Robust EcoHydraulic 3D Modeling Tools for Rivers with Complex Instream Structures

Project report to U.S. Department of Interior, Bureau of Reclamation

Prepared by

Xiaofeng Liu, Ph.D., P.E., Associate Professor

Yuncheng Xu, Ph.D.

Yalan Song, Graduate Research Assistant

Department of Civil and Environmental Engineering Pennsylvania State University

Contact Email: [xliu@engr.psu.edu](mailto:xliu@engr.psu.edu) Web: <http://water.engr.psu.edu/liu>

May, 2021

Table of Contents

[Chapter 1 Introduction 3](#_Toc69392070)

[Chapter 2 Instruction on compilation 4](#_Toc69392071)

[2.1 Windows 4](#_Toc69392072)

[2.2 Linux 4](#_Toc69392073)

[2.3 List of changes in original files 5](#_Toc69392074)

[Chapter 3 IB method implementation in U2RANS 6](#_Toc69392075)

[3.1 Basic algorithm for IB method in U2RANS 6](#_Toc69392076)

[3.2 The processing of IB in u2rans pre 9](#_Toc69392077)

[3.2.1 Find cut (intercepted) cells 9](#_Toc69392078)

[**3.2.2** **Find IB, live, and dead cells** 11](#_Toc69392079)

[**3.2.3** **Octree algorithm** 12](#_Toc69392080)

[**3.2.4** **Finding hit points** 13](#_Toc69392081)

[**3.2.5** **Mapping between IB cells and STL** 14](#_Toc69392082)

[3.3 Data file generated by u2rans pre for use by u2rans 14](#_Toc69392083)

[3.4 IB cell information 17](#_Toc69392084)

[3.5 List of added modules or subroutines 17](#_Toc69392085)

[**3.5.1** **In the pre-processing** 17](#_Toc69392086)

[**3.5.2** **In the solver** 20](#_Toc69392087)

[3.6 Summary of code modification 21](#_Toc69392088)

[**3.6.1** **u2rans pre** 21](#_Toc69392089)

[**3.6.2** **u2rans** 22](#_Toc69392090)

[**3.7 Code structure of IB implementation in U2RANS** 23](#_Toc69392091)

[Chapter 4 Applications 26](#_Toc69392092)

[4.1 Turbulent flow over flat plate 26](#_Toc69392093)

[4.2 Turbulent flow around a cylinder over scoured beds 29](#_Toc69392094)

[4.3 Turbulent flow over 3D dunes 31](#_Toc69392095)

[4.4 Backward facing step flows 34](#_Toc69392096)

[4.5 Flows around the Nittany lion body 36](#_Toc69392097)

[4.6 Flows around complex trees 37](#_Toc69392098)

[4.7 Cases availability 40](#_Toc69392099)

[Chapter 5 Best practice guidance 41](#_Toc69392100)

[Bibliography 42](#_Toc69392101)

# Chapter 1 Introduction

U2RANS is a three-dimensional (3D) unsteady and unstructured Reynolds Averaged Navier- Stokes solver (Lai et al., 2003). The U2RANS solver was developed by Dr. Yong Lai and has been well validated and successfully applied to many research and engineering projects. Here is the link of a brief introduction to [U2RANS](https://www.usbr.gov/tsc/techreferences/computer%20software/models/u2rans/index.html).

The original U2RANS solver requires a 3D mesh to conform to domain geometry, termed the terrain-conforming method in this report. Despite the widespread use of the terrain-conforming method in Computational Fluid Dynamics (CFD) models, generation of a high-quality mesh is still a challenging task in 3D modeling of flows over complex terrain. A very refined near-wall mesh must be used to resolve the boundary layer to produce an accurate solution. The stability and accuracy of the terrain-conforming method highly depend on the mesh quality. In addition, the mesh size increases rapidly with the increase of Reynolds numbers.

In this report, we provide an alternative model to the terrain-conforming method, the U2RANS solver based on the immersed boundary (IB) method (U2RANS IB method), in which terrains are embedded in a background mesh. The boundary conditions on embedded terrain surface are implicitly represented by modifying the governing equations of the flow. Special numerical treatments are developed to implement the solid boundary conditions for turbulent flows near complex terrains and solid objects. This model is user-friendly for practical applications in hydraulic engineering because mesh generation becomes relatively simple, as only a background mesh is needed, and mesh quality is easy to control.

The organization of this report is as follows. Chapter [2](#_bookmark1) provides some instructions on the code compilation. Chapter [3](#_bookmark4) describes the algorithm of the IB method implemented in the U2RANS solver. In Chapter [4](#_bookmark26), the U2RANS IB model is validated against experimental data and its applications to flow around a 3D complex structure are demonstrated.

# Chapter 2 Instruction on compilation

The whole program of U2RANS IB method is divided into two parts, u2rans\_pre and u2rans, source files of which are stored in /Pre and /Src, respectively. u2rans\_pre is used for the pre-processing, such as background mesh processing and IB processing. u2rans is a flow solver including discretization of the governing equations, implementation of embedded terrain boundary conditions, and matrix solvers for momentum and continuity equations.

This chapter describes the development environment used for this project. It is provided as a reference. Other compiler versions and configurations may also work. The target run environment is Windows. However, during the development, the Linux environment was occasionally used for better debugging and more efficient editing. The development environment in both systems is included.

## **2.1 Windows**

To compile the code in Windows, the following software were used:

Microsoft Visual Studio Community 2015. It seems that the latest version of VS may have some problem with certain version of Intel Fortran.

•

Intel(R) Visual Fortran Intel(R) 64 Compiler. Recent or the latest versions should work.

•

ParaView 5.4.1, or lateer version, can be downloaded from <https://www.paraview.org/download/>. It is used for the visualization of the simulation results.

•

The source codes in /Pre and /Src are compiled separately.

## **2.2 Linux**

In Linux, the project is organized with make. The Linux distribution used is Ubuntu. Software package used in this project includes:

* + - make
    - gfortran

ParaView 5.4.1, or lateer version, can be downloaded from

•

<https://www.paraview.org/download/>.

Command ”make” is used for compilation. The source codes in /Pre and /Src are compiled separately.

## **2.3 List of changes in original files**

There are some possible bugs in the original source files. To fix them, the corresponding changes are as follows:

Bug 1 In subroutine FS\_MOVE1 (in file fs\_pmethod.F90)

1 USE mod\_frees, ONLY: IDRY

where IDRY does not exist in moe\_frees. In fact, the module mod\_frees is not used in fs\_pmethod or anywhere else. The simple solution is to delete or comment out this line. The original file is saved to fs\_pmethod.f90.old.

Bug 2 The same problem in srh3d and ss3D\_read\_input.

# Chapter 3 IB method implementation in U2RANS

This chapter will describe the implementation of the IB method algorithm in U2RANS. To reduce the length of this report, not every detail is included. Instead, the essential parts and algorithms are presented so the reader can quickly capture the main idea of the code. The content of this chapter is organized in the following parts:

* + - The algorithm of IB method.
    - The processing of the embedded terrain surface in u2rans\_pre.
    - The code structure of U2RANS IB method.

To better read the document, some texts are highlighted to signify the following:

* + - purple for PROGRAM, MODULE and SUBROUTINE.
    - green for variable and parameters

## **3.1 Basic algorithm for IB method in U2RANS**

The IB method in U2RANS uses a discrete forcing approach, where a discrete-forcing term is added in the discretized Navier-Stokes equations to represent effect of the embedded terrain. The forcing term is cell-based for an unstructured mesh. The IB cells are cells cut by the immersed surface, and their cell centers are located on the fluid side (yellow cells shown in Figure 1). Three different points are identified: (1) the IB cell center (IB); (2) the hit point (HP), which is the intersection of the immersed surface with its normal line through the cell center; and (3) the image point (IP), which is the point on the extended line of the normal vector through the cell center in the fluid field. Three different characteristic lengths are identified: (1) wall distance : the distance from the IB cell center to the corresponding hit point; (2) image distance : the distance from the image point to the corresponding hit point; (3) and IB cell length : the minimum dimension of all IB cells. To enforce the turbulence model conditions at the IB cell centers, the following steps are carried out:

|  |  |
| --- | --- |
| (**a**) | (**b**) |

Figure 3.1: A schematic illustrating the IB models using a two-dimensional (2D) mesh. (a)

; (b) .

* + 1. Based on the log-law velocity profile, the dimensionless distance and are computed (iteratively) as:

|  |  |
| --- | --- |
| , | (1) |

where and .

* + 1. The shear velocity on the immersed surface is calculated as:

|  |  |
| --- | --- |
| , | (2) |

* + 1. The tangential flow velocity , , and at the IB cell center are calculated based on :

|  |  |
| --- | --- |
|  | (3) |
|  | (4) |
|  | (5) |

where and . The implementation of and are similar to the wall functions for and used in OpenFOAM (Greenshields, 2019).

* + 1. Fix the values of flow variables on the IB cell centers when solving the momentum equation and the transport equations for and .
    2. Adjust the flux balance on the faces of IB cell centers on the solid side for mass conservation.

The key to the IB treatment is the estimation of the shear velocity , i.e., the calculation of . Considering the nonlinear and discontinuous nature of velocity profile between the laminar sublayer and log-law sublayer, the result of converges to different values if different velocity profiles are used. To keep the consistency when estimating the shear velocity of all IB cells, Eq. 1 assumes that the image points are located within the log-law sublayer such that only the log-law velocity profile is used in the iteration. Generally, the image point distance is proportional to the wall distance (for example, = 3) as shown in Fig. 3.1(a). However, the wall distance is arbitrarily distributed around the immersed surface, making it impossible to guarantee that the image point is located in the logarithmic sublayer. In addition, an extremely small value of may result in numerical instability and even divergence of the model. In this work, is set to be proportional to the minimum dimension of each IB cell = 3 ). Consequently, the image points are uniformly distributed along the immersed surface as shown in Fig. 3.1(b). The numerical instability due to the small value of is avoided. A similar implementation has been used in [Tamaki et al.](#_bookmark46) ([2017](#_bookmark46)) and [Capizzano](#_bookmark43) ([2011](#_bookmark43)) for model and SA model.

Another problem in the near-wall treatment of the immersed boundary method comes from the inconsistency when the IB cell center and its corresponding image point are located in different sublayers of the velocity profile. Although all image points are designed to be in the log-law layer, the wall distance is still an arbitrary value and the IB cell center may be in the laminar sublayer.

To address this problem, a modified velocity profile is used when the IB cell center is in the laminar sublayer. We assume a linear velocity profile between the wall and the image point (Eq. 3) such that the first derivative of the velocity with respect to the wall distance is still a constant ( Fig. 3.2(a)). This method was proposed in [Tamaki et al.](#_bookmark46) ([2017](#_bookmark46)) to correct the mass flux on the cell boundary by using a slip velocity boundary condition. Here, it is used to extend the logarithmic velocity profile to the wall when the IB cell center is in the laminar sublayer. The eddy viscosity is also modified to be a constant between the wall and the image point (Fig. 3.2(b)). Thus, , and are constant in this region (Eq. 4 and 5). This modification is based on the balance of shear stress on the boundary:

|  |  |
| --- | --- |
| (**a**) | (**b**) |

Figure 3.2: (a) Velocity profile; (b) Eddy viscosity profile. The solid lines are the original profiles. The dash lines are the modified profiles between the image point and the wall.

## **The processing of IB in u2rans pre**

This section illustrates the pre-processing of immersed boundary, including identification of three types of cells (IB, live, dead cells) and points (IB, IP, hit points) used the IB algorithm, and a searching algorithm based on the octree.

## **Find cut (intercepted) cells**

Finding the cut cells, which are the cells intersecting with an IB surface, is perhaps the most difficult part of the whole algorithm. The procedure is as follows:

**Step 1** Import an STL surface, which is immersed in the background mesh.

**Step 1.1** For each triangle in the STL, 3 vertex points and 1 normal vector are calculated.

**Step 1.2** Calculate the center of each triangle from the 3 vertex points.

**Step 2** For each edge in the background mesh.

**Step 2.1** Check if the edge is cut by any triangle (use Octree to speed up the searching).

**Step 2.2** Mark cut edges and corresponding triangles.

**Step 3** Mark the intersected cells who have edges intersecting with the immersed boundary.

**Step 4** User needs to specify a point which is in the fluid domain. With this fluid point, search for live/dead (fluid/solid) cells. Live cells are those on the same side of the immersed boundary as the fluid point.

**Step 5** Search for IB cells which are live cells but intersect the immersed boundary. This step needs to determine whether a cell center is inside fluid region or not.

**Step 5.1** Reorient triangles and make sure the normal direction is towards fluid region.

**Step 5.2** Find the nearest triangle for each cut cell which needs to calculate the distance between a triangle plane and cut cell centers).

**Step 5.3** Check whether a cut cell is in fluid domain. If yes, it is an IB cell. Otherwise, label the neighbouring live cells as IB cells and label this IB cell as a dead cell.

**Step 5.4** For each IB cell, find the hit point and corresponding triangle.

Fig. [3.3](#_bookmark10) provides a 2D schematic of how to determine the cut and fluid cells. The embedded terrain is represented by a STL surface mesh (Yellow lines). First, we find the cells which has at least one face cut by the STL surface mesh (red cells). The fluid cells are the cells on the side of the fluid point (green cells).

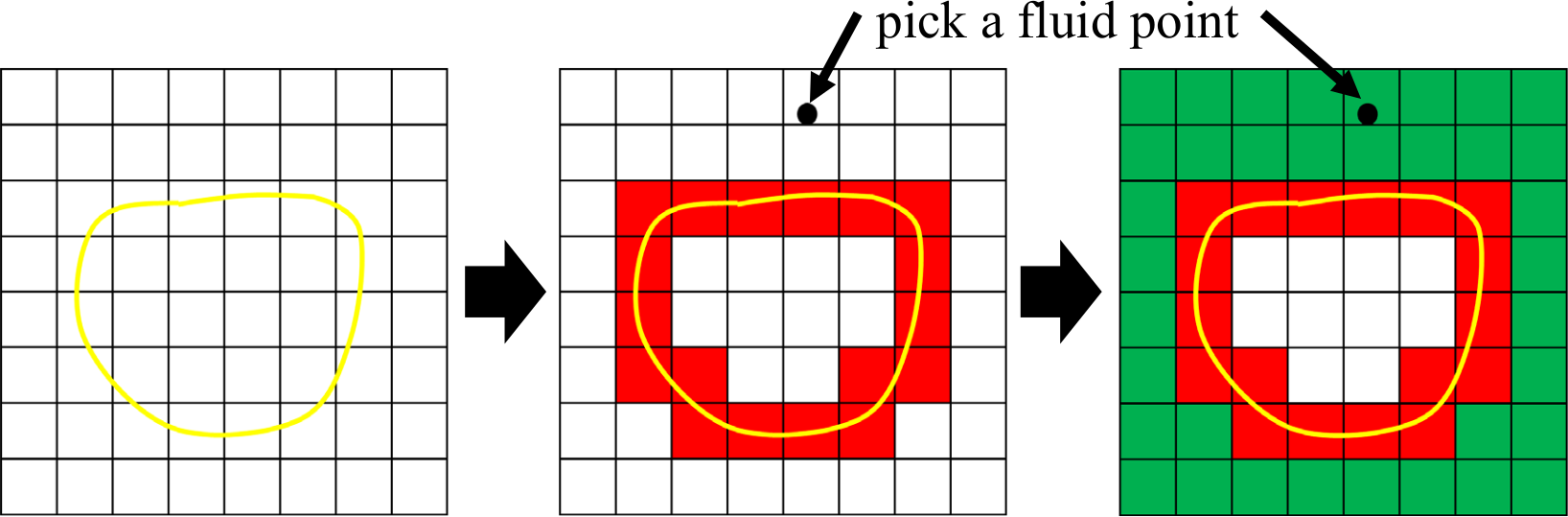


Figure 3.3: 2D schematic of cut and fluid cells.

A good quality of the STL surface mesh is essential to determine the IB, live, and dead cells. Fig. [3.4](#_bookmark12) provides some examples of the STL surface meshes used in the IB method.

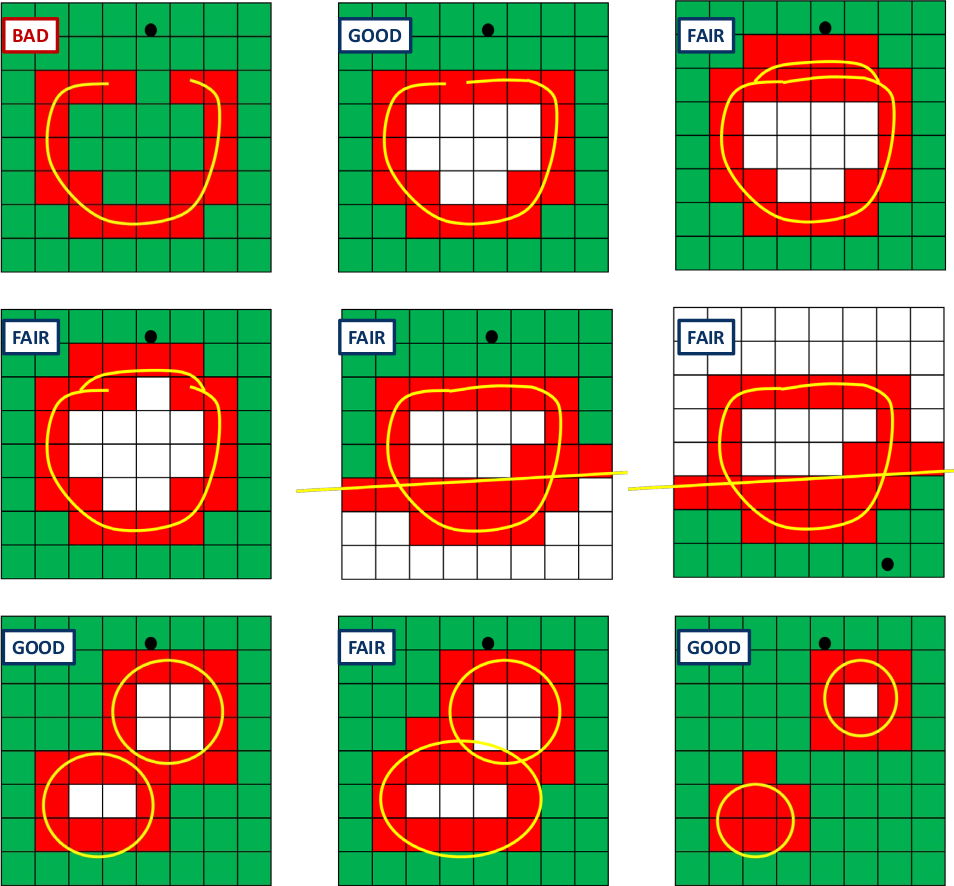


Figure 3.4: Examples of the STL surface meshes used in the IB method.

A good STL mesh should avoid holes or openings on its outside boundary. If the feature of boundary is well represented, a coarser STL mesh is preferred to reduce computational time in pre-processing.

### **Find IB, live, and dead cells**

This section describes how to find and properly mark IB, live, and dead cells.

**Step 1** Collect all the cells intersected by the STL mesh, .

**Step 2** Label all cells, excluding the intersected cells, in fluid (solid) region as live (dead) cells.

**Step 3** For each intersected cell,

* + - * If the cell center is in the fluid region, it is labeled as IB cell;

Otherwise, it is labeled as dead cell, and re-label its adjacent live cells (sharing boundary faces) as IB cells.

•

### **Octree algorithm**

The basic idea of octree algorithm is to recursively subdivide the domain into eight octants, so as to facilitate the searching algorithm. It is from [https://github.com/dongli/](https://github.com/dongli/fortran-octree) [fortran-octree](https://github.com/dongli/fortran-octree). It is noted that this code uses the MIT license. If U2RANS\_IB is to made public, proper license laws and rules should be followed. In particular, the MIT license file of that code should be added the distribution.

The octree algorithm is used in finding edges cutting by triangles and finding triangle edges cut by mesh faces. However, there is potentially a problem in finding mesh edges cut by triangles. If the triangle size is much larger than the mesh cell size, the edges cutting inside the large triangle cannot be detected. An example of such case is shown in Fig. 3.5.

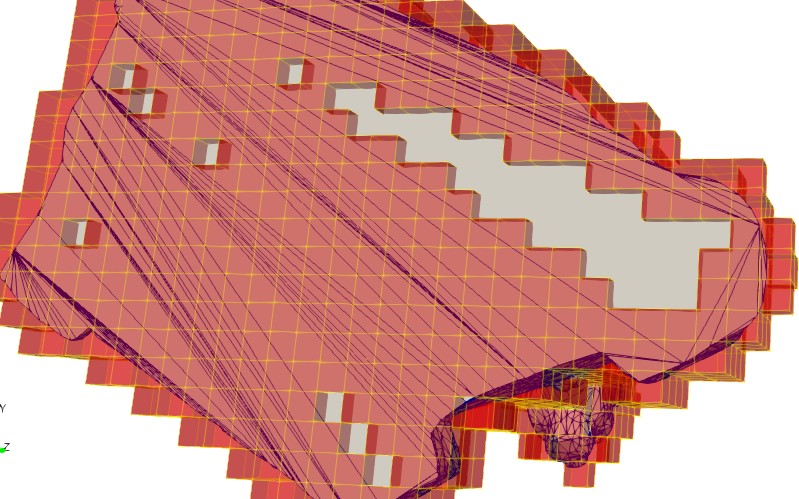


Figure 3.5: A screenshot of cut cells and a complex surface mesh. Yellow lines and red color denote the grid edges and cells, respectively. Blue lines and white color denote the triangulated surface mesh edges and cells, respectively.

The original code only uses face/triangle centers to initialize the octree, and uses edge center to find the nearest face/triangle centers, which sometime misses large size and twisted faces/triangles. A different algorithm for edge to find nearest face needs is implemented as follows:

**Step 1** Initialize the global block using the bounding box (bbox) for surface mesh or grid mesh.

Make a slight extension in all three directions to ensure the bounding box contains the simulation domain.

**Step 2** Divide the block into eight subdivisions for each depth.

**Step 3** Instead of marking only the face/triangle center’s location, mark the locations of the vertexes of face/triangle in each subdivision.

**Step 3.1** To do that, we need to obtain bbox for each face/triangle.

**Step 3.2** Find any subdivision that overlaps with this bbox. The problem at this step is essentially the collision detection of two cubes which have the same orientation. A simple method is to check for each axis whether

If this condition is met, the two boxes overlap.

**Step 4** Subdivide each subdivision in to subsubdivision or children node until reaches the specified maximum level.

**Step 5** For each given edge, find its depth-1 subdivisions, which can be multiples.

**Step 5.1** Since an edge is a line instead of a point, we need to find out all the subdivisions that overlaps with this edge. A straightforward method is to do the same as in [Step 3.2](#_bookmark12).

**Step 6** Collect all faces/triangles that are in the found overlapped depth-1 subdivisions. Make sure to remove all the duplicate ones.

The following is a list of modifications to the original octree code:

The second term in point\_type needs to be changed from real(8)::x(3) to real(8) x(2,3)

* + - * Added a new subroutine to check if two bbox overlap.
      * bbox is used and checked instead of points.

### **Finding hit points**

For each IB cell center, it is computationally expensive to find its corresponding hit point. Indeed, finding hit point is exactly the same as finding the intersection points when checking if a face or triangle is intersected by an edge, which can be found in CHECKHIT. Here, the edge direction complies with the triangle normal vector using the right-hand rule. The problem becomes how to find the nearest triangle. The procedure can be found as follows:

**Step 1** Use octree to find a list of triangles within a certain range of IB cell center. The range should be larger than the cell size. In this octree, it checks whether a point is in a bbox.

**Step 2** Calculate the distance between IB cell center and the triangles in the list.

Calculate the hit points for each triangle, and check whether it is inside the triangle. A project from a point onto a triangle may be outside of the triangle.

•

If the hit point is inside the triangle, the distance is calculated between the hit point and the IB cell center.

•

If the hit point is outside the triangle, the distance is calculated by finding the nearest point on the three edges (implemented in PT2TRI and PT2LINE).

•

**Step 3** The hit point is the point with the shortest distance

### **Mapping between IB cells and STL**

Mapping of flow variables and other quantities between IB cells and IB surfaces is needed in the program. For example, to integrate the quantities over the IB surface, one needs to know the distribution of the quantities on the surface. Here, the mapping from IB cell centers to triangle face centers is described.

**Step 1** Find hit triangle for each hit point

This is done within the function for finding hit points.

**Step 2** Check number of hit points in each triangle face (could be zero).

**Step 3** If the number is less than a certain value, such as seven, search its neighbour triangle faces to find more hit points. This search gradually expands until the specified number of hit points is found.

In order to get the neighbours of each triangle, special addressing of these triangles is required. The STL format only stores the coordinates of each triangle, not the topological relationship among triangles. Thus, vertex coordinates of different triangles need to be compared at a time to check whether they share the same vertex.

•

**Step 4** Find the addressing between triangle face and hit points. In other words, each hit point belongs to one triangle.

**Step 5** Calculate the distance-based weightings for interpolation from hit points to triangle vertexes.

**Step 6** Calculate area vector for each triangle for surface integration.

## **Data file generated by u2rans pre for use by u2rans**

u2rans\_pre is for pre-processing. INTERA() is a major subroutine to be called in u2rans\_pre to process all the input information. In INTERA(), all the input commands can be typed interactively or using a input file, usually named as casename\_SIF.dat. The input commands can be saved as a file named as casename\_SOF.dat, which can be renamed and reused as \_SIF.dat file.

casename.dat is the data file which is created by u2rans\_pre, and for use by u2rans. It stores all the information needed in U2RANS computation.

Here is a summary of the parameters written in the data file:

I2D Ipolar Itrans Iheat Icond Irest Iengine

•

Whether it is a 2D mesh, polar coordinate, transient flow, energy conservation, heart transfer, HotStart Simulation, Engine related.

Ntime Niters Igrav Irotrf prt%npoint

•

Simulation time, iteration number, whether gravity force is turned on, whether it is a rotating reference, number of monitor points.

Igform Igraph Ioutp Ipref mod\_manning

•

Grid format (4 is for FLUENT mesh), graphic write-out format (5 is VTK), unknown, Reference-Pressure-Point, whether Manning approach is used.

NvertT NcellT NfaceT Nbdf Ncycle Ncht NmvZon Nblock Nmvpt

•

Grid information: total number of vertexes (nodes), cells, faces, boundary faces, PERIODIC boundary pairs, Conjugate Heat Transfer, move zone, blocks, move points.

NBOUND NMDOTB NINVLTV NEXITP NMDOTF NTOTPF NCLAKE

•

Number of boundaries, inlet or outlet BCs, inlet-velocity BCs, pressure-outlet BCs, INLETM or EXITM BCs, total pressure boundary faces, for lake model.

IFREES IBFSUR IFSUR JFSUR

•

If free surface, free surface block ID, free surface point x, free surface point y.

NCUT IDRY NFAN NCLFAN NPORO NCPORO NINIT IPHASE NTWALL

•

Cut-away regions, whether it is a wet-dry simulation, number of fans, number of face cells, number of porous media region, number of porous media cells, Constant-Value-Initial- Condition-for-Sub-Domains, whether it has solid phase, the number of thin walls.

ModRo Vis Kap Cp Turb Comb KTurb Nsp Nmix

•

Density mode (1 for constant), viscosity mode (1 for constant), molecular thermal conductivity mode (1 for constant), Cp mode (1 for constant), turbulence-model-selection (1 for laminar), chemical reaction mode, TURBULENT-THERMAL-CONDUCTIVITY (1 for constant), number of chem species, number of mixing species.

* + - afterwards, there is a line of unlabeled variables

ADEN,WMOL,ANU,AKAPA,PRLH,ACP,HTOTAL,PREF,TREF,ZFSUR,ZDATUM,AKAPAT,AKAPAV,PRTH

Density, molecular-weight, viscosity, constant-thermal-conductivity, Prandtl number, Cp, total enthalpy H, reference pressure, isothermal-fluid-temperature, initial free surface elevation relative to Zdatum, Z-coordinate used as datum line, turbulent-thermal-conductivity horizontal, turbulent-thermal-conductivity vertical, Prandtl-Number-for-turbulent-thermal- conductivity

Tstart Dt Zdry

•

start Time, time step, dry elevation

ISOLAR IDSOLA IDATEM IDCCOV IDRHUM IDWIND IDSECC

•

solar model, solar model ID, air-temperature, cloud-cover, relative-humidity, wind Speed, unknown.

* + - NMASS unknown.
    - Lineqs, Ninner about CGS solver.

NTDV NTDVTB NTDVBF NTDVSZ NCLUT1 NCLUT3

•

Double number of time-varying-functions (TDV), number of time-varying-functions (TDV), number of TVF faces, size of TVF points, for clutch pack interfaces.

NTAB NTABSZ

•

Number of table functions, number size of table functions.

Iflag Rflag arrays

•

Unknown indicator flags (size of 20), RFLAG(1)=density1 RFLAG(2)=density2 (size of 20)

Nclvt Nfcvt Nclfc Ndcrs Npl3d Nfpl3d

•

Summation of node numbers for all cells, summation of node numbers for all faces, summation of face numbers for all cells, size of Nfcvt, last two for Structured Grid.

Nctype Nftype

•

Number of cell types (1=2DQ 2=2DT 3=3DH 4=3DT 5=3DP 6=3DA), number of face types (1=2PT 2=4PT 3=3PT 4=ARBITRARY-PT).

The following is mainly for the mesh information:

LCLVT1 stores the starting and ending index of nodes of a cell. LCLVT1(i) and LCLVT1(i+1) mean the starting and ending index of cell i. However, in the input file, LCLVT1(i) has been converted to LCLVT1(i+1)-LCLVT1(i).

•

* + - LCLVT2 stores the node IDs of each cell.
    - LFCVT1 and LFCVT2 are very similar to LCLVT1 and LCLVT2 but node IDs for faces.
    - LCLFC1 and LCLFC2 are face IDs for cells.

LOUT left-right relation between element and face, for faces, 1 for left, 2 for right. It is related to LCLFC2.

* + - LFCCL stores face’s cells.
    - Ibnd has BC type, boundary face ID, boundary cell, boundary patch ID.
    - Rbnd stores the values for boundary faces (U V W P T K E).
    - LVARPT -Write out Print-out variable list.

## **IB cell information**

IB cell information will be written out to casename\_IBM.dat file and used in U2RANS. It includes the following:

* + - IB flag, IBflag
    - number of IB cell ID, NIBT
    - list of hit points (their coordinates in three coordinate directions), LHITPTX, LHITPTY, LHITPTZ
    - list of image points (their coordinates in three coordinate directions), LIMPTX, LIMPTY, LIMPTZ
    - stencil of image points and their interpolation weights, LSTENCIL1, LSTENCIL2, WLSTENCIL2
    - list of hit triangles for each hit points, LHITTRI
    - stencil of hit triangle addressing and their interpolation weights, LADDRESS1,LADDRESS2,WLADDRESS1

## **List of added modules or subroutines**

The principle guideline in adding IB functionality to U2RANS is that the additional code is kept independent from the original code, to the extent possible. This section presents the list of added modules and subroutines in both the preprocessing and the main code.

### **In the pre-processing**

* + - * mod\_ibm is a module to declare global variables for IB method
        + STLNAME name of STL file, located in STLfiles
        + NTRIT total number of triangles in STL file
        + ltri coordinates of 3 points for each triangle in STL file
        + ltrin normal direction for each triangle
        + ltric center for each triangle
        + tribbmax maximum coordinates of the bounding box for triangle points
        + tribbmin minimum coordinates of the bounding box for triangle points
        + LSTLADDR1 triangle face-face addressing
        + LSTLADDR2 triangle face-face addressing
        + LSTLPT list of stl points
        + NLSTLPT number of stl points
        + LTRI2 renumbered triangle
        + LINTRI if trii is in mesh
        + NedgeT total number of edges
        + LEDVT edge list consists of nodes, in sequence of face nodes, (2NedgeT)
        + LCLED1 cell-edge similar to LCLVT1 and LCLFC1
        + LCLED2 cell-edge addressing
        + LFCED1 face-edge similar to LCLVT1 and LCLFC1
        + LFCED2 face-edge addressing
        + LEDCL1 edge-cell similar to LCLVT1 and LCLFC1
        + LEDCL2 edge-cell addressing
        + LEDFC1 edge-face similar to LCLVT1 and LCLFC1
        + LEDFC2 edge-face addressing
        + IBflag flag of IB cell

2, fluid cell

1, IB cell

0, solid cell

* + - * + LIBCL list of IB cell ID
        + NIBT number of IB cells
        + LHITPTX, LHITPTY, LHITPTZ list of hit points X, Y, Z coordinates
        + LIMPTX, LIMPTY, LIMPTZ list of image points X, Y, Z coordinates
        + LSTENCIL1 and LSTENCIL2, list of image points stencil
        + WLSTENCIL2, list of stencil weights
        + WLSTENCIL3, list of stencil weights for hit points
        + LADDRESS1, addressing of stl triangle and surrounding hit points
        + LADDRESS2, addressing of stl triangle and surrounding hit points
        + WLADDRESS2, list of LADDRESS weights, reversed-distance based
        + cross\_product(a, b) return cross product of vector a and b
        + magnitude(a) return magnitude of vector a
        + XYZ(VT,X,Y,Z) return point vector with its ID VT
        + EXPORT\_FOAMFILE(X,Y,Z…) write mesh and cell data in OpenFOAM format. There is one problem in the format of U2RANS: boundary face orders need to be reversed, otherwise it can not be read by paraView or Tecplot.

STLREAD(WORDS,IPOINT) is a subroutine now called in INTERA() to read stl files, obtaining global variables ltri, ltrin, ltric, points for each triangle, normal direction for each triangle, center for each triangle

•

FINDEDGE() is a subroutine called in FINDIB() to calculate edge information and address-ing, including LEDVT,LFCED1,LFCED2,LEDFC1,LEDFC2,LCLED1,LCLED2,LEDCL1,LEDCL2. The

•

algorithm in this subroutine has been optimized.

* + - * CHECKHIT(TRII,EDGEI,ERR1) is a subroutine called in FINDIB() to check if the face edge is intersected by the triangle. Let’s note edge’s two ends as and , triangle center as , triangle’s three vertexes as , and triangle normal as . The algorithm implemented

in this subroutine is as follows:

1. determine if two ends of the edge on the same side, i.e.,

If not, then continue to find the intersection. If yes, there is no intersection.

1. find the intersection point with
2. determine if is inside the triangle with

is inside the triangle only if 0 ≤ *α, β, γ* ≤ 1

CHECKHIT1(TRII,FACEI,ERR1) is similar to CHECKHIT(TRII,EDGEI,ERR1). However, it checks if the triangle edge is intersected by a face.

•

octree is a module from <https://github.com/dongli/fortran-octree> (MIT license). This octree algorithm speeds up the searching in FINDIB().

•

* octree\_init(max\_num\_point, max\_depth, bbox): initialize the octree. max\_num\_point is not used, just set as 10. max\_depth is the maximum depth of the octree and also set as 10. bbox is the domain size of points.
* octree\_build(points): build the octree with the supplied points.
* octree\_search(point\_search, distance, nnbidt, nbids): searching with the octree. nnbidt is the total number of neighbour point ID, and nbids is list of neighbour points. distance will affect the searching speed. It needs to be carefully estimated.
* octree\_final(): finalize the octree.

This octree algorithm is used by both CHECKHIT and CHECKHIT1.

* + Summary of the FINDIB() algorithm

1. use FINDEDGE() to find edge addressing and information
2. use CHECKHIT() to find grid edges cut by triangles
3. use CHECKHIT1() and the above cut edges to find cut faces, which have at least two cut edges or cut at least one triangle edge
4. find cells which has at least two cut faces.

### **In the solver**

* + - * ibm\_set\_memory: allocate memory for IB method-related integers.

ibm\_read(): read from casename\_IBM.dat

•

including IBflag,LHITPTX,LHITPTY,LHITPTZ,LIMPTX,LIMPTY,LIMPTZ LSTENCIL1,LSTENCIL2,LIBCL,WLSTENCIL2

* + - * ibm\_calc(): reconstruct velocity and pressure at image points.
      * IBM\_WALL\_FUNC() reconstruct velocity based on IB wall function. It uses some functions such as uPlus\_two\_layer() and dudy\_two\_layer(), which are in module mod\_ibm.

## **Summary of code modification**

### **u2rans pre**

* + - * Add an extra option to Results-Output-Format, ”6=foamFiles”
        + Modifications are made in Pre/sec\_gen.f90.
        + Accordingly, in Src/graph.f90, a new subroutine FOAM\_WRITE(IR,R) will be called if IGRAPH =6. This new subroutine will be introduced later.

In order to read STL files, a new subroutine STLREAD(WORDS,IPOINT) is called at the end of all inputs.

•

The only trigger for IB method is to add the following lines at the end casename\_SIF.dat (for steady flow)

// Specify-a-flow-rate-at-inlets? (empty or 0 for NO, 1 for YES)

// READ-STL-SURFACE-FILE

READSTL cylinder\_triplesize.stl

The STL file has to be put in the directory STLfiles/.

For unsteady flow, the following is an example to specify IB mwthod:

// monitor point

// Intermediate Result Output Control: INTERVAL(hour) OR List of T1 T2 ... EMPTY means the end

-1

// READ-STL-SURFACE-FILE

READSTL cylinder\_triplesize.stl

A module called mod\_ibm is created for defining global variables for immersed boundary method. It defines the information for STL as well as edge-related addressing of the computational mesh. It also contains some basic functions such as cross\_product(a, b) and magnitude(a).

•

In intera.f90, before writing out data, a subroutine FINDIB() is called. This is the core subroutine of the whole IB method algorithm in U2RANS.

•

Basic steps in FINDIB() are listed as follows:

**Step 1** Make edge addressing for the computational mesh by calling subroutine FINDEDGE().

**Step 2** Find mesh edges cut by triangle faces (creating octree for triangles faces)

**Step 3** Find mesh faces cutting triangle edges (creating octree for mesh faces)

**Step 4** Find cells with cut-faces or cut-edges (at least 2 cut-faces for cut-cell or 2 cut-edges for cut-face)

**Step 5** Find the location (cell ID) of the fluid point which needs to be specified. This fluid point needs to be specified by the user (hard-coded for now).

**Step 6** Find all the fluid cells by searching from a fluid point, ending at boundary or cut-cell.

**Step 7** Move cut-cells whose cell center has to be in the fluid region to IB cell list.

**Step 8** Make sure each IB cell has at least one live (fluid) neighbour cell. (not necessary but helps to increase efficiency).

**Step 9** Calculate IBflag to represent fluid/live cells (=2), IB cells (=1), solid/dead cells (=0).

**Step 10** Find hit-point for each IB cell. Hit point is the nearest point on STL surface from a IB cell center. It is assumed the vector between hit point and its corresponding IB cell center is perpendicular to the STL surface.

**Step 11** Find image point which is on the extended line of the vector between hit point and IB cell center. It should be located in fluid cells.

**Step 12** Find the surrounding non-solid cells of image point, which are named as the interpolation stencil.

**Step 13** Calculate the weight of each element in the stencil based on reversed-distance.

**Step 14** Write out OpenFOAM format files and IBflag to foamFiles/. IBflag should be 2 everywhere if there is no STL.

* + - * WRITE\_IBM() is used to write out IB information to casename\_IBM.dat.

### **u2rans**

Subroutine IBM\_SET\_MEMORY() is called in the main program after SET\_MEMORY() to allocate memory for IBM. It needs to read the number of IB cells and the stencils.

•

Integer variables are defined in the module mod\_ibm. Module mod\_ibm also provides some basic functions as well as exporting.

•

Relative array variables and indexes are defined in module ibm\_mod\_indx(), which is similar to mod\_indx().

Subroutine IBM\_READ() is called in the main program to read IB information from

•

casename\_IBM.dat file.

Subroutine IBM\_CALC(IR,R) is called in solve() before SOLVEM(\*). This subroutine is used to reconstruct velocity in IB cells and set dead values in dead/solid cells.

•

* For Dirichlet BC, reconstruction is conducted by calling IBM\_RECON\_DIRICHLET(\*).
* For Neumann BC, reconstruction is conducted by calling IBM\_RECON\_NEUMANN(\*).
* Dead cell values are set as zero by calling IBM\_SETDEAD(\*). Indeed, only velocity is reconstructed and set dead. Pressure has its own special treatment described below.

Instead of pressure, pressure gradient is reconstructed by calling IBM\_RECON\_DIRICHLET(\*) after gradient(\*) in solve1() and solvep(). Pressure gradient has three components.

•

Another important treatment is to fix velocity in IB cells and dead cells. It is conducted in IBM\_FIXUVW(\*) called by soluvw(). It is very similar to CUT\_FIXUVW(\*) and also located after CUT\_FIXUVW(\*).

•

CALL GRAPH(2,IR,R) is added at the end of every time step in solve() to be able to save results.

•

### **3.7 Code structure of IB implementation in U2RANS**

The IB implementation and modification to the original U2RANS code may be best visualized with flow charts and figures. Thus, this section includes several of these figures. In Fig. [3.6](#_bookmark25), the overview of the IB implementation is presented. It shows the additional files for IB method in yellow. Fig. [3.7](#_bookmark26) shows the IB implementation in u2rans\_pre. The added extra IB method-related functions are in red. Fig. [3.8](#_bookmark27) shows the flow chart of the FINDIB() subroutine and the two extra modules mod\_ibm and octree. Lastly, within the solver, the changes and additions are shown in Fig. [3.9](#_bookmark28). The major places where IB method-related function calls are labeled.

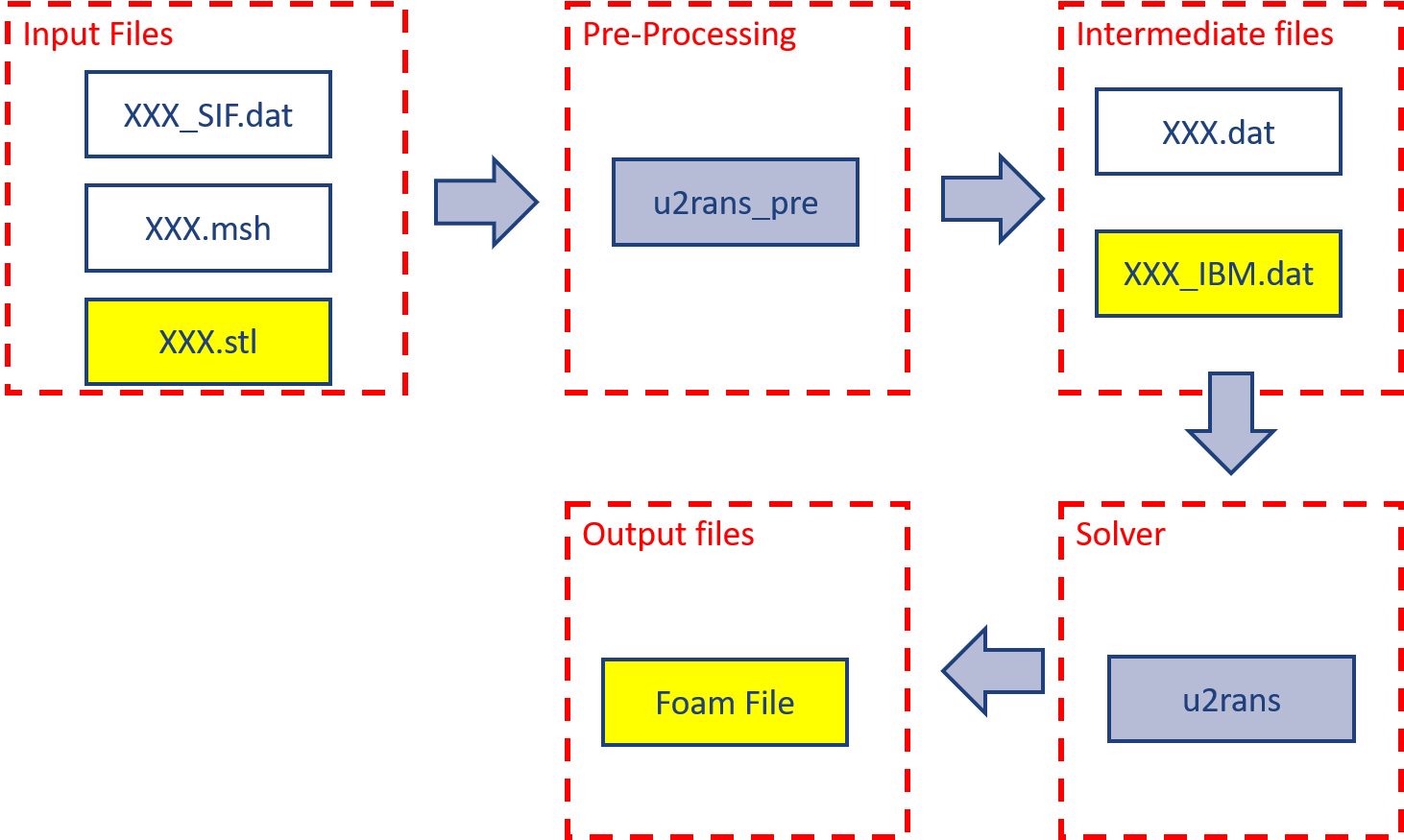


Figure 3.6: A big picture of IB implementation in U2RANS code.

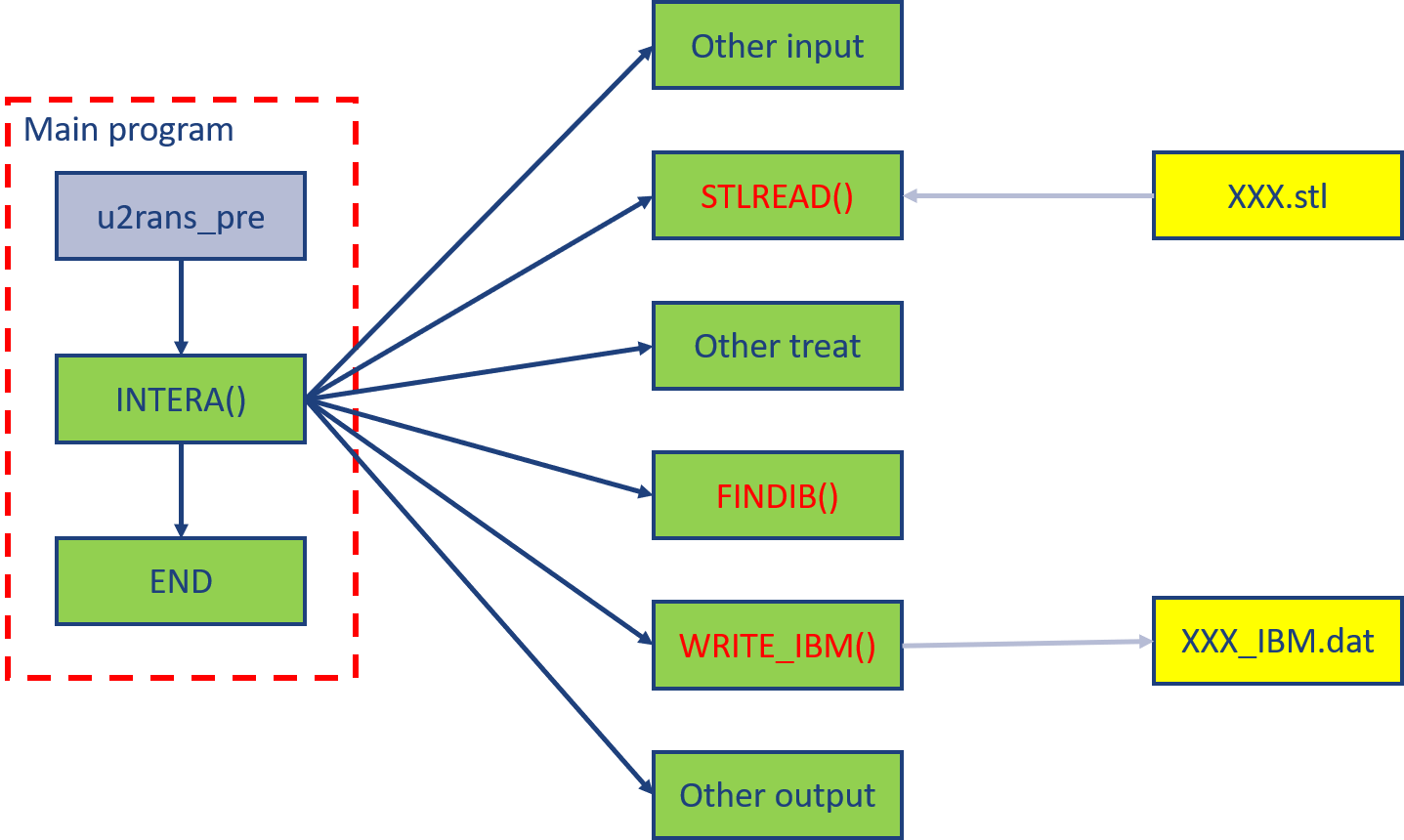


Figure 3.7: IB implementation in pre-processing of U2RANS code.

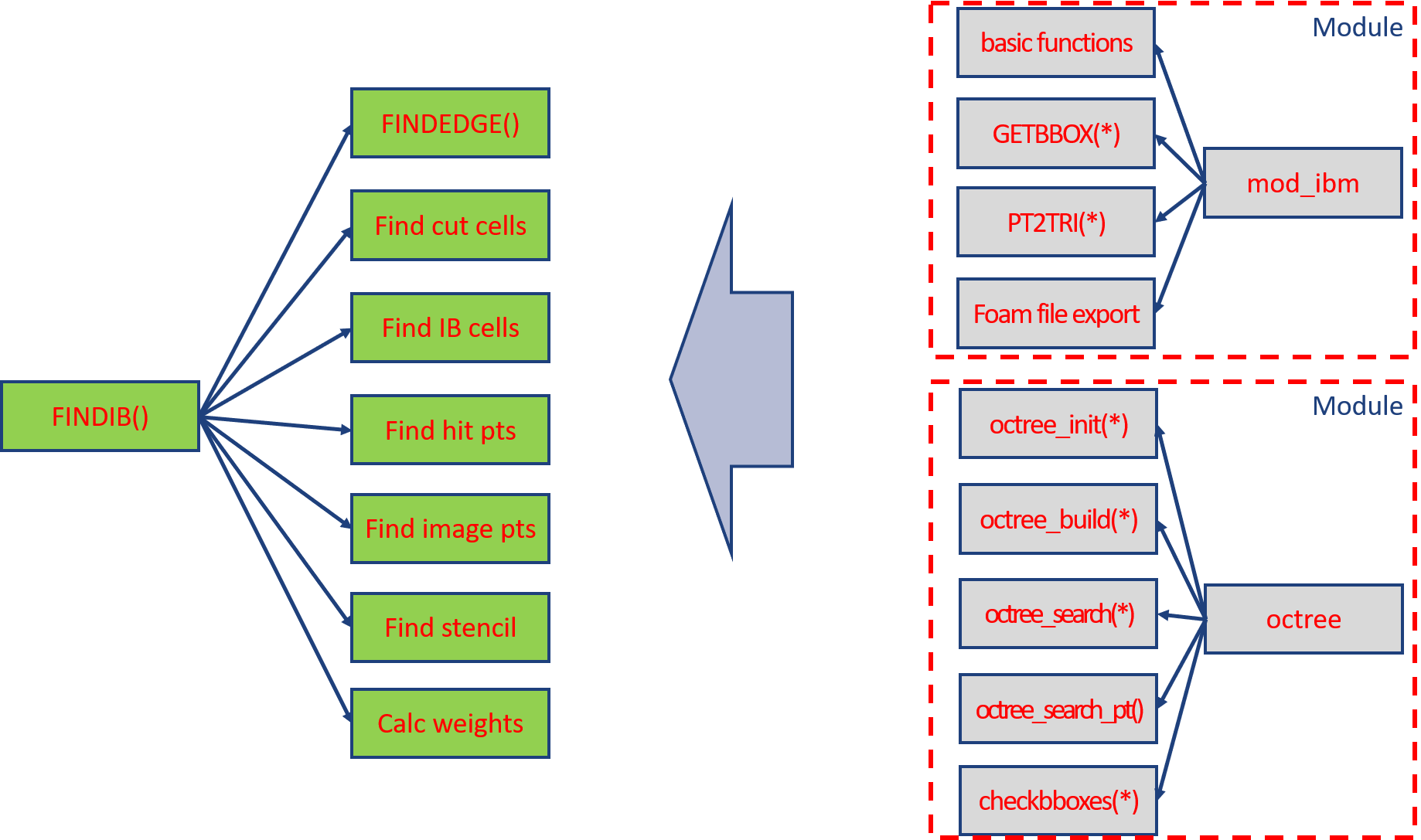


Figure 3.8: FindIB function in the pre-processing of U2RANS code.

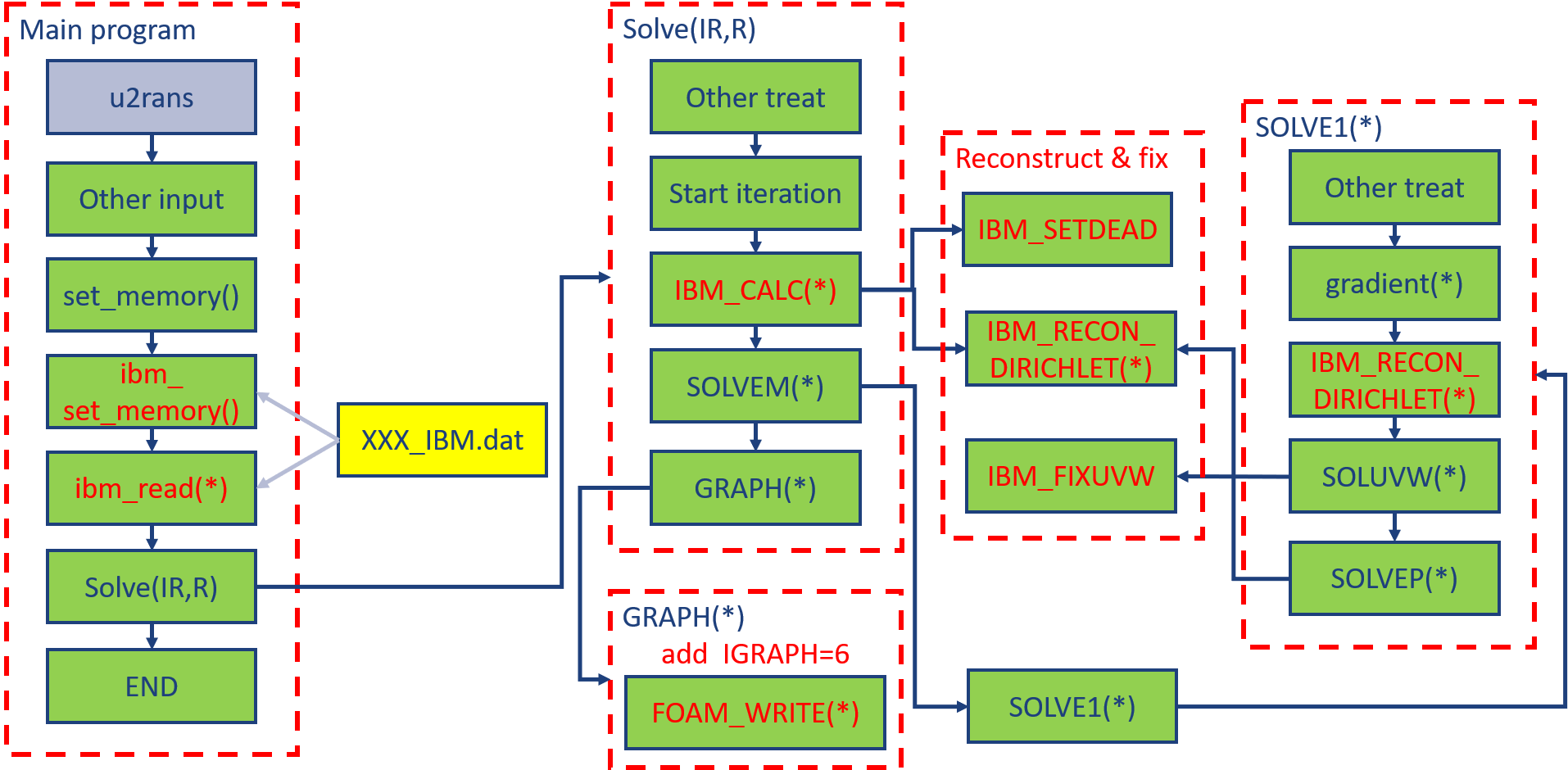


Figure 3.9: IB implementation in solver of U2RANS code.

# Chapter 4 Applications

The above IB method is implemented into U2RANS model. In this section, a number of turbulent flow cases are selected to verify the improved IB method. In particular, model accuracy is examined and discussed.

**4.1 Turbulent flow over flat plate**

The flow over a flat plate was used to investigate a 2D boundary layer without pressure gradient. The case is used to verify the IB method implementation with the proposed wall function. The length L of the plate is 2 m. The Reynolds number based on the plate length is and incoming flow velocity is = 2.5 m/s. The upper boundary is 0.1L away from the flat plate. The Cartesian mesh is refined near the leading edge and the plate. The mesh arrangement is shown in Fig. 4.1. A short slip surface (0.1L) is added in front of the leading edge using the symmetric boundary condition. The flat plate is modeled as an immersed boundary and the wall boundary condition is applied on it. We tested 4 different mesh size by using 4 different cell expansion ratios, , in the y-direction to change the mesh size of the first cell touching the wall.  The cell expansion ratio, , is that of the size of the end cell  to the size of the start cell  along the edge direction (). Different mesh sizes change the value of and . Two wall models were used in this case. One the original wall model only used the log-law to estimate the velocity and turbulences variables. The other one is the modified wall model developed in the previous chapter.

For both original and modified wall models, the computed local friction coefficient () along the plate compares well with the experimental data from in [Wieghardt and Tillmann](#_bookmark52) ([1951](#_bookmark52)). (Fig. 4.2(a) and (b)); the velocity profiles at 0.9L are in good agreement with the log-law (Fig. 4.2(c) and (d)). In addition, as decreases, the simulation results converge to the experimental data. The results show that the proposed IB algorithm with original or modified wall model is insensitive to the image point distance in the log-law layer.

|  |
| --- |
|  |

Figure 4.1: Mesh for turbulence flow over a flat plate. N is the number of cells and R is the cell expansion ratio in each direction.

|  |  |
| --- | --- |
| (a) | (b) |
| (c) | (d) |

Figure 4.2: Flat plate flow: simulated local friction coefficient (and velocity profile at 0.9L of the plate using original and modified wall models. The aligned mesh is used. (a) Local friction coefficient (original); (b) Local friction coefficient (modified); (c) Velocity profile at 0.9L (original); (d) Velocity profile at 0.9L (modified).

To further investigate the stability of the algorithm and its dependence on wall distance, the mesh above the flat plate is rotated as shown in Fig. 4.3. The top line of the grid has the same height to maintain the same water depth throughout the length. The bottom of the mesh is rotated and the downstream end of the bottom is moved down by 0.0025L. The red line represents the position of the flat plate. With this configuration, the mesh lines are not aligned with the flat plate. As the grid rotates (stretches), the wall distance is not uniformly distributed along the plate. Other detail of the mesh arrangement is shown in Fig. 4.3. is the number of mesh refinement in y-direction used in the region between the bottom line and the line 0.1L away from the bottom. We changed the number of () to show the mesh independence of this method. The cell expansion ratio in the y-direction in the refinement zone is 1 to maintain the minimum dimension constant for each IB cell such that the image point distance is the same; but the wall distance distribution is nonuniform over the plate. Fig. 4.4(a) shows the numerical results of local friction coefficient using the original wall model. The predicted results are comparable with the experiment. However, there are some small, semi-periodic oscillations due to the change of wall distance along the plate, especially when the value of decreases and the IB cell center is in the laminar sublayer. To suppress the oscillation from the nonuniform wall distance, the modified wall model is applied and the simulated results are plotted in Fig. 4.4(b). The modified wall model can predict the local friction coefficient more accurately and the oscillation is greatly reduced. As the mesh is refined, the local friction coefficient converges to the experimental data. In addition, velocity profiles in Fig. 4.4(c) and (d) at 0.9L agree well with the log-law. Both the original and modified wall models give a good prediction of velocity.

|  |
| --- |
|  |

Figure 4.3: Sketch of the rotated (stretched) mesh for turbulence flow over a flat plate.

|  |  |
| --- | --- |
| (a) | (b) |
| (c) | (d) |

Figure 4.4: Flat plate flow: simulated local friction coefficient (and velocity profile at 0.9L of the plate using original and modified wall models. The rotated mesh is used. (a) Local friction coefficient (original); (b) Local friction coefficient (modified); (c) Velocity profile at 0.9L (original); (d) Velocity profile at 0.9L (modified).

Table 4.1 provides the computational cost in one iteration of the IB method with modified wall model. The calculation of modified wall model includes the definition of IB cells, identification of image stencil for reconstruction, and implementation of wall function. All simulations were performed on a Dell Precision Tower 5810 with an Intel Xeon CPU ES-1620. The computational cost of the modified wall model is about 60% of the total CPU time in one iteration for each case. However, considering the boundary is immobile in this work, the IB cells and image stencil are only calculated in the first iteration and remain unchanged in the following iterations. The percentage of the computational cost of the modified wall model greatly deceases in the whole simulation time. In addition, the modified wall model avoids the computational cost required to fully resolve the near-wall flow in the laminar layer.

Table 4.1: Modified wall model performances on a single-core PC

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Grid | CPU time in modified wall model (s) | | | Total CPU time (s) | Percent (%) |
| Definition of IB cells | Image stencil | Boundary condition |
|  | 20.68 | 1.281 | 0.0004 | 31.963 | 68.7 |
|  | 23.61 | 1.678 | 0.0003 | 41.943 | 60.3 |
|  | 16.43 | 2.155 | 0.0004 | 52.361 | 54.6 |

**4.2 Turbulent flow around a cylinder over scoured beds**

The proposed IB algorithm is verified next for its capability to simulate a case with an instream structure - a turbulent flow around an instream cylinder over scoured beds. The modified wall model is used. The simulated results are compared with the flume experiment by Jensen et al. (1990). Fig. 4.5 shows the three bed profiles representing three scour phases observed in the experiment by Mao (1987). The cylinder has a diameter of 3 cm and is placed above three scoured bed profiles. The mean inlet flow velocity is 0.2 m/s. The computational domain is 1.1 m in the streamwise direction (*x*-direction). The flow depth at uneroded bed at the exit is maintained at 0.245 m (*y*-direction). Despite 2D flow in nature, the modeling is carried out in a 3D model domain with the dimension of 0.03 m along the cylinder (*z*-direction).

|  |
| --- |
|  |

Figure 4.5: Three cases are simulated corresponding to three bed profiles; the top, middle and bottom profiles correspond to profile 1, 3 and 5, respectively, of the experiments in Jensen et al. (1990). The simulated pressure contours are also displayed.

Using profile 3 as an example, the Cartesian background mesh has 0.1 cm resolution near the cylinder and scour bed, which is the finest resolution similarly used by Smith and Foster (2005). The background mesh has a total of 355,515 3D cells (71,103 2D cells in the *xy* plane and 5 cells along *z*). The cylinder boundary and bed profiles are treated as the immersed boundaries. The fluid and IB cells near the immersed boundaries are shown in Fig. 4.6. The meshes for the two other profiles are similar.

|  |
| --- |
|  |

Figure 4.6: Close-up side views of the fluid mesh cells near the cylinder and bed profile 3.

Fig. 4.7 shows the comparisons between simulation results and measured data from experiments. It is seen that the computed streamwise (*x*) velocity profile approaching the cylinder is near-logarithmic although a constant velocity boundary condition is applied at the inlet. The IB method simulated velocity profiles agree well with the experimental data for all three profiles and at eight streamwise stations. The results show that the recovery from the cylinder is slow - even at the last measured location about eight cylinder diameters downstream (*x* = 24 cm) the wake effect of the cylinder is still significant.

|  |
| --- |
| **(a)** |
| **(b)** |
| **(c)** |

Figure 4.7: Comparisons of the predicted streamwise velocity () with the measured data along eight streamwise stations. Scaling of is such that one unit of x is 10 cm/s. (**a**) Profile 1; (**b**) Profile 2; (**c**) Profile 3.

**4.3 Turbulent flow over 3D dunes**

3D dunes were used to verify the model performance with the turbulence model in a 3D form, especially, the prediction of wall shear stress. The modified wall model is used in this case for a more accurate and smooth wall shear prediction. The simulation results are compared with the experiments of Maddux et al. (2003a). In the experiment, fourteen fixed 3D dunes were placed on the bottom of the flume sequentially and experimental data were measured on the eleventh and twelfth numbered dunes. Only 6 dunes are simulated to reduce the computational cost and the data were collected from the last two dunes as shown in Fig. 4.8(a). The length of the simulation domain is 5.0 m in the x-direction and the width of the domain is 0.9 m in the y-direction. The bed elevation contour of the 3D dunes is shown in Fig. 4.8(b). The incoming velocity is 0.261 m/s in the *x*-direction and the water depth is 0.561 m in z-direction.

|  |
| --- |
| **(a)** |
| **(b)** |

Figure 4.8: Simulation domain and bed boundary. (**a**) Sketch of the computational domain; (**b**) Bed elevation of the 3D dunes in meters.

The Cartesian background mesh is shown in Fig. 4.9. The numbers of cells are 450, 90, and 70 in the *x*, *y* and *z*-direction, respectively. The mesh is refined near the dunes, especially where the bed elevation changes rapidly.

|  |
| --- |
|  |

Figure 4.9: Mesh for flow over 3D dunes.

Fig. 4.10 is a comparison of results from the IB method and the experiment. The streamwise velocities at different locations are well predicted. The only noticeable deviation is at the downstream of the measured two dunes at *y* = 0 m. In this slice, the bed elevation has the largest slope such that a long distance from the inlet is required for the flow to be fully developed. In the experiment, the measured dunes are the eleventh and twelfth. However, the measured dunes in the simulation are at the fifth and sixth due to the limitation of computing capacity. Thus, the flow condition is slightly different.

Comparison of bed shear stress is shown in Fig. 4.11. The simulation results show that the new IB algorithm can provide a smooth wall shear stress distribution. The normalized wall shear stress of the measured data (Fig. 4.11(a)) is estimated using the velocity 5 mm above the bed (Maddux et al., 2003a):

|  |  |
| --- | --- |
| , |  |

where is the velocity at 5 mm above the bed, the distance is 5 mm, and mm is the bed roughness.

The wall shear stress from the IB simulation is shown in Fig. 4.11(b). Even with the distribution of IB cells and image points changing arbitrarily with the bed elevation, the predicted shear stress is smooth. A comparison with the experimental data in Fig. 4.11(a) shows the existence of mismatch between the two. This is mainly because the accuracy of the simulation is limited by the computing capacity such that the velocity prediction at the places with large slope deviates from the experiment. However, the general characteristics of the wall shear stress are captured by the numerical model. The wall shear stress is the highest at the crests of dunes and relatively small elsewhere. The smooth distribution of wall shear stress is very important for modeling the sediment transport and scours, which is currently under development.

|  |
| --- |
| **(a)** |
| **(b)** |

Figure 4.10: The comparison of streamwise velocity from simulation and experiment. (**a**) The streamwise velocity at *y* = 0 m; (**b**) The streamwise velocity at *y* = 0.225 m.

|  |
| --- |
| **(a)** |
| **(b)** |

Figure 4.11: (**a**) Normalized wall shear stress from Maddux et al. (2003b), m2/s; (**b**) Normalized wall shear stress from IB method.

**4.4 Backward facing step flows**

Backward facing step flow is another classic validation case for turbulence model. This case adopts measured data from the experimental work of [Pitz and Daily](#_bookmark50) ([1981](#_bookmark50)). As shown in Fig. [4.12](#_bookmark36), this simulation case consists of a short inlet, a backward facing step, and a converging nozzle at outlet. The average inlet flow velocity is about 13.3 m/s.

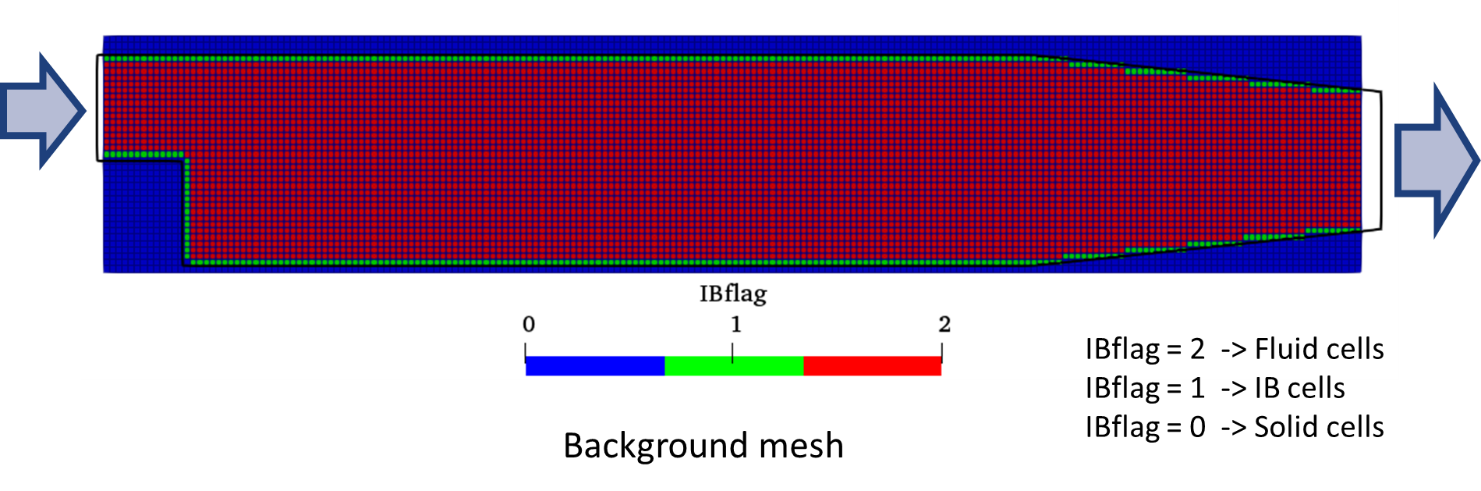


Figure 4.12: Background mesh and IB cell classification for the backward facing step case.

The simulated results are plotted against measure data and body-fitted results in Fig. 4.13. The comparison is reasonable.

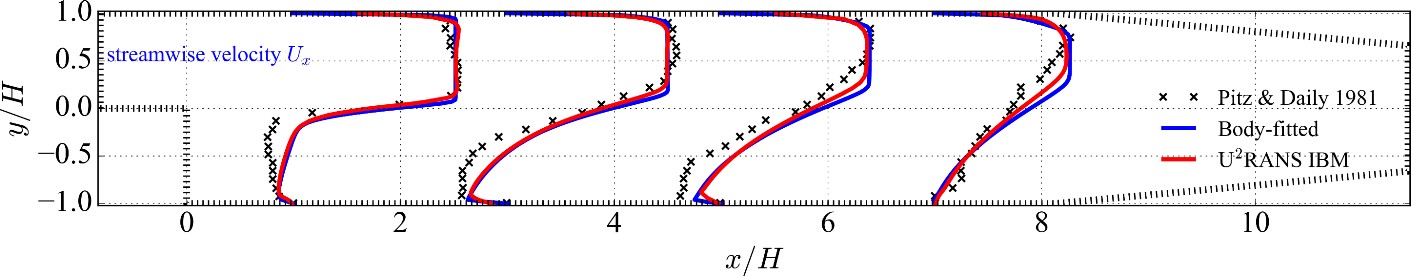


Figure 4.13: Vertical distribution of streamwise velocity at different locations.

The wall shear stresses simulated by U2RANS and body-fitted method are plotted in Fig. 4.14, in which their distance to the wall is proportional to the magnitude of wall shear. The sign of the wall shear (positive in the *x*-direction) is represented by whether the scatters are located inside (positive) or outside (negative) of the domain. The results of these two methods are very close and smooth. However, at the inlet and outlet of the domain, the result of U2RANS is smaller than body-fitted method because of boundary effect.

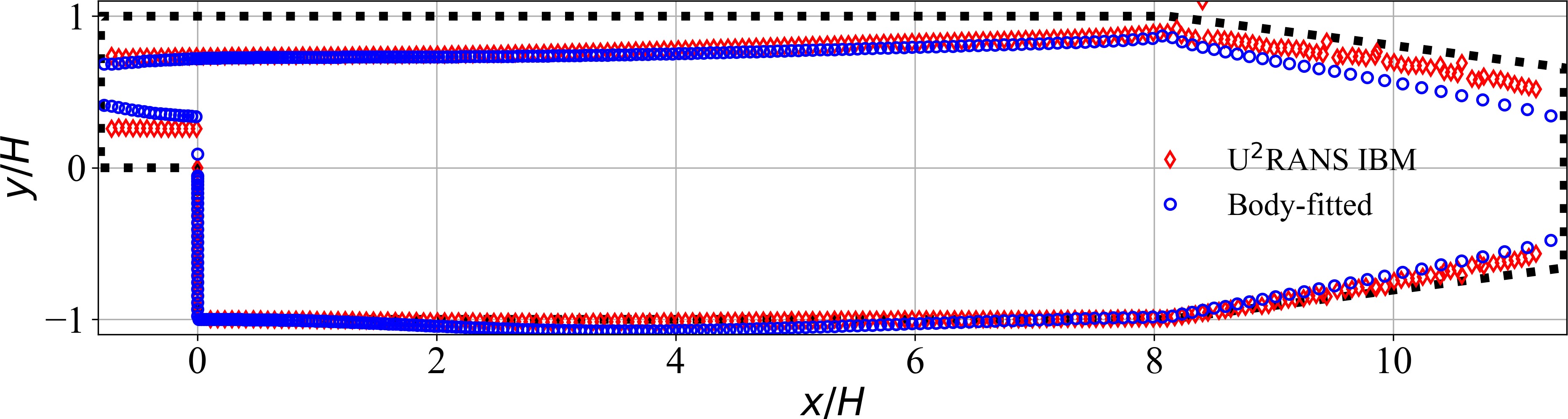


Figure 4.14: Wall shear stress distribution along the lower and upper walls. Wall shear is plotted using scatters in which their distance to the wall is proportional to the magnitude.

To investigate the effect of wall distance and the stability of the code, the immersed boundary is also shifted vertically for each time in this case (= 0.0015 m is the grid size in vertical direction). Fig. [4.15](#_bookmark39) shows the results of velocity distribution when shifting the immersed boundary for distance. These results are close to each other, which means the results of U2RANS are independent of the wall distance in a certain range.

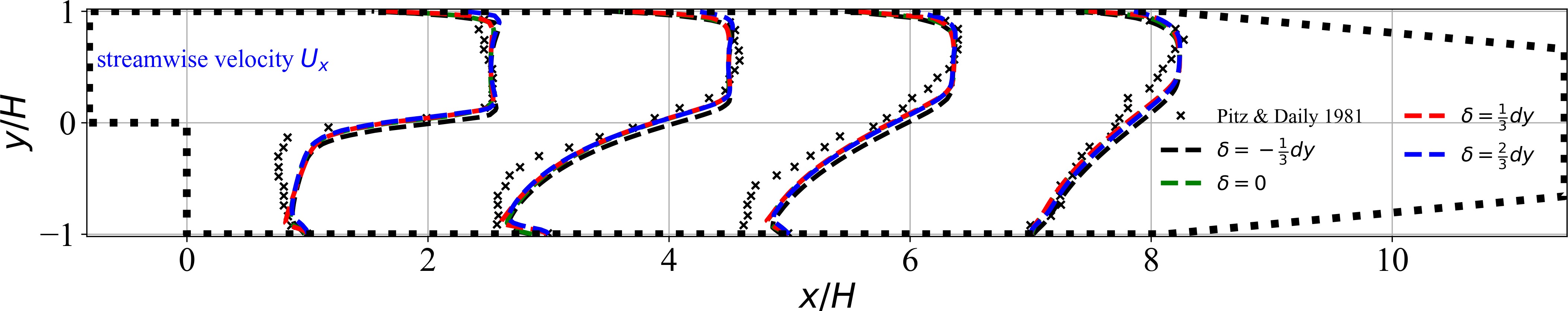


Figure 4.15: Vertical distribution of streamwise velocity when shifting the immersed boundary for

distance.

**4.5 Flows around the Nittany lion body**

This is a fun case to simulate the 3D flow field around the Nittany lion, Penn State University’s mascot. Fig. [4.16](#_bookmark41) shows the IB cells and deal cells surrounding the object and the original STL surface. The object is placed in a rectangular box domain with one inlet on the left and one outlet on the right. The inlet velocity is 1 m/s. Fig. [4.17](#_bookmark42) shows the flow field on the center slice of the domain. The velocity distribution looks reasonable. In Fig. [4.18](#_bookmark43), the flow structure around the object is shown with the isosurface of the calculated *λ*2 criterion. Also shown is the calculated forces in three directions on the object. These force components reach equilibrium after about 20 s.

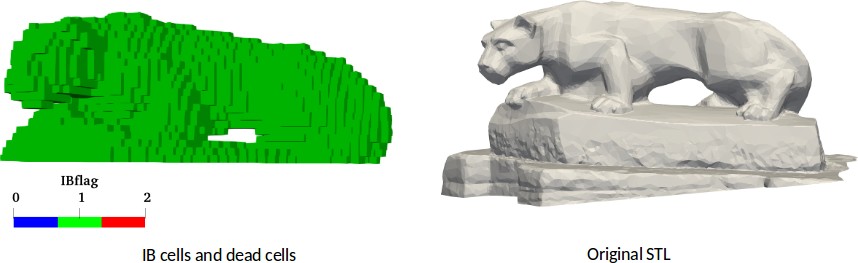


Figure 4.16: IB classification and the original STL of Nittany lion.

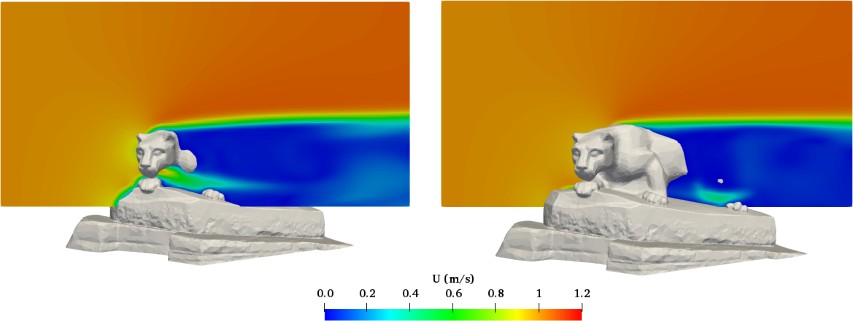


Figure 4.17: Slices of flow field around Nittany lion.

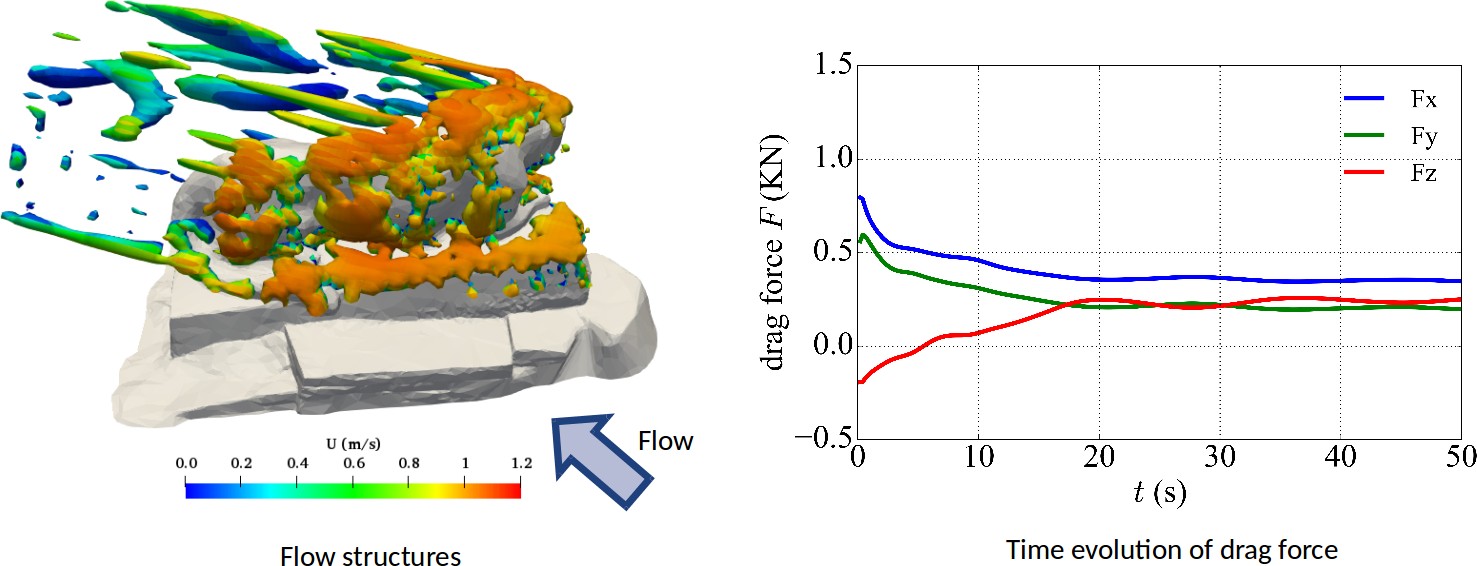


Figure 4.18: Flow structures induced by Nittany lion and time evolution of drag force acting on it.

**4.6 Flows around complex trees**

One important application of the IBM enabled U2RANS code is for river hydraulics. In many river engineering projects, structures, such as large woody debris, are installed. The hydrodynamics around these complex structures is of great interest. However, it is extremely difficult to model the effect the complex geometries using traditional body-fitted mesh methodology. Thus, this case simulates the flow around a mock large wood structure. The scan of the structure was provided by USBR (as seen in Fig. [4.19](#_bookmark45)). As in the previous case, the structure is placed in a rectangular box domain with one inlet and one outlet. The inlet velocity is 1 m/s. Fig. [4.20](#_bookmark46) shows the flow structure behind the elements of the wood structure colored by velocity magnitude. Note this is a coarse mesh simulation. Thus, not all details of the flow structure can be resolved. In the future, more refined mesh simulation can be carried out and the simulated results can even be compared with flume experiments. Fig. [4.21](#_bookmark47)shows the pressure distribution on the STL surface representing the tree. The pressure distribution on the STL surface was mapped from the IB cells using the algorithm described in previous chapter. In this figure, the force components in three directions are also plotted. The forces reached equilibrium after around 30 s.

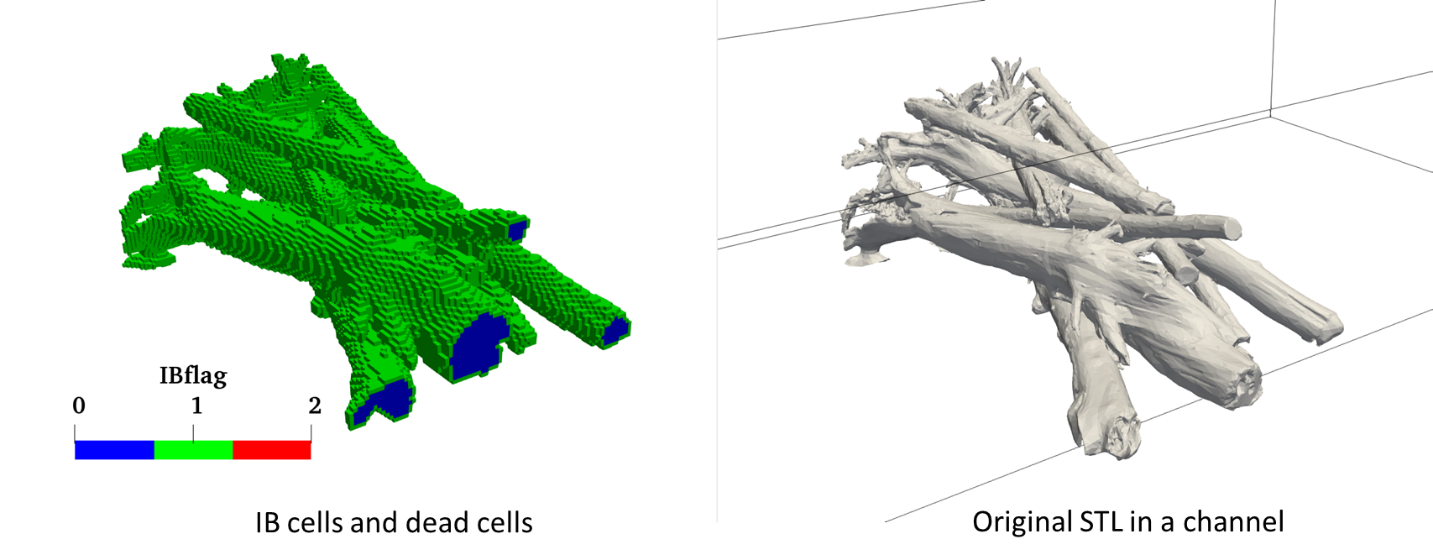


Figure 4.19: IB classification and original STL for complex trees in an open channel.

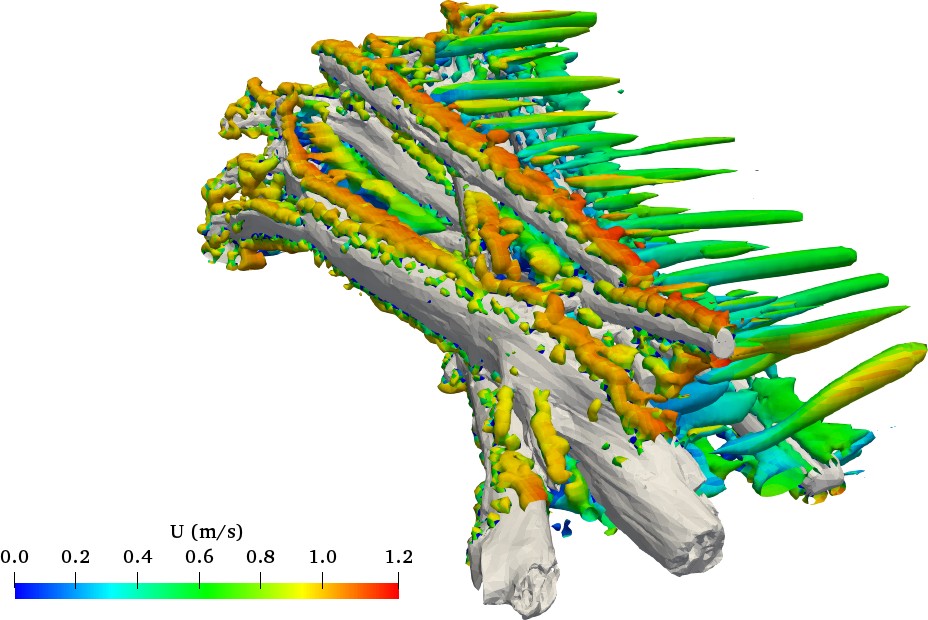


Figure 4.10: Flow structures induced by the surface of complex trees, colored by velocity magnitude.

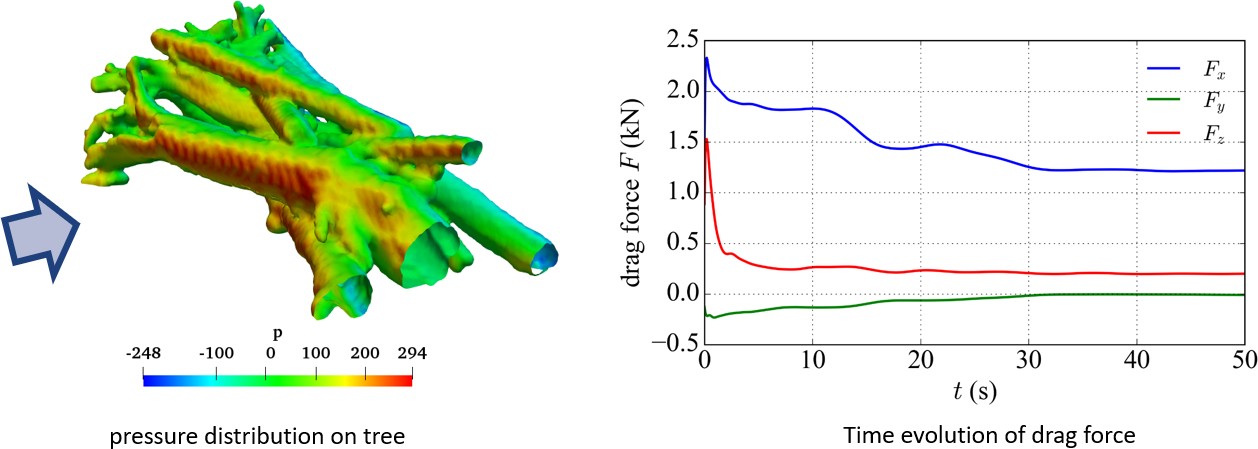


Figure 4.21: Pressure distribution on tree and time evolution of drag force.

**4.7 Cases availability**

All the validation cases can be found in the folder /IBTestCases/, namely

test\_turb\_plate

test\_turb\_backward

test\_turb\_trees

flowOverPlate

dune

For each case, for original U2RANS, it only requires .msh and \_SIF.dat. In the new version of U2RANS with IB method, two new folders are added in the case directory, i.e., foamFile and STLfiles. All the computational results are stored in foamFile, while all the STL input files are stored in STLfiles.

For better visualization, the new version of U2RANS writes out the mesh and the results based on the format of OpenFOAM® . Thus, the results can be viewed directly in ParaView.

Sometime the cases need to be cleaned. On Windows, the results can be cleaned up by executing reset.bat in foamFile, while on Linux, the results can be cleaned up by executing reset in foamFile.

# Chapter 5 Best practice guidance

Based on the experience of the preliminary study, the following best practice guidance can be given. This list of guidance needs to be expanded when more simulation cases are performed in future.

The normal direction of STL surface triangles needs to point to the fluid region. If the IB cells are not at their right position, the normal direction of STL surface should be flipped. To switch this, comment or uncomment line 431 in findib.f90 in u2rans\_pre.

If the IB cells are not cut by the STL, or u2rans\_pre fails, the code in findib.f90 about octree should be examined first. Many times, the octree data structure does not return correct results.

•

The code performs better in the simulation of fully-developed flow. It is important to note that this is not the limitation of the immersed boundary method. Instead, it is the limitation of many RANS models, such as the - model used in this work

# Bibliography

Capizzano, F. (2011). Turbulent wall model for immersed boundary methods. *AIAA journal 49* (11), 2367–2381.

Lai, Y. G., L. J. Weber, and V. Patel (2003). Nonhydrostatic three-dimensional method for hy- draulic flow simulation. ii: Validation and application. *Journal of Hydraulic Engineering 129* (3), 206–214.

Greenshields, C. J. "The OpenFOAM foundation user guide 7.0." The OpenFOAM Foundation Ltd: London, United Kingdom, 10th July (2019).

Jensen, B. L., B. M. Sumer, H. R. Jensen, and J. Fredsoe (1990) "Flow around and forces on a pipeline near a scoured bed in steady current." Journal of Offshore Mechanics and Arctic Engineering 112(3): 206-213.

Maddux, T. B., J. M. Nelson, and S. R. McLean. "Turbulent flow over three‐dimensional dunes: 1. Free surface and flow response." *Journal of Geophysical Research: Earth Surface* 108, no. F1 (2003a).

Maddux, T. B., S. R. McLean, and J. M. Nelson. "Turbulent flow over three‐dimensional dunes: 2. Fluid and bed stresses." *Journal of Geophysical Research: Earth Surface* 108, no. F1 (2003b).

Mao, Ye (1987). "The interaction between a pipeline and an erodible bed." *Series Paper Technical University of Denmark*, No. 39.

Pitz, R. and J. Daily (1981). Experimental study of combustion in a turbulent free shear layer formed at a rearward facing step. In *19th Aerospace Sciences Meeting*, pp. 106.

Roman, F., E. Napoli, B. Milici, and V. Armenio (2009). An improved immersed boundary method for curvilinear grids. *Computer & Fluids 38* (8), 1510–1527.

Smith, Heather D., and Diane L. Foster. "Modeling of flow around a cylinder over a scoured bed." *Journal of waterway, port, coastal, and ocean engineering* 131, no. 1 (2005): 14-24.

Tamaki, Y., M. Harada, and T. Imamura (2017). Near-wall modification of spalart–allmaras turbulence model for immersed boundary method. *AIAA Journal 55* (9), 3027–3039.

Wieghardt, K. and W. Tillmann (1951). On the turbulent friction layer for rising pressure. National Advisory Committee for Aeronautics, Technical Memorandum 1314.