UNIVERSITÀ DI PISA

Facoltà di Ingegneria
Corso di Laurea Specialistica in Ingegneria Aerospaziale

Tesi di laurea

Utilizzo di codici Open Source per valutazioni CFD

Relatore:
Prof. Ing. Giovanni Lombardi

Candidato:
Roberta Guatelli

Anno Accademico 2008/2009

Sommario

Scopo della tesi è capire l'utilizzo di alcuni free software in campo aerodinamico; nello specifico, è stato testato il comportamento di OpenFOAM, un programma libero per CFD, prodotto e distribuito gratuitamente sotto licenza GPL dalla compagnia commerciale OpenCFD Ltd. Il caso fisico affrontato, per provare il programma, è stato il comportamento di una deriva di barca a vela che si muove con velocità ed inclinazione costante. Si sono ricavati i coefficienti di portanza e di resistenza per una condizione di riferimento (incidenza di quattro gradi) e, data la buona risposta del programma, si è ricavata la polare della geometria. Per rafforzare l'analisi, si sono confrontati i risultati con quelli ottenuti dal programma *Star-ccm*$_+$. Infine si sono fatti degli studi di sensibilità alla griglia e ai metodi matematici usati per schematizzare il problema. Al termine dello studio, si è potuto notare globalmente un buon comportamento del programma. Si conclude sottolineando le grandi potenzialità di OpenFOAM, valida alternativa ad altri software non liberi, che consente, per la sua natura, collaborazione tra gruppi di lavoro. Lo strumento, probabilmente di maggior interesse, è il calcolo in parallelo tra più processori, che, essendo gestito direttamente dal programma, è di facile utilizzo.

Indice

Capitolo 1

Introduzione

L'argomento di studio della presente tesi è l'utilizzo dei codici open source in campo aerodinamico; più precisamente, di software definiti FLOSS (Free and Libre Open Source Software).

Questa distinzione è importante poiché sottolinea le caratteristiche fondamentali della natura del progetto. Infatti, come si può dedurre dall'acronimo, *open* sta per aperto, cioè i codici vengono tutti resi noti e disponibili; *libre* vuol indicare invece la possibilità, da parte dell'utente, di modificare i pacchetti e l'obbligo di ridistribuirli nuovamente *open*. Infine il termine *free* gioca sul doppio significato che può assumere nella lingua inglese, ovvero di libertà e di gratis. Nella quasi totalità dei casi, infatti, i prodotti FLOSS sono disponibili senza pagamento, essendo concepiti come mezzo e non fine di lavoro e guadagno.

Nello specifico, si è voluto provare OpenFOAM [1], un *free-open source* CFD software, prodotto dalla compagnia commerciale OpenCFD Ltd, che lo distribuisce gratuitamente sotto licenza GPL (General Public License). Questo tipo di licenza è la più diffusa in campo di software libero e garantisce libertà di utilizzo, copia, modifica e distribuzione.
OpenFOAM sta per Open Field Operation and Manipulation, è un insieme di strumenti utili alla risoluzione delle equazioni di campo, quindi è utilizzabile non solo in fluidodinamica, ma anche in dinamica dei solidi, nell'elettromagnetismo o in finanza. Ha una struttura modulare, redatta in C++, gerarchica, completamente accessibile e, perciò, altamente espandibile.
Una delle caratteristiche più interessanti del programma è l'implementazione della fase di calcolo parallelo; infatti l'utente non deve occuparsi della gestione della divisione tra più processori, è il programma che fa da ponte tra l'algoritmo e la parallelizzazione.

Riassumendo, la scelta di testare OpenFOAM è scaturita da due considerazioni principali. La prima riguarda tutte le conseguenze legate al fatto che sia sotto licenza GPL, quindi aperto, modificabile e gratis; è un software in continua via di sviluppo e supportato da una comunità che, condividendo informazioni, non solo aiuta il singolo utente nella gestione, ma anche contribuisce ad espandere il prodotto in termini di

conoscenza e di professionalità. La seconda considerazione riguarda la vasta gamma di strumenti forniti dal programma in campo fluidodinamico e la possibilità di affrontare il problema in tutte le fasi principali, ovvero dalla generazione della griglia alla visualizzazione dei risultati, interfacciandosi con un altro programma, solo open-source, ParaView [2].

Per capire il comportamento di OpenFOAM, si è scelto di prendere come oggetto di studio un caso precedentemente risolto attraverso il programma *Star-ccm*$_+$, in modo tale da avere un metro di paragone valido.

Nello specifico, l'analisi è stata effettuata sul comportamento della deriva di una barca a vela che si muove con una velocità di 3 $\frac{m}{s}$. Un primo confronto è stato fatto per un'incidenza di 4° della deriva rispetto il flusso d'acqua; data la buona riuscita della prova, si è deciso di proseguire lo studio tracciando la polare e analizzandola alla luce della risposta ottenuta da *Star-ccm*$_+$. In contemporanea, si è condotto uno studio sulla sensibilità del programma rispetto ai parametri che regolano la sezione matematica dello svolgimento del caso e si sono comparate varie mesh, più o meno fitte, elaborate da un numero diverso di processori.

Punto centrale della tesi è quello di spiegare in dettaglio come utilizzare il programma e di trarre delle possibili conclusioni sul suo comportamento. Infatti, nella prima parte della relazione, si darà una visione del problema fisico e della struttura base di OpenFOAM; nella seconda parte si descriveranno i passi necessari per la corretta preparazione del caso, fornendo, dove necessario, spiegazioni specifiche sulla compilazione dei file e riportando i tratti di codice utilizzati. Nella terza ed ultima parte della tesi, si riporteranno l'iter da fare per ottenere la soluzione e i risultati; infine vi saranno le conclusioni sul confronto con *Star-ccm*$_+$, evidenziando da un lato le potenzialità del programma libero, dall'altro le particolarità che potrebbero riservare fonte di errore e impraticabilità.

Capitolo 2

Caso fisico

Come detto nell'introduzione, il caso aerodinamico di partenza, preso ad esempio, è lo studio del comportamento della deriva di una barca a vela che sta viaggiando con una velocità di 3 $\frac{m}{s}$ e la cui inclinazione rispetto all'acqua è di 4°.
Lo studio poi continua con la ricerca della polare, tracciata dall'interpolazione dei coefficienti di portanza e di resistenza calcolati per un intervallo che va da 0° a 16°, con passo 2°. Essendo i profili della geometria NACA0016, è sufficiente andare a ricavare i dati per valori di incidenza positivi.

Da un punto di vista prettamente matematico, si può descrivere il flusso come incomprimibile e stazionario, la cui velocità e angolo di incidenza sono quelle sopra definite.
Seguendo le indicazioni date per il caso studiato con *Star-ccm+*, per risolvere il problema fisico, si è fatto ricorso alle RANS, ovvero alle equazioni di Navier-Stokes mediate, usando il modello di turbolenza $k - \varepsilon$.

La deriva è stata considerata come un solido indipendente dal resto del corpo della barca, posta in un dominio, rappresentante l'acqua, delle seguenti dimensioni:

- 30 metri di lunghezza
- 16 metri di larghezza
- 8 metri di altezza

La posizione della deriva rispetto al campo è tale che:

- il bordo d'attacco alla radice del profilo sia a 10 metri dall'inizio del parallelepipedo
- la corda del profilo sia parallela all'asse delle ordinate ed equidistante dai due lati
- la radice della deriva sia coincidente con il piano superiore del volume

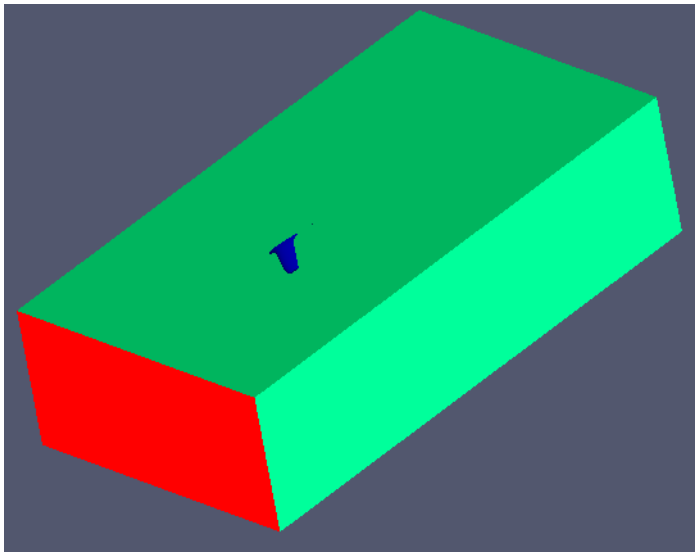

Figura 2.1: Posizione della deriva (blu) all'interno del dominio (verde e rosso).

In figura 2.1, si riporta un'immagine rappresentativa della posizione reciproca tra deriva e dominio.

Sempre con lo scopo di non alterare i risultati, si è recuperata, in formato *.stl, la geometria della deriva utilizzata per *Star-ccm*$_+$, generata con Catia (figura 2.2).

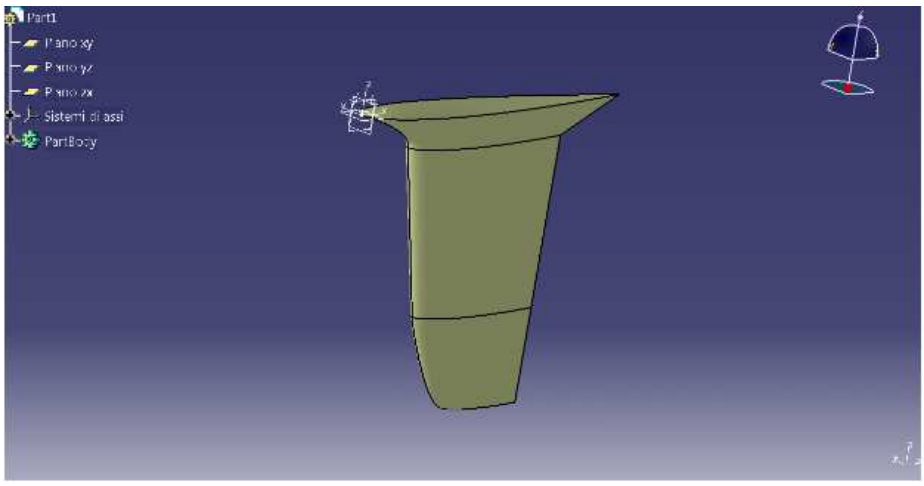

Figura 2.2: Immagine della deriva generata con il programma Catia.

Capitolo 3

Introduzione a OpenFOAM

In questo capitolo si dà una visione generale di OpenFOAM, partendo dalla spiegazione su come scaricarlo, proseguendo con la descrizione dell'installazione e del suo aspetto, per terminare con delucidazioni su come è strutturato un file tipo e su come far eseguire i comandi.

3.1 Scaricare OpenFOAM

Per ottenere OpenFOAM è sufficiente visitare il sito ufficiale, nella sezione *download*, all'indirizzo: http://www.openfoam.com/download/
Vi si possono trovare al momento due distribuzioni:

- una è la versione 1.6, che contiene sia i codici sorgenti che i binari precompilati per le più recenti distribuzioni Linux, per le due architetture a 32 bit e a 64 bit;
- una versione 1.6.x, aggiornata continuamente. Contiene solo i codici sorgenti con gli aggiornamenti e risoluzione di bachi.

Dato che il computer utilizzato per lo studio è un AMD Phenom II X4, è stata selezionata la distribuzione a 64 bit della versione 1.6 e scaricati gli archivi sotto la voce OpenFOAM e Third-party software. Dato che quelli indicati come opzionali non sono necessari per il funzionamento, si è preferito non prenderli, attenendosi ad una versione standard del programma.
I file devono poi essere messi all'interno di una cartella preventivamente creata con il nome *OpenFOAM*, dentro la */home* dell'utente, per rendere l'amministrazione più semplice.

A questo punto si decomprimono gli archivi usando il comando da terminale
tar -xzf nome_file,
iniziando con i file sorgenti (OpenFOAM-1.6.General.gtgz e ThirdParty.General.gtgz). Al termine dell'operazione, si dovrebbero avere, dentro la cartella OpenFOAM, altre due, chiamate *OpenFOAM-1.6* e *ThirdParty*.

All'interno della prima vi sono una serie di cartelle, alcune utili per il funzionamento del programma, altre rivolte all'utente come la *doc* e la *tutorials*, ed alcuni file, tra cui il più significativo è il README, dove è spiegato come installare il programma.

Nella seconda vi sono invece una serie di programmi con cui OpenFOAM si interfaccia.

3.2 Installazione

Prima di iniziare la descrizione della procedura di installazione, si apre una parentesi su quale tipo di sistema operativo utilizzare. OpenFOAM è sviluppato e testato su Linux, ma dovrebbe funzionare anche su sistemi POSIX (sistemi operativi UNIX). Per controllare il tipo presente sulla macchina, si può eseguire lo script *foamSystemCheck* presente nella cartella *bin* di *OpenFOAM/OpenFOAM-1.6*. Per chi non avesse dimestichezza con i comandi da terminale, vi deve scrivere:
cd percorso_cartella
e poi
./foamSystemCheck

Altra premessa importante è la versione del sistema operativo usato, dato che ci sono alcuni requisiti da soddisfare. Il programma di visualizzazione usato da OpenFOAM, ParaView, richiede che sia installato Qt; è ragionevole accertarsi della sua presenza e della versione prima di procedere oltre.

Si può controllare attraverso un qualsiasi gestore di pacchetti, ad esempio nel caso di Ubuntu si può usare *synaptic* oppure, da terminale, *aptitude*, utile anche per gli utilizzatori di Debian.

Da esperienza, la versione di sistema operativo, tra quelle provate, che non ha dato alcun tipo di problema di dipendenza è la Debian GNU/Linux Squeeze. Si consiglia, in generale, la giù aggiornata fornita e, in caso di problemi, di leggere attentamente il file README, presente all'interno della cartella OpenFOAM-1.6 o nella sezione di documentazione del sito [3], che offre svariate possibilità di risoluzione di errori durante l'installazione. Infatti, nel caso in cui l'utente abbia problemi nell'esecuzione di ParaView, è per certo un problema di conflitto di versione di Qt.

Nonostante la premessa che può apparire scoraggiante, OpenFOAM è semplice da installare.

Infatti, una volta decompressi gli archivi, si deve, nel caso venga utilizzata una bash o una ksh (nel dubbio scrivere in terminale: *echo $SHELL*), aggiungere alla fine del file .bashrc della /home la linea:
. $HOME/OpenFOAM/OpenFOAM-1.6/etc/bashrc
Si aggiornano le variabili di ambiente scrivendo nel terminale:
. $HOME/.bashrc
Nel caso di tcsh o csh, si deve fare la stessa operazione, ovvero scrivere nel file .cshrc:
source $HOME/OpenFOAM/OpenFOAM-1.6/etc/cshrc

e aggiornare con il comando:
source $HOME/.cshrc

A questo punto l'installazione è terminata e per testarla, si può eseguire lo script *foamInstallationTest*, dentro al percorso OpenFOAM/OpenFOAM-1.6/bin. Se non vengono dati errori, è probabile che sia perfettamente funzionante. Se ci sono output negativi, si deve invece controllare che sia stato installato correttamente.

Per avere la prova definitiva, si può far girare un caso fornito nei *tutorials*. Come prima cosa, si deve creare una cartella con il nome dell'utente, del tipo *utente-1.6*, al cui interno generarne una dal nome *run*, dove copiare la cartella *tutorials*. Passando al terminale, si deve entrare nel caso *cavity* attraverso il percorso:
OpenFOAM/*utente*-1.6/run/tutorials/incompressible/icoFoam/cavity
e digitare il comando *blockMesh*.
Una volta terminata l'operazione si dà il comando *paraFoam*.
Se non ci sono errori vuol dire che il programma è pronto all'utilizzo, se invece riporta output negativi, significa che molto probabilmente vi sono conflitti dovuti alla versione di Qt, come librerie mancanti o dipendenze non soddisfatte.

3.3 Aspetto

L'aspetto tipico di un caso in OpenFOAM è quello di una cartella, con un nome adatto al riconoscimento del problema, al cui interno vi sono almeno altre tre cartelle per definire le condizioni iniziali, al bordo e tutte le informazioni utili per preparare correttamente la simulazione.

La cartella del caso può essere posta ovunque, ma è consigliato metterla nella *run* di quella di lavoro dell'utente, poichè è una locazione già presente tra le variabili d'ambiente e quindi facilmente richiamabile.

Per avere un'idea pratica della strutturazione di un caso, si può vedere uno degli esempi forniti in *tutorials*.

Passando alla descrizione in dettaglio delle tre cartelle necessarie per preparare un caso, esse sono [4]:

- *constant*, in cui sono poste le informazioni riguardanti la griglia, all'interno della cartella *polyMesh*, e tutti i file che caratterizzano le proprietà fisiche del caso, come per esempio il modello di turbolenza in *turbulenceProperties* e in *RASProperties*.

- *system*, con i file che controllano la definizione dei parametri di gestione della soluzione. Infatti, al suo interno, devono essere presenti almeno tre file, che sono: *controlDict* per la definizione dei parametri temporali e le uscite dei risultati; *fvSchemes*, con i tipi di schemi per la discretizzazione dei termini costituenti le equazioni risolutive del caso; *fvSolution* con i solutori delle equazioni, le tolleranze e gli algoritmi di controllo.

- quella che contiene i file delle condizioni al contorno e all'interno del campo. Questa cartella ha come nome il tempo in secondi. Solitamente si indica con *0* o *0.000000e+00*, a seconda del formato specificato, la cartella di inizializzazione (deve essere sempre presente, anche nei casi fisici che non la richiedono). Durante l'esecuzione, ne verranno create altre in base al salvataggio temporale impostato.

3.4 Struttura tipo del file

Si passa ora alla spiegazione della struttura tipo di un file di OpenFOAM [5].

Il formato base segue alcuni principi del codice C++: le linee non hanno un particolare significato a parte quelle che iniziano con //, che sono righe di commento e quindi non lette durante l'esecuzione. Si può fare un commento tra più linee se racchiuse tra i delimitatori /* e */. Infine i documenti hanno una forma libera, senza dare un significato particolare a colonne o senza la necessità di indicare la continuazione tra più linee.

Per poter specificare i dati, viene utilizzata la definizione di un *dictionary*, una semplice e logica via per organizzare le informazioni attraverso una serie di parole chiavi, spesso a loro volta recanti dei dati in entrata. La natura gerarchica consente di specificare più parole chiave. Riassumendo, il formato tipo è quello del nome del *dictionary* e, tra parentesi graffe, l'elenco di parole chiave con le entrate, ovvero:

⟨Nome della *dictionary*⟩
{
parola-chiave entrata
}

Si può sfruttare una sintassi aggiuntiva per dare ampia flessibilità all'impostazione dei casi: la sostituzione macro e la direttiva. L'ultima è un comando che può essere contenuto dentro un file e inizia con il simbolo del cancelletto #, mentre le sostituzioni macro iniziano con il simbolo del dollaro $.

Il file classico di input o output, letto o scritto da OpenFOAM, inizia con il nome che lo definisce, quindi *FoamFile*, e poi, tra parentesi graffe, vi è un elenco che riporta informazioni base come la versione, il formato, la classe, la cartella in cui è posto e la sua natura. In tabella sottostante si elenca il caso generale.

Parola chiave	**Descrizione**	**Entrata**
version	formato del file	version 2.0
format	formato dei dati	ascii/binary
location	percorso del file	nome cartella
class	classe del file a seconda del tipo di informazioni contenente	Normalmente *dictionary* o per un campo, ad esempio di vettori, *volVectorField*
object	Nome del file	Esempio *controlDict*

Il termine *dictionary* è stato pensato per indicare una classe interna, definita appunto dizionario, a cui appartengono molti file di dati, per far sì che anche i profani di compilazione C++ possano facilmente definirne l'appartenenza.

Una volta definita la parte iniziale del file, si affronta la descrizione della forma del contenuto vero e proprio. Solitamente le informazioni vengono inserite tramite delle liste di dati, organizzate per comunione di contenuto con parentesi tonde. Ci sono alcuni modi per formattare le informazioni:
simple:
⟨Nome lista⟩
(
... elenco di dati ...
);
numbered, in cui si indica il numero di elementi all'interno della lista:
⟨Nome lista⟩
⟨n⟩
(
... elenco di dati ...
);
infine vi è anche la possibilità di identificare il tipo di contenuto della lista, modo detto *token identifier*; ad esempio nel caso di una grandezza scalare si può scrivere:
⟨Nome lista⟩
List ⟨*scalar*⟩
⟨n⟩
(
... elenco di dati ...
);

Avere dei formati diversi di stesura delle liste, rispetto a quello semplice, può essere utile nel caso in cui si abbiano elenchi lunghi con alta richiesta di memoria; infatti, definendo a priori la sua grandezza, si favorisce la velocità di lettura.

A questo punto è utile introdurre il modo in cui si debbano scrivere i dati a seconda della natura dell'informazione. Infatti, nel caso in cui sia uno scalare, è sufficiente riportare il suo valore; per un vettore si devono scrivere le componenti tra parentesi tonde; per un tensore, OpenFOAM fa sempre riferimento ad una matrice di rango

due e dimensione tre, per cui i nove elementi costituenti possono essere scritti effettivamente come una matrice o elencati su una riga, ma sempre compresi tra parentesi tonde, quindi:

```
(
1 0 0
0 1 0          o          ( 1 0 0 0 1 0 0 0 1 )
0 0 1
)
```

Le unità di misura devono anch'esse essere presenti all'interno del file. Per inserirle sarà sufficiente scrivere la parola chiave *dimensions*, seguita da una riga di sette numeri compresi tra parentesi quadre. I sette numeri non sono altro che l'esponente delle unità di misura, così ordinate:

1. la massa (SI: chilogrammi *kg*, USCS: libbra-massa *lbm*)
2. lunghezza (SI: metri *m*, USCS: piedi *ft*)
3. tempo (secondi *s*)
4. temperatura (SI: Kelvin *K*, USCS: Rankine *R*)
5. quantità (SI: chilogrammo-mole *kgmol*, USCS: libbra-mole *lbmol*)
6. corrente (Ampere *A*)
7. intensità luminosa (candela *cd*)

Nell'elenco precedente si indica con SI il sistema internazionale di unità di misura, mentre con USCS l'*United States Customary System*, ovvero le unità di misura imperiali.

OpenFOAM utilizza, per la definizione di alcune costanti universali, l'unità di misura internazionale, per cui, nel caso dovessero essere utilizzate le altre, si deve cambiarne il valore nella sottoparte *DimensionedConstant* del file *controlDict*, posto nella cartella di lavorlo di OpenFOAM */etc*.

Si prende come esempio la densità dell'acqua, che viene così definita:
rho rho [1 -3 0 0 0 0 0] 997,561;
La prima rho è la parola chiave con cui si identifica la grandezza, la seconda è il termine con cui viene salvata nella classe delle parole, tra le parentesi quadre vi sono gli esponenti delle unità di misura ed, infine, vi è il valore scalare.

La maggior parte delle informazioni su cui si lavora durante lo studio di un caso riguardano dei campi, normalmente elaborati nelle cartelle temporali. Il formato tipico presenta una suddivisione in tre parti: la definizione delle dimensioni, il valore all'interno del campo *internalField* e quello ai bordi *boundaryField*.
Per quanto riguarda il campo interno, vi sono due definizioni tra cui scegliere, che sono:

quella uniforme, per cui viene assegnato un solo valore a tutto il campo tramite l'entrata *internalField uniform* ⟨*valore*⟩

quella non-uniforme, per cui il valore dei singoli elementi del campo viene definito tramite un elenco usando: *internalField nonuniform* ⟨*elenco*⟩

Le condizioni al bordo si devono definire per ogni patch introdotta con i file della cartella *polyMesh*, mettendo il tipo e le sue informazioni con l'entrata *type*. OpenFOAM fornisce un vasto numero di opzioni da inserire per modellizzare in maniera ottimale il caso fisico [6].

A conclusione di questo paragrafo, si riporta come esempio esplicativo il file *U* utilizzato per l'inizializzazione del caso di studio. In tutta la tesi, i tratti di codice saranno identificabili perché compresi tra i simboli ††.

```
†
FoamFile
{
version 2.0;
format ascii;
class volVectorField;
object U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 1 -1 0 0 0 0];
internalField uniform (2.992692151 0.209269421 0);//incidenza di 4 gradi,
                                    //velocità di 3 metri al secondo
boundaryField
{
        inlet
                {
                type freestream;
                freestreamValue uniform (2.992692151 0.209269421 0);
                }
        outlet
                {
                type freestream;
                freestreamValue uniform (2.992692151 0.209269421 0);
                }
        ''deriva_solid''
                {
                type fixedValue;
                value uniform (0 0 0);
                }
       box
```

```
        {
        type freestream;
        freestreamValue uniform (2.992692151 0.209269421 0);
        }
}
// ************************************************************************* //
†
```

3.5 Esecuzione di un comando

I comandi vengono dati sempre da terminale in due modi:

- si può scrivere il comando dentro la cartella del caso
- si può far eseguire il comando da un'altra cartella con l'aggiunta di *-case* e per argomento il percorso

Capitolo 4

Generazione della griglia

In questo capitolo si affronta la prima fase di studio vera e propria, ovvero la generazione della griglia, o mesh, che consente la discretizzazione della geometria attraverso un reticolo, utile poi per il calcolo delle grandezze fisiche del fluido.
Come detto in precedenza, si deve creare un campo di dimensioni prestabilite in cui la deriva è posta in maniera ben precisa. OpenFOAM consente la creazione di una griglia tridimensionale, infittita e modellizzata attorno ad una geometria data, attraverso lo strumento *snappyHexMesh*.
L'idea di base è quella di adattare iterativamente ed automaticamente la griglia del dominio al solido fornito ed eventualmente far ritirare la griglia finale per inserire degli strati di celle più fitti sulla superficie.

In generale, si deve sottolineare che questo procedimento nasce solo per casi tridimensionali. Infatti, se si vuole studiare un problema bidimensionale, seppur il manuale del programma indichi di porre la parola chiave *empty* [7] nella definizione del piano parallelo alle direzioni di interesse, l'utilizzo di *snappyHexMesh* viene limitato da errori di tipo computazionale. Alcuni utenti consigliano allora di produrre un caso tridimensionale, attraverso l'estrusione della geometria 2D, e di inserire invece come condizione al bordo il termine *symmetryPlane* per garantire un risultato finale non alterato rispetto a quello bidimensionale.

A seguire, si riportano i passi necessari per poter utilizzare la *snappyHexMesh*; seppur le linee guida siano le medesime, la descrizione dettagliata dei file da compilare è centrata sulla configurazione che verrà poi utilizzata per la stima della polare, eventuali cambiamenti per capire la sensibilità del programma alla griglia verranno evidenziati contestualmente alla presentazione dei risultati.

4.1 I requisiti di *snappyHexMesh*

I requisiti necessari [8] sono:

- avere a disposizione la geometria da processare in formato STL (STereo Lithography). Tale formato parte dalla trasformazione della superficie del solido in

triangoli che vengono elencati nel file attraverso le coordinate xyz dei vertici e il vettore normale alla superficie del triangolo. Il file deve essere posto all'interno di una sottocartella detta *triSurface*, posta a sua volta in *constant*;

- generare il dominio, dove porre poi la geometria, con soli elementi di cella esaedrici;
- stilare un file apposito di nome *snappyHexMeshDict* da salvare all'interno della cartella *system*; esso regola il processo di modellizzazione della griglia sulla superficie di riferimento, ricavandola da quella del dominio.

Si fornisce a seguire una spiegazione più dettagliata dei punti appena elencati.

4.1.1 Salvataggio della geometria in formato *.stl

La geometria di partenza della deriva è stata generata attraverso il programma Catia, che consente il salvataggio nel formato *.stl.

A questo livello preliminare, si deve porre attenzione alla posizione del corpo rispetto agli assi cartesiani utilizzati nella fase di disegno; le informazioni spaziali sono infatti fondamentali per poter centrare correttamente il solido nel dominio.

4.1.2 Creazione del dominio

Il dominio può essere generato attraverso il comando *blockMesh* [9] che opera sul file *blockMeshDict*, preventivamente creato all'interno del percorso *constant/polyMesh*, nella cartella del caso. Genera come uscite, nella stessa *polyMesh*, cinque file descriventi la mesh che sono: *points, faces, owner, neighbour* e *boundary*. Si può facilmente capire che i documenti sono elenchi di numeri che identificano univocamente la struttura della griglia, ad esempio in *points* vi sono i vertici delle celle.

La filosofia con cui è stato concepito il comando *blockMesh* è quella di creare dei blocchi tridimensionali. Nel caso di studio se ne è scelto uno solo, di lati retti, le cui dimensioni sono quelle in precedenza elencate, ovvero 30 metri in lunghezza, 16 in larghezza e 8 in altezza. Ogni lato è stato suddiviso rispettivamente in 200, 100 e 50 parti, in modo tale che il singolo volume di cella avesse un *aspect ratio* tra i lati prossimo all'unità (caratteristica importante per il seguito).

Dato che il dominio così creato è la base di partenza per un successiva elaborazione, non è necessario, a questo livello, che il reticolo sia eccessivamente fitto. Così facendo, si evitano inutili aumenti di tempi di calcolo.

Per questa fase, si deve avere solo assicurare che le celle siano di forma esaedrica, di *aspect ratio* vicino all'unità, almeno in corrispondenza del corpo, e che lo intersechino in tutte le direzioni [10]. La *snappyHexMesh* potrebbe altrimenti non identificare la geometria su cui fare l'infittimento ed incorrere in ulteriori dispendi in termini di calcolo.

Tornando alla descrizione della compilazione del file *blockMeshDict*, al suo interno si devono elencare:

- la conversione in metri, per poter creare un dominio scalato opportunamente;
- i vertici del blocco, definendone le coordinate in modo tale da generare un sistema di riferimento coerente con la regola della mano destra. L'origine del sistema è coincidente con il primo vertice elencato (numerato automaticamente con 0), mentre gli assi sono definiti dalle linee che congiungono 0 a 1, 1 a 2 e 0 a 4 (fig. 4.1);

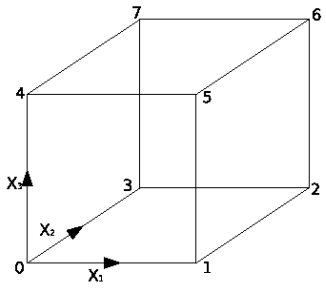

Figura 4.1: Definizione del singolo blocco.

- le caratteristiche del blocco, elencandone i vertici, il tipo (esaedrico *hex*), il numero di celle per lato e il loro rapporto di espansione (*simpleGrading*). Quest'ultima specifica è utile nel momento in cui si vuole infittire progressivamente lungo una direzione. Quando si vuole operare anche con *snappyHexMesh* è preferibile avere tutte le celle di dimensioni uguali per non avere un inuntile aumento di tempi di calcolo;
- le *patches* che compongono il dominio, attraverso la definizione di: tipo (se generica o con particolari caratteristiche geometriche); nome con cui verranno poi richiamate da altri file; vertici che le identificano. I vertici devono essere elencati in modo tale che, guardando da dentro il blocco, la faccia sia percorsa in senso orario.

Questa tesi non ha ovviamente lo scopo di sostituire la guida all'utilizzo, ma data l'importanza dell'argomento, si apre una breve parentesi riguardo alle *patches*, per richiamare l'attenzione sulle possibilità offerte da OpenFOAM [11] nella loro definizione. Questa non è infatti puramente geometrica, ma costituisce una parte importante nelle condizioni al bordo da un punto di vista prettamente fisico. Vi sono tre modi per caratterizzarle: uno base, descrittivo solo della caratteristica geometrica; uno detto primitivo, che consente una definizione del comportamento fisico di base, e uno derivato, che assegna una condizione fisica maggiormente complessa. Nel caso in

esame si è preferito usare la condizione base di *patch* per il dominio e di *wall* (parete solida) per la deriva (come si potrà notare in seguito).

Per maggiore chiarezza si riporta per intero il file *BlockMeshDict* usato per lo studio. Si fa notare che i vertici sono stati assegnati in modo tale che il bordo d'attacco alla radice della deriva, coincidente con l'orgine del sistema di riferimento, fosse a una distanza di 10 metri dal bordo del campo, e che la corda, coincidente con le ascisse, fosse equidistante dai margini del dominio.

```
    †
FoamFile
{
version 2.0;
format ascii;
class dictionary;
object blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 1;
vertices
(
        (-10 -8 -8) // Vertice numero: 0
        (20 -8 -8) // Vertice numero: 1
        (20 8 -8) // Vertice numero: 2
       (-10 8 -8) // Vertice numero: 3
        (-10 -8 0) // Vertice numero: 4
        (20 -8 0) // Vertice numero: 5
        (20 8 0) // Vertice numero: 6
        (-10 8 0) // Vertice numero: 7
);
blocks
(
        hex (0 1 2 3 4 5 6 7) (200 100 50) simpleGrading (1 1 1)
);
patches
(
        patch inlet // caratteristiche geometriche e nome
                (
                (0 4 7 3) //elenco dei vertici
                )
        patch outlet
                (
                (1 2 6 5)
```

```
        )
    patch box
        (
        (0 3 2 1)
        (4 5 6 7)
        (0 1 5 4)
        (7 6 2 3)
        )
);
// ************************************************************ / /
†
```

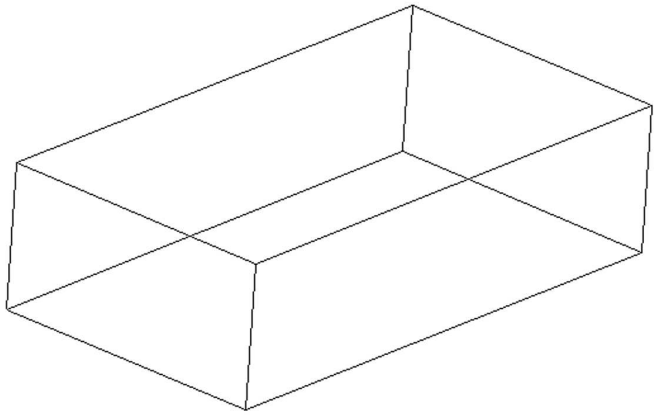

Figura 4.2: Immagine complessiva del blocco.

Una volta fatto compilare il file semplicemente digitando il termine *blockMesh* da terminale, si può controllare la correttezza del risultato scrivendo, sempre da terminale, il comando *paraFoam*.
Si genera così automaticamente un file di estensione adatta per essere processato dal programma ParaView, aperto in contemporanea. Quest'ultimo è un programma open-source e consente la visualizzazione e l'analisi dei dati.
Considerato che il processo legato a *snappyHexMesh* può impiegare anche alcune decine di minuti nei casi più elaborati, è buona norma controllare che il dominio sia corretto, seppur la semplicità del blocco. Inoltre è utile verificare la posizione nel campo della geometria da processare in seguito, aprendo il file *.stl direttamente da ParaView. Spesso infatti si può incorrere in un problema banale ma grave. Durante la conversione del file in formato .stl, può accadere che si "perda"l'informazione dell'unità di misura, per cui spesso una geometria con tutte misure in millimetri può essere trasformata in metri. È sufficiente allora dare il comando da terminale (all'interno della cartella che contiene il file):

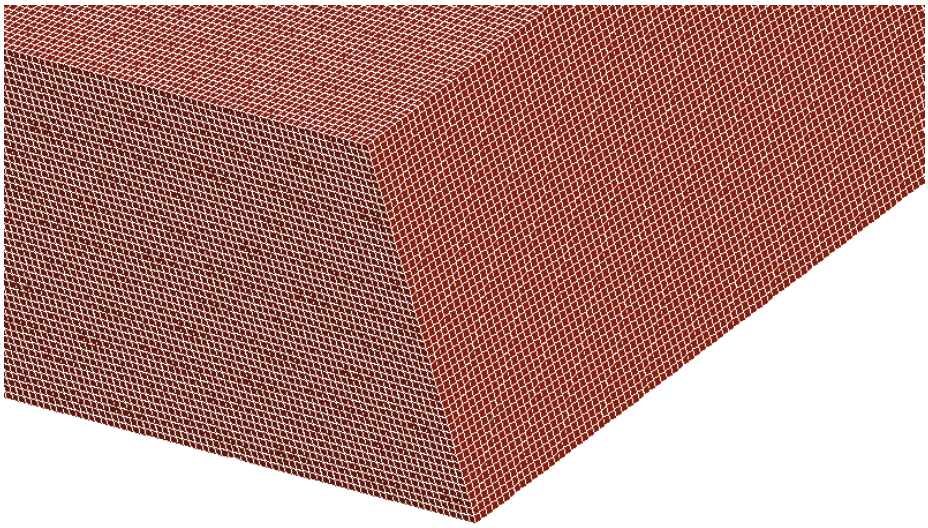

Figura 4.3: Particolare di uno spigolo della mesh del dominio, elaborata con lo strumento *blockMesh*.

surfaceConvert input.stl output.stl -clean -scale 0.001,
con input.stl si indica il file da cambiare, con output.stl quello nuovo e con 0.001 il fattore di scala [12].

Infine si può presentare un altro problema, nel caso in cui ParaView non riesca a visualizzare il risultato; se ciò accade è sufficiente dare il comando
export LC_ALL=C
prima di far eseguire *paraFoam*.

Il risultato di questa fase è mostrato nelle figure 4.2 e 4.3.

4.1.3 Compilazione del file *snappyHexMeshDict*

Una volta creato il dominio, si può procedere alla compilazione del file *snappyHexMeshDict*.

Come partenza, si sono presi a riferimento gli esempi forniti all'interno della cartella *tutorials* di OpenFOAM, cambiando opportunamente i dati in ingresso.

La struttura del file comprende delle sezioni principali che, in alcuni casi, devono essere a loro volta specificate con più entrate. Nella tabella sottostante si riportano i gruppi principali da menzionare nel documento [13].

Entrata	**Descrizione**
castellatedMesh	creare una mesh elaborata
snap	avere fase di rottura delle celle
doLayers	aggiungere degli strati di celle sulla superficie
geometry	specificare la geometria
castellatedMeshControls	entrate di controllo della *castellatedMesh*
snapControls	entrate di controllo della fase *snap*
addLayersControls	entrate di controllo per l'aggiunta di strati
meshQualityControls	entrate di controllo per la qualità della griglia
mergeTolerance	margine di tolleranza in rapporto alla griglia iniziale
debug	controllo del salvataggio dei dati e controllo sugli output da terminale

Per chiarezza, si preferisce descrivere le singole parti esposte nell'elenco precedente riportando, passo passo, il tratto di codice che le riguarda con all'interno anche i commenti esplicativi; quando necessario, vi sarà un'ulteriore spiegazione del loro contenuto nel testo.

Il file *snappyHexMeshDict* inizia col dichiarare quali comandi debbano essere eseguiti:

```
†
FoamFile
{
version 2.0;
format ascii;
class dictionary;
object snappyHexMeshDict;
}
// ************************************************************ //
//Quali passi si vogliono far girare?
castellatedMesh true;
snap true;
addLayers true;
†
```

È evidente che si debba scrivere *true* nel caso affermativo, mentre si deve porre *false* in caso negativo.

Il codice poi prosegue con:

```
†
//Geometria.

//Definizione di tutte le superfici. Appartengono alla classe searchableSurface.
geometry
```

```
{
        deriva.stl // Nome del file .stl della deriva
                {
                type triSurfaceMesh; // Griglia appartenente alla cartella triSurface
                name deriva; // Nome con cui si richiamerà nel resto del file
                }

            refinementBox // Zona in cui si vuole far infittire la mesh
                {
                type searchableBox; // Tipo di superficie di ricerca a forma di scatola
                min (-0.5 -0.5 -2.4); // Limiti inferiori
                max ( 3 0.5 0); // Limiti superiori
                }
};
†
```

All'interno della parte *geometry*, devono essere definite tutte le geometrie in corrispondenza delle quali si vuol fare infittire e modellare la griglia. Devono essere menzionate le superfici del solido preventivamente salvato nel formato *.stl ed eventuali regioni del dominio su cui si vuole raffinare la mesh, sfruttando parole chiave già fornite da OpenFOAM (ad esempio *searchableBox*).

A questo punto del file, devono essere inserite le specifiche alla base della creazione della griglia modellata. Si può schematizzare il procedimento in tre fasi principali:

1. suddivisione e rimozione delle celle attraverso le specifiche inserite nella sezione *castellatedMeshControls*,

2. adattamento della griglia alla superficie di riferimento (parte *snapControls*),

3. inserimento, sul bordo del corpo, di strati di celle in direzione normale alla superficie nella parte *addLayersControls*.

Suddivisione e rimozione delle celle: *castellatedMeshControls*

Le specifiche introdotte nella parte *castellatedMeshControls* servono ad identificare quali celle debbano essere infittite e come debbano essere gestite da un punto di vista computazionale. Per facilitare la comprensione, si riporta prima la parte di codice di interesse, poi le eventuali spiegazioni.

```
†
castellatedMeshControls
{
//Parametri di rifinitura
```

```
    //Numero di celle per processore. Se troppo basso, vengono fatte più iterazioni
//per avere una mesh simile
maxLocalCells 1000000;

    //Limite di celle complessive.
//Rifinitura termina appena raggiunto nella mesh
//ma prima che vengano rimosse. Alla fine possono in effetti essere di meno.
maxGlobalCells 2000000;

    //Per evitare che si impieghino troppe iterazioni per poche celle, si definisce il
//numero minimo per cui l'iterazione termina.
//Ne fa almeno una, a meno che non sia 0
minRefinementCells 10;

    //Numero di strati in memoria tra livelli diversi. 1 è una normale restrizione 2:1,
// più grande implica una rifinitura più lenta
nCellsBetweenLevels 3;

    //Esplicita caratterizzazione della zona da rifinire
features
        (
        // Lasciato commentato perché non utilizzato nel caso
              //{
              // file someLine.eMesh;
               // level 2;
               // }
        );

    // Rifinitura sulla superficie // Specificare due livelli per ogni superficie.
// Ogni cella che interseca queste superfici devono essere
//rifinite fino al livello minimo, le celle che vedono un angolo
// compreso tra le intersezioni maggiore di resolveFeatureAngle sono rifinite fino
// al livello massimo.
refinementSurfaces //Definizione delle superfici da rifinire
        {
              deriva // geometria della deriva
              {
              level (2 3); // Livello minimo e massimo di rifinitura
              }
        }
resolveFeatureAngle 30;
```

```
    //Rifinitura nella regione
//Si specifica il livello di rifinitura per cella in relazione alla superficie.
//Tre possibilità:

 //distance. levels definisce il livello di rifinitura per distanza.
      // Le distanze devono essere specificate in ordine discendente.


 //inside. levels ha una sola entrata.
      // Tutte le celle dentro la superficie sono rifinite a quel livello.
      //La superficie dovrebbe essere chiusa.


 //outside. Lo stesso di inside ma fuori la superficie.


refinementRegions //Regione da rifinire
      {
      refinementBox //Definizione della regione
            {
            mode inside;
            levels (1E15 4);
            }
      }

    // Mesh selection
// Definizione del vettore che identifica la parte di campo le cui celle devono essere
//mantenute. Deve essere sempre dentro una cella, mai su una faccia, anche dopo la
//rifinitura.
locationInMesh (-3 0 -4);
}
†
```

Come si può capire dal codice, la prima parte tratta la gestione delle celle durante il calcolo. Successivamente, vi è l'entrata *features*, che riporta il punto da dove fare partire il processo di elaborazione con il file con le specifiche e il livello di rifinitura. Nel caso di studio si è deciso di non sfruttarla.
Poi, tramite *refinementSurfaces* e *refinementRegions*, si identificano le zone per cui debba essere infittita la mesh. Mentre la prima riguarda le superfici *.stl, la seconda fa riferimento a zone del dominio preventivamente introdotte in *refinementBox*.

Inoltre, per quanto riguarda *refinementSurfaces*, il livello minimo di rifinitura è applicato su tutta la superficie, mentre quello massimo è usato per quelle celle che intersecano la superficie *.stl e che formano un angolo maggiore di *resolveFeatureAngle*.

Per *refinementRegions* si può invece scegliere tra tre modi di infittimento:

1. *inside*, ovvero dentro al volume
2. *outside*, fuori il volume
3. *distance*, si rifinisce a una certa distanza dalla superficie

In tutti e tre i casi si deve specificare prima la distanza e poi il livello; anche nel caso si voglia usare i primi due modi, si deve porre la distanza ma non verrà considerata durante il processo.

Infine si introduce il vettore *locationInMesh*, che ha lo scopo di definire quale parte del dominio deve essere mantenuto nella fase di rimozione delle celle (solitamente contenute dentro a superfici chiuse). Tale vettore deve essere posto nella zona del campo che ricopre almeno il 50% della parte da mantenere.

Adattamento della griglia: *snapControls*

Dopo la fase *castellatedMeshControls*, la superficie del solido risulta dentellata. Con le specifiche introdotte in *snapControls*, la mesh viene smussata in prossimità della superficie del corpo definito dal file *.stl.
Il procedimento è iterativo e consiste in:

- spostare i vertici della griglia sul volume *.stl;
- risolvere i problemi causati nella griglia da tale movimento;
- identificare i vertici che possono violare i parametri della qualità della mesh;
- ridurre lo spostamento di quest'ultimi dal valore iniziale, ripercorrendo quindi i passi precedenti.

Nella parte di codice sottostante, vi sono le entrate da compilare con relativa spiegazione.

```
†
snapControls
{
//Numero di iterazioni per muovere le patch prima di trovare
//corrispondenza alla superficie
nSmoothPatch 3;

    // Distanza relativa per punti che devono essere attratti dalla
// superficie di riferimento o da spigoli. Vera distanza è questo
// fattore moltiplicato la lunghezza massima dello spigolo
tolerance 4.0;
```

```
    // Numero di iterazioni per il rilassamento dello spostamento della griglia
nSolveIter 30;

    // Numero massimo di iterazioni per il rilassamento dello snapping.
// Dovrebbe finire appena prima sia raggiunta una mesh corretta.
nRelaxIter 5;
}
†
```

Inserimento di strati di celle: *addLayersControls*

La fase di inserimento di celle, controllata dalla sezione *addLayersControls*, è opzionale. Il procedimento consiste nel far ritirare la griglia già generata e nell'inserire degli strati di celle allineati al contorno della superficie, in modo tale da risolvere eventuali irregolarità introdotte nelle prime due fasi.
Vi sono cinque passi [14]:

1. la griglia è spostata di uno spessore specificato nella direzione normale alla superficie;
2. si risolvono eventuali problemi nella griglia dati da tale spostamento;
3. si controlla se è una mesh valida, altrimenti si riduce lo spessore di spostamento e si riparte dal punto 2; se nessuno spessore può essere soddisfaciente, non si inserisce alcuno strato;
4. se i criteri di validità sono soddisfatti, si inserisce lo strato di celle;
5. la griglia viene nuovamente controllata; se non supera il controllo lo strato viene rimosso e si riparte dal punto 2.

All'interno della sezione, si devono indicare tutte le parti su cui si vuole inserire uno strato e la quantità; è importante far riferimento alla mesh già esistente e non alla superficie della geometria.
Si riporta la parte di codice utilizzato per questa fase, per cui è stato previsto un solo strato per tutta la griglia che identifica la geometria della deriva.

```
†
addLayersControls
{
relativeSizes true;

    // Informazioni sulle parti di mesh su cui inserie gli strati
      layers
      {
```

```
            deriva_solid //Nome con cui viene salvata la mesh della deriva
            {
            nSurfaceLayers 1;//Numero di strati da porre
            }
        }

    //Fattore di espansione per lo strato della mesh
expansionRatio 1.0;

    // Spessore voluto dello strato di celle finale.
// Per più strati, è lo spessore dello strato più lontano dalla superficie
// Relativo alla dimensione della cella al di fuori dello strato
finalLayerThickness 0.3;

    //Spessore minimo della cella dello strato. Se non può essere maggiore
//non si aggiungono strati.
//Relativo alla dimensione della cella al di fuori dello strato
minThickness 0.1;

    //Se i punti non sono spostati, si fa nGrow strati
//di facce connesse a loro volta non inspessite.
// Aiuta la convergenza del processo di aggiunta dello strato con le caratteristiche
//volute
nGrow 1;

    //Setting avanzati

    //Quando la faccia non è inspessita //0 è superficie piatta, 90 quando due facce
formano un angolo retto
featureAngle 30;

    //Massimo numero di iterazioni del rilassamento da snapping.
//Dovrebbe finire appena prima sia raggiunta una mesh corretta.
nRelaxIter 3;

    //Numero di iterazioni per lo spostamento della normale alla superficie
nSmoothSurfaceNormals 1;

    //Numero di iterazioni per lo spostamento nella direzione del movimento
//della mesh
nSmoothNormals 3;
```

```
    //Spessore di spostamento dello strato sulla patch della superficie
nSmoothThickness 10;

    //Fermata della crescita dello strato sulle celle altamente deformate
maxFaceThicknessRatio 0.5;

    //Crescita dello strato ridotta dove il rapporto tra spessore e
//distanza media è grande
maxThicknessToMedialRatio 0.3;

    // Angolo usato per prendere l'asse medio dei punti
minMedianAxisAngle 130;

    // Creare regione in memoria per nuova interruzione della mesh
nBufferCellsNoExtrude 0;

    // Numero massimo complessivo di iterazioni per l'aggiunta di strati
nLayerIter 50;
}
†
```

Per considerare completa la compilazione del file, si devono inserire le voci mancanti, che sono: *meshQualityControls*, *mergeTolerance* e *debug*. Viene trascritta la parte finale del file, sempre con i commenti esplicativi di ogni entrata.

```
†
//Setting generici della qualità della mesh. Determina quando rifare dei passi
meshQualityControls
{

    //Massima non-ortogonalità consentita. Porre 180 per disabilitarlo.
maxNonOrtho 65;

    //Massima inclinazione consentita. Porre minore di 0 per disabilitarlo.
maxBoundarySkewness 20;
maxInternalSkewness 4;

    //Massima concavità consentita. Angolo in gradi sotto il quale la concavità
//è consentita. 0 è diritta, minore di 0 convessa. Porre 180 per disabilitarlo
maxConcave 80;

    //Rapporto tra la minima area proiettata e l'attuale. Porre -1
//per disabilitarlo.
```

```
minFlatness 0.5;

    //Minimo volume piramidale. Assoluto per cella piramidale.
//Porre un numero negativo molto elevato (es. -1E30) per disabilitarlo.
minVol 1e-13;

    //Area minima della faccia. Porre minore di 0 per disabilitarlo.
minArea -1;

    //Minima torsione della faccia. Porre minore di -1 per disabilitarlo.
// Prodotto tra la normale della faccia e la normale del centro della superficie
// triangolare
minTwist 0.02;

    //Determinante minimo della cella normalizzata, -1 = hex,
// ≤ 0 = cella ripiegata o appiattita illegalmente
minDeterminant 0.001;

    // minFaceWeight (0 → 0.5)
minFaceWeight 0.02;

    //minVolRatio (0 → 1)
minVolRatio 0.01;

    //deve essere maggiore di 0 per la compatibilità con Fluent
minTriangleTwist -1;

    // Caratteristiche avanzate

    // Numero di iterazioni per distribuzione dell'errore
nSmoothScale 4;

    //Quantità per scalare nuovamente allo spostamento nel punto d'errore
errorReduction 0.75;
}

    //Debug
// Output possibili:
// 0 : scrive solo la mesh finale
// 1 : scrive le griglie intermedie
// 2 : scrive volScalarField con cellLevel per il postprocessing
// 4 : scrive l'intersezione corrente come .obj files
```

debug 0;

// Merge tolerance. Frazione del contorno complessivo della mesh iniziale.
mergeTolerance 1E-6;
//*** //
†

L'iter dell'analisi nella sua completezza è articolato, quindi si è preferito rimandare la sua discussione al capitolo dei risultati; stesso approccio è stato previsto anche per riportare il prodotto della generazione della griglia.

4.2 Generazione della griglia in parallelo

Dato che può essere utile dividere il costo computazionale tra più processori, si introduce in questo paragrafo il procedimento di parallelizzazione tra più processori offerto da OpenFOAM, che sfrutta l'implementazione dei domini pubblici *openMPI* dei "standard message passing interface"MPI (interfaccia dei messaggi di passaggio). Il metodo usato è noto come decomposizione del dominio, per cui la geometria e i campi associati sono divisi in parti e trattati da processori diversi.
Lo scopo è quello di dividere il dominio con il minimo impegno ma in modo tale da garantire una buona soluzione come costi computazionali. Per poter gestire questa fase si deve compilare un file nominato *decomposeParDict*, posto nella cartella *system* del caso.

Si può scegliere tra cinque metodi di decomposizione ([1]):

- *simple*, decomposizione geometrica semplice del dominio;
- *hierarchical*, simile al precedente ma specificando l'ordine con cui si svolge la suddivisione;
- *scotch*, non richiede input geometrici, tenta di minimizzare il numero di limitazioni per processore. Attraverso l'entrata *processorWeights* si può suddividere il peso di calcolo tra diversi processori nel caso in cui abbiano diverse prestazioni. Con il termine *strategy* si può invece controllare la strategia di decomposizione;
- *metis*, simile al precedente, ma la libreria richiede una licenza a pagamento se usata per scopi di vendita;
- *manual*, si deve indicare come distribuire ciascuna cella tra i processori.

Una volta suddiviso il campo dando il comando da terminale *decomposePar*, si generano un numero di cartelle, in quella del caso, pari al numero di processori, nominate *processorN*, contenenti una cartella del tempo con la descrizione del campo

[1] Per maggiori informazioni si rimanda a [15]

decomposto e una *constant/polyMesh* con le informazioni sulla griglia.
A questo punto si sfrutta l'implementazione *openMPI*; se i processori sono su una stessa macchina non è richiesta alcuna specifica, mentre se sono distribuiti tra più computer, si deve compilare un file con gli hostname delle macchine. L'hostname deve essere adatto ad una completa risoluzione di dominio nel file */etc/hosts* della macchina su cui si lancia *openMPI*.
Tale lista deve contenere il nome del computer su cui gira *openMPI*. Se una macchina all'interno della rete ha più di un processore, il suo nome può essere seguito da *cpu=n* con *n* numero dei processori da utilizzare. Infine, se i dati sono distribuiti tra più dischi, devono essere specificati i percorsi nel file *decomposeParDict* usando le entrate *distributed yes* e *roots* con il numero di dischi(per maggiori informazioni si rimanda alla guida [15]).
Un applicazione si può far girare in parallelo scrivendo da terminale:
mpirun –hostfile <file con lista delle macchine con indirizzo della cartella>-np <numero di processori><applicazione Foam ><altri argomenti >-parallel
Per ricostruire i file si hanno due possibilità:

- ricostruire la mesh e i campi per ricreare il dominio completo e analizzarlo come se fosse uno solo. Si uniscono le cartelle temporali da ciascuna *processorN*, semplicemente eseguendo da terminale il comando *reconstructPar* (se distribuite su più dischi si devono prima copiare sulla macchina su cui si dà il comando).

- analizzare la singola parte individualmente usando *paraFoam*.

Per il caso di studio si è scelto il metodo *hierarchical*, con la divisione prevista tra due processori e con i dati salvati su un unico disco. Si trascrive di seguito il file *decomposeParDict* utilizzato. Un accortezza di cui si deve tener conto è che il prodotto tra il numero delle suddivisioni nelle tre direzioni xyz deve essere pari al numero di processi impiegati.

```
†
FoamFile
{
version 2.0;
format ascii;
class dictionary;
note "mesh decomposition control dictionary ";
location "system";
object decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
numberOfSubdomains 2;
//Numero dei processori
method hierarchical;
```

```
hierarchicalCoeffs
{
n (1 2 1);
delta 0.001;
order xyz;
}
distributed no;
// ************************************************************************* //
†
```

Nota: OpenFOAM dà la possibilità di generare la griglia anche con altri programmi, infatti sono forniti dei convertitori per i più popolari formati (Fluent, STAR-CD/PROSTAR, GAMBIT, Ansys, CFX). Per maggiori informazioni è possibile consultare la guida del programma [25].

Capitolo 5

Schematizzazione del problema matematico

Come precedentemente indicato, l'argomento di studio è rivolto alla descrizione del comportamento di una deriva di una barca a vela, in movimento a una velocità di 3 $\frac{m}{s}$. Si deve quindi tradurre, da un punto di vista matematico, un flusso incomprimibile, turbolento e stazionario.
Si è deciso di spiegare, in questo capitolo, come schematizzare il caso e come inizializzare il problema, prendendo ad esempio la prima configurazione analizzata ad incidenza di 4°. Per le altre condizioni di progetto e gli studi di sensibilità verranno fatte le opportune distinzioni nel capitolo dei risultati.

5.1 Schematizzazione del caso

OpenFOAM consente la schematizzazione del problema attraverso la compilazione di tre file, compresi nella cartella *system* del caso; essi sono:

- *controlDict*, alla base del controllo temporale della simulazione, in cui vi sono anche: il modello matematico adottato per descrivere il problema e le informazioni riguardo il salvataggio dei dati;
- *fvScheme*, fondamentale per la definizione del tipo di schemi adottati per risolvere gli operatori matematici presenti nelle equazioni;
- *fvSolution*, utile per controllare i solutori delle equazioni, le tolleranze e gli algoritmi.

5.1.1 Il file *controlDict*

Nel documento *controlDict* si elenca, in primo luogo, il tipo di modello matematico con il quale si vuol schematizzare la fisica del problema; nel caso di flusso incomprimibile,

turbolento e stazionario si può utilizzare l'applicazione fornita da OpenFOAM sotto il termine di *simpleFoam* [16]. Si devono poi elencare una serie di entrate temporali, per concludere con le informazioni sul salvataggio dei dati.

Nella tabella seguente, si specificano gli input per la suddivisione temporale [17].

Nome entrata	**Descrizione**
startFrom	Controlla il tempo di inizio della simulazione. Si può scegliere tra: *firstTime* (dal più basso intervallo di tempo salvato) *startTime* (dal tempo specificato nell'entrata *startTime*) *latestTime* (dal più recente passo di tempo salvato)
startTime	Indica da quale tempo si vuole partire
stopAt	Definisce quando far terminare la simulazione. Possibilità tra: *endTime* (tempo introdotto da *endTime*) *writeNow* (completa il passo in atto e si scrivono i dati) *noWriteNow* (completa il passo in atto ma non si scrivono i dati) *nextWrite* (termina la simulazione al completamento del tempo prestabilito da *writeControl*)
endTime	Indica il tempo per cui deve essere terminata la simulazione
deltaT	Intervallo di tempo della simulazione

Le entrate per la preparazione del salvataggio dei dati sono invece nella tabella riportata a seguire [18].

Nome entrata	Descrizione e possibili definizioni
writeControl	Controlla ogni quanto scrivere i file in uscita: *timeStep* (Scrittura per ogni passo di *writeInterval*) *runTime* (Scrittura per ogni secondo di *writeInterval* del tempo simulato) *adjustableRunTime* (Scrittura per ogni secondo di *writeInterval* del tempo simulato, aggiustando il passo temporale, se necessario, per renderlo coincidente con *writeInterval*) *cpuTime* (Scrittura per ad ogni secondo di *writeInterval* del tempo della CPU) *clockTime* (Scrittura per ad ogni secondo di *writeInterval* del tempo reale)
writeInterval	Definizione di un numero legato a *writeControl*
purgeWrite	Limite sul numero di cartelle di salvataggio sovrascritte ciclicamente. Per disabilitarlo è sufficiente porlo pari a 0
writeFormat	Tipo di formato dei dati salvati: *ascii* (Formato ASCII, legato all'entrata *writePrecision*) *binary* (Formato binario)
writePrecision	Numero predefinito pari a 6, legato a *writeFormat*
writeCompression	I file possono essere salvati in cartelle compresse
timeFormat	Formato del nome delle cartelle temporali: *fixed* (±m.dddddd dove il numero delle d è impostato da *timePrecision*) *scientific* (±m.ddddddde±xx, il numero delle d è impostato da *timePrecision*) *general* (Formato scientifico se l'esponente è minore di -4 o maggiore/uguale a quello specificato da *timePrecision*)
timePrecision	Numero predefinito pari a 6, legato a *timeFormat*
graphFormat	Formato dei dati dei grafici *raw* (Formato ASCII in colonne) *gnuplot* (Dati in formato gnuplot) *xmgr* (Dati in formato Grace/xmgr) *jplot* (Dati in formato jPlot)
runTimeModifiable	A seconda che si ponga *yes/no*, il file può essere letto ad ogni passo di calcolo

Nella parte finale del documento, possono essere aggiunte altre due categorie di entrate:

- la *libs*, lista di librerie di cui si vuole usufruire e quindi devono essere caricate al momento della simulazione;
- la *functions*, insieme di funzioni da utilizzare durante il processo.

Per la prima configurazione a 4° di incidenza, si è scelto di far partire il processo al tempo 0 e, nel caso sia interrotto prima di raggiungere il tempo prestabilito, il calcolo deve riprendere dall'ultimo tempo salvato, definito dal *writeInterval*. Il termine della simulazione è stato posto pari a 2000 secondi, ma in generale va inserito il tempo per il quale si pensa che l'applicazione arrivi a convergenza. Come intervallo si è scelto il secondo, infatti, essendo un caso stazionario, è sufficiente che il calcolo proceda, mentre non ha importanza la precisione della simulazione temporale.
Per il salvataggio dei dati, è stato preferito il formato ASCII e cartelle non compresse. Infine si è fatto ricorso ad un paio di funzioni che consentono il calcolo dei valori delle forze e dei momenti aerodinamici agenti sulla deriva e i corrispondenti coefficienti. Tali informazioni vengono salvate in file dentro a cartelle temporali, a loro volta poste in *forces* e *forceCoeffs* [19].

Per completezza di esposizione, si trascrive il file *controlDict* utilizzato per lo studio.

```
†
FoamFile
{
version 2.0;
format ascii;
class dictionary;
location "system ";
object controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application simpleFoam;
startFrom latestTime;
startTime 0;
stopAt endTime;
endTime 2000;
deltaT 1;
writeControl timeStep;
writeInterval 50;
purgeWrite 0;
writeFormat ascii;
writePrecision 6;
writeCompression uncompressed;
timeFormat general;
timePrecision 6;
runTimeModifiable yes;
functions
{
forces
```

```
{
type forces;
functionObjectLibs (libforces.so); //Libreria da caricare
patches (deriva_solid); // Nome della griglia della deriva
rhoName rhoInf;
rhoInf 997.561; //Densità del fluido di riferimento
CofR (0 0 0); //Origine per il calcolo del momento
outputControl timeStep;
outputInterval 1;
}
forceCoeffs
{
// rhoInf - Densità di riferimento
// CofR - Centro di rotazione
// dragDir - Direzione del coefficiente di resistenza
// liftDir - Direzione del coefficiente di portanza
// pitchAxis - Asse momento di beccheggio
// magUinf - Valore della velocità di riferimento
// lRef - Lunghezza di riferimento
// Aref - Area di riferimento
type forceCoeffs;
functionObjectLibs (libforces.so);
patches (deriva_solid);
rhoName rhoInf;
rhoInf 997.561;
CofR (0 0 0);
liftDir (-0.069756474 0.99756405 0);//Incidenza di 4 gradi
dragDir (0.99756405 0.069756474 0);
pitchAxis (0 0 1);
magUInf 3;
lRef 1.26;
Aref 2.038;
outputControl timeStep;
outputInterval 1;
}
}
// ************************************************************************* //
†
```

5.1.2 Il file *fvScheme*

OpenFOAM consente ampia libertà sulla scelta degli schemi applicabili alla discretizzazione degli operatori matematici presenti nell'equazioni del problema.

I termini principali da definire nel file sono [20]:

interpolationSchemes, interpolazione dei valori da punto a punto;

snGradSchemes, componente del gradiente normale al piano della cella:

gradSchemes, gradiente

divSchemes, divergenza

laplacianSchemes, laplaciano

timeSchemes, prima e seconda derivata parziale rispetto al tempo

fluxRequired, campi che richiedono la generazione del flusso

Per ogni entrata si può introdurre uno schema predefinito (*default*) generale e, nel caso in cui per alcuni termini si volesse ricorrere ad altri metodi, è sufficiente elencarli con il tipo di risolutore.

Dato che questa tesi, come già detto, non vuole essere una sostituzione della guida all'utilizzo, e considerata la vastità dell'argomento, si è scelto di dare una descrizione generale del file tipo utilizzato nello studio, mentre si rimanda al manuale [21] e ai *tutorials* forniti da OpenFOAM per approfondire i singoli argomenti.

Per la compilazione dei termini del laplaciano, della divergenza e del gradiente, si è preso spunto da un caso dibattuto tra alcuni utilizzatori del programma. Infatti la discussione era centrata sulla comparazione di uno studio su un profilo, con lo scopo di far avvicinare i risultati ottenuti con OpenFOAM a quelli avuti con Fluent. Alcune indicazioni sono risultate utili per introdurre delle limitazioni già presenti implicitamente in Fluent [22]. Data la similitudine con il caso di studio della tesi, seppur il confronto deve essere sostenuto con *Star-ccm*$_+$, si è pensato di seguire le informazioni consigliate almeno come punto di partenza dell'analisi.

interpolationScheme

Per l'*interpolationSchemes*, che contiene i termini solitamente interpolati dal centro della cella al centro della faccia, OpenFOAM fornisce quattro categorie tra cui scegliere: una di schemi generici e tre usati principalmente con la discretizzazione di Gauss del termine divergenza.
Per il caso generale, si è preferito utilizzare uno schema centrato, come consigliato in [23]; nello specifico, quello lineare.

snGradSchemes

Questo schema contiene i termini del gradiente normale alla superficie, valutato in corrispondenza della faccia della cella; è la componente normale del gradiente dei valori nel centro di due celle connesse da una stessa faccia.
Per lo studio, si è optato per la definizione *corrected*, ovvero per una correzione esplicita non ortogonale.

gradScheme

I termini che contengono i gradienti sono stati schematizzati con l'utilizzo di un modello del secondo ordine: *leastSquares*, nella versione limitata *faceLimited*.

divScheme

Questo schema agisce sull'operatore divergenza, il tipo di formattazione usata all'interno del file è quella di indicare ad esempio il termine $\nabla\cdot(\rho UU)$ con div(phi,U), dove $phi = \rho U$.
Come unica scelta di discretizzazione è possibile utilizzare lo schema di Gauss, di cui si deve selezionare il modello di interpolazione per il campo.
A questo livello si può indicare uno schema del tutto indipendente da quello specificato in precedenza ed ha un'importanza fondamentale sul comportamento numerico della soluzione.

Generalmente si è scelto di usufruire di due metodi di interpolazione, l'*upwind* e il *linear*. Per il termine div(phi,U) si è optato per una altro tipo, il *GammaV*. Gamma sta ad indicare un modello appartenente alla classe NVD (normalised variation diminishing); mentre la V sta per una versione migliorata di alcuni schemi limitati, adatta ai campi vettoriali. Il limite viene formulato tenendo conto della direzione del campo.

laplacianScheme

Da un punto di vista di sintassi, i termini laplaciani vengono così indicati nel file: $\nabla\cdot(\nu\nabla U)$ con laplacian(nu,U).
I termini che contengono l'operatore laplaciano hanno come unica scelta di discretizzazione lo schema di Gauss, che però richiede sia un modello di interpolazione che uno schema del gradiente normale alla superficie.
Si è preferito usare per l'interpolazione il tipo *harmonic*, mentre per il gradiente il *limited*.

timeScheme

Questo schema rappresenta le derivate parziali rispetto al tempo; il caso di studio è stazionario e quindi è stata definita la derivata prima *ddtScheme* come *steadyState*.

Si può eventualmente anche avere la derivata al secondo ordine definendo il termine *d2dt2Schemes*.

fluxRequired

Il flusso è generato dopo che è stata risolta l'equazione della pressione, quindi nel caso in esame è sufficiente menzionare questa grandezza fisica con la lettera che la identifica: *p*.

Per concludere il sottoparagrafo, si riporta il file base *fvScheme* utilizzato per lo studio.

```
†
FoamFile
{
version 2.0;
format ascii;
class dictionary;
location "system ";
object fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
      default steadyState;
}
gradSchemes
{
      default faceLimited leastSquares 0.5;
}
divSchemes
{
       default none;
       div(phi,U) Gauss GammaV 1;
       div(phi,k) Gauss upwind;
       div(phi,epsilon) Gauss upwind;
       div(phi,R) Gauss upwind;
       div(R) Gauss linear;
       div(phi,nuTilda) Gauss upwind;
       div((nuEff*dev(grad(U).T()))) Gauss linear;
}
laplacianSchemes
{
```

```
        default Gauss harmonic limited 0.5;
}
interpolationSchemes
{
        default linear;
        interpolate(U) linear;
}
snGradSchemes
{
        default corrected;
}
fluxRequired
{
        default no;
        p ;
}
//************************************************************** //
†
```

5.1.3 Il file *fvSolution*

Il file *fvSolution* controlla il tipo di solutore per le equazioni, le tolleranze e gli algoritmi [24]. La struttura, tipica dei file OpenFOAM, è divisa in una serie di parti, a loro volta specificate da varie entrate.

Anche in questo caso si propone una visione generale del tipo di documento da compilare, focalizzando poi l'attenzione sulle scelte fatte per l'analisi.

Solvers

La prima entrata da specificare è quella che caratterizza i solutori, *solver*, di tipo lineare, da introdurre per ogni variabile da calcolare.
OpenFOAM consente la scelta tra diversi metodi risolutivi, che sono:

- gradiente (bi-) coniugato precondizionato (PCG per matrici simmetriche/PBiCG per asimmetriche);
- solutore che utilizza uno *smoother* (smoothSolver);
- multi-griglia geometrico-algebrico generalizzato (GAMG);

Tutte le categorie appena elencate hanno bisono di ulteriori specificazioni, ad esempio per la prima si deve indicare come si opera il precondizionamento, per la seconda invece si deve scegliere il tipo di *smoother* da adottare, il terzo metodo deve

invece essere supportato da una serie di informazioni che descrivono il procedimento. Per maggiori informazioni si rimanda a [24].

Una volta scelto il tipo di solutore, si devono definire la tolleranza e la tolleranza relativa. Si determina così quando si debba termiare lo svolgimento dei calcoli. Quando i residui sono inferiori alla tolleranza oppure quando il rapporto tra il valore dei residui finale e iniziale è inferiore alla tolleranza relativa, il conto si interrompe. In tal modo, si ottiene un controllo sull'accuratezza della soluzione. Il residuo iniziale viene calcolato con il valore presente del campo e, nel corso del processo, viene aggiornato ad ogni passo.
Per forzare che la soluzione converga in base alla tolleranza, si deve porre la relativa pari a 0.

Algoritmi

Dato che lo studio è rivolto a un caso fluidodinamico, si deve considerare l'input *SIMPLE*, che fa riferimento al metodo semi-implicito per le equazioni legate alla pressione. Questo algoritmo è una procedura iterativa per risolvere le equazioni per la velocità e la pressione, nel caso stazionario. Si basa sulla valutazione di una soluzione iniziale che viene poi corretta.
All'interno della definizione di *SIMPLE*, deve essere specificata la correzione della non ortogonalità della mesh attraverso la parola chiave *nNonOrthogonalCorrectors*, ponendo 0 si indica una mesh ortogonale mentre valori più alti sono validi per quelle altamente non ortogonali.
Deve poi essere definito un valore di riferimento della pressione *pRefValue* per una cella *pRefCell*, altrimenti il caso incorre in un errore computazioanle.

relaxationFactors

Da ultimo vi sono i fattori di rilassamento *relaxationFactors*, utili per aumentare la stabilità della soluzione, soprattutto per i casi stazionari. Agiscono nel limitare il valore di cui può cambiare una variabile da un'iterazione all'altra. I fattori possono essere assegnati nell'intervallo che va da 0 a 1; nel caso limite in cui sia posto pari a 0, la soluzione non varia da un passo all'altro. La soluzione migliore è quella per cui sia assicurata sia la stabilità ma anche una flessibilità tale da garantire la convergenza. Per introdurli nel file, è sufficiente mettere il valore voluto a fianco della parola chiave che identifica la variabile del problema.

A conclusione si riporta per intero uno dei file *fvSolution* utilizzato per l'analisi; dato che alcuni parametri sono stati cambiati per far migliorare la soluzione, quello ora presentato è quello relativo alla prima configurazione studiata per un'incidenza di 4° e deve essere preso come esempio; comunque, quando richiesto, si specificheranno i cambiamenti adottati.

Per compilarlo si è preso a riferimento il documento presente nella sezione dei *tutorials*, nel caso che utilizza come modello di turbolenza le RANS, trovabile nel percorso *tutorials\incompresssible\pisoFoam\ras*.
I solutori preferiti sono quelli dei gradienti coniugati e, per il precondizionamento, si è fatto uso del metodo della diagonale incompleta-Cholesky (simmetrico DIC, asimmetrico DILU).

```
†
FoamFile
{
version 2.0;
format ascii;
class dictionary;
location "system ";
object fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
        p
        {
                solver PCG;
                preconditioner DIC;
                tolerance 1e-06;
                relTol 0.01;
        }
        U
        {
                solver PBiCG;
                preconditioner DILU;
                tolerance 1e-05;
                relTol 0.1;
        }
        k
        {
                solver PBiCG;
                preconditioner DILU;
                tolerance 1e-05;
                relTol 0.1;
        }
        epsilon
        {
                solver PBiCG;
```

```
        preconditioner DILU;
        tolerance 1e-05;
        relTol 0.1;
    }
    nuTilda
    {
        solver PBiCG;
        preconditioner DILU;
        tolerance 1e-05;
        relTol 0.1;
    }
}
SIMPLE
{
nNonOrthogonalCcorrectors 0;
pRefCell 0;
pRefValue 0;
}
relaxationFactors
{
default 0;
p 0.3;
U 0.7;
k 0.7;
epsilon 0.7;
nuTilda 0.7;
}
// ************************************************************************* //
†
```

5.2 Inizializzazione del problema

Per poter definire il problema ben posto da un punto di vista di schematizzazione matematica, si deve definire il modello di turbolenza da adottare nella risoluzione del caso e le condizioni iniziali delle variabili caratteristiche. Nel seguito si descrive quindi come completare il modello in OpenFOAM.

5.2.1 Modello di turbolenza

Una volta definito il tipo di flusso, si deve specificare il modello di turbolenza. Per attenersi alle condizioni utilizzate nella prova con *Star-ccm*$_+$, si adotta il metodo

RANS con modello $k - \varepsilon$.
La prassi offerta da OpenFOAM è nuovamente quella di compilare alcuni file da salvare nella cartella *constant*. Nell'elenco seguente si enunciano i documenti e se ne trascrive per intero il contenuto:

- con *turbulenceProperties* si definisce il tipo di turbolenza che si vuole utilizzare.
  ```
  †
  FoamFile
  {
  version 2.0;
  format ascii;
  class dictionary;
  location "constant";
  object turbulenceProperties;
  }
  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
  simulationType RASModel;
  //*************************************************************** //
  †
  ```

- con *RASProperties* si caricano i coefficienti caratteristici del modello di turbolenza. Se si vogliono cambiare è sufficiente elencarli.
  ```
  †
  FoamFile
  {
  version 2.0;
  format ascii;
  class dictionary;
  location "constant";
  object RASProperties;
  }
  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
  RASModel kEpsilon;
  turbulence on;
  printCoeffs off;
  // ************************************************************** //
  †
  ```

- con *transportProperties* si introducono le proprietà del modello di trasporto ed alcuni valori tipici del fluido, come la densità e la viscosità cinematica dell'acqua.

```
†
FoamFile
{
version 2.0;
format ascii;
class dictionary;
location ''constant'';
object transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
transportModel Newtonian;
rho rho [ 1 -3 0 0 0 0 0 ] 997.561;
nu nu [ 0 2 -1 0 0 0 0 ] 1.005e-06;
// ************************************************************//
†
```

5.2.2 Condizioni al bordo

A questo punto è necessario introdurre le condizioni iniziali al bordo e nel campo, definendo alcuni file all'interno della cartella *0*.
Per il caso studiato dovranno essere compilati quelli che regolano la pressione p, la velocità U e tutti quei parametri utili per il modello di turbolenza, ovvero k, ε, $\widetilde{\nu}$, ν_t.

Dato che vengono richiesti, per tutte queste grandezze, dei valori iniziali sia all'interno del campo che ai bordi, si sono sfruttate per la loro stima le relazioni:
$\mu_t = \rho C_\mu \frac{k^2}{\varepsilon}$ con $C_\mu = 0.09$,
$\widetilde{\nu} = \sqrt{\frac{3}{2}}(UIl)$
con $I = 1\%$ intensità di turbolenza
$l = 5\%$ lunghezza caratteristica della turbolenza
$k = \frac{3}{2}(UI)^2$
$\varepsilon = C_\mu^{3/4} k^{3/2}/l$.

Anche se si è cercato di dare una stima iniziale di l ed I probabile, il risultato non dovrebbe esserne troppo dipendente dato che i valori cambiano durante il calcolo.

In questi documenti si devono inserire oltre i bordi del dominio anche quello della deriva, identificabile con la patch ''deriva_solid''. Per i parametri k, ε e ν_t, in corrispondenza della parete solida dei profili, sono state utilizzate delle funzioni d'onda fornite dal programma, mentre $\widetilde{\nu}$ è stato posto pari a zero, come il gradiente normale alla superficie della pressione.

Come già fatto in precedenza, si elencano ora i nomi dei vari file, con la loro descrizione e il relativo tratto di codice:

- file *p*. Contiene le informazioni sulla pressione, posta uniforme e pari a zero all'interno del campo, mentre ai bordi del dominio è stata sfruttata la parola chiave fornita dal programma *freestreamPressure*, ovvero pressione della corrente libera.

```
†
FoamFile
{
version 2.0;
format ascii;
class volScalarField;
object p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 2 -2 0 0 0 0];
internalField uniform 0;
boundaryField
{
        inlet
                {
                type freestreamPressure;
                }
        outlet
                {
                type freestreamPressure;
                }
        box
                {
                type freestreamPressure;
                }
        "deriva_solid"
                {
                type zeroGradient;
                }
}
// ************************************************************************* //
†
```

- file *U*. Si introducono le componenti della velocità nelle tre direzioni principali. Sono state definite sia all'esterno che all'interno del dominio; per i bordi si è

utilizzata l'entrata *freestream*, mentre sul corpo è stata posta pari a zero.

```
†
FoamFile
{
version 2.0;
format ascii;
class volVectorField;
object U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 1 -1 0 0 0 0];
internalField uniform (2.992692151 0.209269421 0);//incidenza di 4 gradi,
                                  //velocità di 3 metri al secondo
boundaryField
{
        inlet
                {
                type freestream;
                freestreamValue uniform (2.992692151 0.209269421 0);
                }
        outlet
                {
                type freestream;
                freestreamValue uniform (2.992692151 0.209269421 0);
                }
        ''deriva_solid''
                {
                type fixedValue;
                value uniform (0 0 0);
                }
       box
                {
                type freestream;
                freestreamValue uniform (2.992692151 0.209269421 0);
                }
}
// ************************************************************************* //
†
```

- file *nuTilda* ($\widetilde{\nu}$). Valore definito sia internamente che esternamente al dominio, ai bordi è pari al *freestream*, con valore calcolato precedentemente, mentre in

corrispondenza della deriva è stato posto pari a zero.

```
†
FoamFile
{
version 2.0;
format ascii;
class volScalarField;
object nuTilda;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 2 -1 0 0 0 0];
internalField uniform 0.001837117;
boundaryField
{
        inlet
                {
                type freestream;
                freestreamValue uniform 0.001837117;
                }
        outlet
                {
                type freestream;
                freestreamValue uniform 0.001837117;
                }
        "deriva_solid"
                {
                type fixedValue;
                value uniform 0;
                }
       box
                {
                type freestream;
                freestreamValue uniform 0.001837117;
               }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
†
```

- file *nut* (ν_t). Definito sfruttando sempre la parola chiave *freestream* e la funzione d'onda per la condizione alla parete, il valore di riferimento è quello calcolato precedentemente.

```
†
FoamFile
{
version 2.0;
format ascii;
class volScalarField;
object nut;
}
// ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 2 -1 0 0 0 0];
internalField uniform 0.000089284;
boundaryField
{
    inlet
        {
        type freestream;
        freestreamValue uniform 0.000089284;
        }
    outlet
        {
        type freestream;
        freestreamValue uniform 0.000089284;
        }
    "deriva_solid"
        {
        type nutWallFunction;
        value uniform 0.000089284;
        }
    box
        {
        type freestream;
        freestreamValue uniform 0.000089284;
        }
}
// ****************************************************** //
†
```

- file *epsilon* (ε), definito alla stesso modo del precedente.

```
†
FoamFile
{
```

```
version 2.0;
format ascii;
class volScalarField;
object epsilon;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 2 -3 0 0 0 0];
internalField uniform 0.000163009;
boundaryField
{
        inlet
                {
                type freestream;
                freestreamValue uniform 0.000163009;
                }
        outlet
                {
                type freestream;
                freestreamValue uniform 0.000163009;
                }
        ''deriva_solid''
                {
                type epsilonWallFunction;
                value uniform 0.000163009;
                }
        box
                {
                type freestream;
                freestreamValue uniform 0.000163009;
                }
}
// ************************************************************************* //
†
```

- file *k*, definito come il precedente.

```
†
FoamFile
{
version 2.0;
format ascii;
class volScalarField;
```

```
object k;
}
// ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 2 -2 0 0 0 0];
internalField uniform 0.00135;
boundaryField
{
    inlet
        {
        type freestream;
        freestreamValue uniform 0.00135;
        }
    outlet
        {
        type freestream;
        freestreamValue uniform 0.00135;
        }
    "deriva_solid"
        {
        type kqRWallFunction;
        value uniform 0.00135;
        }
    box
        {
        type freestream;
        freestreamValue uniform 0.00135;
        }
}
// ********************************************************* //
†
```

Capitolo 6

Risultati

In questo capitolo si dà una visione generale di come far svolgere il caso e si descrivono i risultati ottenuti, suddividendoli in due parti: nella prima viene delineata la configurazione presa a riferimento, ovvero quella ad incidenza di 4°; mentre nella seconda, si riportano tutte le variazioni fatte, dalla polare alla sensibilità della soluzione ai cambiamenti della griglia e dei metodi risolutivi dell'equazioni.

6.1 Procedura tipo

Dopo aver descritto i file da compilare per preparare il caso, è necessario trarre le somme dell'esposizione elencando i possibili iter per svolgerlo correttamente.

Si propongono due vie: una standard e una che sfrutta il calcolo in parallelo tra più processori.

Iter standard

Nel caso in cui non si voglia sfruttare il calcolo in parallelo, la procedura da seguire, una volta compilati i file precedentemente spiegati, è la seguente:

1. entrare da terminale nella cartella del caso;
2. digitare il comando *blockMesh* per creare il dominio ed eventualmente controllare il risultato come spiegato nel paragrafo ad esso dedicato nel capitolo della generazione della griglia;
3. assicurarsi preventivamente che sia presente il file *decomposeParDict* in *system*. Infatti, anche se non si usa la parallelizzazione, pare che senza questo documento il comando successivo non riesca a procedere nel calcolo, dando errore;
4. creare la mesh infittita attorno alla deriva con il comando *snappyHexMesh -overwrite*

In genere, se vengono richieste tutte e tre le fasi di modellizzazione della griglia, si creano tre cartelle temporali per ogni parte del processo; questo però può dare dei problemi nella lettura delle condizioni iniziali al momento dello studio fisico. Si utilizza allora l'argomento *overwrite* che fa sovrascrivere il salvataggio. Per conferma, si può andare a controllare nelle cartelle *constant* e *polyMesh*, dove saranno presenti i nuovi file descriventi il reticolo appena generato, per esempio in *boundary* si potrà leggere il nuovo contorno *deriva_solid*;

5. controllare di aver inserito la nuova condizione al contorno nei file di inizializzazione del problema *p, U, k, epsilon, nut, nuTilda*;

6. lanciare l'applicazione *simpleFoam*;

7. una volta terminato, si può visualizzare il risultato tramite *paraFoam*.

Calcolo in parallelo

Se si ha intenzione di parallelizzare il calcolo, si deve invece seguire l'iter:

1. entrare da terminale nella cartella del caso;

2. creare il dominio con il comando *blockMesh*;

3. stilare il file *decomposeParDict* in *system*;

4. suddividere il dominio tra i processori con *decomposePar*;

5. copiare la cartella *triSurface* nella *constant* presente in ogni *processorN*;

6. dare il comando:
   mpirun -np 2 snappyHexMesh -overwrite -parallel
   Anche in questo caso si fa sovrascrivere i salvataggi per non avere problemi con l'inizializzazione del problema;

7. copiare le condizioni al contorno della deriva all'interno dei documenti *p, U, k, epsilon, nut, nuTilda* nella cartella *0* di ciascun processore;

8. lanciare l'applicazione con:
   mpirun -np 2 simpleFoam -parallel

9. ricostruire la griglia per il tempo 0 con:
   reconstructParMesh -mergeTol 1e-03 -time 0
   Questo comando è in via di sviluppo e quindi potrebbe dare dei risultati non soddisfacienti;

10. salire di una cartella tramite *cd ..*

11. ricostruire il campo fisico con:
    reconstructPar -case ⟨ *Nome_cartella_caso* ⟩

12. ritornare nella cartella del caso e visualizzare il risultato con *paraFoam*

Si fa notare che la griglia può essere ricostruita anche prima di lanciare il comando dell'applicazione ma, nel caso in cui si voglia sfruttare il calcolo in parallelo per lo svolgimento del caso, si devono comunque aggiornare le condizioni al contorno nelle cartelle dei processori.

Indicazioni finali

Nel caso in cui si vogliano monitorare i risultati mentre vengono calcolati, si può sfruttare il comando:
mpirun -np 2 simpleFoam -parallel >log
In questa maniera, si crea un file dal nome *log* in cui vengono salvate le uscite delle informazioni sull'applicazione, come il tempo dell'iterazione o i residui.
Per leggerlo, è sufficiente richiamarlo con
tail -f log
A questo punto, se si vuole ad esempio avere una visione istantanea dei residui, si può far tracciare un grafico dal programma *gnuplot* (anch'esso FLOSS e sotto licenza GPL); si compila un documento, chiamato *Residuals* da salvare nel caso, con scritto all'interno:
†
```
set logscale y
set title ''Residui''
set ylabel 'Residui'
set xlabel 'Iterazioni'
plot ''<cat log |grep 'Solving for Ux'|cut -d'  '-f9 |tr -d ','  ''  title 'Ux'with lines,\
   ''<cat log |grep 'Solving for Uy'|cut -d'  '-f9 |tr -d ','  ''  title 'Uy'with lines,\
   ''<cat log |grep 'Solving for Uz'|cut -d' '-f9 |tr -d ','  ''  title 'Uz'with lines,\
   ''<cat log |grep 'Solving for epsilon'|cut -d'  '-f9 |tr -d ','  ''  title 'epsilon'with
lines,\
   ''<cat log |grep 'Solving for k'|cut -d'  '-f9 |tr -d ','  ''  title 'k'with lines,\
   ''<cat log |grep 'Solving for p'|cut -d'  '-f9 |tr -d ','  ''  title 'p'with lines
pause 1
reread
```
†
Da terminale, nella cartella del caso, si scrive
gnuplot Residuals
in questa maniera si apre una finestra con i grafici dei residui in scala logaritmica lungo le ordinate.

Se invece si vuole controllare il coefficiente di portanza, si deve compilare un file nominato *lift*:

```
†
set yr [0:1]
set xlabel 'Tempo [s]'
set ylabel 'Coeff. portanza [-]'
set grid
plot ''./forceCoeffs/1/forceCoeffs.dat''using ($1):($3) with lines title ''lift_coeff''
pause 1
reread
†
```

Sempre da terminale, si digita
gnuplot lift
e si ottiene l'andamento del coefficiente di portanza.

6.2 Risultati

Lo studio è stato condotto in fasi diverse, nella prima si è cercato di rappresentare una condizione di riferimento per cui la deriva è inclinata rispetto al flusso d'acqua di 4°; data la buona risposta del programma si è ricavata la polare della geometria; infine si è testata la sensibilità dei risultati alla mesh e ai cambiamenti dei parametri matematici con cui si è schematizzato il problema.

6.2.1 Prima configurazione di riferimento

La prima configurazione su cui si è posta l'attenzione è quella per cui la deriva ha un'incidenza di 4° rispetto all'acqua. Per preparare il caso allo studio con OpenFOAM, sono stati redatti i documenti inseriti progressivamente nei capitoli precedenti. Sia per il calcolo del comportamento fisico sia per la mesh si è deciso di sfruttare due processori in parallelo.

La griglia

Alla fine del processo di creazione della griglia con *snappyHexMesh*, il risultato sulla deriva è riportato in figura 6.1, dove a sinistra vi è la visione completa mentre a destra vi sono gli ingrandimenti di due particolari.

Per avere un'immagine del dominio, si rimanda alle illustrazioni:

- 6.2, sezione longitudinale del campo in prossimità del corpo;
- 6.3, due particolari della sezione longitudinale della griglia;

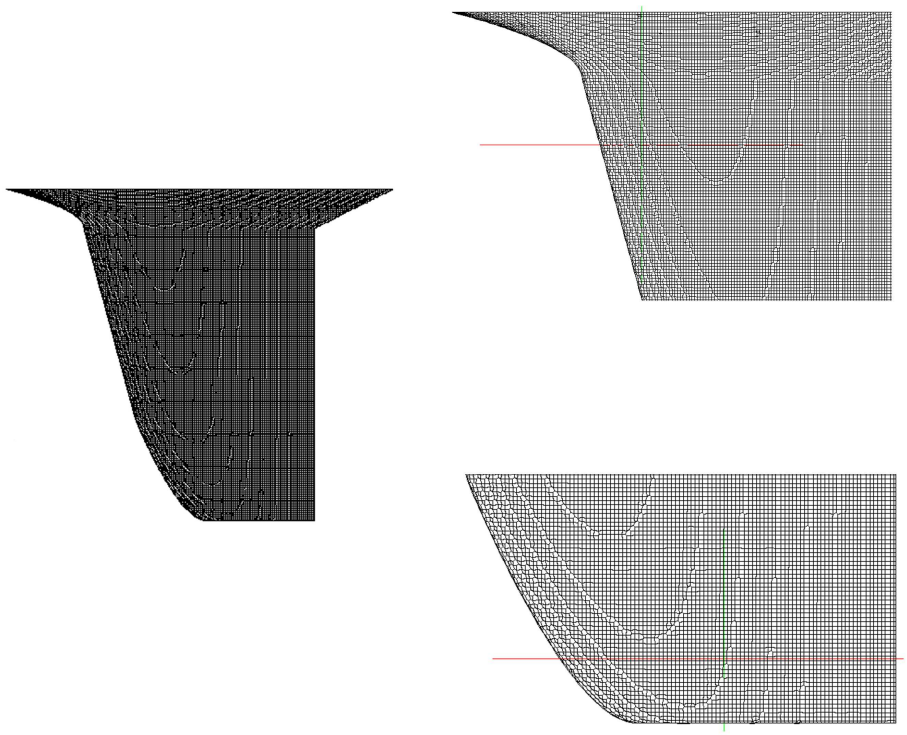

Figura 6.1: A sinistra si riporta l'immagine completa della deriva, mentre a destra vi sono due ingrandimenti su regioni particolari.

- 6.4, due sezioni trasversali della griglia ad un'altezza di 0.3 metri dal piano superiore del dominio;
- 6.5, particolari del bordo d'attacco.

Dall'immagine 6.2 si nota che all'interno della griglia vi sono delle zone che non sono state elaborate correttamente durante il processo di modellizzazione; si è pensato allora di vedere se questo dipendesse dal numero di processori impiegati per calcolare la mesh. Il risultato verrà presentato nella sezione dedicata alla sensibilità del programma alla variazione dei parametri.

Comportamento fisico

Una volta raggiunta la convergenza del calcolo, si sono potuti ricavare i valori del coefficiente di portanza e di resistenza, che sono rispettivamente:
$C_L = 0.275$ e $C_D = 0.0195$
Questa stima è stata fatta prendendo gli ultimi cento valori assunti una volta raggiunta la stabilità del risultato; si nota che, mentre il coefficiente di portanza varia in un intervallo compreso tra 0.2730 e 0.2760, quello di resistenza oscilla tra 0.01930 e

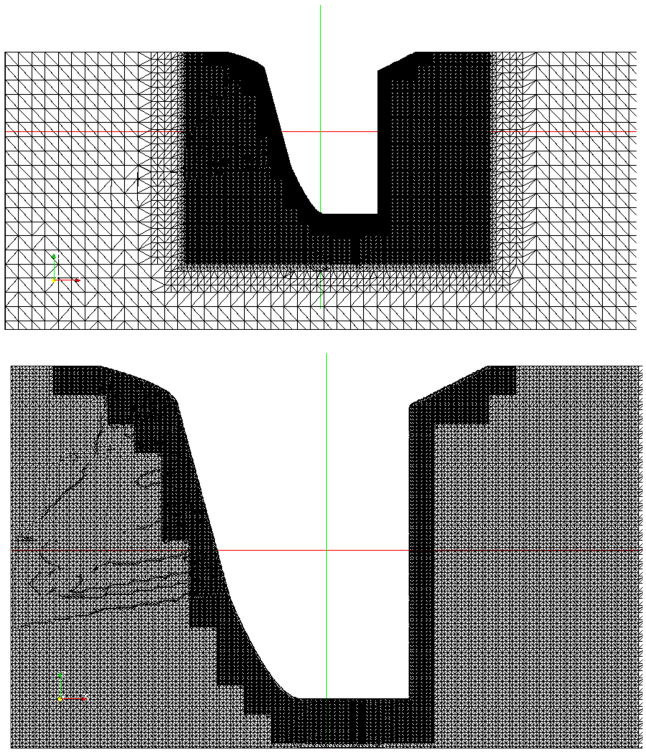

Figura 6.2: Ingrandimenti della sezione longitudinale della griglia in prossimità della deriva.

0.01960.
Inoltre, si è riscontrato che il numero di iterazioni necessarie per poter definire l'andamento stabile è risultato diverso a seconda del parametro, infatti per quello di resistenza, sono stati necessari 1200 passi (identificabili nel programma con i secondi della simulazione), per quello di portanza 1600, come si può vedere dalle due figure 6.6 e 6.7.

Andando a fare il confronto con i dati ottenuti da *Star-ccm*$_+$, essi sono:
$C_L = 0.245$ e $C_D = 0.0173$
Risultano quindi non eccessivamente distanti da quelli di OpenFOAM, se ne deduce che è come se il profilo lavorasse ad un'incidenza maggiore di circa il 10 % rispetto a quella di paragone. Comunque, andando a calcolare l'efficienza in entrambi i casi, si nota che con OpenFOAM è pari a 14.11, con *Star-ccm*$_+$ è di 14.16, quindi fondamentalmente uguali.

La differenza riscontrata può essere attribuita a una mesh non ottimizzata sulla parete della deriva, infatti pur essendo il numero delle celle 3485165, è possibile che un reticolo più fitto sul contorno consenta una migliore stima dello strato limite. Altra causa può essere il valore diverso assunto da alcuni parametri di controllo che *Star-ccm*$_+$ impone in automatico, mentre OpenFOAM richiede di inserirli a discrezione

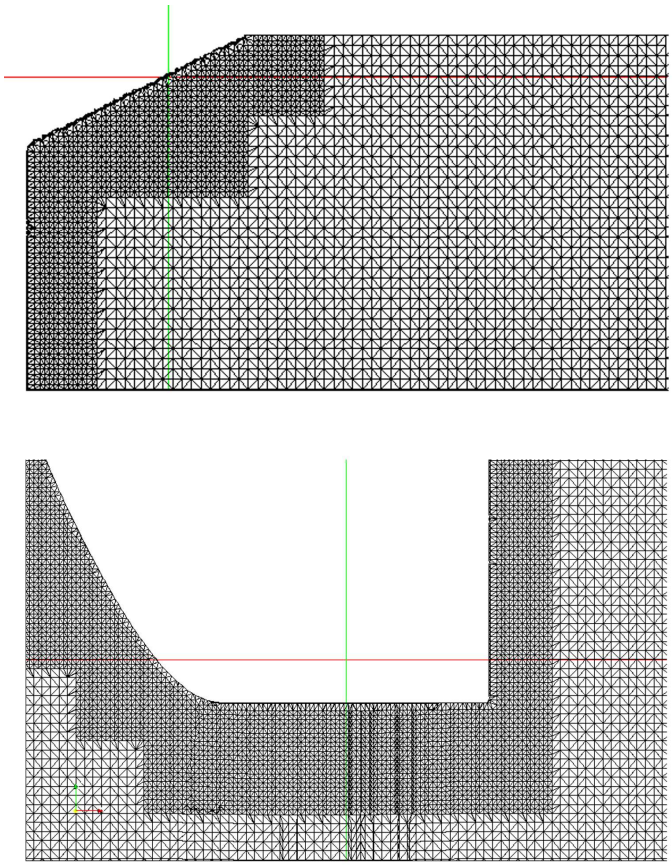

Figura 6.3: Particolari della sezione longitudinale della griglia.

dell'utilizzatore. Va comunque sottolineato che, facendo una stima con la teoria di Prandtl, il valore del coefficiente di portanza risulta pari a 0.28, per cui più attinente a quello uscente da questo studio.

Completando il confronto della soluzione, si è notato che non solo i valori aerodinamici ma anche il comportamento della soluzione hanno dinamiche differenti; la convergenza del calcolo nel caso di *Star-ccm*$_+$ è stata raggiunta dopo circa 400 iterazioni. Si è indagato, per quanto fosse possibile, su quali impostazioni rendessero il processo più rapido e stabile. Si è capito allora che come solutore delle variabili delle equazioni il programma imposta il metodo Gauss-Seidel e dei fattori di rilassamento pari a 0.8. Pertanto si è deciso di rifare il calcolo inserendo queste informazioni nel caso di studio. Nel sottoparagrafo seguente si analizzerà il risultato di tali cambiamenti.

6.2.2 Risultato dei cambiamenti dei parametri

Per cambiare l'impostazione dei solutori e renderli coerenti con quelli di *Star-ccm*$_+$, si è modificato il file *fvSolution*. Come già introdotto nel capitolo della schematizzazione

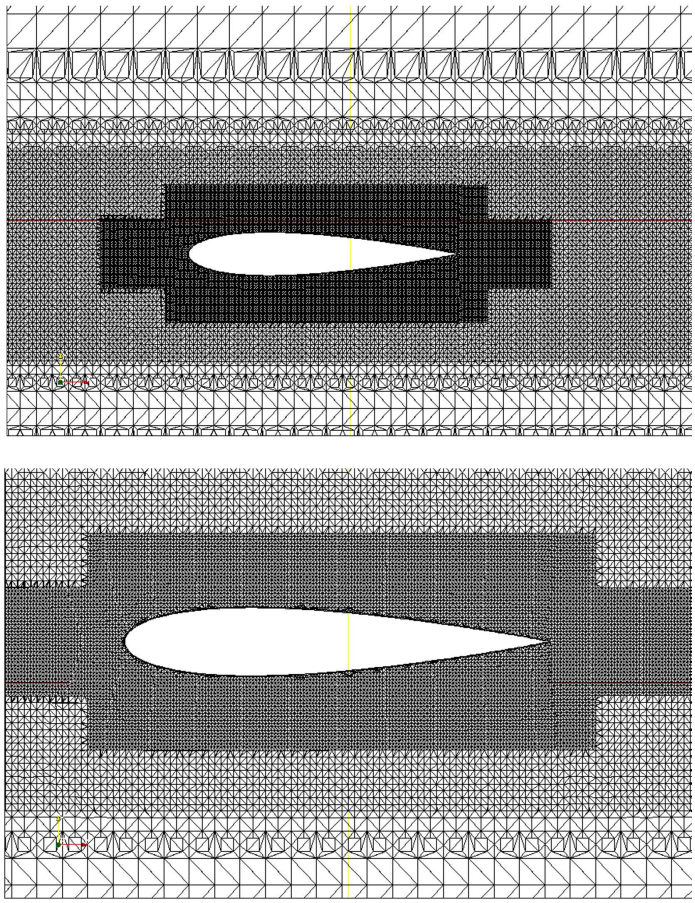

Figura 6.4: Sezione trasversale del dominio ad un'altezza di 0.3 metri dal piano superiore dello stesso, la seconda è una visione maggiormente ingrandita.

del problema matematico, OpenFOAM consente tre tipi di solutori, ma per poter sfruttare l'impostazione Gauss-Seidel, si sono adottati i metodi:

- *GAMG*, (metodo generale di una multi-griglia geometrica-algebrica), per la pressione. L'algoritmo genera, in primo luogo, una soluzione su una mesh con poche celle, che viene riportata su una griglia più fitta e usata come stima iniziale per una soluzione successiva più accurata. Ne consegue una velocità maggiore di calcolo, se non risente eccessivamente del costo di rifinitura e mappatura della mesh. Per il caso di studio è risultato un metodo più agile per la risoluzione della pressione rispetto agli altri modelli e, per tale motivo, preferito. Dato che esso iniza con una griglia stabilita dall'utente da rifinire passo passo, è necessario introdurre una dimensione approssimata di partenza in termini di numero di celle (*nCoarsestCells*) e definire alcune entrate utili all'algoritmo come l'agglomerazione di celle e il suo metodo di calcolo (consigliato il *faceAreaPair*).

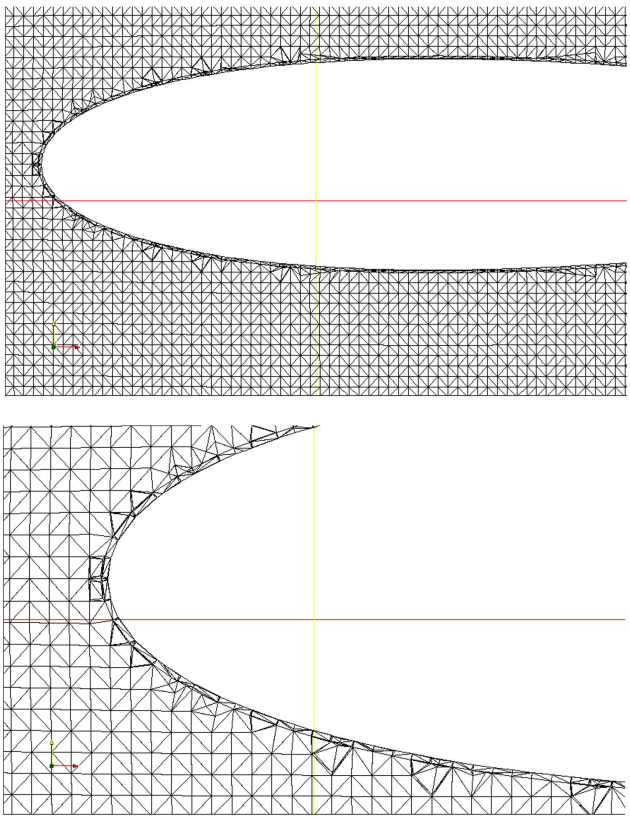

Figura 6.5: Particolari della griglia in corrispondenza del bordo d'attacco del profilo all'altezza di 0.3 metri dal piano superiore del dominio.

La specifica desiderata di *Gauss-Seidel* si utilizza nella definizione del tipo di *smoother*; infine devono essere riportati il numero di ripassi nei momenti differenti di rifinitura della mesh (*nPreSweeps, nPostSweeps, nFinestSweeps*) e il controllo della velocità operativa (*mergeLevels*);

- *smoother*, per le altre variabili. Se ne deve definire il tipo (appunto *GaussSeidel*) e il numero di passate prima che sia ricalcolato il residuo (*nSweeps*).

Per quanto riguarda i fattori di rilassamento, mentre *Star-ccm*$_+$ adotta in automatico il valore 0.8, in OpenFOAM si è preferito lasciare per la pressione 0.3 e per le altre variabili 0.7, in modo tale da consentire una variazione veloce dei valori ma stabilità di soluzione.

A seguire si riporta il file utilizzato per condurre la prova; si vuole specificare che, ponendo la tolleranza relativa pari a 0, si è costretta l'iterazione a terminare solo quando la differenza tra due valori successivi fosse inferiore alla tolleranza.

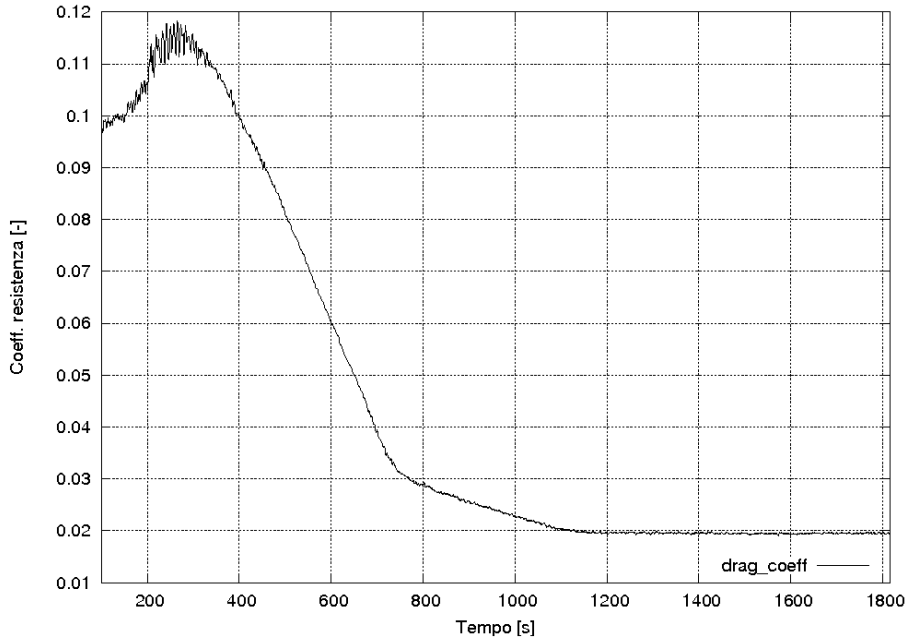

Figura 6.6: Andamento del coefficiente di resistenza per la configurazione di riferimento a 4° di incidenza. I passi di iterazione sono identificabili con i secondi della simulazione.

```
    †
FoamFile
{
version 2.0;
format ascii;
class dictionary;
location "system ";
object fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
        p
        {
                solver GAMG;
                tolerance 1e-06;
                relTol 0;
                smoother GaussSeidel;
                nPreSweeps 0;
                nPostSweeps 2;
```

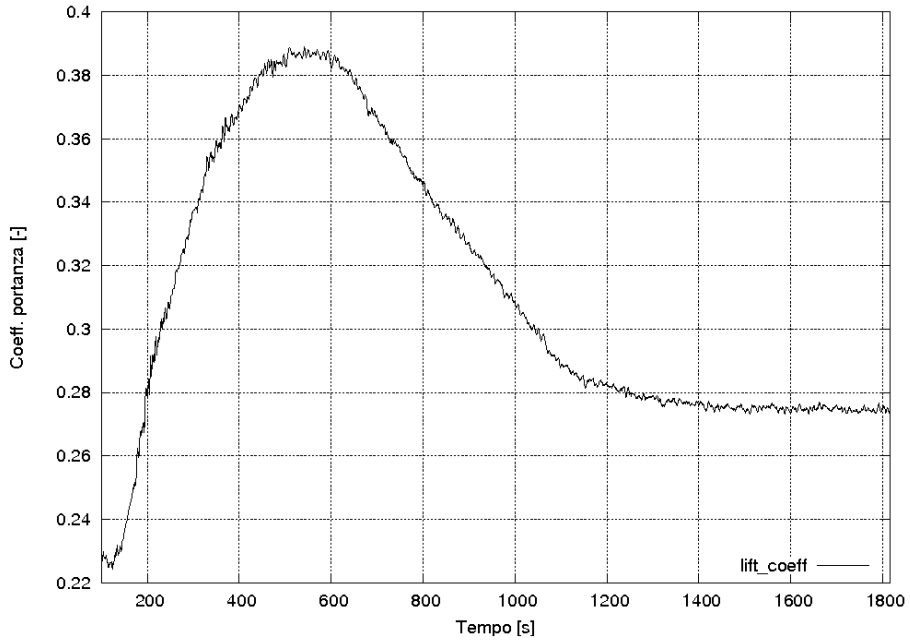

Figura 6.7: Andamento del coefficiente di portanza per la configurazione di riferimento a 4° di incidenza. I passi di iterazione sono identificabili con i secondi della simulazione.

```
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator faceAreaPair;
        mergeLevels 1;
    }
    U
    {
        solver smoothSolver;
        smoother GaussSeidel;
        nSweeps 2;
        tolerance 1e-05;
        relTol 0;
    }
    k
    {
        solver smoothSolver;
        smoother GaussSeidel;
        nSweeps 2;
        tolerance 1e-05;
        relTol 0;
```

```
        }
        epsilon
        {
                solver smoothSolver;
                smoother GaussSeidel;
                nSweeps 2;
                tolerance 1e-05;
                relTol 0;
        }
        nuTilda
        {
                solver smoothSolver;
                smoother GaussSeidel;
                nSweeps 2;
                tolerance 1e-05;
                relTol 0;
        }
}
SIMPLE
{
nNonOrthogonalCorrectors 0;
pRefCell 0;
pRefValue 0;
}
relaxationFactors
{
default 0;
p 0.3;
U 0.7;
k 0.7;
epsilon 0.7;
nuTilda 0.7;
}
// ************************************************************************* //
†
```

I cambiamenti fatti hanno dato i risultati sperati, infatti la convergenza è stata raggiunta con minori iterazioni rispetto al caso di partenza; circa 500 passi per il coefficiente di resistenza e 700 per quello di portanza (figg. 6.8, 6.9); anche i loro valori sono coerenti con quelli precedentemente calcolati, essendo:
$C_L = 0.275$, $C_D = 0.0197$ e l'efficienza pari a 14.04.
Anche in questo caso, i risultati sono stati stimati facendo la media sugli ultimi 100

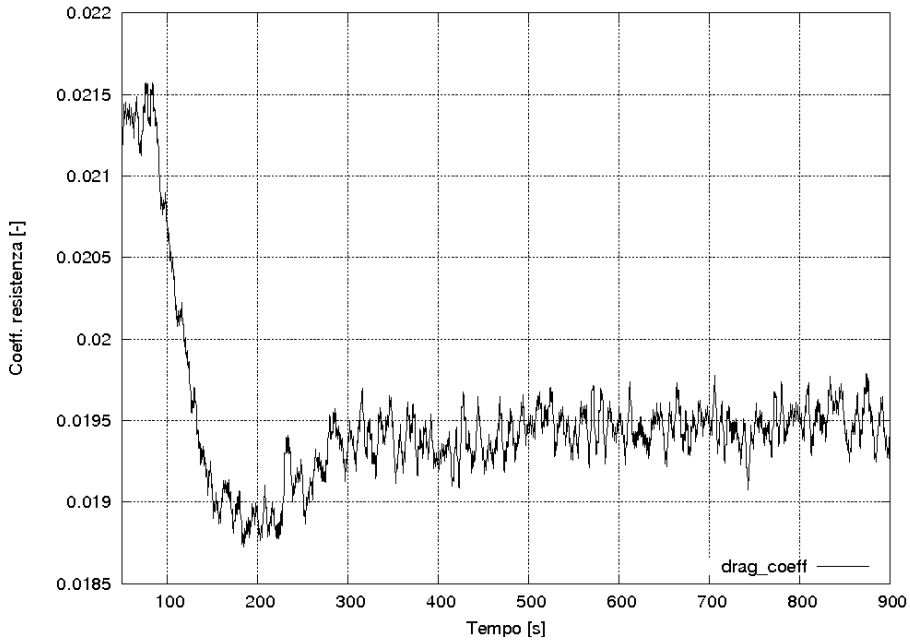

Figura 6.8: Andamento del coefficiente di resistenza per la configurazione di riferimento a 4° di incidenza e impostazioni di tipo di solutore delle equazioni coerenti con quelle di *Star-ccm+*. I passi di iterazione sono identificabili con i secondi della simulazione.

passi dopo raggiunta convergenza, dato che i valori variano tra loro alla terza cifra decimale per la portanza e alla quarta per la resistenza.

Questo tipo di impostazione è stato utilizzato per il calcolo della polare, dato il migliore comportamento complessivo.

6.2.3 Calcolo della polare

Per il calcolo della polare è stato risolto il medesimo problema fisico, però con angoli di incidenza appartenenti ad un intervallo che va da 0° a 16°, con passo 2°.

Per ogni variazione d'inclinazione, si è generata una cartella del caso con le stesse impostazioni del primo studio, a parte la correzione del file della velocità U, all'interno di *0*; per cui le tre componenti sono state periodicamente aggiornate a seconda delle condizioni di flusso; lo stesso approccio è stato adottato per le direzioni lungo cui si calcolano i coefficienti di forza, nella sezione delle funzioni del documento *controlDict* in *system*.

Per fare un esempio, per incidenza nulla, il nuovo file U sarà:

†
FoamFile
{

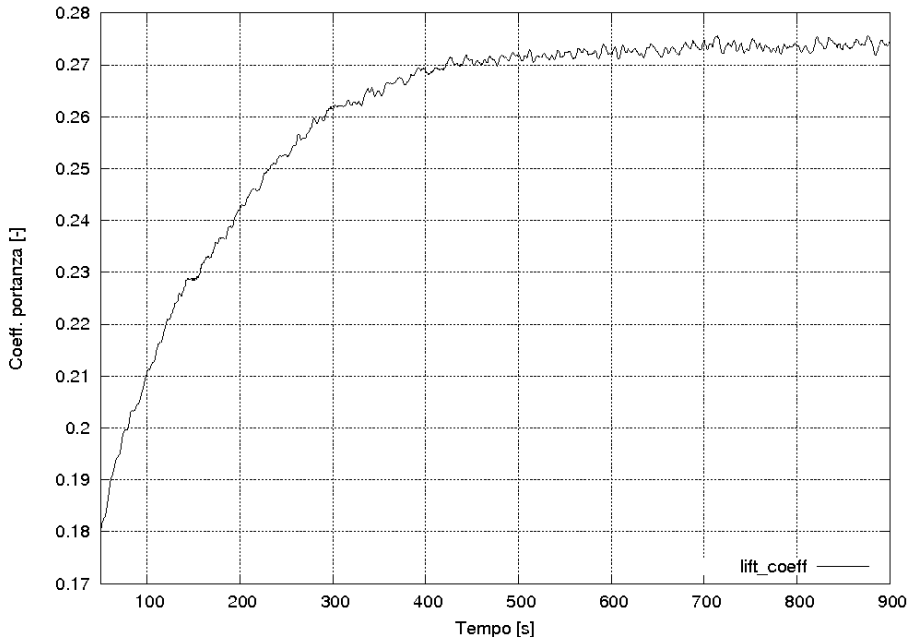

Figura 6.9: Andamento del coefficiente di portanza per la configurazione di riferimento a 4° di incidenza e impostazioni di tipo di solutore delle equazioni coerenti con quelle di *Star-ccm*$_+$. I passi di iterazione sono identificabili con i secondi della simulazione.

```
version 2.0;
format ascii;
class volVectorField;
object U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions [0 1 -1 0 0 0 0];
internalField uniform (3 0 0);
boundaryField
{
        inlet
                {
                type freestream;
                freestreamValue uniform (3 0 0);
                }
        outlet
                {
                type freestream;
                freestreamValue uniform (3 0 0);
                }
```

```
        ''deriva_solid''
                {
                type fixedValue;
                value uniform (0 0 0);
                }
        box
                {
                type freestream;
                freestreamValue uniform (3 0 0);
                }
}
// ************************************************************************* //
```

†

Mentre il nuovo file *controlDict* sarà:

†

```
FoamFile
{
version 2.0;
format ascii;
class dictionary;
location ''system '';
object controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application simpleFoam;
startFrom latestTime;
startTime 0;
stopAt endTime;
endTime 2000;
deltaT 1;
writeControl timeStep;
writeInterval 50;
purgeWrite 0;
writeFormat ascii;
writePrecision 6;
writeCompression uncompressed;
timeFormat general;
timePrecision 6;
runTimeModifiable yes;
functions
{
forces
```

```
{
type forces;
functionObjectLibs (libforces.so); //Libreria da caricare
patches (deriva_solid); // Nome della griglia della deriva
rhoName rhoInf;
rhoInf 997.561; //Densità del fluido di riferimento
CofR (0 0 0); //Origine per il calcolo del momento
outputControl timeStep;
outputInterval 1;
}
forceCoeffs
{
// rhoInf - Densità di riferimento
// CofR - Centro di rotazione
// dragDir - Direzione del coefficiente di resistenza
// liftDir - Direzione del coefficiente di portanza
// pitchAxis - Asse momento di beccheggio
// magUinf - Valore della velocità di riferimento
// lRef - Lunghezza di riferimento
// Aref - Area di riferimento
type forceCoeffs;
functionObjectLibs (libforces.so);
patches (deriva_solid);
rhoName rhoInf;
rhoInf 997.561;
CofR (0 0 0);
liftDir (0 1 0);
dragDir (1 0 0);
pitchAxis (0 0 1);
magUInf 3;
lRef 1.26;
Aref 2.038;
outputControl timeStep;
outputInterval 1;
}
}
// ************************************************************************* //
†
```

Si vuole precisare in questa sede che solitamente, nel file che controlla la gestione del calcolo (*controlDict*), si è preferito lasciare come termine del processo un tempo pari a 2000 secondi, ovvero si consentono 2000 iterazioni per raggiungere un risultato

stabile. È chiaro però che, monitorando i residui e i valori dei coefficienti di portanza e di resistenza, si possa interrompere l'applicazione prima del tempo prestabilito. Questa scelta è stata fatta poiché, col crescere dell'incidenza, i casi impegano un maggior numero di passi per arrivare a convergenza; ciò è stato attribuito al fatto che, avvicinandosi a condizioni di stallo, si abbia bisogno di una più lunga fase di calcolo per rappresentare problemi fisici sempre più complessi.

Si vuole anche aprire una breve parentesi sull'utilizzo del programma. Se, nel momento in cui si voglia monitorare l'andamento della soluzione tramite alcune routine generate per gnuplot, si abbiano problemi di dipendenza, si può adottare la seguente soluzione: si commenta il tratto *. $HOME/OpenFOAM/OpenFOAM-1.6/etc/bashrc* alla fine del file .bashrc della /home, si aggiornano le variabili di ambiente con il comando *. $HOME/.bashrc* e, nel terminale in cui si vuol far eseguire OpenFOAM, è sufficiente scrivere la frase prima commentata.

Fatte le premesse necessarie, si riportano nelle tabelle seguenti i valori ottenuti del coefficiente di portanza, di resistenza e di efficienza per le varie incidenze, sia con OpenFOAM che con *Star-ccm+*.

Angolo di incidenza	C_L		C_D	
	Star-ccm+	OpenFOAM	*Star-ccm+*	OpenFOAM
0	0.000	0.008	0.0109	0.0196
2	0.122	0.142	0.0124	0.0126
4	0.245	0.273	0.0173	0.0195
6	0.367	0.411	0.0249	0.0295
8	0.491	0.551	0.0360	0.0427
10	0.609	0.676	0.0498	0.0608
12	0.727	0.789	0.0667	0.0812
14	0.841	0.905	0.0862	0.1063
16	0.946	0.997	0.1074	0.1349

Angolo di incidenza	E	
	Star-ccm+	OpenFOAM
0	0	0.42
2	9.84	11.21
4	14.16	14.04
6	14.74	13.91
8	13.64	12.88
10	12.23	11.11
12	10.9	9.72
14	9.76	8.51
16	8.81	7.39

Usando i dati appena riportati, è stato possibile tracciare sia la polare della deriva (fig. 6.10) che il grafico $C_L - \alpha$ (fig. 6.11) per i due studi.

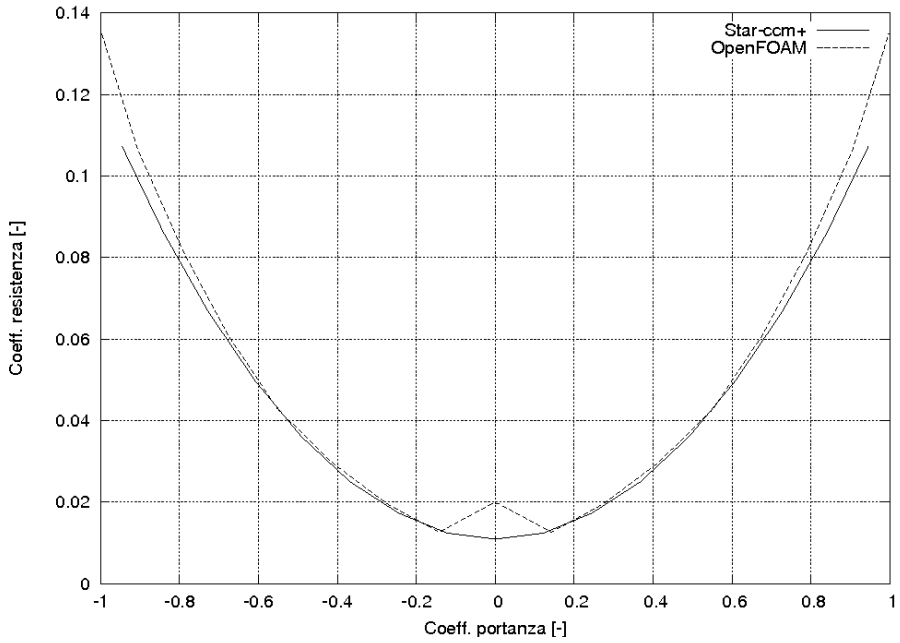

Figura 6.10: Confronto tra le polari della deriva dei due programmi.

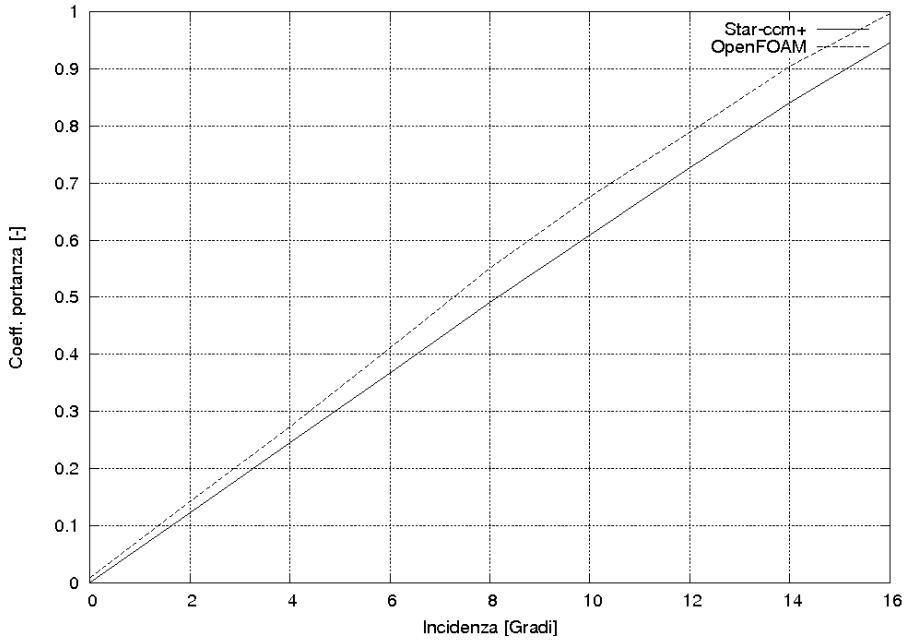

Figura 6.11: Confronto tra le curve $C_L - \alpha$ della deriva dei due programmi.

Analizzando l'immagine della polare, si nota che per un certo intervallo di C_L, che va circa da 0.15 a 0.7, le due curve sono molto simili, quindi indicativamente

OpenFOAM fornisce dei risultati validi per angoli di incidenza da 2° a 10°.

Se invece si raffrontato i valori delle efficienze, si può affermare che, dai 4° in poi, i risultati ottenuti con il software libero è come se anticipassero le stime di *Star-ccm+* di un passo di calcolo, dando quindi degli stati fisici molto legati tra loro.

Per quanto riguarda le curve del $C_L - \alpha$, si nota che le due pendenze sono diverse. Si è voluto condurre allora un studio per i due comportamenti. Si è calcolato il $C_{L\alpha}$ medio per entrambi, prendendo i valori fra 2° e 14°, partendo dal presupposto che l'incidenza a 16° fosse già fuori dal rapporto di linearità. Per *Star-ccm+* il $C_L\alpha$ è pari a 0.0609, mentre per OpenFOAM è pari a 0.06779. Dato che in teoria la pendenza della curva dovrebbe essere costante, si è stimato di quanto si discostano i valori per ogni incidenza da quello medio. Dalla figura 6.12 è evidente che il comportamento ottenuto da OpenFOAM sia meno attinente alla teoria rispetto a quello di *Star-ccm+*.

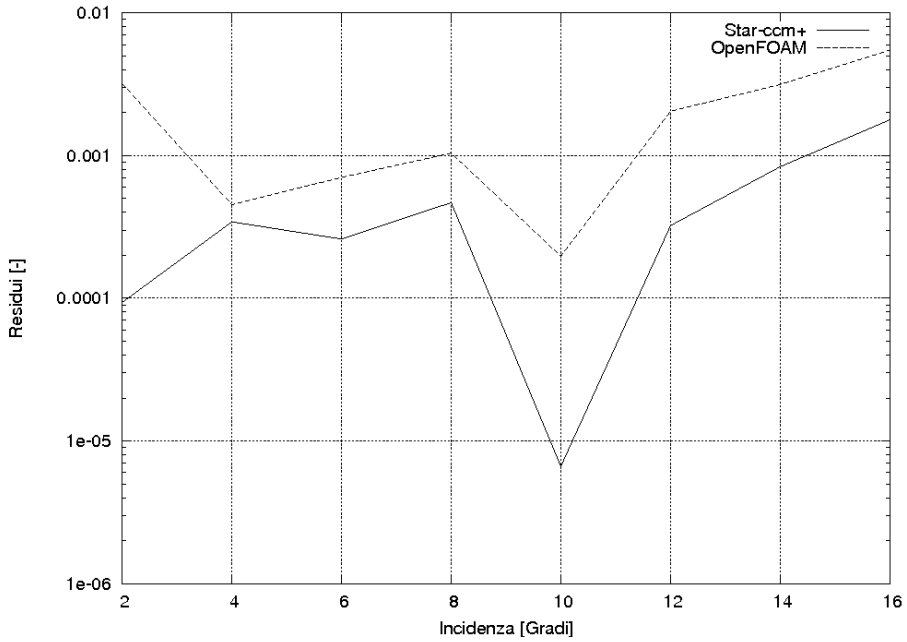

Figura 6.12: Grafico delle differenza tra i valori assunti dal $C_{L\alpha}$ per ogni incidenza e il suo valore medio per entrambi i programmi.

Anche se complessivamente OpenFOAM fornisce un risultato piuttosto affidabile, soprattutto per incidenze intermedie, ciò che però è poco spiegabile è il comportamento per $\alpha = 0°$; infatti, per incidenza nulla, non è assolutamente accettabile un valore talmente diverso da zero del coefficiente di portanza e uno di resistenza simile a quello ottenuto per $\alpha = 4°$. Si è tentato di capire se ciò potesse essere dovuto ad una griglia poco raffinata attorno al bordo del corpo e quindi poco rappresentativa dello strato limite (ipotesi avanzata anche per motivare la differenza, in generale, tra i due programmi). Nello studio di sensibilità verranno riportati i cambiamenti sulla griglia, nel tentativo di dimostrare tale ipotesi.

Altra motivazione potrebbe essere legata ad un'eccessiva energia di turbolenza, ovvero all'impossibilità di simulare la presenza di strato limite laminare nella prima parte del profilo, soprattutto per incidenza nulla, dal momento in cui si schematizza il flusso completamente turbolento. Questo tipo di influenza giustificherebbe il coefficiente di resistenza elevato per $\alpha = 0$ e una sovrastima per la portanza con l'aumento dell'inclinazione.

In effetti, è stato notato che, se per *Star-ccm+* il C_P è 0.92 nel punto di ristagno per incidenza nulla (in teoria dovrebbe essere pari a 1), con OpenFOAM è vistosamente maggiore dell'unità, con un valore massimo di 1.91 raggiunto alla radice della deriva. Andando a studiare una sezione trasversale posta a 0.3 metri dal piano superiore del dominio, per le incidenze a 0°, 4° e 8°, si è visto che il coefficiente diminuisce, passando attraverso i valori: 1.74, 1.08 e 1.09.

Si può dunque dire che un'eccessiva energia di turbolenza, rispetto al caso reale, potrebbe influire sugli errori accumulati nei risultati.

6.2.4 Studi di sensibilità

Cambiamenti sulla griglia

Lo studio di sensibilità sulla griglia è diviso in due parti: nella prima si espongono le prove effettuate per motivare valori dei coefficienti di forza discordanti da quelli attesi, nella seconda si riportano alcune considerazioni sulla generazione della griglia con l'utilizzo di un numero differente di processori.

In primo luogo si sono approntati alcuni cambiamenti alla raffinatezza del reticolo attorno la geometria della deriva, nel tentativo di rappresentare meglio lo strato limite. Si è modificato il file *snappyHexMeshDict*, che controlla la modellizzazione della griglia attorno al corpo. Più precisamente, sono stati cambiati la zona di infittimento e il numero di celle per cui termina il processo, attraverso le entrate:

```
†
refinementBox
{ type searchableBox;
min (-0.3 -0.3 -2.4);
max ( 3 0.3 0);
}
†
e
†
maxGlobalCells 3000000;
†
```

In questo modo si restringe il campo di ricerca e si aumentano le celle presenti, globalmente la griglia risulta più raffinata in prossimità del bordo (complessivamente vi sono cinque milioni di celle).

Nonostante ciò, i risultati ottenuti non sono quelli sperati, essendo molto simili,

se non peggiori, a quelli precedenti:
$C_D = 0.020$ e $C_L = 0.0081$
Si è cercata una conferma tramite un caso con stessa griglia ma incidenza pari a 4°; anche in questo caso l'infittimento è risultato inutile per lo scopo, dato che i valori sono ancora simili ai precedenti:
$C_L = 0.277$ e $C_D = 0.0197$.
Per vedere se in qualche modo si fosse riusciti ad ottenere una variazione dipendente dalla griglia, si è ripreso il caso ad incidenza nulla e si è schematizzato il dominio con un numero di celle complessive di circa quattro milioni, ma questa volta distribuite su un campo di infittimento che comprendesse una più vasta sezione di scia, operando sempre sul file *snappyHexMeshDict*, nelle entrate:

```
†
refinementBox
{ type searchableBox;
min (-0.3 -0.3 -2.4);
max ( 8 0.3 0);
}
†
e
†
maxGlobalCells 3000000;
†
```

Ancora una volta i valori ottenuti non sono quelli sperati, dato che:
$C_D = 0.020$ e $C_L = 0.008$
Traendo le ovvie conclusioni dall'analisi, si deve affermare che i valori non consoni alle previsioni non possono essere attribuiti alla qualità della griglia.

Il secondo tipo di studio di sensibilità è stato condotto partendo dall'osservazione della presenza di alcuni difetti nella mesh (fig. 6.13), visibili nelle sezioni longitudinali del campo.

Come primo passo, si è voluto accertare se questi potessero essere legati ad un passaggio di informazioni scorretto nella divisione tra processori. Si è così confrontato il risultato di una mesh fatta da due processori e da quattro e, pur essendo sempre presenti delle incoerenze nel reticolo, esse cambiano da un caso all'altro. Per generare il calcolo tra quattro processori è stato sufficiente decomporre il dominio tramite il comando *decomposePar*, che esegue il file *decomposeParDict* in *system*:

```
†
FoamFile
{
version 2.0;
format ascii;
```

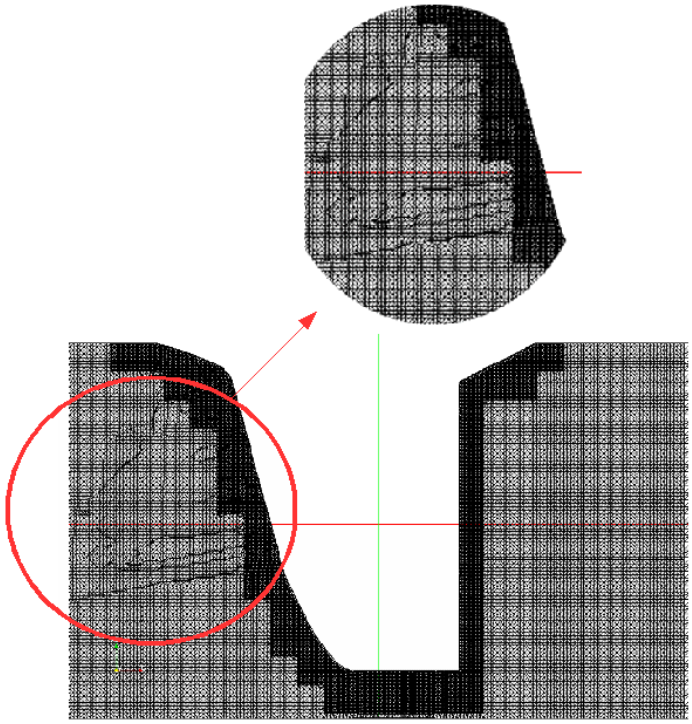

Figura 6.13: Particolare della sezione longitudinale del campo in prossimità della deriva, nel cerchio rosso sono evidenziati alcuni difetti del reticolo, a loro volta ingranditi nell'immagine più in alto.

```
class dictionary;
note "mesh decomposition control dictionary ";
location "system";
object decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
numberOfSubdomains 4;
//Numero dei processori
method hierarchical;
hierarchicalCoeffs
{
n (2 2 1);
delta 0.001;
order xyz;
}
distributed no;
//*********************************************************** //
```

†

Nelle immagini 6.14 si possono vedere alcuni particolari della mesh a confronto, a sinistra quella generata con quattro, a destra quella creata tramite due processori.

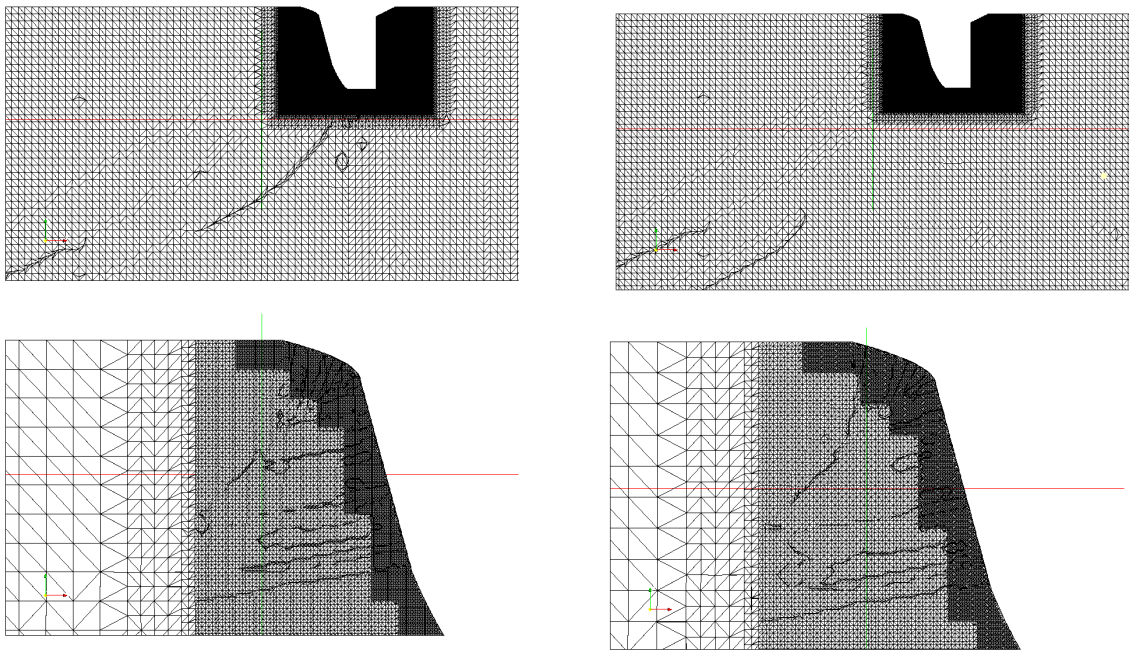

Figura 6.14: Particolari dei difetti nelle mesh a confronto: a sinistra è riportata la griglia generata con quattro processori, mentre a destra quella derivante dal calcolo con due. Le disomogeneità sono in entrambi i casi presenti, ma con caratteristiche ed andamenti diversi.

Per scrupolo è stata fatta una prova anche con un solo processore, ma anche in questo caso si sono riproposti i medesimi problemi.

Seppur il numero di processori possa influenzare il tipo di difetto, è del tutto legittimo avanzare l'ipotesi che tali deformazioni siano attribuibili ad una scorretta gestione della griglia nel momento in cui lo strumento *snappyHexMesh* affronta la fase di aggiunta di strati superficiali di celle. È anche vero che quest'ultima manipolazione è difficilmente evitabile dato che consente una rappresentazione continua ed accurata della geometria della deriva.

Cambiamento dei solutori

Come dimostrato in precedenza, agire sul tipo di solutori e sui fattori di rilassamento influisce fortemente sul comportamento della convergenza del calcolo.

Cambiamento dei metodi di discretizzazione

Per verificare la sensibilità ai metodi di discretizzazione dei termini matematici presenti nelle equazioni, si è considerato il caso ad incidenza a 4°, con le impostazioni simili a quelle utilizzate per calcolare la polare, tranne per il file *fvScheme*, per cui sono stati fatti alcuni cambiamenti, come si può notare a seguire:

```
†
FoamFile
{
version 2.0;
format ascii;
class dictionary;
location ''system '';
object fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
      default steadyState;
}
gradSchemes
{
      default Gauss linear;
}
divSchemes
{
       default Gauss linearUpwind Gauss linear;
       div(phi,U) Gauss linearUpwindV Gauss linear;
       div(phi,k) Gauss upwind;
       div(phi,epsilon) Gauss upwind;
       div(phi,R) Gauss upwind;
       div(R) Gauss linear;
       div(phi,nuTilda) Gauss upwind;
       div((nuEff*dev(grad(U).T()))) Gauss linear;
}
laplacianSchemes
{
       default Gauss linear corrected;
}
interpolationSchemes
{
       default linear;
       interpolate(U) linear;
```

```
}
snGradSchemes
{
        default corrected;
}
fluxRequired
{
        default no;
        p ;
}
//********************************************************************** //
†
```

Nello specifico il file è stato riprodotto sulla base di quelli utilizzati nella sezione dei *tutorials* dedicata ai casi incomprimibili e completamente turbolenti; si sono intenzionalmente omesse tutte le limitazioni che, come detto nel capitolo precedente, erano state consigliate da alcuni utenti per omologare i risultati di OpenFOAM a quelli di Fluent.

Operando questi cambiamenti, si è però notato che si incorreva in problemi di divergenza della soluzione, è stato necessario modificare in parte anche il file *fvSolution* per stabilizzare il risultato, che comunque ha impiegato un maggior numero di iterazioni per arrivare ad asindoto orizzontale. Di seguito si riportano le correzioni eseguite sui fattori di rilassamento:

```
†
relaxationFactors
{
default 0;
p 0.3;
U 0.7;
k 0.3;
epsilon 0.3;
nuTilda 0.7;
}
†
```

I risultati così ottenuti si discostano delle stime precedentemente condotte, essendo:
$C_L = 0.262$, $C_D = 0.0221$ e l'efficienza pari a $E = 11.83$
Rispetto allo stesso caso affrontato con *Star-ccm*$_+$, il coefficiente di resistenza è molto più alto mentre quello di portanza vi si avvicina, quindi globalmente l'efficienza si allontana dal valore di riferimento di 14.

Questo test è stato utile per evidenziare la forte dipendenza della soluzione finale

dai modelli matematici adottati.

Capitolo 7

Conclusioni

Al termine dello studio si sono acquisite esperienze di utilizzo sufficienti per trarre alcune conclusioni sul comportamento di OpenFOAM.

Sia i vantaggi che gli svantaggi principali scaturiscono dalla natura stessa del programma, ovvero dal fatto che sia aperto e libero. Si tenta nel seguito di spiegare meglio questo punto di vista.

Il primo lato positivo è evidentemente la gratuità del prodotto, reperibile dal sito ufficiale; un altro è legato al fatto che si interfaccia facilmente con programmi sia sotto licenza GPL che non. Infatti, da un lato, appartenendo al mondo free-software, offre un collegamento rapido ad altri strumenti della stessa natura, come ad esempio gnuplot, utile per tracciare i grafici in tempo reale, o ParaView, fondamentale per la trattazione dei risultati. Dall'altro lato, dovendo necessariamente confrontarsi ed interagire con programmi non liberi ma largamente usati in ingegneria, offre convertitori di file per agevolare il lavoro agli utenti che vogliono comunque usare altri software, come ad esempio nel caso della griglia di calcolo. Non presenta quindi alcun aspetto di chiusura nei confronti di strumenti commerciali anche se agevola l'uso di quelli liberi.

Il secondo lato positivo deriva dalla sua natura aperta, poiché tutti i file non solo sono leggibili, ma anche di facile ed immediata comprensione, anche per coloro che hanno un livello base di conoscenza in campo di programmazione.

Altro vantaggio di OpenFOAM è la vasta quantità di informazioni a disposizione e facilmente ottenibili; infatti vengono fornite guide per l'utilizzatore e per il programmatore, esempi per ogni tipo di applicazione presente, un forum di utenti molto vivo e in continuo scambio di idee e soluzioni.

È comprensibile quindi come sia naturale la possibilità di collaborazione tra e in gruppi di lavoro, per sviluppare parti di software adatti a un certo studio, non dimenticando anche la presenza di una società a cui poter far riferimento per eventuali commissioni, la OpenCFD Ltd.

Proprio da quest'ultimo vantaggio, si prende spunto per indicare gli aspetti negativi del sofware. Infatti, se da un lato è possibile metter mano completamente ai

file del programma, dall'altro si richiede una perfetta conoscenza dei vari input per far girare un caso. Si deve avere quindi alta professionalità ed esperienza, entrambe le caratteristiche però si ottengono con dispendio di tempo e lavoro, con conseguente ricaduta economica implicita.

Non si può assolutamente trascurare questo aspetto, perché, come visto dai risultati, cambiamenti sui solutori e sui tipi di metodi di discretizzazione possono incidere fortemente sulla convergenza del calcolo e sul risultato fisico finale.

L'utente, che ha intenzione di usare OpenFOAM per scopi lavorativi, non può non tenere conto dello stato della propria preparazione, quindi si giunge alla necessità di più persone che si dedicano allo studio, ognuna delle quali ha competenze specifiche nel campo della fludodinamica o del computazionale, possibilmente seguite da un supervisore con esperienza di simulazione al computer.

Se questo non è possibile, si deve allora prevedere un periodo di ricerca sul forum degli utilizzatori, uno di prove per vedere la risposta del programma e un caso con cui confrontarsi, ovvero il percorso seguito per condurre questa tesi.

Detto questo, si vuole concludere la trattazione sottolineando le grandi potenzialità di OpenFOAM, che, nato nel 2004, è riuscito in pochi anni ad offrire un'alternativa seria e professionale a programmi commerciali. L'aspetto di maggior interesse è la semplicità di parallelizzazione del processo, dato che è il software stesso a gestire l'interfaccia tra utente e divisione tra più processori. Soprattutto per quanto riguarda la mesh, si auspica un forte sviluppo dei mezzi per crearne di complesse con lo strumento della parallelizzazione; infatti, contestualmente a questa tesi, si sono avute alcune difficoltà nella gestione del processo di infittimento, essendo poco intuitivo e un po' acerbo nella modellizzazione di geometrie accurate.

Bibliografia

[1] *http://www.openfoam.com/*

[2] *http://www.paraview.org/*

[3] *http://www.openfoam.com/docs/README.php*

[4] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 101-102, 24th July 2009.

[5] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 102-108, 24th July 2009.

[6] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 132-133, 24th July 2009.

[7] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pag. 1130, 24th July 2009.

[8] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 140-148, 24th July 2009.

[9] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 132-140, 24th July 2009.

[10] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 142-143, 24th July 2009.

[11] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp 128-133, 24th July 2009.

[12] *http://www.cfd-online.com/Forums/openfoam/67168-stl-snappyhexmesh-dimensions-mismatch.html*

[13] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pag. 142, 24th July 2009.

[14] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pag. 147, 24th July 2009.

[15] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 81-85, 24th July 2009.

[16] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 85-86, 24th July 2009.

[17] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pag. 108, 24th July 2009.

[18] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 108-110, 24th July 2009.

[19] *http://www.cfd-online.com/Forums/openfoam/67984-forces-v1-6-a.html*

[20] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pag. 111, 24th July 2009.

[21] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 110-117, 24th July 2009.

[22] *http://www.cfd-online.com/Forums/openfoam-solving/57870-wing-aerodynamics-fluent-15-comparison.html*

[23] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pag. 112, 24th July 2009.

[24] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 117-121, 24th July 2009.

[25] *OpenFOAM. The Open Source CFD ToolBox. User Guide*, Version 1.6, pp. 148-155, 24th July 2009.

Ringraziamenti

Giunta al termine della tesi, vorrei ringraziare il professor Lombardi, che mi ha appoggiato in ogni scelta presa sin dall'inizio del lavoro. Senza i suoi aiuti nel risolvere i problemi di interpretazione dei dati e di analisi dei risultati, non sarei stata in grado di dare un'impronta così pratica ed interessante all'utilizzo del programma.

Fondamentali per il lavoro sono stati i dati con cui fare il confronto con *Star-ccm*$_+$ e quindi un ringraziamento va ad Alessandro Mariotti che mi ha fornito tutte le informazioni necessarie e si è sempre dimostrato molto disponibile.

Guardando alla mia carriera universitaria, voglio ringraziare i miei genitori per il sostegno negli anni, non solo da studentessa di ingegneria ma anche e soprattutto di liceo, e mia sorella per avermi fatto vedere i lati positivi e umoristici dei problemi, ridimensionando sempre tutto.

Un grazie particolare va a Gipo che mi ha dato assistenza tecnologica e mentale, supporto nei momenti più difficili e valvola di sfogo strappandomi sempre un sorriso.

Infine ringrazio alcuni amici che mi hanno accompagnato in questi anni: Vjola, con cui ho condiviso tutto il percorso universitario e tutti i suoi alti e bassi, Sara "la Bolle ", che è da una vita che ascolta le mie paturnie; le mie storiche coinquiline Sara e Valentina che mi hanno sopportato per quattro anni; i miei compagni di progetto Gabriele e Lorenzo; e poi Sofia, Goran, Eliana, il gruppo di Euroavia e del Kung-Fu.

Un grazie generale va a tutte quelle persone che negli hanni mi hanno consigliato su esami, imprestato appunti e aspettato con me ai ricevimenti.