Vortex-In-Cell method for fluid dynamic simulations

Aleixandre, J., Hering, D., Hutter, A., López, C., Meier, J.

ETH Zürich

May 31, 2024

Outline

1. Theoretical background
2. Methodology
3. Test cases
4. IPPL implementation

Theoretical background (I)

Objective

This project aims to implement the Vortex-in-Cell (VIC) method in the performance portable C++ library IPPL, both in 2D and 3D schemes.

Starting from the Navier-Stokes equation for an incompressible viscous fluid, one can use the incompressibility condition of the velocity field ($\nabla \cdot \vec{u} = 0$) as well as the definition of the vorticity field ($\vec{\omega} := \nabla \times \vec{u}$) to obtain the corresponding equation in the so-called velocity-vorticity formulation:

$$\boxed{\frac{\partial \vec{\omega}}{\partial t} + (\vec{u} \cdot \nabla)\,\vec{\omega} = (\vec{\omega} \cdot \nabla)\,\vec{u} + \nu \Delta \vec{\omega}}.$$

Thereby, ν is the kinematic viscosity.

Theoretical background (II)

The VIC scheme is a hybrid method because it tracks the evolution of the vortices as moving particles (Lagrangian frame) while solving the field equations on a fixed Cartesian grid (Eulerian frame). In this Eulerian-Lagrangian frame, the time evolution of the vortex is given by

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{u}\,,$$
$$\frac{\mathrm{d}\vec{\omega}}{\mathrm{d}t} = (\vec{\omega} \cdot \nabla)\,\vec{u} + \nu \Delta \vec{\omega}\,.$$

Note that on the right-hand side of the last equation, the first term vanishes for 2D schemes, whereas the second term is equal to zero for inviscid regimes.

General overview of the VIC method

In the following, we consider inviscid regimes, i.e. , with $\nu = 0$. The VIC scheme consists of the following steps (steps for the 3D scheme):

- ⋆ Generate an initial distribution of vortex positions and strengths.
- ⋆ Interpolate the vortex strengths to the grid: $\vec{\omega}_\alpha \mapsto \vec{\omega}_i$.
- ⋆ Calculate the stream function on the grid: $\omega_i \mapsto A_i$.
- ⋆ Calculate the velocity values on the grid: $A_i \mapsto u_i$.
- ⋆ Calculate the vortex stretching term on the grid: $(\vec{u}_i, \vec{\omega}_i) \mapsto \delta\vec{\omega}_i$.
- ⋆ Interpolate back the velocity values to the vortices: $\vec{u}_i \mapsto \vec{u}_\alpha$.
- ⋆ Interpolate back the vortex stretching term to the vortices: $\delta\vec{\omega}_i \mapsto \delta\vec{\omega}_\alpha$.
- ⋆ Push the vortices forward in time: $\vec{x}_\alpha^{(n)} \mapsto \vec{x}_\alpha^{(n+1)}$.
- ⋆ Push the vorticity values forward in time: $\vec{\omega}_\alpha^{(n)} \mapsto \vec{\omega}_\alpha^{(n+1)}$.

Numerical methods (I)

Generation of **initial conditions**:

- ⋆ $N_p \sim \mathcal{O}\left(L^2\right)$
- ⋆ Deterministic: one per grid point.
- ⋆ Random: particles generated via uniformly distributed random sampling

Interpolation: CIC interpolation kernel → the fields are weighted based on the relative area (in 3D, volume) between the particle and the corresponding grid point.

Calculation of the **stream function** on the grid: each component of the vector $\vec{A}$ is obtained by solving the Poisson equation

$$\Delta A_i = -\omega_i \,.$$

Since we use periodic boundary conditions for our simulation domain, we opted for a fast Poisson solver.

Numerical methods (II)

Calculation of the **velocity** on the grid: from the stream function $\rightarrow \vec{u} = \nabla \times \vec{A}$. This is solved with a central difference FD scheme. In 2D, it reads

$$\vec{u}(j,k) = \left(\frac{A(j,k+1) - A(j,k-1)}{2h_y}, -\frac{A(j+1,k) - A(j-1,k)}{2h_x} \right).$$

Calculation of the **vortex stretching term** on the grid (only in 3D): is defined component-wise as

$$\delta\vec{\omega}_i = (\vec{\omega} \cdot \nabla)\,\vec{u} = (\omega_x \partial_x + \omega_y \partial_y + \omega_z \partial_z)\,\vec{u}\,,$$

and is also calculated by means of a central difference FD scheme.

Numerical methods (III)

Updating the **positions** of the particles: considering a set of even and odd time steps results in a Leapfrog-type scheme so that the particles are pushed according to

$$\vec{x}_{\alpha}^{(n+1)} = \vec{x}_{\alpha}^{(n-1)} + \vec{u}_{\alpha}^{(n)}\left(\vec{x}_{\alpha}^{(n)}\right) \cdot 2\Delta t \,.$$

Updating the **vorticity** values (only in 3D): this is done by a simple (explicit) Euler step $\rightarrow \vec{\omega}_{\alpha}^{(n+1)} = \vec{\omega}_{\alpha}^{(n)} + \Delta t \cdot \delta\vec{\omega}_{i}^{(n)}$.

Test cases and results (I)

Geometries in 2D:

- Disk, single band, double band, concentric rings, jet streams.
- These define the vortex within the physical domain → particles within these regions are assigned an initial non-zero vorticity value.

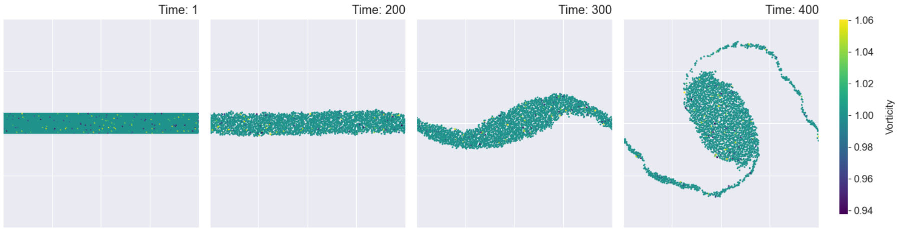
Extension to 3D:

- In principle, all geometries can be extended to 3D.
- A natural extension for the disk is a sphere.
- An alternative to extend these geometries is to consider several vortex sheets (along the z-direction), each one with the corresponding 2D geometry.

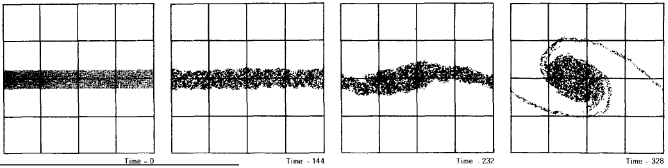
Vorticity distributions: uniform and Gaussian distributions have been tested.

Test cases and results (II)

- Single band (deterministic, uniform with noise):



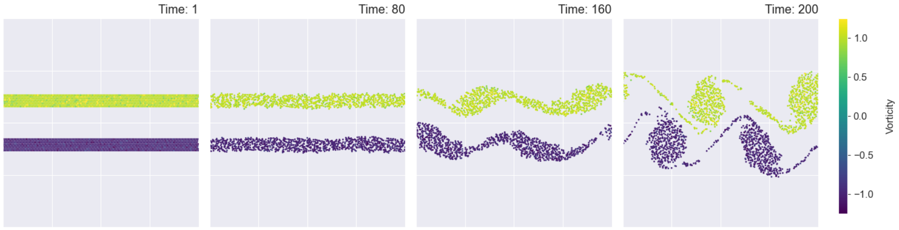
- Single band results by Christiansen[1]



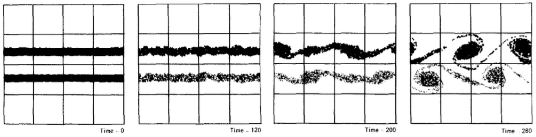
[1] I. Christiansen, "Numerical simulation of hydrodynamics by the method of point vortices", Journal of Computational Physics 13, 363 (1973).

Test cases and results (III)

- Double band (deterministic, uniform with noise):

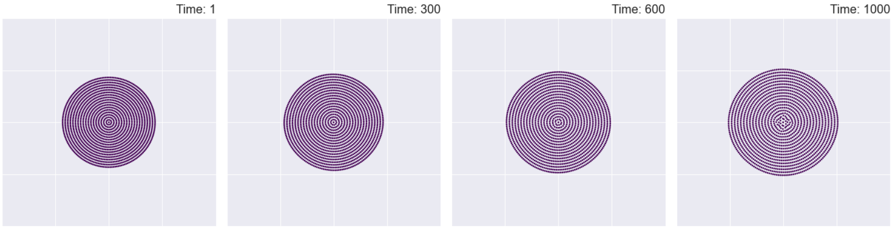


- Double band results by Christiansen



Test cases and results (IV)

- Concentric rings (deterministic, uniform):



- Reference:

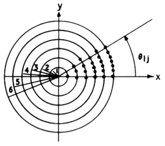
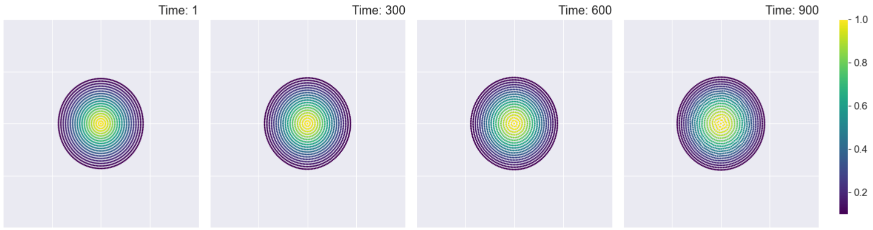


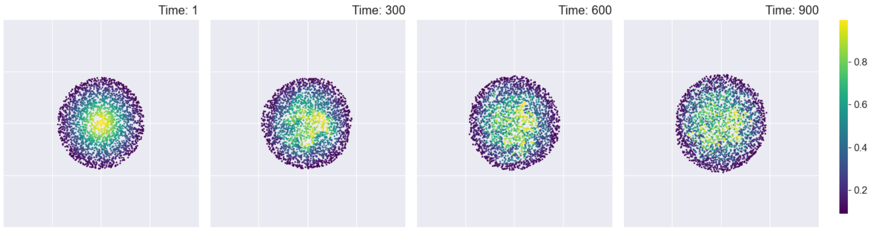
FIG. 2. Point vortices arranged to simulate Rankine's combined vortex.

Test cases and results (V)

- Gaussian disk (deterministic, gaussian):

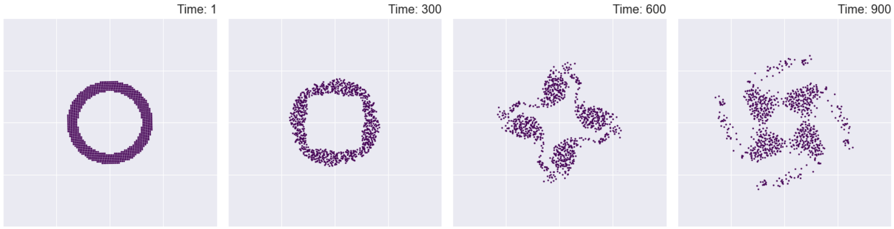


- Gaussian disk (random, gaussian):

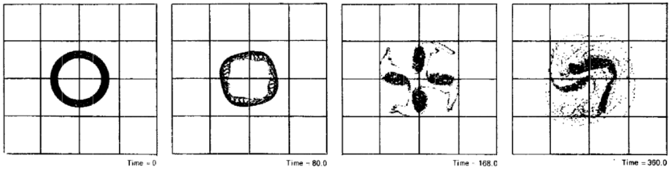


Test cases and results (VI)

- Ring (deterministic, uniform):

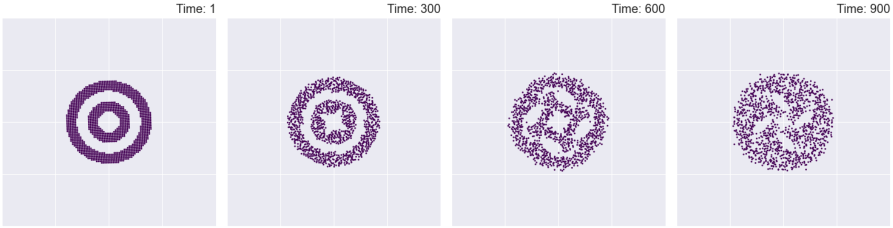


- Ring results by Christiansen

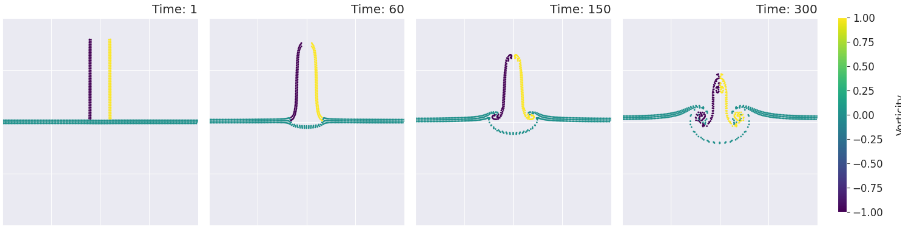


Test cases and results (VII)

- Concentric rings (deterministic, uniform):

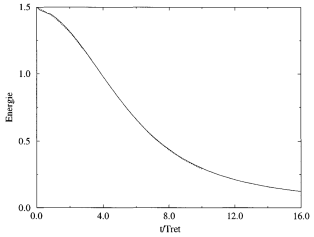


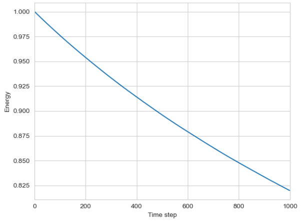
- Jet streams (deterministic, uniform):



Test cases and results (VIII)

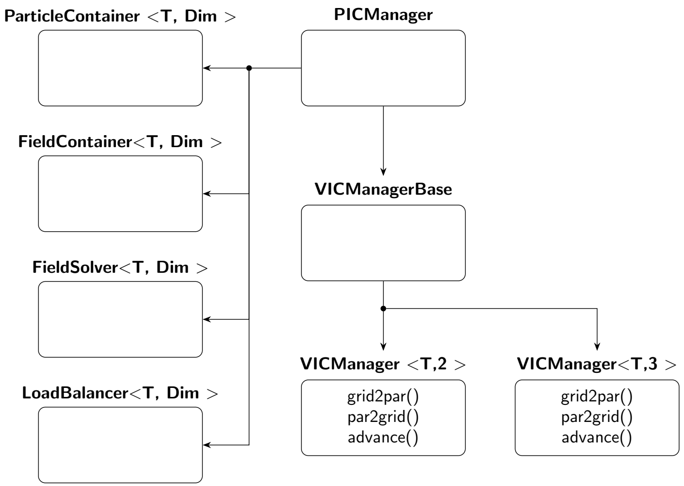
Quantitative tests: for an incompressible inviscid fluid, the total circulation, linear momentum, angular momentum, and kinetic energy should be dynamic invariants. According to Cottet et al.[2], the transfer (i.e., interpolation) between particles and grid introduces numerical dissipation that leads to losses in the energy. This is further motivated by the shear effects due to time discretization. Qualitatively, it increases the vortex area, as it was shown by Kudela et al., which in order to keep the circulation constant, results in a decrease in the velocities.



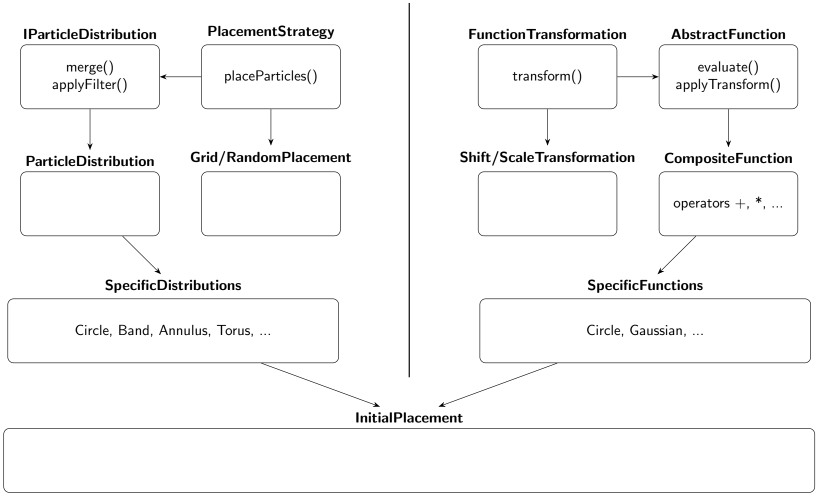


[2] G.-H. Cottet, P. D. Koumoutsakos, et al., Vortex methods: theory and practice, Vol. 313 (Cambridge University Press, Cambridge, 2000).

The VIC scheme in IPPL (I)



The VIC scheme in IPPL (II)


IParticleDistribution
merge()
applyFilter()
PlacementStrategy
placeParticles()
ParticleDistribution
Grid/RandomPlacement
SpecificDistributions
Circle, Band, Annulus, Torus, ...
FunctionTransformation
transform()
AbstractFunction
evaluate()
applyTransform()
Shift/ScaleTransformation
CompositeFunction
operators +, *, ...
SpecificFunctions
Circle, Gaussian, ...
InitialPlacement