Towards an explicit, second-order, semi-Lagrangian 3D solver for Navier–Stokes equations on low-cost GPU architectures

Roberto Ferretti

Department of Mathematics and Physics, Roma Tre University

ferretti@mat.uniroma3.it

Catania, 21 Feb 2023

joint works (partly in progress) with
L. Bonaventura, S. Cacace, E. Carlini, E. Calzola, L. Rocchi

Outline

Basic concepts on SL schemes – hyperbolic case

Model equation: linear, constant-coefficient advection

$$\begin{cases} u_t(x,t) + a u_x(x,t) = 0 \\ u(x,0) = u_0(x) \end{cases}$$

Representation formula

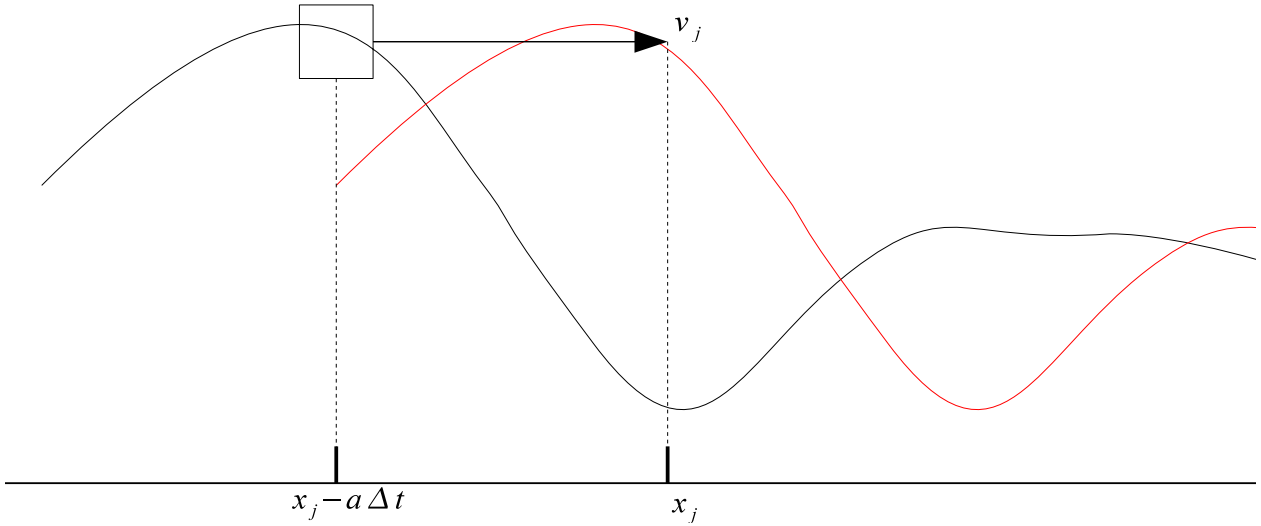
$$u(x,t) = u_0(x - at)$$

Time discretization

$$u(x_j, t_{n+1}) = u(x_j - a\Delta t, t_n)$$

Semi-Lagrangian (SL) schemes stem from the so-called **Courant–Isaacson–Rees (CIR) method** ('52) which discretizes the representation formula (instead of the equation)

Fully discrete SL schemes



Semi-Lagrangian (SL) discretization

$$v_j^{n+1} = I[V^n](x_j - a\Delta t)$$

The most classical choice for the interpolation $I[V]$ is a **symmetric Lagrange interpolation** on a structured uniform mesh. Various other options are possible, among which **Galerkin projection**

Extension to second order operators (1)

In the constant coefficient, **advection–diffusion case** we have, via the **Taylor expansion**:

$$\begin{aligned} u(x_j + a\Delta t + \sqrt{2\nu\Delta t}) &= u(x_j) + \Delta t\, a u_x(x_j) + \sqrt{2\nu\Delta t}\, u_x(x_j) + \\ &\quad + \Delta t\, \nu u_{xx}(x_j) + O(\Delta t^{3/2}) + O(\Delta t^2) \end{aligned}$$

$$\begin{aligned} u(x_j + a\Delta t - \sqrt{2\nu\Delta t}) &= u(x_j) + \Delta t\, a u_x(x_j) - \sqrt{2\nu\Delta t}\, u_x(x_j) + \\ &\quad + \Delta t\, \nu u_{xx}(x_j) - O(\Delta t^{3/2}) + O(\Delta t^2) \end{aligned}$$

Extension to second order operators (1)

In the constant coefficient, **advection–diffusion case** we have, via the **Taylor expansion**:

$$\begin{aligned} u(x_j + a\Delta t + \sqrt{2\nu\Delta t}) &= u(x_j) + \Delta t\, a u_x(x_j) + \sqrt{2\nu\Delta t}\, u_x(x_j) + \\ &\quad + \Delta t\, \nu u_{xx}(x_j) + O(\Delta t^{3/2}) + O(\Delta t^2) \end{aligned}$$

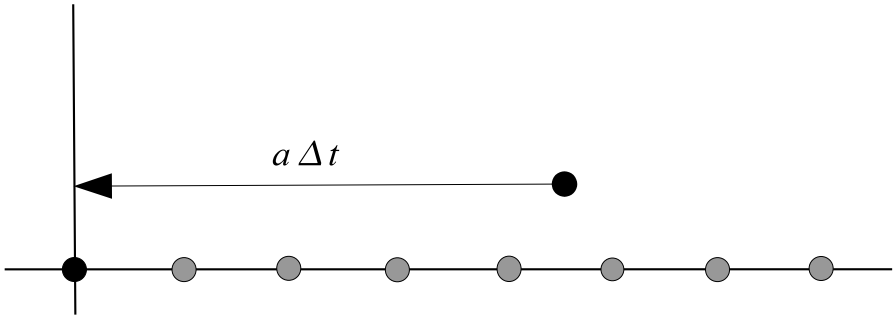
$$\begin{aligned} u(x_j + a\Delta t - \sqrt{2\nu\Delta t}) &= u(x_j) + \Delta t\, a u_x(x_j) - \sqrt{2\nu\Delta t}\, u_x(x_j) + \\ &\quad + \Delta t\, \nu u_{xx}(x_j) - O(\Delta t^{3/2}) + O(\Delta t^2) \end{aligned}$$

Abstract difference operator for advection–diffusion

$$\begin{aligned} &\frac{1}{2}\left[u(x_j + a\Delta t + \sqrt{2\nu\Delta t}) + u(x_j + a\Delta t - \sqrt{2\nu\Delta t})\right] = \\ &\qquad = u(x_j) + \Delta t\,[a u_x(x_j) + \nu u_{xx}(x_j)] + O(\Delta t^2) \end{aligned}$$

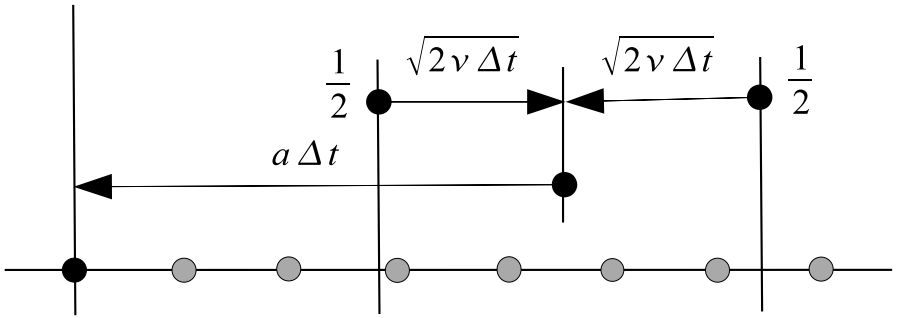
First proposed for Dynamic Programming equations [KD01, CF95]

Extension to second order operators (2)



- A first **upwinding** of magnitude $a\,\Delta t$ follows the **advection** (if advection occurs)

Extension to second order operators (2)



- A first **upwinding** of magnitude $a\,\Delta t$ follows the **advection** (if advection occurs)
- A second **symmetric displacement** of magnitude $\sqrt{2\nu\Delta t}$ is related to the **diffusion** (possibly asymmetric when **close to the boundary**)

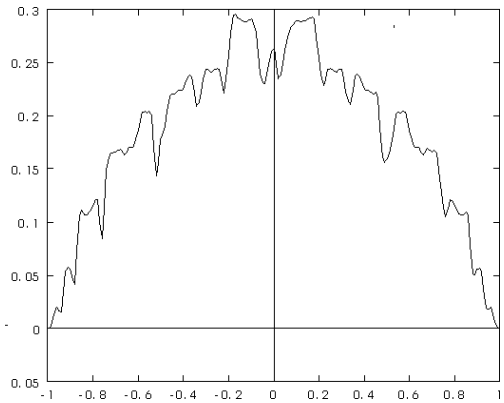
Resolution of the smaller scales (1)

The numerical domain of dependence is made of **two regions of reconstruction which are $2\sqrt{2\nu\Delta t}$ apart**. This "hole" in the stencil **may cause the smaller scales to be underresolved** [F10].

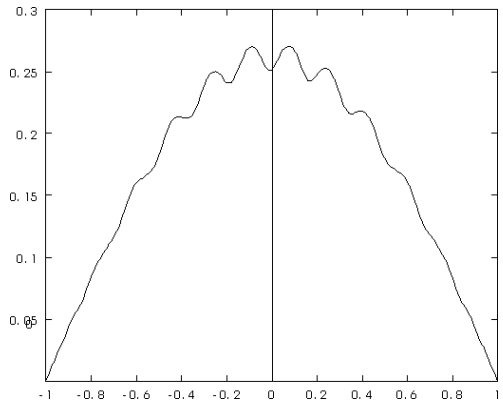
Resolution of the smaller scales (1)

The numerical domain of dependence is made of **two regions of reconstruction which are $2\sqrt{2\nu\Delta t}$ apart**. This "hole" in the stencil **may cause the smaller scales to be underresolved** [F10].

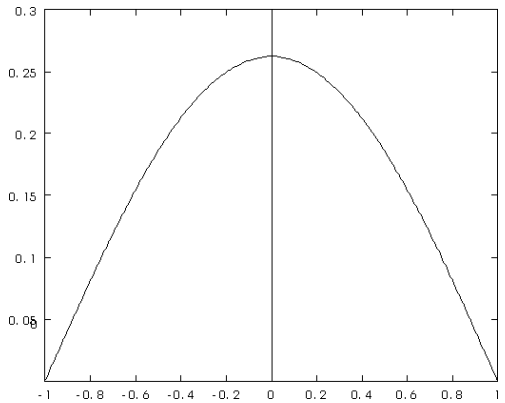
Heat equation, $\Delta x = 0.02$, discontinuous initial condition



$\Delta t = 0.1$



$\Delta t = 0.01$



$\Delta t = 0.001$

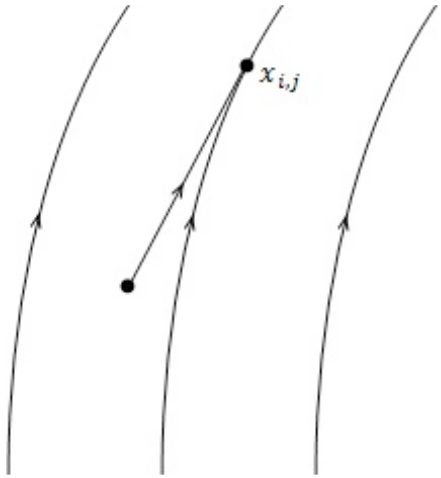
Resolution of the smaller scales (2)

- **Avoiding any such "hole"** in the numerical domain of dependence would require **the parabolic CFL condition** $\Delta t \sim \Delta x^2$
- Asymptotically, **"holes" are filled at a given time T under the weaker condition:**

$$\Delta t = o\left(T^{2/3}\Delta x^{2/3}\right)$$

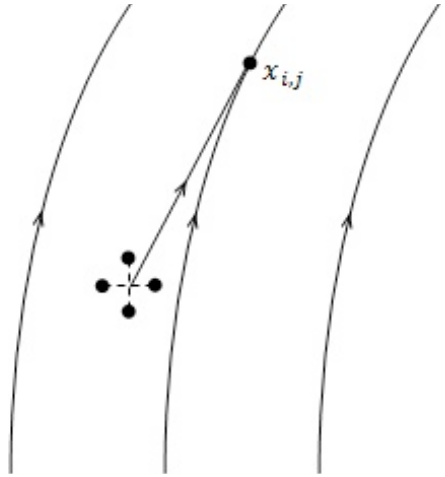
  in particular, **hyperbolic type** $\Delta t/\Delta x$ relationships are acceptable
- **Adaptive strategies** of choice for the time step are possible

Extension to 2D



- The first **upwinding** is driven by the advective term (higher order implementations are possible)

Extension to 2D



- The first **upwinding** is driven by the advective term (higher order implementations are possible)
- The **symmetric displacement** is scaled and replicated on each direction: $\delta_k = \{(\pm\sqrt{4\nu\Delta t}, 0), (0, \pm\sqrt{4\nu\Delta t})\}$

General features of the scheme

- **Consistent and stable** (stability proved for high-order space reconstructions and constant viscosity)

General features of the scheme

- **Consistent and stable** (stability proved for high-order space reconstructions and constant viscosity)
- **Explicit** (although **first-order**) treatment of the diffusion term, accuracy improving at high Reynolds numbers (the consistency error is $O(\Delta t/Re)$)

General features of the scheme

- **Consistent and stable** (stability proved for high-order space reconstructions and constant viscosity)
- **Explicit** (although **first-order**) treatment of the diffusion term, accuracy improving at high Reynolds numbers (the consistency error is $O(\Delta t/Re)$)
- **High-order** implementations for the hyperbolic part relatively easy to construct

General features of the scheme

- **Consistent and stable** (stability proved for high-order space reconstructions and constant viscosity)
- **Explicit** (although **first-order**) treatment of the diffusion term, accuracy improving at high Reynolds numbers (the consistency error is $O(\Delta t/Re)$)
- **High-order** implementations for the hyperbolic part relatively easy to construct
- **Large time steps** allowed with weak requirements on the $\Delta t/\Delta x$ relationship

Navier–Stokes Equations

Goal: integrating the SL advection–diffusion solver in a code for the incompressible NSE

$$\begin{cases} \boldsymbol{u}_t + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nu \Delta \boldsymbol{u} + \nabla p = 0 \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases}$$

Navier–Stokes Equations

Goal: integrating the SL advection–diffusion solver in a code for the incompressible NSE

$$\begin{cases} \boldsymbol{u}_t + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nu \Delta \boldsymbol{u} + \nabla p = 0 \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases}$$

This is typically achieved via intermediate formulations of the NSE:

Vorticity–Streamfunction formulation

- Suitable only **for the 2D case**
- Difficult and inaccurate treatment of **boundary conditions for ω**

Navier–Stokes Equations

Goal: integrating the SL advection–diffusion solver in a code for the incompressible NSE

$$\begin{cases} \boldsymbol{u}_t + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nu \Delta \boldsymbol{u} + \nabla p = 0 \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases}$$

This is typically achieved via intermediate formulations of the NSE:

Vorticity–Streamfunction formulation

- Suitable only **for the 2D case**
- Difficult and inaccurate treatment of **boundary conditions for ω**

Velocity–Pressure (projection) formulation

- Suitable for both **the 2D and the 3D case**
- Easier treatment of **boundary conditions for $\boldsymbol{u}$**

Navier–Stokes Equations: projection formulation

General idea: linearizing the NSE by neglecting the incompressibility constraint, then correcting the solution via a suitable pressure term

$$\begin{cases} \frac{\boldsymbol{u}^{n+1/2}-\boldsymbol{u}^n}{\Delta t} = -(\boldsymbol{u}^n \cdot \nabla)\boldsymbol{u}^n + \nu \Delta \boldsymbol{u}^n \\ -\Delta p^{n+1} = -\frac{1}{\Delta t} \nabla \cdot \boldsymbol{u}^{n+1/2} \\ \boldsymbol{u}^{n+1} = \boldsymbol{u}^{n+1/2} - \Delta t \nabla p^{n+1} \end{cases}$$

with proper BCs (typically, Dirichlet in the first step and homogeneous Neumann in the second for no-slip boundary)

Navier–Stokes Equations: projection formulation

General idea: linearizing the NSE by neglecting the incompressibility constraint, then correcting the solution via a suitable pressure term

$$\begin{cases} \frac{\boldsymbol{u}^{n+1/2}-\boldsymbol{u}^n}{\Delta t} = -(\boldsymbol{u}^n \cdot \nabla)\boldsymbol{u}^n + \nu \Delta \boldsymbol{u}^n \\ -\Delta p^{n+1} = -\frac{1}{\Delta t} \nabla \cdot \boldsymbol{u}^{n+1/2} \\ \boldsymbol{u}^{n+1} = \boldsymbol{u}^{n+1/2} - \Delta t \nabla p^{n+1} \end{cases}$$

with proper BCs (typically, Dirichlet in the first step and homogeneous Neumann in the second for no-slip boundary)

- **Critical operations**: advection–diffusion and Poisson solvers
- Time advancing is performed in the form of **fractional steps**
- Applicable to **3D problems**, easier treatment of **BCs for *u***
- Easier treatment of **unstructured geometries**
- Avoids systematic errors for the numerical divergence of ***u***

Navier–Stokes Equations: SL projection scheme

SL scheme

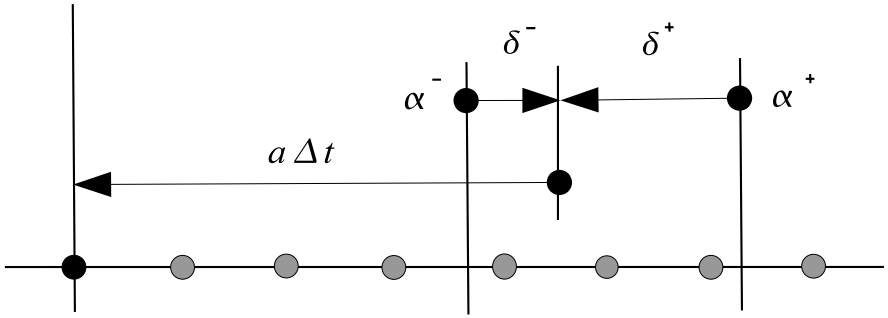
$$\begin{cases} \boldsymbol{u}_j^{n+1/2} = \frac{1}{4}\sum_{k=1}^{4} I[\boldsymbol{U}^n](x_j - \Delta t\, \boldsymbol{u}_j^n + \delta_k) & \text{SL advection–diffusion solver} \\ -\Delta p^{n+1} = -\frac{1}{\Delta t}\, \mathrm{div}_\Delta \boldsymbol{u}^{n+1/2} & \text{FD or other approximations} \\ \boldsymbol{u}^{n+1} = \boldsymbol{u}^{n+1/2} - \Delta t \nabla p^{n+1} \end{cases}$$

where $\delta_k = \{(\pm\sqrt{4\nu\Delta t}, 0), (0, \pm\sqrt{4\nu\Delta t})\}$ (in 2D)

A **first order** implementation can be easily constructed via built-in Matlab functions:

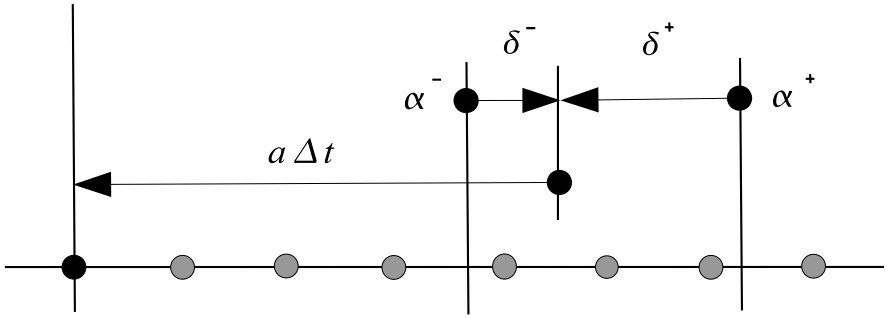
- **Monotonized cubic** interpolation for $I[\boldsymbol{U}^n]$ (structured case)
- **Triangle-based interpolation** from `PDEtoolbox` (unstructured case)
- **Sparse LU/Cholesky** solver for the Poisson eqn
- Centered FD (or FE discrete divergence) for div_Δ

Boundary conditions for velocity/vorticity (1)



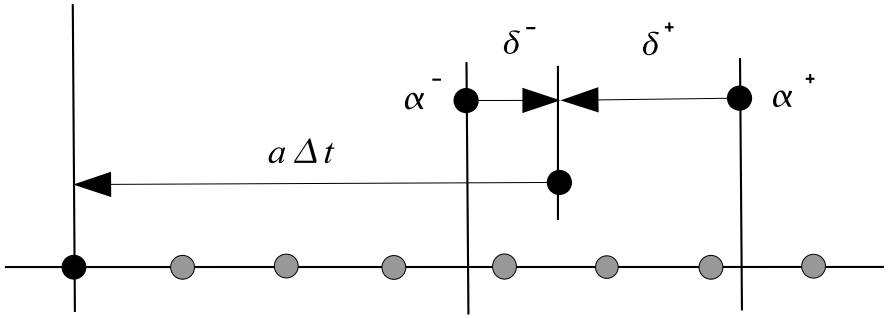
- It is possible to obtain a consistent scheme with **non-symmetric weights**, and this allows to re-define the scheme (although with **reduced consistency rate**) when close to the boundary

Boundary conditions for velocity/vorticity (1)



- It is possible to obtain a consistent scheme with **non-symmetric weights**, and this allows to re-define the scheme (although with **reduced consistency rate**) when close to the boundary
- The **local drop in consistency** (to order 1/2) occurs in a band of width $O(\sqrt{\Delta t})$, so that it **does not affect the global order**

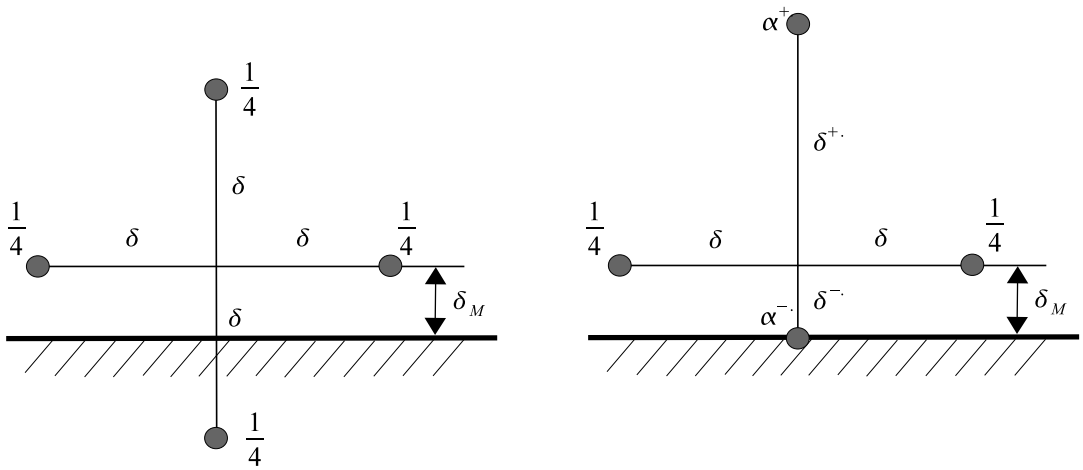
Boundary conditions for velocity/vorticity (1)



- It is possible to obtain a consistent scheme with **non-symmetric weights**, and this allows to re-define the scheme (although with **reduced consistency rate**) when close to the boundary
- The **local drop in consistency** (to order 1/2) occurs in a band of width $O(\sqrt{\Delta t})$, so that it **does not affect the global order**
- Consistency conditions:

$$\begin{cases} \alpha^+ + \alpha^- = 1/2 \\ \alpha^+\delta^+ - \alpha^-\delta^- = 0 \\ \alpha^+(\delta^+)^2 + \alpha^-(\delta^-)^2 = 2\nu\Delta t. \end{cases}$$

Boundary conditions for velocity/vorticity (2)



Using these further degrees of freedom on weights and displacements and the consistency conditions, we can then enforce

Dirichlet BCs via modified weights/displacements

$$\delta^- = \delta_M, \quad \delta^+ = \frac{4\Delta t \nu}{\delta_M}, \quad \alpha^- = \frac{1}{2}\frac{\delta^+}{\delta^+ + \delta^-}, \quad \alpha^+ = \frac{1}{2} - \alpha^-.$$

Lid-driven cavity for the NSE, projection scheme (1)

(Loading...)

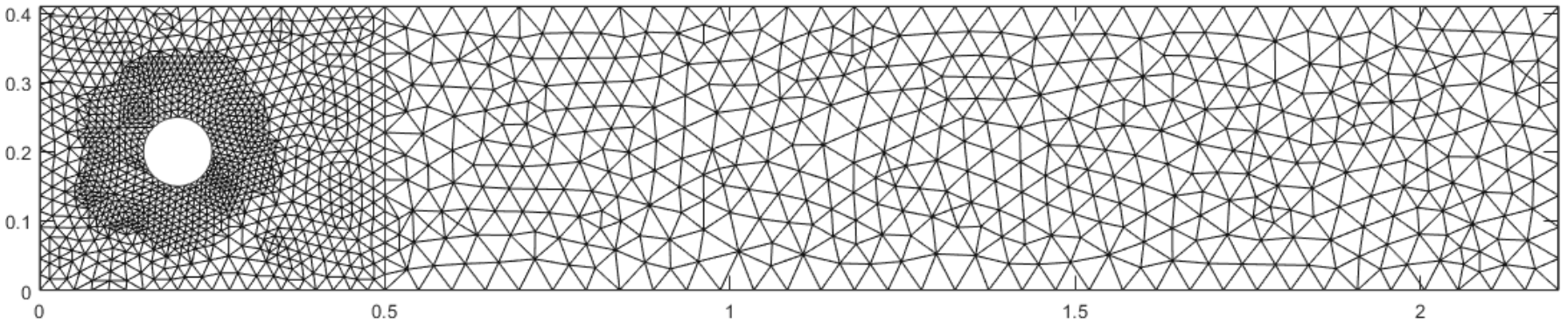
Lid-driven cavity at $Re = 1000$
200×200 nodes, Courant number ≈ 5

Lid-driven cavity for the NSE, projection scheme (2)

(Loading...)

Lid-driven cavity at $Re = 10000$
200×200 nodes, Courant number ≈ 5

Von Karman vortex street, projection scheme



(Loading...)

Von Karman vortex street at $Re = 100$, grid and solution

Evaluation of the scheme

Advantages

- Possibility of **(relatively) large Courant numbers**
- **Reduced complexity** for the advection–diffusion solver
- Accurate **transition between laminar and turbulent regime**

Evaluation of the scheme

Advantages

- Possibility of **(relatively) large Courant numbers**
- **Reduced complexity** for the advection–diffusion solver
- Accurate **transition between laminar and turbulent regime**

Bottlenecks

- Relatively **inaccurate treatment of BCs**
- Difficult to treat efficiently a **complicate geometry**

Evaluation of the scheme

Advantages

- Possibility of **(relatively) large Courant numbers**
- **Reduced complexity** for the advection–diffusion solver
- Accurate **transition between laminar and turbulent regime**

Bottlenecks

- Relatively **inaccurate treatment of BCs**
- Difficult to treat efficiently a **complicate geometry**

Improvements

- A **second-order** advection–diffusion solver
- **Higher order treatment of BCs**
- **Efficient point location strategies** for unstructured geometries
- **Scalability** on GPU architectures $\rightarrow$ C++ implementation

Second-order advection–diffusion solver (1)

A possible second-order improvement of the previous AD solver is borrowed from the literature on approximation schemes for Stochastic Differential Equations, and is a **stochastic version of the Crank–Nicolson scheme**:

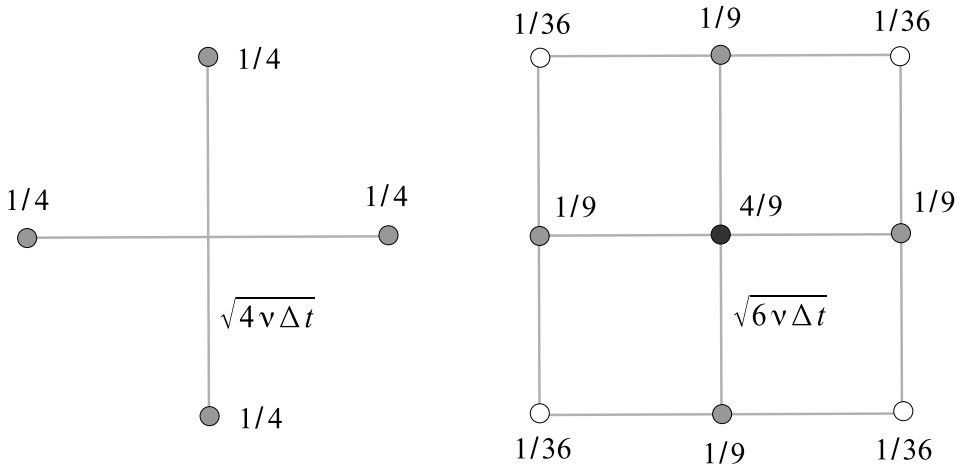
Stochastic Crank–Nicolson scheme

$$z_{k,i}^{n+1} = x_i - \frac{\Delta t}{2}\left(\boldsymbol{u}(x_i, t_{n+1}) + \boldsymbol{u}(z_{k,i}^{n+1}, t_n)\right) + \delta_k.$$

- In order to obtain second-order consistency, advection and diffusion operators **cannot any longer be considered as decoupled**.
- In this case, the increase in the order of approximation requires that moments of the probability density of $\sqrt{2\nu}\Delta W$ **are reproduced by the discrete density up to the fifth moment**. This motivates the introduction of further displacements and weights.

Second-order advection–diffusion solver (2)

Passing from first- to second-order, the displacements δ_k and associated weights are modified as

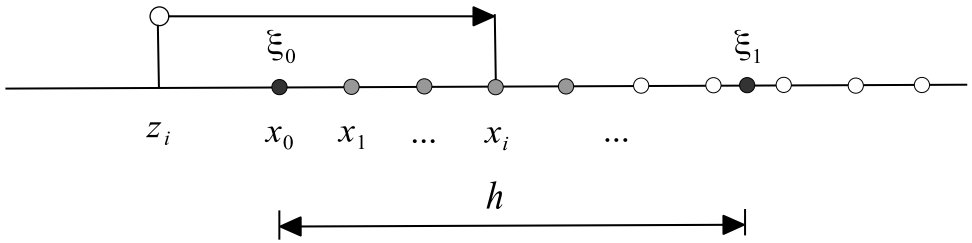


first order second order

- **Consistency analysis** for the resulting SL scheme is conceptually simple, but very technical
- **Well scalable** on SIMD architectures

High-order treatment of boundary conditions (1)

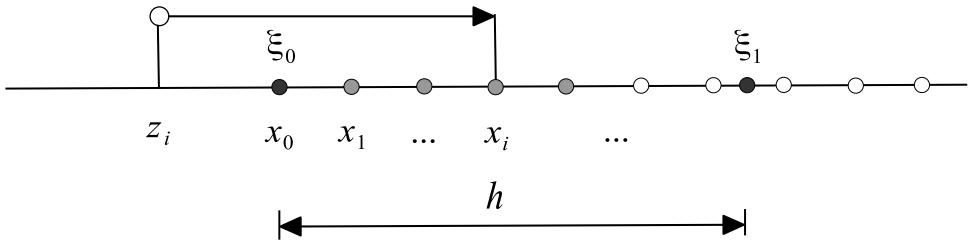
Boundary conditions are a typical bottleneck of SL schemes. To **avoid the loss of consistency** outlined above, we can compute the value at the foot of a characteristic by **extrapolation** [BCCF]



- To obtain a **stable extrapolation**, it should be performed on a **coarser grid** (of space step $h > \Delta x$)

High-order treatment of boundary conditions (1)

Boundary conditions are a typical bottleneck of SL schemes. To **avoid the loss of consistency** outlined above, we can compute the value at the foot of a characteristic by **extrapolation** [BCCF]

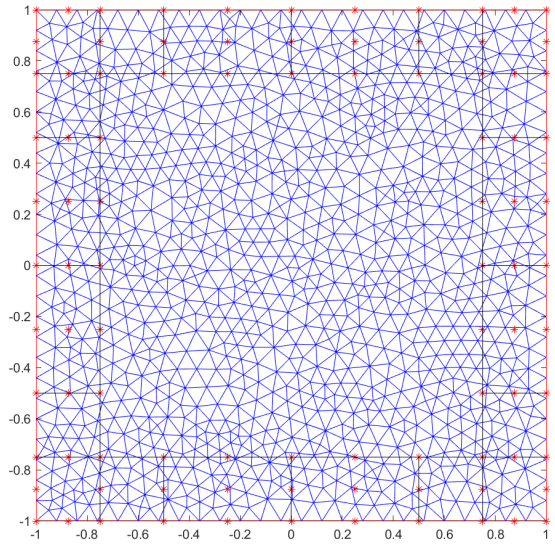


- To obtain a **stable extrapolation**, it should be performed on a **coarser grid** (of space step $h > \Delta x$)
- The analysis of this procedure provides a **stable region of thickness $O(h)$** outside of the boundary, where extrapolation is allowed:

$$d(z_{k,i}^{n+1}, \Omega) < C(n_{ex})h$$

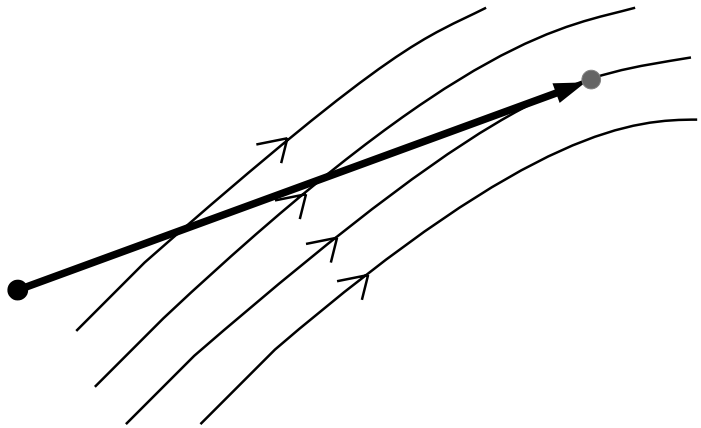
- A **careful coupling** between h and Δx is required to keep both stability and consistency rate

High-order treatment of boundary conditions (2)



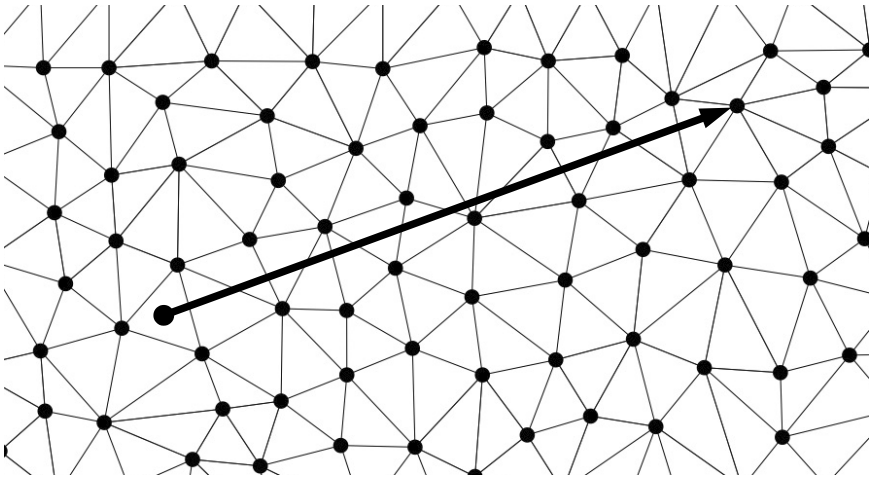
- The **extrapolation grid** ($\mathbb{Q}_2$ in this example) is constructed near the boundary
- The coupling between $\mathbb{P}_2$ internal interpolation and $\mathbb{P}_2/\mathbb{Q}_2$ extrapolation results in a **second-order treatment of BCs**
- **Not easy** to obtain a general-purpose implementation on **genuinely unstructured geometries**

Upwinding and point location



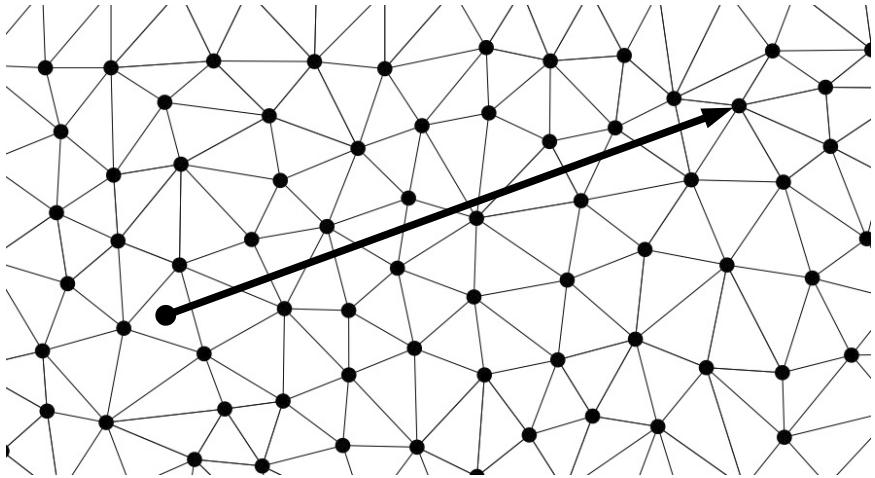
- An (approximate) **upwinding** is performed along the advection flow lines (higher order implementations are possible)

Upwinding and point location



- An (approximate) **upwinding** is performed along the advection flow lines (higher order implementations are possible)
- A local interpolation at the foot of a characteristic requires to **locate the point** on the space grid

Upwinding and point location



- An (approximate) **upwinding** is performed along the advection flow lines (higher order implementations are possible)
- A local interpolation at the foot of a characteristic requires to **locate the point** on the space grid
- On structured grids, the point location has $O(1)$ complexity
- On unstructured grids, the element containing the point should be located **using adjacency information** (no direct access)

Algorithms for (efficient) point location

The fast location of a point on an unstructured triangular/tetrahedral grid is a classical problem in computational geometry. Two major solutions are available:

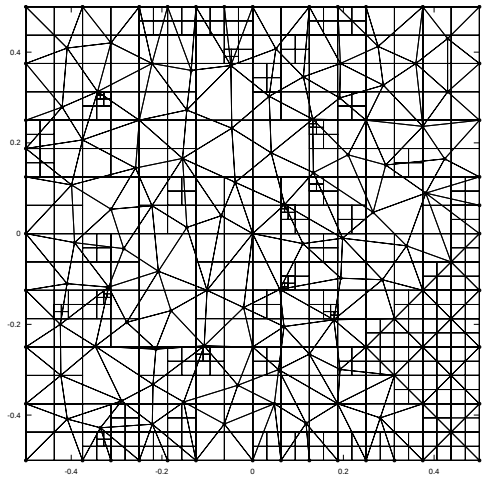
- **Quadtree/octree algorithms**: the computational domain is structured in a tree, and the search requires to visit the tree.
  - Complexity of the visit is **logarithmic**
  - Requires to build an **additional data structure** with **additional memory occupation**

Algorithms for (efficient) point location

The fast location of a point on an unstructured triangular/tetrahedral grid is a classical problem in computational geometry. Two major solutions are available:

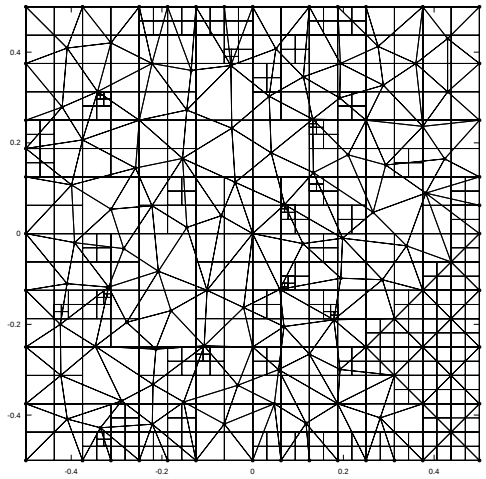
- **Quadtree/octree algorithms**: the computational domain is structured in a tree, and the search requires to visit the tree.
  - Complexity of the visit is **logarithmic**
  - Requires to build an **additional data structure** with **additional memory occupation**
- **Walking algorithms**: the algorithm walks along the computational domain towards the final element.
  - Complexity of the visit **depends on the distance between the starting and the final element** ($O(\sqrt[d]{\text{number of elements}})$ in the worst case, with d=dimension)
  - **No relevant additional data structure** required

Quadtree(/octree) algorithms



In this algorithm, a **tree structure** is associated to the mesh, starting from **a rectangle containing all the mesh**. A rectangle is refined into **four subrectangles** until:

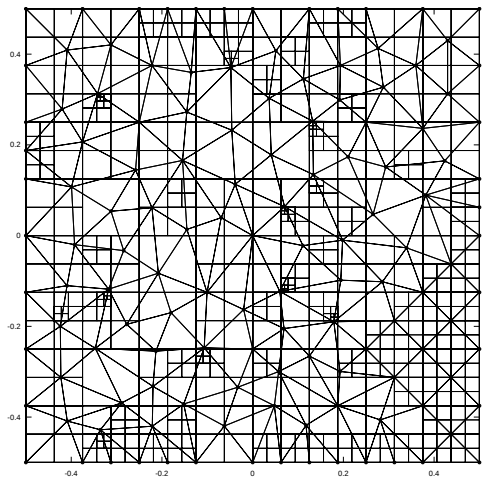
Quadtree(/octree) algorithms



In this algorithm, a **tree structure** is associated to the mesh, starting from **a rectangle containing all the mesh**. A rectangle is refined into **four subrectangles** until:

- It does not intersect the mesh, or

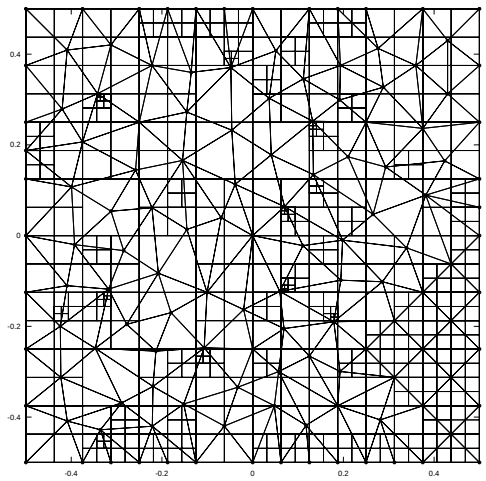
Quadtree(/octree) algorithms



In this algorithm, a **tree structure** is associated to the mesh, starting from **a rectangle containing all the mesh**. A rectangle is refined into **four subrectangles** until:

- It does not intersect the mesh, or
- It intersects a number of triangles $n_t \in [1, q]$ ($q = 3$ in the example above) and no vertex, or

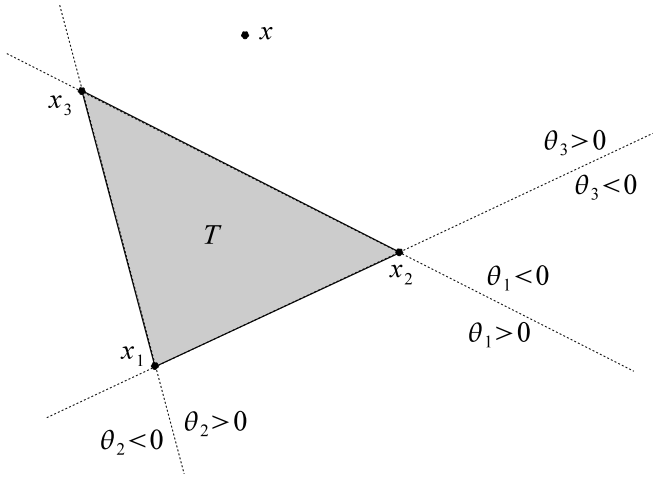
Quadtree(/octree) algorithms

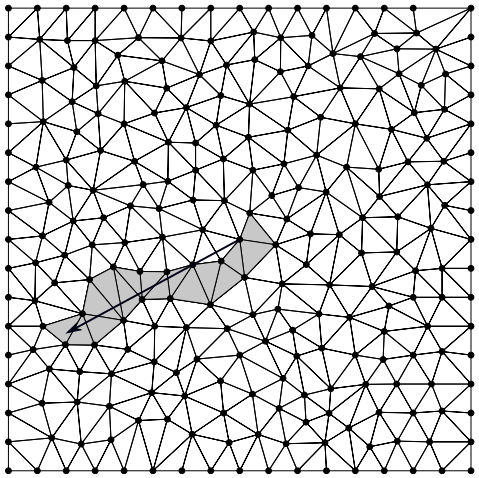


In this algorithm, a **tree structure** is associated to the mesh, starting from **a rectangle containing all the mesh**. A rectangle is refined into **four subrectangles** until:

- It does not intersect the mesh, or
- It intersects a number of triangles $n_t \in [1, q]$ ($q = 3$ in the example above) and no vertex, or
- It contains one vertex, regardless of the number of triangles

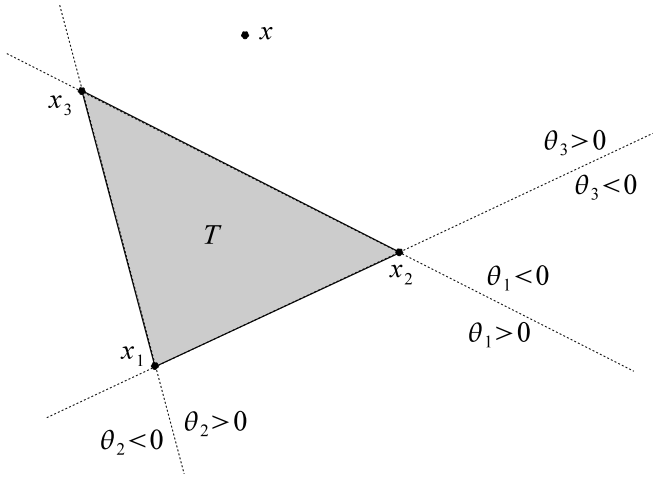
Walking algorithms (1)

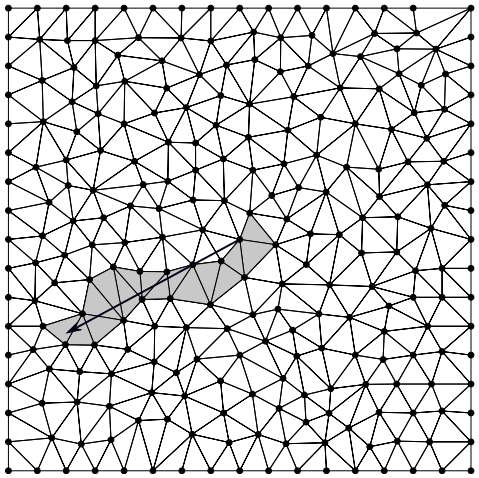




We will use here the **barycentric walk**. In this algorithm, moving from **an initial element**:

Walking algorithms (1)

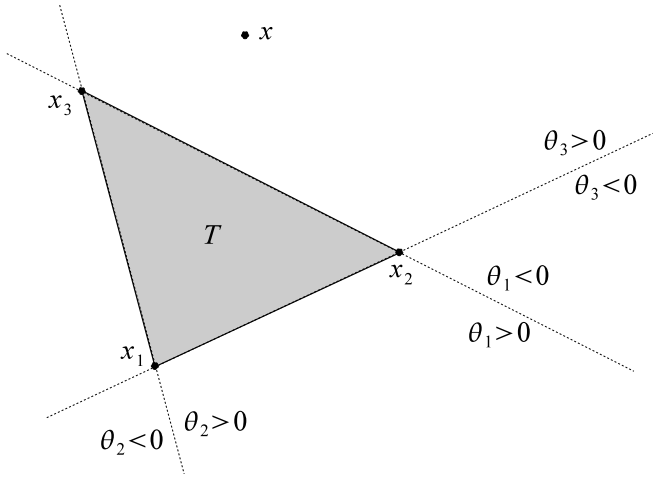


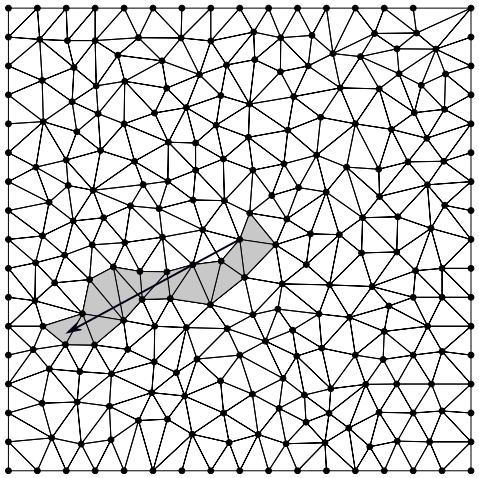


We will use here the **barycentric walk**. In this algorithm, moving from **an initial element**:

- If the barycentric coordinates w.r.t. the current element are **all positive**, then the element contains the point; otherwise

Walking algorithms (1)





We will use here the **barycentric walk**. In this algorithm, moving from **an initial element**:

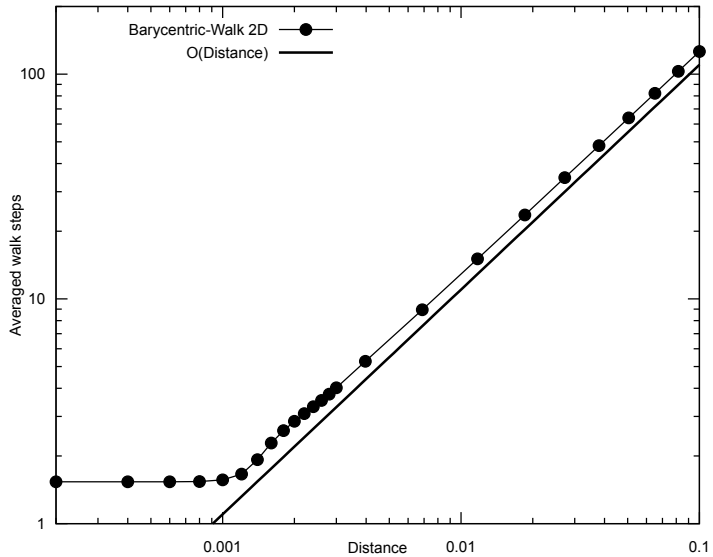
- If the barycentric coordinates w.r.t. the current element are **all positive**, then the element contains the point; otherwise
- If (at least) **one barycentric coordinate is negative**, then the current element is updated by stepping to the element which faces the point associated to **the negative barycentric coordinate of largest magnitude**

Walking algorithms (2)

- In a **preliminary phase**, it might be necessary to construct the **matrix of adjacency between elements**
- The memory occupation for this structure is **not critical**

Walking algorithms (2)

- In a **preliminary phase**, it might be necessary to construct the **matrix of adjacency between elements**
- The memory occupation for this structure is **not critical**
- Each point location requires a walk **from the initial to the final element**. On a regular Delaunay triangulation with N elements, the complexity is asymptotically linear w.r.t. the distance ($O(\sqrt{N})$ in the worst case)



Speed-up strategies for BW

Comparison of quadtree (QT) versus barycentric walk (BW) strategies:

Quadtree

- Best complexity for general point location problems – however, slower than direct access (structured mesh)
- Difficult to code
- Requires relevant additional memory

Speed-up strategies for BW

Comparison of quadtree (QT) versus barycentric walk (BW) strategies:

Quadtree

- Best complexity for general point location problems – however, slower than direct access (structured mesh)
- Difficult to code
- Requires relevant additional memory

Barycentric walk

- Complexity depends on initialization, slower in the general case
- Easy to code
- Requires little additional memory

Speed-up strategies for BW

Comparison of quadtree (QT) versus barycentric walk (BW) strategies:

Quadtree

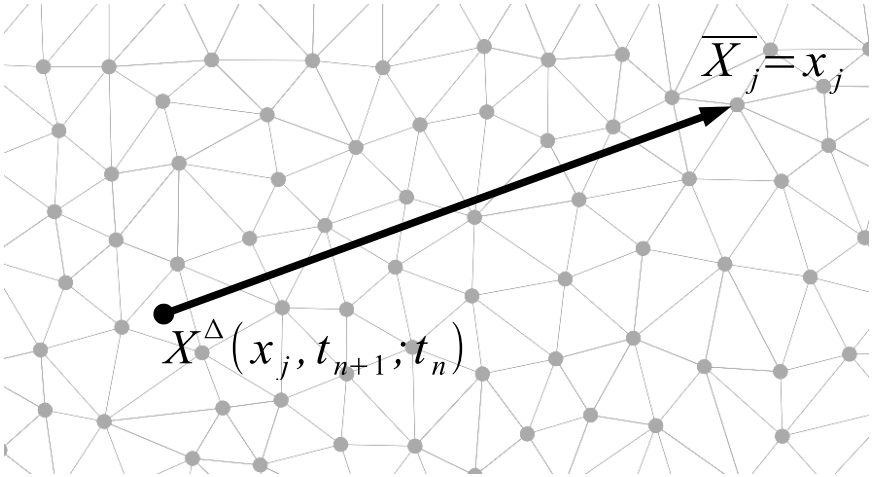
- Best complexity for general point location problems – however, slower than direct access (structured mesh)
- Difficult to code
- Requires relevant additional memory

Barycentric walk

- Complexity depends on initialization, slower in the general case
- Easy to code
- Requires little additional memory

This motivates the search for a **clever choice** of the initial point $\overline{X}_j$ to initialize the BW (in practice, we can obtain $O(1)$ complexity [CF21])

Speed-up strategy A: following characteristics

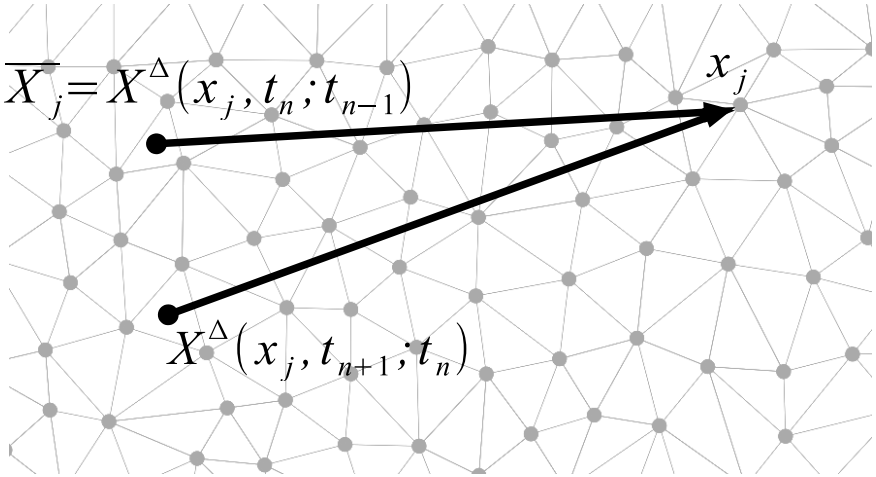


The easiest choice is to start the walk **from the node x_j itself**:

$$\overline{X}_j = x_j.$$

- Already used, in particular in combination with **substepping** along trajectories
- Sensitive to an increase of the Courant number

Speed-up strategy B: using the previous time step

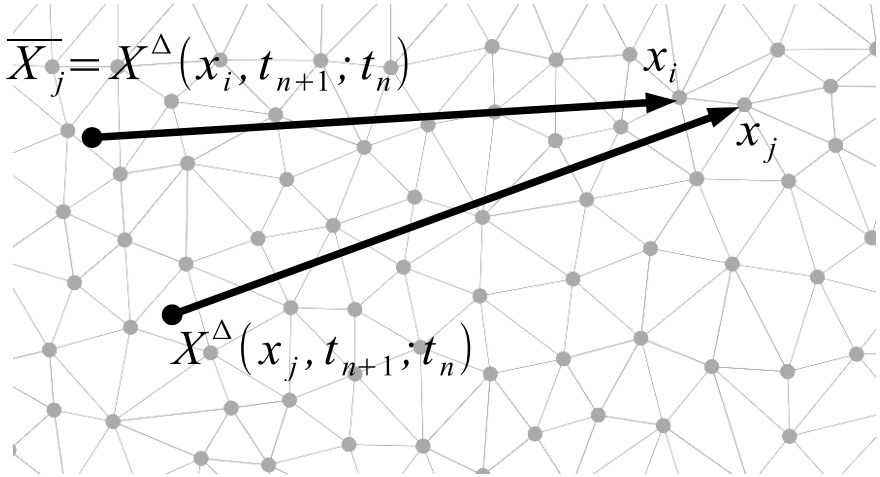


The second choice is to start the walk **from the foot of the characteristic at the previous time step**:

$$\overline{X}_j = X^{\Delta}(x_j, t_n; t_{n-1}).$$

- Provides the best guess in case of **stationary advection terms**
- Less sensitive to an increase of the Courant number
- Relatively **scalable on SIMD architectures**

Speed-up strategy C: using a neighbouring node



The third choice is to start the walk from the foot of the characteristic **of a neighbouring node x_i, at the same time step**:

$$\overline{X}_j = X^{\Delta}(x_i, t_{n+1}; t_n).$$

- Requires to build a **spanning tree** of the grid
- Insensitive to an increase of the Courant number
- Intrinsically **serial**

Evaluation of the various strategies (1)

Once verified that complexity of the BW is **linear w.r.t. the distance**, i.e.,

$$\mathcal{O}\left(C_1 + C_2 \frac{\|X^{\Delta}(x_i, t_{n+1}; t_n) - \overline{X}_i\|}{\Delta x}\right),$$

then the various strategies outlined result in a complexity

Evaluation of the various strategies (1)

Once verified that complexity of the BW is **linear w.r.t. the distance**, i.e.,

$$\mathcal{O}\left(C_1 + C_2 \frac{\|X^{\Delta}(x_i, t_{n+1}; t_n) - \overline{X}_i\|}{\Delta x}\right),$$

then the various strategies outlined result in a complexity

(A) $\mathcal{O}\left(C_1 + C_2 \frac{\|f(x_i, t_{n+1})\| \Delta t}{\Delta x}\right)$

- Heavy dependence on the (local) Courant number

Evaluation of the various strategies (1)

Once verified that complexity of the BW is **linear w.r.t. the distance**, i.e.,

$$\mathcal{O}\left(C_1 + C_2 \frac{\|X^{\Delta}(x_i, t_{n+1}; t_n) - \overline{X}_i\|}{\Delta x}\right),$$

then the various strategies outlined result in a complexity

(A) $\mathcal{O}\left(C_1 + C_2 \frac{\|f(x_i, t_{n+1})\| \Delta t}{\Delta x}\right)$

- Heavy dependence on the (local) Courant number

(B) $\mathcal{O}\left(C_1 + C_2 L_t \frac{\Delta t^2}{\Delta x}\right)$

- Light dependence on the (local) Courant number
- Takes advantage of (locally) stationary advection terms

Evaluation of the various strategies (1)

Once verified that complexity of the BW is **linear w.r.t. the distance**, i.e.,

$$\mathcal{O}\left(C_1 + C_2 \frac{\|X^{\Delta}(x_i, t_{n+1}; t_n) - \overline{X}_i\|}{\Delta x}\right),$$

then the various strategies outlined result in a complexity

(A) $\mathcal{O}\left(C_1 + C_2 \frac{\|f(x_i, t_{n+1})\| \Delta t}{\Delta x}\right)$

- Heavy dependence on the (local) Courant number

(B) $\mathcal{O}\left(C_1 + C_2 L_t \frac{\Delta t^2}{\Delta x}\right)$

- Light dependence on the (local) Courant number
- Takes advantage of (locally) stationary advection terms

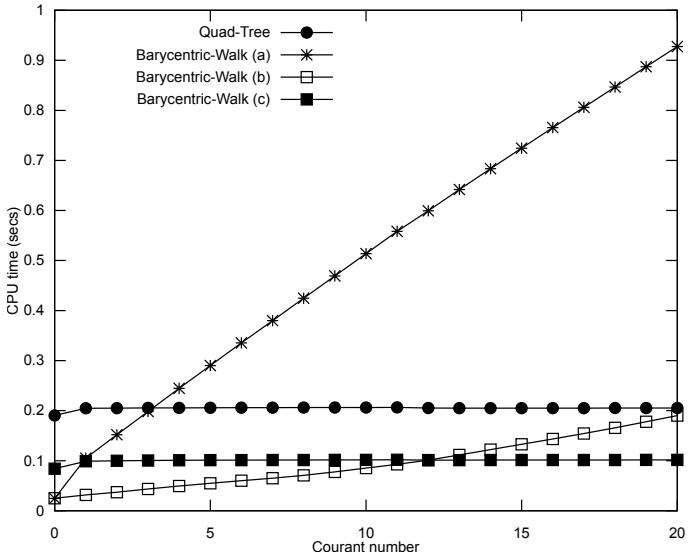
(C) $\mathcal{O}\left((C_1 + C_2) + C_2 L_x \Delta t\right)$

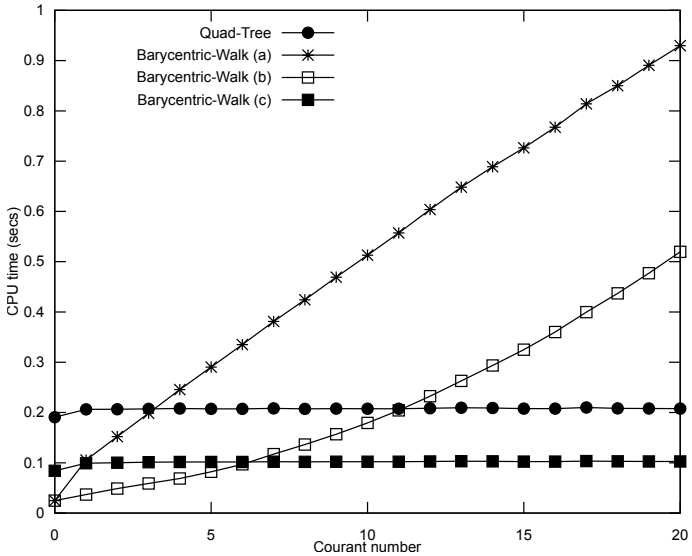
- No dependence on the (local) Courant number
- Takes advantage of (locally) constant in space advection terms

In all three cases, we obtain $\boldsymbol{O(1)}$ **cost under linear refinements**

Evaluation of the various strategies (2)

Typical behaviour of the various strategies in term of CPU time, for varying time Lipschitz constant and as a function of the Courant number:

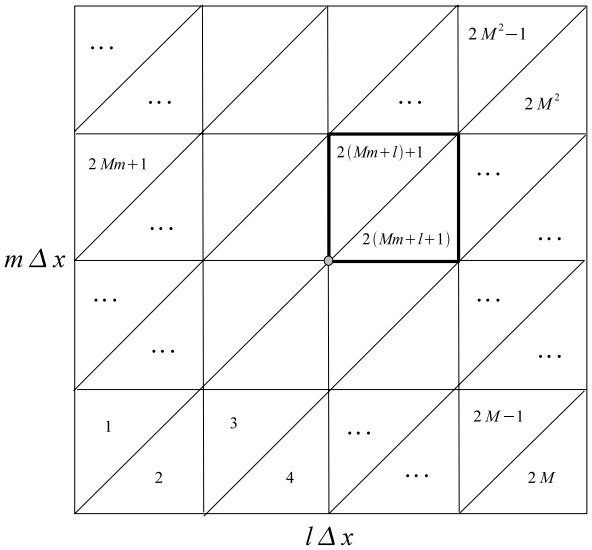


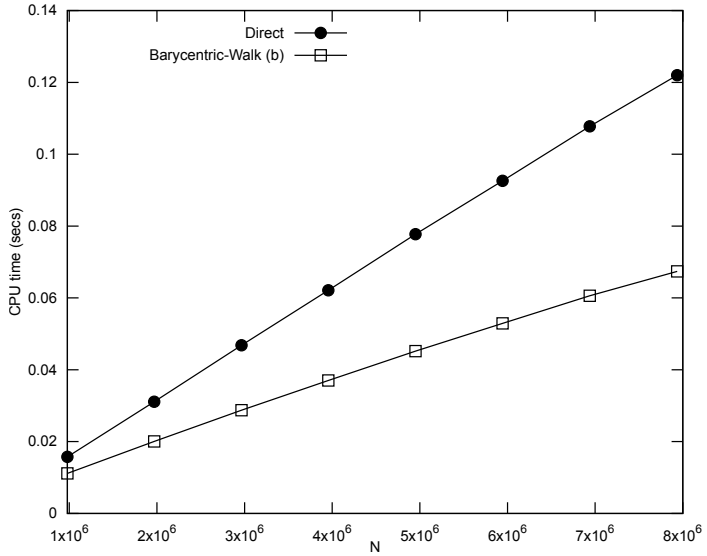


- Strategy A eventually becomes the worst choice at the increase of the Courant number
- Strategy B always performs the best for low or moderate Courant numbers
- Strategy C always performs the best for large Courant numbers

Evaluation of the various strategies (3)

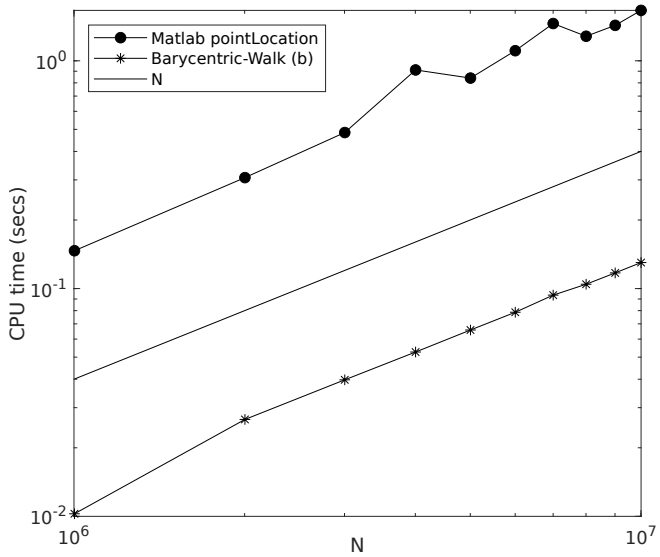
At a comparison with a **structured triangulation**, the walking strategy (of type B in this case) **may even turn to be faster**. It is to be remarked that the computation of barycentric coordinates is performed **in both cases to obtain the interpolate of the solution**:





Evaluation of the various strategies (4)

Last, we perform a comparison of the search strategy B with the general-purpose **Matlab function** `pointLocation`:



To obtain a fair evaluation, the comparison is performed by replacing `pointLocation` with a Mex-compiled version of the BW, resulting in **a speedup of 10–20 times**.

3D + Boundary conditions

(Loading...)

- Straightforward adaptation of the BW strategies to the 3D case
- However, BW may need to switch from an internal walk to a boundary walk and vice versa

CUDA parallelization

- On a SIMD architecture, we can set in parallel form:
  - Computation of feet of characteristics
  - Point location on the grid
  - Interpolation
- A careful parallelization allows to obtain relevant speed-ups with a relatively simple and unexpensive architecture

CUDA parallelization

- On a SIMD architecture, we can set in parallel form:
  - Computation of feet of characteristics
  - Point location on the grid
  - Interpolation
- A careful parallelization allows to obtain relevant speed-ups with a relatively simple and unexpensive architecture
- **Test**: plain advection, time-dependent transport field and boundary conditions on the cylinder $(0, 2) \times B(0, 1)$.

$$\boldsymbol{u}(x, y, z, t) = (0.5, 0.25\cos(2\pi t), 0.25\sin(\pi t))$$

- **Scheme**: inviscid SL, point location via strategy B, $\mathbb{P}_2$ interpolation
  - Space grids from about 1M to about 9M elements
  - Average Courant numbers 4 to 8; number of time steps 16 to 64
- **Hardware**: Intel i9–9900K, 16 cores, 3.6 GHz, 32 GB RAM; Nvidia RTX–2080ti

CUDA parallelized code

(Loading...)

- Maximum serial CPU time about 300s, maximum CPU time in parallel mode about 3.7s
- Speed-ups ranging from 70x to 93x, higher for larger Courant numbers and refined grids

A preliminary 2D test

(Loading...)

- Airfoil at $Re = 5000$
- Barycentric walk-based point location, Chorin–Temam scheme, $\mathbb{P}_2$ interpolation

Conclusions

- Explicit SL advection–diffusion solvers seem a viable alternative to reduce the complexity of more classical solvers in NSE codes

Conclusions

- Explicit SL advection–diffusion solvers seem a viable alternative to reduce the complexity of more classical solvers in NSE codes
- Various drawbacks of this class of schemes have been removed, or at least reduced

Conclusions

- Explicit SL advection–diffusion solvers seem a viable alternative to reduce the complexity of more classical solvers in NSE codes
- Various drawbacks of this class of schemes have been removed, or at least reduced
- First-order schemes work nicely, second-order in progress

Conclusions

- Explicit SL advection–diffusion solvers seem a viable alternative to reduce the complexity of more classical solvers in NSE codes
- Various drawbacks of this class of schemes have been removed, or at least reduced
- First-order schemes work nicely, second-order in progress
- GPU-based 3D implementation in progress, too – promising results so far

References

L. Bonaventura, E. Calzola, E. Carlini, R. Ferretti, Second-order fully semi-Lagrangian discretizations of advection–diffusion–reaction systems, J. Sci. Comp., **88** (2021), 1–29

L. Bonaventura, R. Ferretti and L. Rocchi, *A fully semi-Lagrangian discretization or the 2D Navier–Stokes equations in the vorticity–streamfunction formulation*, Appl. Math. Comp., **323** (2018), 132 – 144

S. Cacace, R. Ferretti, *Efficient implementation of characteristic-based schemes on unstructured triangular grids*, Comp. Appl. Math., to appear.

F. Camilli and M. Falcone, *An approximation scheme for the optimal control of diffusion processes*, M^2AN Math. Model. Numer. Anal., **29** (1995), 97 – 122

O. Devillers, S. Pion, M. Teillaud, *Walking in a triangulation*, Proceedings of the seventeenth annual symposium on Computational geometry (2001), 106–114.

M. Falcone and R. Ferretti, Semi-Lagrangian approximation schemes for linear and Hamilton–Jacobi equations, SIAM, Philadelphia, 2013

R. Ferretti, *A technique for the high-order treatment of diffusion terms in semi-Lagrangian schemes*, Commun. Comp. Phys. **8** (2010), 445 – 470

R.A. Finkel, J.L. Bentley, *Quad trees a data structure for retrieval on composite keys*, Acta Informatica, **4** (1974), 1–9.

H. Kushner and P. Dupuis, Numerical methods for stochastic control problems in continuous time, Springer, New York, 2001

Triangle, A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator, `https://www.cs.cmu.edu/~quake/triangle`.