Implementing Immersed Boundary Method for particle representation in OpenFOAM-v2112

Chit Yan Toe

Hydraulic Engineering Department,
Delft University of Technology,
Delft, The Netherlands

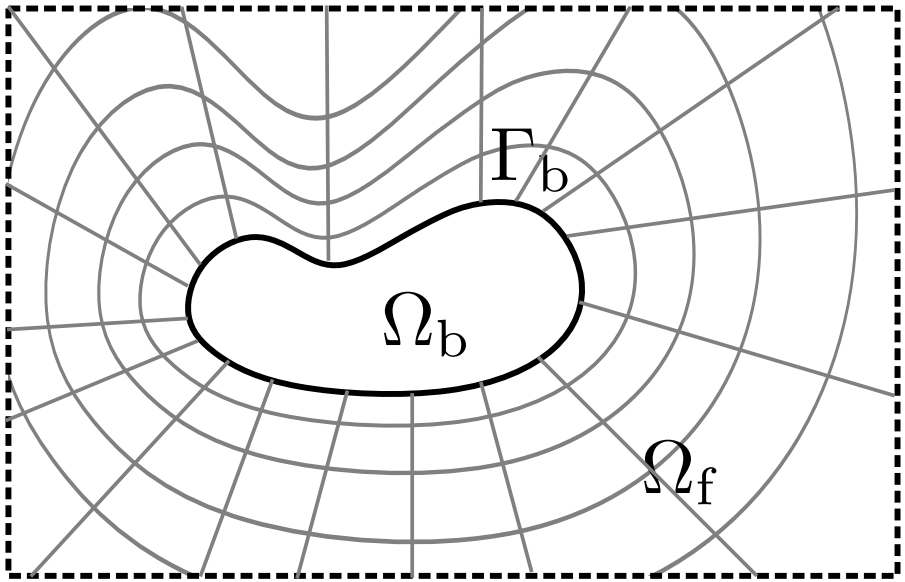
January 16, 2024

Simulations of particle motion

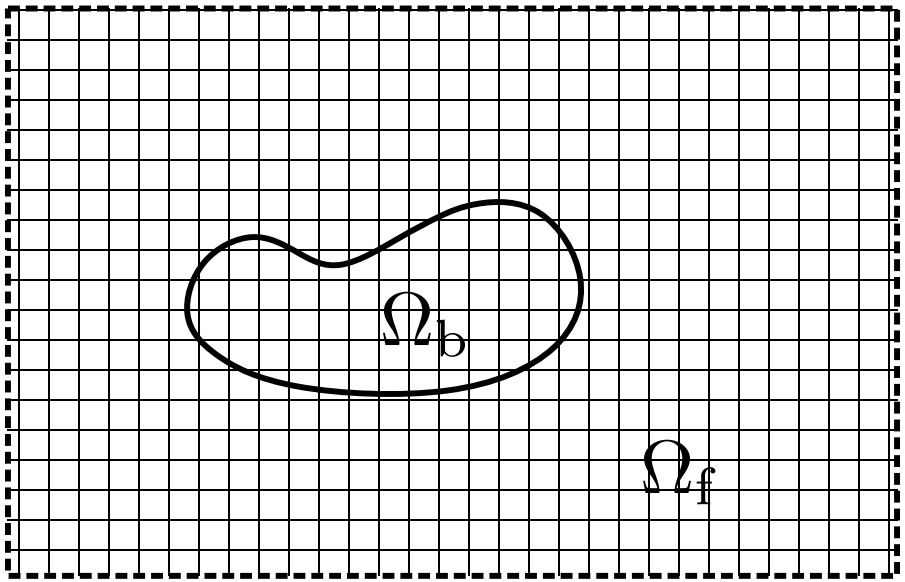
Fluid-particle interaction problems —

- powder technology
- sediment transport
- granular mechanics
- plastic waste transport

Different types of Computational mesh



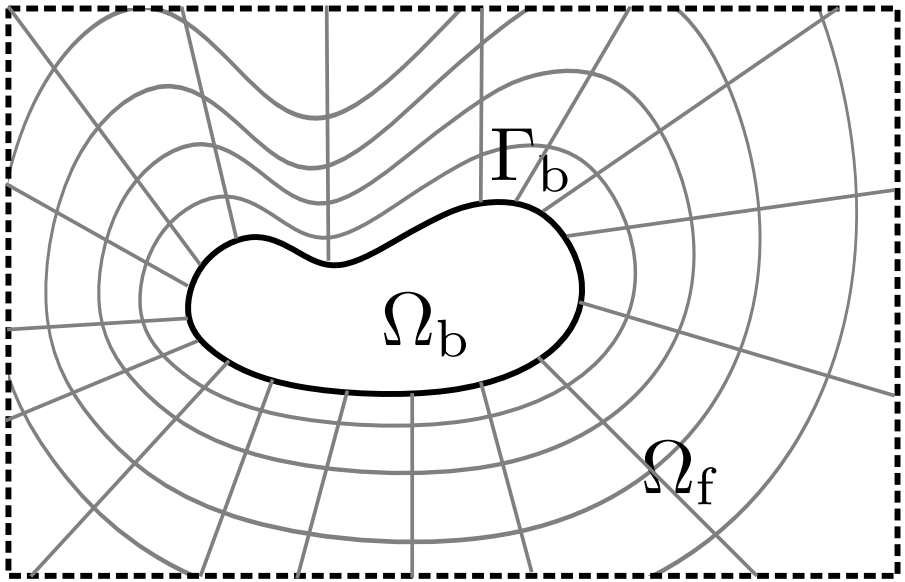
(a) Body-conformal mesh



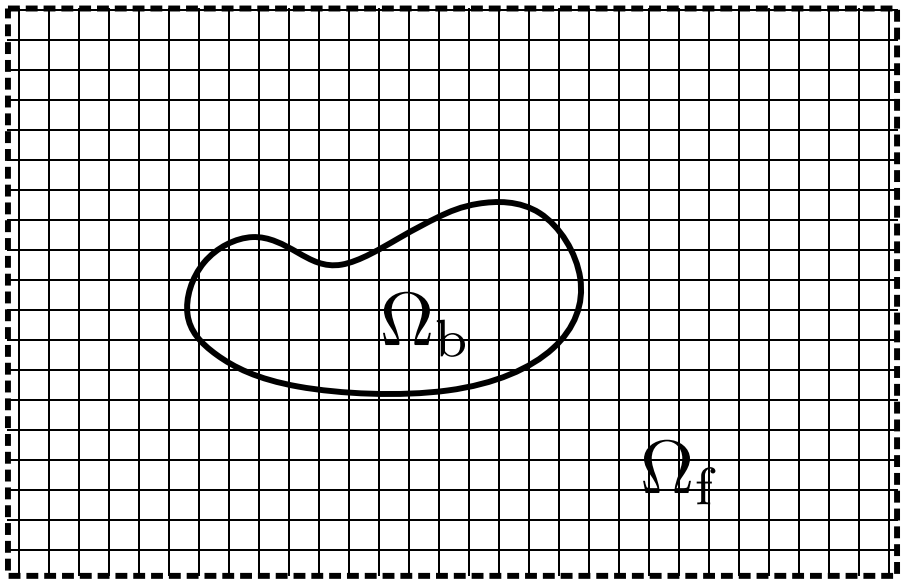
(b) Immersed boundary mesh

- Time-dependent mesh
- Imposes no-slip BC on the particle surface
- Requires re-meshing for dynamic meshes

Different types of Computational mesh



(a) Body-conformal mesh



(b) Immersed boundary mesh

- Time-dependent mesh
- Imposes no-slip BC on the particle surface
- Requires re-meshing for dynamic meshes

- Time-independent mesh
- A single mesh during the simulation
- No mesh for particles
- So, how can particles be recognized by the flow?

How to recognize the particles?

Navier-Stokes Equations for incompressible flow

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u}$$

How to recognize the particles?

Navier-Stokes Equations for incompressible flow

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u}$$

Added a forcing term

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u} + \boldsymbol{f}$$

where $\boldsymbol{f}$ is the interaction force between the particle and the flow.

How to recognize the particles?

Navier-Stokes Equations for incompressible flow

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u}$$

Added a forcing term

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u} + \boldsymbol{f}$$

where $\boldsymbol{f}$ is the interaction force between the particle and the flow.

This is the so-called Immersed Boundary Method (IBM).

How to recognize the particles?

Navier-Stokes Equations for incompressible flow

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u}$$

Added a forcing term

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u} + \boldsymbol{f}$$

where $\boldsymbol{f}$ is the interaction force between the particle and the flow.

This is the so-called Immersed Boundary Method (IBM).
How to impose $\boldsymbol{f}$?

Two groups of IBM

- Continuous Forcing Approach — $\boldsymbol{f}$ is added *before* numerical discretization

Two groups of IBM

- Continuous Forcing Approach — $\boldsymbol{f}$ is added *before* numerical discretization
- Discrete Forcing Approach — $\boldsymbol{f}$ is added *after* numerical discretization

Two groups of IBM

- Continuous Forcing Approach — $\boldsymbol{f}$ is added *before* numerical discretization
- Discrete Forcing Approach — $\boldsymbol{f}$ is added *after* numerical discretization
  - Indirect imposition technique — Use of distribution function
  - Direct imposition technique — directly impose boundary condition on the IB

IBM in OpenFOAM

- OpenFOAM-v2006 → porousPimpleIbFoam solver developed by Vergassola [1] — Continuous forcing

IBM in OpenFOAM

- OpenFOAM-v2006 → porousPimpleIbFoam solver developed by Vergassola [1] — Continuous forcing
- foam-extend-5.0 → pimpleDyMIbFoam solver by Jasak [2] — Direct forcing

IBM in OpenFOAM

- OpenFOAM-v2006 → porousPimpleIbFoam solver developed by Vergassola [1] — Continuous forcing
- foam-extend-5.0 → pimpleDyMIbFoam solver by Jasak [2] — Direct forcing
- OpenFOAM v9 → sdfibm solver by Zhang [3] — Direct forcing

IBM in OpenFOAM

- OpenFOAM-v2006 → porousPimpleIbFoam solver developed by Vergassola [1] — Continuous forcing
- foam-extend-5.0 → pimpleDyMIbFoam solver by Jasak [2] — Direct forcing
- OpenFOAM v9 → sdfibm solver by Zhang [3] — Direct forcing
- This project → sdfIbmESI solver — Direct forcing + OpenFOAM-v2112

Direct forcing technique

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u} + \boldsymbol{f}$$

Direct forcing technique

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u} + \boldsymbol{f}$$

Using general time discretization

$$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^{n}}{\Delta t} = \mathrm{RHS}^{n+1/2} + \boldsymbol{f}^{n+1/2}$$

Direct forcing technique

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\frac{\nabla p}{\rho} + \nu \Delta \boldsymbol{u} + \boldsymbol{f}$$

Using general time discretization

$$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^{n}}{\Delta t} = \mathrm{RHS}^{n+1/2} + \boldsymbol{f}^{n+1/2}$$

To obtain $\boldsymbol{u}^{n+1} = \boldsymbol{V}^{n+1}$,

$$\boldsymbol{f}^{n+1/2} = -\mathrm{RHS}^{n+1/2} + \frac{\boldsymbol{V}^{n+1} - \boldsymbol{u}^{n}}{\Delta t}$$

for some grid nodes.

Volume-average discrete forcing method — sdfibm solver

Reconsider

$$\nabla \cdot \boldsymbol{u}_{\mathrm{f}} = 0 \tag{1}$$

and

$$\frac{\partial \boldsymbol{u}_{\mathrm{f}}}{\partial t} + \boldsymbol{u}_{\mathrm{f}} \cdot \nabla \boldsymbol{u}_{\mathrm{f}} = -\frac{1}{\rho_{\mathrm{f}}} \nabla p + \nu_{\mathrm{f}} \nabla^2 \boldsymbol{u}_{\mathrm{f}} \tag{2}$$

Volume-average discrete forcing method — sdfibm solver

Reconsider

$$\nabla \cdot \boldsymbol{u}_{\mathrm{f}} = 0 \tag{1}$$

and

$$\frac{\partial \boldsymbol{u}_{\mathrm{f}}}{\partial t} + \boldsymbol{u}_{\mathrm{f}} \cdot \nabla \boldsymbol{u}_{\mathrm{f}} = -\frac{1}{\rho_{\mathrm{f}}} \nabla p + \nu_{\mathrm{f}} \nabla^2 \boldsymbol{u}_{\mathrm{f}} \tag{2}$$

Define the volume-weighted average of velocity as

$$\boldsymbol{u} = (1 - \alpha)\boldsymbol{u}_{\mathrm{f}} + \alpha \boldsymbol{u}_{\mathrm{p}}$$

where

$$\boldsymbol{u}_{\mathrm{p}} = \boldsymbol{v}_{\mathrm{p}} + \boldsymbol{\omega}_{\mathrm{p}} \times \boldsymbol{r}.$$

Volume-average discrete forcing method — sdfibm solver

Reconsider

$$\nabla \cdot \boldsymbol{u}_{\mathrm{f}} = 0 \tag{1}$$

and

$$\frac{\partial \boldsymbol{u}_{\mathrm{f}}}{\partial t} + \boldsymbol{u}_{\mathrm{f}} \cdot \nabla \boldsymbol{u}_{\mathrm{f}} = -\frac{1}{\rho_{\mathrm{f}}} \nabla p + \nu_{\mathrm{f}} \nabla^2 \boldsymbol{u}_{\mathrm{f}} \tag{2}$$

Define the volume-weighted average of velocity as

$$\boldsymbol{u} = (1 - \alpha)\boldsymbol{u}_{\mathrm{f}} + \alpha \boldsymbol{u}_{\mathrm{p}}$$

where

$$\boldsymbol{u}_{\mathrm{p}} = \boldsymbol{v}_{\mathrm{p}} + \boldsymbol{\omega}_{\mathrm{p}} \times \boldsymbol{r}.$$

We also have

$$\nabla \cdot \boldsymbol{u} = 0 \tag{3}$$

and

$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{H} - \nabla P + \boldsymbol{f}_{\mathrm{p}} \tag{4}$$

where

$$\boldsymbol{H} \equiv -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nu_{\mathrm{f}} \nabla^2 \boldsymbol{u}$$

and $P \equiv p/\rho_{\mathrm{f}}$ and $\boldsymbol{f}_{\mathrm{p}}$ is the interaction force.

Volume-average discrete forcing method — sdfibm solver

Contd.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{H} - \nabla P + \boldsymbol{f}_{\mathrm{p}}$$

where

$$\boldsymbol{H} \equiv -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nu_{\mathrm{f}} \nabla^2 \boldsymbol{u}$$

In other words, $\boldsymbol{f}_{\mathrm{p}}$ is the force required to adjust the single-phase fluid velocity $\boldsymbol{u}_{\mathrm{f}}$ to the averaged velocity $\boldsymbol{u}$.

Volume-average discrete forcing method — sdfibm solver

Contd.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{H} - \nabla P + \boldsymbol{f}_{\mathrm{p}}$$

where

$$\boldsymbol{H} \equiv -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nu_{\mathrm{f}} \nabla^2 \boldsymbol{u}$$

In other words, $\boldsymbol{f}_{\mathrm{p}}$ is the force required to adjust the single-phase fluid velocity $\boldsymbol{u}_{\mathrm{f}}$ to the averaged velocity $\boldsymbol{u}$.
Using general time discretization

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^{n} + \Delta t \left(\boldsymbol{H} - \nabla P + \boldsymbol{f}_{\mathrm{p}} \right)$$

Volume-average discrete forcing method — sdfibm solver

Contd.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{H} - \nabla P + \boldsymbol{f}_{\mathrm{p}}$$

where

$$\boldsymbol{H} \equiv -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nu_{\mathrm{f}} \nabla^2 \boldsymbol{u}$$

In other words, $\boldsymbol{f}_{\mathrm{p}}$ is the force required to adjust the single-phase fluid velocity $\boldsymbol{u}_{\mathrm{f}}$ to the averaged velocity $\boldsymbol{u}$.
Using general time discretization

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^{n} + \Delta t \left(\boldsymbol{H} - \nabla P + \boldsymbol{f}_{\mathrm{p}} \right)$$

Without interaction force,

$$\hat{\boldsymbol{u}} = \boldsymbol{u}^{n} + \Delta t \left(\boldsymbol{H} - \nabla P \right)$$

Volume-average discrete forcing method — sdfibm solver

Contd.

$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{H} - \nabla P + \boldsymbol{f}_{\mathrm{p}}$$

where

$$\boldsymbol{H} \equiv -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nu_{\mathrm{f}} \nabla^2 \boldsymbol{u}$$

In other words, $\boldsymbol{f}_{\mathrm{p}}$ is the force required to adjust the single-phase fluid velocity $\boldsymbol{u}_{\mathrm{f}}$ to the averaged velocity $\boldsymbol{u}$.
Using general time discretization

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^{n} + \Delta t \left(\boldsymbol{H} - \nabla P + \boldsymbol{f}_{\mathrm{p}} \right)$$

Without interaction force,

$$\hat{\boldsymbol{u}} = \boldsymbol{u}^{n} + \Delta t \left(\boldsymbol{H} - \nabla P \right)$$

Therefore, $\boldsymbol{u}^{n+1} - \hat{\boldsymbol{u}} = \Delta t \boldsymbol{f}_{\mathrm{p}}$

Volume-average discrete forcing method — sdfibm solver

Contd. We know that $\boldsymbol{u}^{n+1} - \hat{\boldsymbol{u}} = \Delta t \boldsymbol{f}_{\mathrm{p}}$.

Volume-average discrete forcing method — sdfibm solver

Contd. We know that $\boldsymbol{u}^{n+1} - \hat{\boldsymbol{u}} = \Delta t \boldsymbol{f}_{\mathrm{p}}$.

- For the solid cell, we require $\boldsymbol{u}^{n+1} = \boldsymbol{u}_p$, therefore we get

$$\boldsymbol{f}_{\mathrm{p}} = \frac{\boldsymbol{u}_{\mathrm{p}} - \hat{\boldsymbol{u}}}{\Delta t}$$

Volume-average discrete forcing method — sdfibm solver

Contd. We know that $\boldsymbol{u}^{n+1} - \hat{\boldsymbol{u}} = \Delta t \boldsymbol{f}_{\mathrm{p}}$.

- For the solid cell, we require $\boldsymbol{u}^{n+1} = \boldsymbol{u}_p$, therefore we get

$$\boldsymbol{f}_{\mathrm{p}} = \frac{\boldsymbol{u}_{\mathrm{p}} - \hat{\boldsymbol{u}}}{\Delta t}$$

- For the fluid cell, we require no interaction force, therefore $\boldsymbol{f}_{\mathrm{p}} = \boldsymbol{0}$

Volume-average discrete forcing method — sdfibm solver

Contd. We know that $\boldsymbol{u}^{n+1} - \hat{\boldsymbol{u}} = \Delta t \boldsymbol{f}_{\mathrm{p}}$.

- For the solid cell, we require $\boldsymbol{u}^{n+1} = \boldsymbol{u}_p$, therefore we get

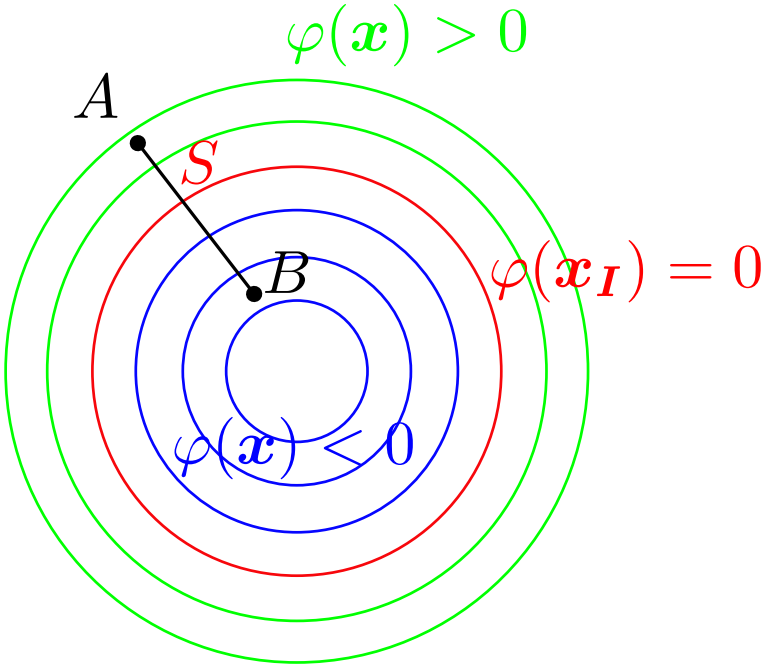
$$\boldsymbol{f}_{\mathrm{p}} = \frac{\boldsymbol{u}_{\mathrm{p}} - \hat{\boldsymbol{u}}}{\Delta t}$$

- For the fluid cell, we require no interaction force, therefore $\boldsymbol{f}_{\mathrm{p}} = \boldsymbol{0}$
- For the interface cell, we get

$$\boldsymbol{f}_{\mathrm{p}} = \alpha \frac{\boldsymbol{u}_{\mathrm{p}} - \hat{\boldsymbol{u}}}{\Delta t}$$

where α needs to determined by geometrical functions.

How to calculate α?



Fraction of AB lying within the surface S is simply the ratio: $-\varphi_A/(\varphi_B - \varphi_A)$

Implementation of sdfibm solver

Directory of sdfibm solver

```
|-- CMakeLists.txt
|-- LICENSE.txt
|-- README.md
|-- examples
|-- figs
|-- src
`-- tool_vof
```

Implementation of sdfibm solver

Directory of sdfibm solver

```
|-- CMakeLists.txt
|-- LICENSE.txt
|-- README.md
|-- examples
|-- figs
|-- src
`-- tool_vof
```

src directory of sdfibm solver

```
src/
|-- CMakeLists.txt
|-- libcollision
|-- libmaterial
|-- libmotion
|-- libshape
|-- main.cpp
|-- solid.cpp
|-- solid.h
|-- solidcloud.cpp
|-- solidcloud.h
`-- utils.h
```

Implementation of sdfibm solver

Listing 1: Creation of solidcloud object in main.cpp of sdfibm solver

```
#include "solidcloud.h"

int main(int argc, char *argv[])
{
    // #include "--.H"
    std::string dictfile;
    if(runTime.time().value() > 0)
    {
        if(!Foam::Pstream::parRun())
            dictfile = mesh.time().timeName() + "/solidDict";
        else
            dictfile = "processor0/" + mesh.time().timeName() + "/solidDict";
    }
    else
    {
        dictfile = "solidDict";
    }
    sdfibm::SolidCloud solidcloud(args.path() + "/" + dictfile, U, runTime.value
     ());
    solidcloud.saveState();  // write the initial condition
```

Implementation of sdfibm solver

Listing 2: Solving momentum equations in main.cpp of sdfibm solver

```
20      while (runTime.loop())
21      {
22          Foam::Info << "Time = " << runTime.timeName() << Foam::endl;
23          #include "CourantNo.H"
24          Foam::dimensionedScalar dt = runTime.deltaT();
25          if(solidcloud.isOnFluid())
26          {
27              Foam::fvVectorMatrix UEqn(
28                  fvm::ddt(U)
29                + 1.5*fvc::div(phi, U) - 0.5*fvc::div(phi.oldTime(), U.oldTime())
30                ==0.5*fvm::laplacian(nu, U) + 0.5*fvc::laplacian(nu, U));
31              UEqn.solve();
32
33              phi = linearInterpolate(U) & mesh.Sf();
34              Foam::fvScalarMatrix pEqn(fvm::laplacian(p) == fvc::div(phi)/dt -
     fvc::div(Fs));
35              pEqn.solve();
```

Implementation of sdfibm solver

Listing 3: Solving momentum equations in main.cpp of sdfibm solver

```
37          U   = U   - dt*fvc::grad(p);
38          phi = phi - dt*fvc::snGrad(p)*mesh.magSf();
39
40          Foam::fvScalarMatrix TEqn(
41              fvm::ddt(T)
42            + fvm::div(phi, T)
43            ==fvm::laplacian(alpha, T));
44          TEqn.solve();
45      }
```

Implementation of sdfibm solver

Listing 4: Fluid-particle interaction main.cpp of sdfibm solver

```
46      solidcloud.interact(runTime.value(), dt.value());
47      if(solidcloud.isOnFluid())
48      {
49          U = U - Fs*dt;
50          phi = phi - dt*(linearInterpolate(Fs) & mesh.Sf());
51          U.correctBoundaryConditions();
52          adjustPhi(phi, U, p);
53          T = (1.0 - As)*T + Ts;
54          T.correctBoundaryConditions();
55          #include "continuityErrs.H"
56      }
57      solidcloud.evolve(runTime.value(), dt.value());
58      solidcloud.saveState();
59      if(solidcloud.isOnFluid())
60      {
61          solidcloud.fixInternal(dt.value());
62      }
```

Implementation of sdfibm solver

Listing 5: Saving output file main.cpp of sdfibm solver

```
63          if(runTime.outputTime())
64          {
65              runTime.write();
66
67              if(Foam::Pstream::master())
68              {
69                  std::string file_name;
70                  if(Foam::Pstream::parRun())
71                      file_name = "./processor0/" + runTime.timeName() + "/
    solidDict";
72                  else
73                      file_name = "./" + runTime.timeName() + "/solidDict";
74                  solidcloud.saveRestart(file_name);
75              }
76          }
77      }
78      Foam::Info << "DONE\n" << endl;
79      return 0;
80  }
```

Simulation of laminar flow around a sphere

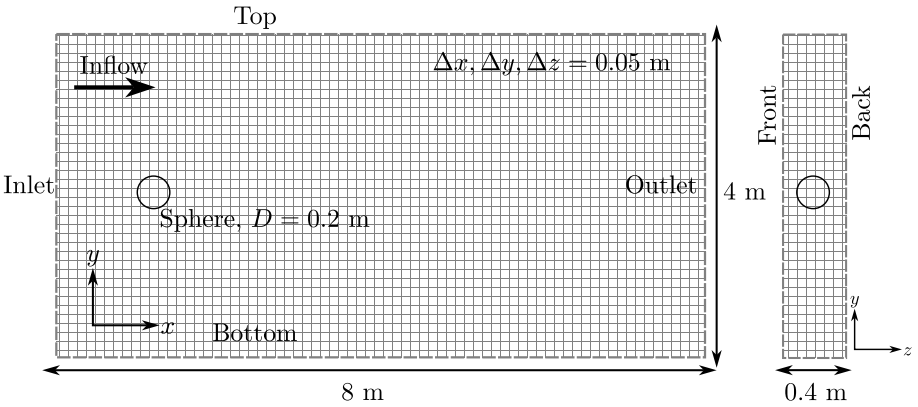


Figure: Computational domain.

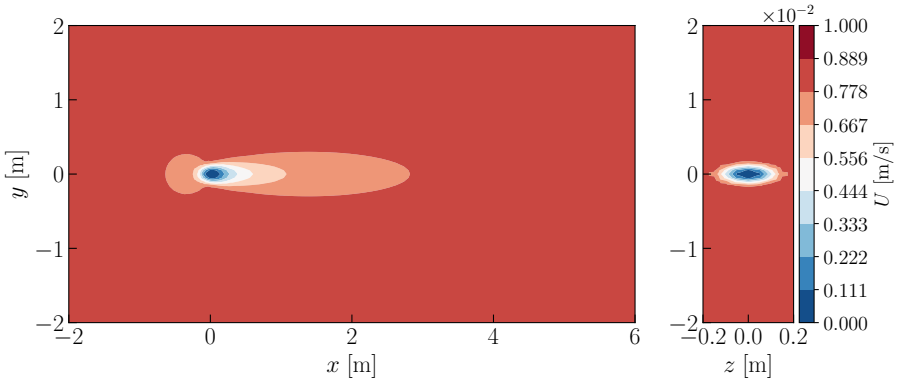


Figure: Result of `pimpleDyMIbFoam` solver.

Simulation of laminar flow around a sphere

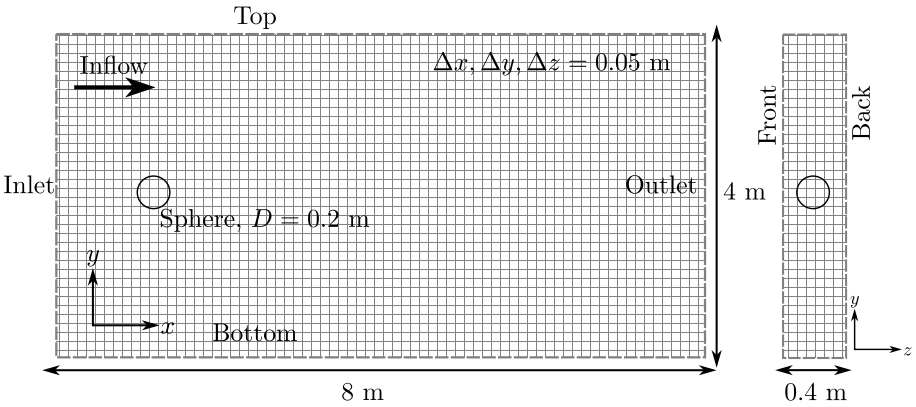


Figure: Computational domain.

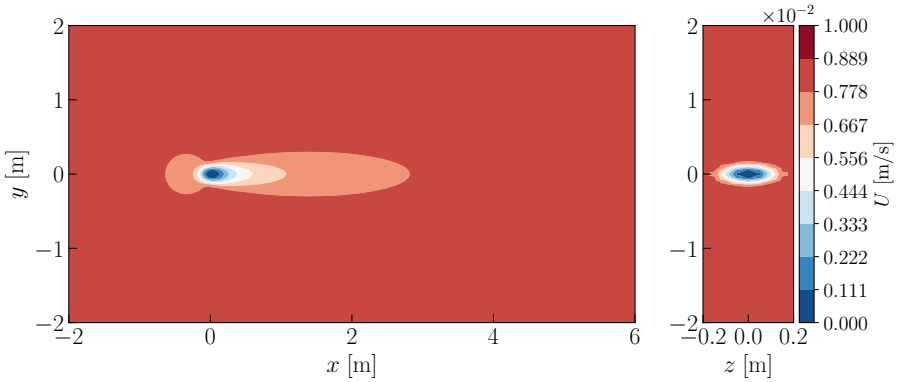
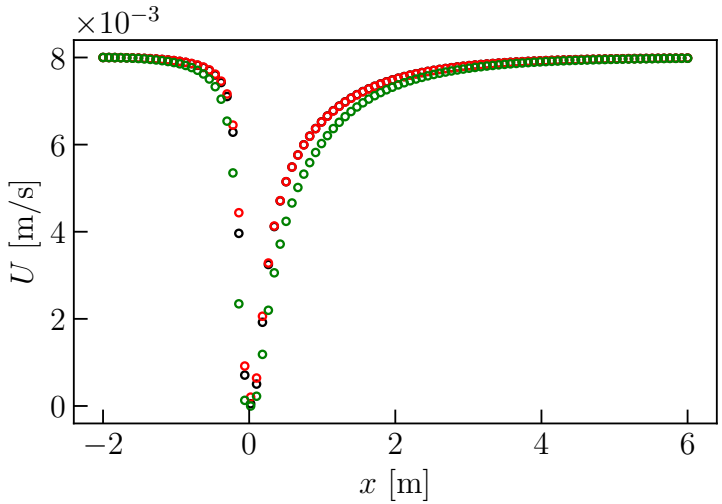
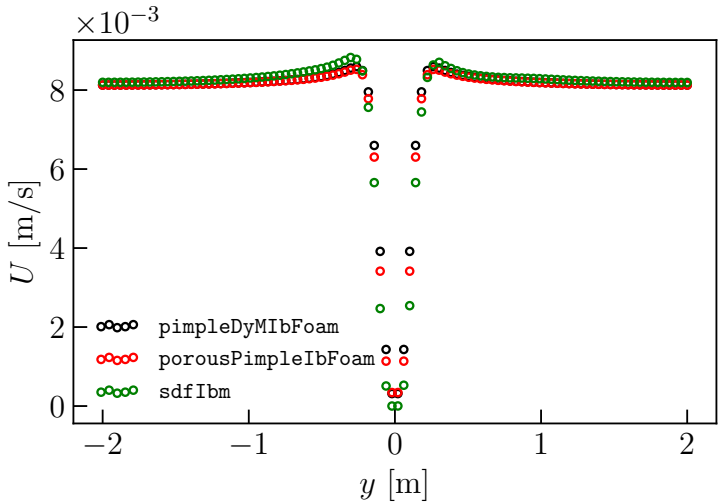


Figure: Result of `pimpleDyMIbFoam` solver.





Re-implementation — sdfIbmESI solver

- Main difference is in compilation procedure
  - CMake in OpenFOAM v9
  - wmake in OpenFOAM-v2112 → Make folder is required.

Re-implementation — sdfIbmESI solver

- Main difference is in compilation procedure
  - CMake in OpenFOAM v9
  - wmake in OpenFOAM-v2112 → Make folder is required.
- Some minor differences e.g.
  - type conversion — vector() function, readScalar function
  - #include "IFstream.H" in solidcloud.C file

Re-implementation — sdfIbmESI solver

- Main difference is in compilation procedure
  - CMake in OpenFOAM v9
  - wmake in OpenFOAM-v2112 → Make folder is required.
- Some minor differences e.g.
  - type conversion — vector() function, readScalar function
  - #include "IFstream.H" in solidcloud.C file
- no main.cpp file in OpenFOAM-v2112, but sdfIbmESI.C file

Directory of sdfibmESI solver

```
sdfIbmESI/
|-- Make
|-- UEqn.H
|-- correctPhi.H
|-- createFields.H
|-- pEqn.H
|-- sdfIbmESI.C
`-- setRDeltaT.H
```

Re-implementation — sdfIbmESI solver

Start with library files — libcollision, libshape, libmaterial, libmotion

collision.h of sdfibm solver

```
1 #ifndef COLLISION_HPP
2 #define COLLISION_HPP
3
4 #include "../types.h"
5 #include "../utils.h"
6 #include "../solid.h"
```

collision.H of sdfIbmESI solver

```
1 #ifndef COLLISION_HPP
2 #define COLLISION_HPP
3
4 #include "types.H"
5 #include "utils.H"
6 #include "solid.H"
```

Re-implementation — sdfIbmESI solver

Start with library files — libcollision, libshape, libmaterial, libmotion

collision.h of sdfibm solver

```
#ifndef COLLISION_HPP
#define COLLISION_HPP

#include "../types.h"
#include "../utils.h"
#include "../solid.h"
```

collision.H of sdfIbmESI solver

```
#ifndef COLLISION_HPP
#define COLLISION_HPP

#include "types.H"
#include "utils.H"
#include "solid.H"
```

options file in Make folder of libcollision library

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/OpenFOAM/lnInclude \
    -I../libshape/lnInclude \
    -I../libmotion/lnInclude \
    -I../libmaterial \
    -I..
```

Re-implementation — sdfIbmESI solver

files file in Make folder

```
solidcloud.C
solid.C
meshinfo.C
logger.C
geometrictools.C
cellenumerator.C
sdfIbmESI.C

EXE = $(FOAM_USER_APPBIN)/sdfIbmESI
```

options file in Make folder

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/
     lnInclude \
    # skipped some lines
    -Ilibshape/lnInclude \
    -Ilibmotion/lnInclude \
    -Ilibcollision/lnInclude \
    -Ilibmaterial

EXE_LIBS = \
    -lfiniteVolume \
    # skipped some lines
    -Llibshape \
    -lshape \
    -Llibmotion \
    -lmotion \
    -Llibcollision \
    -lcollision
```

Practical part

- Download the files from GitHub repository — `sdfIbmESI` solver

  ```
  $ git clone  https://github.com/ChitYanToe/sdfIbmESI.git
  ```

- Run the script for compiling and running the simulation.

  ```
  $ ./Allrun
  ```

- Or, for manual compilation instead of using `Allrun` file, one needs to run

  ```
  $ source exportFile.sh
  ```

Thank you for your time.

Q & A.

Bibliography I

[1] Marco Vergassola. "A continuous forcing immersed boundary approach to solve the VARANS equations in a volumetric porous region". In: *In Proceedings of CFD with OpenSource Software, 2021*. Edited by Nilsson. H, 2021.

[2] Hrvoje Jasak. "Immersed boundary surface method in foam-extend". In: *Workshop OpenFOAM in Hydraulic Engineering*. Vol. 21. 2018, p. 22.

[3] Chenguang Zhang. "sdfibm: A signed distance field based discrete forcing immersed boundary method in OpenFOAM". In: *Computer Physics Communications* 255 (2020), p. 107370.