

# CFD Code Development Frameworks

Python, C, OpenFOAM, Fenics

Dr. Rodolfo Monico, Taher Chegini, M. Sarraf Josaghani

CTFM 2018  
University of Houston

*rostilla@uh.edu*  
*tchegini@uh.edu*  
*m.sarraf.j@uh.edu*

June 2, 2018



# Table of Content

1 Governing Equations

2 Solution algorithm

3 Grid

4 Problem Statement

## Governing Equations

Incompressible Navier-Stokes Equations (INSE) in Eulerian form:

$$\partial_t \rho + \rho \nabla \cdot \mathbf{u} = 0 \quad (1)$$

$$\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g} \quad (2)$$

Eqn. 1: Density can be omitted, but let's keep it for now.

Eqn. 2:

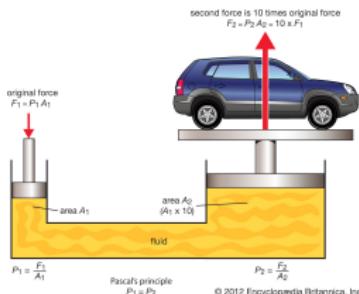
- LHS is acceleration which is intrinsically Lagrangian and when it is translated to Eulerian coordinates it becomes non-linear
  - RHS describes influence of pressure gradient, viscosity and body forces.

# Governing Equations

## Complexities

Difficulties for numerical solution:

- Nonlocality of pressure gradient (Solution: None)
- Nonlinearity of acceleration term (Solution: e.g. Picard's Method)
- Pressure and velocity are coupled and there is no separate equation for pressure (Solution: e.g. SIMPLE, PISO and ACM)



# Governing Equations

## Artificial Compressibility Method

ACM first was developed by A.J. Chorin in 1967 and assumes a small compressibility for the fluid and isothermal condition for the flow:

$$\rho = \rho(p) \therefore \partial_t \rho = \frac{\partial \rho}{\partial p} \frac{\partial p}{\partial t} = \frac{1}{c^2} \frac{\partial p}{\partial t}$$

where  $c$  is artificial sound speed. So INSE becomes:

$$\partial_t P + c^2 \nabla \cdot \mathbf{u} = 0$$

$$\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla P + \nu \nabla^2 \mathbf{u} + \mathbf{g}$$

where  $P = \frac{\rho}{\rho}$ .

# Governing Equations

Remarks on ACM

- Limited scope of applicability: incompressible steady state problems.
- Small time step:  $t$  is pseudo-time step and depends on  $c$ . Therefore, if an explicit discretization method is used time step should have a small value.
- Efficient parallelization: no elliptic PDE

Although applicability of the method is very limited, ACM provides a simple yet informative framework for pedagogical purposes



# Table of Content

1 Governing Equations

2 Solution algorithm

3 Grid

4 Problem Statement

# Solution algorithm

## Pseudo Code

- 1 Initialization of the domain and the variables ( $u, v, p$ )
- 2 While convergence is reached do:
  - 1 Solve momentum equation in x-direction to get  $u$
  - 2 Solve momentum equation in y-direction to get  $v$
  - 3 Update the boundary conditions for  $u$  and  $v$
  - 4 Solve continuity equation to get  $P$
  - 5 Check convergence criteria
- 3 Output the fields data

Note: a flowchart is provided in the repository  
(UHWorkshop/workshop3/flowchart.pdf).

# Solution algorithm

## Boundary Conditions

The two common boundary conditions can be formulated as follows:

- Dirichlet:  $\phi_g = 2\phi_b - \phi_i$
- Neumann:  $\phi_g = \phi_i - \left( \Delta n \frac{\partial \phi}{\partial n} \right)_b$

Indices:  $g$ ,  $b$  and  $i$  denote a ghost, boundary and inner node, respectively

Variables:  $\phi$  is a quantity of interest ( $u$ ,  $v$  or  $P$ ) and  $n$  represents a direction normal to the boundary cell face

# Solution algorithm

## Convergence Criteria

$$1 \quad E_u = \sqrt{(\Delta t \Delta x \Delta y) \sum_{i,j} (u_{i,j}^{n+1} - u_{i,j}^n)^2}$$

$$2 \quad E_v = \sqrt{(\Delta t \Delta x \Delta y) \sum_{i,j} (v_{i,j}^{n+1} - v_{i,j}^n)^2}$$

$$3 \quad E_p = \sqrt{\frac{\Delta t \Delta x \Delta y}{c^2} \sum_{i,j} (P_{i,j}^{n+1} - P_{i,j}^n)^2}$$

$$4 \quad E_\nabla = (\Delta t \Delta x \Delta y) \nabla \cdot \mathbf{u}$$

$$5 \quad E_{total} = \max\{E_u, E_v, E_p, E_\nabla\} < \varepsilon$$

where  $\varepsilon$  is the desired tolerance.

# Table of Content

1 Governing Equations

2 Solution algorithm

3 Grid

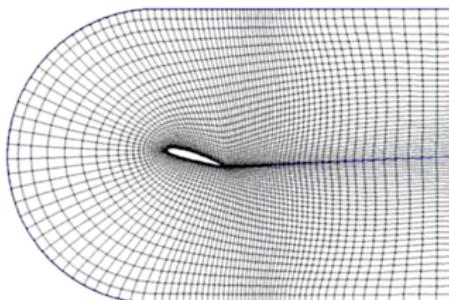
4 Problem Statement

# Grid

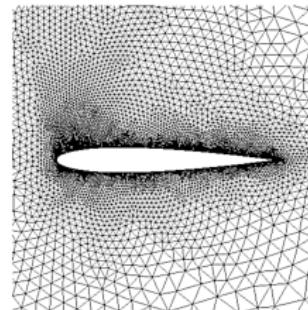
Structured Mesh (SM) vs. Unstructured Mesh (UM)

- Node connection: SM has regular connectivities while UM has irregular ones
- Quality: SM gives better results if applicable
- Memory usage: SM requires less memory and converge faster while UM requires storage of cell-to-cell pointers so take up more memory and is slower

Structured



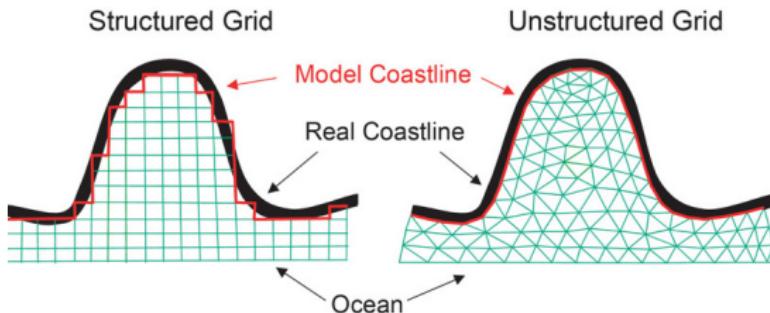
Unstructured



# Grid

## Structured Mesh (SM) vs. Unstructured Mesh (UM)

- Generation: UMs are usually highly automated and take less time to generate while generating SMs could be cumbersome
- Complex geometry: UM can fit complex geometries much better

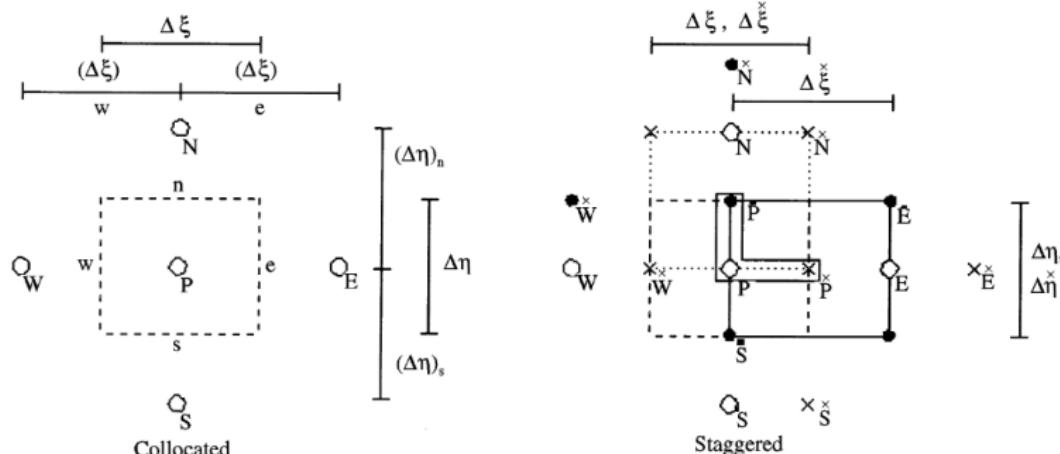


Chen, C., R.C. Beardsley, and G. Cowles. 2006. An unstructured grid, finite-volume coastal ocean model (FVCOM) system. *Oceanography* 19(1):78–89, <https://doi.org/10.5670/oceanog.2006.92>.

# Grid

## Staggered (SG) vs. Collocated (CG)

- Variables: for CG, all at node centers while for SG scalars at node centers and vectors at face centers



Meier, H. F., Alves, J. J. N., & Mori, M. (1999). Comparison between staggered and collocated grids in the finite-volume method performance for single and multi-phase flows. *Computers & Chemical Engineering*

# Grid

Staggered (SG) vs. Collocated (CG)

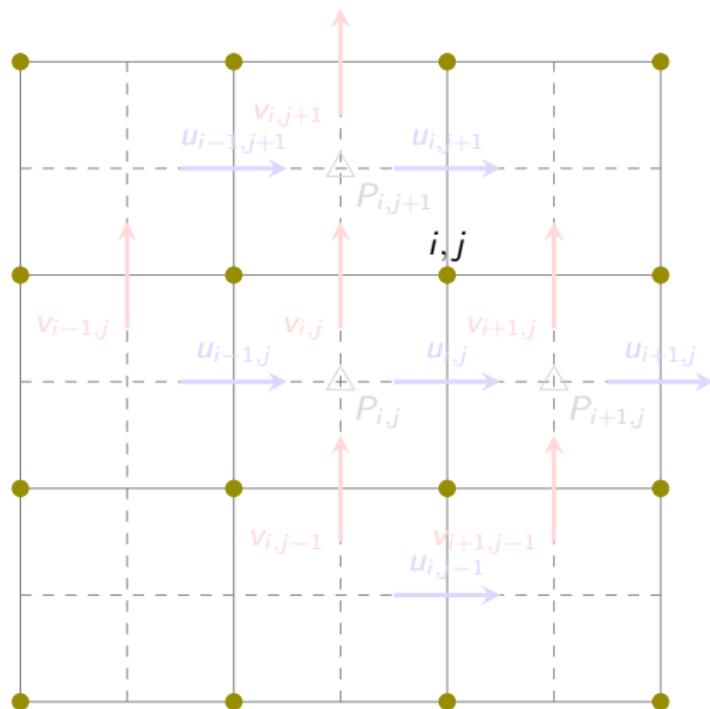
- Accuracy: SG is more accurate while CG has issues such as checkerboard (Rhie–Chow interpolation is one of the remedies)
- Implementation: CG is easier to implement and extend

We use SG for this project:

- Simple geometry
- No additional remedy is required for implementing the numerical schemes
- Get more accurate results

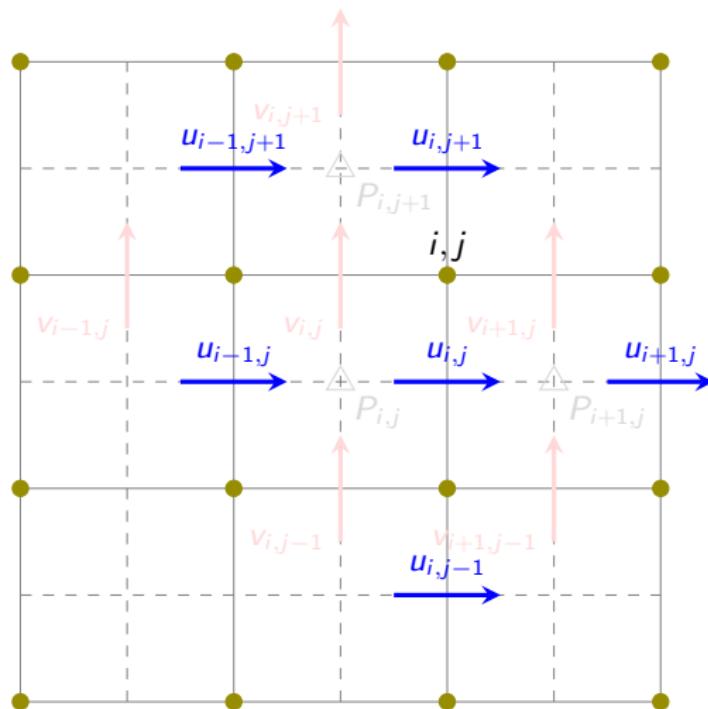
# Grid

## Staggered Grid



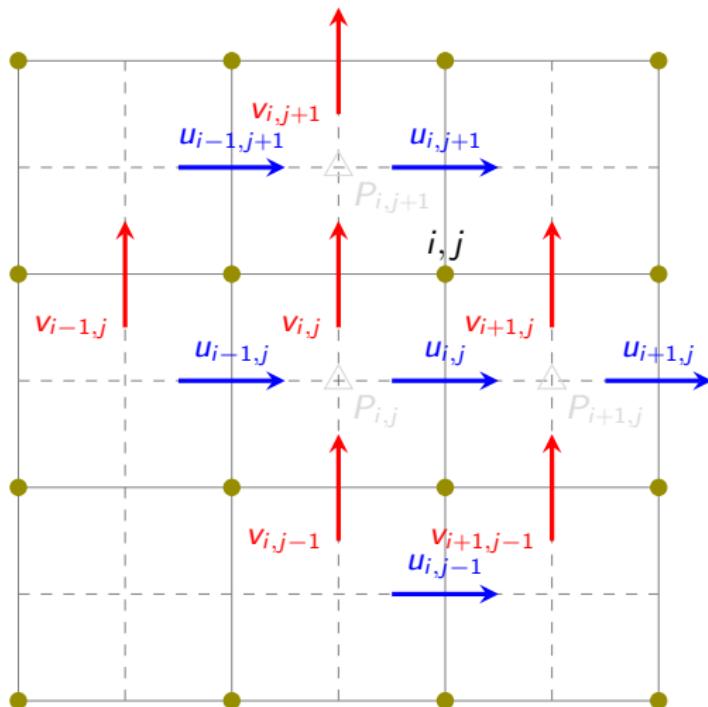
# Grid

## Staggered Grid



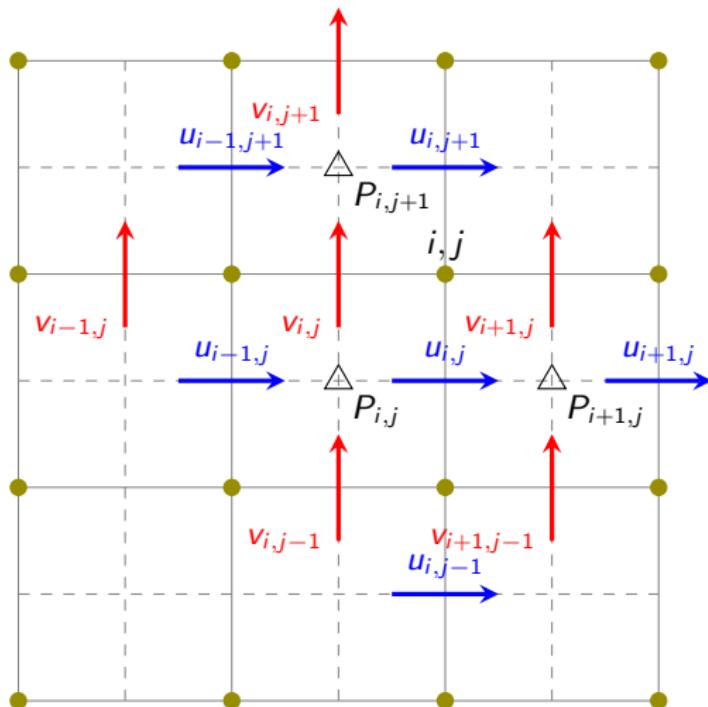
# Grid

## Staggered Grid



# Grid

## Staggered Grid



# Solution algorithm

## Discretization

Spatial: central difference, second order

$$\partial_x \phi = \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x}$$

Temporal: backward difference, first order

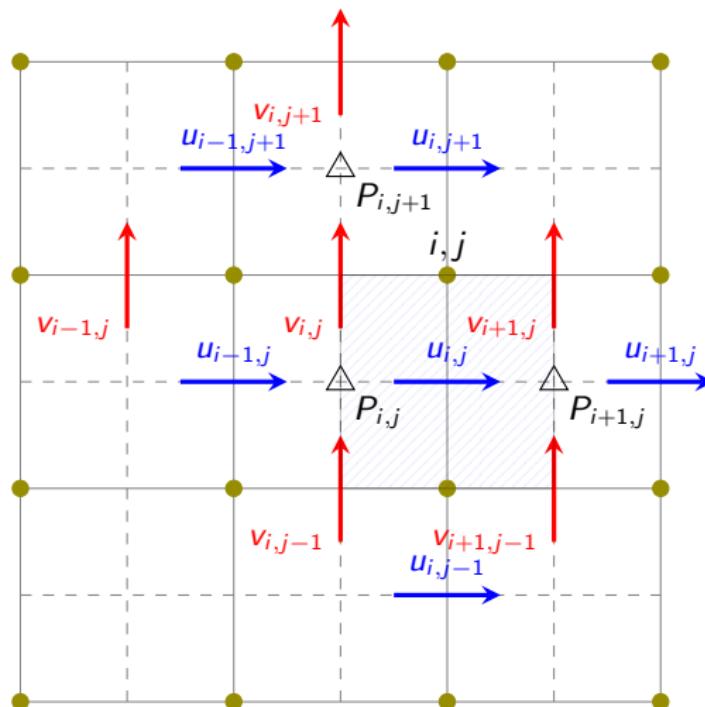
$$\partial_t \phi = \frac{\phi^{n+1} - \phi^n}{\Delta t}$$

Note: for brevity we use the superscript  $n + 1$  to denote the variable in the current time step and no superscript for the previous time step.

Let's discretize the equations based on a staggered grid node numbering.

# Discretization

Control Volume for Momentum Equation in  $x$ -direction



# Discretization

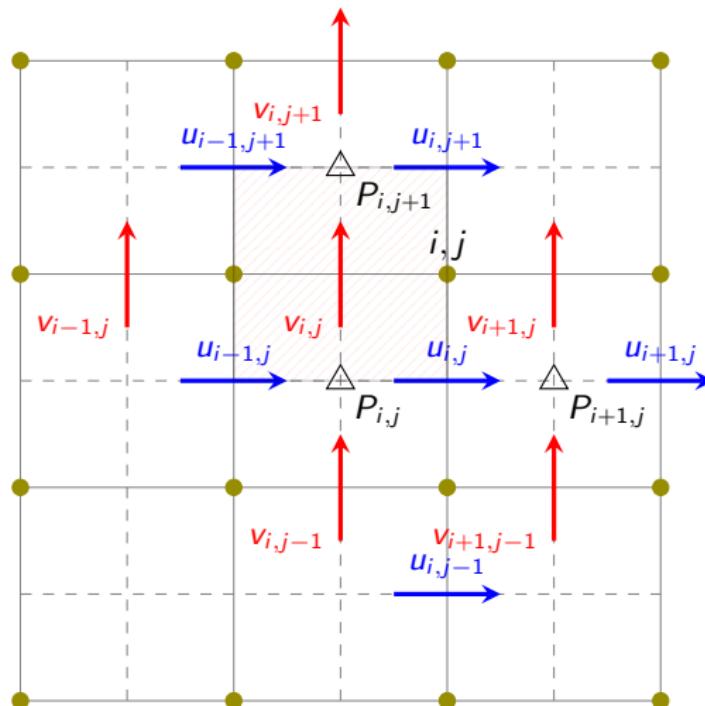
Momentum Equation in  $x$ -direction

$$\partial_t u + \partial_x(uu) + \partial_y(uv) = -\partial_x P + \nu (\partial_{xx} u + \partial_{yy} u)$$

$$\begin{aligned} \frac{u_{i,j}^{n+1} - u_{i,j}}{\Delta t} &+ \frac{\left(\frac{u_{i+1,j} + u_{i,j}}{2}\right)^2 - \left(\frac{u_{i,j} + u_{i-1,j}}{2}\right)^2}{\Delta x} \\ &+ \frac{\left(\frac{u_{i,j+1} + u_{i,j}}{2}\right) \left(\frac{v_{i+1,j} + v_{i,j}}{2}\right) - \left(\frac{u_{i,j} + u_{i,j-1}}{2}\right) \left(\frac{v_{i+1,j-1} + v_{i,j-1}}{2}\right)}{\Delta y} \\ &= \\ &- \frac{P_{i+1,j} - P_{i,j}}{\Delta x} \\ &+ \nu \left( \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} \right) \end{aligned}$$

# Discretization

Control Volume for Momentum Equation in  $y$ -direction



# Discretization

Momentum Equation in  $y$ -direction

$$\partial_t v + \partial_x(uv) + \partial_y(vv) = -\partial_y P + \nu (\partial_{xx} v + \partial_{yy} v)$$

$$\frac{v_{i,j}^{n+1} - v_{i,j}}{\Delta t} + \frac{\left(\frac{u_{i,j+1} + u_{i,j}}{2}\right)\left(\frac{v_{i+1,j} + v_{i,j}}{2}\right) - \left(\frac{u_{i-1,j+1} + u_{i-1,j}}{2}\right)\left(\frac{v_{i,j} + v_{i-1,j}}{2}\right)}{\Delta x}$$

$$+ \frac{\left(\frac{v_{i,j+1} + v_{i,j}}{2}\right)^2 - \left(\frac{v_{i,j} + v_{i,j-1}}{2}\right)^2}{\Delta y}$$

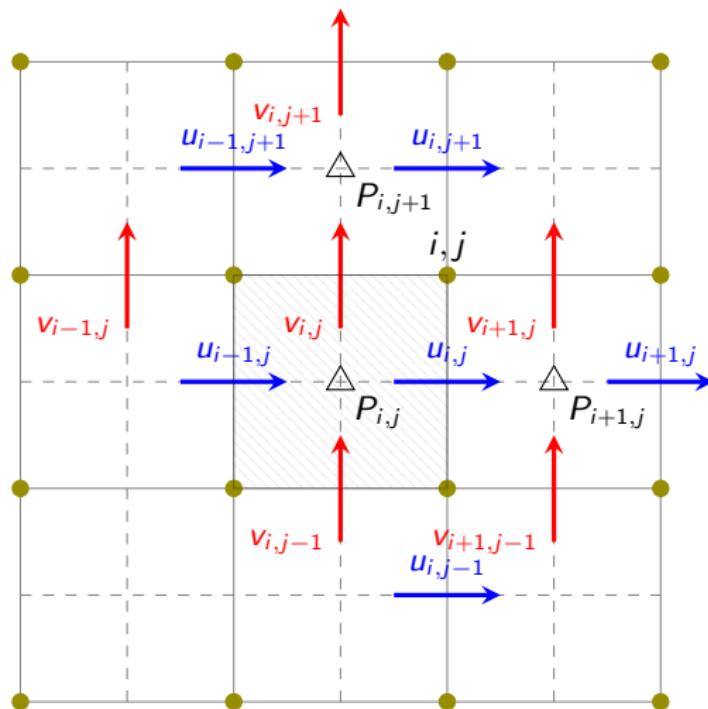
=

$$- \frac{P_{i,j+1} - P_{i,j}}{\Delta y}$$

$$+ \nu \left( \frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{\Delta x^2} + \frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{\Delta y^2} \right)$$

# Discretization

Control Volume for Continuity Equation



# Discretization

Continuity Equation

$$\partial_t P + c^2 (\partial_x u + \partial_y v) = 0$$

$$\frac{P_{i,j}^{n+1} - P_{i,j}^n}{\Delta t} + c^2 \left( \frac{u_{i,j}^n - u_{i-1,j}^n}{\Delta x} + \frac{v_{i,j}^n - v_{i,j-1}^n}{\Delta y} \right) = 0$$

# Table of Content

1 Governing Equations

2 Solution algorithm

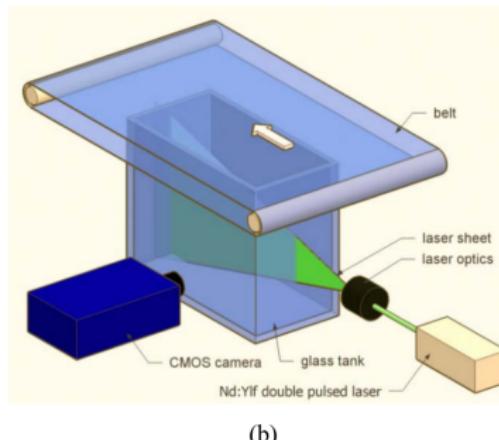
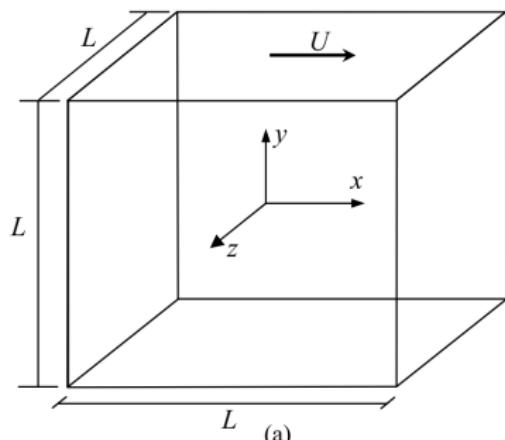
3 Grid

4 Problem Statement

# Problem Statement

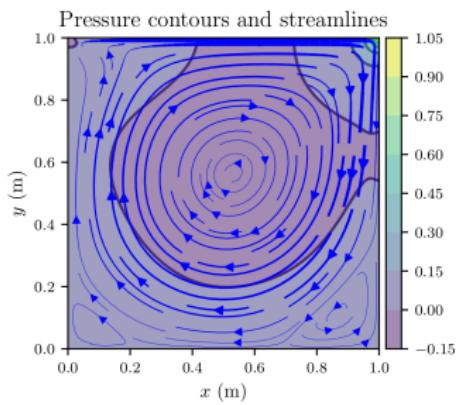
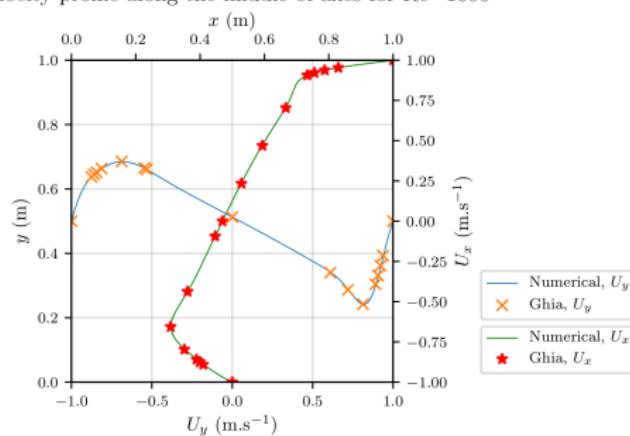
## Lid-Driven Cavity

- Characteristics: steady-state, incompressible
- Boundary Conditions: walls everywhere (top wall is moving)
  - Velocity: Dirichlet ( $\mathbf{U}_x = \mathbf{U}_y = 0$  except for the top wall where  $\mathbf{U}_x = 1$  and  $\mathbf{U}_y = 0$ )
  - Pressure: Neumann ( $\nabla p = 0$ )



# Problem Statement

## Outputs

Velocity profile along the middle of axes for  $Re=1000$ 

# Problem Statement

## Workshop

A numerical solution to this benchmark problem will be presented based on:

- (Finite Difference) Coding from scratch: Python, C
- (Finite Volume) Modifying one of the solvers in OpenFOAM
- (Finite Element) Using libraries provided by FEniCS



# Problem Statement

Exercise: Taylor–Green Vortex

A numerical solution to Taylor–Green Vortex benchmark problem:

- Initial Condition:

$$u(x, y) = \sin(x) \cos(y)$$

$$v(x, y) = -\cos(x) \sin(y)$$

- Boundary condition: walls all around
- Parameters:  $L = 2\pi$ ,  $\nu = 0.2$ ,  $\Delta t = 10^{-4}$  and  $Re = 1000$ .
- Analytical Solution:

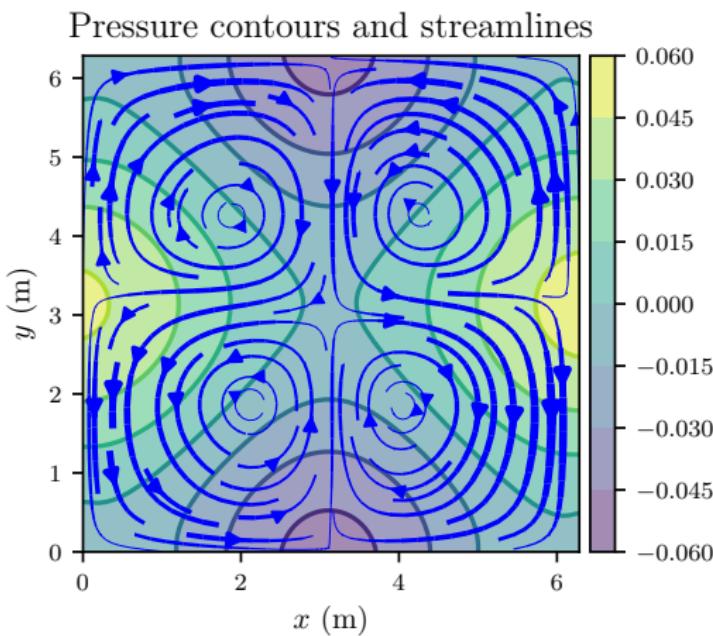
$$u(x, y) = \sin(x) \cos(y) e^{-2\nu t}$$

$$v(x, y) = \cos(x) \sin(y) e^{-2\nu t}$$

$$p(x, y) = \frac{\rho}{4}(\cos(2x) + \cos(2y)) e^{-4\nu t}$$

# Problem Statement

Exercise: Output



# Projects Structure

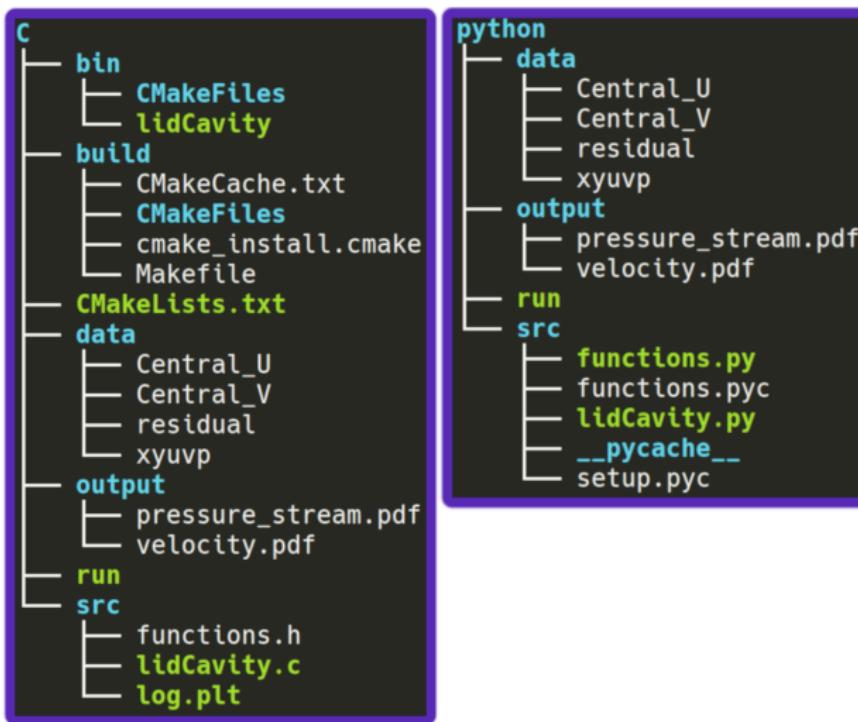
## Tools

The following tools are required:

- Python 3.0+: Python 2 is deprecated
- Numpy: array operation and manipulation
- Numba: code speed-up
- Matplotlib: post-processing
- Flake8: PEP8 style guide enforcement
- CMake: managing the build process

# Projects Structure

## Folders Content



## Final Remarks

Some other interesting open source CFD projects to look into:

- **Basilisk**: solution of partial differential equations on adaptive Cartesian meshes. (C)

<http://basilisk.fr>

- **FluidDyn**: some Python packages specialized for different tasks, in particular for 2D and 3D Fast Fourier Transforms, numerical simulations, laboratory experiments and processing of images of fluid.

<https://fluiddyn.readthedocs.io/en/latest>

- **Firedrake**: an automated system for the solution of partial differential equations using the finite element method. (Python)

<https://firedrakeproject.org>

# Let's Get Started!

```
$ mkdir UHWS  
  
$ cd UHWS  
  
$ git clone https://github.com/taataam/UHWorkshop.git  
  
$ cd UHWorkshop/workshop3
```

