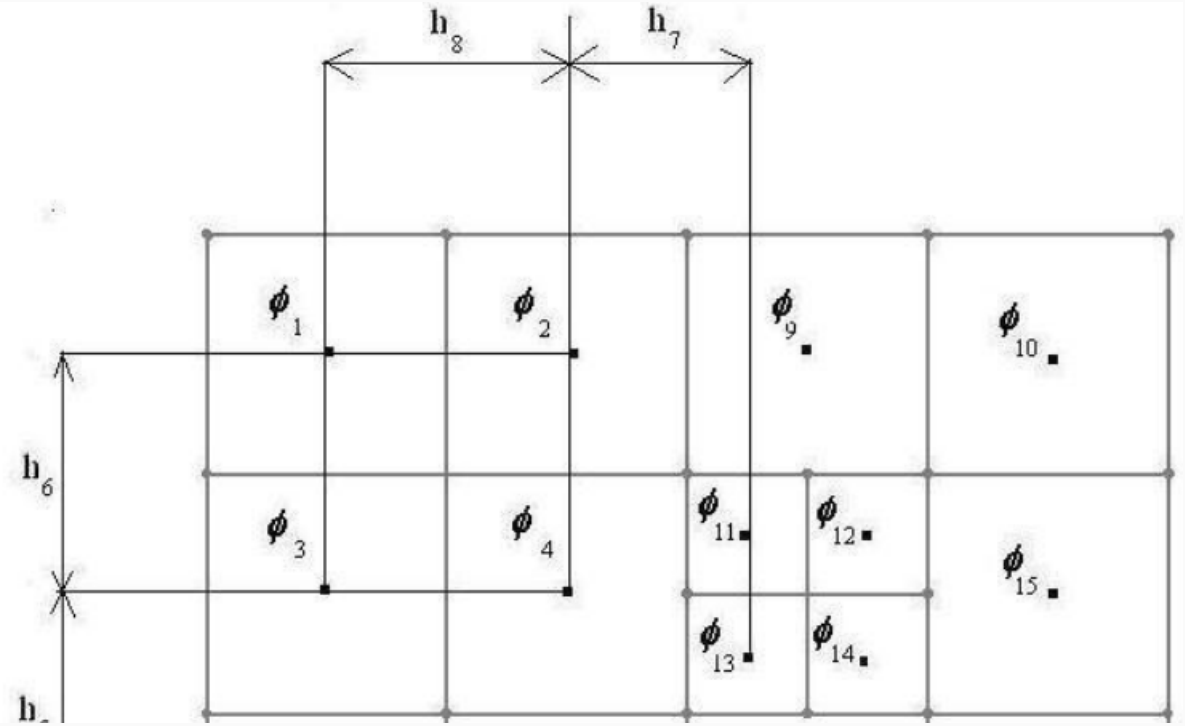


Derivatives have to be generalized, e.g.:

$$\frac{\partial^2 f}{\partial x^2} = \frac{2}{h_1 h_2}\left(\frac{h_2}{h_1 + h_2} f(x + h_1, y) - f(x, y) + \frac{h_1}{h_1 + h_2} f(x - h_2, y)\right)$$

Non-uniform grids

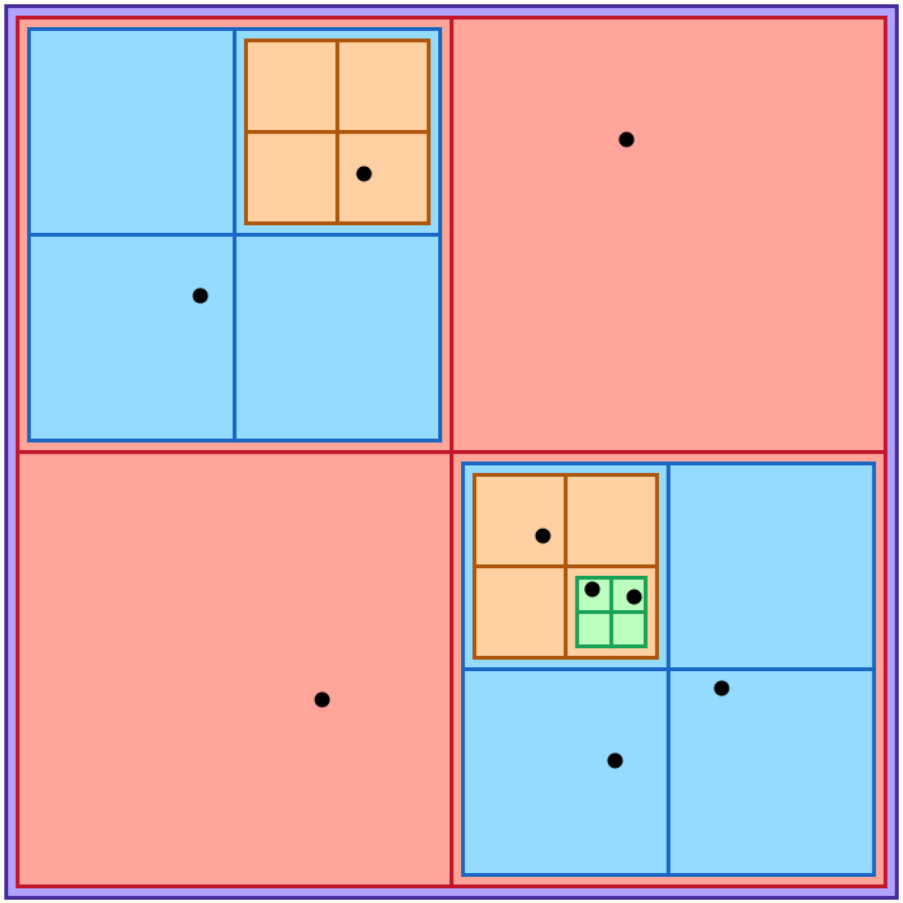
- Some cells have multiple neighbors, e.g. ϕ_4:
  $\longrightarrow$ use average of ϕ_{11} and ϕ_{13}

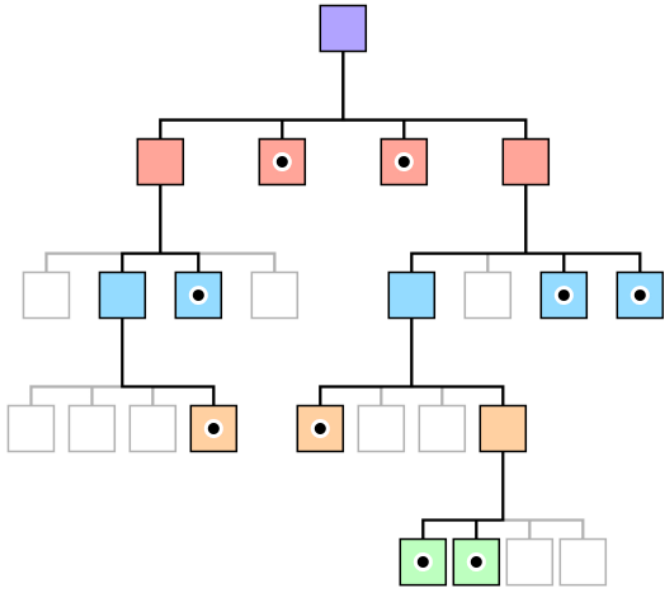


- Cell centered vs. node centered discretization
- Typically implemented using quadtree (octree)

Quadtree

- Cells have sizes 2^n
- $\mathcal{O}(\log N)$ queries





Stability

- Discretization approximates exact solution at $t = t_0 \longrightarrow$ discretization approximates exact solution at $t \to \infty$?
- Errors damped over time $\longrightarrow$ numerically stable solution
- Errors grow over time $\longrightarrow$ numerically unstable solution

Stability

- Discretization approximates exact solution at $t = t_0 \longrightarrow$ discretization approximates exact solution at $t \to \infty$?
- Errors damped over time $\longrightarrow$ numerically stable solution
- Errors grow over time $\longrightarrow$ numerically unstable solution
- von Neumann stability analysis — solution as superposition of harmonic waves

$$u(x,t) \sim \exp(ikx)\exp(-i\omega t)$$

Stability analysis — example

Diffusion equation:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = D \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}$$

$$u_j^{n+1} = u_j^n + R(u_{j+1}^n - 2u_j^n + u_{j-1}^n), \quad R = \frac{D\Delta t}{\Delta x^2}$$

Substitute $u_j^n = \exp(ik\Delta xj)$:

$$u_j^{n+1} = e^{ik\Delta xj} + R(e^{ik\Delta x(j+1)} - 2e^{ik\Delta xj} + e^{ik\Delta x(j-1)})$$

$$u_j^{n+1} = e^{ik\Delta xj} \left(1 + R(e^{ik\Delta x} + e^{-ik\Delta x} - 2)\right)$$

Stability analysis — example

Define a growth factor:

$$G = e^{-i\omega t} = 1 - 2R\,(1 - \cos(k\Delta x))$$

Numerical scheme is stable if $|G| \leq 1\ \forall k$, thus:

$$|G(k)| = |1 - 2R\,(1 - \cos(k\Delta x))\,| \leq |1 - 4R|$$

Numerical scheme is stable for:

$$R = \frac{D\Delta t}{\Delta x^2} \leq \frac{1}{2}$$

Solving linear systems

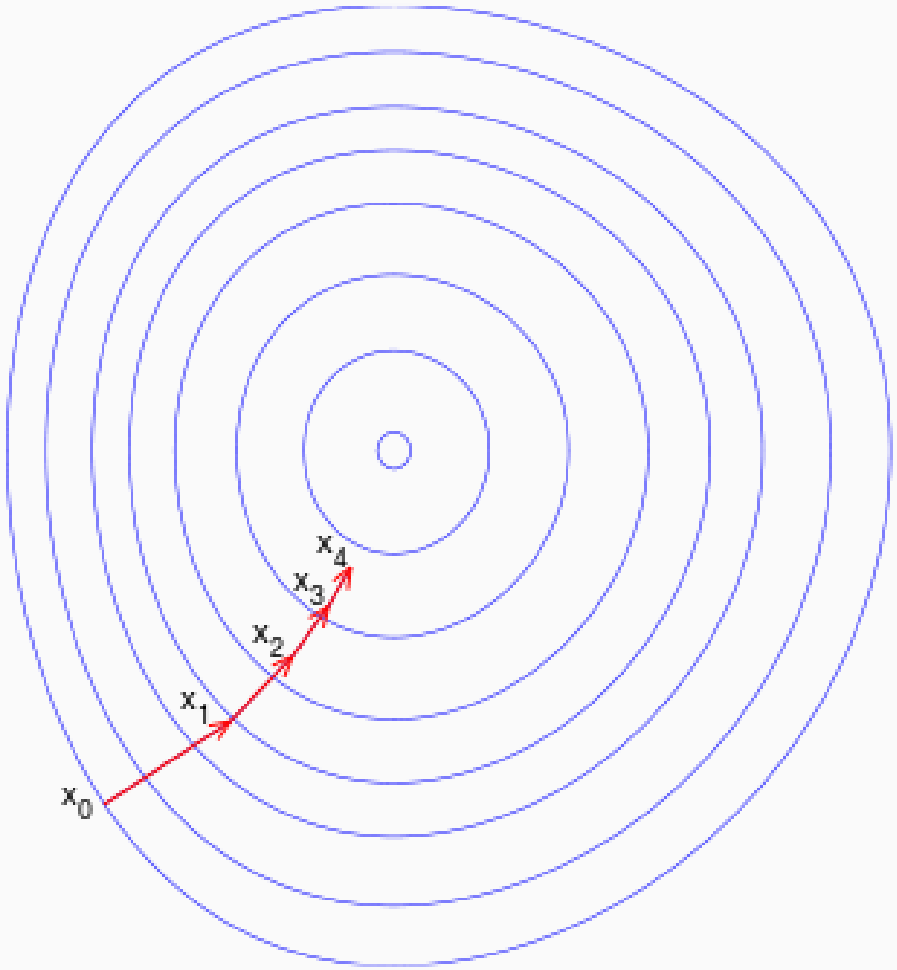
- Implicit schemes typically lead to a set of equations:

$$A_{ij}x_j = b_i$$

  where A_{ij} is a (sparse) matrix, b_j is a known vector of coefficients, x_j is the vector of solutions
- Precise methods (SVD-decomposition, LU, Cholesky)
- Iterative methods (Conjugate gradient)

Gradient descent

- Minimization of differentiable function



Gradient descent

Find a minimum of a function $F(\boldsymbol{x})$:

1. Start with a guess $\boldsymbol{x}_0$
2. Compute gradient $\nabla F(\boldsymbol{x}_n)$
3. Do a step
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \gamma_n \nabla F(\boldsymbol{x}_n)$$
4. If $\|\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\| > \epsilon$, go to 2

Gradient descent

Find a minimum of a function $F(\boldsymbol{x})$:

1. Start with a guess $\boldsymbol{x}_0$
2. Compute gradient $\nabla F(\boldsymbol{x}_n)$
3. Do a step
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \gamma_n \nabla F(\boldsymbol{x}_n)$$
4. If $\|\boldsymbol{x}_{n+1} - \boldsymbol{x}_n\| > \epsilon$, go to 2

To solve a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, do least-squares minimization:

$$F(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2$$

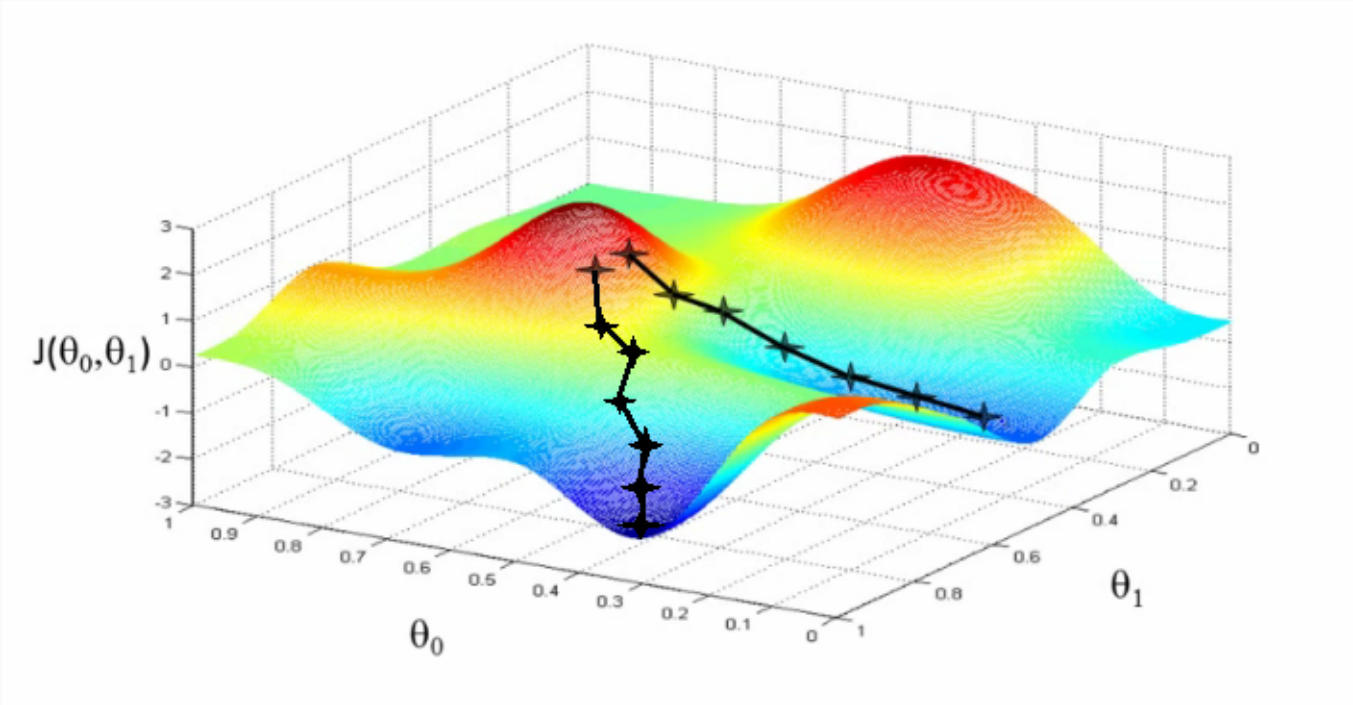
Using Euclidean metric:

$$\nabla F(\boldsymbol{x}) = 2\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$$

Gradient descent

- Iterative method
- Faster for low precision
- Convergence very slow near the minimum

  May end up in local minimum — depends on the initial guess $\boldsymbol{x}_0$



Example: wave equation

Solve

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

Discretized:

$$\dot{u}^{n+1}_{x,y} = \dot{u}^{n}_{x,y} + c^2 \left(\frac{u^{n+1}_{x+1,y} - 2u^{n+1}_{x,y} + u^{n+1}_{x-1,y}}{dx^2} + \frac{u^{n+1}_{x,y+1} - 2u^{n+1}_{x,y} + u^{n+1}_{x,y-1}}{dx^2} \right) \mathrm{d}t$$

$$u^{n+1}_{x,y} = u^{n}_{x,y} + \dot{u}^{n+1}_{x,y} \mathrm{d}t$$

Example: wave equation

Implicit discretization — rewrite to form $\boldsymbol{Ax} = \boldsymbol{b}$:

$$\boldsymbol{x} = \begin{pmatrix} u_{0,0}^{n+1} \\ \dots \\ u_{X,Y}^{n+1} \\ \dot{u}_{0,0}^{n+1} \\ \dots \\ \dot{u}_{X,y}^{n+1} \end{pmatrix} \qquad \boldsymbol{b} = \begin{pmatrix} u_{0,0}^{n} \\ \dots \\ u_{X,Y}^{n} \\ \dot{u}_{0,0}^{n} \\ \dots \\ \dot{u}_{X,y}^{n} \end{pmatrix}$$

Source code:

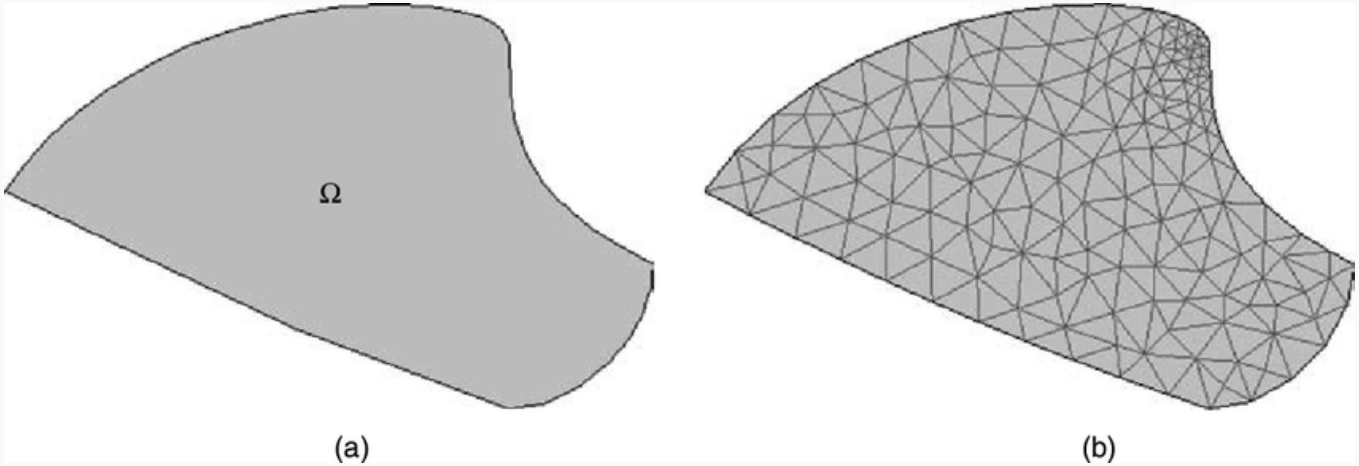
`https://gitlab.com/sevecekp/pdesolvers`

Finite element method

- Popular method for engineering applications
- Handles arbitrary domains
  — FDM requires *parameterizable* domains (rectangular, spherical, ...)
- Adaptive spatial resolution
  — FDM cells have sizes of 2^n
- Implicit handling of boundary conditions
- *Much* more difficult to implement compared to FDM
- Used for hydrodynamics, heat diffusion, structural analysis, ...

Finite element method

- Domain is discretized in elements — typically triangles in 2D, tetrahedra in 3D



Finite element method

- Goal: Given a differential operator $\mathcal{L}$, find a function $u(\boldsymbol{r})$ for which:

$$\mathcal{L}(u) = 0$$

(for example $\nabla^2 u = 0$)

Finite element method

- Goal: Given a differential operator $\mathcal{L}$, find a function $u(\boldsymbol{r})$ for which:

$$\mathcal{L}(u) = 0$$

  (for example $\nabla^2 u = 0$)
- Replace function $u(\boldsymbol{r})$ with a linear combination of basis functions:

$$\hat{u}(\boldsymbol{r}) \equiv \sum_{i=1}^{M} u_i N_i(\boldsymbol{r})$$

- Problem of finding $u(\boldsymbol{r})$ (infinite dimensions)
  $\longrightarrow$ finding finite number of u_i

Finite element method

- Goal: Given a differential operator $\mathcal{L}$, find a function $u(\boldsymbol{r})$ for which:

$$\mathcal{L}(u) = 0$$

  (for example $\nabla^2 u = 0$)
- Replace function $u(\boldsymbol{r})$ with a linear combination of basis functions:

$$\hat{u}(\boldsymbol{r}) \equiv \sum_{i=1}^{M} u_i N_i(\boldsymbol{r})$$

- Problem of finding $u(\boldsymbol{r})$ (infinite dimensions)
  $\longrightarrow$ finding finite number of u_i
- Generally, NO linear combination will yield:

$$\mathcal{L}(\hat{u}) = 0$$

Finite element method

- Goal: Given a differential operator $\mathcal{L}$, find a function $u(\boldsymbol{r})$ for which:

$$\mathcal{L}(u) = 0$$

  (for example $\nabla^2 u = 0$)
- Replace function $u(\boldsymbol{r})$ with a linear combination of basis functions:

$$\hat{u}(\boldsymbol{r}) \equiv \sum_{i=1}^{M} u_i N_i(\boldsymbol{r})$$

- Problem of finding $u(\boldsymbol{r})$ (infinite dimensions)
  $\longrightarrow$ finding finite number of u_i
- Generally, NO linear combination will yield:

$$\mathcal{L}(\hat{u}) = 0$$

- Instead, we minimize value $\|\mathcal{L}(\hat{u})\|$ — residuum

Minimizing the residuumm

- Minimize $\|\mathcal{L}(\hat{u})\|$ — what is $\|\cdot\|$?

Minimizing the residuumm

- Minimize $\|\mathcal{L}(\hat{u})\|$ — what is $\|\cdot\|$?
- Least squares: minimize the square of $\mathcal{L}(\hat{u})$

$$\frac{\partial}{\partial u_j} \int \mathcal{L}^2(\hat{u}) \, \mathrm{d}\Omega = 0 \qquad \forall j = 1, ..., M$$

Minimizing the residuumm

- Minimize $\|\mathcal{L}(\hat{u})\|$ — what is $\|\cdot\|$?
- Least squares: minimize the square of $\mathcal{L}(\hat{u})$

$$\frac{\partial}{\partial u_j} \int \mathcal{L}^2(\hat{u}) \, \mathrm{d}\Omega = 0 \qquad \forall j = 1, ..., M$$

- Method of weighted residuals — minimizing by solving set of equations:

$$\int \mathcal{L}(\hat{u}) W_j \, \mathrm{d}\Omega = 0 \qquad \forall j = 1, ..., M$$

where W_j are weighting (test) functions (weak formulation)

Minimizing the residuumm

- Minimize $\|\mathcal{L}(\hat{u})\|$ — what is $\|\cdot\|$?
- Least squares: minimize the square of $\mathcal{L}(\hat{u})$

$$\frac{\partial}{\partial u_j} \int \mathcal{L}^2(\hat{u}) \, \mathrm{d}\Omega = 0 \qquad \forall j = 1, ..., M$$

- Method of weighted residuals — minimizing by solving set of equations:

$$\int \mathcal{L}(\hat{u}) W_j \, \mathrm{d}\Omega = 0 \qquad \forall j = 1, ..., M$$

  where W_j are weighting (test) functions (weak formulation)
- Using $W_i = N_i \longrightarrow$ Galerkin method

Minimizing the residuumm

- Solving the set of equations $\int \mathcal{L}(\hat{u}) W_j \, \mathrm{d}\Omega = 0$ is problem-specific
- We need to get

$$a(\hat{u}, N_j) = b(N_j)$$

where a is a bilinear form, b is a linear form

Minimizing the residuumm

- Solving the set of equations $\int \mathcal{L}(\hat{u}) W_j \, \mathrm{d}\Omega = 0$ is problem-specific
- We need to get

$$a(\hat{u}, N_j) = b(N_j)$$

where a is a bilinear form, b is a linear form

- Then

$$\sum_i a(N_i, N_j) u_i = b(N_j)$$

i.e. solve a linear system

Example: Poisson equation

Solve one-dimensional equation:

$$\mathcal{L}(u) = u''(x) + f(x) = 0$$

with boundary condition $u(0) = u(1) = 0$

Example: Poisson equation

Solve one-dimensional equation:

$$\mathcal{L}(u) = u''(x) + f(x) = 0$$

with boundary condition $u(0) = u(1) = 0$

- Galerkin method:

$$\sum_j u_j \int N_j''(x) N_i(x)\,\mathrm{d}x = -\int f(x) N_i(x)\,\mathrm{d}x$$

Example: Poisson equation

Solve one-dimensional equation:

$$\mathcal{L}(u) = u''(x) + f(x) = 0$$

with boundary condition $u(0) = u(1) = 0$

- Galerkin method:

$$\sum_j u_j \int N_j''(x) N_i(x)\,\mathrm{d}x = -\int f(x) N_i(x)\,\mathrm{d}x$$

- Choose $N_i(x) = \sin i\pi x \longrightarrow N_i''(x) = -i^2\pi^2 N_i(x)$
  - Automatically satisfies the boundary conditions
  - Basis functions orthogonal $\longrightarrow$ leads to diagonal matrix

Example: Poisson equation

- Integral on LHS can be computed:

$$\int N_j''(x) N_i(x)\,\mathrm{d}x = -\frac{j^2\pi^2}{2}\delta_{ij}$$

Example: Poisson equation

- Integral on LHS can be computed:

$$\int N_j''(x) N_i(x)\,\mathrm{d}x = -\frac{j^2\pi^2}{2}\delta_{ij}$$

- We get the solution:

$$u_i = \frac{2}{i^2\pi^2}\int f(x)\sin i\pi x\,\mathrm{d}x$$

Time dependence

- Use finite differences for temporal integration
- Make coefficients u_i functions of time

$$\hat{u}(\boldsymbol{r}, t) = \sum_{i=1}^{M} u_i(t) N_i(\boldsymbol{r})$$

- We already solve a matrix problem
  — implicit time stepping "for free"

Basis functions

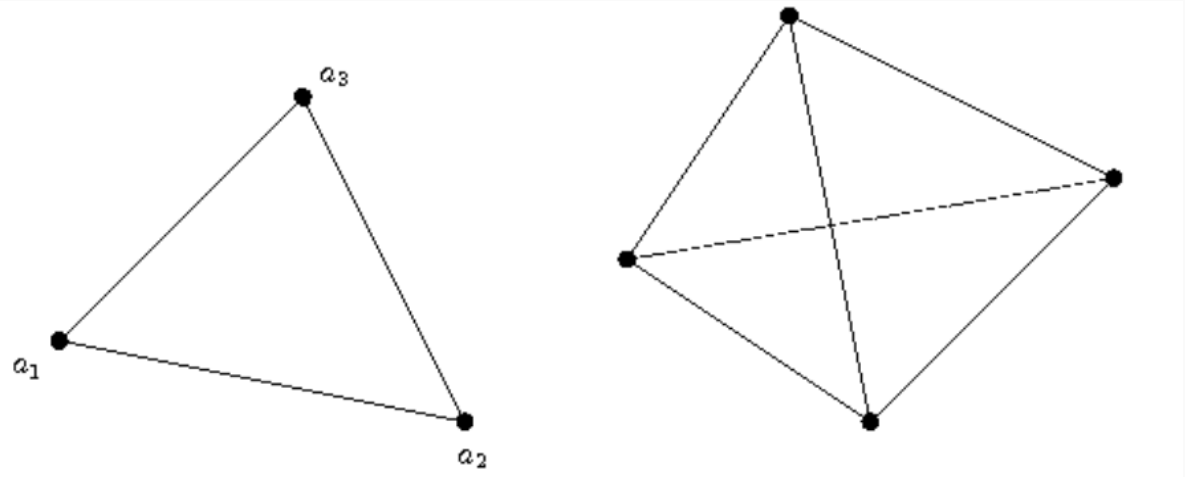
- Connected to mesh elements (vertices, edges, ...)
- Typically piecewise polynomial functions — Lagrange elements

Basis functions

- Connected to mesh elements (vertices, edges, ...)
- Typically piecewise polynomial functions — Lagrange elements
- Piecewise constant $\mathbf{P}^0$ — associated with the barycenter of the triangle (tetrahedron)

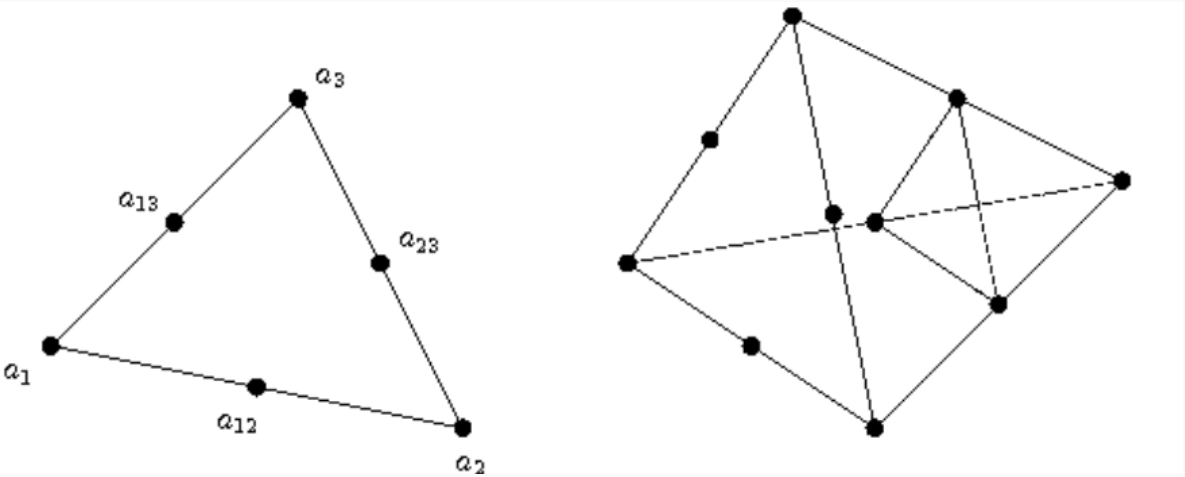
Basis functions

- Connected to mesh elements (vertices, edges, ...)
- Typically piecewise polynomial functions — Lagrange elements
- Piecewise constant $\mathbf{P}^0$ — associated with the barycenter of the triangle (tetrahedron)
- Piecewise linear $\mathbf{P}^1$ — associated with mesh vertices



Basis functions

- Piecewise quadratic $\mathbf{P}^2$ — associated with vertices and edge midpoints



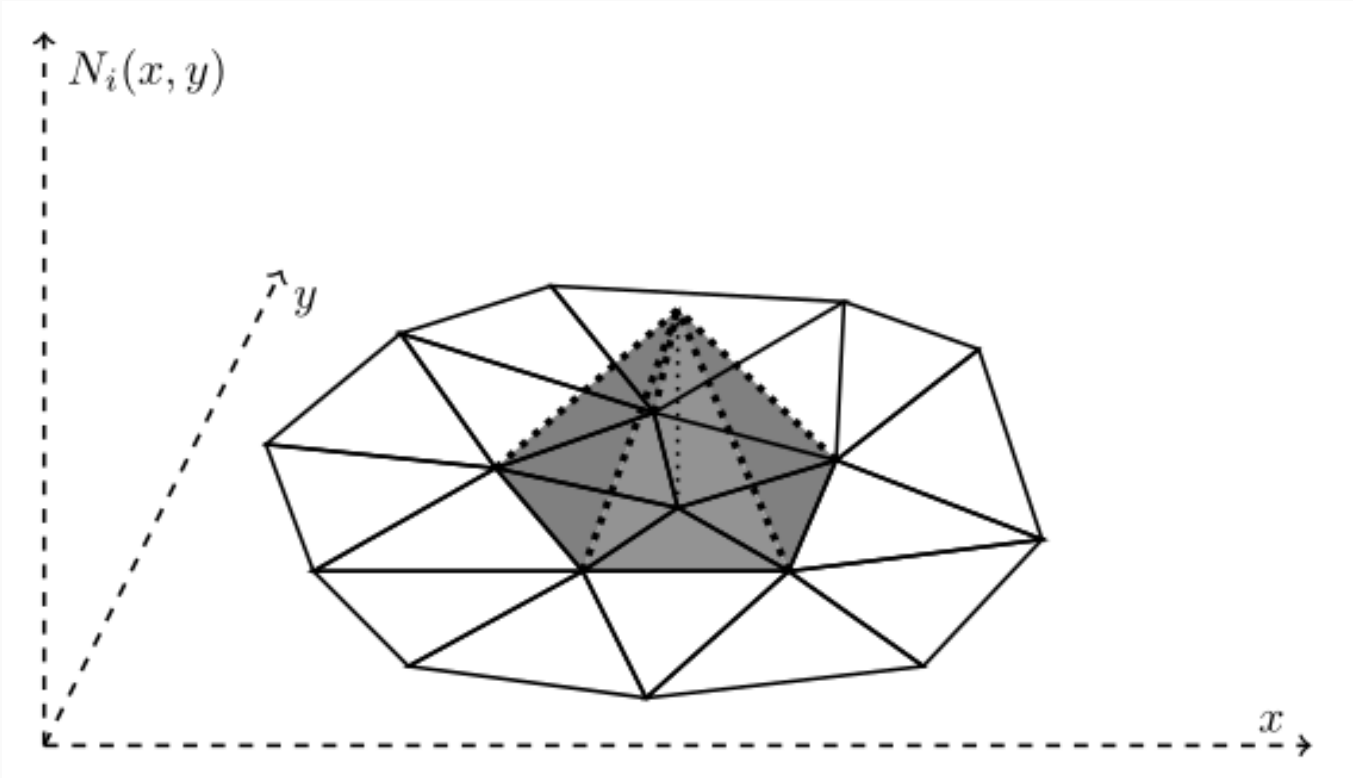
- ... and many others (`FreeFem++` has $\sim$ 35 different elements)
- Only non-zero in neighborhood $\longrightarrow$ leads to sparse matrix

Basis functions

- Basis function $N(x, y)$: sum of shape functions

$$\phi_i(x, y) = a_i + b_i x + c_i y \text{ for } (x, y) \in \text{triangle}(i)$$
$$= 0 \text{ elsewhere}$$

- Single $\mathbf{P^1}$ basis function:



Computing P^1 functions

- 2D triangle with vertices (x_i, y_i)
- Shape function associated with vertex j:

$$\phi_j(x, y) = a_j + b_j x + c_j$$

Computing P^1 functions

- 2D triangle with vertices (x_i, y_i)
- Shape function associated with vertex j:

$$\phi_j(x, y) = a_j + b_j x + c_j$$

- Coefficients a, b, c determined from constraint:

$$\phi_j(x_i, y_i) = \delta_{ij}$$

Computing P^1 functions

- 2D triangle with vertices (x_i, y_i)
- Shape function associated with vertex j:

$$\phi_j(x, y) = a_j + b_j x + c_j$$

- Coefficients a, b, c determined from constraint:

$$\phi_j(x_i, y_i) = \delta_{ij}$$

- Rewriting to a matrix problem:

$$\begin{pmatrix} 1 & x_j & y_j \\ 1 & x_k & y_k \\ 1 & x_l & y_l \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Computing matrices

Applied on diffusion problem:

$$\frac{\partial u}{\partial t} = c\nabla^2 u$$

- Weak formulation:

$$\int \frac{\partial u}{\partial t} N_i \mathrm{d}\Omega = c \int \nabla^2 u N_i \mathrm{d}\Omega$$

Computing matrices

Applied on diffusion problem:

$$\frac{\partial u}{\partial t} = c\nabla^2 u$$

- Weak formulation:

$$\int \frac{\partial u}{\partial t} N_i \mathrm{d}\Omega = c \int \nabla^2 u N_i \mathrm{d}\Omega$$

- Explicit time integration — subtitute:

$$\nabla^2 u = \sum_j u_j^0 \nabla^2 N_j$$

$$\partial u/\partial t = (u_j - u_j^0)/\Delta t$$

where $\boldsymbol{u}^0$ is solution from previous time step (known values). Then:

$$\sum_j \frac{u_j - u_0}{\Delta t} \int N_i N_j \mathrm{d}\Omega = c \sum_j u_j^0 \int \nabla^2 N_i N_j \mathrm{d}\Omega$$

Computing matrices

- Assuming $\nabla u = 0$ at $\partial\Omega$ — apply divergence theorem:

$$\sum_j \frac{u_j - u_0}{\Delta t} \int N_i N_j \mathrm{d}\Omega = -c \sum_j u_j^0 \int \nabla N_i \cdot \nabla N_j \mathrm{d}\Omega$$

Computing matrices

- Assuming $\nabla u = 0$ at $\partial\Omega$ — apply divergence theorem:

$$\sum_j \frac{u_j - u_0}{\Delta t} \int N_i N_j \mathrm{d}\Omega = -c \sum_j u_j^0 \int \nabla N_i \cdot \nabla N_j \mathrm{d}\Omega$$

- Integral on LHS — mass matrix:

$$M_{ij} = \int N_i N_j \, \mathrm{d}\Omega$$

Computing matrices

- Assuming $\nabla u = 0$ at $\partial\Omega$ — apply divergence theorem:

$$\sum_j \frac{u_j - u_0}{\Delta t} \int N_i N_j \mathrm{d}\Omega = -c \sum_j u_j^0 \int \nabla N_i \cdot \nabla N_j \mathrm{d}\Omega$$

- Integral on LHS — mass matrix:

$$M_{ij} = \int N_i N_j \, \mathrm{d}\Omega$$

- Integral on RHS — stiffness matrix:

$$K_{ij} = \int \nabla N_i \cdot \nabla N_j \, \mathrm{d}\Omega$$

Computing matrices

- Assuming $\nabla u = 0$ at $\partial\Omega$ — apply divergence theorem:

$$\sum_j \frac{u_j - u_0}{\Delta t} \int N_i N_j \mathrm{d}\Omega = -c \sum_j u_j^0 \int \nabla N_i \cdot \nabla N_j \mathrm{d}\Omega$$

- Integral on LHS — mass matrix:

$$M_{ij} = \int N_i N_j \, \mathrm{d}\Omega$$

- Integral on RHS — stiffness matrix:

$$K_{ij} = \int \nabla N_i \cdot \nabla N_j \, \mathrm{d}\Omega$$

- We get:

$$\boldsymbol{M} \frac{\boldsymbol{u} - \boldsymbol{u}^0}{\Delta t} = -c \boldsymbol{K} \boldsymbol{u}_0$$

Mass lumping

- Both $\boldsymbol{M}$ and $\boldsymbol{K}$ depend only on the domain subdivision and selected basis function
- For piecewise linear elements $\mathbf{P}^1$, ∇N_i is piecewise constant
  $\longrightarrow$ integral $\boldsymbol{K}$ is reduced to the area of triangle

Mass lumping

- Both $\boldsymbol{M}$ and $\boldsymbol{K}$ depend only on the domain subdivision and selected basis function
- For piecewise linear elements $\mathbf{P}^1$, ∇N_i is piecewise constant
  $\longrightarrow$ integral $\boldsymbol{K}$ is reduced to the area of triangle
- We get:

$$\boldsymbol{M}\frac{\boldsymbol{u}}{\Delta t} = -c\boldsymbol{K}\boldsymbol{u}^0 + \boldsymbol{M}\frac{\boldsymbol{u}^0}{\Delta t}$$

- RHS — known values
- LHS — requires $\boldsymbol{M}^{-1}$
  $\longrightarrow$ can be further simplified using mass lumping (diagonalization)

Mass lumping

- Diagonalization of $\boldsymbol{M}$ — tradeoff between precision and performance/robustness of the solver
- Row sum method:

$$\tilde{M}_{ii} = \sum_j M_{ij}$$

- Diagonal scaling:

$$\tilde{M}_{ii} = fM_{ii}$$

Non-linear equations

Goal: reduce the discretized equation into:

$$\sum_i a(N_i, N_j) u_i = b(N_j)$$

or in matrix form:

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$$

Non-linear equations

Goal: reduce the discretized equation into:

$$\sum_i a(N_i, N_j) u_i = b(N_j)$$

or in matrix form:

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$$

i For non-linear equations, we get e.g.:

$$\boldsymbol{A}(\boldsymbol{u})\boldsymbol{u} = \boldsymbol{b}$$

$\longrightarrow$ Picard iteration method

Picard iterations

Replace $\boldsymbol{A}(\boldsymbol{u})\boldsymbol{u} = \boldsymbol{b}$ with $\boldsymbol{A}(\boldsymbol{u}_0)\boldsymbol{u} = \boldsymbol{b}$, using a guess $\boldsymbol{u}_0$.
Then:

1. Solve the linear problem $\longrightarrow$ yields solution $\boldsymbol{u}_1$
2. Replace $\boldsymbol{u}_0$ with the solution $\boldsymbol{u}_1$, compute new matrix $\boldsymbol{A}$
3. Solve new problem $\boldsymbol{A}(\boldsymbol{u}_k)\boldsymbol{u}_{k+1} = \boldsymbol{b}$
4. Iterate until $\|\boldsymbol{u}_{k+1} - \boldsymbol{u}_k\| < \epsilon$

Picard iterations

Replace $\boldsymbol{A}(\boldsymbol{u})\boldsymbol{u} = \boldsymbol{b}$ with $\boldsymbol{A}(\boldsymbol{u}_0)\boldsymbol{u} = \boldsymbol{b}$, using a guess $\boldsymbol{u}_0$.
Then:

1. Solve the linear problem $\longrightarrow$ yields solution $\boldsymbol{u}_1$
2. Replace $\boldsymbol{u}_0$ with the solution $\boldsymbol{u}_1$, compute new matrix $\boldsymbol{A}$
3. Solve new problem $\boldsymbol{A}(\boldsymbol{u}_k)\boldsymbol{u}_{k+1} = \boldsymbol{b}$
4. Iterate until $\|\boldsymbol{u}_{k+1} - \boldsymbol{u}_k\| < \epsilon$

Will it always converge?

Picard iterations

Replace $\boldsymbol{A}(\boldsymbol{u})\boldsymbol{u} = \boldsymbol{b}$ with $\boldsymbol{A}(\boldsymbol{u}_0)\boldsymbol{u} = \boldsymbol{b}$, using a guess $\boldsymbol{u}_0$.
Then:

1. Solve the linear problem $\longrightarrow$ yields solution $\boldsymbol{u}_1$
2. Replace $\boldsymbol{u}_0$ with the solution $\boldsymbol{u}_1$, compute new matrix $\boldsymbol{A}$
3. Solve new problem $\boldsymbol{A}(\boldsymbol{u}_k)\boldsymbol{u}_{k+1} = \boldsymbol{b}$
4. Iterate until $\|\boldsymbol{u}_{k+1} - \boldsymbol{u}_k\| < \epsilon$

Will it always converge? $\longrightarrow$ NO

Relaxation method

We can improve the convergence using relaxation method, i.e. weight new and previous solution:

$$\boldsymbol{A}(\boldsymbol{u}_k)\boldsymbol{u}^\star = \boldsymbol{b}$$

$$\boldsymbol{u}_{k+1} := \omega \boldsymbol{u}^\star + (1-\omega)\boldsymbol{u}_k$$

where ω is the relaxation parameter.

Relaxation method

We can improve the convergence using relaxation method,
i.e. weight new and previous solution:

$$\boldsymbol{A}(\boldsymbol{u}_k)\boldsymbol{u}^\star = \boldsymbol{b}$$

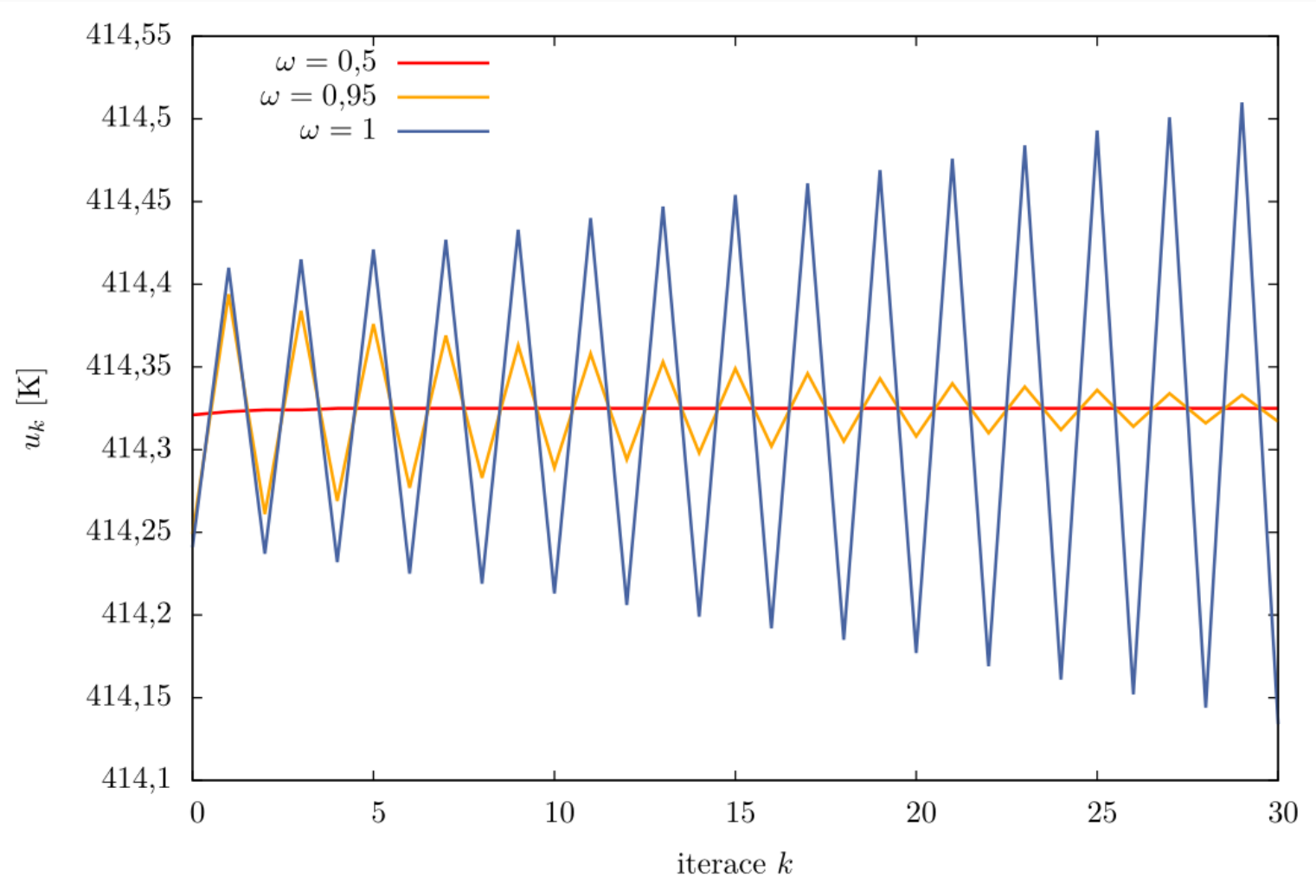
$$\boldsymbol{u}_{k+1} := \omega \boldsymbol{u}^\star + (1-\omega)\boldsymbol{u}_k$$

where ω is the relaxation parameter.

- Essentially decreases the iteration step
- General method, usable outside FEM
- In some cases can be used with $\omega > 1$ to speed up the convergence
  $\longrightarrow$ over-relaxation

Relaxation method

Example: linearization of the radiative term $\propto u^4$,
i.e. iterative solution of $\boldsymbol{A}(\boldsymbol{u}_k^3)\boldsymbol{u}_{k+1} = \boldsymbol{b}$



Boundary conditions

- Dirichlet or Neumann

Boundary conditions

- Dirichlet or Neumann
- Essential boundary conditions — explicitly imposed on the solution:

$$\int_{\Omega} \mathcal{L}(\hat{u}) W_j \, \mathrm{d}\Omega + \sum_{k=1}^{n} \oint_{\Gamma_k} \mathcal{S}_k(\hat{u}) W_j^k \, \mathrm{d}\Gamma = 0$$

  for the k-th boundary condition

Boundary conditions

- Dirichlet or Neumann
- Essential boundary conditions — explicitly imposed on the solution:

$$\int_{\Omega} \mathcal{L}(\hat{u}) W_j \, \mathrm{d}\Omega + \sum_{k=1}^{n} \oint_{\Gamma_k} \mathcal{S}_k(\hat{u}) W_j^k \, \mathrm{d}\Gamma = 0$$

  for the k-th boundary condition
- Natural boundary conditions — embedded into the equations, satisfied automatically when finding the solution

Boundary conditions

- Dirichlet or Neumann
- Essential boundary conditions — explicitly imposed on the solution:

$$\int_{\Omega} \mathcal{L}(\hat{u}) W_j \, \mathrm{d}\Omega + \sum_{k=1}^{n} \oint_{\Gamma_k} \mathcal{S}_k(\hat{u}) W_j^k \, \mathrm{d}\Gamma = 0$$

  for the k-th boundary condition
- Natural boundary conditions — embedded into the equations, satisfied automatically when finding the solution
- Usually Dirichlet $\sim$ essential and Neumann $\sim$ natural (but not always)

Natural boundary conditions

Example: diffusion equation in domain Ω

$$\frac{\partial u}{\partial t} = c\nabla^2 u$$

with boundary condition at $\partial\Omega$:

$$\boldsymbol{n} \cdot \nabla u = f$$

Natural boundary conditions

Example: diffusion equation in domain Ω

$$\frac{\partial u}{\partial t} = c\nabla^2 u$$

with boundary condition at $\partial\Omega$:

$$\boldsymbol{n} \cdot \nabla u = f$$

We thus solve:

$$\int_\Omega \frac{\partial u}{\partial t} N_i \mathrm{d}\Omega = c \int_\Omega \nabla^2 u N_i \mathrm{d}\Omega$$

From divergence theorem:

$$\int_\Omega \nabla^2 u N_i \mathrm{d}\Omega = \oint_{\partial\Omega} \nabla u \cdot \boldsymbol{n} N_i \mathrm{d}\Sigma - \int_\Omega \nabla u \cdot \nabla N_i \mathrm{d}\Omega$$

Natural boundary conditions

Plug our BC into the surface term

$$\int_\Omega \frac{\partial u}{\partial t} N_i \mathrm{d}\Omega = c \left(\oint_{\partial\Omega} f N_i \mathrm{d}\Sigma + \int_\Omega \nabla u \cdot \nabla N_i \mathrm{d}\Omega \right)$$

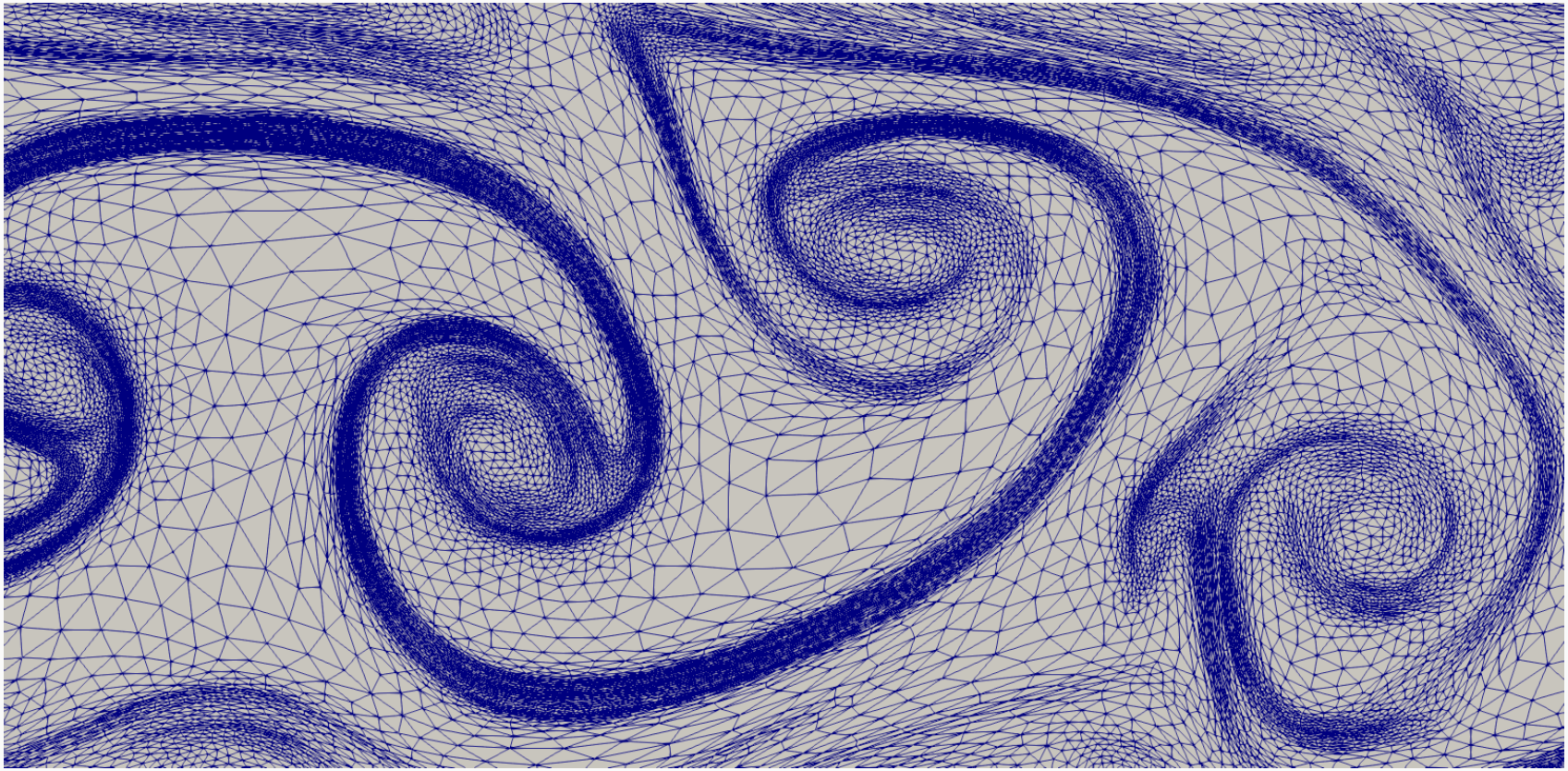
Natural boundary conditions

Plug our BC into the surface term

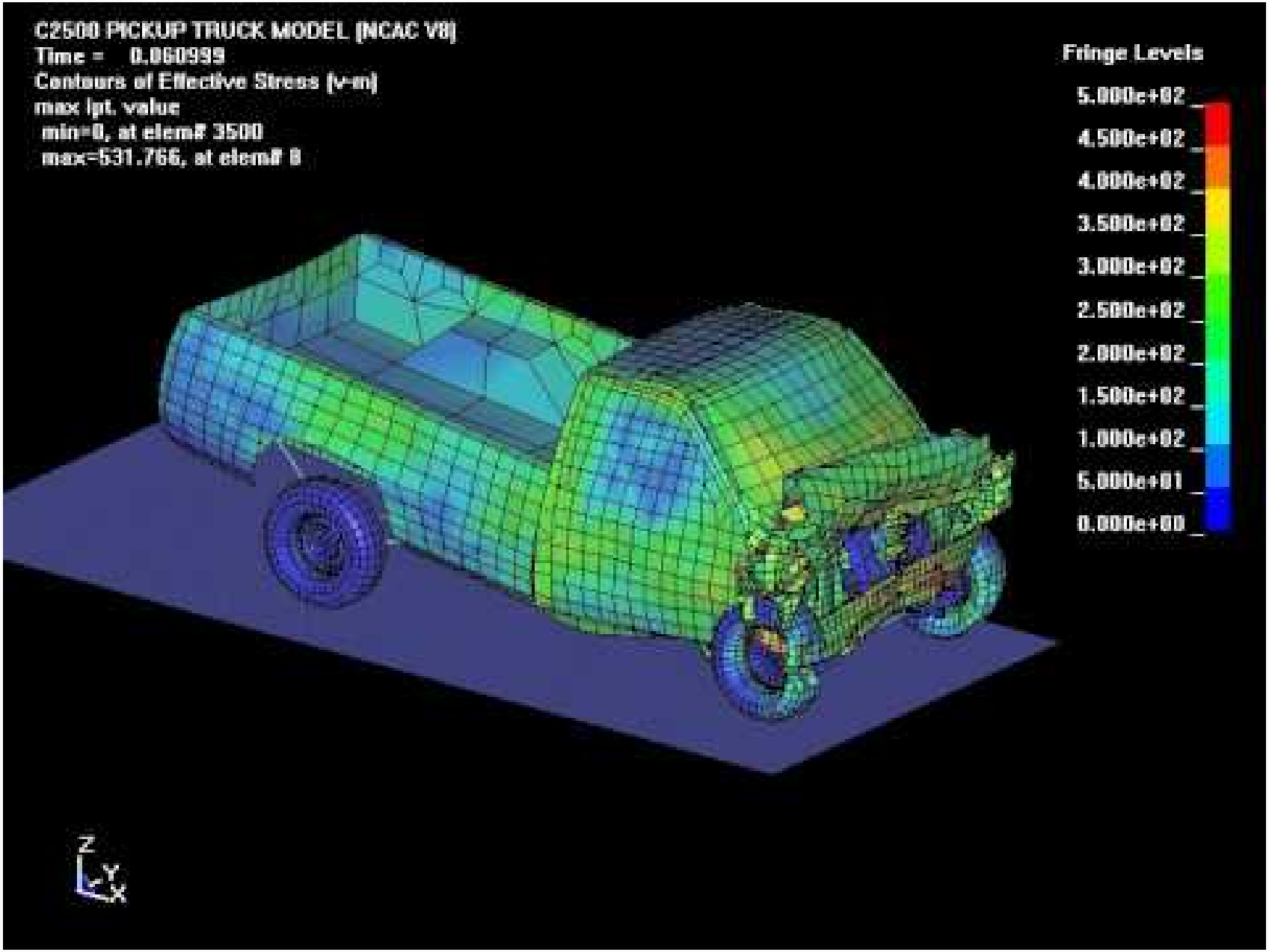
$$\int_\Omega \frac{\partial u}{\partial t} N_i \mathrm{d}\Omega = c \left(\oint_{\partial\Omega} f N_i \mathrm{d}\Sigma + \int_\Omega \nabla u \cdot \nabla N_i \mathrm{d}\Omega \right)$$

- Term $\boldsymbol{n} \cdot \nabla u$ no longer in the equation
- For $f = 0$, the equation has no surface term, yet still satisfies $\boldsymbol{n} \cdot \nabla u$ at $\partial\Omega$

Usage

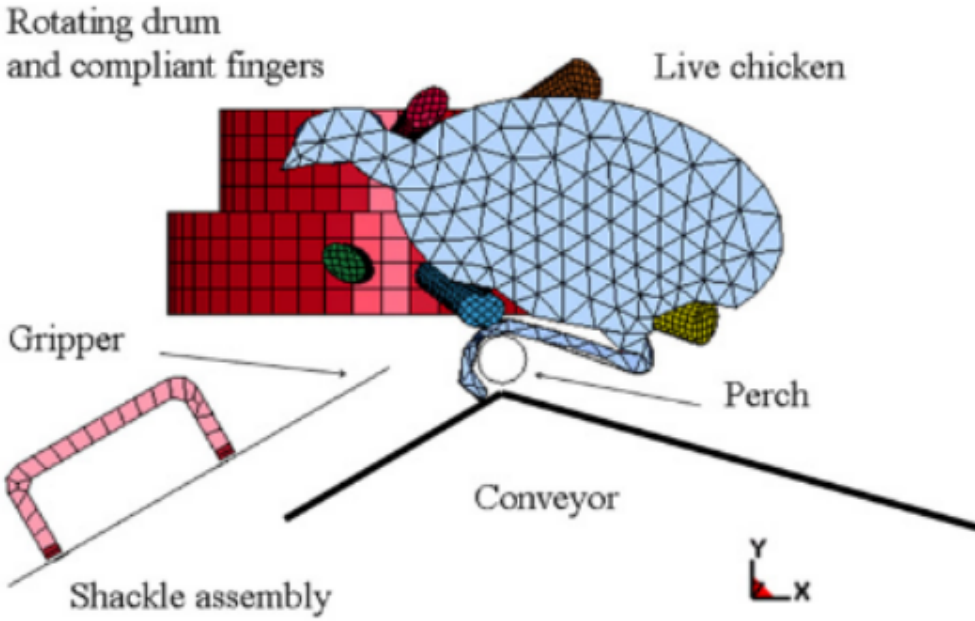


Usage


C2500 PICKUP TRUCK MODEL (NCAC V8)
Time = 0.060999
Contours of Effective Stress (v-m)
max ipt. value
min=0, at elem# 3500
max=531.766, at elem# 0
Fringe Levels
5.000e+02
4.500e+02
4.000e+02
3.500e+02
3.000e+02
2.500e+02
2.000e+02
1.500e+02
1.000e+02
5.000e+01
0.000e+00
Z
Y
X

Usage

(a) Compliant grasping [23].

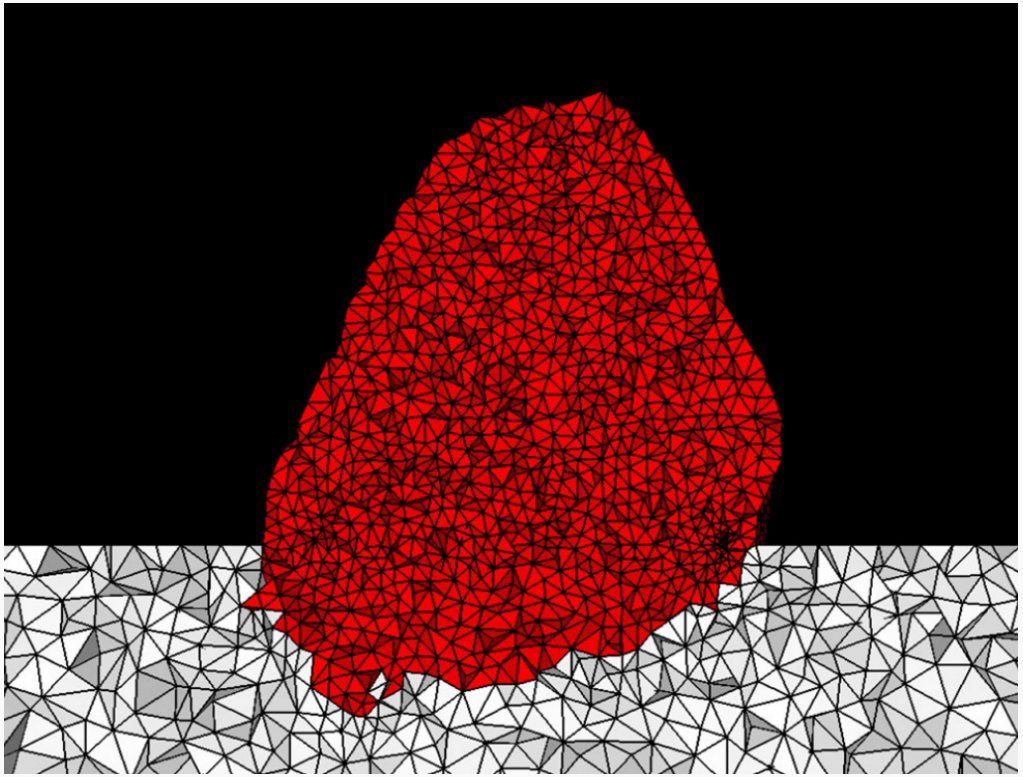


(b) Simulation scheme.

Fig. 1. Compliant grasping of a live bird transfer system (LBTS).

Usage (astrophysics)

- For spherical/cylindrical/rectangular domains, FDM or FVM preferred
- Example: Solve a heat diffusion equation in a boulder on the surface of an asteroid



FEM Codes

- libMesh
- FreeFEM++
- deal.II
- ParaFEM
- OOFEM
- ...

Example: Planar strain-stress problem

- Solve distribution of stresses inside the body
- Input:
  - external forces deforming the body
  - fixed nodes (constraints)
- Using:
  - `Eigen` library (matrix solvers)
  - `CGAL` library (mesh generation)
- Source code:
  `https://gitlab.com/sevecekp/pdesolvers/fem`

Example: Planar strain-stress problem

- Linear strain-stress relation — Hooke's law

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\epsilon}$$

  where $\boldsymbol{\sigma}$ is the stress tensor, ϵ the strain tensor.

- Planar strain $\longrightarrow$ only non-zero components are:

$$\boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{pmatrix} = \begin{pmatrix} \frac{\partial \delta_x}{\partial x} \\ \frac{\partial \delta_y}{\partial y} \\ \frac{\partial \delta_x}{\partial y} + \frac{\partial \delta_y}{\partial x} \end{pmatrix}$$

  where $\boldsymbol{\delta}$ is the displacement vector

- Relation between displacements and external forces (loads):

$$\boldsymbol{K}\boldsymbol{\delta} = \boldsymbol{F}$$

  where $\boldsymbol{F}$ is the vector of forces, $\boldsymbol{K}$ is the stiffness matrix

Example: Planar strain-stress problem

- Basis functions $\longrightarrow$ used for interpolation of displacements

$$\delta(x, y) = a_1 + a_2 x + a_3 y$$

- Displacement values at nodes (x_i, y_i), (x_j, y_j), (x_k, y_k):

$$\delta_i = a_1 + a_2 x_i + a_3 y_i$$
$$\delta_j = a_1 + a_2 x_j + a_3 y_j$$
$$\delta_k = a_1 + a_2 x_k + a_3 y_k$$

- Inverting the matrix ($\boldsymbol{C}$):

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{pmatrix}^{-1} \begin{pmatrix} \delta_i \\ \delta_j \\ \delta_k \end{pmatrix}$$

Example: Planar strain-stress problem

- Shape functions:

$$\begin{pmatrix} N_i & N_j & N_k \end{pmatrix} = \begin{pmatrix} 1 & x & y \end{pmatrix} \boldsymbol{C}^{-1}$$

- Using Ns, we can compute strain ϵ:

$$\begin{pmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{pmatrix} = \begin{pmatrix} \frac{\partial N_i}{\partial x} & 0 & \frac{\partial N_j}{\partial x} & 0 & \frac{\partial N_k}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 & \frac{\partial N_j}{\partial y} & 0 & \frac{\partial N_k}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & \frac{\partial N_j}{\partial y} & \frac{\partial N_j}{\partial x} & \frac{\partial N_k}{\partial y} & \frac{\partial N_k}{\partial x} \end{pmatrix} \begin{pmatrix} \delta_i^x \\ \delta_i^y \\ \delta_j^x \\ \delta_j^y \\ \delta_k^x \\ \delta_k^y \end{pmatrix}$$

- Denoting the RHS matrix as $\boldsymbol{B}$:

$$\boldsymbol{\epsilon} = \boldsymbol{B}\boldsymbol{\delta}_e$$

where $\boldsymbol{\delta}_e$ are displacement components

Example: Planar strain-stress problem

- Stress $\boldsymbol{\sigma}$:

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{B}\boldsymbol{\delta}_e$$

- Relation between the displacements and forces — use the principle of virtual work
- Infinitesimal displacements $\mathbf{d}\boldsymbol{\delta}$ — compute the work $\mathrm{d}A$

$$\mathrm{d}A = \mathbf{d}\boldsymbol{\epsilon} \cdot \boldsymbol{\sigma} = \mathbf{d}\boldsymbol{\delta}\boldsymbol{B}^T\boldsymbol{\sigma}$$

- Plugging in the $\boldsymbol{\sigma}$:

$$\mathrm{d}A = \mathbf{d}\boldsymbol{\delta}_e\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}\boldsymbol{\delta}_e$$

- Virtual work of external forces = total work of internal stresses, hence:

$$\mathbf{d}\boldsymbol{\delta}_e \cdot \boldsymbol{F} = \int_e \mathbf{d}\boldsymbol{\delta}_e\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}\boldsymbol{\delta}_e\mathrm{d}V$$

Example: Planar strain-stress problem

- Equation holds for any displacement $\mathbf{d}\boldsymbol{\delta}_e$:

$$\boldsymbol{F} = \int_e \boldsymbol{B}^T \boldsymbol{D} \boldsymbol{B} \boldsymbol{\delta}_e \mathrm{d}V$$

- Nodal displacements $\boldsymbol{\delta}_e$ constant
  $\longrightarrow$ integral yields the stiffness matrix:

$$\boldsymbol{K} = \int_e \boldsymbol{B}^T \boldsymbol{D} \boldsymbol{B} \mathrm{d}V$$

- Homogeneous material $\longrightarrow$ integral reduces to:

$$\boldsymbol{K} = \boldsymbol{B}^T \boldsymbol{D} \boldsymbol{B} \frac{\det \boldsymbol{C}}{2}$$

- Once $\boldsymbol{K}$ is computed, solve:

$$\boldsymbol{K} \boldsymbol{\delta} = \boldsymbol{F}$$

Example: Planar strain-stress problem

- Problem ill-posed without contraints — whole body could be displaced
- The simplest contraint — stationary node
- Apply constraint on node $i \longrightarrow$ set force $\boldsymbol{F}_i = 0$ and elements of the stiffness matrix:

$$K_{ij} = \delta_{ij}$$

$$K_{ji} = \delta_{ij}$$

Smoothed particle hydrodynamics

- Mainly used for solution of hydrodynamical equations

Smoothed particle hydrodynamics

- Mainly used for solution of hydrodynamical equations
- Lagrangian method!

Smoothed particle hydrodynamics

- Mainly used for solution of hydrodynamical equations
- Lagrangian method!
- Grid-less, no computational domain
  $\longrightarrow$ suitable for problems where the domain is not *a priori* known

Smoothed particle hydrodynamics

- Mainly used for solution of hydrodynamical equations
- Lagrangian method!
- Grid-less, no computational domain
  $\longrightarrow$ suitable for problems where the domain is not *a priori* known
- Automatic adapative spatial resolution

Smoothed particle hydrodynamics

- Mainly used for solution of hydrodynamical equations
- Lagrangian method!
- Grid-less, no computational domain
  $\longrightarrow$ suitable for problems where the domain is not *a priori* known
- Automatic adapative spatial resolution
- Easily extensible (not fixed to a particular problem)

Smoothed particle hydrodynamics

- Mainly used for solution of hydrodynamical equations
- Lagrangian method!
- Grid-less, no computational domain
  $\longrightarrow$ suitable for problems where the domain is not *a priori* known
- Automatic adapative spatial resolution
- Easily extensible (not fixed to a particular problem)
- Simple to implement

Smoothed particle hydrodynamics

- Mainly used for solution of hydrodynamical equations
- Lagrangian method!
- Grid-less, no computational domain
  $\longrightarrow$ suitable for problems where the domain is not *a priori* known
- Automatic adapative spatial resolution
- Easily extensible (not fixed to a particular problem)
- Simple to implement
- Generally slower compared to grid-based methods (grid has fixed topology, particle neighbors change over time)

Smoothed particle hydrodynamics

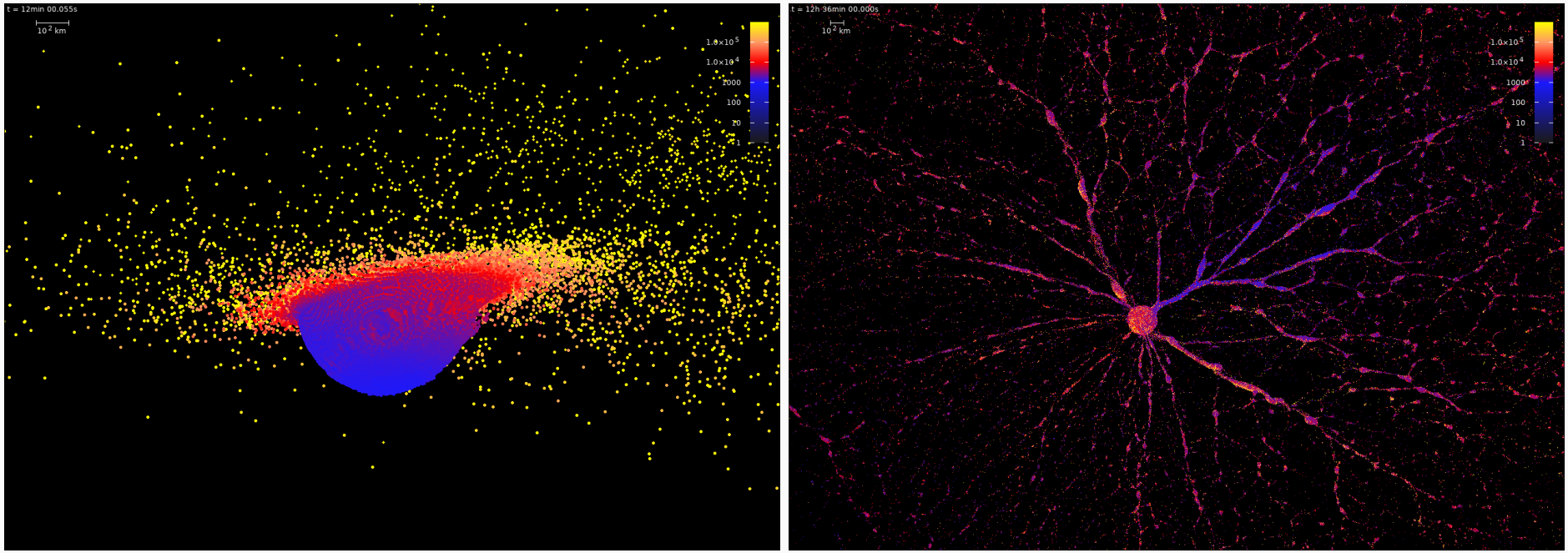
- Mainly used for solution of hydrodynamical equations
- Lagrangian method!
- Grid-less, no computational domain
  $\longrightarrow$ suitable for problems where the domain is not *a priori* known
- Automatic adapative spatial resolution
- Easily extensible (not fixed to a particular problem)
- Simple to implement
- Generally slower compared to grid-based methods (grid has fixed topology, particle neighbors change over time)
- Always gives *some* result (no convergence issues), although it may not be correct (particle interpenetration, artificial clumping, unphysical mixing of fluids, etc.)

Smoothed particle hydrodynamics

- Mainly used for solution of hydrodynamical equations
- Lagrangian method!
- Grid-less, no computational domain
  $\longrightarrow$ suitable for problems where the domain is not *a priori* known
- Automatic adapative spatial resolution
- Easily extensible (not fixed to a particular problem)
- Simple to implement
- Generally slower compared to grid-based methods (grid has fixed topology, particle neighbors change over time)
- Always gives *some* result (no convergence issues), although it may not be correct (particle interpenetration, artificial clumping, unphysical mixing of fluids, etc.)
- Comes in a lot of "flavors" (SSPH, CSPH, ASPH, GSPH, XSPH, δ-SPH, DISPH, ...)

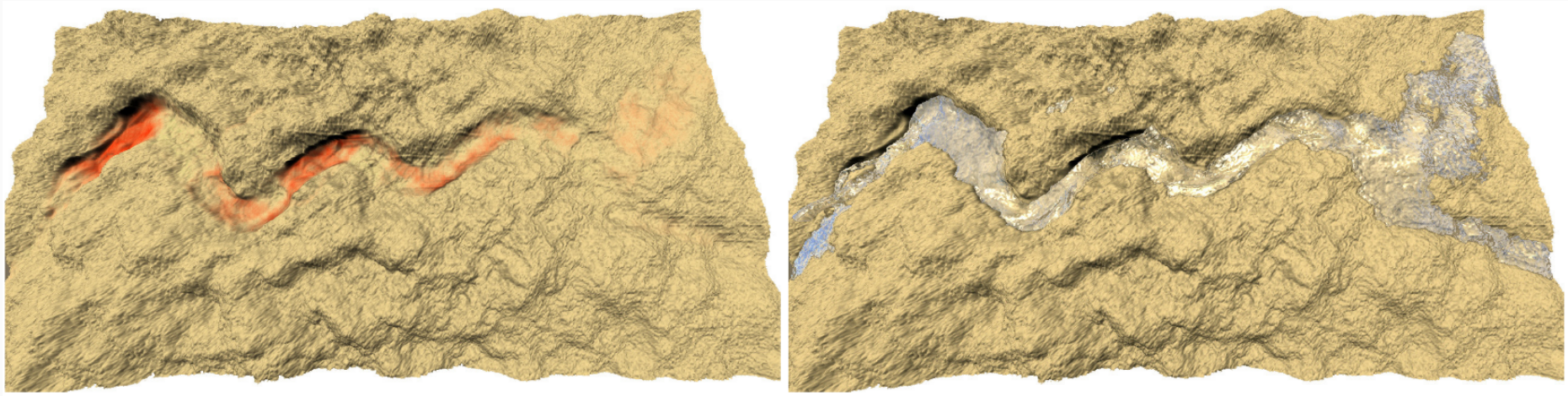
Motivation

Asteroid impact — fragments may end up “anywhere”



Motivation

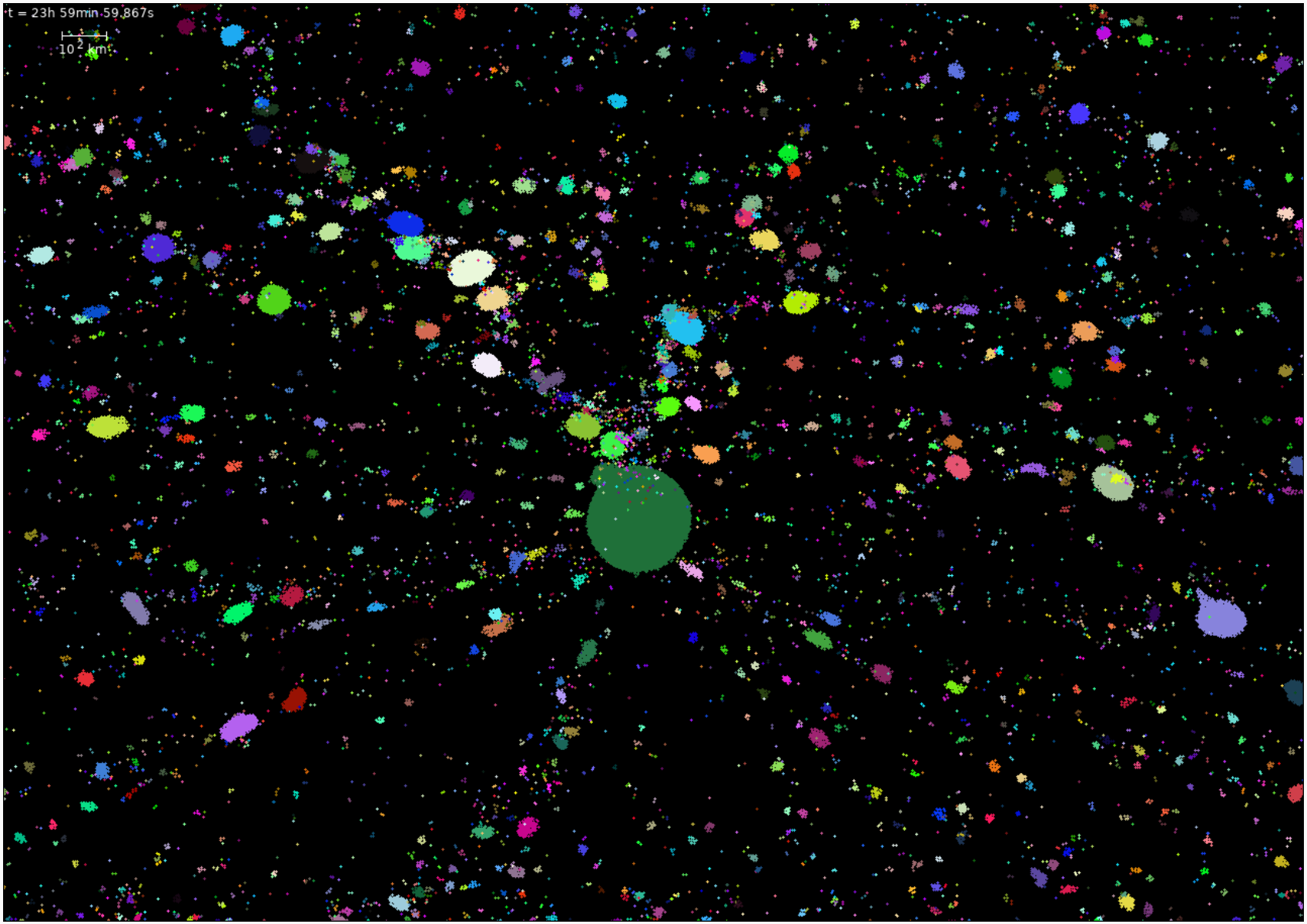
Hydraulic erosion — let the water carve the terrain



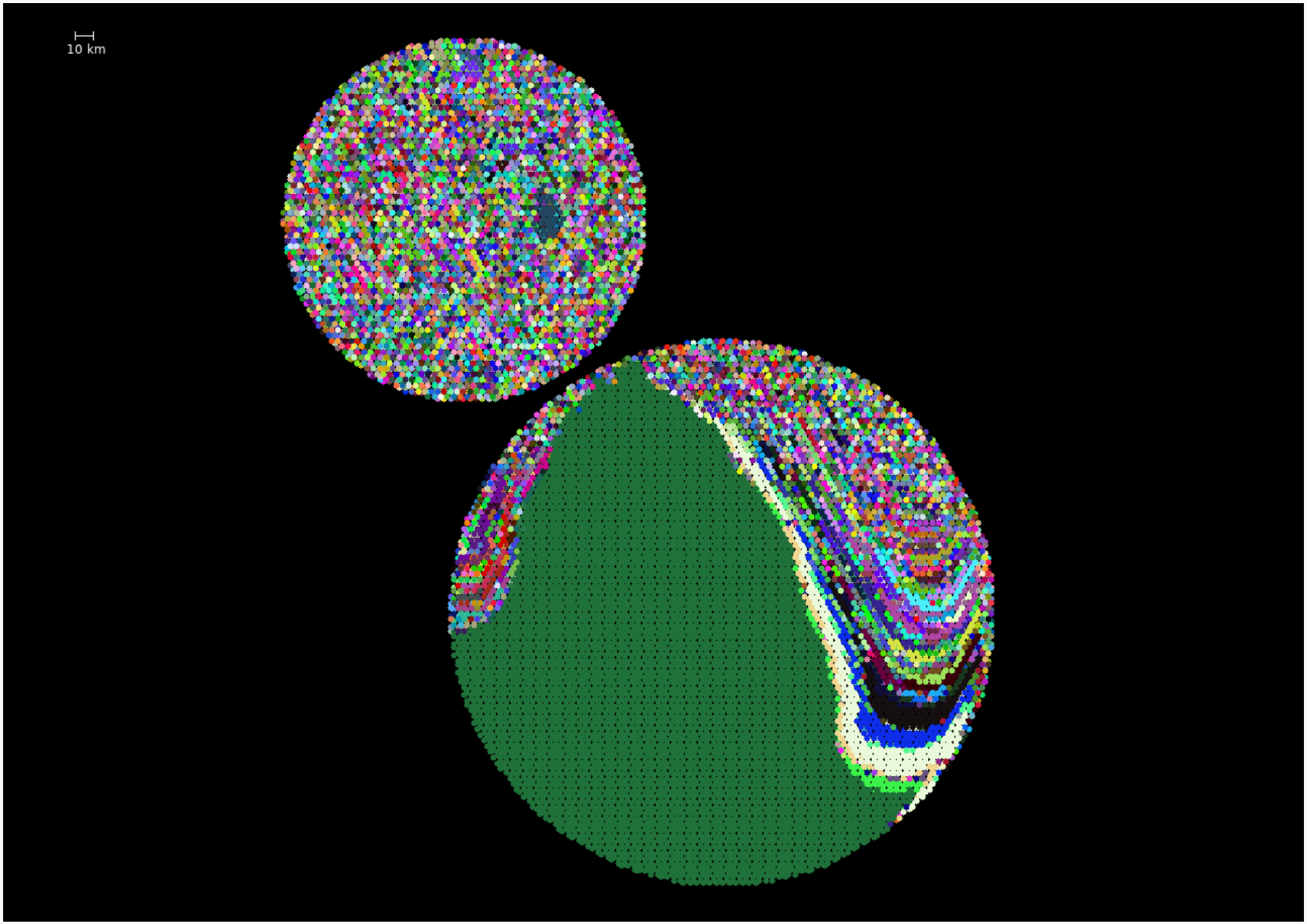
Motivation - tracers

- How does the end state relate to the initial configuration?
- Lagrangian methods: initial configuration is the reference, particles are not created nor destroyed, their "names" are fixed during simulation
- Eulerian methods: add tracer particles

Tracers



Tracers



Continuum mechanics: crash course

Eulerian description

- Describes velocities (and other quantities) at fixed points in space
- "Person standing in river measuring the velocity of the water"

Eulerian description

- Describes velocities (and other quantities) at fixed points in space
- "Person standing in river measuring the velocity of the water"
- Mapping function using immediate configuration as reference:

$$\boldsymbol{r}_0 = \xi^{-1}(\boldsymbol{r}(t), t)$$

- Material derivative:

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v} \cdot \nabla)\boldsymbol{v}$$

Continuum mechanics: crash course

Lagrangian description

- Describes velocities of pieces of continuum
- "Person on a boat measuring the velocity with GPS"

Lagrangian description

- Describes velocities of pieces of continuum
- "Person on a boat measuring the velocity with GPS"
- Mapping function using initial configuration as reference:

$$\boldsymbol{r}(t) = \xi(\boldsymbol{r}_0, t)$$

- No need for the convective derivative:

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \boldsymbol{f}$$

Continuum mechanics: crash course

Set of equations:

- Continuity equation — convervation of mass:

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} + \rho\nabla \cdot \boldsymbol{v} = 0$$

Continuum mechanics: crash course

Set of equations:

- Continuity equation — convervation of mass:

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} + \rho\nabla \cdot \boldsymbol{v} = 0$$

- Equation of motion — Euler equation:

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = -\frac{1}{\rho}\nabla P$$

$\longrightarrow$ describes adiabatic inviscid fluid

Continuum mechanics: crash course

Set of equations:

- Continuity equation — convervation of mass:

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} + \rho\nabla \cdot \boldsymbol{v} = 0$$

- Equation of motion — Euler equation:

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = -\frac{1}{\rho}\nabla P$$

  $\longrightarrow$ describes adiabatic inviscid fluid
- 5 variables, 4 equations $\longrightarrow$ Equation of state (EoS):

$$P = P(\rho)$$

  for example:

$$P = A\left(\frac{\rho}{\rho_0} - 1\right)$$

Continuum mechanics: crash course

- EoS is usually has two independent parameters
- Function of specific internal energy u:

$$P = P(\rho, u)$$

  for example ideal gas:

$$P = (\gamma - 1)u\rho$$

- Thus we need to add energy equation:

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -\frac{P}{\rho}\nabla \cdot \boldsymbol{v}$$

Continuum mechanics: crash course

- Function of specific entropy s:

$$P = K(s)\rho^\gamma$$

- Equation for the entropy function $K(s)$:

$$\frac{\mathrm{d}K}{\mathrm{d}t} = \frac{\gamma - 1}{\rho^{\gamma-1}} \left(\frac{\mathrm{d}u}{\mathrm{d}t} - \frac{P}{\rho^2} \frac{\mathrm{d}\rho}{\mathrm{d}t} \right)$$

where

$$u = \frac{K}{\gamma - 1} \rho^{\gamma-1}$$

Continuum mechanics: crash course

- Viscosity / material strength
- Navier-Stokes equation

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{1}{\rho}\nabla \cdot \boldsymbol{\sigma}$$

where

$$\boldsymbol{\sigma} = -P\boldsymbol{I} + \boldsymbol{S}$$

- Constitutive equation — linearly depend on $\nabla \boldsymbol{v}$:

$$\boldsymbol{\sigma} = \lambda(\nabla \cdot \boldsymbol{v})\boldsymbol{I} + 2\mu\dot{\boldsymbol{\epsilon}}$$

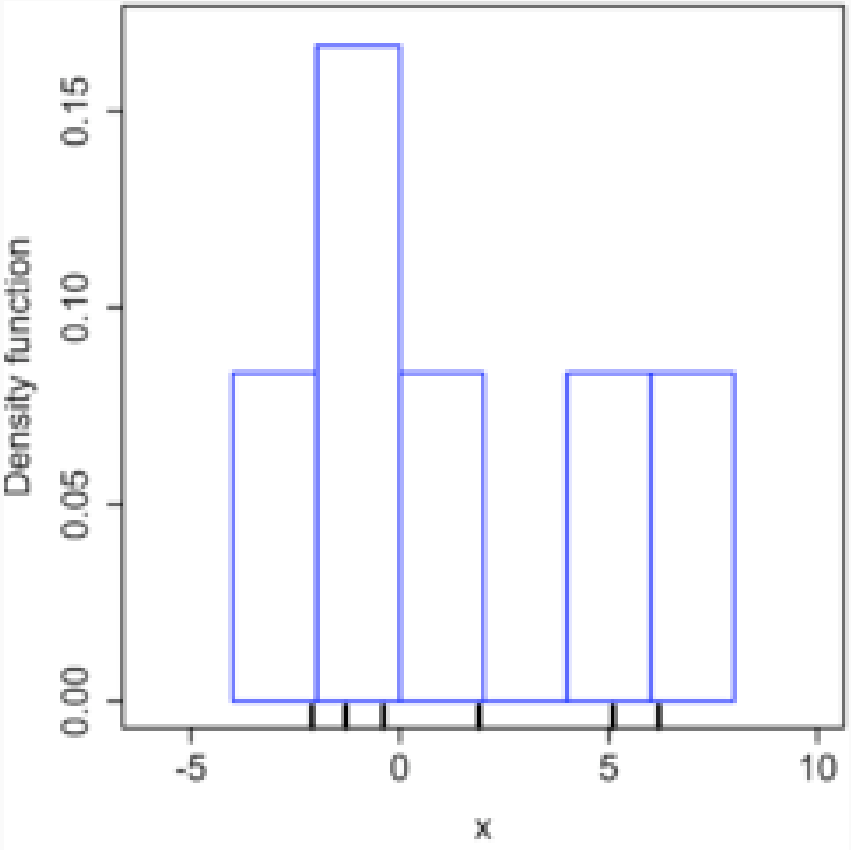
where λ and μ are Lamé parameters

Estimate probability from samples

- Motivation: We have set of samples from (unknown) probability distribition P
- Goal is to estimate P

Estimate probability from samples

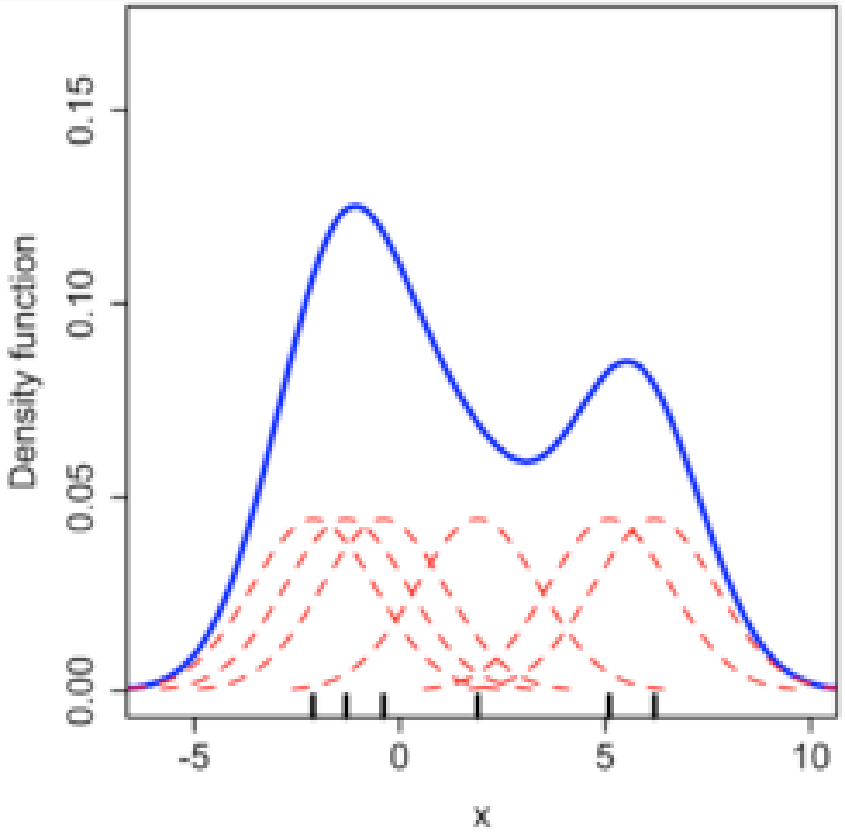
- Motivation: We have set of samples from (unknown) probability distribition P
- Goal is to estimate P
- Approach 1: construct a histogram



Estimate probability from samples

- Approach 2: estimate P as a sum of kernel functions placed at sample points, i.e.:

$$P(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$



Kernel density estimate

- Called kernel density estimate (Parzen window)
- Provides continuous (even C^∞) function
- Bandwidth h — controlls precision vs. resolution
  (similarly to the bin size in histogram)
- Used frequently in statistics, signal processing, computer vision, ...

SPH — basic idea

- Continuum representent by a set of particles
- Each particle has a fixed mass m_i

SPH — basic idea

- Continuum representent by a set of particles
- Each particle has a fixed mass m_i
- Density — similar to kernel density estimate:

$$\rho(\boldsymbol{r}) = \sum_i m_i W(\boldsymbol{r} - \boldsymbol{r}_i, h_i)$$

where W is the kernel function and h is the smoothing length

SPH — basic idea

- Continuum representent by a set of particles
- Each particle has a fixed mass m_i
- Density — similar to kernel density estimate:

$$\rho(\boldsymbol{r}) = \sum_i m_i W(\boldsymbol{r} - \boldsymbol{r}_i, h_i)$$

  where W is the kernel function and h is the smoothing length
- Smoothing kernel $W(\boldsymbol{r}, h)$ — known function, represents a density profile of particles

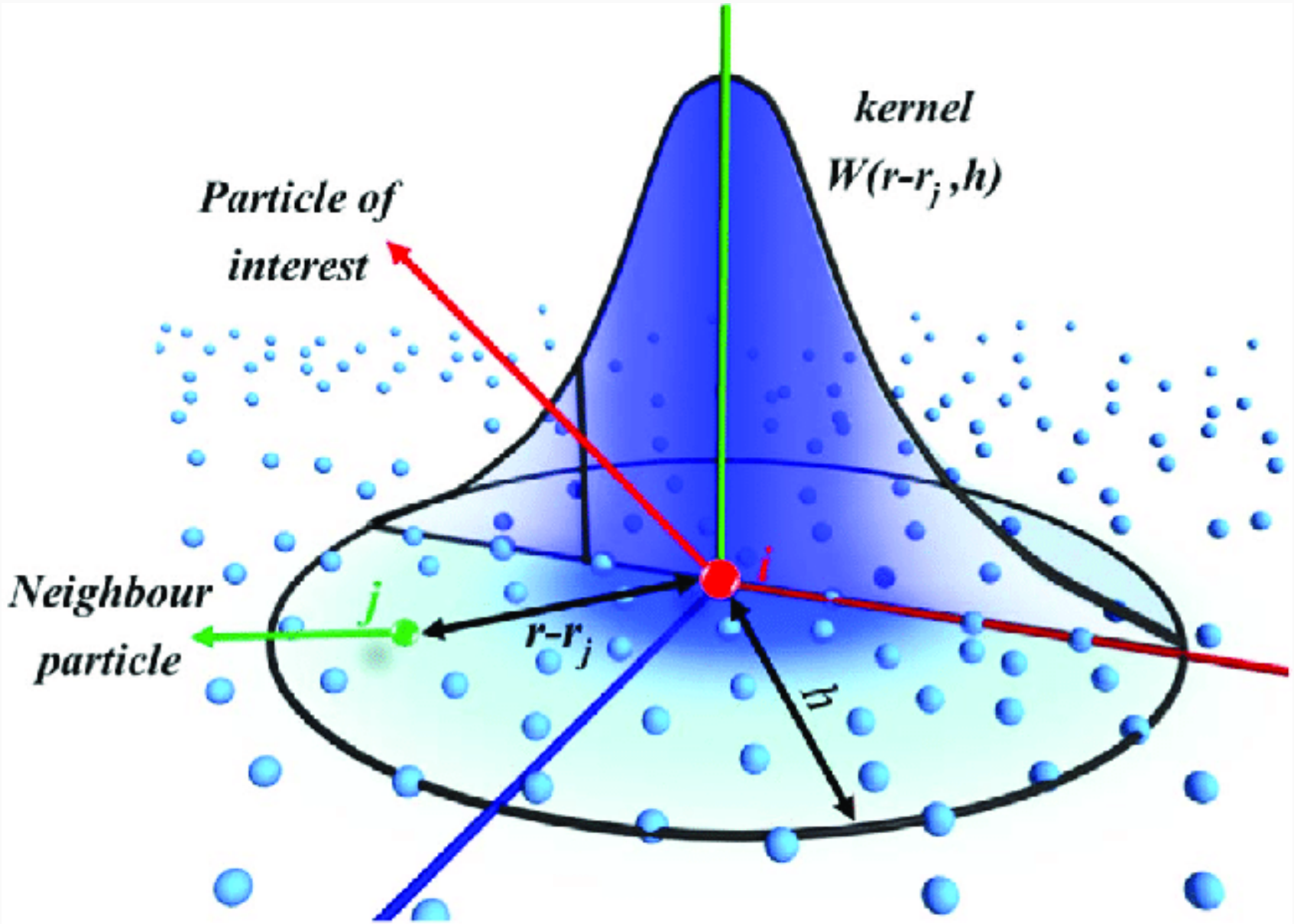
SPH — basic idea

- Continuum representent by a set of particles
- Each particle has a fixed mass m_i
- Density — similar to kernel density estimate:

$$\rho(\boldsymbol{r}) = \sum_i m_i W(\boldsymbol{r} - \boldsymbol{r}_i, h_i)$$

  where W is the kernel function and h is the smoothing length
- Smoothing kernel $W(\boldsymbol{r}, h)$ — known function, represents a density profile of particles
- Smoothing length h_i — unit of spatial resolution, generally different for each particle

Smoothing kernel



Kernel properties

- Continuous & smooth
- Monotonic — density decreases with increasing distance
- Non-negative — to avoid non-physical negative densities

Kernel properties

- Continuous & smooth
- Monotonic — density decreases with increasing distance
- Non-negative — to avoid non-physical negative densities
- Normalization

$$M = \int \rho(\boldsymbol{r})\,\mathrm{d}V = \sum_i m_i \int W(\boldsymbol{r} - \boldsymbol{r}_i)\,\mathrm{d}V$$
$$= \sum_i m_i$$

thus:

$$\int W(\boldsymbol{r})\,\mathrm{d}V = 1$$

Kernel properties

- Continuous & smooth
- Monotonic — density decreases with increasing distance
- Non-negative — to avoid non-physical negative densities
- Normalization

$$M = \int \rho(\boldsymbol{r}) \, \mathrm{d}V = \sum_i m_i \int W(\boldsymbol{r} - \boldsymbol{r}_i) \, \mathrm{d}V$$
$$= \sum_i m_i$$

thus:

$$\int W(\boldsymbol{r}) \, \mathrm{d}V = 1$$

- Isotropy

$$W(\boldsymbol{r}) = w(\|\boldsymbol{r}\|)$$

SPH kernels

- Kernel is an approximation of δ-function:

$$\lim_{h \to 0} W(\boldsymbol{r}, h) = \delta(\boldsymbol{r})$$

SPH kernels

- Kernel is an approximation of δ-function:

$$\lim_{h \to 0} W(\boldsymbol{r}, h) = \delta(\boldsymbol{r})$$

- Usually finite support — performance reasons
- Example: Gaussian, Wendland functions

SPH kernels

- Kernel is an approximation of δ-function:

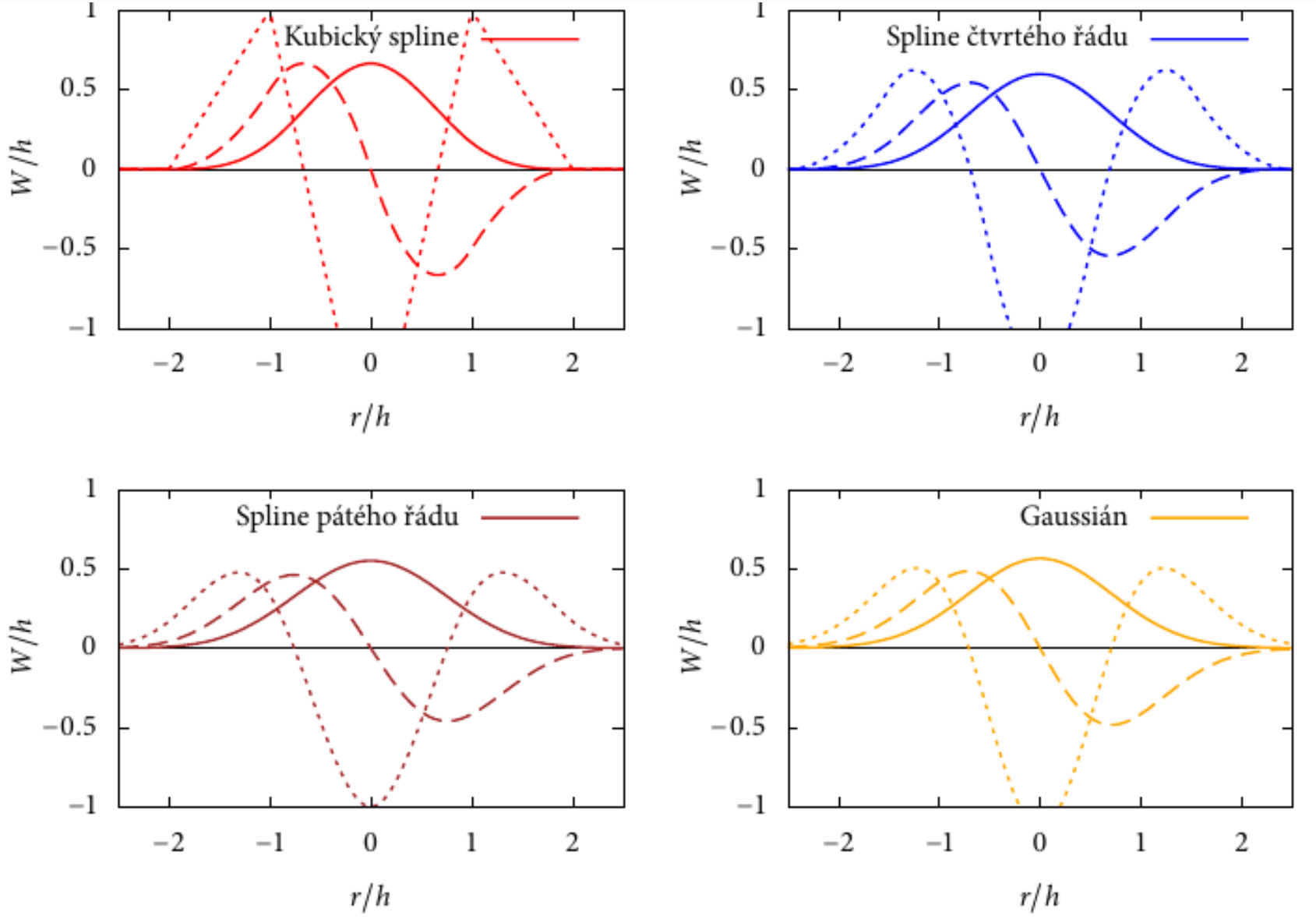
$$\lim_{h \to 0} W(\boldsymbol{r}, h) = \delta(\boldsymbol{r})$$

- Usually finite support — performance reasons
- Example: Gaussian, Wendland functions
- Typically piecewise polynomial:

$$W(r, h) = \frac{\sigma}{h^3} \begin{cases} \frac{1}{4}(2-q)^3 - (1-q)^3\,, & 0 \le q < 1\,, \\ \frac{1}{4}(2-q)^3\,, & 1 \le q < 2\,, \\ 0 & q \ge 2\,, \end{cases} \qquad (1)$$

where σ is normalization constant

SPH kernels


Kubický spline
Spline čtvrtého řádu
Spline pátého řádu
Gaussián
W/h
r/h
1
0.5
0
−0.5
−1
−2
−1
0
1
2

SPH interpolation

- Start off with an identity:

$$A(\boldsymbol{r}) = \int A(\boldsymbol{r}')\delta(\boldsymbol{r} - \boldsymbol{r}')\,\mathrm{d}V'$$

SPH interpolation

- Start off with an identity:

$$A(\boldsymbol{r}) = \int A(\boldsymbol{r}')\delta(\boldsymbol{r} - \boldsymbol{r}')\,\mathrm{d}V'$$

- Discretization $\longrightarrow$ replace δ with W:

$$A(\boldsymbol{r}) \simeq \int A(\boldsymbol{r}')W(\boldsymbol{r} - \boldsymbol{r}', h)\,\mathrm{d}V'$$

SPH interpolation

- Start off with an identity:

$$A(\boldsymbol{r}) = \int A(\boldsymbol{r}')\delta(\boldsymbol{r} - \boldsymbol{r}')\,\mathrm{d}V'$$

- Discretization $\longrightarrow$ replace δ with W:

$$A(\boldsymbol{r}) \simeq \int A(\boldsymbol{r}')W(\boldsymbol{r} - \boldsymbol{r}', h)\,\mathrm{d}V'$$

- Replace the volume element $\mathrm{d}V$ with $\frac{m_i}{\rho_i}$ — convert integral into a finite sum:

$$A(\boldsymbol{r}) \simeq \sum_i A_i \frac{m_i}{\rho_i} W(\boldsymbol{r} - \boldsymbol{r}', h_i)$$

SPH interpolation

- Similarly discretize spatial derivatives:

$$\nabla A(\boldsymbol{r}) = \sum_i A_i \frac{m_i}{\rho_i} \nabla W(\boldsymbol{r} - \boldsymbol{r}_i, h_i)$$

$$\nabla \cdot \boldsymbol{A}(\boldsymbol{r}) = \sum_i \boldsymbol{A}_i \frac{m_i}{\rho_i} \cdot \nabla W(\boldsymbol{r} - \boldsymbol{r}_i, h_i)$$

$$\nabla \times \boldsymbol{A}(\boldsymbol{r}) = \sum_i \boldsymbol{A}_i \frac{m_i}{\rho_i} \times \nabla W(\boldsymbol{r} - \boldsymbol{r}_i, h_i)$$

- Discretization error is of order $\mathcal{O}(h^2)$

SPH interpolation

- Similarly discretize spatial derivatives:

$$\nabla A(\boldsymbol{r}) = \sum_i A_i \frac{m_i}{\rho_i} \nabla W(\boldsymbol{r} - \boldsymbol{r}_i, h_i)$$

$$\nabla \cdot \boldsymbol{A}(\boldsymbol{r}) = \sum_i \boldsymbol{A}_i \frac{m_i}{\rho_i} \cdot \nabla W(\boldsymbol{r} - \boldsymbol{r}_i, h_i)$$

$$\nabla \times \boldsymbol{A}(\boldsymbol{r}) = \sum_i \boldsymbol{A}_i \frac{m_i}{\rho_i} \times \nabla W(\boldsymbol{r} - \boldsymbol{r}_i, h_i)$$

- Discretization error is of order $\mathcal{O}(h^2)$
- We can use this to discretize the equations:

$$\frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = -\frac{1}{\rho_i} \nabla P_i \simeq -\frac{1}{\rho_i} \sum_j \frac{m_j}{\rho_j} P_j \nabla W(\boldsymbol{r}_j - \boldsymbol{r}_i, h_j)$$

Constant functions in SPH

- Naïve discretization has suboptimal properties

Constant functions in SPH

- Naïve discretization has suboptimal properties
- Problem 1: Constant functions are no longer contant after discretization

$$\nabla C = C \sum_j \frac{m_j}{\rho_j} \nabla W_{ij} \neq \mathbf{0}$$

Constant functions in SPH

- Naïve discretization has suboptimal properties
- Problem 1: Constant functions are no longer contant after discretization

$$\nabla C = C \sum_j \frac{m_j}{\rho_j} \nabla W_{ij} \neq \mathbf{0}$$

- Fix: subtract "gradient of 1"

$$\nabla A = \nabla A - A\nabla 1 = \sum_j \frac{m_j}{\rho_j} A \nabla W_{ij} - A \sum_j \frac{m_j}{\rho_j} \nabla W_{ij} = \mathbf{0}$$

- Leads to discretization

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

Linear function in SPH

- Can be improved further — linear functions still linear after discretization
- Instead of subtracting a constant value, we multiply the kernel by a correction tensor:

$$\boldsymbol{C}_i = \left(\sum_j \frac{m_j}{\rho_j} (\boldsymbol{r}_j - \boldsymbol{r}_i) \otimes \nabla W_{ij} \right)^{-1}$$

- Leads to discretization:

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

- More precise at a cost of higher overhead (matrix inversion for each particle)

SPH discretizations

- Can we simply replace gradient with arbitrary discretization?
  $\longrightarrow$ YES
- SPH has a lot of different gradients
- discretization error is always $\mathcal{O}(h^2)$
- So which one is the "correct" one?
- Problem dependent ...

Conservation of integrals

- Problem 2: Discretization does not conserve linear momentum, angular momentum, energy, ...

$$m_i \frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} \neq -m_j \frac{\mathrm{d}\boldsymbol{v}_j}{\mathrm{d}t}$$

- Derive conservative equations — use Lagrange's equations:

$$\frac{\partial L}{\partial \boldsymbol{r}_i} - \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \boldsymbol{v}_i}\right) = 0$$

Consistent SPH equations

- Lagrangian $L = T - V$

$$L = \sum_j \left(\frac{1}{2} m_j \boldsymbol{v}_j^2 - m_j u_j \right)$$

  where u_j is specific internal energy of j-th particle

- Internal energy — first law of thermodynamics:

$$\mathrm{d}U = T\mathrm{d}S - p\mathrm{d}V$$

  or in intensive quantities:

$$\mathrm{d}u = T\mathrm{d}s + \frac{p}{\rho^2}\mathrm{d}\rho$$

Consistent SPH equations

- Internal energy does not depend on velocity:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \boldsymbol{v}_i}\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\sum_j \frac{1}{2} m_j \frac{\partial \boldsymbol{v}_j^2}{\partial \boldsymbol{v}_i}\right) = m_i \frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t}$$

- First term — gradient of internal energy u:

$$\frac{\partial L}{\partial \boldsymbol{r}_i} = -\sum_j m_j \frac{\partial u(\rho_j, s_j)}{\partial \boldsymbol{r}_i}$$

$$\frac{\partial u(\rho_j, s_j)}{\partial \boldsymbol{r}_i} = T_i \frac{\mathrm{d}s_i}{\mathrm{d}\boldsymbol{r}_i} + \frac{p_i}{\rho_i^2} \frac{\mathrm{d}\rho}{\mathrm{d}\boldsymbol{r}_i}$$

- Assuming isentropic process $\mathrm{d}s = 0$

Consistent SPH equations

- Gradient of density with respect to position of i-th particle:

$$\frac{\partial \rho_j}{\partial \boldsymbol{r}_i} = \sum_k m_k \frac{\partial W_{jk}}{\partial \boldsymbol{r}_i} = \sum_k m_k (\delta_{ji} - \delta_{ki}) \nabla W_{jk}$$

- Equation of motion:

$$m_i \frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = -\sum_j m_i m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) \nabla W_{ij}$$

- Energy equation:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = \frac{P_i}{\rho_i^2} \sum_j m_j (\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \nabla W_{ij}$$

- Continuity equation:

$$\frac{\mathrm{d}\rho_i}{\mathrm{d}t} = \sum_j m_j (\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \nabla W_{ij}$$

SPH equations — notes

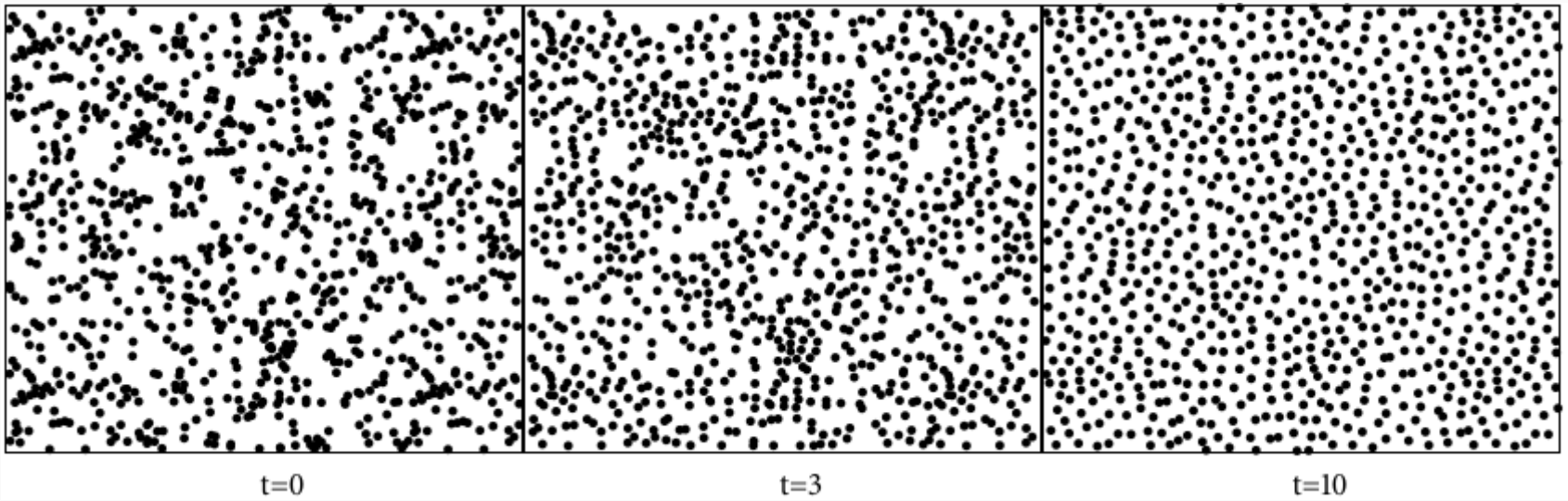
- Depends only on the difference of velocities $\boldsymbol{v}_i - \boldsymbol{v}_j$
  $\longrightarrow$ Galilean invariance
- If constant velocity ($\boldsymbol{v}_i = \boldsymbol{v}_j$) $\longrightarrow$ density and energy constant

SPH equations — notes

- Depends only on the difference of velocities $\boldsymbol{v}_i - \boldsymbol{v}_j$
  $\longrightarrow$ Galilean invariance
- If constant velocity ($\boldsymbol{v}_i = \boldsymbol{v}_j$) $\longrightarrow$ density and energy constant
- Depends on the sum of (scaled) pressures P_i and P_j
  $\longrightarrow$ generally non-zero acceleration even if $P = \text{const.}$

SPH equations — notes

- Depends only on the difference of velocities $\boldsymbol{v}_i - \boldsymbol{v}_j$
  $\longrightarrow$ Galilean invariance
- If constant velocity ($\boldsymbol{v}_i = \boldsymbol{v}_j$) $\longrightarrow$ density and energy constant
- Depends on the sum of (scaled) pressures P_i and P_j
  $\longrightarrow$ generally non-zero acceleration even if $P = \text{const.}$
- Provides numerical repulsive force — regularization



SPH equations — notes

- Conserves total linear momentum
- Conserves total angular momentum if no viscous forces are used
- Conserves total energy
- With viscous forces: angular momentum conservation can be improved by adding the correction tensor:

$$m_i \frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = -\sum_j m_i m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) \boldsymbol{C}_i \nabla W_{ij}$$

Smoothing lengths h

- Balances locality and discretization error
- Large smoothing lengths $\longrightarrow$ lot of neighbors, precise interpolation
  BUT need more particle for the same spatial resolution
- Smaller smoothing lengths $\longrightarrow$ fewer neighbors, faster, less precise, noisy

Smoothing lengths h

- Balances locality and discretization error
- Large smoothing lengths $\longrightarrow$ lot of neighbors, precise interpolation BUT need more particle for the same spatial resolution
- Smaller smoothing lengths $\longrightarrow$ fewer neighbors, faster, less precise, noisy
- Can we get arbitrarily precise interpolation by increasing number of neighbors?

Smoothing lengths h

- Balances locality and discretization error
- Large smoothing lengths $\longrightarrow$ lot of neighbors, precise interpolation
  BUT need more particle for the same spatial resolution
- Smaller smoothing lengths $\longrightarrow$ fewer neighbors, faster, less precise, noisy
- Can we get arbitrarily precise interpolation by increasing number of neighbors?
  $\longrightarrow$ unfortunately no

Smoothing lengths h

- Balances locality and discretization error
- Large smoothing lengths $\longrightarrow$ lot of neighbors, precise interpolation BUT need more particle for the same spatial resolution
- Smaller smoothing lengths $\longrightarrow$ fewer neighbors, faster, less precise, noisy
- Can we get arbitrarily precise interpolation by increasing number of neighbors?
  $\longrightarrow$ unfortunately no
- Pairing instability — when particle density exceeds critical value, numerical repulsive forces become attractive
- Leads to particles collapsing on top of each other — effectively losing half the spatial resolution

Adaptive spatial resolution

- Desirable to have large h in places with few particles, small h in places with a lot of particles
- Automatically balance h — "continuity equation"

$$\frac{\mathrm{d}h_i}{\mathrm{d}t} = \frac{h_i}{3}\nabla \cdot \boldsymbol{v}_i$$

- Major strength of SPH — adaptive mesh refinement is order of magnitude more diffucilt to implement

Artificial viscosity

- SPH continuum description does not handle discontinuities
- Shock waves, material interfaces, ...
- Particle interpenetration — velocity field becomes multivalued

Artificial viscosity

- SPH continuum description does not handle discontinuities
- Shock waves, material interfaces, ...
- Particle interpenetration — velocity field becomes multivalued
- Solution is to smooth the discontinuity over several hs
- Add artificial viscosity:

$$\Pi_i = \sum_j m_j \frac{-\alpha \bar{c}_{ij} \mu_{ij} + \beta \mu_{ij}^2}{\bar{\rho}_{ij}} \nabla W_{ij}$$

where μ_{ij} is an approximation of $\nabla \cdot \boldsymbol{v}$:

$$\mu_{ij} = \frac{h(\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot (\boldsymbol{r}_i - \boldsymbol{r}_j)}{\|\boldsymbol{r}_i - \boldsymbol{r}_j\|^2 + \epsilon h^2}$$

SPH algorithm

For each time step:

1. Compute P and c_s from EoS
2. Update smoothing lengths h_i, determine the search radius
3. Find neighbors of each particle
4. Loop over neighbors, compute sum ∇P, $\nabla \cdot \boldsymbol{S}$, $\nabla \cdot v$, ...
5. Compute LHS: $\dot{\rho}, \dot{v}, \dot{u}, ...$
6. Integrate using predictor-corrector, RK4, ...
7. Using LHSs, determine new time step

SPH algorithm

The most basic SPH code:

```
for (float t = 0; t < duration; t += dt) {
  // update pressure values using the equation of state
  for (int i = 0; i < num_particles; i++) {
    p[i] = eos(rho[i]);
    dv[i] = divv[i] = drho[i] = 0;
  }
  // compute derivatives by summing up neighbor values
  for (int i = 0; i < num_particles; i++) {
    for (int j : neighbors(i)) {
      Vector grad = kernel.gradient(r[i] - r[j], 0.5*(h[i] + h[j]));
      divv[i] += m[j]/rho[j] * (v[j] - v[i]) * grad;
      dv[i] += m[j]/rho[j] * (p[i]/rho[i] + p[j]/rho[j]) * grad;
    }
    drho[i] = -rho[i] * divv[i];
  }
  // integrate time-dependent quantities
  for (int i = 0; i < num_particles; i++) {
    r[i] += v[i] * dt;
    v[i] += dv[i] * dt;
    rho[i] += drho[i] * dt;
  }
}
```

Density evolution

Density is now given by two different equations

- Direct summation:

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

- Continuity equation:

$$\frac{\mathrm{d}\rho_i}{\mathrm{d}t} = \sum_j m_j (\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \nabla W_{ij}$$

Density evolution

Density is now given by two different equations

- Direct summation:

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

- Continuity equation:

$$\frac{\mathrm{d}\rho_i}{\mathrm{d}t} = \sum_j m_j (\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \nabla W_{ij}$$

- We can use either one

Density evolution

Pros & cons:

- Direct summation requires two loops over all particles – first to update the density, second to compute all derivatives

Density evolution

Pros & cons:

- Direct summation requires two loops over all particles – first to update the density, second to compute all derivatives
- Direct summation has issues on surfaces and density interfaces (artificially low density)

Density evolution

Pros & cons:

- Direct summation requires two loops over all particles – first to update the density, second to compute all derivatives
- Direct summation has issues on surfaces and density interfaces (artificially low density)
- Continuity equation is less robust (prone to unphysical high-frequency oscillations)

Density evolution

Pros & cons:

- Direct summation requires two loops over all particles – first to update the density, second to compute all derivatives
- Direct summation has issues on surfaces and density interfaces (artificially low density)
- Continuity equation is less robust (prone to unphysical high-frequency oscillations)
- Direct summation enforces smoothing of density field

Gradient of smoothing length

- So far, we assumed $h = \text{const}$ when deriving the SPH equations
- Particles have generally different hs — ∇W will contain terms related to the smoothing length gradient

Gradient of smoothing length

- So far, we assumed $h = \text{const}$ when deriving the SPH equations
- Particles have generally different hs — ∇W will contain terms related to the smoothing length gradient
- Grad-h terms:

$$\Omega_i = 1 - \frac{\partial h_i}{\partial \rho_i} \sum_j m_j \frac{\partial W_{ij}(h_i)}{\partial h_i}$$

Gradient of smoothing length

- So far, we assumed $h = \text{const}$ when deriving the SPH equations
- Particles have generally different hs — ∇W will contain terms related to the smoothing length gradient
- Grad-h terms:

$$\Omega_i = 1 - \frac{\partial h_i}{\partial \rho_i} \sum_j m_j \frac{\partial W_{ij}(h_i)}{\partial h_i}$$

- Set of equations is then modified as:

$$\frac{\mathrm{d}\rho_i}{\mathrm{d}t} = \frac{1}{\Omega_i} \sum_j m_j (\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \nabla W_{ij}(h_i)$$

$$\frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = -\sum_j m_j \left(\frac{P_i}{\Omega_i \rho_i^2} \nabla W_{ij}(h_i) + \frac{P_j}{\Omega_j \rho_j^2} \nabla W_{ij}(h_j) \right)$$

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \frac{P_i}{\Omega_i \rho_i^2} \sum_j m_j (\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \nabla W_{ij}(h_i)$$

Gradient of smoothing length

- Ensure momentum conservation — action and reaction principle
- Generally resolved by kernel symmetrization

Gradient of smoothing length

- Ensure momentum conservation — action and reaction principle
- Generally resolved by kernel symmetrization
- Symmetrize smoothing lenghts:

$$W_{ij} \longrightarrow W(\boldsymbol{r}_i - \boldsymbol{r}_j, \frac{h_i + h_j}{2})$$

- Symmetrize kernels:

$$W_{ij} \longrightarrow \frac{W(\boldsymbol{r}_i - \boldsymbol{r}_j, h_i) + W(\boldsymbol{r}_i - \boldsymbol{r}_j, h_j)}{2}$$

Boundary conditions

- Difficult to realize arbitrary boundary condition
- "Implicit" BCs — $\rho(\boldsymbol{r}) = 0$ (vacuum BC)
- Alternatively, we can use periodic boundary conditions
- Reflecting boundaries?

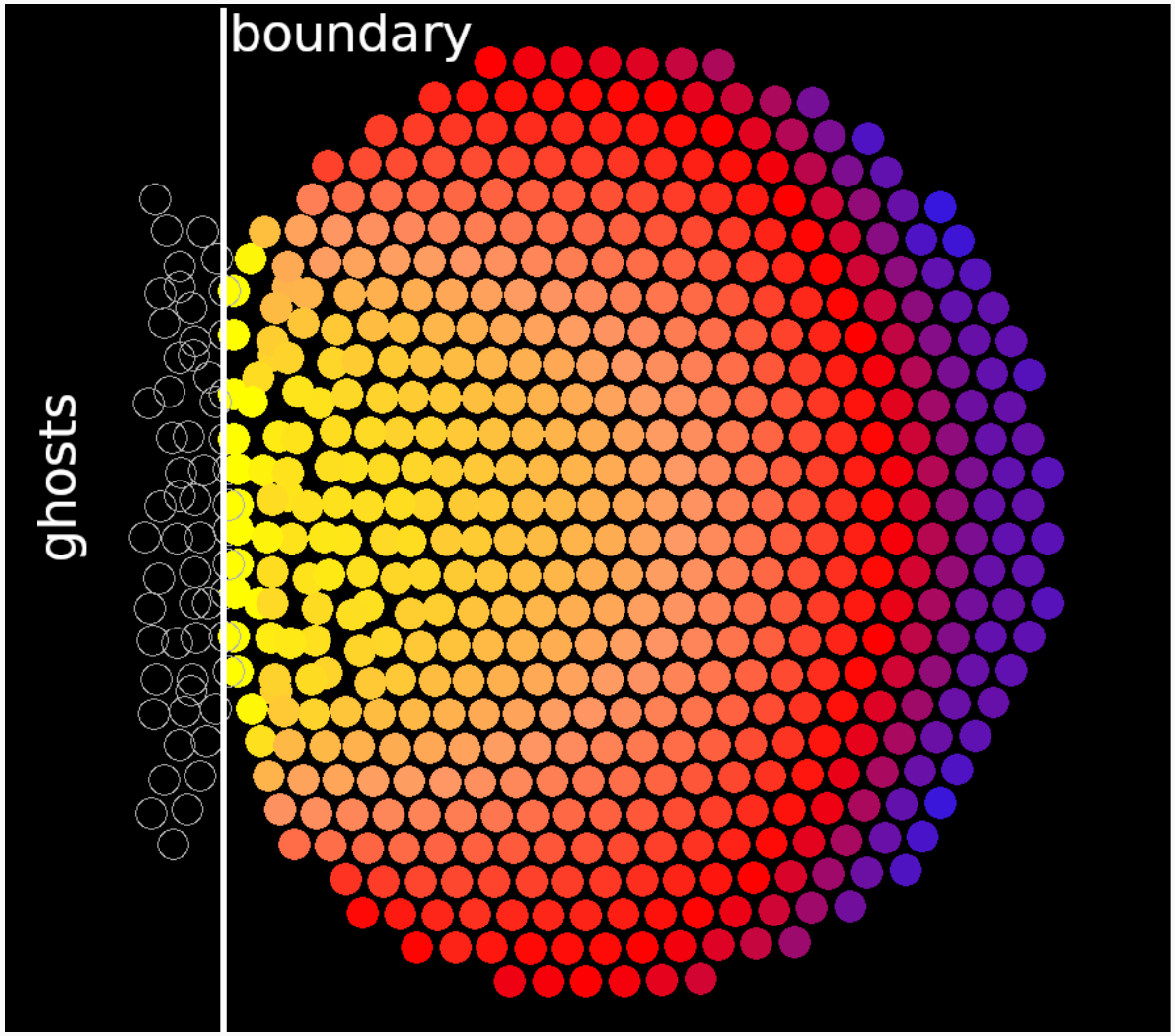
Boundary conditions

- Difficult to realize arbitrary boundary condition
- "Implicit" BCs — $\rho(\boldsymbol{r}) = 0$ (vacuum BC)
- Alternatively, we can use periodic boundary conditions
- Reflecting boundaries?
- Create physical domain using particles — e.g. solid boundary interacting with fluid particles

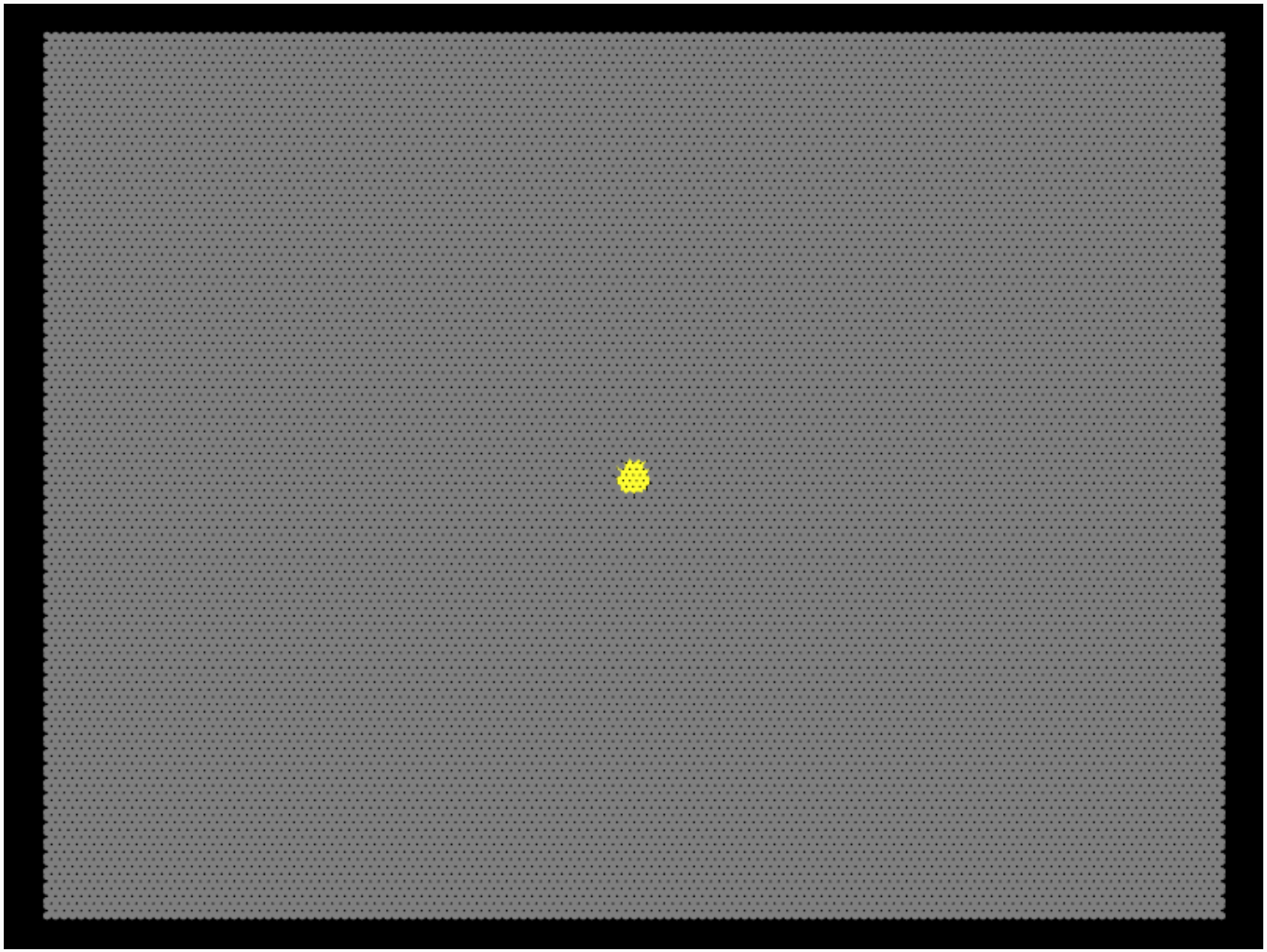
Boundary conditions

- Difficult to realize arbitrary boundary condition
- "Implicit" BCs — $\rho(\boldsymbol{r}) = 0$ (vacuum BC)
- Alternatively, we can use periodic boundary conditions
- Reflecting boundaries?
- Create physical domain using particles — e.g. solid boundary interacting with fluid particles
- Or use ghost particles
  $\longrightarrow$ create dummy particles symmetrically along the boundary

Ghost particles



Periodic boundary



Initial conditions

- We need to generate particles
- Particle positions $\boldsymbol{r}$, smoothing lengths h + other quantities ($\boldsymbol{v}$, ρ, u, ...)
- If $\rho_i = \rho_0$, then

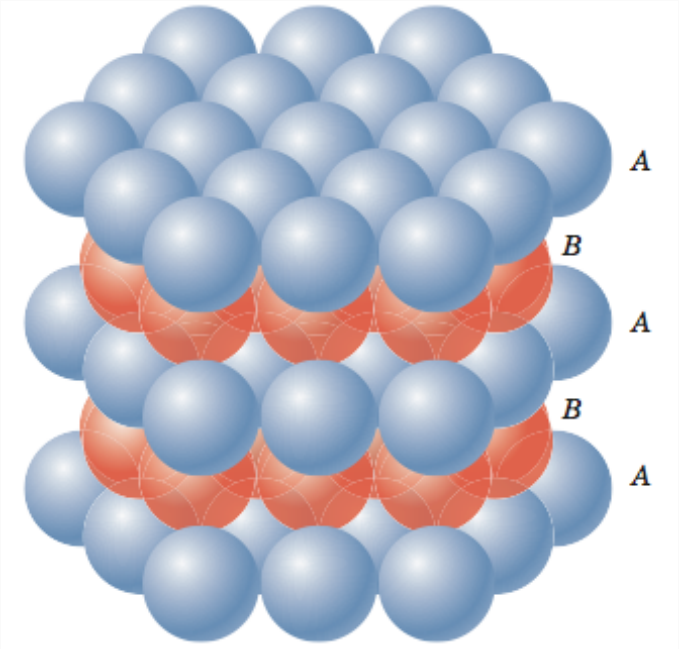
$$m_i = \frac{\rho_0 V}{N}$$

$$h_i = \left(\frac{V}{N}\right)^{\frac{1}{3}}$$

- Particle distribution?

Close hexagonal packing

- Distribute the particles in a hexagonal grid
- Easy to set up
- Stable — energy minimum
- Uniform, no particle disorder
- Not isotropic — may create numerical clumping, ...



Halton sequence

- Distribute particles using quasi-random number sequence
- Halton — low-discrepancy sequence
- n-th number of the sequence is n written in binary representation, inverted and written after the decimal point

$$\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, \frac{1}{16}, \frac{9}{16}, \dots$$

- Multiple dimensions — use other base (prime), e.g.

$$\frac{1}{3}, \frac{2}{3}, \frac{1}{9}, \frac{4}{9}, \frac{7}{9}, \frac{2}{9}, \frac{5}{9}, \frac{8}{9}, \frac{1}{27}, \dots$$

- Worse interpolation — particle clumps

Blue noise distribution

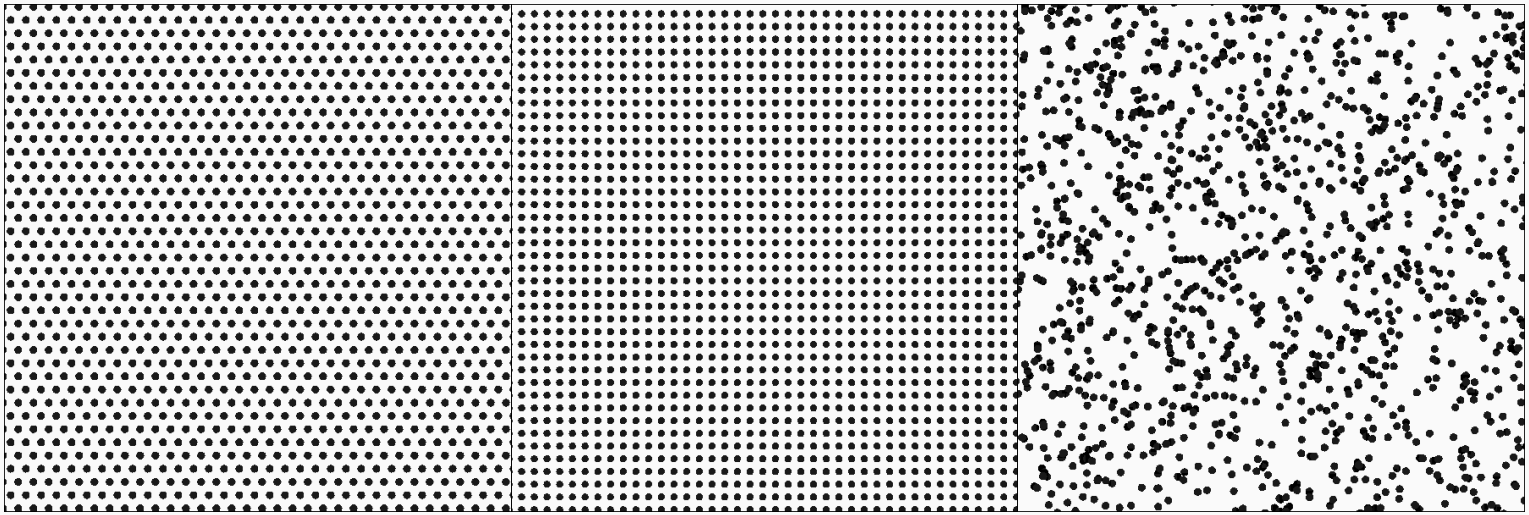
- Relaxation-based sampling
- Initially distribute particles (quasi-)randomly
- Compute repulsive forces — move particles away from each other:

$$\Delta r_i \propto \frac{\boldsymbol{r}_i - \boldsymbol{r}_j}{\|\boldsymbol{r}_i - \boldsymbol{r}_j\|^3}$$

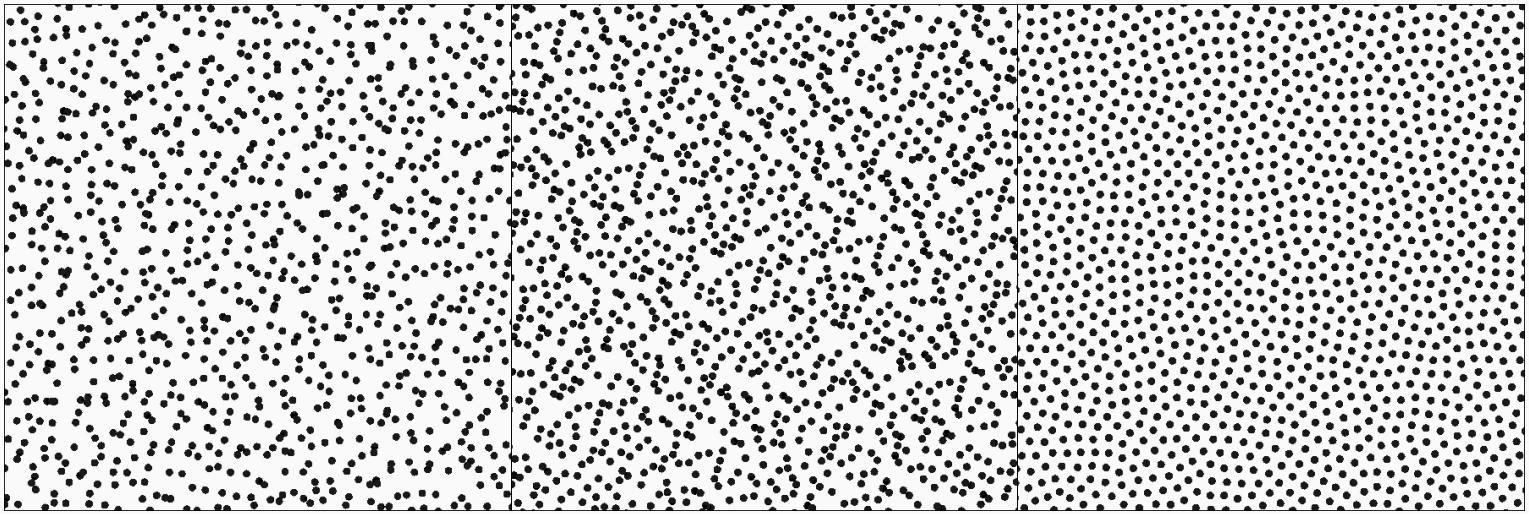
- Repeat until convergence
- Particles (almost) uniformly distributed
- Isotropic!
- More difficult to implement (especially with arbitrary domain shapes)

Particle distributions

hexagonal grid | cubic grid | white noise

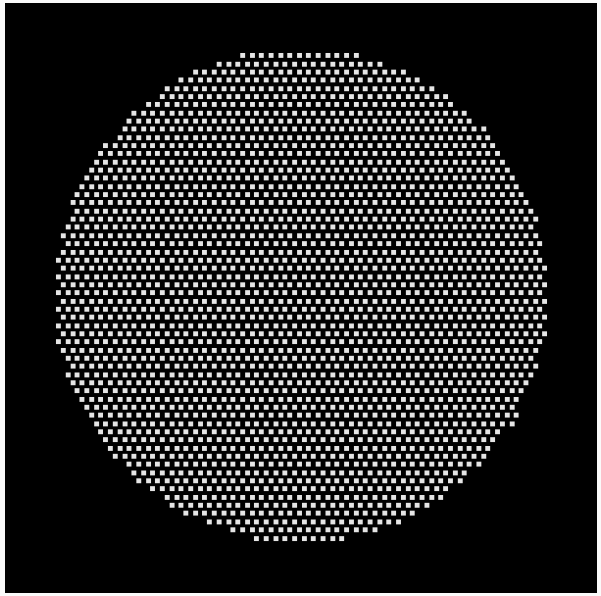


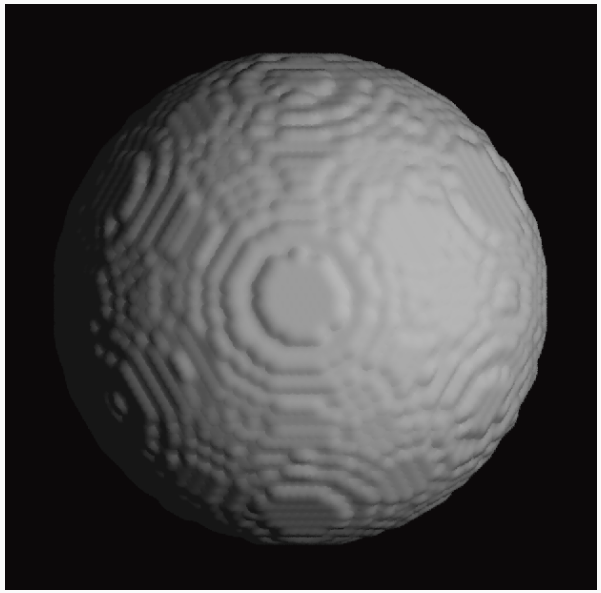
stratified | Halton | blue noise



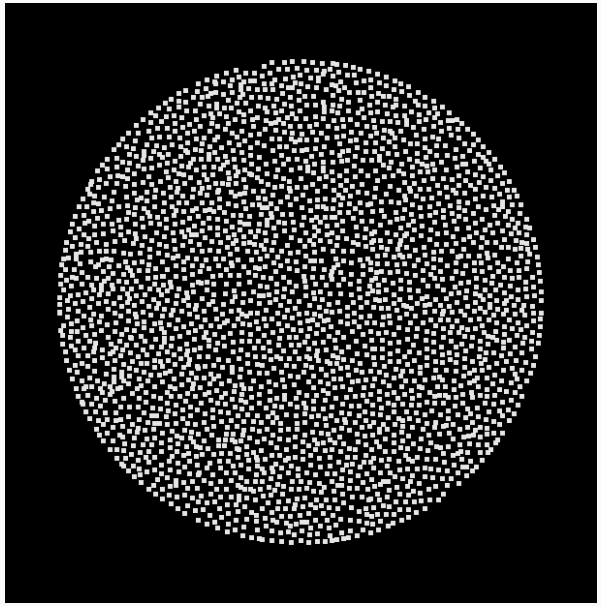
Initial conditions

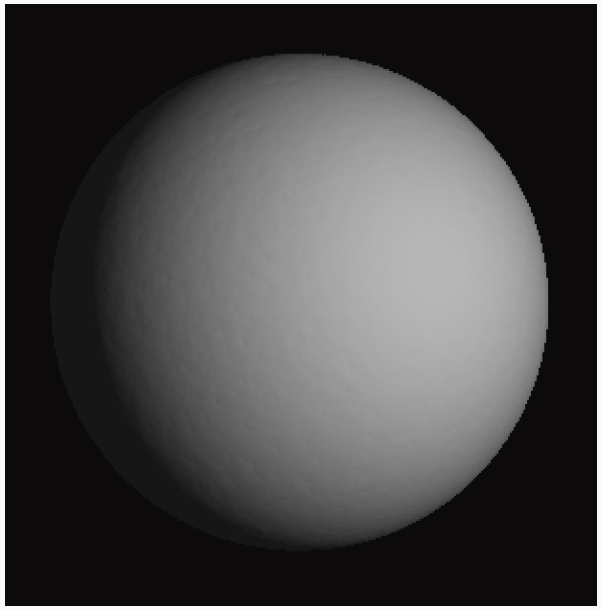
Hexagonal grid





Isotropic distribution





SPH symmetry

- All terms either symmetric or antisymmetric in particle indices i and j

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \propto \frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}$$

$$\frac{\mathrm{d}u}{\mathrm{d}t} \propto v_i - v_j$$

  etc.

- Code optimization — terms can be computed only once and added into sums for particles i and j

SPH symmetry

- All terms either symmetric or antisymmetric in particle indices i and j

$$\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \propto \frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}$$

$$\frac{\mathrm{d}u}{\mathrm{d}t} \propto v_i - v_j$$

  etc.
- Code optimization — terms can be computed only once and added into sums for particles i and j
- Cannot be used with the correction tensor $\boldsymbol{C}$ — need to be computed before other derivatives, but all the sums are only partial
- Also more difficult to parallelize

Parallelization

- Asymmetric solver — we accumulate to each particle independently $\longrightarrow$ embarrassingly parallel

```
#pragma omp parallel for
```

or

```
tbb::parallel_for(0, num_particles, [](int i) {
```

- Symmetric solver — we accumulate to both the particle and its neighbors $\longrightarrow$ parallelization more complex
- For example: using thread-local sums or domain decomposition

Finding neighbors

How to find neighbors of particle i?

- Check all particles — extremely inefficient (but correct, useful for tests)
- Look only in radius $2h_i$ — do we find all of them?

Finding neighbors

How to find neighbors of particle i?

- Check all particles — extremely inefficient (but correct, useful for tests)
- Look only in radius $2h_i$ — do we find all of them?

  NO!

  Symmetrized kernel uses $h = \frac{h_i + h_j}{2}$, it could be nonzero for $h_j > h_i$!
- Look in radius $2h_{\text{max}}$ — again inefficient for big differences in h

Can we do better?

SPH symmetric solver

- Symmetric solver processes each pair i and j only once
- Trick: sort particles by their smoothing lengths h
- Since we know order of particles (in h), we can only look for neighbors with lower rank

  $\longrightarrow$ each pair will be evaluated exactly once
  $\longrightarrow$ symmetrized length $\frac{h_i+h_j}{2}$ will be always lower than h_i

  We will not miss any neighbors!
- Cannot be used for antisymmetric solver — we need to find all neighbors :(

Second derivative in SPH

- Laplacian could be obtained using:

$$\nabla^2 A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W_{ij}$$

- Highly sensitive to particle disorder

Second derivative in SPH

- Laplacian could be obtained using:

$$\nabla^2 A_i = \sum_j \frac{m_j}{\rho_j} A_j \nabla^2 W_{ij}$$

- Highly sensitive to particle disorder
- Instead, we use approximation:

$$\nabla^2 A(\boldsymbol{r}) = \int 2 \frac{A(\boldsymbol{r}') - A(\boldsymbol{r})}{\|\boldsymbol{r}' - \boldsymbol{r}\|} (\boldsymbol{r}' - \boldsymbol{r}) \cdot \nabla W(\boldsymbol{r} - \boldsymbol{r}')$$

- Discretization:

$$\nabla^2 A_i = -2 \sum_j \frac{m_j}{\rho_j} (A_j - A_i) \frac{(\boldsymbol{r}_i - \boldsymbol{r}_j) \cdot \nabla W_{ij}}{\|\boldsymbol{r}_i - \boldsymbol{r}_j\|^2}$$

Linear consistency

- Discretization from Lagrangian: velocity gradient $\nabla \boldsymbol{v}_i$ is corrected by constant
- Problem fixed only partially — now gradients of linear functions are generally not constant $\longrightarrow$ angular momentum not conserved
- Rotations misinterpreted as deformation

Linear consistency

- Discretization from Lagrangian: velocity gradient $\nabla \boldsymbol{v}_i$ is corrected by constant
- Problem fixed only partially — now gradients of linear functions are generally not constant $\longrightarrow$ angular momentum not conserved
- Rotations misinterpreted as deformation
- We can still correct it at a cost of some computational overhead, using the correction tensor $\boldsymbol{C}$:

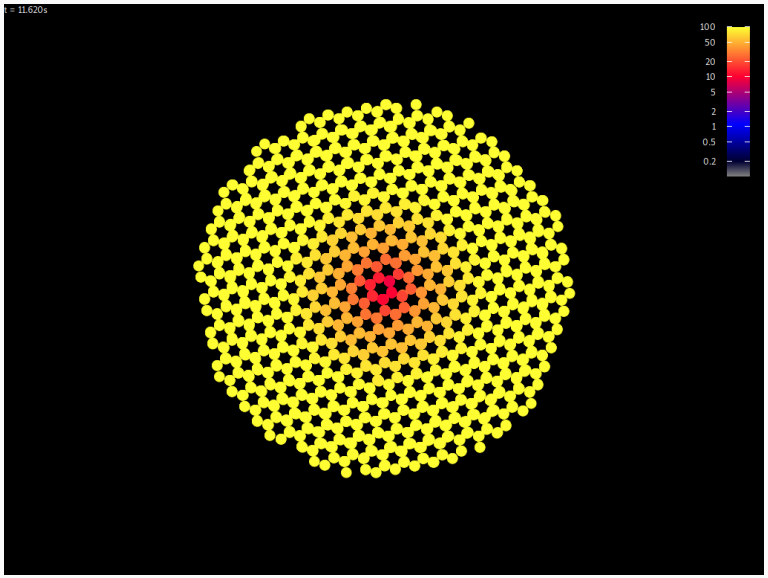
$$\boldsymbol{C}_i \equiv \left(\sum_j \frac{m_j}{\rho_j} (\boldsymbol{r}_j - \boldsymbol{r}_i) \otimes \nabla W_{ij} \right)^{-1}$$

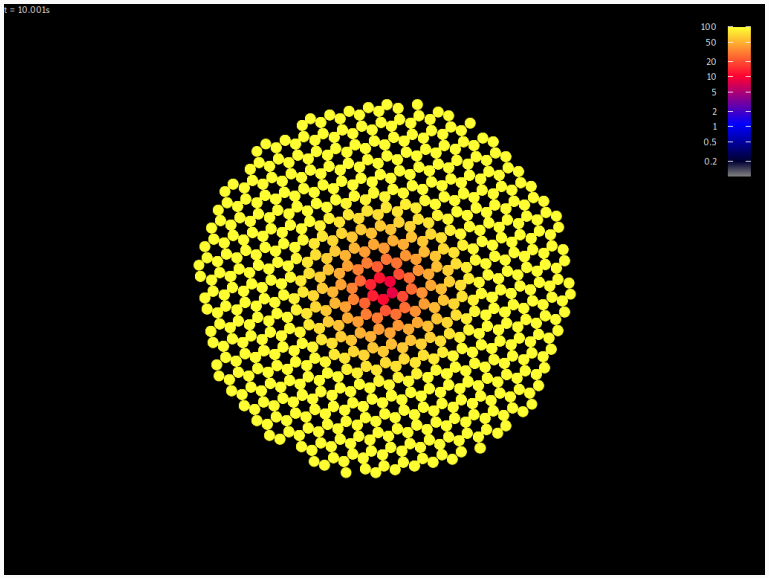
- Then the velocity gradient is:

$$\nabla \boldsymbol{v}_i \equiv \sum_j \frac{m_j}{\rho_j} (\boldsymbol{v}_j - \boldsymbol{v}_i) \boldsymbol{C}_i \cdot \nabla W_{ij}$$

Linear consistency

Linear inconsistency can be neglected for short timescales, but it is essential for long-term evolution!





Tensile instability

Standard SPH not stable for negative pressure. To solve it:

- Fixed neighborhood for each particle

$$\frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = -\sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) \nabla W(\boldsymbol{r}_i^0 - \boldsymbol{r}_j^0)$$

  (Lagrangian SPH)
- Useful to simulate elastic deformations — allows for large deformations without tensile instability
- Does not handle changes of topology

Tensile instability

Standard SPH not stable for negative pressure. To solve it:

- Fixed neighborhood for each particle

$$\frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = -\sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) \nabla W(\boldsymbol{r}_i^0 - \boldsymbol{r}_j^0)$$

  (Lagrangian SPH)
- Useful to simulate elastic deformations — allows for large deformations without tensile instability
- Does not handle changes of topology
- Alternatively, add artificial stress $\zeta^{\alpha\beta}$

Tensile instability

- Add extra term to equation of motion:

$$\zeta_i^{\alpha\beta} = \sum_j m_j R_{ij}^{\alpha\beta} f_{ij}^n$$

  where $f_{ij} = W(\boldsymbol{r}_i - \boldsymbol{r}_j)/W(\Delta p)$, Δp is the initial particle spacing, n is a fixed exponent

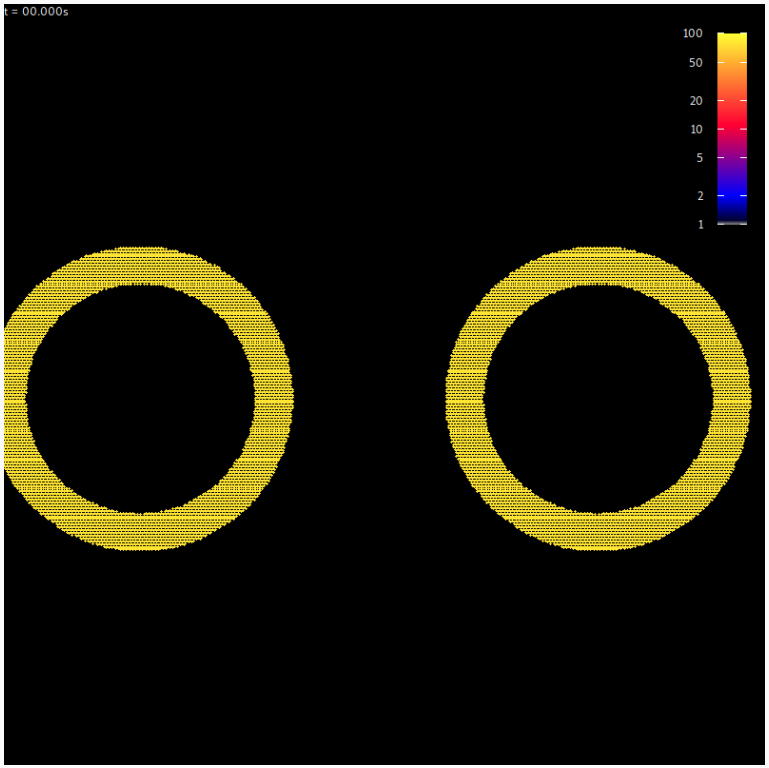
- Tensor $\boldsymbol{R}_{ij} = \boldsymbol{R}_i + \boldsymbol{R}_j$ is specified using principal axes of $\boldsymbol{\sigma}$ as:

$$R_i = -\epsilon \frac{\sigma_i}{\rho_i^2} \text{ if } \sigma_i > 0$$

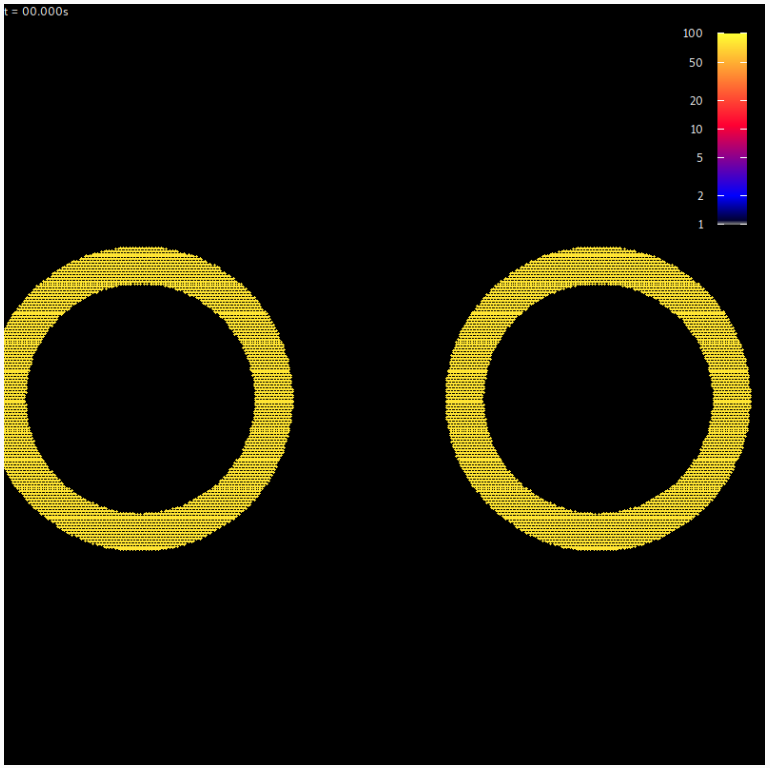
  otherwise $R_i = 0$

Tensile instability

Standard SPH



With artificial stress



Neighbor search

- Simple timestep is generally $\mathcal{O}(N_{\text{part}}^2)$
  — impossibly slow for $N_{\text{part}} > 1000$
- BUT kernel W usually has compact support — $N_{\text{neigh}} \ll N_{\text{part}}$
- Requires efficient search of neighbors
- Build an acceleration structure at the beginning of each time step
  $\longrightarrow$ makes a single timestep $\mathcal{O}(N_{\text{part}} \log N_{\text{part}})$ or $\mathcal{O}(N_{\text{part}})$

Grid-based neighbor search

- Place particles into cells of a grid
- We can compute cell indices from the particle positions in $\mathcal{O}(1)$ — potential neighbors are in neighboring cells, depending on h_i
- Inefficient if particle concentration varies significantly (=useless for impact simulations)
- SPH no longer a "gridless method" :(

Grid-based neighbor search

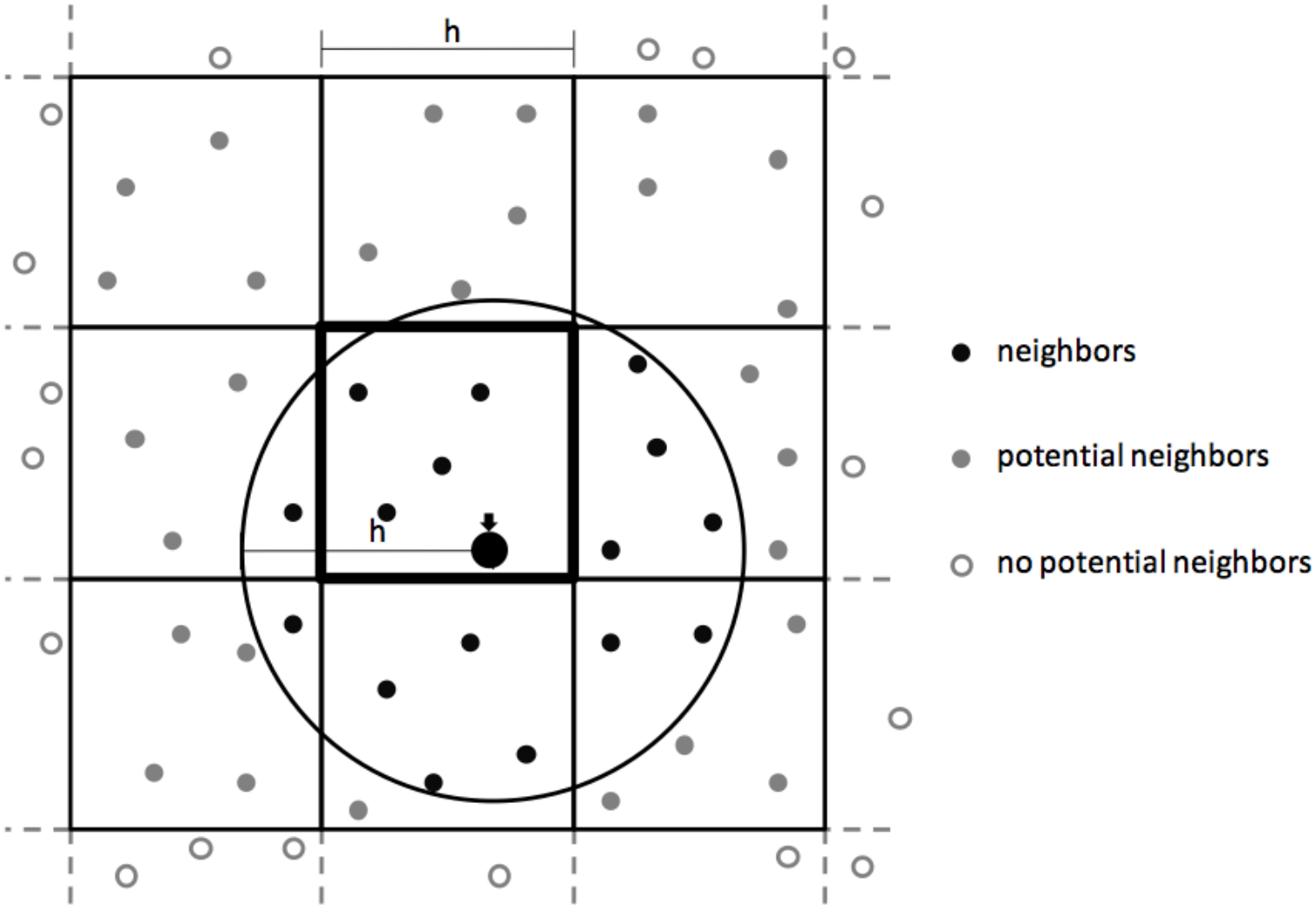
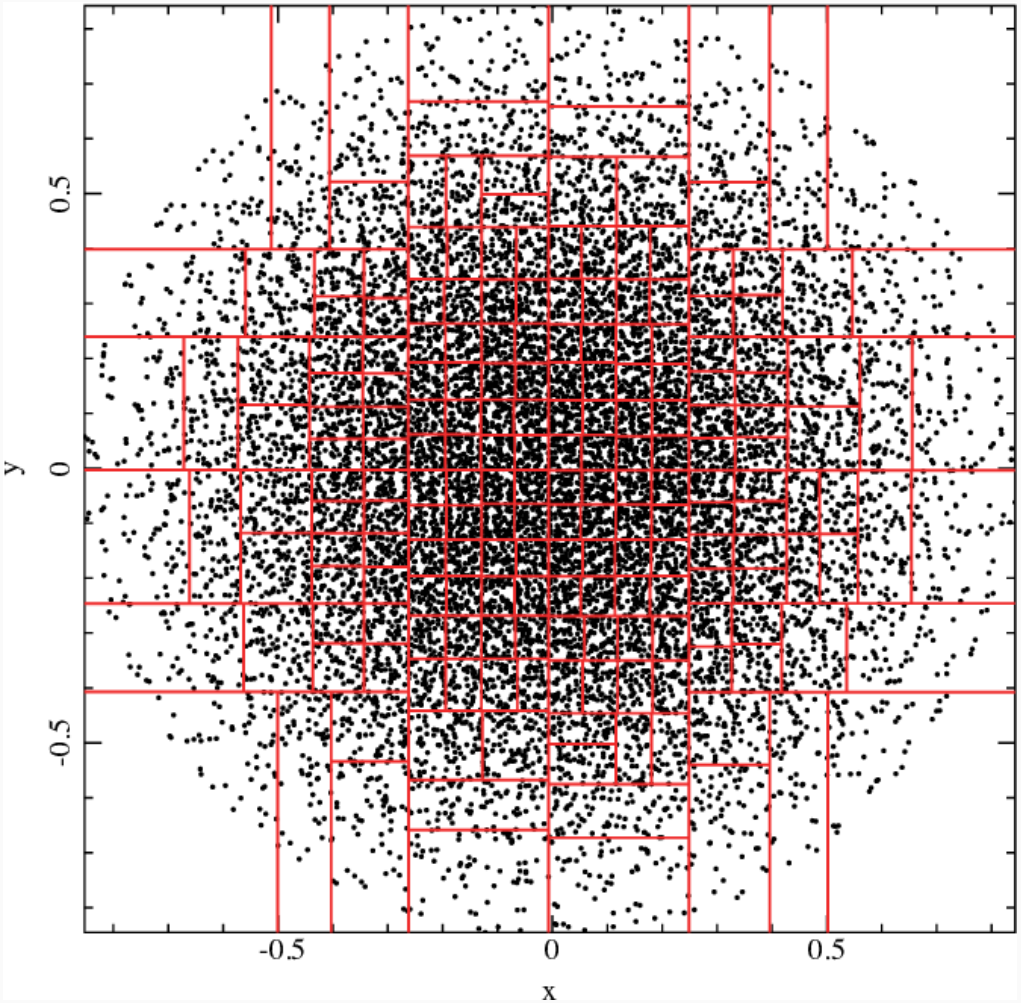


Figure 10: Grid based neighbour search

Tree-based neighbor search

- Cluster particles hierarchically into a tree (octree, K-d tree, ...)
- Neighbor lookup in $\mathcal{O}(\log N)$
- Tree might be also used for gravity evaluation — "2 in 1"



Surface handling in SPH

- SPH particles are volume elements
- Quantities continuously approach zero, where is the surface?

Surface handling in SPH

- SPH particles are volume elements
- Quantities continuously approach zero, where is the surface?
- We define the color field:

$$C(\boldsymbol{r}) \equiv \sum_j \frac{m_j}{\rho_j} W(\boldsymbol{r} - \boldsymbol{r}_j)$$

  For $N \to \infty$, $h \to 0$, color field $C = 1$ for the body and $C = 0$ for vacuum
- Then the surface of the asteroid is an isosurface $C(\boldsymbol{r}) = c_0$
- Construction of the surface: either convert to triangle mesh using marching cubes or find the intersection with rays using raymarching (depending on application)

Surface forces

- SPH normally computes volumetric forces
- Surface forces quite tricky, e.g. surface tension
- Surface area minimization term

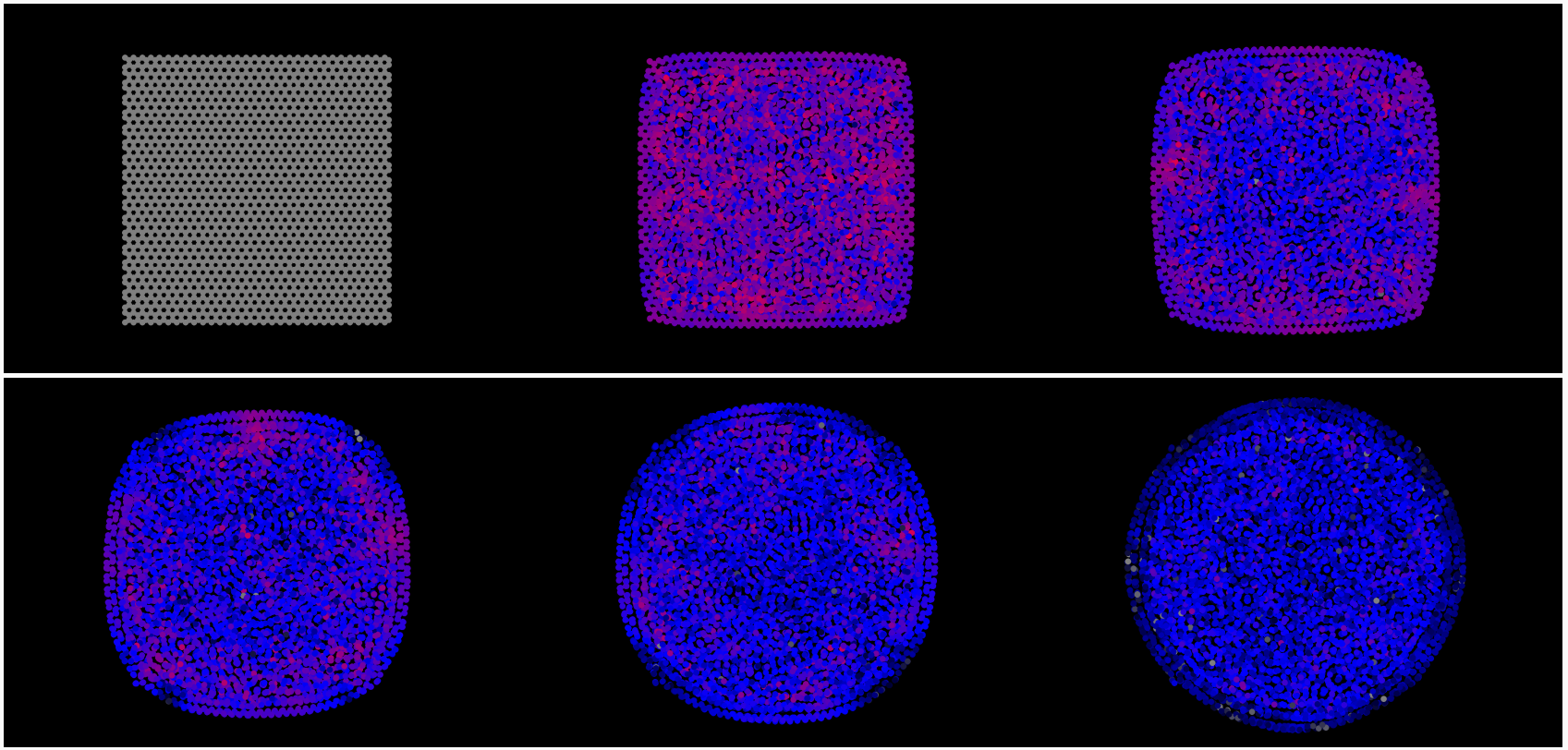
$$\boldsymbol{F}_i^{\text{surface}} = -\sum_j \gamma m_i (\boldsymbol{n}_i - \boldsymbol{n}_j)$$

where $\boldsymbol{n}_i$ is the surface normal:

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

and $\gamma \sim 1$

Surface tension



SPH codes

- `OpenSPH` — my humble contribution, mostly for impact modeling and N-body simulations
- `GADGET` — cosmological SPH/N-body simulations
- `Spheral` — hydrodynamical & gravitational numerical modeling
- `OpenFOAM` — large CFD package for engineering applications
- `DualSPHysics`
- ... and much more

Homework

- Mail: `sevecek@sirrah.troja.mff.cuni.cz`
- Code examples: `https://gitlab.com/sevecekp/pdesolvers`
- SPH code: `https://gitlab.com/sevecekp/sph`