ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

CORSO DI LAUREA IN INGEGNERIA AEROSPAZIALE
CLASSE L9

CAMPUS DI FORLÌ

INTERFACCIA GRAFICA OPEN-SOURCE PER IL SET-UP DI MESH E SOLVER CFD PER OPENFOAM

Tesi di laurea in Laboratorio di Aerodinamica Sperimentale

Relatore

Prof. Guglielmo Minelli

Presentata da

Emanuele Rossi

Sessione Dicembre 2025

Anno Accademico 2024/2025

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

CORSO DI LAUREA IN INGEGNERIA AEROSPAZIALE
CLASSE L9

CAMPUS DI FORLÌ

INTERFACCIA GRAFICA OPEN-SOURCE PER IL SET-UP DI MESH E SOLVER CFD PER OPENFOAM

Tesi di laurea in Laboratorio di Aerodinamica Sperimentale

Relatore

Prof. Guglielmo Minelli

Presentata da

Emanuele Rossi

Sessione Dicembre 2025

Anno Accademico 2024/2025

Al me stesso di qualche anno fa, che non immaginava l'incredibile viaggio che lo attendeva.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione e obiettivi

Nell'ambito della fluidodinamica computazionale (CFD - *Computational Fluid Dynamics*), `OpenFOAM` è uno degli strumenti open–source più diffusi per la simulazione di flussi complessi in domini tridimensionali e bidimensionali. La grande flessibilità del codice, basato su dizionari di testo e su una struttura modulare, permette di descrivere una vasta gamma di problemi industriali e accademici. D'altro canto, questa stessa flessibilità va di pari passo con una curva di apprendimento ripida: l'utente deve familiarizzare con numerosi file di configurazione (`blockMeshDict`, `snappyHexMeshDict`, `fvSchemes`, `fvSolution`, `controlDict`, ecc.) e con una sequenza di comandi `CLI` da eseguire nell'ordine corretto.

Il lavoro presentato in questa tesi nasce proprio dall'esigenza di ridurre tale barriera d'ingresso, fornendo un sistema di interfacce grafiche (*GUI*) che guidino l'utente attraverso le fasi principali di preparazione del caso: import e orientamento della geometria, generazione della mesh, scelta del modello di turbolenza, configurazione del solver ed esecuzione locale (o, in prospettiva, da remoto) della simulazione.

1.1 Motivazioni e contesto

L'utilizzo di `OpenFOAM` richiede spesso una combinazione di editing (modifica) manuale di numerosi dizionari di testo, lancio di utility e solver da riga di comando e controlli iterativi sulla qualità della mesh e sulla stabilità numerica del caso.

Questa modalità di lavoro è molto flessibile per l'utente esperto, ma può essere fonte di errori ripetitivi (copie errate di file, refusi nei nomi dei campi, incoerenze tra modello di turbolenza e solver, ecc.) e rende difficile la diffusione del codice presso utenti meno abituati all'interazione tramite linea di comando.

L'obiettivo del sistema di *GUI* sviluppato in questo lavoro è:

- automatizzare la creazione della cartella di un caso di simulazione a partire da un template, riducendo al minimo le modifiche manuali dei dizionari;
- rendere esplicito, attraverso schermate dedicate, il *workflow* tipico di preparazione di un caso CFD con `OpenFOAM`;

- rendere più accessibili alcune funzionalità avanzate (*boundary layers*, preset di qualità, monitoraggio dei log) che da linea di comando richiederebbero una conoscenza dettagliata della sintassi dei file di input.

Il risultato è una pipeline guidata che mantiene la trasparenza tipica dell'ambiente open–source (i file generati sono normali casi `OpenFOAM`, totalmente ispezionabili e modificabili), ma ne semplifica l'uso nelle fasi più ripetitive e soggette a errore.

1.2 Lavori esistenti e contributo precedente

Nel corso degli anni sono stati sviluppati diversi strumenti grafici a supporto di `OpenFOAM`, sia commerciali sia open–source. In generale questi strumenti si concentrano su una o più delle seguenti fasi: costruzione della geometria e del dominio di calcolo, generazione e raffinamento della mesh, gestione di casi preconfezionati per scenari tipici (flussi interni/esterni standard) oppure post–processing e visualizzazione dei risultati.

Molte di queste interfacce, tuttavia sono orientate a casi standard e risultano poco flessibili quando si vogliono introdurre modifiche non previste ai dizionari oppure sono legate a soluzioni commerciali, non sempre compatibili con un flusso di lavoro interamente open–source [1, 2, 3].

Un ulteriore riferimento per questo lavoro è costituito dal progetto sviluppato durante il tirocinio curriculare [4], in cui è stata implementata una pipeline automatizzata per la sola generazione della mesh. In quella fase il flusso di lavoro era gestito esclusivamente tramite pochi script e riga di comando: la sequenza `blockMesh` → `surfaceFeatureExtract` → `snappyHexMesh` veniva orchestrata da script Python e `bash`, ma senza alcuna interfaccia grafica. L'utente doveva comunque: modificare manualmente i dizionari di mesh, ricordare l'ordine di esecuzione dei comandi e interpretare i log direttamente dal terminale.

La presente tesi estende e generalizza quel lavoro concentrandosi su due aspetti principali:

- l'introduzione di una doppia pipeline *mesh* / *turbolenza–solver*, interamente guidata da GUI;
- la progettazione di schermate dedicate per ciascuna fase (orientamento STL, `snappyHexMesh`, preset di qualità, configurazione della turbolenza, scelta del solver, esecuzione locale/remota), mantenendo al tempo stesso il controllo esplicito sui file `OpenFOAM`.

1.3 Strumenti e tecnologie utilizzate

Un obiettivo dichiarato del progetto è l'uso esclusivo di strumenti open–source, in modo da rendere il flusso di lavoro completamente riproducibile e liberamente estendibile da altri utenti o gruppi di ricerca. Le principali tecnologie impiegate sono:

- **Qt** e **PyQt**: Qt è un toolkit (set di strumenti) multipiattaforma per lo sviluppo di interfacce grafiche, ampiamente utilizzato in ambito scientifico e industriale. `PyQt` fornisce i binding Python a Qt e permette di sfruttare il modello a widget (finestre, pulsanti, caselle di testo, ...) all'interno di Python. Nel sistema sviluppato, tutte le GUI (pipeline di mesh, widget di turbolenza, helper per l'esecuzione remota, anteprima log, ecc.) sono implementate in `PyQt` [5, 6];
- **NumPy**: libreria fondamentale per il calcolo numerico in Python, usata quando è necessario gestire vettori e matrici (ad esempio per semplici operazioni geometriche, trasformazioni di coordinate o manipolazione di dati numerici estratti dai log) [7, 8, 9];
- **Matplotlib** e backend **Agg**: Matplotlib è una libreria per la generazione di grafici 2D open–source. Nel progetto viene utilizzata insieme al backend `Agg` (motore di render off–screen) per generare anteprime grafiche [10, 11];
- **OpenFOAM**: software CFD open–source utilizzato come motore di calcolo. La pipeline non sostituisce `OpenFOAM`, ma ne automatizza la preparazione del caso (creazione della cartella `case/`, scrittura dei dizionari, lancio degli utility e del solver) mantenendo intatta la struttura standard del codice [12];

L'intero sistema è pensato per essere eseguito su piattaforme GNU/Linux, dove `OpenFOAM` e le librerie Python menzionate sono facilmente installabili tramite i normali gestori di pacchetti o ambienti virtuali.

1.4 Limiti di applicabilità e possibili estensioni

Il sistema di GUI sviluppato in questa tesi ha un ambito di applicabilità ben definito, che conviene esplicitare fin dall'inizio. In primo luogo, le funzionalità sono state progettate e testate su una classe di problemi CFD relativamente specifica:

- flussi incomprimibili, simulati con solver della famiglia `pisoFoam` / `pimpleFoam` / `simpleFoam` e `potentialFoam`;
- modelli di turbolenza di tipo RANS, in particolare il modello `kOmegaSST` utilizzato nel caso dimostrativo e, più in generale, modelli compatibili con la struttura dei campi iniziali forniti nel template, e modelli di turbolenza di tipo LES;
- geometrie descritte da file STL importati dall'utente, con mesh generata tramite `blockMesh` e `snappyHexMesh`.

L'estensione a casi comprimibili, a simulazioni multi–fase o a modelli con equazione dell'energia esplicitamente risolta richiederebbe:

- la definizione di nuovi template di caso (campi aggiuntivi in `0/`, proprietà termofisiche più complesse in `constant/`);
- l'introduzione di opzioni specifiche nella GUI (ad esempio per la scelta delle proprietà termiche e dei modelli di trasferimento di calore);

- una serie di test con solver diversi da quelli considerati nel presente lavoro per verificarne l'effettivo e corretto funzionamento.

In secondo luogo, sebbene il codice includa uno strumento in particolare (`remote_run_helper`) per la generazione di script di sincronizzazione e lancio su cluster remoti (via SSH/SLURM), l'attività sperimentale svolta nell'ambito di questa tesi si è concentrata sull'esecuzione in locale. Le funzionalità di "run" remoto devono quindi essere considerate una prima implementazione prototipale, che richiede la validazione su infrastrutture HPC reali. Infine, anche dal punto di vista dell'interfaccia grafica, il sistema è stato pensato come base estendibile: nuove finestre, nuovi preset e nuove pipeline (ad esempio per casi multi–regione o per accoppiamenti CFD–strutturali) possono essere aggiunti riutilizzando la stessa architettura Qt/PyQt e lo stesso approccio basato su template di caso. Questo rende il lavoro non solo un supporto operativo per il caso dimostrativo trattato nella tesi, ma anche un possibile punto di partenza per sviluppi futuri in contesti di ricerca più ampi.

Capitolo 2

Workflow

2.1 Schema generale della pipeline

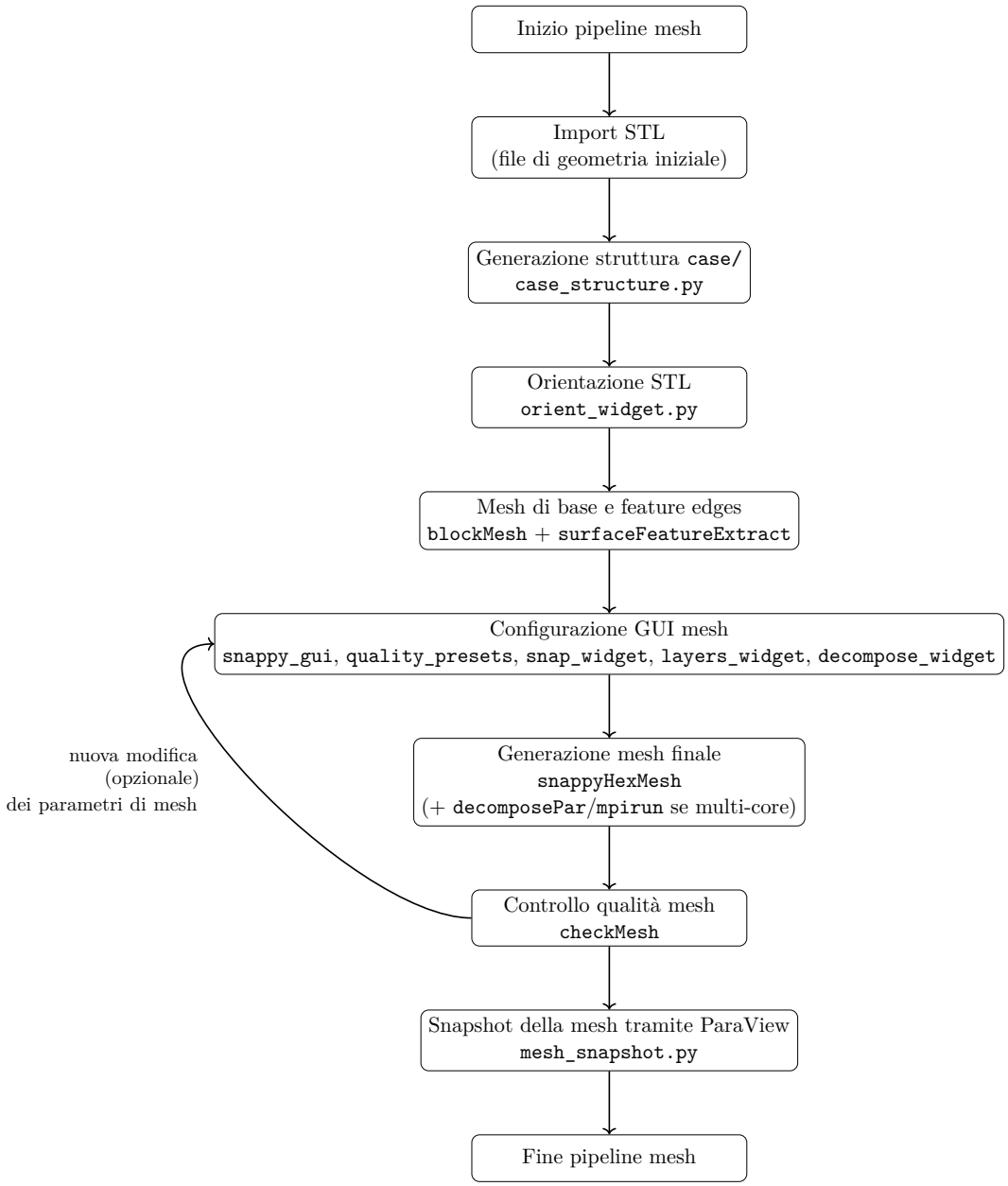


Figura 2.1: Schema a blocchi della pipeline di generazione mesh.

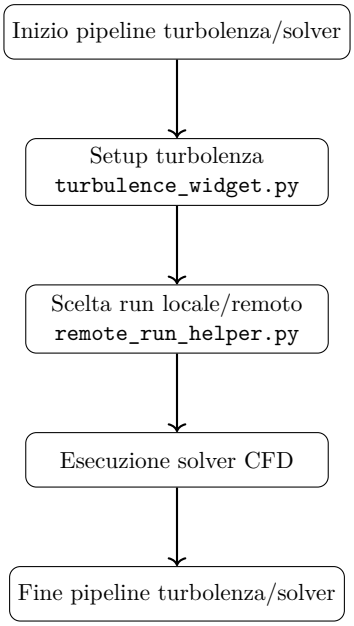


Figura 2.2: Schema a blocchi della pipeline di setup turbolenza ed esecuzione del solver.

In questo capitolo si presenta la struttura del sistema sviluppato per l'automazione della preparazione dei casi `OpenFOAM`. Il codice è organizzato all'interno della directory `src-main/src`, che contiene gli script Python per la generazione della mesh, la configurazione del modello di turbolenza e del solver, oltre a template (modelli) di file `OpenFOAM` che vengono copiati e modificati attraverso gli script stessi.

2.2 Struttura generale del progetto

La directory principale del progetto è organizzata come segue:

- `run.py`: script principale che avvia l'intero processo e permette di scegliere quale pipeline avviare tra quella di generazione della mesh e quella di setup del modello di turbolenza e del solver;
- `clear_all.py`: per pulire il caso di lavoro (rimozione di risultati, log e directory temporanee) e riportarlo a uno stato iniziale;
- `requirements.txt`: elenco delle dipendenze Python e degli strumenti necessari per eseguire l'applicazione;
- `inputSTL/`: cartella che ospita il file di geometria in formato STL fornito dall'utente;
- `mesh/`: contiene i template dei dizionari `OpenFOAM` relativi alla generazione della mesh;
- `scripts/`: cartella principale degli script Python che implementano la logica delle pipeline e delle interfacce grafiche;
- `templateCase/`: contiene i template utilizzati come base per la creazione del caso di lavoro.

2.3 Cartella `inputSTL`

La cartella `inputSTL/` contiene il file STL di ingresso. Nel caso dimostrativo considerato in questa tesi è presente il file `monkey.stl`, che rappresenta la geometria dell'oggetto su cui viene generata la mesh e impostata la simulazione CFD, mostrata in Figura 2.3.

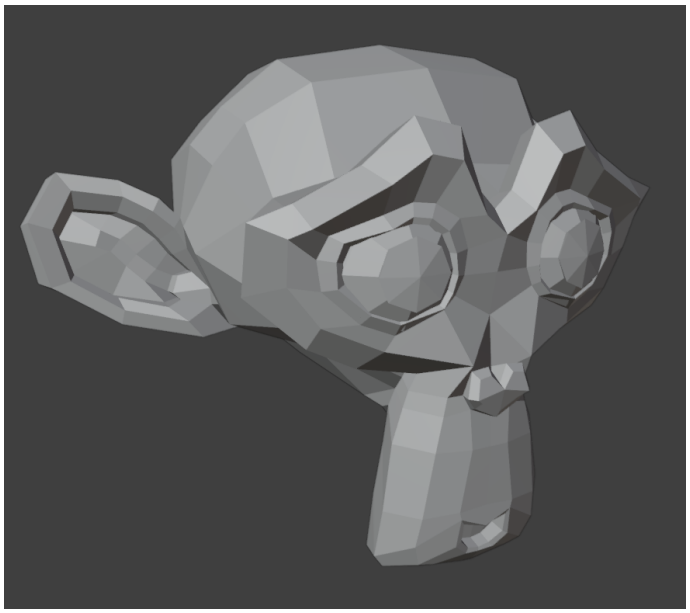


Figura 2.3: Contenuto della cartella `inputSTL/` con il file `monkey.stl`.

L'utente può sostituire questo file con una propria geometria, mantenendo invariata la struttura della directory; all'interno di questa cartella va inserito un solo file STL [13, 14, 15].

2.4 Cartella `mesh`

La cartella `mesh/` contiene i dizionari `OpenFOAM` di base per la generazione della mesh; abbiamo `snappyHexMeshDict`, file principale che controlla le fasi di `castellatedMesh`, `snap` e `addLayers`. Esso viene modificato dagli script della GUI (`snappy_gui.py`, `snap_widget.py`, `layers_widget.py`) in funzione delle scelte dell'utente. Inoltre è presente `surfaceFeatureExtractDict`, dizionario per l'estrazione delle feature edges (spigoli e contorni particolari) a partire dalla geometria STL tramite `surfaceFeatureExtract`. La pipeline copia questi file all'interno della cartella di lavoro e ne aggiorna automaticamente i parametri [16, 17, 18].

2.5 Cartella `scripts`

La cartella `scripts/` contiene gli script Python che implementano le due pipeline e le relative interfacce grafiche:

- `mesh_pipeline_gui.py`: gestisce il flusso di lavoro dalla scelta dell'STL fino al salvataggio della mesh finale;

- `snappy_gui.py`: pannello per la configurazione iniziale di `snappyHexMesh` (attivazione di `castellatedMesh`, `snap`, `addLayers`);
- `orient_widget.py`: GUI per orientare/ruotare e allineare la geometria STL prima della generazione della mesh;
- `snap_widget.py`, `layers_widget.py`, `decompose_widget.py`: widget dedicati, rispettivamente, alla configurazione di `snapControls`, `addLayersControls` e `decomposeParDict`;
- `turbulence_widget.py`: gestisce l'interfaccia per la scelta del tipo di simulazione (ad esempio RANS), del modello di turbolenza (ad esempio Spalart–Allmaras) e del solver compatibile;
- `remote_run_helper.py`: per la scelta tra esecuzione locale e remota via SSH/SLURM, con impostazione di `user@host` e del percorso remoto del caso;
- `preview3d.py` e `mesh_snapshot.py`: script per l'anteprima 3D della geometria a seguito delle scelte effettuate nella finestra gestita da `orient_widget.py` e per il salvataggio di immagini (snapshot) della mesh [10, 11, 19];
- `log_monitor.py`: finestra che visualizza in tempo reale i log di `snappyHexMesh` [20];
- `case_structure.py`: crea e aggiorna la struttura della cartella `case/` copiando i dict contenuti in `templateCase` e generando i file necessari;
- `quality_presets.py`: definisce preset di qualità unificati per `snappyHexMesh` e `checkMesh`, applicandoli ai rispettivi dizionari [21, 22];
- `sanity_check.py` e `ui_theme.py`: script di supporto per il controllo di qualità del file STL e per la definizione del tema grafico della GUI [23, 24, 25].

2.6 Cartella `templateCase`

La cartella `templateCase/` contiene un caso `OpenFOAM` completo, utilizzato come template per tutti i casi generati dalla pipeline. La sua struttura è la seguente:

- `0/`: directory dei campi iniziali (pressione `p`, velocità `U`, variabili di turbolenza `k`, `epsilon`, `omega`, `nuTilda`, `nut`);
- `constant/`: contiene i file fisici, `transportProperties` e `turbulenceProperties`, che definiscono le proprietà del fluido e il modello di turbolenza;
- `system/`: raccoglie i dizionari numerici:
  - `blockMeshDict` per la generazione della mesh di base;
  - `decomposeParDict` per la decomposizione del dominio e l'esecuzione multicore;
  - `fvSchemes` e `fvSolution` per gli schemi numerici e per gli algoritmi dei solver;

- i file `fvSolutionTemplate_PISO`, `_PIMPLE`, `_SIMPLE`, `_POTENTIAL` usati come template per le diverse tipologie di simulazione testate;
- `controlDict` per il controllo della simulazione (tempo di inizio/fine, passo temporale, frequenza di scrittura, nome del solver) [26, 27];
- `meshQualityDict` con i criteri di qualità della mesh utilizzati da `checkMesh` [21, 22, 18].

Durante l'esecuzione della pipeline, il contenuto di `templateCase/` viene copiato in una cartella chiamata `case/` e aggiornato dagli script descritti nel paragrafo precedente, in funzione delle scelte dell'utente su geometria, parametri di mesh, modello di turbolenza e modalità di esecuzione (locale o remota).

2.7 Strumenti per la preparazione del caso

Un elemento comune a tutte le finestre della GUI è il comportamento dei pulsanti. Ogni volta che si preme il pulsante `Apply` oppure `Write` (a seconda del widget in uso) le scelte correnti vengono salvate e trascritte nei corrispondenti file di `OpenFOAM` all'interno della cartella `case/`, nel formato sintattico corretto (graffe, punti e virgola, header `FoamFile`, ecc.). In questo modo si cerca di limitare al massimo la modifica dei file a mano da parte dell'utente: è la GUI che si occupa di aggiornare i dict con una struttura coerente con quella di `OpenFOAM`.

In tutte le finestre, il pulsante `Continue` ha invece il solo effetto di chiudere il widget corrente e di proseguire alla fase successiva della pipeline, senza applicare ulteriori modifiche ai file oltre a quelle già confermate con `Apply`/`Write`.

2.7.1 Avvio della pipeline, preparazione del caso e orientamento STL

All'avvio l'utente importa la geometria di ingresso in formato STL e viene guidato nella preparazione del caso attraverso una serie di passi assistiti. La prima operazione consiste nel caricamento dell'STL e nella definizione di un sistema di riferimento coerente con il dominio di calcolo [28, 29, 30, 31].

La GUI mostrata in Figura 2.4 consente di ruotare il modello rispetto agli assi principali del dominio numerico. In questa fase l'utente può riallineare la geometria agli assi x–y–z del canale di prova e correggere eventuali rotazioni indesiderate ereditate dal CAD [32, 33, 34, 35];

Una volta definito l'orientamento, cliccando sul pulsante `Apply_Save STL` viene generato un file STL "orientato" che verrà utilizzato nelle fasi successive della pipeline (generazione della mesh con `blockMesh` e `snappyHexMesh`). Sempre in Figura 2.4 si vede a destra l'anteprima 3D della geometria all'interno del dominio: questa vista è pensata come controllo qualitativo immediato, per verificare che l'orientamento dell'STL sia quello desiderato prima di procedere alla fase di generazione vera e propria della mesh [13, 14, 10, 11].

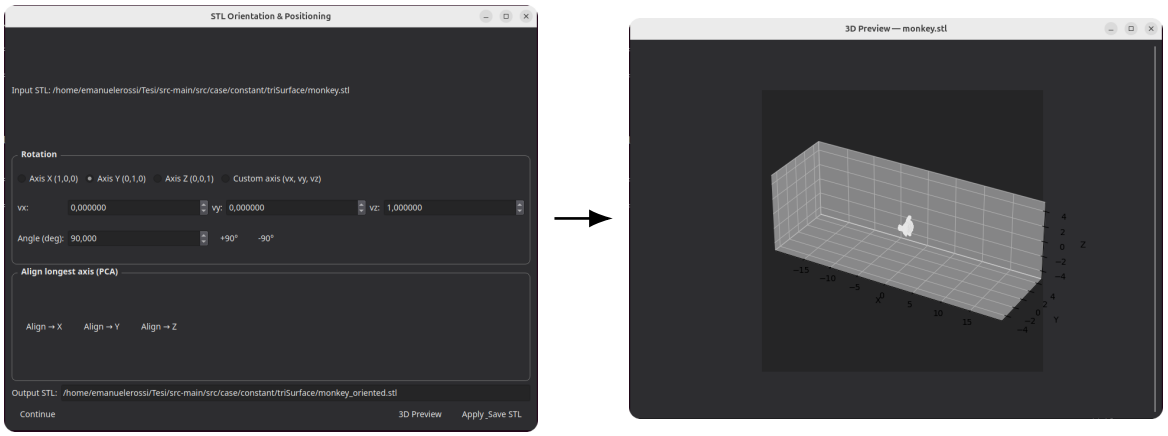


Figura 2.4: Orientamento e anteprima STL: a sinistra l'interfaccia per ruotare/riallineare la geometria; a destra l'anteprima 3D nel dominio disponibile cliccando il pulsante "3D Preview" .

Già in questa prima finestra, premendo `Apply_Save STL` le scelte di rotazione e orientamento vengono salvate e il nuovo file STL viene scritto nella struttura del caso; premendo invece `Continue` si chiude la finestra e si passa allo step successivo.

2.7.2 Struttura della cartella `case/`

Una volta scelto e orientato l'STL, la pipeline provvede a creare automaticamente una cartella di lavoro `case/`, clonando la struttura del caso template e aggiornando solo i file necessari. In questo modo l'utente non deve copiare manualmente i dizionari `OpenFOAM`, ma si limita a interagire con la GUI.

In sintesi, la cartella `case/` contiene: la directory `0/` con i campi iniziali già pronti, quantomeno in una versione "generale", la directory `constant/` con le proprietà del fluido e i parametri di turbolenza e la directory `system/` con i dizionari numerici (`blockMeshDict`, `snappyHexMeshDict`, `fvSchemes`, `fvSolution`, `decomposeParDict`, `controlDict`, ecc.). Rispetto al caso template, i file presenti in `case/` risultano già personalizzati sulla base della geometria in ingresso (ad esempio dimensioni della mesh di base, regioni di raffinamento, decomposizione del dominio), riducendo il rischio di errori dovuti a modifiche manuali dei dizionari [16, 17, 18].

2.7.3 Editor delle fasi di snappyHexMesh

La fase successiva riguarda la configurazione del processo di generazione della mesh con `snappyHexMesh`. L'editor mostrato in Figura 2.5 fornisce una vista compatta dei tre step principali:

- `castellatedMesh`, che si occupa del primo raffinamento cartesiano del reticolo di base intorno alla geometria;
- `snap`, per l'aggancio delle celle alla superficie STL;

- `addLayers`, fase deputata alla creazione (estrusione) dei cosidetti "layers" vicino alle pareti.

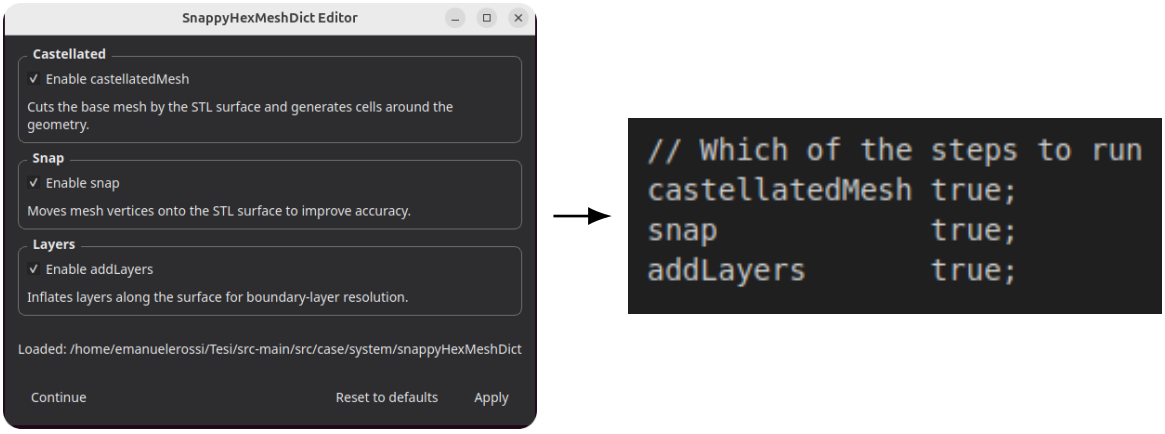


Figura 2.5: Editor di `snappyHexMeshDict`: pannello con le opzioni principali per abilitare `castellatedMesh`, `snap` e `addLayers`.

Nel riquadro principale è presente anche il pulsante `Reset to defaults`, che riporta i vari parametri (attivazione di `castellatedMesh`, `snap`, `addLayers` ai valori di default interni dell'applicazione, equivalenti a quelli del caso template. Solo dopo aver premuto nuovamente `Write` tali valori vengono effettivamente riscritti nell'apposito file del caso [16, 17, 36].

2.7.4 snapControls: parametri di snapping

Il pannello `snapControls`, riportato in Figura 2.6, espone in forma guidata i parametri che controllano la fase di snapping. Tra i principali troviamo: il numero di iterazioni dello snapping e del solver associato, la tolleranza di snapping, che determina fino a che punto le celle vengono spostate verso la superficie e i parametri di smoothing, che influenzano la regolarità e l'omogeneità della mesh alla fine di questa fase.

In un setup manuale questi valori andrebbero modificati direttamente nel dizionario `snappyHexMeshDict`; l'interfaccia proposta consente invece di concentrarsi sull'effetto numerico (maggiore aderenza alla geometria o maggiore robustezza) senza dover ricordare la sintassi dei file di input. Le modifiche vengono applicate immediatamente al dizionario del caso `case/`, garantendo una configurazione specifica per il problema in esame [37, 16].

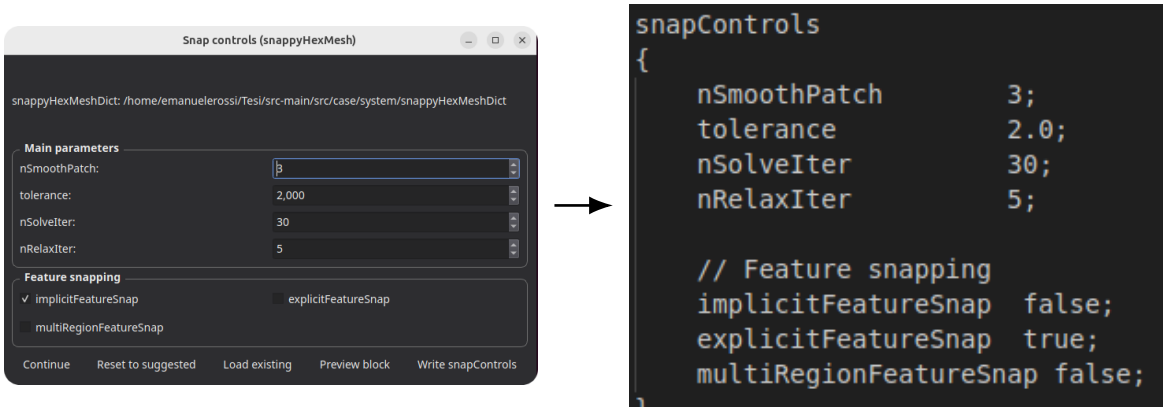


Figura 2.6: Finestra di configurazione di `snapControls`: parametri per smoothing della superficie, tolleranza di snapping e numero di iterazioni del solver.

In tutte le finestre di configurazione (ad eccezione di quella di orientamento STL) il pulsante `Preview` mostra un'anteprima testuale della sezione del file `OpenFOAM` corrispondente al widget in uso, già compilata con i parametri inseriti dall'utente e nel formato sintattico corretto. Questo permette di verificare immediatamente l'effetto delle modifiche prima di lanciare effettivamente la fase di `snappyHexMesh`. In questo widget sono disponibili due pulsanti addizionali:

- `Load existing`: legge i valori già presenti nel file `snappyHexMeshDict` della cartella `case/` (copiato dal relativo template) e li ricarica nei campi della GUI.
- `Reset to suggested`: sostituisce i valori correnti con un set di parametri suggeriti dall'applicazione (una combinazione "ragionevole" per il caso dimostrativo), mantenendo coerenza con i preset di qualità selezionati. Anche in questo caso, le modifiche vengono effettivamente scritte nel dizionario solo dopo aver premuto `Write`.

2.7.5 addLayersControls: gestione della zona in prossimità della superficie dell'oggetto

I boundary layers sono fondamentali per una corretta risoluzione dello strato limite vicino alle pareti. La GUI `addLayersControls`, mostrata in Figura 2.7, permette di impostare i principali parametri di questo blocco:

- numero di strati da estrudere (`nSurfaceLayers`);
- rapporto di espansione tra uno strato e il successivo (`expansionRatio`);
- spessore dell'ultimo layer, ovvero quello più esterno (`finalLayerThickness`) e spessore totale minimo (`minThickness`);
- attivazione (o disattivazione) del calcolo degli spessori ("thicknesses") in termini di dimensioni relative rispetto alla dimensione della cella di base

(`relativeSizes`) (dove per cella di base si intende la cella a contatto con la superficie dell'oggetto STL in seguito alla fase di snap).

L'utente può quindi controllare la distribuzione dei layers in funzione:

- dell'obiettivo di y^+ sul primo strato (ad esempio $y^+ \approx 1$ per simulazioni near–wall oppure $30 \lesssim y^+ \lesssim 100$ in presenza di wall functions);
- dello spessore complessivo dello strato limite che si desidera risolvere, ottenuto da stime di teoria dello strato limite;
- dei vincoli di qualità imposti dai controlli di `checkMesh` (volumi minimi, skewness, non–orthogonality), che limitano rapporti di espansione e spessori troppo estremi [21, 22, 38, 19].

Anche in questo caso, le impostazioni scelte nella GUI vengono tradotte automaticamente in una sezione coerente del dizionario `snappyHexMeshDict` del caso di simulazione [36, 39, 40].

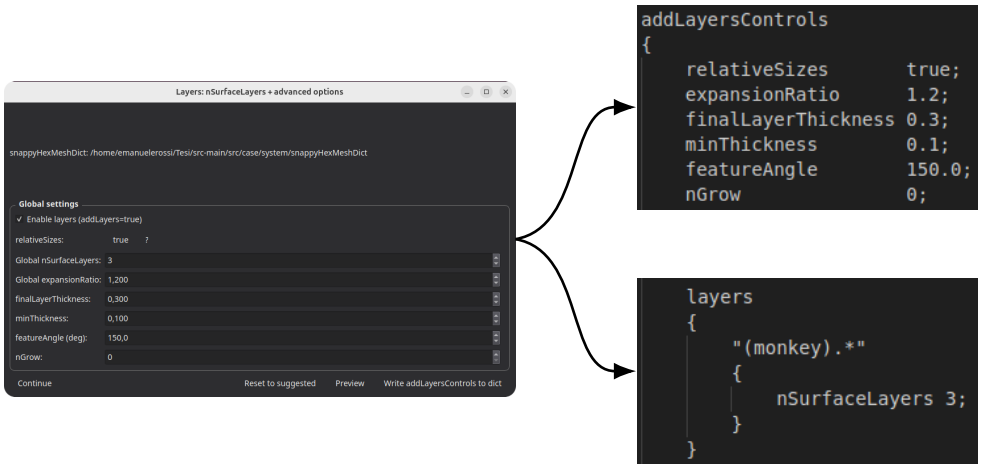
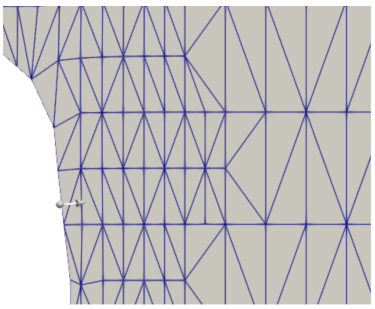
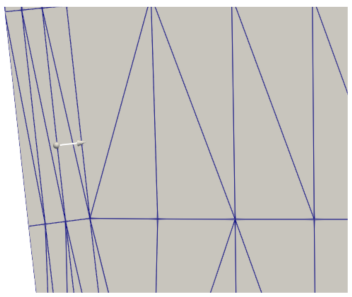


Figura 2.7: GUI per `addLayersControls`: impostazione di `relativeSizes`, numero di layers, `expansionRatio`, `finalLayerThickness` e parametri globali per l'estrusione dei layers.



(a) Misura locale (dopo la fase di snap e prima della fase di addLayers).



(b) Misura locale (dopo la fase di addLayers).

Figura 2.8: Confronto delle misure locali della mesh vicino alla parete (a–b), ottenute con `relativeSizes=true`, `expansionRatio=1.2`, `finalLayerThickness=0.3`.

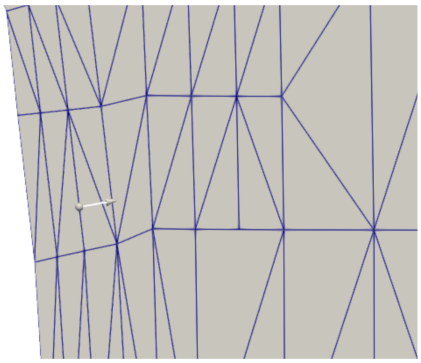


Figura 2.9: Misura locale della mesh con impostazioni alternative: `relativeSizes=false`, `expansionRatio=1.2`, `finalLayerThickness=0.03`.

Le Figure 2.8 e 2.9 rappresentano dei test di verifica del funzionamento della generazione dei *boundary layers* con i parametri `relativeSizes`, `finalLayerThickness` ed `expansionRatio` indicati nelle didascalie. Nel caso `relativeSizes=true` con `expansionRatio=1.2` si è misurato uno spessore dell'ultimo layer pari a **0.0142342**, mentre la dimensione della cella di base a contatto con la superficie (dopo `snap`, prima di `addLayers`) è **0.0506198**. Nel caso `relativeSizes=false` (sempre con `expansionRatio=1.2` e con `finalLayerThickness=0.030`) lo spessore dell'ultimo layer risulta **0.0301673**. Queste misure confermano la coerenza tra i parametri impostati tramite la GUI e la mesh effettivamente generata [21, 22, 38]. Anche in questa finestra è presente il pulsante `Reset to suggested`, che ripristina una configurazione consigliata per il caso corrente (ad esempio `relativeSizes=true`, `expansionRatio=1.2`, `finalLayerThickness=0.3` e numero di layers standard pari a 3). Come per gli altri widget, queste impostazioni vengono applicate ai file di input solo dopo aver premuto il pulsante `Write`, mentre `Continue` si limita a chiudere la finestra e proseguire nella pipeline.

2.8 Qualità della mesh: preset e verifica

In questo paragrafo si descrivono gli strumenti messi a disposizione dalla GUI per controllare la qualità della mesh generata e per uniformare i criteri di verifica tra `snappyHexMesh` e `checkMesh`. L'obiettivo è ridurre al minimo la modifica manuale dei relativi dizionari, proponendo alcuni profili predefiniti (preset) e una visualizzazione compatta dei log di esecuzione [21, 22, 18, 38, 40].

2.8.1 Preset unificati per snappyHexMesh e checkMesh

La qualità della mesh in `OpenFOAM` è controllata da un insieme di soglie e limiti (non–orthogonality, skewness, volumi minimi, ecc.) che compaiono sia all'interno di `snappyHexMeshDict` (blocco `meshQualityControls`) sia nel file `meshQualityDict` letto da `checkMesh`. La configurazione manuale di questi due insiemi di parametri è soggetta a errori e, soprattutto, rende difficile mantenere criteri coerenti tra fase di generazione della mesh e fase di verifica [21, 22, 17].

La GUI dei preset di qualità, mostrata in Figura 2.10, introduce alcuni profili predefiniti (ad esempio “More permissive”, “Balanced”, “Stricter”) che vengono applicati in modo simultaneo ai due dizionari:

- nel blocco `meshQualityControls` all’interno di `snappyHexMeshDict`, che controlla l’arresto del processo o l’emissione di warning durante la generazione della mesh;
- nel file `meshQualityDict`, che definisce i criteri con cui `checkMesh` controlla il risultato finale e classifica le celle come accettabili o problematiche.

In questo modo, selezionando un singolo preset, l’utente imposta in maniera consistente i limiti su non–orthogonality, skewness, concavità delle celle, volumi minimi e altri indicatori geometrici. Il profilo “More permissive”, ad esempio, consente di completare la generazione della mesh anche in presenza di celle più distorte, risultando utile per fasi preliminari di sviluppo del caso, mentre profili più restrittivi sono indicati per le simulazioni finali [38, 40].

Figura 2.10: GUI dei preset di qualità (in alto) e, sotto, le anteprime testuali dei blocchi `meshQualityControls` (sinistra) e `meshQualityDict` (destra), utili per confrontare i valori dei preset con i blocchi che verranno effettivamente scritti nei dizionari.

Per dare all’utente la possibilità di vedere sin da subito le modifiche apportate tramite le scelte in questa sezione della pipeline, la GUI offre un’anteprima testuale dei blocchi `meshQualityControls` e `meshQualityDict`, come si può vedere

sempre in Figura 2.10. In questa vista nella finestra di sinistra vengono riportati i valori dei parametri che verranno scritti nella sezione `meshQualityControls` di `snappyHexMeshDict` mentre nella colonna di destra compaiono i valori che andranno invece nel file `meshQualityDict`. Il pulsante `Refresh status` permette di ricaricare lo stato effettivo dei dizionari del caso `case/`; la GUI inoltre legge i file `snappyHexMeshDict` e `meshQualityDict` correnti e indica se i valori presenti corrispondono ad uno dei preset disponibili oppure se la configurazione è stata modificata manualmente (stato "custom"). Questo è utile quando si alternano modifiche tramite GUI e modifiche esterne, o quando si vuole semplicemente verificare che i parametri applicati al caso siano quelli attesi.

2.8.2 Log di snappyHexMesh

L'esecuzione di `snappyHexMesh` produce un log piuttosto esteso, e la finestra mostrata in Figura 2.11 permette di seguire in tempo reale la scrittura di tale file (`log_snappyHexMesh.txt`): la finestra esegue una "live" continua del log, evidenziando il numero di *Warnings* e di *Errors* trovati nel testo.

L'utente può decidere se mantenere lo scorrimento automatico verso il fondo (*Auto–scroll*) oppure fermarsi su una porzione specifica del log. Attraverso il pulsante `Open log...` è inoltre possibile aprire il file completo in un editor esterno, per una consultazione più dettagliata dei messaggi di `snappyHexMesh` [20, 18].

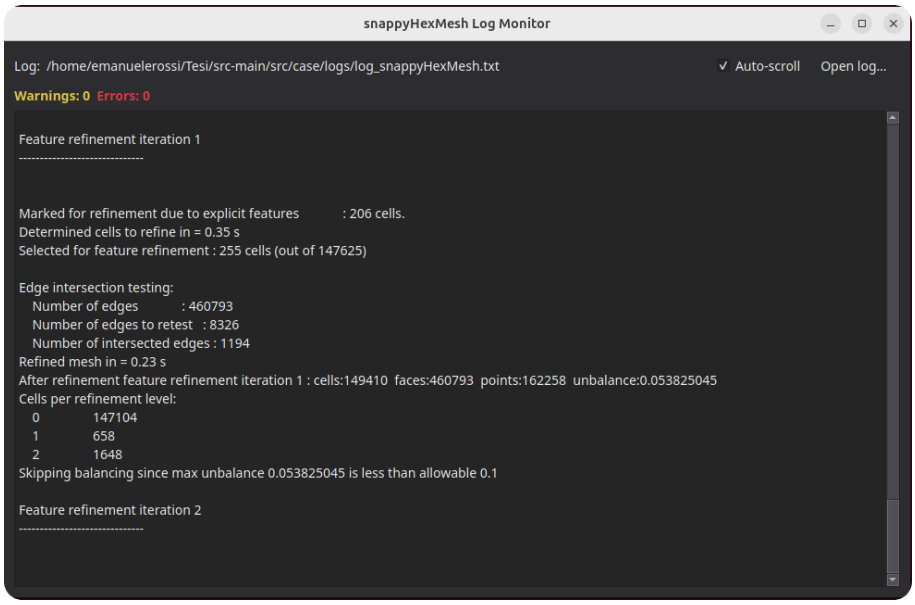


Figura 2.11: Log monitor di `snappyHexMesh`: durante la fase di raffinamento vengono mostrati numero di celle selezionate, feature edges e tempi di esecuzione.

2.9 Turbulenza e setup solver

2.9.1 Configurazione dei modelli RANS/LES

La configurazione della turbolenza viene gestita attraverso un pannello dedicato, mostrato in Figura 2.12. Lo script `turbulence_widget.py` fornisce un'interfaccia

per impostare:

- il *tipo di simulazione* (ad esempio simulazione laminare, modello RANS oppure LES), corrispondente al campo `simulationType` nel file `constant/turbulenceProperties`;
- il *modello di turbolenza* specifico (ad esempio `SpalartAllmaras`, `kOmegaSST`, `kEpsilon`, ecc.), selezionato tramite il parametro `RASModel` o `LESModel` nello stesso dizionario;
- il *solver* `OpenFOAM` da utilizzare, che viene scritto nel campo `application` del file `system/controlDict` in modo consistente con il modello scelto (ad esempio `pisoFoam` per simulazioni transitorie incomprimibili RANS).

In un flusso di lavoro manuale, queste scelte richiederebbero modifiche parallele a più file (`turbulenceProperties`, campi iniziali nella cartella `0/`, `controlDict`), con il rischio di introdurre incoerenze tra modello fisico e solutore numerico. La GUI si occupa invece di:

- aggiornare il file `turbulenceProperties` in base al tipo di simulazione selezionato (ad esempio impostando `simulationType RAS` e `RASModel kOmegaSST` nel caso dimostrativo);
- assicurare che i campi presenti nella directory `0/` siano compatibili con il modello scelto (ad esempio la presenza di `nuTilda` per Spalart–Allmaras oppure di `k` e `epsilon`/`omega` per modelli a due equazioni);
- impostare nel `controlDict` un solver adatto al modello selezionato, evitando accoppiamenti non fisicamente significativi (ad esempio modello RANS incomprimibile con solver compressibile, o viceversa).

In questo modo l'utente può concentrarsi sugli aspetti fisici della modellazione (tipo di turbolenza, livello di complessità del modello, scelta tra RANS e LES) senza doversi preoccupare della sintassi dei dizionari `OpenFOAM` [26, 27].

Figura 2.12: GUI `turbulence_widget.py` (sinistra) e mappatura dei parametri scelti verso i dizionari/anteprime (destra).

2.10 Esecuzione locale/remota

2.10.1 Decomposizione dominio e multi-core

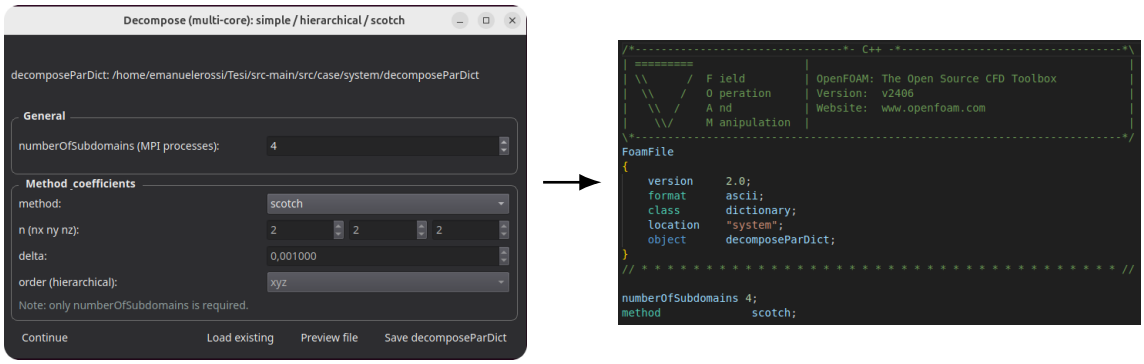


Figura 2.13: Finestra di configurazione di `decomposeParDict`: scelta del metodo `scotch`, definizione del numero di subdomain e dei parametri di decomposizione per l'esecuzione multi-core del caso.

La decomposizione del dominio (multi-core) consente di suddividere la mesh in più sotto-domini indipendenti che possono essere elaborati in parallelo da processi distinti (tipicamente uno per core). Il metodo `scotch` è una scelta robusta e automatica per il bilanciamento del carico: minimizza le interfacce tra partizioni e distribuisce in modo omogeneo il numero di celle. In genere il numero di *subdomains* viene posto uguale (o multiplo) al numero di core disponibili. Una buona decomposizione riduce i tempi di calcolo e di comunicazione tra processi, con beneficio diretto sul tempo totale di simulazione e sulla scalabilità [41, 42].

2.10.2 Scelta tra run locale e remoto ed esecuzione del solver

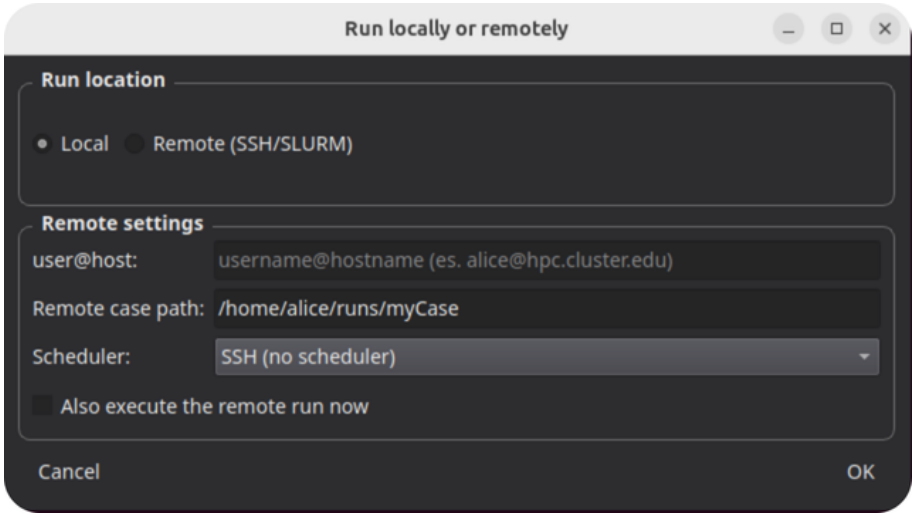


Figura 2.14: Helper `remote_run_helper.py`: scelta tra esecuzione locale e remota via SSH/SLURM, con impostazione di `user@host`, percorso remoto del caso e opzione per eseguire subito la sincronizzazione e il lancio della simulazione.

Questa schermata permette di decidere se eseguire la simulazione in locale (sulla macchina dell'utente) oppure su una macchina remota. L'opzione remota è utile quando si dispone di cluster con più nodi, code di calcolo (*scheduler* come SLURM) e storage ad alte prestazioni. L'`helper` gestisce le credenziali `SSH`, la sincronizzazione dei file tra cartella locale e percorso remoto, e la preparazione del comando di lancio. Da sottolineare che per *cluster* si intende un insieme di nodi di calcolo (più macchine fisiche o virtuali) interconnessi tramite rete ad alta velocità e gestiti da uno *scheduler* (es. SLURM) che distribuisce i job, bilancia il carico e coordina l'esecuzione parallela. In questo modo è possibile sfruttare in modo trasparente decine/centinaia di core e grandi quantità di memoria rispetto a una singola macchina [41, 42]. L'esecuzione del solver prosegue con: preparazione dei dizionari (inclusi i modelli di turbolenza), decomposizione del dominio (nel caso questa opzione sia stata selezionata) e lancio del solver. Al termine, la fase di `reconstructPar` riunisce, in caso di calcolo multi-core, i risultati dei *subdomains* in un campo unico per il post–processing. Questa sequenza consente di automatizzare il flusso di lavoro [18].

Capitolo 3

Discussione dei risultati

3.1 Risultati della simulazione e salvataggio dei campi

Al termine dell'esecuzione del solver, la cartella `case/` contiene l'intera storia temporale della simulazione, organizzata in directory identificate dai cosiddetti *timesteps*, oltre ai file di controllo in `system/` e ai file fisici in `constant/`. Questa struttura permette di riaprire il caso in ParaView e di accedere a qualsiasi istante salvato per analizzare i campi di interesse (velocità, pressione, variabili di turbolenza).

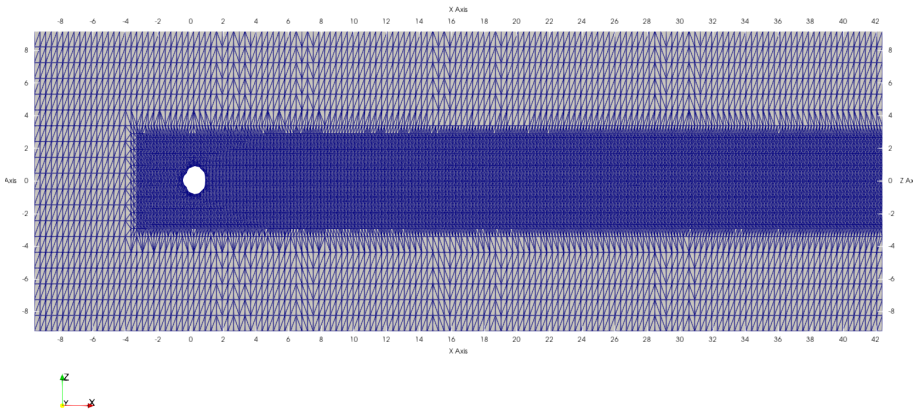


Figura 3.1: Vista complessiva della mesh in ParaView: si osserva il blocco di raffinamento intorno alla geometria STL e la discretizzazione regolare del dominio a monte e a valle.

I parametri riportati in Tabella 3.1 riassumono le scelte principali effettuate nella fase di setup. Dal punto di vista geometrico, la mesh contiene poco meno di un milione di celle, in gran parte esaedriche, con un raffinamento mirato sulla superficie `monkey` e nella regione di interesse a valle dell'ostacolo. La presenza di un numero limitato di poliedri e i valori di volume minimo/massimo risultano compatibili con una discretizzazione adatta a simulazioni RANS a scala ingegneristica. I parametri di layers (numero di strati, `expansionRatio`, `finalLayerThickness`) sono scelti in modo da risolvere lo strato limite vicino alla superficie STL senza introdurre celle

Tabella 3.1: Parametri principali del caso dimostrativo.

Geometria e mesh	
Numero totale di celle	980 430 (da `checkMesh`)
Numero di punti	1 043 554 (da `checkMesh`)
Tipo di celle prevalente	esaedri (943 884), con 36 160 poliedri (da `checkMesh`)
Volume minimo di cella	$3.87 \times 10^{-6}\,\mathrm{m}^3$ (da `checkMesh`)
Volume massimo di cella	$1.22 \times 10^{-1}\,\mathrm{m}^3$ (da `checkMesh`)
Volume totale del dominio	$1.73 \times 10^{4}\,\mathrm{m}^3$ (da `checkMesh`)
Lunghezza minima degli spigoli	$2.73 \times 10^{-3}\,\mathrm{m}$ (da `checkMesh`)
Lunghezza massima degli spigoli	$9.75 \times 10^{-1}\,\mathrm{m}$ (da `checkMesh`)
Livelli di raffinamento sulla superficie STL	`level (2 3)` su patch `monkey` (da `snappyHexMeshDict`)
Raffinamento nella regione di interesse	`refinementBox` con livello 2 (`levels ((1E15 2))`) (da `snappyHexMeshDict`)
Layer	`addLayers true`, `nSurfaceLayers 3` su `monkey` (da `snappyHexMeshDict`)
Parametri principali dei layers	`relativeSizes true`, `expansionRatio 1.2`, `finalLayerThickness 0.3` (da `snappyHexMeshDict`)
Partizionamento	`numberOfSubdomains 4`, metodo `scotch` (da `decomposeParDict`)
Modello fisico e numerico	
Tipo di simulazione	RANS (`simulationType RAS`) (da `turbulenceProperties`)
Modello di turbolenza	`RASModel kOmegaSST`, `turbulence on` (da `turbulenceProperties`)
Solver	`pimpleFoam` (da `controlDict`)
Intervallo temporale	`startTime 0`, `endTime 10` s (da `controlDict`)
Passo temporale base	`deltaT 0.001` s, con `adjustTimeStep yes`, `maxCo 0.7`, `maxDeltaT 0.1` (da `controlDict`)
Frequenza di scrittura	`writeControl timeStep`, `writeInterval 1` (output ad ogni passo temporale) (da `controlDict`)
Criteri di convergenza per p	`solver GAMG`, `tolerance 1e-5`, `relTol 0` (da `fvSolution`)
Criteri di convergenza per U, k, ϵ	`tolerance 1e-5`, `relTol 0` (da blocco `(U|k|epsilon)`) (da `fvSolution`)
Criteri di convergenza per ω e $\tilde{\nu}$	`tolerance 1e-5`, `relTol 0` (da `fvSolution`)

eccessivamente distorte, mentre il partizionamento in più "subdomain" (quattro in questo caso specifico) consente, se necessario, l'esecuzione parallela del caso.

Dal punto di vista fisico–numerico, la simulazione è impostata come una RANS con modello `kOmegaSST` e solver `pimpleFoam`, con intervallo temporale $0 \leq t \leq 10$ s e passo di base $\Delta t = 10^{-3}$ s. L'opzione `adjustTimeStep yes` e il limite `maxCo 0.7` mantengono il numero di Courant sotto una soglia conservativa, mentre le tolleranze dei solver lineari per p, U e le variabili di turbolenza sono fissate a 10^{-5} con `relTol 0`, garantendo una riduzione consistente dei residui ad ogni passo.

Prima di analizzare i campi di velocità è utile verificare l'andamento del numero di Courant (CFL), che misura in modo adimensionale quanto "velocemente" l'informazione numerica si sposta da una cella all'altra ad ogni passo temporale [26, 27]. In termini molto semplici, un valore di CFL pari a 1 significa che in un singolo Δt il fluido percorrerebbe circa una cella di mesh; valori più piccoli corrispondono a passi temporali più cautelativi e quindi a una simulazione più stabile.

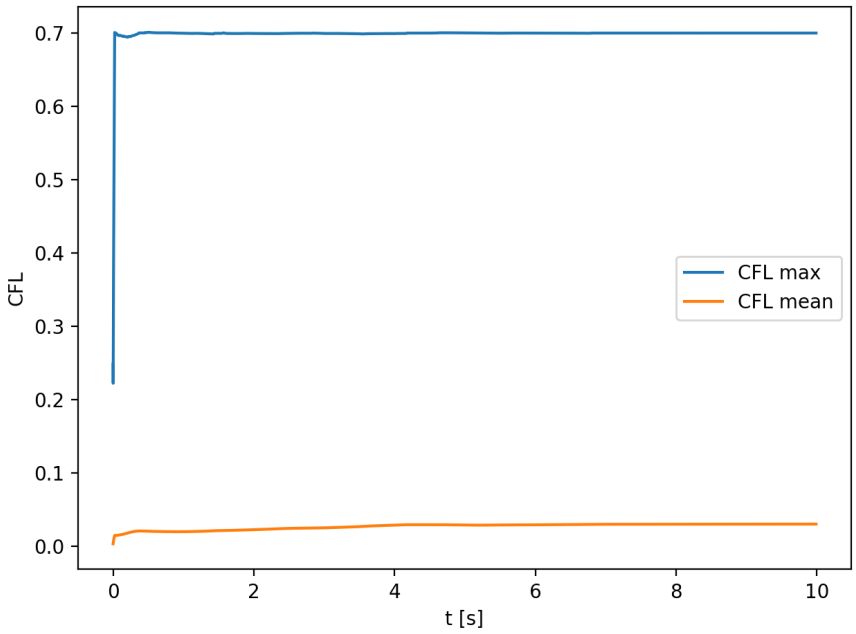


Figura 3.2: Evoluzione nel tempo del numero di Courant (*CFL*) medio e massimo durante la simulazione.

Come mostrato in Figura 3.2, dopo un breve transitorio iniziale il valore massimo di CFL si assesta intorno a 0.7, in accordo con il limite imposto in `controlDict` (`maxCo 0.7`) [26, 27], mentre il valore medio rimane sempre inferiore a ≈ 0.03. Questo conferma che la strategia `adjustTimeStep yes` mantiene il passo temporale sufficientemente piccolo da garantire un'andamento numericamente stabile e coerente con l'impostazione conservativa adottata per la simulazione. Oltre al controllo sul numero di Courant, è utile verificare anche l'andamento del parametro adimensionale y^+ sulla superficie dell'ostacolo, che misura in modo qualitativo quanto la prima cella di mesh adiacente alla parete sia vicina o lontana rispetto allo spessore dello strato limite viscoso. Valori molto piccoli ($y^+ \sim 1$) indicano una risoluzione diretta dello strato limite (approccio "tipo DNS/LES near–wall"), mentre valori più elevati ($y^+ \gtrsim 30$) sono tipici di impostazioni basate su wall function, in cui lo strato limite più interno non è completamente risolto ma modellato [43, 40].

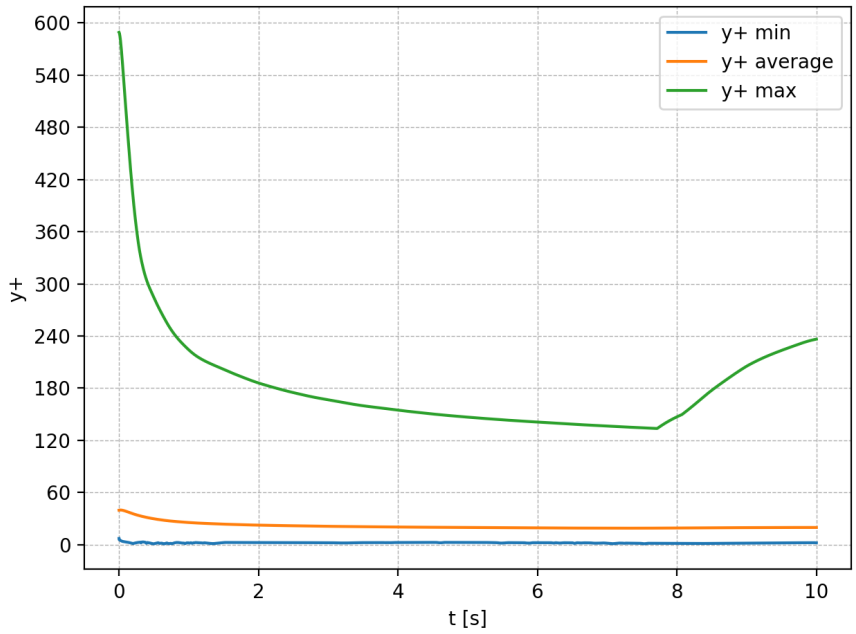


Figura 3.3: Evoluzione nel tempo dei valori minimo, medio e massimo di y^+ sulla superficie dell'oggetto (`monkey`).

Come si osserva in Figura 3.3, dopo un transitorio iniziale in cui $y^+_{\max}$ supera 500, i valori massimi decrescono rapidamente e si assestano nell'intervallo $y^+_{\max} \approx 130$–250, mentre il valore medio rimane intorno a $y^+_{\text{avg}} \approx 20$. Il valore minimo resta invece sempre dell'ordine dell'unità. Nel complesso, la prima cella di mesh si colloca nella regione logaritmica dello strato limite per la maggior parte della superficie, pur mantenendo localmente valori più bassi in zone di forte accelerazione o separazione del flusso. Questo è coerente con l'impiego di wall function nel modello RANS adottato, che richiedono un y^+ tipicamente dell'ordine di alcune decine per operare in condizioni ottimali [43].

Figura 3.4: Andamento del modulo della velocità lungo una linea verticale che attraversa la zona immediatamente a valle della geometria nella direzione del flusso principale (da sinistra a destra): il profilo mostra la decelerazione in prossimità dell'ostacolo e il successivo recupero a valle.

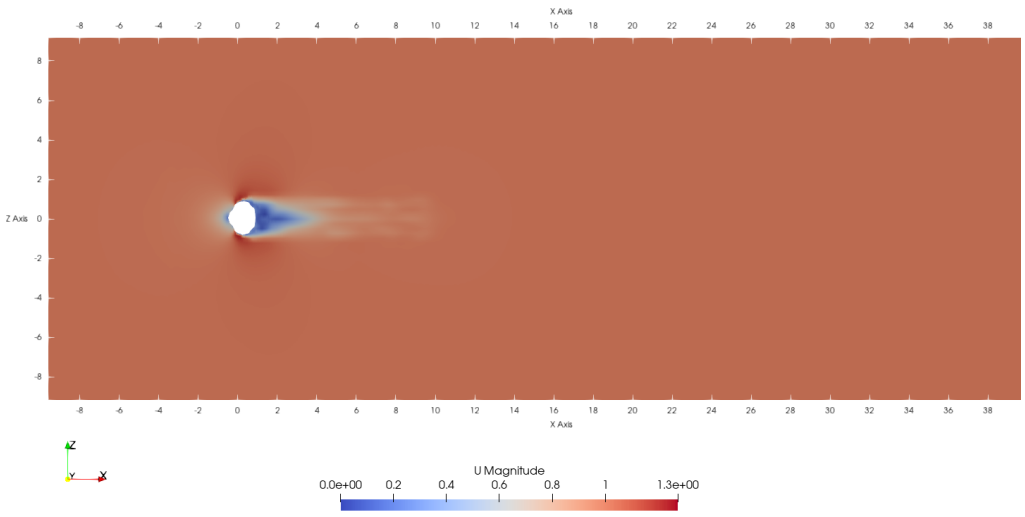


Figura 3.5: Campo di velocità (modulo) su un piano individuato in corrispondenza dell'oggetto STL: si osserva la zona di rallentamento a monte dell'ostacolo e l'accelerazione laterale del flusso che attraversa la zona superiore e quella inferiore della geometria.

L'andamento della velocità in Figura 3.4 mostra un valore pressoché uniforme pari

a $U \approx 1\,\mathrm{m/s}$ lontano dall'oggetto, una zona di forte decelerazione in prossimità dell'ostacolo e un lieve aumento della velocità nelle regioni in cui il flusso si restringe per aggirare la geometria. La forma della curva è coerente con la distribuzione tipica di velocità attorno a un corpo *bluff* ("tozzo") immerso in un flusso di canale [44], e il campo riportato in Figura 3.5 conferma la presenza di una regione di ristagno a monte e di una scia rallentata a valle.

Si sottolinea che i risultati presentati sono limitati a una finestra temporale relativamente breve ($t \leq 10$ s). I tempi di calcolo e le risorse computazionali disponibili non hanno permesso di effettuare simulazioni più lunghe o uno studio parametrico esteso; le analisi riportate rappresentano quindi una prima validazione qualitativa della pipeline e del setup numerico, lasciando a sviluppi futuri l'approfondimento su orizzonti temporali maggiori e su configurazioni alternative.

Capitolo 4

Conclusioni

Il lavoro presentato in questa tesi ha avuto come obiettivo principale la realizzazione di una pipeline guidata, basata su interfacce grafiche (*GUI*), per la preparazione e l'esecuzione di casi `OpenFOAM` a partire da una geometria STL fornita dall'utente. L'idea di fondo è stata quella di ridurre la distanza tra l'utente e la struttura, talvolta complessa, dei dizionari di input (`blockMeshDict`, `snappyHexMeshDict`, `fvSchemes`, `fvSolution`, `controlDict`, ...), mantenendo però la piena trasparenza e modificabilità dei file tipica dell'ambiente open–source, cercando allo stesso tempo di automatizzare, e quindi anche di velocizzare, alcune parti della preparazione del caso.

Dal punto di vista della *pipeline di mesh*, il sistema sviluppato permette di partire da un singolo file STL, orientarlo e posizionarlo nel dominio di calcolo, generare automaticamente la cartella `case/` a partire da un suo template e configurare in maniera guidata le fasi di `snappyHexMesh`. In particolare, la presenza di preset di qualità unificati per `snappyHexMesh` e `checkMesh` riduce il rischio di incoerenze tra criteri di generazione e criteri di verifica della mesh, facilitando il passaggio da mesh preliminari più permissive a mesh finali più restrittive. I test condotti sul caso dimostrativo hanno confermato la coerenza tra i parametri impostati nella GUI (e.g. `relativeSizes`, `expansionRatio`, `finalLayerThickness`) e la mesh effettivamente ottenuta, verificata sia attraverso `checkMesh` sia mediante misure locali in ParaView.

Per quanto riguarda la *pipeline di turbolenza e solver*, il widget dedicato consente di selezionare il tipo di simulazione (laminare, RANS, LES), il modello di turbolenza e il solver compatibile, aggiornando in modo automatico i file `turbulenceProperties`, `controlDict` e i campi iniziali in `0/`. Questo approccio aiuta a evitare combinazioni non consistenti tra modello fisico e configurazione numerica e rende più esplicito il legame tra le scelte effettuate nelle GUI e il contenuto finale dei dizionari. La possibilità di impostare la decomposizione del dominio e di generare, tramite `remote_run_helper.py`, script di lancio per esecuzioni multi–core locali o su cluster remoti, fornisce inoltre un primo supporto all'utilizzo di risorse HPC senza dover scrivere manualmente i relativi script.

Il caso RANS dimostrativo, basato sul modello `kOmegaSST` e su `pimpleFoam`, ha permesso di validare qualitativamente l'intera pipeline. L'analisi del numero di Courant ha mostrato che la combinazione `adjustTimeStep yes` + `maxCo 0.7` mantiene il CFL massimo stabilmente intorno a 0.7 e il valore medio su livelli decisamente più bassi, garantendo una simulazione numericamente robusta. Il monitoraggio del

parametro y^+ sulla superficie `monkey` ha evidenziato un comportamento coerente con l'impiego di wall function: dopo un transitorio iniziale, $y^+_{\max}$ scende su valori dell'ordine di 10^2, mentre la media si stabilizza intorno a $y^+_{\text{avg}} \approx 20$, con minimi prossimi all'unità in alcune zone. Complessivamente la prima cella di mesh si colloca prevalentemente nella regione logaritmica dello strato limite, pur mantenendo localmente una risoluzione più fine, in linea con le aspettative per una simulazione RANS ingegneristica.

Il campo di velocità ottenuto mostra una struttura di flusso ragionevole attorno all'ostacolo (regione di ristagno a monte, scia rallentata a valle, accelerazioni laterali), confermando che il caso generato dalla pipeline si comporta in modo coerente con l'impostazione fisica del problema. È importante sottolineare che lo scopo del caso dimostrativo non era quello di produrre un risultato perfettamente validato, da un punto quantitativo, dai dati sperimentali, ma di verificare la correttezza e la coerenza del workflow automatizzato dalla GUI, dalla preparazione della mesh fino al post–processing dei principali indicatori numerici (CFL, y^+, campi di flusso).

Naturalmente, il sistema presenta anche alcuni limiti. La pipeline è stata progettata e testata su flussi incomprimibili con modelli di turbolenza RANS (con semplici estensioni a LES) e geometrie descritte da singoli file STL. L'estensione a casi comprimibili, a simulazioni multi–fase, a problemi accoppiati (ad esempio CFD–strutturale) o a template con modelli fisici più complessi richiederebbe la definizione di nuovi dizionari, l'aggiunta di campi iniziali e la progettazione di GUI dedicate per le nuove sezioni di input. Allo stesso modo, le funzionalità di run remoto devono essere considerate come una prima implementazione, che necessita di ulteriori prove su infrastrutture HPC reali.

Nonostante questi limiti, il lavoro svolto rappresenta un passo concreto verso un uso più accessibile di `OpenFOAM`, in particolare per utenti alle prime esperienze con la riga di comando e la modifica manuale dei dizionari. Le interfacce sviluppate mantengono la filosofia open–source del codice di calcolo (i casi generati sono ordinari casi `OpenFOAM`, completamente ispezionabili e modificabili), ma ne alleggeriscono la fase di setup, automatizzando le operazioni più ripetitive e soggette a errore. In prospettiva, l'architettura modulare adottata per le GUI e l'uso di template di caso rende il sistema facilmente estendibile: nuove pipeline, nuovi preset di qualità e nuovi modelli fisici possono essere integrati riutilizzando gli stessi meccanismi di base, facendo di questa tesi non solo uno strumento operativo per il caso dimostrativo presentato, ma anche una piattaforma di partenza per ulteriori sviluppi in ambito di ricerca e didattica CFD. Il codice sorgente e gli script sviluppati nell'ambito di questo lavoro sono resi pubblicamente disponibili nel repository GitHub dell'autore [45].

Bibliografia

[1] "Simflow: Cfd software." https://sim-flow.com/.

[2] "Helyx: Cfd software by engys." https://engys.com/products/helyx.

[3] "Simscale cloud simulation platform." https://www.simscale.com/.

[4] E. Rossi, "Meshing_workflow." https://github.com/EmanueleRossi-pg/Meshing_workflow. GitHub repository.

[5] "Qt documentation." https://doc.qt.io/.

[6] "Pyqt documentation." https://www.riverbankcomputing.com/static/Docs/PyQt6/.

[7] "Numpy." https://numpy.org/.

[8] "Numpy documentation." https://numpy.org/doc/.

[9] "Numpy reference — numpy v2.4.dev0 manual." https://numpy.org/devdocs/reference/.

[10] "Matplotlib." https://matplotlib.org/.

[11] "Matplotlib backends (agg)." https://matplotlib.org/stable/users/explain/backends.html.

[12] "Openfoam foundation (openfoam.org)." https://openfoam.org/.

[13] "Welcome to numpy-stl's documentation! — numpy stl 2.16.3 documentation." https://numpy-stl.readthedocs.io/en/latest/.

[14] "numpy-stl — numpy stl 2.16.3 documentation." https://numpy-stl.readthedocs.io/en/latest/usage.html.

[15] "numpy-stl · pypi." https://pypi.org/project/numpy-stl/.

[16] "Openfoam: User guide: snappyhexmesh." https://www.openfoam.com/documentation/guides/latest/doc/guide-meshing-snappyhexmesh.html.

[17] "Openfoam v13 user guide - 5.5 mesh generation with snappyhexmesh." https://doc.cfd.direct/openfoam/user-guide-v13/snappyhexmesh.

[18] "Openfoam v13 user guide - 3.7 standard utilities." https://doc.cfd.direct/openfoam/user-guide-v13/standard-utilities.

[19] "Paraview reference manual — paraview documentation 6.0.0 documentation." https://docs.paraview.org/en/latest/ReferenceManual/index.html.

[20] "Foamlog - openfoamwiki." https://openfoamwiki.net/index.php/FoamLog.

[21] "Checkmesh - openfoamwiki." https://openfoamwiki.net/index.php/CheckMesh.

[22] "Openfoam: User guide: Mesh quality." https://www.openfoam.com/documentation/guides/latest/doc/guide-meshing-snappyhexmesh-meshquality.html.

[23] "Openfoam: Manual pages: surfacecheck(1)." https://www.openfoam.com/documentation/guides/v2112/man/surfaceCheck.html.

[24] "Openfoam: Api guide: applications/utilities/surface/surfacecheck/surfacecheck.c file reference." https://www.openfoam.com/documentation/guides/v2206/api/surfaceCheck_8C.html.

[25] "Surface check and repair utilities - need explanation – cfd online discussion forums." https://www.cfd-online.com/Forums/openfoam-solving/239846-surface-check-repair-utilities-need-explanation.html.

[26] "6.1 time and data input/output control." https://www.openfoam.com/documentation/user-guide/6-solving/6.1-time-and-data-inputoutput-control.

[27] "Openfoam: User guide: controldict." https://www.openfoam.com/documentation/guides/latest/doc/guide-case-system-controldict.html.

[28] "TransformPoints - OpenFOAMWiki." https://openfoamwiki.net/index.php/TransformPoints.

[29] "OpenFOAM: API Guide: applications/utilities/mesh/manipulation/transformPoints/transformPoints.C File Reference." https://www.openfoam.com/documentation/guides/latest/api/transformPoints_8C.html.

[30] "Openfoam: Manual pages: transformpoints(1)." https://www.openfoam.com/documentation/guides/v2112/man/transformPoints.html.

[31] "Openfoam: Manual pages: surfacetransformpoints(1)." https://www.openfoam.com/documentation/guides/v1912/man/surfaceTransformPoints.html.

[32] "linear algebra - relationship between the singular value decomposition (svd) and the principal component analysis (pca). a radical result - mathematics stack exchange." https://math.stackexchange.com/questions/816884/relationship-between-the-singular-value-decomposition-svd-and-the-principal-co.

[33] F. L. Gewers, G. R. Ferreira, H. F. D. Arruda, F. N. Silva, C. H. Comin, D. R. Amancio, and L. D. F. Costa, "Principal component analysis: A natural approach to data exploration," *ACM Comput. Surv.*, vol. 54, pp. 70:1–70:34, 5 2021. https://dl.acm.org/doi/abs/10.1145/3447755.

[34] "The math behind pca • learnpca." https://bryanhanson.github.io/LearnPCA/articles/Vig_06_Math_Behind_PCA.html.

[35] "Pca — scikit-learn 1.7.2 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[36] "Mesh generation using snappyhexmesh 2," https://www.wolfdynamics.com/wiki/meshing_OF_SHM.pdf.

[37] "Openfoam: User guide: Snapping." https://www.openfoam.com/documentation/guides/latest/doc/guide-meshing-snappyhexmesh-snapping.html.

[38] "[other] mesh quality visualization in paraview – cfd online discussion forums." https://www.cfd-online.com/Forums/openfoam-meshing/199238-mesh-quality-visualization-paraview.html.

[39] "[snappyhexmesh] snappyhexmesh and 2 phase modelling – cfd online discussion forums." https://www.cfd-online.com/Forums/openfoam-meshing/61488-snappyhexmesh-2-phase-modelling.html.

[40] "Node144 – cfd support." https://www.cfdsupport.com/openfoam-training-by-cfd-support/node144/.

[41] "3.2 running applications in parallel." https://www.openfoam.com/documentation/user-guide/3-running-applications/3.2-running-applications-in-parallel.

[42] "Openfoam: User guide: Parallel." https://www.openfoam.com/documentation/guides/latest/doc/openfoam-guide-parallel.html.

[43] "Best practice: Scale-resolving simulations in ansys cfd." https://cfd.grs.de/sites/default/files/downloads/ansys/ANSYS_BPG-SRS-2.01.pdf.

[44] "Synopsis of lift, drag, and vortex frequency data for rigid circular cylinders." https://engines.egr.uh.edu/sites/engines/files/talks/vortexcylinders.pdf.

[45] E. Rossi, "Openfoam-mesh-solver-gui." https://github.com/EmanueleRossi-pg/OpenFOAM-Mesh-Solver-GUI/tree/main. GitHub repository.