POLITECNICO DI TORINO

Corso di Laurea
in Ingegneria Aerospaziale

Tesi di Laurea

Analisi di un'ala bio-ispirata tramite codice di calcolo OpenFOAM

Relatori

prof. Renzo Arina
ing. Stefania Scarsoglio

Candidato

Andrea Nalin

Ottobre 2015

Sommario

In questa tesi verrà analizzata un'ala con profilo NACA0021 caratterizzata da un bordo di attacco sinusoidale ispirato alle pinne pettorali della *Megaptera novaengliae*. Attraverso il codice di calcolo OpenFOAM verrà analizzato il flusso indotto in prossimità dell'ala, con particolare attenzione alle strutture vorticose presenti in scia. Proprio questi vortici inducono una variazione periodica nell'angolo di attacco effettivo dell'ala; variazione che, aumentando l'angolo di stallo e rendendo la perdità di portanza più graduale, provoca un aumento delle prestazioni. Il caso di OpenFOAM verrà impostato per risolvere numericamente le equazioni di Navier-Stokes utilizzando il modello di turbolenza k - ϵ e, infine, il software di post-processing ParaView sarà utilizzato per ottenere una rappresentazione dei vortici di scia.

Indice

Elenco delle figure

1 Introduzione

1.1 Megaptera novaengliae

La megattera è un cetaceo misticeto noto per la sua incredibile capacità di eseguire complicate manovre acrobatiche sottomarine. Questa eccezionale agilità, utilizzata soprattutto per cacciare attraverso la tecnica del *bubble net feeding*, può essere raggiunta grazie alle grandi pinne pettorali (il nome deriva appunto da questa caratteristica, *méga pterón grande ala*). Queste pinne però, oltre all'allungamento maggiore rispetto a tutti gli altri cetacei, presentano un'altra caratteristica di fondamentale importanza: dei tubercoli sul bordo di attacco.

Figura 1: Tubercoli visibili sul bordo di attacco delle pinne pettorali. Photograph by REUTERS/Mike Hutchings, obtained from www.focus.it

Queste protuberanze rendono lo stallo delle pinne meno "grave". Ritardano il sopraggiungere di quest'ultimo permettendo alla pinna di raggiungere angoli di attacco più elevati, lo rendono più graduale, e diminuiscono l'ampiezza della perdità improvvisa di portanza, caratteristica fondamentale dello stallo aerodinamico.

1.2 Stato dell'arte

Uno dei primi a confermare l'idea che questa particolare conformazione stesse alla base della grande agilità delle megattere fu Miklosovic *et al.* [1] nel 2004.

Attraverso esperimenti in galleria del vento riuscì a mostrare che un bordo d'attacco con andamento sinusoidale è in grado di incrementare l'angolo di stallo di circa il 40%, raggiungendo valori più elevati di portanza e minori di resistenza. Grazie a queste caratteristiche, pur raggiungendo un coefficiente di portanza massimo inferiore a un'ala con bordo di attacco retto, l'efficienza massima risulta superiore, e la configurazione presenta un ampio range di funzionamento a livelli di efficienza accettabili. Successivamente Johari *et al.* [2] provò che un bordo di attacco così modificato, in aggiunta ai benifici di cui sopra, rende anche lo stallo più graduale.

Nel 2008 van Nierop *et al.* [3] sviluppò un modello aerodinamico, basato sulla linea portante di Prandtl, capace di spiegare questo incremento delle prestazioni del profilo in prossimità dello stallo. La variazione della corda, causata da un bordo di attacco sinusoidale, provoca una variazione nella circuitazione Γ (determinata dalla condizione di Kutta).

$$\Gamma = -\pi U_o (c + \frac{4}{3\sqrt{3}} t)\alpha^e + O((\alpha^e)^3, t^3)$$

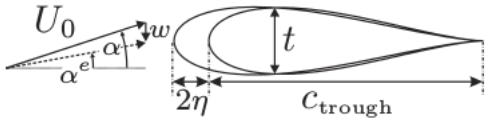

Figura 2: Variazione della corda in direzione dell'apertura alare [3]

Questa variazione (sinusoidale, come l'andamento della corda) comporta la presenza di vortici di scia, a cui è associato un downwash (componente ω) che modifica localmente l'angolo di attacco. L'intensità di questo downwash risulta maggiore dietro le "creste" della nostra ala, diminuendo dunque il loro angolo di attacco effettivo e ritardando lo stallo rispetto ai "ventri". La struttura di questi complessi vortici di scia è stata analizzata con precisione nel 2011 da Swanson e Isaac [6].

Traendo ispirazione dai risultati finora raggiunti invece, Rostamzadeh *et al.* [4] ha sviluppato un'alternativa all'ala "tubercled" denominata "wavy wing". I risultati raggiunti sono stati soddisfacenti e hanno sottilineato una similitudine di comportamento fisico nei due casi.

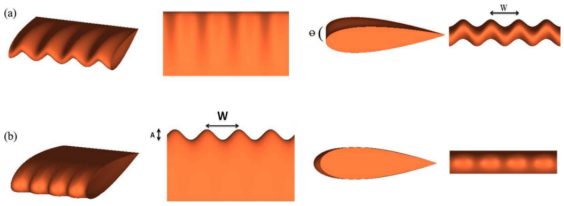

Figura 3: (a) wavy wing (b) tubercled wing [4]

1.3 Obiettivo Tesi

Obiettivo di questa tesi è impostare una simulazione fluidodinamica 3D in grado di visualizzare e analizzare il flusso attorno a un'ala con bordo di attacco sinusoidale (FIG. 3, (b) tubercled wing). La comprensione dei meccanismi fisici presenti consentirebbe infatti di sfruttare i vantaggi di questa particolare geometria (aumento dell'angolo di stallo, aumento del campo di utilizzo efficiente, ritardo dello stallo) in molti ambiti industriali, in particolare in campo aeronautico e in campo automotive. Particolare attenzione verrà posta sulla distrubuzione di pressioni sull'ala stessa e sulla presenza dei vortici di scia. Verranno usati i software:

- Gmsh, per la generazione della griglia di calcolo
- OpenFOAM, in particolare il solutore simpleFoam, per eseguire la simulazione
- ParaView per il post-processing e l'analisi dei dati

2 Impostazione fisico-matematica

2.1 Equazioni di Navier-Stokes

Un qualsiasi flusso fluido newtoniano a proprietà costanti, sotto l'ipotesi del continuo, è correttamente descritto da un sistema di equazioni differenziali alle derivate parziali dette equazioni di Navier-Stokes.

Una volta definita una generica legge di conservazione:

$$\frac{\partial q}{\partial t} + \nabla \cdot \boldsymbol{f}(q, \boldsymbol{V}) = q_v + \nabla \cdot \boldsymbol{q}_s \textit{ forma differenziale per una quantità scalare} \tag{2.1}$$

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \boldsymbol{F}(\mathbf{q}, \boldsymbol{V}) = \boldsymbol{q}_\mathbf{v} + \nabla \cdot \boldsymbol{Q}_s \textit{ forma differenziale per una quantità vettoriale} \tag{2.2}$$

Dove Q indica una generica variabile, q la variabile per unità di volume, $\boldsymbol{f}(q, \boldsymbol{V})$ il flusso di q attraverso una generica superficie nell'unità di tempo con velocità del fluido $\boldsymbol{V}$, e q_v e $\boldsymbol{q}_s$ rispettivamente le sorgenti volumiche e superficiali della variabile Q.

Le equazioni di Navier-Stokes possono essere ricavate dalle tre classiche equazioni di bilancio.

EQUAZIONE DELLA CONSERVAZIONE DELLA MASSA

"la massa non si crea e non si distrugge"

$$\frac{\partial p}{\partial t} + \nabla \cdot (\rho \boldsymbol{V}) = 0 \tag{2.3}$$

EQUAZIONE DELLA CONSERVAZIONE DELLA QUANTITÀ DI MOTO

"la variazione della quantità di moto è uguale alla somma delle forze applicate"

$$\frac{\partial \boldsymbol{V}}{\partial t} + \nabla(\frac{V^2}{2}) - \boldsymbol{V} \times \boldsymbol{\omega} = -\frac{\nabla p}{\rho} + \frac{1}{\rho}(\nabla \cdot \boldsymbol{\tau}) + \boldsymbol{f} \tag{2.4}$$

EQUAZIONE DELLA CONSERVAZIONE DELL'ENERGIA

"l'energia non si crea e non si distrugge"

$$\rho \frac{\partial E}{\partial t} + \rho \boldsymbol{V} \cdot \nabla E = -\nabla \cdot (p\boldsymbol{V}) + \nabla \cdot (\boldsymbol{\tau} \cdot \boldsymbol{V}) + \rho\, \boldsymbol{f} \cdot \boldsymbol{V} - \nabla \cdot \boldsymbol{q} + Q_v \tag{2.5}$$

Queste tre equazioni, insufficienti alla chiusura del problema della determinazione del campo di moto del fluido, vengono completate dall'aggiunta di relazioni costitutive esplicite legate alle proprietà del fluido (fluido newtoniano, ipotesi di

Stokes, legge di Fourier) e di condizionali iniziali e al contorno, trattandosi di equazioni differenziali.

Si ottiene così l'insieme delle equazioni

$$\frac{\partial p}{\partial t} + \nabla \cdot (\rho \boldsymbol{V}) = 0 \tag{2.6}$$

$$\rho \left(\frac{\partial \boldsymbol{V}}{\partial t} + \boldsymbol{V} \cdot \nabla \boldsymbol{V} \right) = -\nabla p + \nabla \cdot \left[\mu \left(\nabla \boldsymbol{V} + \nabla \boldsymbol{V}^T \right) - \frac{2}{3} \mu \left(\nabla \cdot \boldsymbol{V} \right) \boldsymbol{I} \right] + \rho \boldsymbol{f} \tag{2.7}$$

$$\rho \left(\frac{\partial E}{\partial t} + \boldsymbol{V} \cdot \nabla E \right) = -\nabla (p \boldsymbol{V}) + \nabla \cdot \left\{ \left[\mu \left(\nabla \boldsymbol{V} + \nabla \boldsymbol{V}^T \right) - \frac{2}{3} \mu \left(\nabla \cdot \boldsymbol{V} \right) \boldsymbol{I} \right] \cdot \boldsymbol{V} \right\} + \rho \boldsymbol{f} \cdot \boldsymbol{V} + \nabla \left(\kappa \nabla T \right) + Q_v \tag{2.8}$$

Nel caso di flusso incomprimibile, ipotesi soddisfatta nel nostro caso di studio, possiamo ottenere una notevole semplificazione. L'equazione dell'energia diventa superflua e , insieme alla condizione $\rho = \textit{costante}$, la conservazione della massa si riduce a

$$\nabla \cdot \boldsymbol{V} = 0 \ \textit{il campo delle velocità è solenoidale} \tag{2.9}$$

e quella della quantità di moto diventa

$$\rho \left(\frac{\partial \boldsymbol{V}}{\partial t} + \boldsymbol{V} \cdot \nabla \boldsymbol{V} \right) = -\nabla p + \nabla \cdot \left[\mu \left(\nabla \boldsymbol{V} + \nabla \boldsymbol{V}^T \right) \right] + \rho \boldsymbol{f} \tag{2.10}$$

2.2 Decomposizione di Reynolds

Le tre equazioni sopra presentate, (2.6) (2.7) (2.8), possono essere risolte analiticamente solo in casi estremamente semplificati e caratterizzati da correnti laminari. Una corrente si presenta laminare quando le perturbazioni, inevitabilmente presenti in ogni flusso non ideale, vengono smorzate. Col crescere del numero di Reynolds l'effetto smorzante della viscosità viene a diminuire e la corrente diventa turbolenta.

"Le principali caratteristiche del moto turbolento sono la presenza di fluttuazioni sia spaziali che temporali, di vorticità, di alti livelli di dissipazione e diffusività, e l'estrema non linearità del moto"[8]

Queste fluttuazioni, riguardanti tutte le quantità del campo di moto fluido, rendono il moto turbolento altamente non stazionario, tridimensionale e non prevedibile.

Si possono identificare varie regioni, in varie scale spaziali, caratterizzate da particelle fluide dotate di vorticità. Queste regioni, denominate eddy, si dividono

in eddy di grande scala, che contengono la maggior parte dell'energia cinetica, e eddy di piccola scala, dove avviene il processo di dissipazione. Il trasporto di energia da grandi scale a piccole scale viene denominato "cascata inerziale". Per maggiori informazioni riguardanti le caratteristiche del moto turbolento si rimanda il lettore a testi specifici (Stephen B. Pope, Turbulent Flows, Cambridge University Press 2000).

Un approccio numerico di calcolo (DNS, *Direct Numerical Simulation*) applicato a un flusso turbolento, come quello in esame, risulta tecnicamente molto oneroso a causa dell'elevato costo computazionale che ne deriverebbe. Viene quindi adottato un metodo che tenga in considerazione solo la componente media delle variabili, tralasciando le fluttuazioni che, come detto sopra, sono ben presenti in un campo di moto turbolento.

Un campo di moto statisticamente stazionario difatti può essere scomposto nella somma di una componente media e di una fluttuazione, intesa come "deviazione dal segnale istantaneo del valor medio". Il metodo comunemente usato è quello della *decomposizione di Reynolds.*

Considerate le componenti cartesiane, ad esempio, della velocità ; possiamo scrivere

$$u_i(\boldsymbol{x},t) = U_i(\boldsymbol{x}) + u_i'(\boldsymbol{x},t) \tag{2.11}$$

dove

$$U_i(\boldsymbol{x}) = \overline{u}_i(\boldsymbol{x}) = \frac{1}{T}\int_{t_0}^{t_0+T} u_i(\boldsymbol{x},t)dt \ \textit{campo medio stazionario} \tag{2.11}$$

e le fluttuazioni $u_i'(\boldsymbol{x},t)$ sono caratterizzate da

$$\overline{u_i'}(\boldsymbol{x},t) = 0 \tag{2.12}$$

Sostituendo espressioni analoghe per tutte le variabili nelle equazioni di Navier-Stokes si ottengono le equazioni che governano il moto medio, comunemente chiamate equazioni RANS (*Raynolds Averaged Navier-Stokes*).

$$\frac{\partial U_i}{\partial x_i} = 0 \tag{2.13}$$

$$\rho U_j \frac{\partial U_i}{\partial x_i} = -\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_i}(\tau_{ij} - \rho\overline{u_i'u_j'}) \tag{2.14}$$

Formalmente uguali alle equazioni di Navier-Stokes le RANS differiscono solamente per la derivata di $\rho\overline{u_i'u_j'}$; questo termine ancora contiene termini riguardanti le fluttuazioni del campo di moto. In particolare rappresenta gli sforzi

turbolenti medi esercitati dalle fluttuazioni turbolente sul campo di moto medio. Questo termine è un tensore detto tensore degli sforzi di Reynolds, in seguito indicato come

$$R_{ij}(\boldsymbol{X}, t) = -\rho\overline{u_i' u_j'} \tag{2.15}$$

Proprio questo termine, introducendo 6 incognite aggiuntive (tensore simmetrico), rende il sistema non chiuso. Per poter risolvere le RANS occorre introdurre delle relazioni aggiuntive per la chiusura del sistema (problema della chiusura) che "modellino" il tensore degli sforzi di Reynolds.

2.3 Modelli di turbolenza

Il problema della chiusura può essere risolto tramite diverse strategie; in questa tesi verranno analizzati i cosidetti modelli EVM *(Eddy Viscosity Model)*, in quanto maggiormente utilizzati.

L'ipotesi di Boussinesq, che per primo propose questo modello nel 1887, può essere scritta come

$$-\overline{u_i' u_j'} = 2\nu_t(\tfrac{\partial \bar{v}_i}{\partial x_j} + \tfrac{\partial \bar{v}_j}{\partial x_i}) - \tfrac{2}{3}k\delta_{ij} \tag{2.16}$$

dove ν_t è la viscosità turbolenta, $k = {}^1\!/_2\overline{u_i' u_i'}$ è l'energia cinetica turbolenta e δ_{ij} è il delta di Kronecker. Questa equazione lega parte del tensore degli sforzi di Reynolds a un secondo tensore tramite lo scalare viscosità turbolenta. Essendo quest'ultimo tensore, e l'altro termine dell'equazione, determinato dal moto medio, per la chiusura del sistema è sufficiente definire la viscosità turbolenta. Per modellare quest'ultima quantità scalare sono state presentate numerose soluzioni. Nei modelli a 0 equazioni vengono usate delle assunzioni empiriche per modellare la viscosità turbolenta; risolvendo le RANS solo per il campo mediato. Nei modelli a 1 o 2 equazioni invece vengono introdotte delle quantità turbolente, delle quali la viscosità turbolenta è funzione, per le quali è necessario risolvere delle equazioni di trasporto.

Modello di ordine 0: Mixing length di Prandtl

Non vengono introdotte ulteriori equazioni differenziali, ci si limita a definire

$\nu_t = l_m^2 \frac{\partial u}{\partial y}$;

dove l_m è definita come "lunghezza di mescolamento" e y è la distanza dalla parete

Modello di ordine 1: Spalart-Allmaras

Viene introdotta una sola equazione differenziale alle derivate parziali che consenta di descrivere l'andamento spaziale del campo di viscosità turbolenta. L'incognita dell'equazione è una variabile simile alla viscosita, $\tilde{\nu}$.

$\nu_t = \tilde{\nu} f_{v1}$

$\frac{\partial \tilde{\nu}}{\partial t} + u_j \frac{\partial \tilde{\nu}}{\partial x_j} = C_{b1}\left[1 - f_{t2}\right]\tilde{S}\tilde{\nu} + \frac{1}{\sigma}\left\{\nabla \cdot \left[\left(\nu + \tilde{\nu}\right)\nabla\tilde{\nu}\right] + C_{b2}|\nabla\nu|^2\right\} - \left[C_{\omega 1} f_\omega - \frac{C_{b1}}{\kappa^2} f_{t2}\right]\left(\frac{\tilde{\nu}}{d}\right)^2 + f_{t1}\triangle U^2$

Modello di ordine 1: Modello standard k

La variabile k ($m^2 s^{-2}$) indica l'energia cinetica turbolenta ed è ricavabile da un'equazione differenziale di trasporto. La viscosità turbolente viene definita come

$\nu_t = \sqrt{2k} l_m$

Modello di ordine 2: Modello standar k $-\epsilon$

Il più diffuso modello a 2 equazioni; si appoggia al campo scalare k (energia cinetica turbolenta, $m^2 s^{-2}$) e, in aggiunta, al campo scalare ϵ (rateo di dissipazione dell'energia cinetica turbolenta, $m^2 s^{-3}$)

$\frac{\partial(\rho k)}{\partial t} + \frac{\partial(\rho k u_i)}{\partial x_i} = \frac{\partial}{\partial x_i}\left[\frac{\nu_t}{\sigma_k}\frac{\partial k}{\partial x_j}\right] + 2\nu_t E_{ij} E_{ij} - \rho\epsilon$

$\frac{\partial(\rho \epsilon)}{\partial t} + \frac{\partial(\rho k \epsilon u)}{\partial x_i} = \frac{\partial}{\partial x_j}\left[\frac{\nu_t}{\sigma_\epsilon}\frac{\partial k}{\partial x_j}\right] + C_{1\epsilon}\frac{\epsilon}{k} 2\nu_t E_{ij} E_{ij} - C_{2\epsilon}\rho\frac{\epsilon^2}{k}$

Una volta ricavati i campi scalari di k e ϵ si ha in tutto il campo di moto

$\nu_t = \rho C_\nu \frac{k^2}{\epsilon}$

Pur essendo il modello più utilizzato grazie al suo compromesso tra accuratezza e velocità di risoluzione, presenta alcune criticità in particolare in presenza di forti gradienti avversi di pressione.

Modello di ordine 2: Modello standar k $-\omega$

Simile al precedente, mantiene l'equazione di bilancio per k e aggiunge una nuova variabile ω (frequenza della turbolenza) definita come *dissipation rate* fratto *turbulent kinetic energy*

$\frac{\partial(\rho\omega)}{\partial t} + \frac{\partial(\rho u_j \omega)}{\partial x_i} = \frac{\gamma\omega}{k} P - \beta\rho\omega^2 + \frac{\partial}{\partial x_i}\left[\left(\mu + \sigma_\omega \frac{\rho k}{\omega}\right)\frac{\partial\omega}{\partial x_i}\right] + \frac{\rho\sigma_d}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial\omega}{\partial x_i}$

Questo modello è generalmente più affidabile del modello $k - \epsilon$; si comporta meglio in prossimita di pareti.

MODELLO DI ORDINE 2: Modello standard SST (*Shear Stress Transport*)

Questo modello riesce a combinare il meglio dei due modelli precedenti. Il modello $k - \omega$, dato il suo buon funzionamento a parete ma l'eccessiva sensibilità a un flusso libero, viene utilizzato nello strato limite. All'esterno di questa regione viene invece utilizzato il modello $k - \epsilon$. Questo modello è largamente utilizzato in quanto fornisce buoni risultati anche per bassi numeri di Reynolds, gradienti avversi di pressione e separazione del flusso.

3 Strumenti di Analisi

3.1 Introduzione al CFD

La fluidodinamica computazionale (CFD, *Computational Fluid Dynamics*) è l'analisi dei sistemi che trattano movimenti fluidi, scambio di calore e reazioni chimiche, attraverso l'uso di simulazioni effettuate col computer.

Le equazioni di Navier-Stokes, risolvibili analiticamente solo in casi molto semplici, vengono qui risolte tramite un approccio numerico. I modelli attraverso cui risolvere queste equazioni possono avere un costo computazionale più o meno elevato, i principali sono:

- *Direct Numerical Simulation* (DNS); si discretizzano lo spazio e il tempo con delle griglie di dimensione variabile e si eseguono i calcoli su di esse. E' il metodo che favorisce i risultati più precisi e accurati, ma implica un elevatissimo costo computazionale.
- *Large Eddy Simulation* (LES); i fenomeni turbolenti vengono divisi in scale spaziali. I comportamenti delle scale turbolente di maggiori dimensioni vengono calcolati numericamente, mentre quelli delle scale più piccole vengono modellate. Più accurato delle RANS, ha tuttavia un costo computazionale molto inferiore a quello del DNS.
- *Reynolds Averaged Navier-Stokes* (RANS); consiste nel risolvere le equazioni mediate che sono state analizzate nel precedente capitolo. E' necessario introdurre dei modelli di turbolenza per la chiusura del sistema.

Il primo passo necessario per esaminare un qualsiasi problema fisico, simulandolo attraverso l'uso di un calcolatore, è la discretizzazione; ovvero il processo di trasformazione di modelli a equazioni continue in equazioni discrete. Il metodo ai volumi finiti è il più utilizzato dai codici CFD. Questo metodo permette di integrare le equazioni differenziali alle derivate parziali in un volume di controllo discreto, sui cui bordi sono imposte delle condizioni al contorno.

L'approccio tipico seguito da un programma CFD consiste in 5 punti principali:

1. discretizzazione del dominio di calcolo (creazione mesh)
2. scelta del modello fisico

3. scelta del modello numerico
4. imposizione delle condizioni al contorno ed, eventualmente, delle condizioni iniziali
5. risoluzione iterativa delle equazioni

In questo modo gli analisti possono avere una predizione qualitativa (e a volte anche quantitativa) del fenomeno studiato. Possono essere eseguiti degli "esperimenti numerici" in sostituzione di quelli reali, con tutti i vantaggi del caso.

L'uso del CFD si è diffuso, a partire dalle prime applicazioni e sviluppi in ambito aerospaziale duranti gli anni '70, in ogni ramo della progettazione ingegneristica e architettonica, ma anche in ambito medico e chimico. Questi esperimenti numerici danno la possibilità di avere una visualizzazione del flusso fluido che sarebbe difficile da ottenere con un esperimento condotto coi metodi più tradizionali. Le predizioni fornite da un computer sul comportamento di un fluido possono riguardare tutte le quantità legate a quel fluido e possono essere condotte in condizioni e dominio uguali a quelli del fenomeno studiato. A differenza di esperimenti condotti in laboratorio, limitati alla misurazione e visualizzazione solo di alcune grandezze; in condizioni simili a quelle di studio, ma non identiche, in quanto scalate per essere eseguibili in laboratorio.

Le maggiori criticità di questo potentissimo strumento si possono sicuramente identificare nelle numerose sorgenti di errore. Un'errata scelta del modello fisico, o dell'algoritmo di risoluzione ad esempio, può portare a una scorretta rappresentazione del fenomeno e quindi a dei risultati inaffidabili.

3.2 OpenFOAM

OpenFOAM (*Open Field Operation and Manipulation*) è una libreria C++ open source in grado di creare eseguibili chiamati *applications*. Il suo principale compito è quello di risolvere equazioni differenziali alle derivate parziali; può quindi trovare applicazioni nei più svariati campi di studio (finanza, dinamica molecolare, flussi multifasici) pur essendo utilizzato principalmente per problemi CFD. Le "applications" creati da OpenFOAM si dividono in due categorie:

- *solvers*; creati per risolvere uno specifico problema nell'ambito della meccanica dei continui

- *utilities*; create per eseguire compiti riguardanti la manipolazione di dati

Una delle maggiori potenzialità di OpenFOAM risiede nel rendere visibile agli utenti il proprio codice sorgente. In questo modo chiunque, con le dovute conoscenze, potrebbe partire da una solida base funzionante per creare i solvers e le utilities più consone alle sue esigenze.

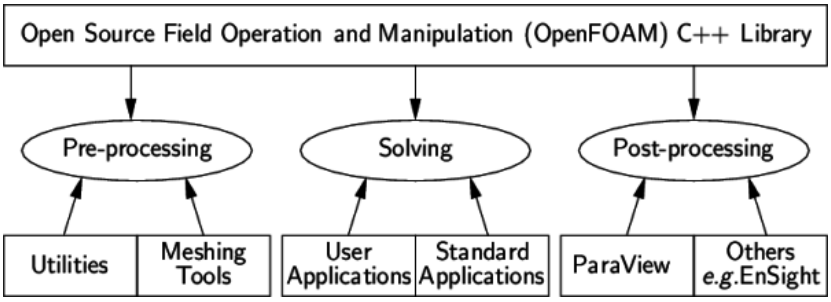

Figura 4: Struttura di OpenFOAM [10]

OpenFOAM è corredato da un ambiente di pre- e post-processing gestito da delle particolari utilities. Moltissimi software di terze parti sono interfacciabili con OpenFOAM tramite queste utilities; *e.g.* Paraview e Gmsh, utilizzati in questa tesi rispettivamente per l'analisi dei dati successiva alla simulazione e la creazione della mesh.

Caratteristica fondamentale di OpenFOAM, essendo scritto in linguaggio C++, è la programmazione *object-oriented.* In questo tipo di programmazione vengono introdotte le *classi,* oggetti in cui è possibile raggruppare dati e funzioni, in grado di interagire tra di loro attraverso lo scambio di messaggi. In questo modo la scrittura delle equazioni risulta molto più simile al linguaggio matematico, e l'intero codice risulta molto più comprensibile, compatto e meno ripetitivo.

3.3 Struttura di un caso di OpenFOAM

Un generico caso di studio in OpenFOAM è composto da una cartella al cui interno è possibile trovare tutte le informazioni necessarie alla risoluzione del problema, e i risultati una volta eseguiti i calcoli. Prima di eseguire una simulazione, all'interno della cartella del caso si trovano tre sottocartelle principali. A titolo d'esempio verranno analizzati i file presenti nella cartella del tutorial *cavity* presentato sul sito di OpenFOAM

`OpenFOAM/run/tutorials/incompressible/icoFoam/cavity`

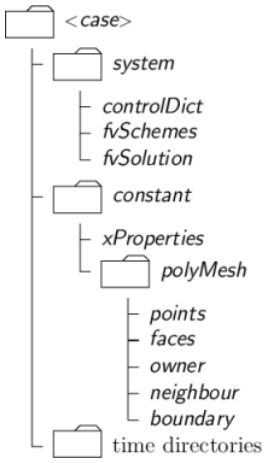

Figura 5: Struttura cartelle OpenFOAM [10]

3.3.1 Cartella "0"

All'interno di questa cartella è possibile trovare i file che descrivono le condizioni iniziali di tutte le variabili di interesse. Il file riguardante U, il campo vettoriale della velocità, si presenta in questo modo:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}

dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
    movingWall
    {          type            fixedValue;
               value           uniform (1 0 0);
    }
    fixedWalls
    {          type            fixedValue;
               value           uniform (0 0 0);
```

```
    }
    frontAndBack
    {           type            empty;
    }
}
```

In cui viene inizialmente definita la variabile, il campo vettoriale U, e la sua unità di misura. Viene poi specificato il valore della velocità all'istante iniziale, prima nel campo interno (in questo caso tutte e tre le componente del vettore valgono $0^{m}/_{s}$) e successivamente agli estremi del dominio. Nella cartella "0" è presenta un file per ogni variabile (U, p, epsilon, k ecc....).

3.3.2 Cartella "constant"

Contiene una cartella *"polyMesh"* in cui sono presenti le informazioni riguardanti la griglia di calcolo. Questa cartella può essere creata o tramite l'utility *blockMesh,* che permette di creare una mesh a partire da un file di controllo (*blockMeshDict*), o tramite alcune utilities di importazione di mesh in OpenFOAM. Uno dei principali file presenti nella cartella *polyMesh* è il file *"bundary"*

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}

3
(
    movingWall
    {
        type            wall;
        inGroups        1(wall);
        nFaces          20;
        startFace       760;
    }
    fixedWalls
    {
        type            wall;
        inGroups        1(wall);
```

```
        nFaces          60;
        startFace       780;
    }
    frontAndBack
    {
        type            empty;
        inGroups        1(empty);
        nFaces          800;
        startFace       840;
    }
)
```

che definisce gli estremi del dominio di calcolo, gli assegna un nome (movingWall, fixedWalls ecc...) e ne determina il tipo. In questo esempio, essendo un caso bi-dimensionale, la faccia anteriore e la faccia posteriore della mesh sono definite come *"empty"*, mentre gli altri elementi sono dei semplici "muri".

In aggiunta alla cartella "*polyMesh*" possono essere presenti degli altri file. I più importanti sono sicuramente: *"transportProperties"*, che determina il comportamento viscoso del fluido; *"RASProperties"*, che gestisce il tipo di simulazione e i modelli di turbolenza.

3.3.3 Cartella "system"

Questa cartella contiene una serie di file, detti dictionaries e denominati dal suffisso *Dict*, volti al controllo del processo di risoluzione. I principali dictionaries che possiamo trovare in questa cartella sono sicuramente: *"controlDict"*, *"fvSchemes"* e *"fvSolution"*.

Il primo file si presenta così

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}

application     icoFoam;
startFrom       startTime;
```

```
startTime       0;
stopAt          endTime;
endTime         0.5;
deltaT          0.005;
writeControl    timeStep;
writeInterval   20;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
```

Dopo la scelta del solutore (*icoFoam*, solver per flussi non stazionari incompressibili e laminari, nell'esempio) vengono scelti parametri riguardanti l'integrazione temporale, la lettura e la scrittura dei risultati.

Il file *"fvSchemes"* determina quali metodi numerici applicare per ogni termine presente nelle equazioni del caso.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}

ddtSchemes
{     default          Euler;
}

gradSchemes
{
    default          Gauss linear;
    grad(p)          Gauss linear;
}
divSchemes
{
    default          none;
    div(phi,U)       Gauss linear;
}
```

```
laplacianSchemes
{
    default         Gauss linear orthogonal;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         orthogonal;
}
fluxRequired
{
    default         no;
    p               ;
}
```

Mentre il file *"fvSolution"* controlla i solutori delle equazioni, le tollerenze per arrivare a convergenza e gli algoritmi di risoluzione.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}

solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;
    }
    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
```

```
        relTol          0;
    }
}
PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

Per l'analisi dettagliata di tutti i file presenti in un caso si rimanda alla guida di OpenFOAM [10].

Oltre a queste tre cartelle principali, col proseguire della simulazione si verranno a creare tante cartelle quante sono le iterazione fatte dal solutore (per un *timeStep* di 0.1 ad esempio avremo le cartelle "0.1", "0.2","0.3", ecc...), contenenti ognuna i valori assunti dalle variabili presenti nella cartella "0" in ogni punto del dominio di calcolo, in un determinato momento. Le informazioni presenti in ogni cartella potranno poi essere visualizzate tramite apposite applicazioni di post-processing.

3.4 Principali solver OpenFOAM

I principali solutori utilizzati da OpenFOAM nell'ambito del CFD sono sotto riportati. Le definizioni sono riportate integralmente dal sito ufficiale di OpenFOAM [11].

- *potentialFoam*; simple potential flow solver which can be used to generate starting fields for full Navier-Stokes codes
- *icoFoam*; transient solver for incompressible, laminar flow of Newtonian fluids
- *pisoFoam*; transient solver for incompressible flow
- *simpleFoam*; steady-state solver for incompressible, turbulent flow
- *rhoSimpleFoam*; steady-state SIMPLE solver for laminar or turbulent RANS flow of compressible fluids

In questa tesi verrà utilizzato inizialmente *potentialFoam* per risolvere in breve tempo il campo potenziale del caso; successivamente i risultati ottenuti saranno usati come condizione iniziale per il solutore *simpleFoam*.

3.5 Gmsh

Gmsh è un generatore di mesh open source composto da 4 moduli principali: geometry description, meshing, solving e post-processing. Questo programma accetta due modalita di input. La prima, *Interactive mode*, consiste nell'usare un'interfaccia grafica per definire le varie entità della mesh (point, line, line-loop ecc...). La seconda invece, *Non-Interactive mode*, consiste nel creare un file di testo (con un linguaggio simile al C) per definire la geometria; questa modalità si rivela particolarmente utile nel creare geometrie parametriche.

Una volta creato il file di testo è possibile aprirlo con Gmsh e effettuare il meshing. Il programma dà la possibilità di scegliere tra vari algoritmi (sostanzialmente *Delaunay* e *Frontal*) e fornisce anche alcuni strumenti di ottimizzazione. Nel prossimo capitolo verranno analizzati in ogni particolare i file usati per le simulazioni.

4 Impostazione del caso

4.1 Mesh

Per creare la mesh dell'ala "tubercled" è stata utilizzata la *Non-Interactive mode* di Gmsh. Il file .geo creato può essere trovato nella Appendice I di questa tesi.

Nella prima parte di questo documento vengono per prima cosa inclusi due file contenenti i vettori di punti di due profili NACA (0021 profilo principale, 0023 profilo boundary layer). Poi vengono definiti i parametri principali della mesh; per la scelta di parametri come apertura alare, corda, e ampiezza della sinusoide si è preso spunto dal documento di Rostamzadeh *et al.* [4].

In seguito si prosegue alla costruzione 2D del profilo dell'ala e del profilo del boundary layer; a partire da questi due profili vengono copiati i profili all'altra estremita dell'ala e anche i profili (opportunamente ridimensionati) nella stazione intermedia. I 3 profili vengono uniti per creare leading edge e trailing edge.

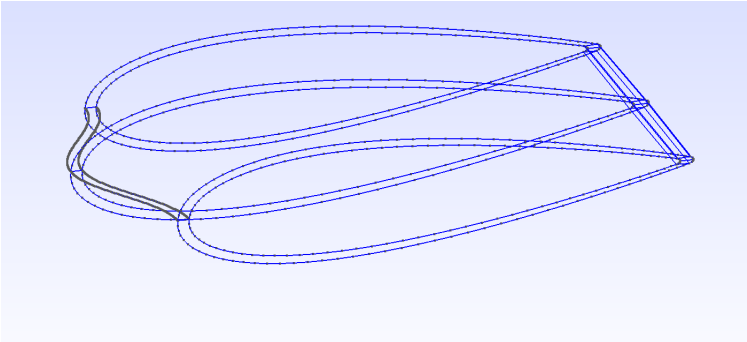

Figura 6: Tre stazioni principali Ala "tubercled" 1D

Per la creazione del leading edge sono stati usati 4 cicli FOR per definire 100 punti secondo un andamento sinusoidale, successivamente uniti tramite una spline.

Inizialmente la prima versione di questa mesh presentava un bordo di fuga con punto angoloso, come un classico profilo NACA. In seguito a dei problemi nella creazione della mesh 2D strutturata si è optato per un bordo arrotondato. Per crearlo, a causa di alcune limitazione nella definizioni delle superfici di Gmsh, è stato necessario dividere ulteriormente le due superfici dorsale e ventrale nella loro parte finale (FIG. 8).

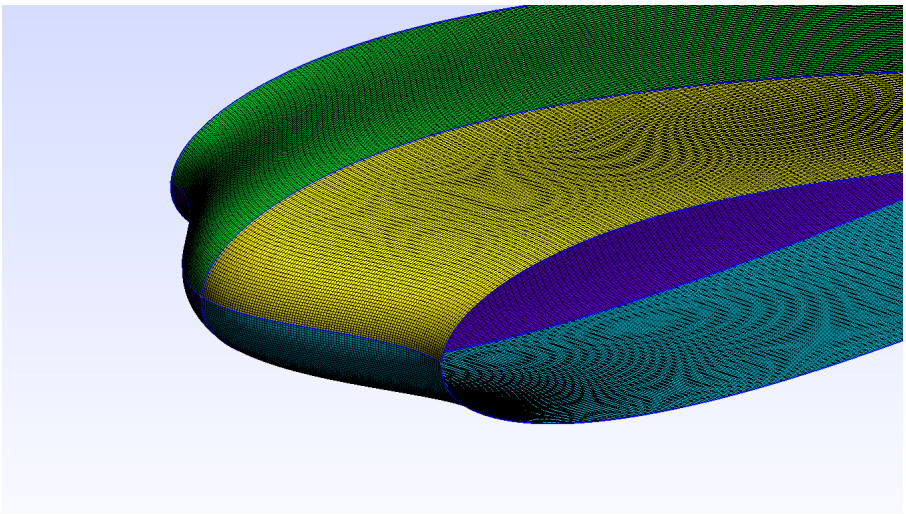

Figura 7: Superfice alare Ala “Tubercled” 2D

E’ stata scelta una mesh ibrida strutturata-non strutturata; la zona strutturata posta in prossimità dell’ala permette di avere un’ottima densita di celle, in particolare in direzione normale alla superfice (dove si hanno i maggiori gradienti di velocita), senza però averne un numero eccessivo. Una risoluzione equivalente si sarebbe potuta raggiungere anche con degli elementi tetraedrici, però il numero di elementi sarebbe stato decisamente superiore, appesantendo la mesh. Una volta definite tutte le superfici necessarie è stato quindi possinile definire i 4 volumi strutturati contenenti il boundary layer.

L’ultimo passo è stato definire il dominio esterno, diviso in due zone. La zona rettangolare più interna è caratterizzata da una dimensione caratteristica minore, in modo da rappresentare con una maggiore densità di celle una zona critica in prossimità dell’ala. Alla fine del documento sono state definite le cosiddette “Physical Entities”, ovvero le definizioni dei boundaries che saranno successivamente riconosciuti da OpenFOAM.

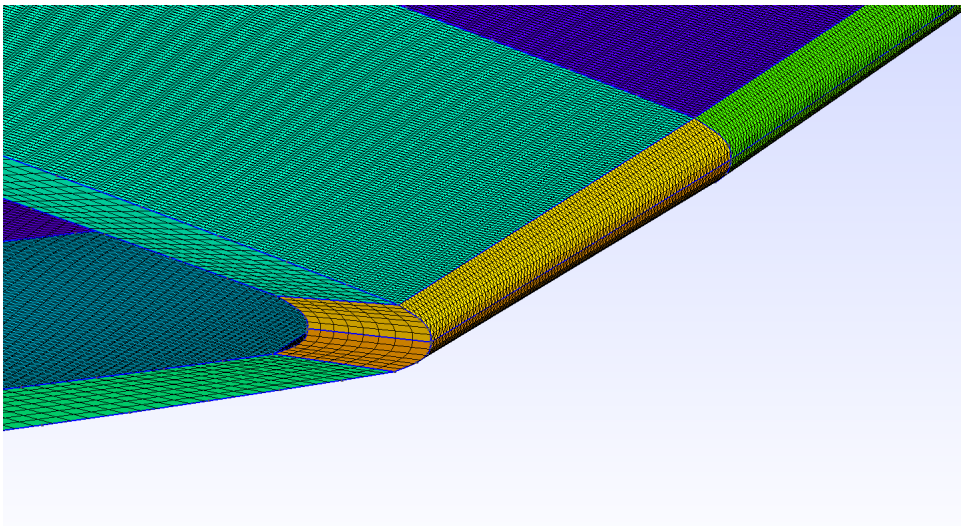

Figura 8: Particolare bordo di fuga arrotondato

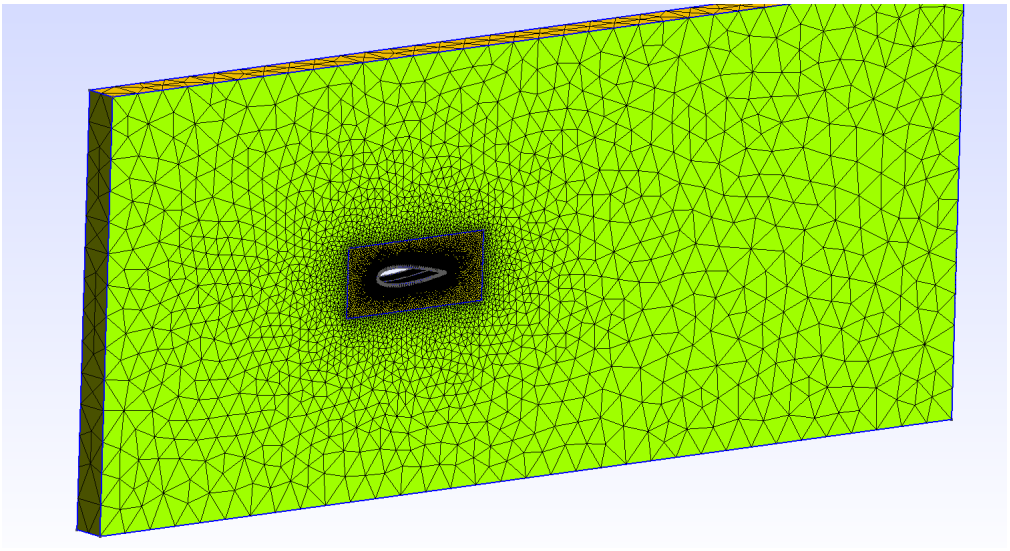

Figura 9: Visione d'insieme Mesh 2D Surface

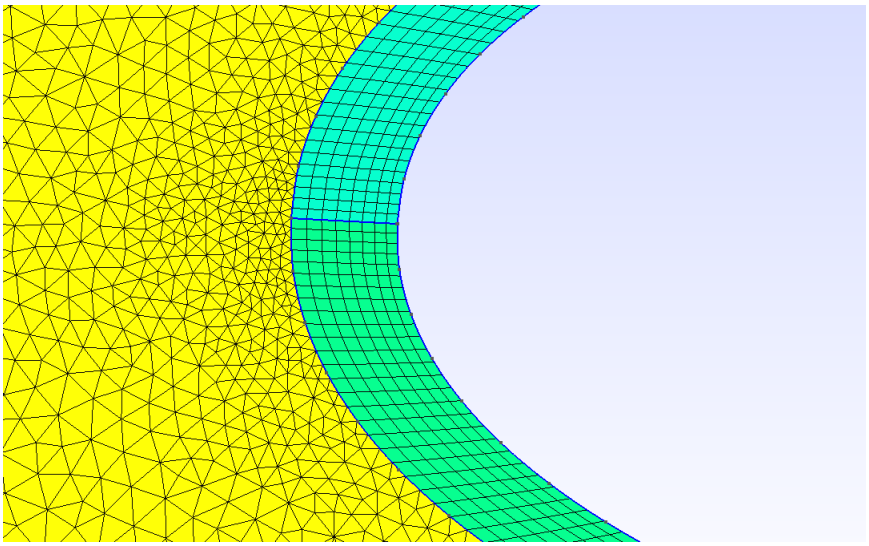

Figura 10: Particolare boundary layer strutturato

4.2 Cartelle caso OpenFOAM

Si inizia ora l'analisi di tutti i file presenti nella cartella di OpenFOAM del nostro caso. Tutti i file di testo a cui si fa riferimento in questa sezione possono essere trovati nella Appendice I.

4.2.1 Cartella "0"

Qui sono presenti tutti i file riguardanti le condizioni al contorno del caso. Nel file riguardante U viene definita la velocità iniziale nel campo interno, velocità che coincide con quella (fissa) presente al boundary *inlet*. Sui muri superiore e inferiore della nostra "galleria" viene imposta la condizione *slip*, condizione che permette al codice di non considerare gli effetti viscosi su quel bordo del dominio. Invece sulla nostra ala viene imposta una *no-slip condition*, una condizione di aderenza, obbligando il vettore velocità ad assumere il valore (0 0 0) su quel bordo. Infine sui muri anteriore e posteriore è stata imposta la condizione *symmetryPlane,* condizione che verrà spiegata successivamente.

Nel file p con il comando *zeroGradient* si impone che il gradiente normale a una superfice della pressione sia uguale a zero. Essendo interessati solamente alle differenze di pressione nel dominio, piuttosto che al valore corretto della pressione, è stato possibile imporre la pressione uguale a zero all'*outlet* (ovvero all'uscita della "galleria") e la condizione *zeroGradient* all'ingresso (suppongo costante il valore della pressione all'*inlet*).

Nei file *nut* e *nuTilda* sono inizializzate le due variabili ν_t e $\tilde{\nu}$, ovvero la viscosità turbolenta e la variabile di Spalart-Allmaras. L'inizializzazione viene effettuata imponendo il valore 0 sui vari muri e un valore piccolo (ma non uguale a zero) nelle regioni di flusso libero, comprese *inlet* e *outlet*. Questi valori sono usati solo come punto di partenza per la simulazione, e assumeranno valori più precisi col procedere del calcolo.

Parallelamente nei file k e *epsilon* vengono inizializzate queste due variabili; le formule usate per l'inizializzazione sono

$k = \frac{3}{2}(UI)^2$

dove I indica l'intensità turbolenta, considerata pari all'1% come fatto da Rostamzadeh *et al.* [4].

$\epsilon = C_\mu \frac{k^{\frac{3}{2}}}{l}$, $l = 0.038 d_h$

dove l è definita come "turbulent lenght scale" [9], C_μ è costante e vale 0.09 e d_h è il diametro idraulico dela nostra "galleria".

4.2.2 Cartella "constant"

Nel file *boundary* tutti i bordi sono stati definiti come semplici muri, tranne *inlet* e *outlet* definiti come *patch,* e *front* e *back* definiti come *symmetryPlane*. Questa particolare dicitura permette a OpenFOAM di interpretare la superfice a cui è applicata come un piano di simmetria. Questo vuol dire che se al codice serve il valore di una qualsiasi variabile su una faccia della superfice in questione, questo valore viene preso in maniera simmetrica sulla faccia opposta. Questo particolare *base type* è stato utilizzato in modo combinato su due superfici opposte, in modo da ottenere un'ala di apertura alare infinita. Ovviamente l'ala definita dalla mesh tra questi due *symmetryPlane* dovrà essere simmetrica.

Negli altri due file presenti viene scelto il modello di turbolenza Spalart-Allmaras (e successivamente k-ϵ) e viene definito il valore della viscosità cinematica.

4.2.3 Cartella "system"

Nel file *controlDict* viene scelto il solver *simpleFoam*; questa scelta è motivata dal fatto che il flusso studiato presenterà sicuramente dei moti turbolenti, e avendo imposto una velocità all'ingresso pari a 25 $^m/_s$, il flusso potrà essere considerato incompressibile. Questo solver restituirà una soluzione *steady-state*, ovvero una volta arrivato a convergenza mostrerà una situazione stabile, di equilibrio nel flusso. *simpleFoam* è stato utilizzato in combinazione con due diversi modelli di turbolenza. Inizialmente la simulazione è stata portata avanti col modello Spalart-Allmaras; successivamente i risultati raggiunti con questo modello sono stati usati come condizioni iniziali per la simulazione col modello k-ϵ. Questo procedimento è stato seguito in quanto i risultati dati dal k-ϵ sono più affidabili rispetto agli altri; però questo modello richiede una certa precisione nelle condizioni iniziali, tanto da divergere nel caso in cui sia eseguito per primo, senza i risultati di Spalart-Allmaras come punto di partenza. In questo file vengono ancora impostati i parametri temporali e, nell'ultima riga, viene incluso un'altro file, *forceCoeffs*, sempre incluso in questa cartella.

In questo file vengono fornite le istruzioni necessarie al calcolo delle forze e dei loro coefficienti; si sceglie la physical entities su cui calcolare queste forze

(*wing*), la direzione dei vettori forza e resistenza, la coordinate del fuoco e infine dei parametri fisici.

Nel file *decomposeParDict* si divide il dominio di calcolo in 8 regioni, ognuna da calcolare tramite un diverso processore. Il metodo di suddivisione è un semplice criterio geometrico che impone la suddivisione dello spazio in due regioni per ogni direzione x,y e z; per un totale di 8.

All'interno di *fvSchemes* sono stati scelti i metodi numerici da applicare al caso; principalmente sono stati usati due schemi: uno di interpolazione lineare e un *Upwind*, un metodo di discretizzazione usato per risolvere equazioni differenziali alle derivate parziali iperboliche.

Infine il file *fvSolution* determina i solutori delle equazioni e controlla il valore target dei residuals necessario a fermare il calcolo in quanto arrivato a "convergenza". I residuals rappresentano l'errore assoluto nella soluzione dell'equazione alle differenze finite collegata a una particolare variabile[8]. Raggiunto un livello successivamente basso (nella nostra simulazione il livello target è stato impostato per tutte le quantità a 10^{-4}) la simulazione è arrivata a convergenza, ovvero il solver impostato ha risolto in maniera corretta le sue equazioni differenziali.

4.3 Calcolo

Il caso è stato svolto seguendo alcuni parametri usati da Rostamzadeh *et al.* [4], ovvero:

- corda 70 mm
- lunghezza d'onda del bordo di attacco sinusoidale 30 mm
- ampiezza picco-picco 8 mm
- velocità flusso indisturbato 25 $^{m}/_{s}$
- numero di Reynolds $1.17 \cdot 10^5$
- angolo di incidenza di 3 *deg*

Il calcolatore utilizzato per lo svolgimento delle simulazioni è stato un Cluster fornito dal Politecnico di Torino; in particolare le simulazioni sono state eseguite in parallelo su 8 processori diversi.

Il primo passo è stato creare e ottimizzare (Netgen) la mesh a partire dal file .geo attraverso il programma Gmsh. Questa mesh è stata poi convertita in un formato accettabile da OpenFOAM tramite il comando *gmshToFoam* e successivamente controllata con *checkMesh*. Dopo degli iniziali problemi di skewness e high aspect ratio, la mesh è stata accettata restituendo solo un warning relativo alla presenza di alcune facce "severely non-orthogonal (> 70 degrees)".

In seguito, dopo aver diviso il dominio di calcolo in 8 regioni, una per ogni processore, tramite il comando *decomposePar* , è stato possibile iniziare il calcolo vero e proprio.

```
mpirun -np 8 simpleFoam -parallel > log.simpleFoam &
```

Questa comando, inserito in un terminale con accesso alla cartella del caso, permette di far partire il solutore *simpleFoam* in parallelo su 8 processori e impone la scrittura dell'output del terminale in un file log. Questo file, in continuo aggiornamento durante lo svolgimento dei calcoli, si presenta in questo modo

```
Time = 0.4
smoothSolver:  Solving for Ux, Initial residual = 0.0547988,
Final residual = 0.00282415, No Iterations 4
smoothSolver:  Solving for Uy, Initial residual = 0.0751674,
Final residual = 0.0048384, No Iterations 4
smoothSolver:  Solving for Uz, Initial residual = 0.197378,
Final residual = 0.006533, No Iterations 4
GAMG:  Solving for p, Initial residual = 0.046827,
Final residual = 0.00452453, No Iterations 5
GAMG:  Solving for p, Initial residual = 0.106932,
Final residual = 0.00632988, No Iterations 1
GAMG:  Solving for p, Initial residual = 0.0183111,
Final residual = 0.00116971, No Iterations 2
time step continuity errors : sum local = 0.0644714,
global = -0.00216123, cumulative = -0.00256467
smoothSolver:  Solving for nuTilda, Initial residual = 0.0563086,
Final residual = 0.00358087, No Iterations 4
ExecutionTime = 56.48 s  ClockTime = 57 s
forces forces output:
sum of forces:         pressure : (-262.926 -182.108 -0.612578)
                       viscous  : (0.0336523 -0.00521239 6.69333e-06)
                       porous   : (0 0 0)
sum of moments:        pressure : (-0.00507625 0.000272072 -3.5914)
                       viscous  : (-9.63938e-08 -3.43654e-07 -0.0001127)
                       porous   : (0 0 0)
```

```
forceCoeffs forceCoeffs output:
    Cm     = -83.2756
    Cd     = -419.657
    Cl     = -290.71
    Cl(f)  = -228.63
    Cl(r)  = -62.0793
```

Per ogni iterazione il *Time* aumenta. La simulazione verrà interrotta solo nel caso in cui si raggiunga l'*endTime* impostato, oppure nel caso in cui i residuals sopra indicati raggiungano il valore indicato nel file *fvSolution*. Al termine della simulazione è necessario riunire i dati eleborati separatamente da ogni processore, e immagazzinati in 8 cartelle all'interno del caso, tramite il comando *reconstructPar*.

Una volta eseguito il caso col modello Spalart-Allmaras, la procedura viene ripetuta allo stesso modo per usare il k-ϵ. Il caso viene svolto nella stessa cartella in modo da partire (si noti la dicitura *startFrom latestTime* nel file *controlDict)* dai risultati ottenuti dall'altro modello; ovviamente alcuni dettagli nei files del caso sono stati cambiati (file *RASProperties, fvSolution, fvSchemes*).

Parallelamente al caso dell'ala con bordo di attacco sinusoidale è stata svolta anche una simulazione di un'ala infinita con la stessa corda di 70 mm e profilo NACA0021, inclinata anch'essa di 3 *deg*. Sia la mesh che l'impostazione del caso in OpenFOAM sono identiche a quelle sopra trattate. I risultati di questa simulazione saranno utilizzati come riferimento per interpretare i risultati forniti dall'ala "tubercled".

5 Risultati

5.1 Validazione

Per valutare la bontà dell'impostazione del caso, sia nella scelta delle condizioni iniziale, sia nella scelta del modello matematico e numerico, sono stati usati i dati forniti da Rostamzadeh *et al.* [4] nella parte IIIB, FIG. 10.

I valori di C_l e C_d trovati simulando un'ala A8W30 (tubercoli, ampiezza picco-picco 8mm, lunghezza d'onda 30mm) tramite il modello k-ϵ sono

- $C_d = 0.0545$
- $C_l = 0.2392$

e sono coerenti con quelli sopracitati. Il C_l corrisponde a quello indicato nell'articolo, mentre il C_d risulta leggermente superiore. Tale errore è probabilmente causato da un modello fisico non perfetto o da una mesh non sufficientemente fine. Si ricorda che i dati forniti da Rostamzadeh *et al.* [4] sono stati trovati sperimentalmente.

5.2 Distribuzione di pressione

Come si può notare in FIG. 11 e in FIG. 12 la distribuzione di pressione sul dorso dell'ala dotata di tubercoli è molto particolare e si discosta nettamente da quella classica dell'ala retta. Nel caso dell'ala coi tubercoli si possono identificare due zone: la zona dietro ai ventri della funzione sinusoidale che descrive il bordo di attacco, e quella dietro alle creste. La prima zona è caratterizatta da una pressione inferiore al valore, che possiamo considerare medio, assunto nella stessa zona dell'ala retta. La seconda zona, dietro alle creste, è invece caratterizzata da una pressione mediamente più alta.

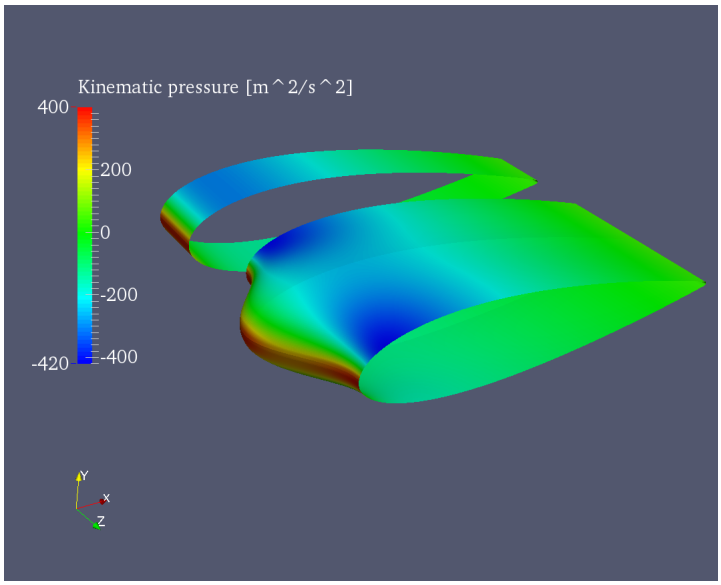

Figura 11: Vista in prospettiva della distribuzione di pressione cinematica su ala con tubercoli e ala retta

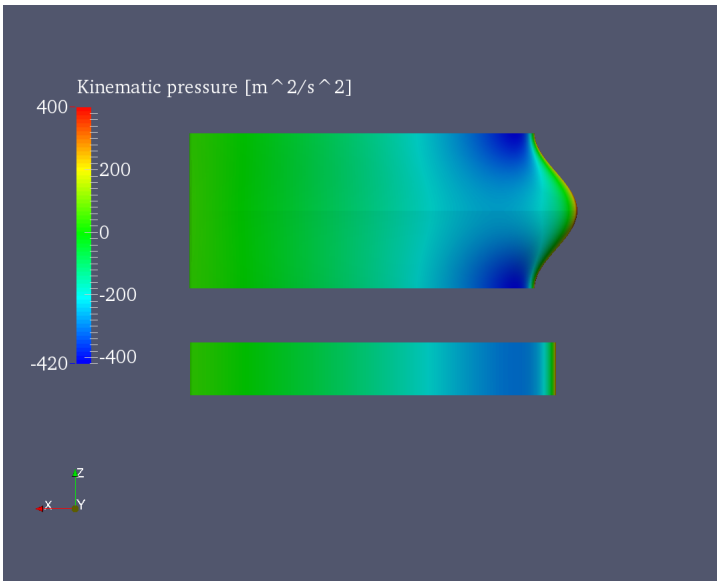

Figura 12: Vista in piante della distribuzione di pressione cinematica su ala con tubercoli e ala retta

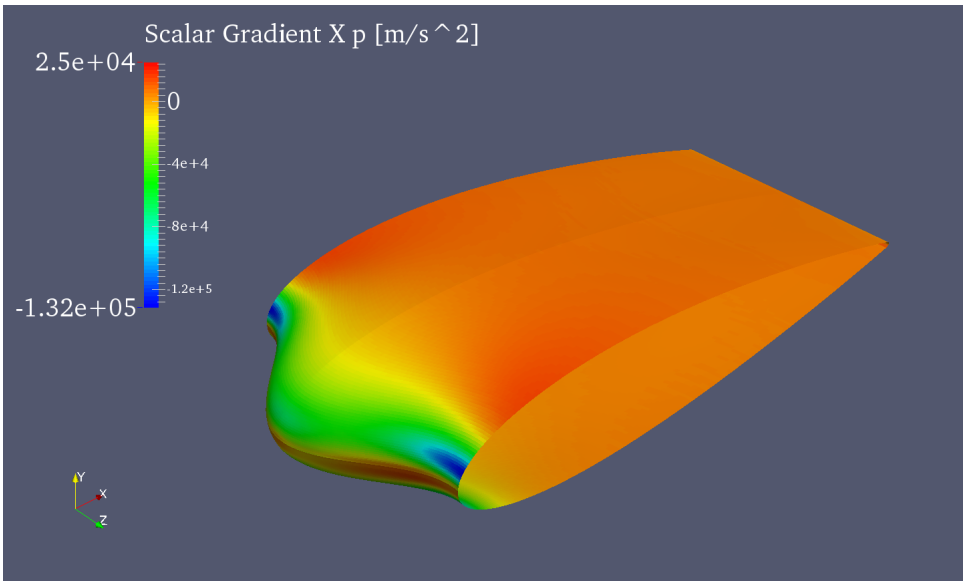

Figura 13: Vista in prospettiva della distribuzione del gradiente di pressione lungo l'asse x

Essendo la pressione così distribuita, avrà un andamento particolare anche il gradiente della pressione (FIG. 13) lungo la direzione x, ovvero il gradiente di pressione avverso, il responsabile della separazione dei filetti fluidi. Il gradiente avrà un valore più elevato in corrispondenza dei ventri della funzione sinusoidale a causa di due fattori: una pressione maggiore in valore assoluto rispetto alla zona dietro alle creste, una minor lunghezza su cui "distribuire" il salto di pressione. Infatti in questa zona la corda assume il valore $c - A/2$ dove c è la corda del profilo e A l'ampiezza della sinusoide.

Vengono inserite anche le immagini riguardanti le distribuzioni di pressioni su piani verticali passanti per: cresta del bordo di attacco, ventre del bordo di attacco, bordo di attacco dell'ala retta.

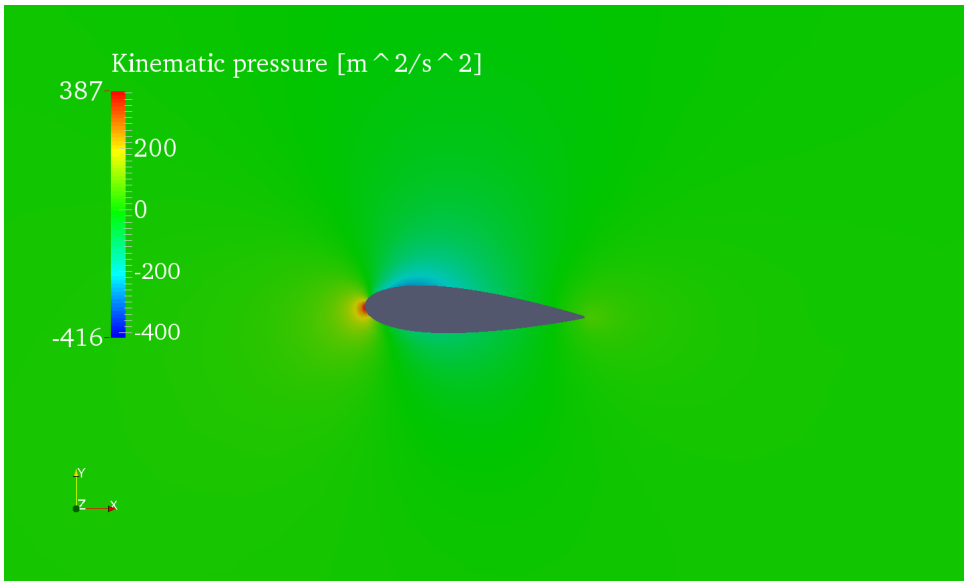

Figura 14: Distribuzione di pressione su un piano verticale passante per la cresta del bordo di attacco dell'ala coi tubercoli

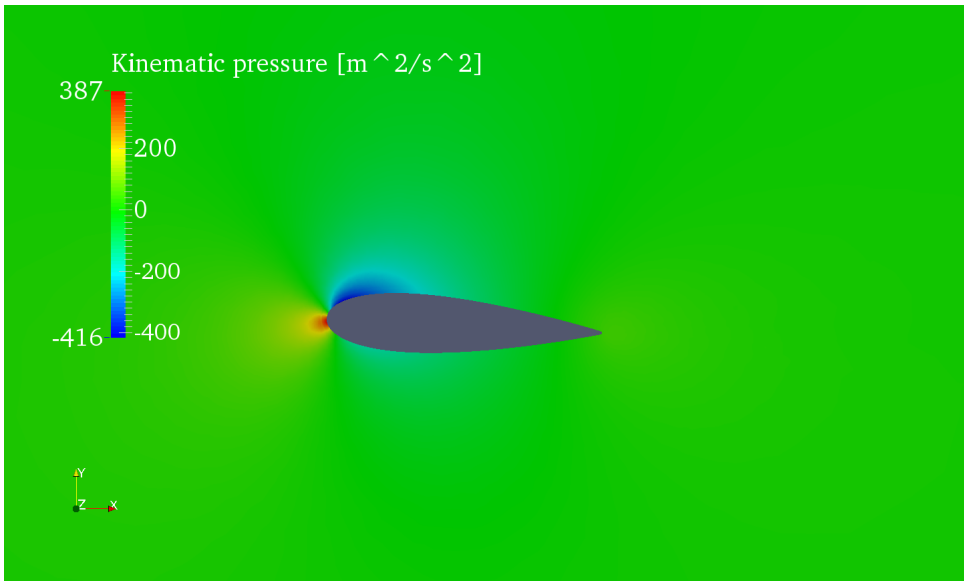

Figura 15: Distribuzione di pressione su un piano verticale passante per il ventre del bordo di attacco dell'ala coi tubercoli

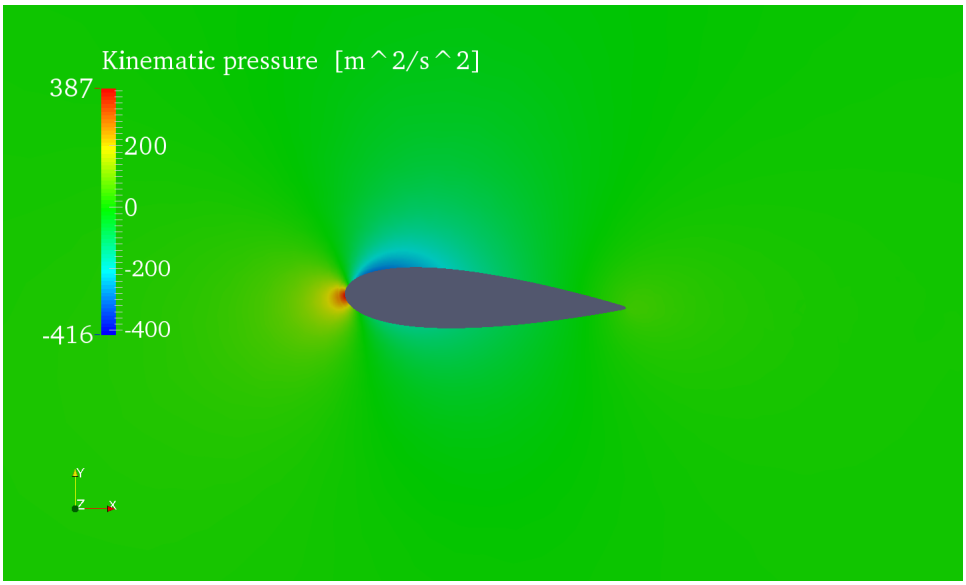

Figura 16: Distribuzione di pressione su un piano verticale secante l'ala retta

Anche attraverso le ultime 3 immagini è possibile evidenziare quanto sottolineato in precedenza: la pressione è inferiore nella zona dietro i ventri della funzione sinusoidale, viceversa per le creste. Una situazione intermedia tra le due si trova nel piano secante l'ala retta.

5.3 Distribuzione di velocità

Le distribuzioni di velocita si presentano ovviamente collegate a quelle sopra riportate riguardanti le pressioni. Dove la pressione è minore, ovvero in prossimità dei ventri, la velocità è superiore; e viceversa. Si noti con particolare attenzione come la scia nella FIG. 17-19 , riferita alla stazione che presenta la cresta del'ala, risulti piu sottile sia rispetto a quella riferita al ventre, sia rispetto all'ala retta.

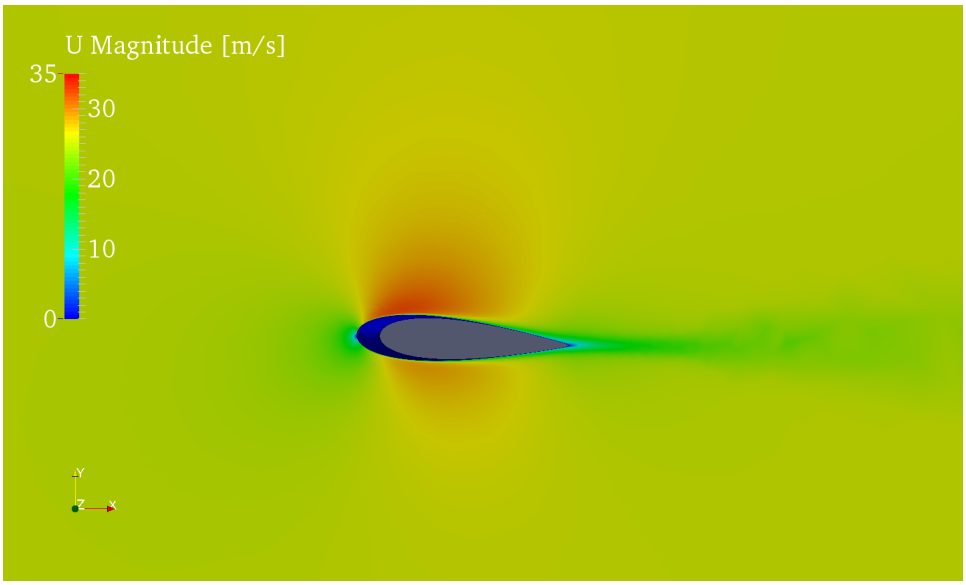

Figura 17: Distribuzione di velocità su un piano verticale passante per la cresta del bordo di attacco dell'ala coi tubercoli

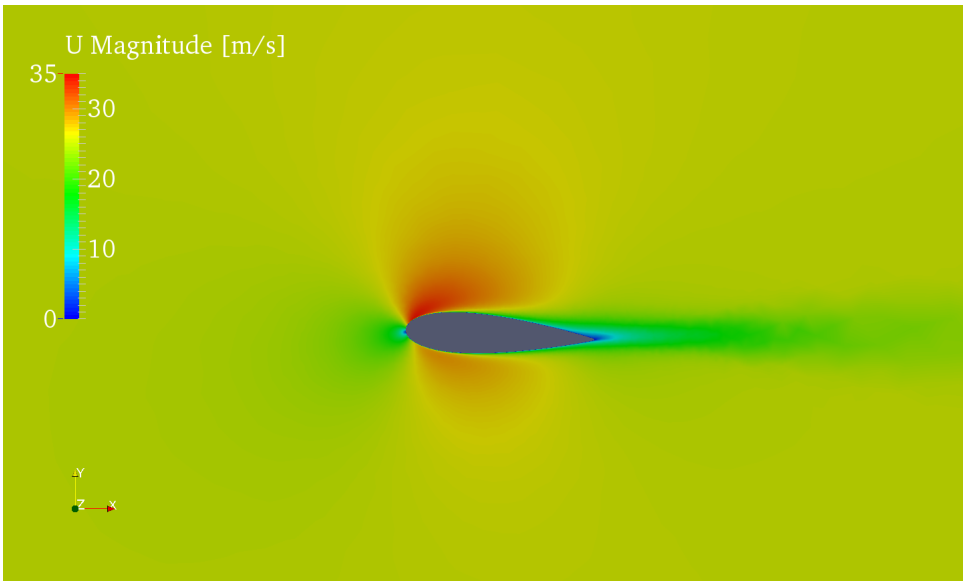

Figura 18: Distribuzione di velocità su un piano verticale passante per il ventre del bordo di attacco dell'ala coi tubercoli

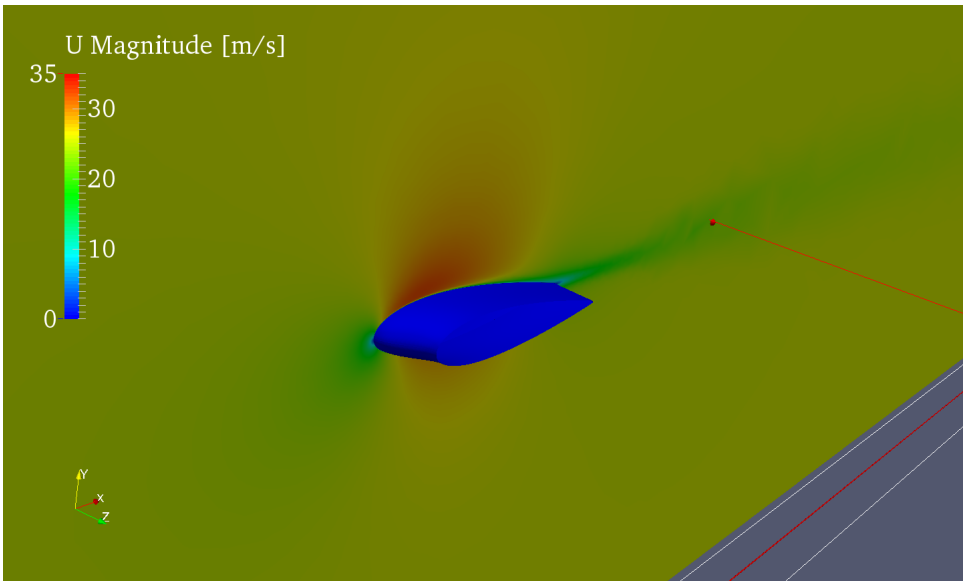

Figura 19: Vista in prospettiva della distribuzione di velocità su un piano verticale passante per la cresta del bordo di attacco dell'ala coi tubercoli

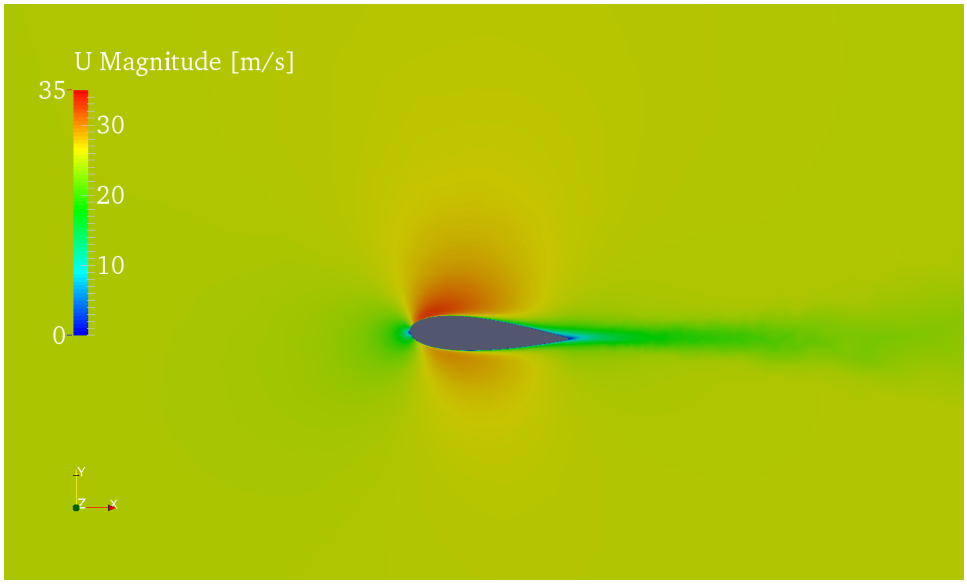

Figura 20: Distribuzione di velocità su un piano verticale secante l'ala retta

5.4 Distribuzione di vorticità

Utilizzando il filtro di ParaView *Compute Derivatives* è stato possibile calcolare, partendo dal campo della velocità, la distribuzione della vorticità in tutto il dominio di calcolo. Visualizzando questa quantità vettoriale sul dorso dell'ala e nella scia, in prossimità del bordo di fuga, è possibile visualizzare il nucleo dei filetti vorticosi rappresentanti la scia dell'ala (FIG. 21-22-23). Come mostrato da Taylor Swanson e K. M. Isaac [5] questi filetti sono 4, controrotanti, posizionati a metà strada tra ventre e cresta della sinusoide. I due vortici sviluppatisi dalla superfice ventrale dell'ala sono di intensità decisamente inferiore rispetto ai due superiori, sviluppatisi dalla superfice dorsale; questo è dovuto all'angolo di incidenza positivo assunto dall'ala. E' possibile visualizzare le strutture vorticose anche con l'ausilio delle iso-linee della turbulent kinetic energy.

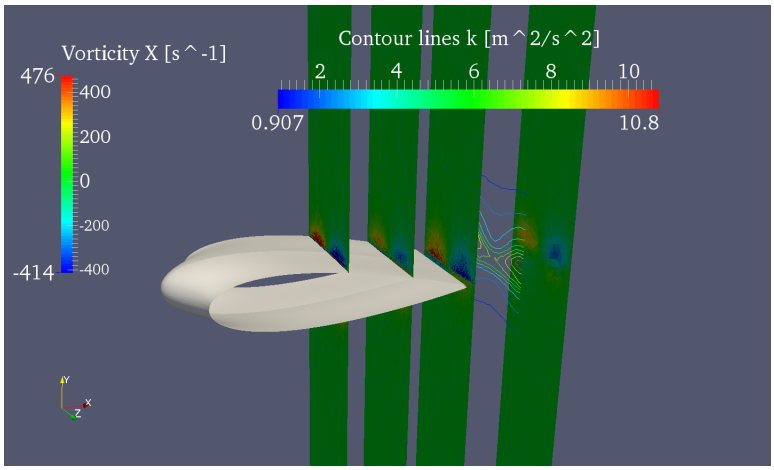

Figura 21: Andamento del modulo della componente x della vorticità lungo la corda dell'ala coi tubercoli

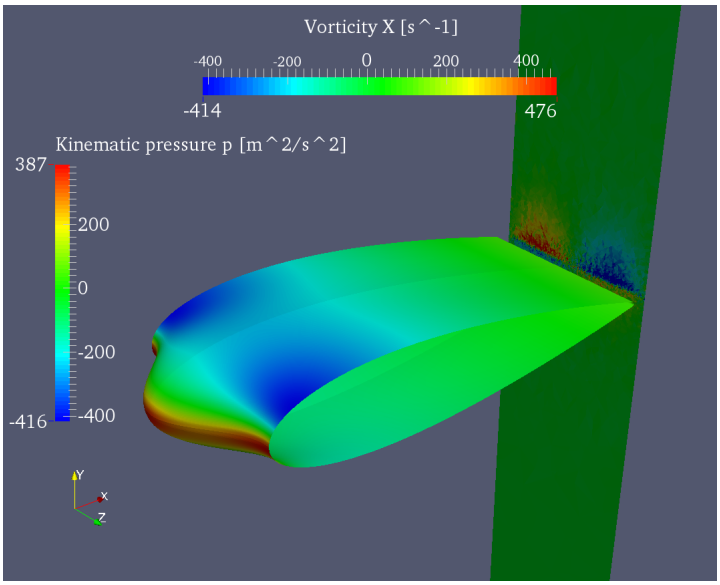

Figura 22: Componente x della vorticità in prossimità del bordo di fuga del'ala coi tubercoli

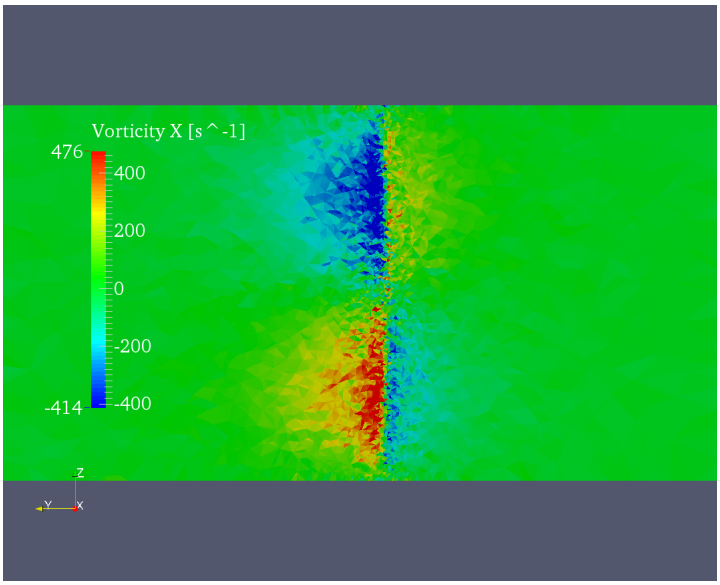

Figura 23: Vista frontale dei 4 filetti vorticosi in prossimita del bordo di fuga dell'ala coi tubercoli

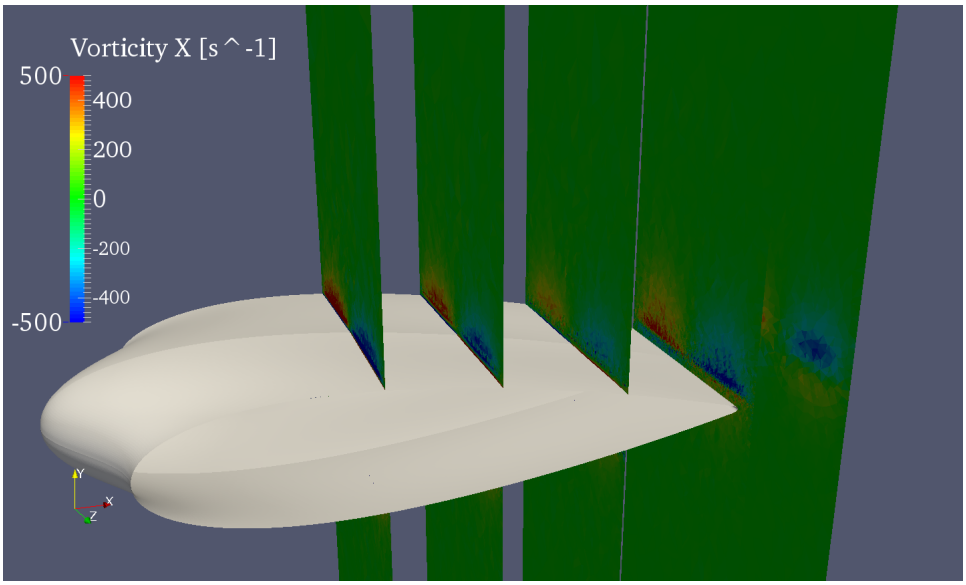

Figura 24: Particolare andamento del modulo della componente x della vorticità, con un picco in prossimita del bordo di fuga dell'ala coi tubercoli

E' possibile anche visualizzare (FIG. 24) un fenomeno che, stando alle conoscenze bibliografiche possedute e qui citate, non è stato ancora osservato. L'intensità della vorticità lungo l'asse x sembra avere un valore elevato a parete nella parte iniziale dell'ala, intensità che quindi decresce procedendo lungo la corda, per poi avere un picco di intensità una volta finita l'ala, subito dietro il bordo di fuga, e poi diminuire nuovamente. Una tale variazione lungo la corda potrebbe essere spiegata considerando il processo di "diffusione" della vorticità nel flusso. In prossimità del bordo di attacco la vorticità risulta più concentrata; mentre nella parte finale dell'ala il suo valore nel nucleo risulta inferiore, ma una regione di spazio maggiore risulta avere valori di vorticità diversi da zero.

Queste strutture vorticose sono state rappresentate anche tramite l'ausilio della variabile k. Oltre alle sopracitate iso-linee, sono state utilizzate delle iso-superfici per dare una forma tridimensionale ai vortici di scia (FIG. 25). L'utilizzo di k è lecito in quanto l'energia cinetica turbolenta è evidentemente superiore vicino al nucleo dei filetti vorticosi, e viceversa.

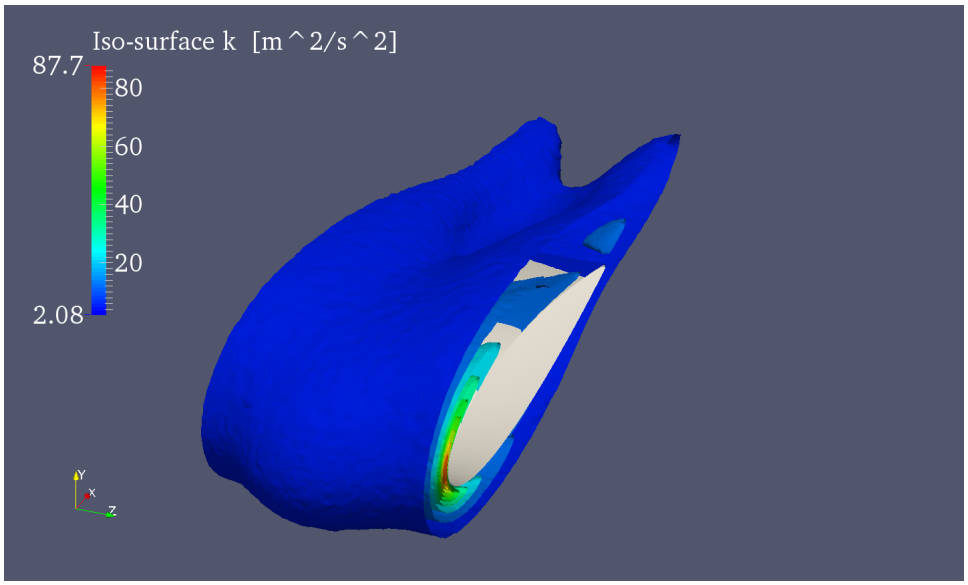

Figura 25: Iso-superfici k presenti nell'ala coi tubercoli

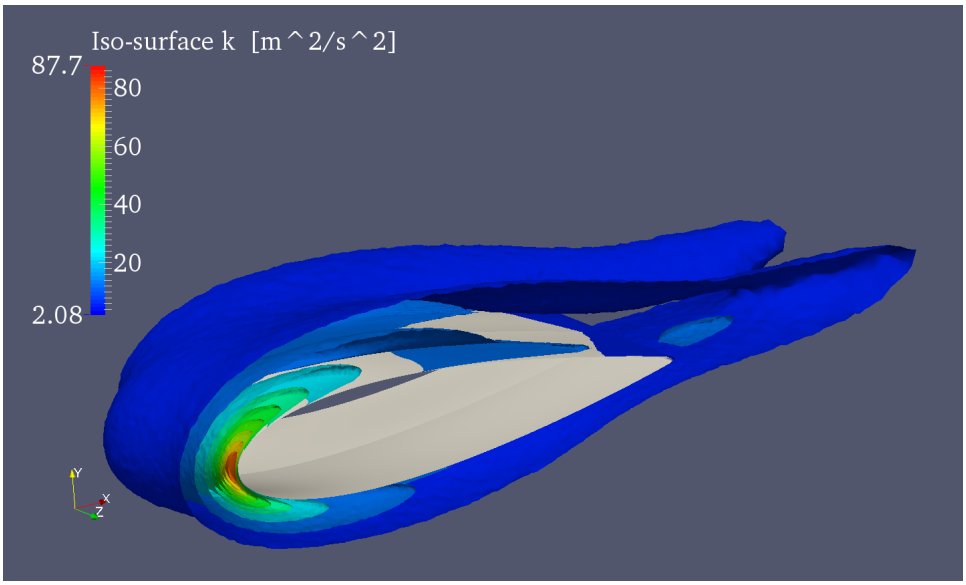

Figura 26: Iso-superfici k presenti nell'ala coi tubercoli, seconda vista

6 Conclusioni

I risultati ricavati nel capitolo precedente confermano i meccanismi osservati da Johari *et al.* [2] e van Nierop *et al.* [3]. L'aumento di prestazioni dell'ala coi tubercoli può essere giustificato osservando la distrubuzione di pressione e il suo gradiente sul dorso dell'ala. Una pressione minore combinata con una corda del valore $c - A/2$ porta ad avere un gradiente avverso di pressione maggiore dietro un ventre rispetto a quello dietro a una cresta. Questa distribuzione ritarda lo stallo di una parte dell'ala rispetto all'altra.

Una ragionevole giustificazione all'aumento dell'angolo di attacco critico e allo stallo più graduale può essere trovata anche nella dinamica delle scie vorticose dell'ala. Come mostrato in FIG. 21, considerando un periodo del bordo di attacco sinusoidale, sono presenti 2 vortici controrotanti in scia (in realtà 4, ma solo 2 sono predominanti). Questi filetti vorticosi, uno uscente dall'ala nella direzione x, l'altro entrante, possono essere giustificati analizzando la circuitazione variabile lungo l'apertura alare. Van Nierop *et al.* [3] richiama la teoria della linea portante di Prandtl in quanto, avendo un'ala finita, la circuitazione varia obbligatoriamente lungo l'apertura alare (i filetti vorticosi non possono essere troncati né richiudersi su se stessi [7]) attraverso la nascita di filetti vorticosi diretti lungo l'asse x, che formano una s*uperfice vorticosa libera*. Nel caso analizzato dall'articolo la variazione nella circuitazione è dovuta sia all'ala finita, sia al bordo di attacco sinusoidale; nel nostro caso invece entra in gioco solo la variazione sinusidale della corda. La circuitazione, che per un'ala infinita non deve variare lungo l'apertura alare, qui ha comunque un andamento periodico nel tempo. Quindi i vortici di scia che verranno a crearsi saranno coerenti con quelli analizzati da Prandtl.

Questi vortici controrotanti causano un *downwash* sulle creste e un *upwash* sui ventri che modificano localmente l'angolo di attacco effettivo; ecco spiegate quindi le distribuzioni di pressioni e velocità differenti tra cresta, ventre, e la stessa ala retta inclinata di 3 *deg*, distribuzioni compatibili con angoli di attacco diversi da profilo a profilo. Proprio il minore angolo di attacco effettivo dell'ala in prossimità di una cresta causa lo stallo dell'ala intera a un angolo di attacco geometrico maggiore rispetto al normale; inoltre lo stallo si presenta inizialmente solo sulla parte dell'ala dietro ai ventri, per poi estendersi all'ala completa, rendendo la perdità di portanza collegata allo stallo più graduale. Si possono quindi identificare nei vortici di scia i maggiori responsabili dell'aumento di prestazioni di questa configurazione alare.

6.1 Possibili applicazioni

Configurazioni del bordo di attacco simili a quella analizzata potrebbero trovare varie e numerose applicazioni nell'industria attuale. Ritardando lo stallo forniscono un più ampio margine di utilizzo "sicuro" nel grafico dell'efficenza $^{L}/_{D}$ [1]; infatti in ambito aeronautico il volo ad alti C_l è sempre abbastanza critico in quanto ci si trova in prossimità dello stallo, che potrebbe incorrere rapidamente nel caso di piccole variazioni di assetto dovute a disturbi esterni o raffiche improvvise. Il grafico $^{L}/_{D}$ di un'ala con bordo di attacco sinusoidale risulta mantenere livelli accettabili di efficenza per tratti piuttosto ampi [1], senza variazioni troppo repentine in prossimità dello stallo.

Nel 2014 la scuderia automobilistica McLaren ha introdotto in Formula 1 una configurazione simile a quella qui analizzata. In FIG. 26 si può individuare un andamento sinusoidale del bordo di fuga dell'alettone principale e del bordo di attacco del DRS, l'alettone mobile.

Figura 27: Bordo di fuga dell'alettone principale e bordo di attacco del DRS con andamento sinusoidale, rear wing McLaren MP4-29. Photograph obtained from www.newsf1.it

La WhalePower Corporation [16] ha trovato applicazione per il bordo di attacco sinusoidale nel campo delle turbine eoliche, raggiungendo un funzionamento più stabile e continuativo anche in caso di flussi turbolenti o non stazionari e diminuendo il rumore dovuto alla vibrazione del *wing tip* nel caso in cui l'estremità alare stalli.

Appendice I

Nelle pagine successive verranno riportati inizialmente i file (uno per pagina) facenti parte del caso di OpenFOAM sopra trattato, per ultimo verrà poi incluso il file .geo dell'ala "tubercled" inclinata di tre gradi. Il nome del file e la sua posizione potranno essere trovati rispettivamente nelle righe *object* e *location* posizionate all'inizio del file.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      epsilon;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [ 0 2 -3 0 0 0 0 ];

internalField   uniform 0.4358;

boundaryField
{
    front
    {
        type            symmetryPlane;
    }
    back
    {
        type           symmetryPlane;
    }
    wing
    {
        type            epsilonWallFunction;
        value           uniform 0;
    }

    inlet
    {
        type            turbulentMixingLengthDissipationRateInlet;
        mixingLength    0.005928;
        value           uniform 0.4358;
    }

    outlet
    {
        type            zeroGradient;
    }

    top
    {
        type           zeroGradient;
    }
    bottom
    {
        type            zeroGradient;
    }
}


// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [ 0 2 -2 0 0 0 0 ];

internalField   uniform 0.09375;

boundaryField
{
   front
    {
        type            symmetryPlane;
    }
    back
    {
        type            symmetryPlane;
    }

    wing
    {
        type            kqRWallFunction;
        value           uniform 0;
    }
    inlet
    {

        type            turbulentIntensityKineticEnergyInlet;
        intensity       0.01;
        value           uniform 0.09375;
    }

    outlet
    {
         type            zeroGradient;

    }

    top
    {
        type            zeroGradient;
    }
    bottom
    {
        type            zeroGradient;
    }


}


// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0.14;

boundaryField
{
    front
    {
        type            symmetryPlane;
    }
    back
    {
        type            symmetryPlane;
    }
    wing
    {
       type            nutUSpaldingWallFunction;
        value           uniform 0;
        }
    inlet
    {
         type            freestream;
        freestreamValue uniform 0.14;
    }

    outlet
    {
         type            freestream;
         freestreamValue uniform 0.14;
    }

    top
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }
    bottom
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }
}

// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nuTilda;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0.14;

boundaryField
{
    front
    {
        type            symmetryPlane;
    }
    back
    {
        type            symmetryPlane;
    }
    wing
    {
        type            fixedValue;
        value           uniform 0;
    }
    inlet
    {
         type            freestream;
        freestreamValue uniform 0.14;
    }

    outlet
    {
         type            freestream;
        freestreamValue uniform 0.14;
    }

    top
    {
         type            fixedValue;
        value           uniform 0;
    }
    bottom
    {
         type            fixedValue;
        value           uniform 0;
    }
}

// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{

    wing
    {
        type            zeroGradient;
    }
    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    top
    {
        type            zeroGradient;
    }
    bottom
    {
        type            zeroGradient;
    }
    front
    {
        type           symmetryPlane;
    }
    back
    {
        type            symmetryPlane;
    }
}

// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (25 0 0);

boundaryField
{
     front
    {
        type            symmetryPlane;
    }
    back
    {
        type            symmetryPlane;
    }
    wing
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    inlet
    {
        type            fixedValue;
        value           uniform (25 0 0);
    }

    outlet
    {
        type            zeroGradient;
    }

    top
    {
        type            slip;
    }
    bottom
    {
        type            slip;
    }
}

// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

7
(
    back
    {
        type            symmetryPlane;
        nFaces          24380;
        startFace       5102206;
    }
    front
    {
        type            symmetryPlane;
        nFaces          24380;
        startFace       5126586;
    }
    bottom
    {
        type            wall;
        physicalType    wall;
        nFaces          134;
        startFace       5150966;
    }
    outlet
    {
        type            patch;
        physicalType    patch;
        nFaces          52;
        startFace       5151100;
    }
    top
    {
        type            wall;
        physicalType    wall;
        nFaces          134;
        startFace       5151152;
    }
    inlet
    {
        type            patch;
        physicalType    patch;
        nFaces          52;
        startFace       5151286;
    }
    wing
    {
        type            wall;
        physicalType    wall;
        nFaces          81576;
        startFace       5151338;
    }
)

// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      RASProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

RASModel        kEpsilon;

turbulence      on;

printCoeffs     on;


// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

rho             rho [ 1 -3 0 0 0 0 0 ] 1;

nu              nu [ 0 2 -1 0 0 0 0 ] 1.5e-05;

CrossPowerLawCoeffs
{
    nu0             nu0 [ 0 2 -1 0 0 0 0 ] 1e-06;
    nuInf           nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
    m               m [ 0 0 1 0 0 0 0 ] 1;
    n               n [ 0 0 0 0 0 0 0 ] 1;
}

BirdCarreauCoeffs
{
    nu0             nu0 [ 0 2 -1 0 0 0 0 ] 1e-06;
    nuInf           nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
    k               k [ 0 0 1 0 0 0 0 ] 0;
    n               n [ 0 0 0 0 0 0 0 ] 1;
}


// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     simpleFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         550;

deltaT          0.1;

writeControl    timeStep;

writeInterval   10;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

functions
{
    #include "forceCoeffs"
}

// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 8;

method          simple;

simpleCoeffs
{
    n               ( 2 2 2 );
    delta           0.001;
}

hierarchicalCoeffs
{
    n               ( 1 1 1 );
    delta           0.001;
    order           xyz;
}

manualCoeffs
{
    dataFile        "";
}

distributed     no;

roots           ( );


// ************************************************************************* //
```

```
forces
{
    type                forces;
    functionObjectLibs  ("libforces.so");
    outputControl       timeStep;
    outputInterval      1;

    patches             ( wing );
    pName               p;
    UName               U;
    rhoName             rhoInf;
    log                 true;

    CofR                (0.9309407329 0 0.01455882353);

    rhoInf              1.0;
}


forceCoeffs
{
    type                forceCoeffs;
    functionObjectLibs  ( "libforces.so" );
    outputControl       timeStep;
    outputInterval      1;

    patches             ( wing );
    pName               p;
    UName               U;
    rhoName             rhoInf;
    log                 true;

    liftDir             (0 1 0);
    dragDir             (1 0 0);
    CofR                (0.9309407329 0 0.01455882353);
    pitchAxis           (0 0 1);

    magUInf             25.00;
    rhoInf              1.0;
    lRef                0.0688456895;
    Aref                2.004624488e-03;
}
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         steadyState;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      bounded Gauss linearUpwind grad(U);
    div(phi,nuTilda) bounded Gauss linearUpwind grad(nuTilda);
    div((nuEff*dev(T(grad(U))))) Gauss linear;
    div(phi,k)      bounded Gauss upwind;
    div(phi,epsilon) bounded Gauss upwind;
    div(phi,R)      bounded Gauss upwind;
    div(R)          Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p               ;
}


// ************************************************************************* //
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.1;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }

    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }

    nuTilda
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-08;
        relTol          0.1;
    }
    "(k|epsilon|R)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0.1;
    }
}

SIMPLE
{
    nNonOrthogonalCorrectors 2;
    pRefCell        0;
    pRefValue       0;

    residualControl
    {
        p               1e-4;
        U               1e-4;
        "(k|epsilon|omega)" 1e-4;
    }
}

relaxationFactors
```

```
{
    fields
    {
        p               0.3;
    }
    equations
    {
        U               0.7;
        k               0.7;
        epsilon         0.7;
        R               0.7;
        nuTilda         0.7;
    }
}


// ************************************************************************* //
```

```
Include "NACA0021.geo";
Include "NACA0023.geo";
//************************************************************
//              |PARAMETRI|
//************************************************************
//Parametri ala
b=.495/17;      // apertura alare [m]
c=.07;          // corda media profilo [m] vera=0.0689571135
alfa = 3;       // aoa [deg]
n=100;          // punti bordo attacco
T=1;            // numero periodi funzione sinusoidale
k1=.004;        // ampiezza picco-picco/2
lc=0.015*c;     // dimensione caratt mesh
lcbl=0.015*c;   // dimensione caratt mesh profilo

boundarylayer_cells = 8;                //numero livelli bounday layer
boundarylayer_cells = boundarylayer_cells + 1;
boundarylayer_progression = 1.05;       //coeff. progressione geometrica altezza celle BL

//Parametri domini
h=0.8*c;        //altezza
in=0.6*c;       //distanza anteriore
out=0.3*c;      //distanza posteriore
ld=2*lc;        //dimensione caratteristica della mesh

blheight=1.05;  //grandezza strato BL

ld1=12*ld;      //dimensione caratteristica della mesh esterna
h1=5*c;         //altezza ext.
in1=4*c;        //distanza anteriore ext.
out1=7*c;       //distanza posteriore ext.
//************************************************************
//              |COSTRUZIONE ALA|
//************************************************************

//CREAZIONE PROFILO INTERNO
Call NACA0021;
te1={pointlist[0]};
le1={pointlist[np/2]};
dorsob=newl;
Spline(dorsob)={6:le1};
d1=newl;Spline(d1)={1:6};
ventreb=newl;
Spline(ventreb)={le1:pointlist[#pointlist[]-6],96};
v1=newl;Spline(v1)={96:100,1};
profilob[]={d1,dorsob,ventreb,v1};

Dilate {{0.982575, 0, 0}, c-k1} {               //modifica la corda
  Point{pointlist[]};
}


//CREAZIONE PROFILO ESTERNO
Call NACA0023;
Dilate {{0.982575, 0, 0}, c-k1} {       //modifica la corda
  Point{pointlist1[]};
}

Point(151)={0.91772505,0,0,lcbl};
Point(101)={0.982575,0,0,lcbl};
tebl[]={pointlist1[0]};

dorsobl=newl; Spline(dorsobl)={106:151};
d1bl=newl;Spline(d1bl)={101:106};
ventrebl=newl; Spline(ventrebl)={151:pointlist1[#pointlist1[]-6],196};
v1bl=newl;Spline(v1bl)={196:200,101};
profilobl[]={d1bl,dorsobl,ventrebl,v1bl};


Dilate {{0.982575-c/2, 0, 0}, blheight} {       //allarga profilo esterno in modo da creare
BL
  Line {profilobl[]};
}
```

```
        profilob1[]={Translate {0, 0, b} {              //duplica profilo int
          Duplicata { Line{profilob[]}; }
        }};

        profilobl1[]={Translate {0, 0, b} {             //duplica profilo ext
          Duplicata { Line{profilobl[]}; }
        }};

        profilob2[]= {Dilate {{0.982575, 0, 0}, (c+k1)*1.0018/(c-k1)}{Translate {0, 0, b/2}{Duplicata
{ Line{profilob[]}; }}}};

        profilobl2[]= {Dilate {{0.982575, 0, 0}, (c+k1)*0.99946/(c-k1)}{Translate {0, 0, b/2}
{Duplicata { Line{profilobl[]}; }}}};

        Translate {0, 0, -0.001794088235} {             //duplica profilo int mid
           Line{profilob2[]};
        }
        Translate {0, 0, -0.001755891176} {             //duplica profilo ext mid
           Line{profilobl2[]};
        }


        //CREAZIONE RETTE SUDDIVISIONE BL
        teretta=newl; Line(teretta)={1,101};
        leretta=newl; Line(leretta)={51,151};

        teretta1=newl; Line(teretta1)={201,310};
        leretta1=newl; Line(leretta1)={254,363};

        teretta2=newl; Line(teretta2)={419,528};
        leretta2=newl; Line(leretta2)={472,581};

        terettabl01=newl; Line(terettabl01)={101,528};
        terettab01=newl; Line(terettab01)={1,419};

        terettabl21=newl; Line(terettabl21)={419,201};
        terettab21=newl; Line(terettab21)={528,310};


        //CREAZIONE BORDO ATTACCO SIN INTERNO
        dz=b/2/(n-1);
        z=0;
        lep[]={};

        For i In {1:n-2}
                z=z+dz;
                p=newp; Point(p)={k1*Sin(2*Pi*T*z/b+2*Pi*T/4)+0.982575-c*0.9835,0,z,lc};
                lep[]={lep[],p};
        EndFor

        lel=newl; Spline(lel)={le1,lep[],472};

        dz=b/2/(n-1);
        z=0+b/2;
        lep[]={};

        For i In {1:n-2}
                z=z+dz;
                p=newp; Point(p)={k1*Sin(2*Pi*T*z/b+5*2*Pi*T/4)+0.982575-c*0.9835,0,z,lc};
                lep[]={lep[],p};
        EndFor

        lel=newl; Spline(lel)={472,lep[],254}; // leading edge line

        //CREAZIONE BORDO ATTACCO SIN ESTERNO
        dz=b/2/(n-1);
        z=0;
        lep1[]={};

        For i In {1:n-2}
                z=z+dz;
                p1=newp; Point(p1)={k1*Sin(2*Pi*T*z/b+2*Pi*T/4)+0.982575-c*blheight*0.957,0,z,lc};
                lep1[]={lep1[],p1};
```

```
        EndFor

        lelbl=newl; Spline(lelbl)={151,lep1[],581};

        dz=b/2/(n-1);
        z=0+b/2;
        lep1[]={};

        For i In {1:n-2}
                z=z+dz;
                p1=newp; Point(p1)={k1*Sin(2*Pi*T*z/b+5*2*Pi*T/4)+0.982575-c*blheight*0.957,0,z,lc};
                lep1[]={lep1[],p1};
        EndFor

        lelbl=newl; Spline(lelbl)={581,lep1[],363}; // leading edge line

        //RETTE SUDDIVISIONE BL TRAILING EDGE
        d11=newl;
        Line(d11)={6,424};
        v11=newl;
        Line(v11)={96,520};
        d11bl=newl;
        Line(d11bl)={106,533};
        v11bl=newl;
        Line(v11bl)={196,629};

        d112=newl;
        Line(d112)={424,206};
        v112=newl;
        Line(v112)={520,302};
        d112bl=newl;
        Line(d112bl)={533,315};
        v112bl=newl;
        Line(v112bl)={629,411};

        fd11=newl;
        Line(fd11)={206,315};
        fv11=newl;
        Line(fv11)={302,411};
        bd11=newl;
        Line(bd11)={6,106};
        bv11=newl;
        Line(bv11)={96,196};
        bd1122=newl;
        Line(bd1122)={424,533};
        bv1122=newl;
        Line(bv1122)={520,629};

        //SUDDIVISIONE TRANSFINITE LINEE MESH STRUTTURATA
        //suddivisione livelli BL
        Transfinite Line{47,27,48,28,49,25,50,26,51,29,52,30} = boundarylayer_cells Using Progression
boundarylayer_progression;

        Transfinite Line{14,10,11,15,22,18,19,23,5,1,3,7} = 200 Using Bump 0.30;
              //suddivisione profili
        Transfinite Line{6,2,4,8,17,20,24,21,9,12,16,13} =
8 ;                                          //suddivisione parte finale profili

        Transfinite Line{38,36,35,37,43,44,33,45,46,34,39,40,32,42,41,31} = 100;
              //suddivisione spanwise


        //CREAZIONE SUPERFICI ALA
        ds=newll; Line Loop(ds)={18,-35,-1,39};
        ds1=newll; Line Loop(ds1)={17,-39,-2,32};
        vs=newll; Line Loop(vs)={-19,-35,3,40};
        vs1=newll; Line Loop(vs1)={-20,-40,4,32};
        ventreSurf=news; Ruled Surface(ventreSurf)={vs};
        dorsoSurf=news; Ruled Surface(dorsoSurf)={ds};
        ventreSurf1=news; Ruled Surface(ventreSurf1)={vs1};
        dorsoSurf1=news; Ruled Surface(dorsoSurf1)={ds1};
        Transfinite Surface{ventreSurf,dorsoSurf,ventreSurf1,dorsoSurf1};                    //mesh
strutturata
        Recombine Surface{ventreSurf,dorsoSurf,ventreSurf1,dorsoSurf1};
```

```
        ds3=newll; Line Loop(ds3)={10,-36,-18,43};
        ds13=newll; Line Loop(ds13)={9,-43,-17,33};
        vs3=newll; Line Loop(vs3)={-11,-36,19,44};
        vs13=newll; Line Loop(vs13)={-12,-44,20,33};
        ventreSurf3=news; Ruled Surface(ventreSurf3)={vs3};
        dorsoSurf3=news; Ruled Surface(dorsoSurf3)={ds3};
        ventreSurf13=news; Ruled Surface(ventreSurf13)={vs13};
        dorsoSurf13=news; Ruled Surface(dorsoSurf13)={ds13};
        Transfinite Surface{ventreSurf3,dorsoSurf3,ventreSurf13,dorsoSurf13};                  //mesh
strutturata
        Recombine Surface{ventreSurf3,dorsoSurf3,ventreSurf13,dorsoSurf13};

        //CREAZIONE SUPERFICI BL
        dsbl=newll; Line Loop(dsbl)={22,-37,-5,41};
        dsbl1=newll; Line Loop(dsbl1)={6,41,-21,-31};
        vsbl=newll; Line Loop(vsbl)={-23,-37,7,42};
        vsbl1=newll; Line Loop(vsbl1)={8,31,-24,-42};
        ventreSurfbl=news; Ruled Surface(ventreSurfbl)={vsbl};
        ventreSurfbl1=news; Ruled Surface(ventreSurfbl1)={vsbl1};
        dorsoSurfbl=news; Ruled Surface(dorsoSurfbl)={dsbl};
        dorsoSurfbl1=news; Ruled Surface(dorsoSurfbl1)={dsbl1};
        Transfinite Surface{ventreSurfbl,dorsoSurfbl,ventreSurfbl1,dorsoSurfbl1};

        dsbl3=newll; Line Loop(dsbl3)={14,-38,-22,45};
        dsbl13=newll; Line Loop(dsbl13)={21,45,-13,-34};
        vsbl3=newll; Line Loop(vsbl3)={-15,-38,23,46};
        vsbl13=newll; Line Loop(vsbl13)={24,34,-16,-46};
        ventreSurfbl3=news; Ruled Surface(ventreSurfbl3)={vsbl3};
        ventreSurfbl13=news; Ruled Surface(ventreSurfbl13)={vsbl13};
        dorsoSurfbl3=news; Ruled Surface(dorsoSurfbl3)={dsbl3};
        dorsoSurfbl13=news; Ruled Surface(dorsoSurfbl13)={dsbl13};
        Transfinite Surface{ventreSurfbl3,dorsoSurfbl3,ventreSurfbl13,dorsoSurfbl13};

        //CREAZIONE SUPERFICI DIVISIONE BL
        leblsur=newll; Line Loop(leblsur)={-30,37,26,-35};
        teblsur=newll; Line Loop(teblsur)={-29,-32,25,31};
        teblsurd=newll; Line Loop(teblsurd)={-51,-39,49,41};
        teblsurv=newll; Line Loop(teblsurv)={-40,50,42,-52};
        leSurfbl=news; Ruled Surface(leSurfbl)={leblsur};
        teSurfbl=news; Ruled Surface(teSurfbl)={teblsur};
        teSurfbld=news; Ruled Surface(teSurfbld)={teblsurd};
        teSurfblv=news; Ruled Surface(teSurfblv)={teblsurv};
        Transfinite Surface{leSurfbl,teSurfbl,teSurfbld,teSurfblv};
        Recombine Surface{leSurfbl,teSurfbl,teSurfbld,teSurfblv};

        leblsur1=newll; Line Loop(leblsur1)={28,-38,-30,36};
        teblsur1=newll; Line Loop(teblsur1)={-27,-33,29,34};
        teblsurd1=newll; Line Loop(teblsurd1)={-47,-43,51,45};
        teblsurv1=newll; Line Loop(teblsurv1)={-48,-44,52,46};
        leSurfbl1=news; Ruled Surface(leSurfbl1)={leblsur1};
        teSurfbl1=news; Ruled Surface(teSurfbl1)={teblsur1};
        teSurfbld1=news; Ruled Surface(teSurfbld1)={teblsurd1};
        teSurfblv1=news; Ruled Surface(teSurfblv1)={teblsurv1};
        Transfinite Surface{leSurfbl1,teSurfbl1,teSurfbld1,teSurfblv1};
        Recombine Surface{leSurfbl1,teSurfbl1,teSurfbld1,teSurfblv1};

        //CREAZIONE SUPERFICI BL BACK
        blsurv=newll; Line Loop(blsurv)={-3,26,7,-50};
        blsurv2=newll; Line Loop(blsurv2)={-4,50,8,-25};
        blsurd=newll; Line Loop(blsurd)={1,26,-5,-49};
        blsurd2=newll; Line Loop(blsurd2)={25,6,-49,-2};
        Surfblv=news; Ruled Surface(Surfblv)={blsurv};
        Surfblv2=news; Ruled Surface(Surfblv2)={blsurv2};
        Surfbld=news; Ruled Surface(Surfbld)={blsurd};
        Surfbld2=news; Ruled Surface(Surfbld2)={blsurd2};
        Transfinite Surface{Surfblv,Surfbld,Surfblv2,Surfbld2};
        Recombine Surface{Surfblv,Surfbld,Surfblv2,Surfbld2};

        //CREAZIONE SUPERFICI BL MID
        blsurv3=newll; Line Loop(blsurv3)={-19,30,23,-52};
        blsurv23=newll; Line Loop(blsurv23)={52,24,-29,-20};
        blsurd3=newll; Line Loop(blsurd3)={51,22,-30,-18};
        blsurd23=newll; Line Loop(blsurd23)={29,21,-51,-17};
```

```
        Surfblv3=news; Ruled Surface(Surfblv3)={blsurv3};
        Surfblv23=news; Ruled Surface(Surfblv23)={blsurv23};
        Surfbld3=news; Ruled Surface(Surfbld3)={blsurd3};
        Surfbld23=news; Ruled Surface(Surfbld23)={blsurd23};
        Transfinite Surface{Surfblv3,Surfbld3,Surfblv23,Surfbld23};
        Recombine Surface{Surfblv3,Surfbld3,Surfblv23,Surfbld23};

        //CREAZIONI SUPERFICI BL FRONT
        blsurv1=newll; Line Loop(blsurv1)={-11,28,15,-48};
        blsurv12=newll; Line Loop(blsurv12)={48,16,-27,-12};
        blsurd1=newll; Line Loop(blsurd1)={47,14,-28,-10};
        blsurd12=newll; Line Loop(blsurd12)={27,13,-47,-9};
        Surfblv1=news; Ruled Surface(Surfblv1)={blsurv1};
        Surfblv12=news; Ruled Surface(Surfblv12)={blsurv12};
        Surfbld1=news; Ruled Surface(Surfbld1)={blsurd1};
        Surfbld12=news; Ruled Surface(Surfbld12)={blsurd12};
        Transfinite Surface{Surfblv1,Surfbld1,Surfblv12,Surfbld12};
        Recombine Surface{Surfblv1,Surfbld1,Surfblv12,Surfbld12};

        //ROTAZIONE ALA ANGOLO ALFA
        Rotate {{0, 0, 1}, {0.982575-c/2, 0, 0}, -alfa*Pi/180} {
          Surface
{ventreSurf,dorsoSurf,ventreSurf1,dorsoSurf1,ventreSurf3,dorsoSurf3,ventreSurf13,dorsoSurf13,ventreSurfbl,d

ventreSurfbl1,dorsoSurfbl1,ventreSurfbl3,dorsoSurfbl3,ventreSurfbl13,dorsoSurfbl13,leSurfbl,teSurfbl,teSurf

teSurfbld1,teSurfblv1,Surfblv,Surfbld,Surfblv2,Surfbld2,Surfblv3,Surfbld3,Surfblv23,Surfbld23,Surfblv1,Surf
        }


        //CREAZIONE VOLUMI BOUNDARY LAYER
        BLVs=newsl; Surface Loop(BLVs)={ventreSurf,ventreSurfbl,leSurfbl,teSurfblv,Surfblv,Surfblv3};
        BLV=newv; Volume(BLV)={BLVs};
        Transfinite Volume{BLV};                                //mesh strutturata
        TransfQuadTri{BLV};

        BLDs=newsl; Surface Loop(BLDs)={dorsoSurf,dorsoSurfbl,leSurfbl,teSurfbld,Surfbld,Surfbld3};
        BLD=newv; Volume(BLD)={BLDs};
        Transfinite Volume{BLD};
        TransfQuadTri{BLD};

        BLVs3=newsl; Surface Loop(BLVs3)=
{ventreSurf3,ventreSurfbl3,leSurfbl1,teSurfblv1,Surfblv3,Surfblv1};
        BLV3=newv; Volume(BLV3)={BLVs3};
        Transfinite Volume{BLV3};                               //mesh strutturata
        TransfQuadTri{BLV3};

        BLDs3=newsl; Surface Loop(BLDs3)=
{dorsoSurf3,dorsoSurfbl3,leSurfbl1,teSurfbld1,Surfbld3,Surfbld1};
        BLD3=newv; Volume(BLD3)={BLDs3};
        Transfinite Volume{BLD3};
        TransfQuadTri{BLD3};

        BLVs1=newsl; Surface Loop(BLVs1)=
{ventreSurf1,ventreSurfbl1,teSurfbl,teSurfblv,Surfblv2,Surfblv23};
        BLV1=newv; Volume(BLV1)={BLVs1};
        Transfinite Volume{BLV1};
        TransfQuadTri{BLV1};

        BLDs1=newsl; Surface Loop(BLDs1)=
{dorsoSurf1,dorsoSurfbl1,teSurfbl,teSurfbld,Surfbld2,Surfbld23};
        BLD1=newv; Volume(BLD1)={BLDs1};
        Transfinite Volume{BLD1};
        TransfQuadTri{BLD1};

        BLVs13=newsl; Surface Loop(BLVs13)=
{ventreSurf13,ventreSurfbl13,teSurfbl1,teSurfblv1,Surfblv23,Surfblv12};
        BLV13=newv; Volume(BLV13)={BLVs13};
        Transfinite Volume{BLV13};
        TransfQuadTri{BLV13};

        BLDs13=newsl; Surface Loop(BLDs13)=
{dorsoSurf13,dorsoSurfbl13,teSurfbl1,teSurfbld1,Surfbld23,Surfbld12};
        BLD13=newv; Volume(BLD13)={BLDs13};
```

```
        Transfinite Volume{BLD13};
        TransfQuadTri{BLD13};

        //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        //              |DOMINIO ESTERNO|
        //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

        //RETTANGOLO INTERNO
        p1=newp; Point(p1)={1-c-in,-h/2,0,ld};
        p2=newp; Point(p2)={1+out,-h/2,0,ld};
        p3=newp; Point(p3)={1+out, h/2,0,ld};
        p4=newp; Point(p4)={1-c-in, h/2,0,ld};

        l1=newl; Line(l1)={p1,p2};
        l2=newl; Line(l2)={p2,p3};
        l3=newl; Line(l3)={p3,p4};
        l4=newl; Line(l4)={p4,p1};

        backl[]={l1,l2,l3,l4};

        out[]={Extrude {0, 0, b} {
          Line{backl[]};
        }};

        profll[0]=newll; Line Loop(profll[0]) ={6,5,7,8};
        profll[1]=newll; Line Loop(profll[1]) ={13,14,15,16};

        ll=newll; Line Loop(ll)={backl[]};
        backSurf=news; Plane Surface(backSurf)={ll,profll[0]}; //back

        frontl[]={out[0],out[4],out[8],out[12]};
        ll2=newll; Line Loop(ll2)={frontl[]};
        frontSurf=news; Plane Surface(frontSurf)={ll2,profll[1]}; //front

        blSurf[]=
{ventreSurfbl3,dorsoSurfbl3,ventreSurfbl13,dorsoSurfbl13,ventreSurfbl,dorsoSurfbl,ventreSurfbl1,dorsoSurfbl
        sl=newsl; Surface Loop(sl)={backSurf,frontSurf,out[1],out[5],out[9],out[13],blSurf[]};
        vol[]={newv}; Volume(vol[0])={sl};

        //RETTANGOLO ESTERNO
        p11=newp; Point(p11)={1-c-in1,-h1/2,0,ld1};
        p21=newp; Point(p21)={1+out1,-h1/2,0,ld1};
        p31=newp; Point(p31)={1+out1, h1/2,0,ld1};
        p41=newp; Point(p41)={1-c-in1, h1/2,0,ld1};

        l11=newl; Line(l11)={p11,p21};
        l21=newl; Line(l21)={p21,p31};
        l31=newl; Line(l31)={p31,p41};
        l41=newl; Line(l41)={p41,p11};

        backl1[]={l11,l21,l31,l41};

        out1[]={Extrude {0, 0, b} {
          Line{backl1[]};
        }};


        ll1=newll; Line Loop(ll1)={backl1[]};
        backSurf1=news; Plane Surface(backSurf1)={ll1,ll}; //back

        frontl1[]={out1[0],out1[4],out1[8],out1[12]};
        ll3=newll; Line Loop(ll3)={frontl1[]};
        frontSurf1=news; Plane Surface(frontSurf1)={ll3,ll2}; //front

        intSurf[]={out[1],out[5],out[9],out[13]};
        sl1=newsl; Surface Loop(sl1)={backSurf1,frontSurf1,out1[1],out1[5],out1[9],out1[13],intSurf[]};
        vol1[]={newv}; Volume(vol1[0])={sl1};

        //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        //              |DEFINIZIONI PHYSICAL ENTITIES|
        //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

        Physical Surface("wing") =
{ventreSurf,dorsoSurf,ventreSurf1,dorsoSurf1,ventreSurf3,dorsoSurf3,ventreSurf13,dorsoSurf13};
```

```
        Physical Surface("front") = {frontSurf,frontSurf1,Surfblv1,Surfbld1,Surfblv12,Surfbld12};
        Physical Surface("back") = {backSurf,backSurf1,Surfblv,Surfbld,Surfblv2,Surfbld2};
        Physical Surface("bottom") = {out1[1]};
        Physical Surface("outlet") = {out1[5]};
        Physical Surface("top") = {out1[9]};
        Physical Surface("inlet") = {out1[13]};

        Physical Volume("insideVolume") = {vol[0],vol1
[0],BLV,BLD,BLV1,BLD1,BLV3,BLD3,BLV13,BLD13};
```

Riferimenti bibliografici

[1] D. S. Miklosovic, M. M. Murray, L. E. Howle, and F. E. Fish, "Leading edge tubercles delay stall on humpback whale flippers," Phys. Fluids 16(5), L39–L42 (2004).

[2] H. Johari, C. Henoch, D. Custodio, and A. Levshin, "Effects of leading edge protuberances on airfoil performance," AIAA J. 45, 2634–2642 (2007).

[3] E. van Nierop, S. Alben, and M. P. Brenner, "How bumps on whale flippers delay stall: an aerodynamic model," Phys. Rev. Lett. 100, 054502 (2008).

[4] N. Rostamzadeh, R. M. Kelso, B. B. Dally, and K. L. Hansen,"The effect of undulating leading-edge modifications on NACA 0021 airfoil characteristics",PHYSICS OF FLUIDS 25, 117101 (2013)

[5] Taylor Swanson and K. M. Isaac,"American Institute of Aeronautics and Astronautics 1 Biologically Inspired Wing Leading Edge for Enhanced Wind Turbine and Aircraft Performance", AIAA 2011-3533

[6] A. Baron, Appunti del corso di Fluidodinamica

[7] R. Arina, "Fondamenti di aerodinamica Parte I, Parte II', Levrotto&Bella (2015)

[8] http://support.esi-cfd.com

[9] http://www.cfd-online.com

[10] http://cfd.direct/

[11] http://www.openfoam.org

[12] https://en.wikipedia.org

[13] http://www.mathematik.uni-dortmund.de

[14] http://geuz.org

[15] http://icaro.dief.unifi.it

[16] http://www.whalepowercorporation.com