



HOCHSCHULE BREMEN
UNIVERSITY OF APPLIED SCIENCES

Fakultät 5 – Natur und Technik

Numerische Untersuchungen instationärer Hochauftriebssysteme nach biologischem Vorbild

Projektbericht

im Fachgebiet Bionik

vorgelegt von: Aljoscha N. Sander

Matrikel Nr.: 322128

Fachrichtung: Numerische Strömungssimulation

Erstprüfer: Prof. Dr.-Ing. Albert Baars

Zweitprüfer: Prof. Dr. Antonia Kesel

Datum der Abgabe: 20.07.2015

Zusammenfassung

Eine neue für OpenFOAM zur Verfügung stehende Bibliothek (*dynamicTopoFvMesh*) wird in dieser Arbeit auf ihre Verwendung zur Simulation des Schlagflugs hin untersucht. Zusätzlich werden zwei weitere, bereits etablierte numerische Techniken für bewegte Gitter auf Ihre Verwendbarkeit zur Simulation des Schlagflugs untersucht. Die Eigenheiten und Details der drei Methoden werden zunächst detailliert beschrieben um sie anschließend zu testen. Es kann gezeigt werden, dass trotz bestehender softwaretechnischer Probleme die Bibliothek *dynamicTopoFvMesh* grundsätzlich ein innovativer, neuer Ansatz zur Simulation bewegter Gitter ist. Es gelingt jedoch nicht, sie für die Simulation des Schlagflugs einzusetzen. Die verbleibenden Methoden, *Radiale Basisfunktionen (RBF)* und *Arbitrary Mesh Interface (AMI)* eignen sich beide, abhängig von der Art der Simulation (2D / 3D) und den Randbedingungen sehr gut zur Simulation des Schlagflugs und damit zur Untersuchung instationärer Hochauftriebseffekte. Es wurde ein dem Schwebeflug des Kolibris (*Trochilidae*) nachempfundener Schlagflug bei einer mittleren Reynoldszahl von etwa 13.000 simuliert. Die mit *RBF* erzielten Ergebnisse weisen alle erwarteten strömungsmechanischen Phänomene auf. Das gleiche gilt für die mit *AMI* umgesetzten Simulationen. Ein quantitativer Vergleich der Ergebnisse mit der Literatur fällt, aufgrund der mageren Datenlage und der als kritisch einzustufenden absoluten Werte der Simulationen schwer.

Schlagworte: Schlagflug, AMI, RBF, *dynamicTopoFvMesh*

Abstract

A new and innovative numerical method, *dynamicTopoFvMesh*, is tested towards its usability to simulate the movement of generalised flapping wings. Additionally, two other already established methods *Radial Basis Functions (RBF)* and *Arbitrary Mesh Interface (AMI)* are used to simulate 2D, respectively 3D wing movement. The details and requirements for all three methods are explored and explained in detail to be tested thereafter. While *dynamicTopoFvMesh* showed promising results initially, it could not be used for general wing movement due to programming error. However, *RBF* and *AMI* are both, in the authors opinion, (depending on suitable boundary conditions and dimensionality) capable of simulating arbitrary wing movement and therefore are the methods of choice to investigate transient high-lift effects. Eventually, a flapping wing movement, based on the wing movement of Hummingbirds (*Trochilidae*), has been implemented and simulated with an intermediate Reynoldsnumber of 13.000. Results for *RBF* and *AMI* show good qualitative compliance with experimental results, however a quantitative comparison is not possible, due to lack of data and questionable quality of achieved numerical results.

Keywords: flapping flight, AMI, RBF, *dynamicTopoFvMesh*

Inhaltsverzeichnis

1 Einleitung	1
2 Material und Methode	4
2.1 <i>dynamicTopoFvMesh</i>	4
2.1.1 Gitterqualität	5
2.1.2 Lokale Gitterverfeinerung	6
2.1.3 Gitterglättung	10
2.1.4 <i>dynamicMeshDict</i>	11
2.1.5 Testszenarien	14
2.2 Radiale Basisfunktionen	18
2.2.1 Kinematische Modelle	21
2.2.2 Numerischer Aufbau	26
2.3 <i>Arbitrary Mesh Interface</i>	31
2.3.1 Numerischer Aufbau	32
3 Ergebnisse	39
3.1 <i>dynamicTopoFvMesh</i>	39
3.1.1 Testszenario A	39
3.1.2 Testszenario B	40
3.2 Radiale Basisfunktionen	44
3.2.1 Gitterqualität & Gitterdeformation	44
3.2.2 Wirbelstärke	45
3.3 <i>Arbitrary Mesh Interface</i>	49
3.3.1 Gitterbewegung	49
3.3.2 Strömungsfeld & Strukturen	51
3.3.3 Sekundärströmungen	53
3.3.4 Kräfte	53
4 Diskussion	55
4.1 <i>dynamicTopoFvMesh</i>	55
4.1.1 Testszenario A	55
4.1.2 Testszenario B	55
4.1.3 <i>dynamicTopoFvMesh</i>	56
4.2 Radiale Basisfunktionen	58

4.3	<i>Arbitrary Mesh Interface</i>	60
5	Fazit & Ausblick	62
6	Danksagung	63
	Anhang	I

1 Einleitung

Die Natur bietet eine große Anzahl von faszinierenden Flugkünstlern. Allen gemein ist, dass im Gegensatz zu Flugzeugen ihr Flug ein hochdynamische Vorgang mit starken Bewegungen der Flügel ist. Im Reich der Insekten versprechen insbesondere Libellen mit ihren vier unabhängig steuerbaren Flügeln fluidmechanisch äußerst interessante Phänomene. Eine weitere faszinierende Spezies Stellen die Kolibries (*Trochilidae*) dar. Diese vergleichsweise kleinen Vögel sind zu spektakulären Flugmanövern befähigt. Auf Grund ihrer Nahrungsaufnahme, die Organismen ernähren sich ausschließlich von Nektar, sind sie außerdem zum Schwebeflug befähigt. Aber auch hohe Geschwindigkeiten von bis zu $13m \cdot s^{-1}$ können erreicht werden. Damit stellen sie eine der wenigen fliegenden Spezies dar, welche sich sowohl in laminaren als auch turbulenten Strömungen bewegen. Charles P. Ellington 2006 vermutet sogar, dass die kritische Reynoldszahl für diese Tiere unterhalb von $10 \cdot 10^3$ liegt, da ein wesentlich höheres Verhältnis von Auftrieb zu Widerstand erreicht wird als bei Insekten in vergleichbaren Reynoldsnummern.

Im Rahmen dieser Projektarbeit sollen die instationären Hochauftriebseffekte, welche bei verschiedenen Organismen beobachtet werden können numerisch untersucht werden. Dazu soll sowohl ein neuartiger Code, der als Erweiterung der frei zugänglichen CFD-Software OpenFOAM veröffentlicht wurde, als auch bereits etablierte numerische Techniken genutzt werden.

Die heute verfügbaren numerischen Methoden zur Untersuchung fluidmechanische Phänomene sind bereits sehr ausgereift und vielfach überprüft. Nichtsdestotrotz bietet das Forschungsgebiet nach wie vor große Innovationsmöglichkeiten. Insbesondere im Bereich der numerischen Modellierung und Diskretisierung. Während die Entwicklung verschiedener Modelle (Turbulenz, Phasenmodellierung) bereits auf Prandtl zurück geht (Prandtlsches Mischungswegmodell) und es bereits viele etablierte und vielfach überprüfte Modelle gibt (Beispiel Turbulenzmodellierung: $k-\varepsilon$ oder $k-\omega$), ist die räumliche Diskretisierung nach wie vor stark an die verfügbaren Rechenkapazitäten gebunden. Die bedeutet, dass unter Umständen Phänomene mit geometrischen Eigenschaften, welche sich über mehrerer Größenordnungen erstrecken oder instationäre Phänomene mit bewegten Objekten starken Abstraktionen unterworfen sind. (Beispiele dafür wäre die korrekte Darstellung der Wandrauheit einer Haifischhaut oder ein bewegter Insektenflügel mit einer maximalen Winkelabsenkung von 170°) Aufgrund der heute verfügbaren Rechenleistung können diese Abstraktionen nach und nach reduziert oder sogar ganz unterlassen werden. Eine weitere Möglichkeit diese Abstraktionen zu reduzieren besteht im Einsatz innovativer räumlicher Diskretisierungsmethoden.

Während die Abbildung von geometrischen Objekten über mehrere Größenordnungen immer an die verfügbaren Rechenressourcen gebunden ist, können zumindest die Bewegungen von Objekten durch den Einsatz neuer Methoden mit akzeptablen Rechenzeiten realisiert werden.

Um bewegte Objekte mit Hilfe von CFD zu simulieren, muss das zugrunde liegende räumliche Gitter, welches zur numerischen Diskretisierung eingesetzt wird, manipuliert werden. Sind die Deformationen klein (kleiner oder gleich der mittleren Größe der Zellen), so lässt sich die Deformation des Gitters einfach über die Verschiebung der Eckpunkte der Zellen realisieren. Größere Deformationen lassen sich ebenfalls über die Verschiebung der Zelleckpunkte realisieren, benötigen aber eine aktive Glättung der Deformation. Ohne aktive Glättung, werden die Zellen schnell so stark deformiert, dass die Schiefe, Volumina und nicht-Orthogonalität die zulässigen Grenzwerte überschreiten.

Diese Glättung lässt sich sehr einfach durch eine physikalische Analogie realisieren. Als Beispiel sei hier das (einfache) Federmodell erwähnt. Unter der Annahme, dass zwischen allen benachbarten Punkten Federn mit gleicher Federkonstante gespannt sind, lässt sich bei der Translation eines Eckpunktes im Raum über das Hoocksche Gesetz eine Kraft auf die benachbarten Zellpunkte errechnen. Die daraus resultierende Kraftverteilung über alle Punkte im Gitter kann unter der Translation der benachbarten Punkte minimiert werden. Dadurch ergibt sich ein gleichmäßigeres Gitter. Es existieren zahlreiche Modelle für diesen Ansatz, welche mit unterschiedlichen Federmodellen und Rechenverfahren arbeiten. Auch komplexere Glättungsmechanismen mit Hilfe von Lagrangefunktion oder sogenannter radialer Basisfunktionen sind bereits implementiert und erprobt worden. Weiter mögliche Glättungen basieren u. A. auf der Erhaltung (konstanter) Zellvolumen, -flächen und -kanten. Ziel der Glättung ist es die Interpolationsfehler der numerischen Lösung zu verringern und die Gitterqualität zu erhalten. Diese Techniken eignen sich gut für translativen Bewegungen beschränkter Amplitude, jedoch nur bedingt für rotative (Boer, Schoor und Bijl 2006; F. Bos u. a. 2007; Jasak und Rusche 2009).

An dieser Stelle sei noch erwähnt, dass für reine Rotationsbewegungen weitere, weitaus einfachere Methoden zur Verfügung stehen. OpenFOAM bietet für die Berechnung reiner Rotationsbewegungen beispielsweise eine spezielle, als AMI (Arbitrary Mesh Interface), bezeichnete Technik. Bei diesem Ansatz wird der rotierende Teil einer Geometrie in ein rotationssymmetrisches Gitter eingebettet, welches wiederum Teil eines größeren Gitters ist. An den Grenzflächen der Gitter werden die berechneten Größen interpoliert. Um entstehenden Interpolationsfehlern entgegen zu wirken, können die Zellen an den Grenzflächen stark verfeinert werden (Jasak und Tukovic 2006; Kassiotis 2008; Möller 2011; Jasak und Rusche 2009).

Ist die Bewegungsamplitude jedoch sehr groß und damit ebenfalls die Deformation der Zellen, reichen die oben genannten Ansätze jedoch nicht mehr aus (Jasak und Rusche 2009). Eine grundlegende Möglichkeit ist es, die Zellen bis zum Erreichen einer Zellqualitätsgrenze zu deformieren und beim Erreichen dieser Grenze die Simulation zu stoppen und ein neues

1 Einleitung

Gitter zu generieren. Dieser Ansatz ermöglicht zwar theoretisch beliebige Bewegungen, hat jedoch einige Nachteile. Die bis dahin errechneten Ergebnisse müssen auf ein komplett neues Gitter interpoliert werden, was Interpolationsfehler erzeugt und das Erstellen eines neuen Gitters ist rechenaufwändig, insbesondere für komplexe Geometrien. Daher liegt die Idee nahe, zunächst nur einzelne Zellbereiche neu zu generieren, deren Qualität einen Grenzwert unterschritten haben. Mit dieser Technik können große Bewegungsamplituden mit einem akzeptablen Rechenaufwand realisiert werden, ohne jedoch die Zellqualität zu gefährden (Menon 2011).

Eine weitere wünschenswerte Eigenschaft eines adaptiven Gitters sind lokale Zellverfeinerungen, beispielsweise an der Oberfläche eines Objekts, um die Grenzschicht sauber aufzulösen oder an Positionen, an denen starke Gradienten auftreten (durch Veränderungen der Geometrie).

2 Material und Methode

2.1 dynamicTopoFvMesh

Im Zuge dieser Projektarbeit soll die in der Einleitung vorgestellte Methode (*dynamicTopoFvMesh*) der lokalen Zell-Regenerierung getestet werden.

Die Arbeit wurde mit der frei verfügbaren Programmmbibliothek OpenFOAM (OpenFOAM Stiftung in Delaware, USA) durchgeführt. Die Implementierung *dynamicTopoFvMesh* wurde von Herrn Sandeep Menon in seiner Doktorarbeit 2011 vorgestellt und zu OpenFOAM kompatibel frei veröffentlicht. Der ursprünglich von Menon (2011) vorgesehene Anwendungsbereich liegt in der Berechnung Tropfenformen in Tintenstrahldruckern und gehört damit zu den Zweiphasenströmungen. Bereits in seiner Zusammenfassung weist Menon (2011) darauf hin, dass die Software auch für andere Zwecke eingesetzt werden könnte, insbesondere für die Berechnung von bewegten Flügeln.

Da sich die von ihm veröffentlichte Bibliothek *dynamicTopoFvMesh* noch in einem experimentellen Stadium befindet, werden nur Tetraedergitter im dreidimensionalen und Gitter basierend auf Prismen im zweidimensionalen unterstützt. Die Software ist in C++ als Erweiterungsbibliothek für OpenFOAM implementiert und nutzt die von OpenFOAM zur Verfügung gestellten Datentypen, Strukturen und Klassen. Der Namensraum (C++: *namespace*) der Bibliothek lautet *dynamicTopoFvMesh*.

Zunächst werden die verschiedenen für die Software benötigten und verfügbaren Techniken erläutert um dann zwei Testszenarien festzulegen. Im Anschluss daran werden die Ergebnisse dieser Testszenarien vorgestellt und diskutiert.

dynamicTopoFvMesh wird über eine Inputdatei (in OpenFOAM als *dict* bezeichnet)¹ gesteuert. In dieser Inputdatei wird zunächst der für die Gitterdeformation genutzte Gleichungslöser (*mesquiteMotionSolver*), der Typ von Gitter (*dynamicTopoFvMesh*) und die zu verwendende Bibliothek (*libdynamicTopoFvMesh.so*) festgelegt. Danach folgen die für den Löser und die Gitterbibliothek spezifischen Einträge. Menon (2011) sieht vor, dass die Bibliothek in Kombination mit der Bibliothek *Mesquite Mesh Quality Improvement Toolkit* (Sandia National Laboratories, Albuquerque, New Mexico, USA) genutzt wird. Diese Bibliothek stellt den für die Gitterqualität unerlässliche Gitter-Glättungs-Algorithmus zur Verfügung. Dieser wird über die Mesquite spezifischen Einstellungen innerhalb des *dynamicMeshDict* festgelegt. *Mesquite* stellt hierbei neun verschiedene Algorithmen zur Verfügung, unter an-

¹Die zu *dynamicTopoFvMesh* gehörende Datei heißt *dynamicMeshDict* und wird von allen OpenFOAM-Lösern, welche dynamische Gitter unterstützen, als Input genutzt.

derem mehrere einfache auf der Feder-Analogie und zwei auf einer Laplace-Gleichung der Gitterdeformationsgeschwindigkeit basierende. Im Zuge dieser Arbeit wurde der von Menon (2011) voreingestellte Algorithmus *feasibleNewton* eingesetzt. Kapitel 2.2.3 geht auf die zur Verfügung stehenden Methoden genauer ein.

Ein weiterer innerhalb der Mesquite-Einstellungen im *dynamicMeshDict* festgelegter Parameter ist die Toleranz. Diese wird zur Gitterglättung als Abbruchkriterium benötigt. Auf diesen und andere Parameter wird im Zuge dieser Arbeit nicht weiter eingegangen, da sie nicht dem Fokus der Arbeit entsprechen. Auch das zu bewegende Objekt wird innerhalb der Mesquite-Einstellungen festgelegt. Mesquite bietet insgesamt 22 Optionen die Bewegung eines Objekts zu steuern. Auf die wichtigsten drei soll im Folgenden kurz eingegangen werden.

- *angularOscillatingDisplacement*: Periodische, rotierende Gitterdeformation um einen fixen Punkt. Angegeben werden muss ein Normalenvektor, der Ursprungspunkt der Bewegung und die Winkelgeschwindigkeit.
- *angularDisplacement*: Periodische Deformation, oszillierend um einen Punkt. Es muss ebenfalls der Ursprung, ein Normalenvektor, und die Bewegungsamplitude angegeben werden.
- *calculated*: Diese Einstellung bietet die Möglichkeit die Gitterdeformation durch auf das Objekt wirkende Kräfte berechnen zu lassen. Es wird somit eine Kopplung von Strömung und Gitter ermöglicht.

Um das Verhalten der Gitterneugenerierung zu steuern, existiert im *dynamicMeshDict* ein Einstellungsbereich für *dynamicTopoFvMesh*². Hier wird zunächst festgelegt wie viele Zellmodifikationen maximal pro Zeitschritt durchgeführt werden dürfen. Des Weiteren wird auch festgelegt ob Zellen überhaupt neu generiert bzw. zerstört werden dürfen. Dies ermöglicht zwei grundlegende Simulationsmodi: Die Kombination aus Gitterglättung und Gitterneugenerierung oder lediglich Gitterglättung. Soll sowohl Gitterneugenerierung und Gitterglättung zum Einsatz kommen, muss in einem weiteren *SubDict* das Verhalten des dynamischen Gittergenerators definiert werden. Hierauf soll in den nachfolgenden Unterkapitel eingegangen werden.

2.1.1 Gitterqualität

Die elementar notwendige Größe für das Funktionieren der Gitterneugenerierung ist ein skalares Maß für die Gitterqualität. Diese skalare Größe wird für jede Zelle zu jedem Zeitschritt berechnet. Es stehen innerhalb von *dynamicTopoFvMesh* verschiedene skalare

²Diese 'Untereinstellungen' werden in OpenFOAM als *SubDict* bezeichnet.

Größen zur Verfügung, da OpenFOAM zunächst kein eigenes Maß für die Qualität von einzelnen Tetraederzellen zur Verfügung stellt.

Menon 2011 implementiert für Tetraeder folgendes skalares Qualitätsmaß:

$$q = \frac{3\sqrt{6}V_c}{L_{RMS}A_{RMS}} \quad (2.1)$$

Wobei V_c dem Zellvolumen, L_{RMS} dem quadratischen Mittel der Zellkantenlängen und A_{RMS} dem quadratischen Mittel der Zelloberflächen entspricht. Diese skalare Größe schwankt zwischen 0 und 1, wobei 1 für eine optimale Zellqualität und 0 für eine vollständig degenerierte Zelle steht. Sollten Werte unter 0 erreicht werden, bedeutet dies, dass die entsprechende Zelle ein negativtes Zellvolumen aufweist.

Eine weitere skalare Größe als Maß der Zellqualität wurde ebenfalls in der Bibliothek implementiert, jedoch im Rahmen dieser Arbeit nicht genutzt.

$$q = \frac{12(3V_c)^{2/3}}{\sum L_{e,i}^2} \quad (2.2)$$

Mit V_c = dem Zellvolumen und $L_{e,i}$ = der Kantenlängen der Zelle.

Des Weiteren gilt es weitere Gitterqualitätsparameter während einer Simulation zu beobachten. Dazu gehören unter anderem die nicht-Orthogonalität (OF: *nonOrthogonality*) und die Schiefe (OF: *skewness*). Dies wird über das Einbinden des *checkMesh*-Tools in die Strömungslöser, beispielsweise *pimpleDyMFoam* gewährleistet. Im Rahmen der zwei implementierten Testszenarien wurde nur mit dem skalaren Qualitätmaßstab nach Gleichung 2.1 gearbeitet.

Mit Hilfe der so berechneten skalaren Gitterqualität können die Zellen einer Rechendomäne Ω in Zellen mit guter Qualität ($q > \text{Grenzwert}$) und Zellen, welche zum Erhalt der Gesamtqualität manipuliert werden müssen ($q < \text{Grenzwert}$) unterteilt werden.

Sind diese Zellen identifiziert, stehen mehrere Methoden zur Verbesserung der Zellqualität zur Verfügung. Die Kombination von zwei Zellen zu einer einzelnen Zelle (Zellkollaps), das Aufteilen von einer Zelle in zwei Zellen (Zellteilung), das Aufteilen einer Zellfläche in drei Unterflächen und das Vertauschen von Zellgeraden nach dem Delaunay-Kriterium.

Des Weiteren sieht *dynamicTopoFvMesh* die Möglichkeit einer lokalen Gitterverfeinerung vor. Diese ermöglicht, unabhängig von der Bewegung des Objekts einen gleichbleibenden Größengradienten der Gitterzellen normal zur Oberfläche des bewegten Objekts.

2.1.2 Lokale Gitterverfeinerung

Um die automatische Verfeinerung der Gitterzellen in der Nähe oder an der Oberfläche von Objekten zu ermöglichen, berechnet *dynamicTopoFvMesh* eine lokale Vergleichslänge. Diese ermöglicht es für Oberflächen (OF: *patch*) eine maximal zulässige Zellgröße zu definieren,

welche in Richtung des freien Gittervolumens zu oder abnehmen kann. Um dies umzusetzen wurde in *dynamicTopoFvMesh* das sogenannte *h-refinement* eingeführt, welche das Gitter durch das Einfügen weiterer Gitterpunkte verfeinert. Diese Art von Verfeinerung wird auch als topologische Verfeinerung bezeichnet, da durch das Einfügen neuer Zellen die Topologie der Rechendomäne geändert wird. Diese lokale Vergleichslänge $L(x)$ wird für alle Gitterzellen, ähnlich wie das skalare Qualitätsmaß, berechnet und muss folgende Bedingungen erfüllen: Der Wert von $L(x)$ für $x = \partial\Omega$ muss für gekrümmte Oberflächen hinreichend klein sein um die Krümmung sauber abzubilden. Des Weiteren muss $L(x)$ für $x \in \Omega$ stetig zunehmen, so dass in Regionen mit geringen Gradienten der Rechenaufwand durch ein größeres Gitter reduziert werden kann. Um das Anwachsen der Zellgrößen zu realisieren wurde in *dynamicTopoFvMesh* ein Wachstumsfaktor implementiert, welcher pro zusätzlicher Zellschicht mit der lokalen Länge multipliziert werden kann. Dieser Wachstumsfaktor kann limitiert werden, sodass die Zellen nur bis zu einer definierten Größe L_{MEAN} anwachsen. Diese lokale Länge muss nicht für alle Oberflächen definiert werden, mindestens jedoch für eine.

Grundsätzlich stellt *dynamicTopoFvMesh* drei Möglichkeiten die lokale Vergleichslänge für *patches* zu definieren zur Verfügung.

- *Constant surface length scale*: definiert für eine Oberfläche einen konstanten Wert.
- *Length scale dictated by locale curvature*: Diese Konfiguration errechnet eine lokale Vergleichslänge anhand der Krümmung der Oberfläche. Dies bietet sich insbesondere für komplexe, gekrümmte Oberflächen, wie Flügelprofile an.
- *Length scale dictated by the proximity to a surface*: Diese Einstellung erlaubt es die lokale Länge anhand der Entfernung zu einem *patch* zu definieren. Dies ist insbesondere nützlich um ein nicht-lineares Wachstum der Zellgrößen zu realisieren.

Im Rahmen dieser Arbeit wurde das Festlegen eines konstanten Werts für den bewegten *patch* genutzt. Ist innerhalb des *dynamicTopoFvMeshDict* kein fester Wert für einen *patch* festgelegt, so wird die lokale Vergleichslänge mit folgender Formel berechnet:

$$l = \sqrt{2 \cdot A_{\partial\Omega,i}} \quad (2.3)$$

Mit $A_{\partial\Omega,i}$ = der Fläche der i -ten Zelle an der Oberfläche der Geometrie.

Ausgehend von der Oberfläche eines *patches* und den lokalen Vergleichslängen der ersten an den *patch* anschließende Zellschicht wird über alle Zellen iteriert und die Längenskala für alle Zellen in der Domäne errechnet. Um ein zu starkes Zellwachstum zu verhindern, können für die Vergleichslänge Minima und Maxima definiert werden.

Ist die Verfeinerung der Zellen an einer Oberfläche eines *patches* nicht erwünscht (beispielsweise an Wänden ohne Haftbedingung oder an Ein- und Ausgang), so können diese *Patches* als *freeLengthScalePatches* definiert werden; Dies führt dazu, dass die Zellgröße an diesen

Patches nicht festgelegt wird. Ist die Modifikation von *Patches* in keiner Weise erwünscht, so lässt sich dies über die Definition der Patches als *noModificationPatches* erreichen.

Da diese lokale Vergleichslänge nicht generisch in jedem Gittergenerator vorhanden ist, wurde im Rahmen dieser Arbeit eine Software geschrieben, welche für alle *patches* eines Gitters die mittlere Vergleichslänge berechnet. Dies ermöglicht das korrekte Einstellen der Vergleichslängen in der Datei *dynamicMeshDict*. Der Quellcode dieser Software findet sich im Anhang (siehe Listing 1) und ist unter der GNU GPL-Lizenz frei verfügbar.

Die nachfolgend beschriebenen Mechanismen werden von *dynamicTopoFvMesh* automatisch, in Abhängigkeit der festgelegten Werte im *dynamicMeshDict* zu jedem Zeitschritt durchgeführt.

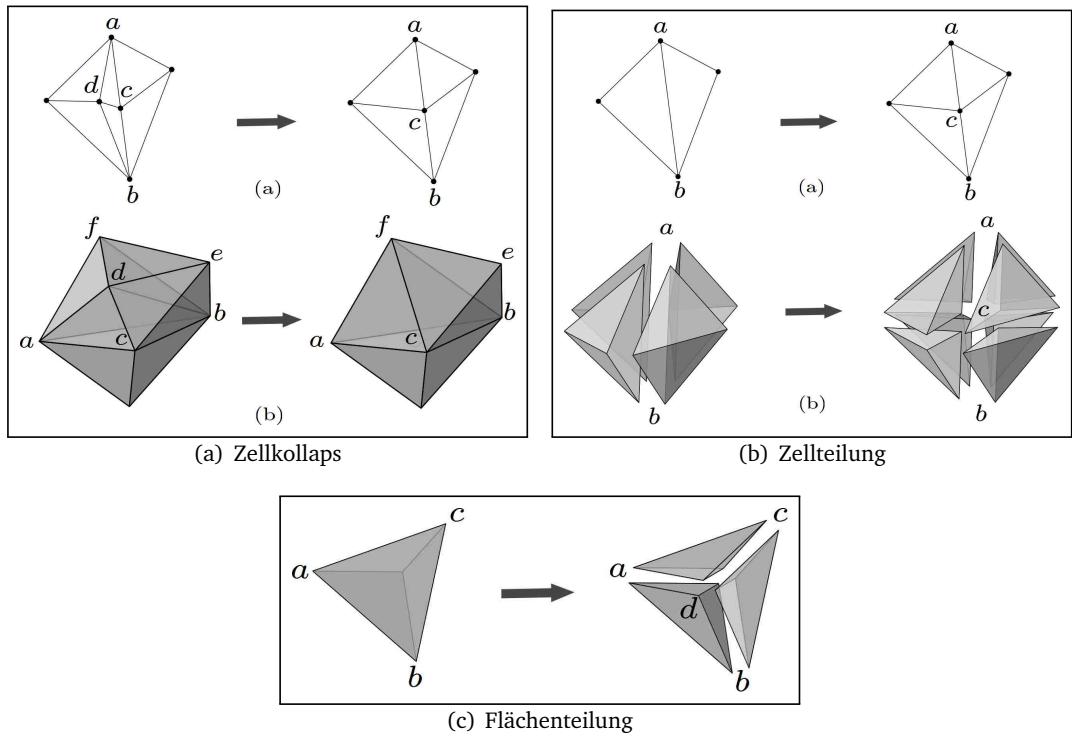


Abbildung 2.1: Die drei verschiedenen von *dynamicTopoFvMesh* zur Verfügung gestellten Methoden der Zellmanipulation. (a) Das Zusammenfügen zweier gestauchter Zellen zu einer. (b) das Zerteilen einer großen Zelle in zwei. (c) Das Aufteilen einer Zelle in drei Zellen durch das Einfügen einer neuen Zellkante. Verändert nach Menon 2011

Zellkollaps

Werden durch die Gitterdeformationen zwei aneinander grenzende Zellen stark zusammen gedrückt, können diese Zellen zu einer neuen Zelle kombiniert werden. Um festzulegen,

ab was für einem Deformationsverhältnis die Zellen zusammengefügt werden sollen (in *dynamicTopoFvMeshDict: collapseRatio*), muss in *dynamicMeshDict* ein skalarer Wert für das Verhältnis der lokalen Vergleichslängen festgelegt werden. Dabei muss beachtet werden, dass dieses Verhältnis mit dem festgelegten Wachstumsfaktor verrechnet wird. Abbildung 2.1(a) zeigt schematisch das Kollabieren von stark gedrückten, zwei- und dreidimensionalen Zellen. Zu beachten ist, dass der Wert des Verhältnisses der Vergleichslängen kleiner als eins sein muss.

Zellteilung

Ähnlich dem Kollabieren der Zellen, können stark gedehnte Zellen auch zerteilt werden. Abbildung 2.1(b) zeigt dies schematisch für zwei- und dreidimensionale Zellen. Auch hier muss das Verhältnis der lokalen Vergleichslängen angegeben werden (OF: *dynamicTopoFvMeshDict: bisectionRatio*). Dieser Wert muss jedoch größer als 1 sein.

Flächenteilung

Eine dritte Möglichkeit zur lokalen Gittermanipulation ist das Zerteilen einer Zelle in drei neue Zellen durch das Aufsplitten im Mittelpunkt einer der Dreiecksflächen. Dies wird in *dynamicTopoFvMesh* als *slicing* bezeichnet. Auch hier muss ein skalarer Wert angeben werden, der sich aus dem Verhältnis von Flächen zu Höhen ergibt. Dieses Verhältnis muss kleiner als 1 sein. Abbildung 2.1(c) zeigt das *slicing* schematisch.

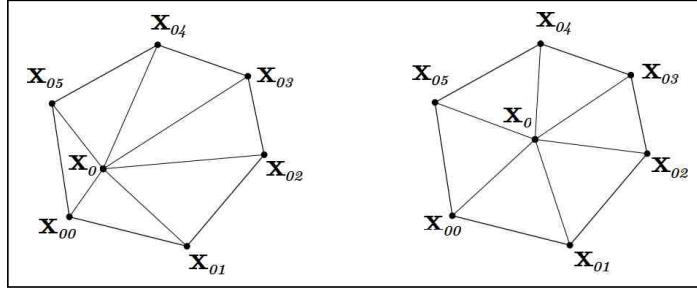


Abbildung 2.2: Die Zellglättung nach Gleichung 2.4. Schematische Darstellung nach Menon 2011. Durch das Anpassen der Zellabstände zwischen dem Punkt x_0 und den umliegenden Punkten $x_{0,i}$ wird der Punkt x_0 in die Mitte der umliegenden Punkte bewegt. Verändert nach Menon 2011.

2.1.3 Gitterglättung

Wie in Kapitel 2.1 bereits erwähnt ist die Gitterglättung von eklatanter Wichtigkeit um eine akzeptable Gitterqualität zu erhalten. Wie bereits kurz erläutert, liegt den meisten Methoden eine physikalische Analogie gespannter Federn zwischen den Gitterpunkten zugrunde. Mathematisch ausgedrückt bedeutet die Federanalogie, dass die Strecke zwischen den verbundenen Gitterpunkten möglichst überall dem mittleren Zellabstand entspricht. Das bedeutet, dass der folgende Ausdruck minimiert werden muss:

$$\sum_j (x_{ij} - x_i) = 0 \quad (2.4)$$

Wobei x_{ij} allen benachbarten Punkten des Punktes x_i entspricht. Abbildung 2.2 zeigt schematisch für eine zweidimensionale Zelle diesen Ansatz.

Dieses Problem kann gut mit dem konjugierten-Gradienten Verfahren gelöst werden. Es existieren eine Vielzahl von Implementierungen der obigen Methode mit unterschiedlichen Gewichtungen und Federanalogen. Ein grundsätzliches Problem dieser Methode ist jedoch, dass Zellen mit extrem schlechter Qualität, negativem Zellvolumen oder starker Deformation nicht mehr optimiert werden können. Aus diesem Grund wurde von Menon (2011) eine eigene Methode für die Optimierung von stark gekrümmten Oberflächen entwickelt. Für zweidimensionale Fälle wird in *dynamicTopoFvMesh* auf den obigen Ansatz zurückgegriffen. Für dreidimensionale Fälle wird für das innere Gitter die Mesquite-Bibliothek zur Gitterglättung genutzt, die Gitterzellen an den *Patches* werden mit der Methode von Menon (2011) geglättet. Bei dieser Methode wurde Optimierungsparameter die Zellqualität nach Gleichung 2.1 genutzt. Es werden verschiedene Operationen durchgeführt (u.a. das nach den Zellflächen gewichtete Verschieben von Gitterpunkten) um die Gitterqualität zu verbessern. Kann keine Verbesserung erreicht werden, so wird eine der Gitterverfeinerungsmethoden aus Kapitel 2.2 angewendet.

2.1.4 *dynamicMeshDict*

Im Nachfolgenden ist das im ersten Testszenario genutzte *dynamicMeshDict* als Beispiel aufgeführt.

Listing 2.1: Beispiel für *dynamicMeshDict* für Testszenario A

```

1 solver           mesquiteMotionSolver; // Bewegungslöser
2 dynamicFvMesh   dynamicTopoFvMesh; // Gittertyp
3 dynamicFvMeshLibs ("libdynamicTopoFvMesh.so"); // Name der Bibliothek
4 mesquiteOptions { // optionen für den Bewegungslöser und das Glätten
5     optMetric    AspectRatioGamma; // Vergleichsgröße zum Glätten des Gitters
6     objFunction  LPtoP;
7     optAlgorithm FeasibleNewton; // Optimierungsalgorithmus
8     tcInner { // Abbruchkriterium für das Glätten
9         absGradL2 1e-4;
10        cpuTime    1.25;
11    }
12    // tcOuter
13    // {}
14    // Folgende Parameter werden für hybride Glättungsfunktionen benötigt
15    firstFunction LPtoP;
16    secondFunction LInf;
17    // scaleFunction PMeanP;
18    // scale      1.5;
19    pValue       2;
20    power        2;
21    tolerance    1e-4; // Toleranz für das Konjugierte-Gradienten-Verfahren
22    nSweeps      3; // Anzahl der KG-Iterationen
23    slipPatches { } // Slip-Patches können hier definiert werden
24    debug        1; // Debugging-Informationen
25    surfInterval 1; // Wie oft soll geglättet werden?
26    fixedValuePatches { // Feste Randbedingungen für bewegte Patches
27        wing {
28            type      angularOscillatingDisplacement;
29            amplitude 0.1;
30            axis      (1 0 0);
31            origin    (5 5 5);
32            angle0   0.0;
33            omega     0.05;
34            value     uniform (0 0 0);
35        }
36    }
37 }
38 coupledPatches { } // hier können patches gekoppelt werden

```

```

39 // Haupteinstellungen für die Gitterregeneration
40 dynamicTopoFvMesh {
41     debug 1;
42     allOptionsMandatory no;
43     removeSlivers no;
44     sliverThreshold 0.2;
45     dumpLengthScale yes; // lokale Vergleichslänge zu jedem Zeitschritt
46     aus schreiben?
47     loadMotionSolver true;
48     threads 1; // anzahl benutzter System-Prozesse
49     interval 1; // wie oft soll Regeneriert werden?
50     maxModifications 10000; // Anzahl maximaler Modifikationen
51     swapDeviation 0.5; // Ab wann sollen Kanten getauscht werden?
52     // maxTetsPerEdge 500;
53     allowTableResize yes;
54     // Options for edge-bisection/collapse
55     edgeRefinement yes;
56     refinementOptions {
57         collapseRatio 0.5; // Vergleichslängen-Verhältnis für Zellkollaps
58         bisectionRatio 3; // Vergleichslängen-Verhältnis für Zellteilung
59         growthFactor 1.20; // Wachstumsfaktor -> 20 \%
60         sliceThreshold 0.5; // Vergleichslängenverhältnis für Flächenteilung
61         maxLengthScale 5; // Maximale Vergleichslänge -> deckelt die Zellgröße
62         minLengthScale 0.01; // Minimale Vergleichslänge -> legt die min. Zellgröß
63         e fest.
64         sliverThreshold 0.2;
65         fixedLengthScalePatches { } // hier können Vergleichslängen für patches
66         festgelegt werden
67         freeLengthScalePatches { } // ist die Vergleichslänge egal, werden die
68         patches hier definiert
69         inlet; outlet; walls;
70     }
71     noModificationPatches { } // sollen an bestimmten patches keine
72     modifikationen durchgeführt werden, werde diese patches hier definiert
73     curvaturePatches { } // soll die Vergleichslänge über die Krümmung
74     berechnet werden, müssen die patches hier definiert werden.
75     curvatureDeviation 0.5; // Krümmung
76     proximityPatches { } // soll die Vergleichslänge über den Abstand eines
77     bestimmten patches berechnet werden, so müssen die patches hier
78     definiert werden
79     /*
80     // eine weitere Verfeinerungsmöglichkeit ist das festlegen eines Zellgrößen
81     -Gradienten
82     fieldRefinement gamma;
83     fieldLengthScale 0.005;

```

```
75     lowerRefineLevel 0.001;
76     upperRefineLevel 0.999;
77     maxRefineLevel    4;
78     meanScale         0.015;
79     fieldScaleMethod direct;
80     fieldGradient    1;
81     */
82 }
83 tetMetric Knupp; // Lokale skalare Gitterqualität
84 noSwapPatches { // soll das vertauschen von Zellkanten an bestimmten patches
85     unterbunden werden, müssen diese patches hier festgelegt werden
86     wing;
87 }
88 loadBalancing { // für die Nutzung in Kombination mit MPT
89     enabled  false;
90     interval 100;
91     method   parMetis;
92     numberOfSubdomains 2;
93     mergeTol 1e-06;
94 }
```

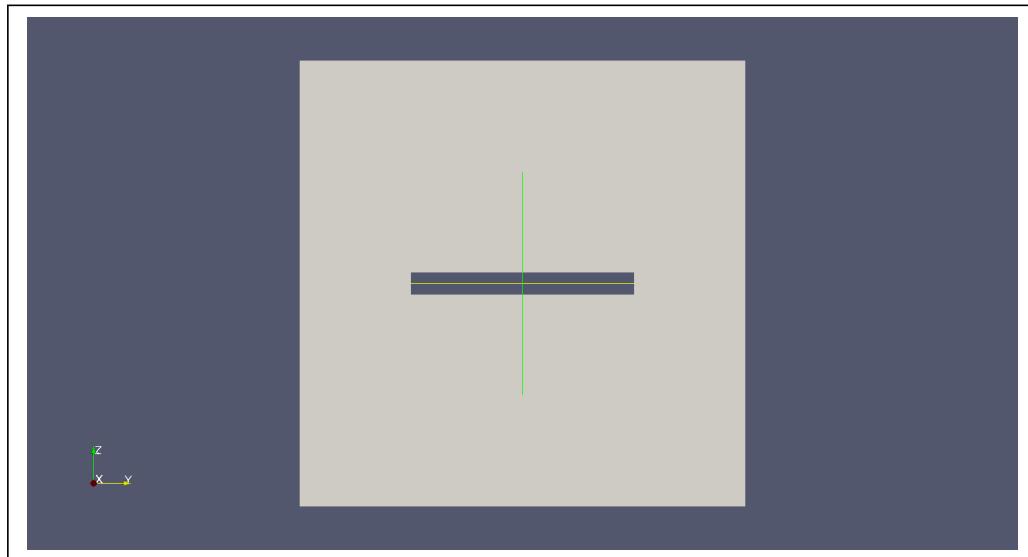


Abbildung 2.3: Die Testgeometrie für das Testszenario A. Die Platte hat die Dicke 0.1, die Länge 1 und die quadratische Box hat die Kantenlänge von 2

2.1.5 Testszenarien

Es wurden zwei Testfälle implementiert. Diese Testfälle wurden nicht als CFD-Simulationen ausgelegt, sondern nur mit dem von OpenFOAM zur Verfügung gestellten Gitterlöser *moveDynamicMesh* auf das Verhalten des Gitters getestet. Dies ermöglichte relative grobe Gitter und damit einhergehend geringe Zellanzahlen.

Testszenario A

Ziel des ersten Testszenarios war es zu untersuchen, wie das Zusammenspiel der Parameter von *dynamicTopoFvMesh* funktioniert. Es wurde sowohl Translation als auch Rotation getestet. Aufgrund der Punktsymmetrie der Geometrie wurde Rotation nur um die X-Achse und Translation in X und Z Richtung getestet.

Geometrie Es wurde eine quadratische Platte mit der Kantenlänge 1 LE und der Dicke 0.1 LE in einen Würfel mit der Kantenlänge von 2 LE eingebettet. Abbildung 2.3 zeigt die Geometrie. Die Entdimensionierung der Länge erfolgte über die Länge der Platte. Die Geometrie wurde mit Hilfe der frei verfügbaren CAD-Software Salome erstellt.

Gitter Die Gittergenerierung wurde mit der frei verfügbaren Software Netgen (Quelle einfügen) durchgeführt. Netgen generiert ausgehend von einer definierten Länge zunächst eindimensionale diskrete Punkte an der Oberfläche der Geometrie. Diese werden dann zu

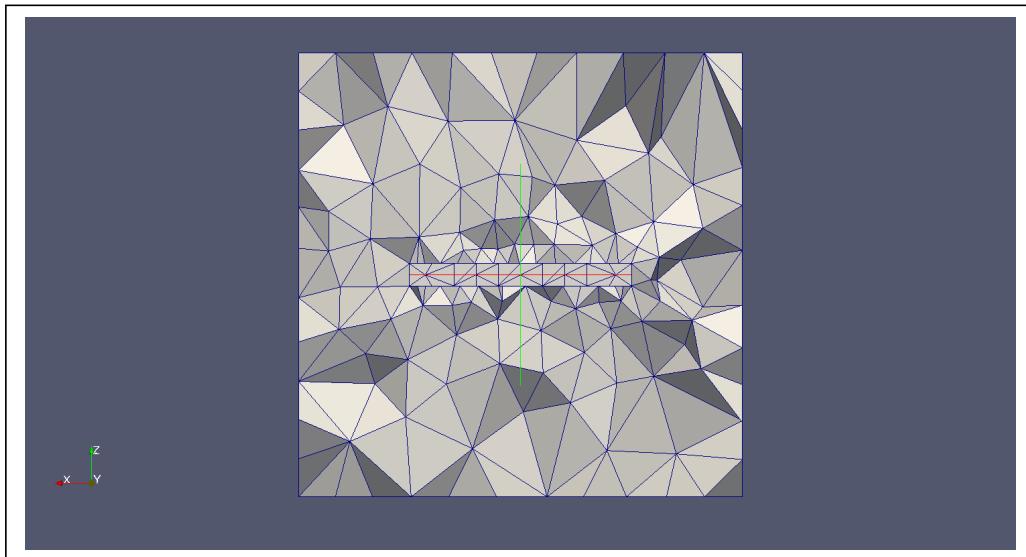


Abbildung 2.4: Das Gitter für das erste Testszenario. Schnitt entlang der Zellen normal zur Y-Achse durch den Mittelpunkt.

Dreiecken zusammengefügt und schließlich das Volumen mit Tetraedern aufgefüllt. Tabelle Tabelle 2.1 zeigt die wichtigsten Gitterparameter.

Tabelle 2.1: Die Gitterparameter des ersten Testszenarios. Es sind lediglich die für *dynamicTopoFv-Mesh* relevanten Parameter aufgeführt.

Parameter	Wert
Anzahl der Zellen (Tetraeder)	7391
Orthogonalität	≤ 60.537
Schiefe	≤ 0.9107
Zellflächen	min. $0.0058 LE^2$ / max. $0.5376 LE^2$
Zellvolumen	min. $7.156 \cdot 10^{-4} LE^3$ / max. $0.2246 LE^3$

Wichtig für eine erfolgreiche CFD-Simulation sind die maximale nicht-Orthogonalität (OF: *non-orthogonality*) und die maximale Schiefe (OF: *skewness*). Die nicht-Orthogonalität sollte 65° nicht übersteigen, während die Schiefe innerhalb der Rechendomäne 4 und an den Rändern 20 nicht übersteigen sollte.

Abbildung 2.4 zeigt das Gitter zum Beginn der Simulation.

Lösungsalgorithmus und Gleichungslöser Da keine Überströmung gerechnet wurde, sondern lediglich die Gitterdeformation, kam der zu OpenFOAM gehörende Gleichungslöser *moveDynamicMesh* zum Einsatz. Dabei müssen insgesamt zwei Gleichungssysteme gelöst wer-

den: Die Verschiebung der Gitterpunkte und die dazugehörige Geschwindigkeit mit welcher sich die Gitterpunkte durch die Domäne bewegen. Dieser wird für das Glätten des Gitters benötigt. Für beide Gleichungssysteme wurde ein präkonditioniertes Gradientenverfahren eingesetzt (OF: PCG). Das dazugehörige Prädiktionsverfahren ist ein diagonales unvollständiges nach Cholesky.

Tabelle 2.2 zeigt im einzelnen die im *dynamicMeshDict* gesetzten Parameter.

Tabelle 2.2: Einstellungen in *dynamicMeshDict* für Testszenario A

Parameter	Wert
Optimierungsalgorithmus	<i>feasibleNewton</i>
Toleranz Gleichungslöser	$1 \cdot 10^{-4}$
Glättungsintervall	jeder Zeitschritt
Winkelgeschwindigkeit ω	0.05
Entfernen spezieller, degenrierter Zellen (<i>slivers</i>)	negativ
Regenerierungsintervall	jeder Zeitschritt
max. Zellmodifikationen	$1 \cdot 10^4$
<i>collapseRatio</i>	0.5
<i>bisectionRatio</i>	3
<i>growthFactor</i>	1.2
<i>maxLengthScale</i>	5
<i>minLengthScale</i>	0.01
<i>fixedLengthScalePatches</i>	Platte (0.25)
<i>freeLengthScalePatches</i>	Wände
Qualitätsgröße für einzelne Zellen	Gleichung 2.1

Testszenario B

Im zweiten Testszenario sollte insbesondere die Translation eines gekrümmten Objekts durch die Rechendomäne untersucht werden. Dieser Testfall wurde auch von Menon (2011) eingesetzt.

Geometrie Es wurde eine Kugel mit dem Durchmesser 1 LE und dem Wandabstand 1.5 LE in eine Rechteckige Domäne mit der Länge 10 LE und den Seitenkantenlängen von 4 LE eingebettet. Abbildung 2.5 zeigt die Geometrie zum Anfangzeitpunkt.

Gitter Die Gittergenerierung wurde ebenfalls mit Netgen durchgeführt.

Tabelle 2.3 zeigt die wichtigsten Gitterparameter zum Beginn der Simulation.

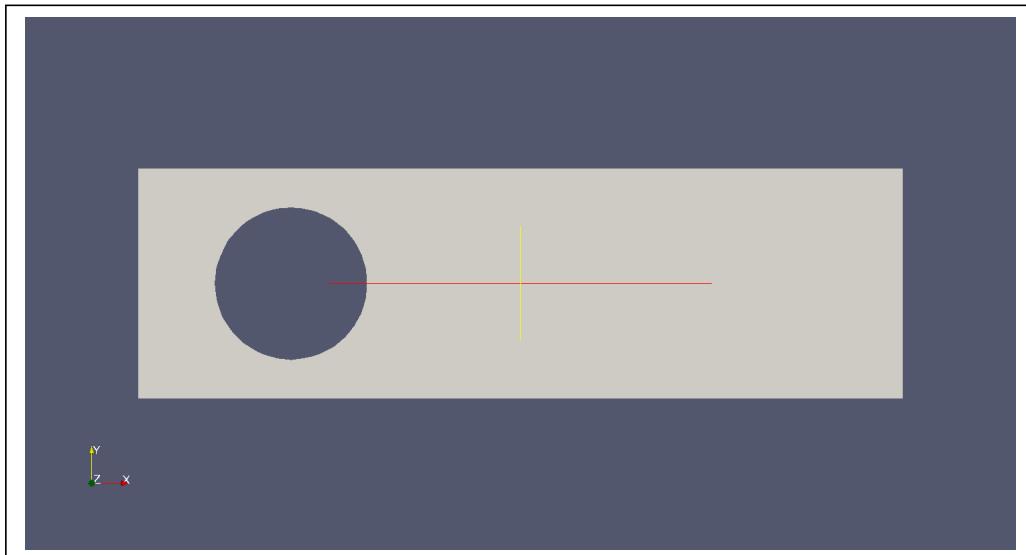


Abbildung 2.5: Die für das zweite Testszenario eingesetzte Geometrie.

Tabelle 2.3: Die Gitterparameter des zweiten Testszenarios. Auch hier wurden nur die für *dynamic-TopoFvMesh* relevanten Parameter angegeben.

Parameter	Wert
Anzahl der Zellen (Tetraeder)	10828
max. Orthogonalität	≤ 45.1402
max. Schiefe	≤ 0.405
Zellflächen	min. $0.00432LE^2$ / max. $0.46335 LE^2$
Zellvolumen	min. $1.91 \cdot 10^{-4}LE^3$ / max. $0.1012 LE^3$

Die maximal zulässige nicht-Orthogonalität und die Schiefe wurden auch hier nicht überschritten. Abbildung 2.6 zeigt das Gitter zum Beginn der Simulation.

Lösungsalgorithmus und Gleichungslöser Es wurde für das zweite Testszenario ebenfalls *moveDynamicMesh* eingesetzt. Als Gleichungslöser kam auch hier das präkonditionierte Gradientenverfahren mit dem unvollständigen, diagonalen Cholesky Prädiktionierer zum Einsatz. Tabelle 2.4 listet die gesetzten Parameter für die Simulation.

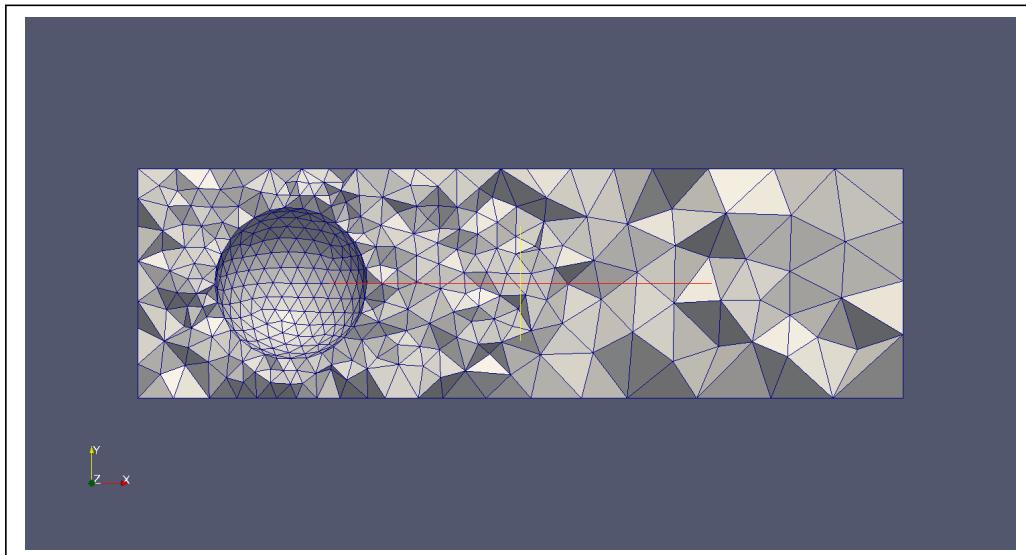


Abbildung 2.6: Das für das zweite Testszenario eingesetzte Gitter.

Tabelle 2.4: Einstellungen in *dynamicMeshDict* für das Testszenario B

Parameter	Wert
Optimierungsalgorithmus	<i>feasibleNewton</i>
Toleranz Gleichungslöser	$1 \cdot 10^{-2}$
Glättungsintervall	jeder Zeitschritt
Winkelgeschwindigkeit ω	0.005
Entfernen spezieller, degenrierter Zellen (<i>slivers</i>)	positiv
Regenerierungsintervall	jeder Zeitschritt
max. Zellmodifikationen	nicht limitiert
<i>collapseRatio</i>	0.7
<i>bisectionRatio</i>	1.5
<i>growthFactor</i>	1.02
<i>sliverThreshold</i>	0.4
<i>maxLengthScale</i>	nicht gesetzt
<i>minLengthScale</i>	nicht gesetzt
<i>fixedLengthScalePatches</i>	Kugel (0.2)
<i>freeLengthScalePatches</i>	Wände
Qualitätsgröße für einzelne Zellen	Knupp (Gleichung 2.1)

2.2 Radiale Basisfunktionen

Radiale Basisfunktionen (*RBF*) umfassen eine Gruppe von Funktionen welche nur von einem Radius, einem Koordinatenursprung und einer Anzahl von Funktionsparametern

abhangen. Diese Funktionen können genutzt werden um die Bewegungen der Gitterpunkte in numerischen Gittern zu berechnen. Sie haben den Vorteil, dass sie unabhängig von der Verbundenheit der Gitterpunkte sind. Ausgehen von den Punkten auf der Oberfläche eines bewegten Objekts, wird die Bewegung auf die umliegenden Punkte interpoliert (Boer, Schoor und Bijl 2006). In OpenFOAM ist die Möglichkeit *RBF* zu nutzen seit Version 1.6 der Community-Edition von OpenFOAM, *foam-extended* vorhanden. Dabei wird zur Laufzeit ein kinematisches Modell zur Beschreibung der Bewegung kompiliert und in den genutzten Löser eingebunden. Dabei bietet der *RBF*-Löser in OpenFOAM noch die Möglichkeit einen Radius um das Objekt zu definieren, innerhalb dessen die Gitterpunkte nicht bewegt werden dürfen, sondern ihre Ausrichtung zum bewegten Objekt beibehalten ((F. Bos u. a. 2007)).

Es stehen innerhalb von *foam-extended* verschiedene *RBF*-Funktionen zur Verfügung. Im Rahmen dieser Arbeit wurde eine inverse quadratische harmonische Funktion genutzt welche folgender Gleichung genügt:

$$IMQB(r) = \sqrt{\frac{1}{a^2 + r^2}} \quad (2.5)$$

Nach (Boer, Schoor und Bijl 2006) sollte der Parameter a sehr klein gewählt, aus diesem Grund wurden die Einstellungen aus dem Beispielfall von Boer, Schoor und Bijl (2006) ($a = 0.001$) beibehalten.

Um *RBF* in OpenFOAM nutzen zu können, musste zunächst der passende Eintrag in der Datei *dynamicTopoFvMesh* angelegt werden. Listing 2.2 listet die Einstellungen in der Datei *dynamicMeshDict*.

Listing 2.2: *dynamicTopoFvMesh* für *RBF*-Simulationen

```

1 // hier wird der Gittertyp festgelegt.
2 dynamicFvMesh dynamicMotionSolverFvMesh;
3
4 // Dieser Eintrag weist den Lösungsalgorithmus an, welcher Bewegungslöser
5 // genutzt werden soll
6 solver RBFMotionSolver;
7
8 // Definition des zu bewegenden Gitterobjekts
9 movingPatches ( wing );
10
11 // Sollen bestimmte Ränder des Gitters von der Bewegung unabhängig bleiben,
12 // müssen diese hier definiert werden.
13 staticPatches ( left right top bottom );
14
15 coarseningRatio 5;
16
17 includeStaticPatches    no;
18 frozenInterpolation     yes;
```

```

19 // hier wird die interpolationsmethode festgelegt
20 interpolation
21 {
22     RBF IMQB; // Inverse Multiquadratic Biharmonics
23     focalPoint (0.25 0 0); // Nullpunkt von Rotation und Translation
24
25     // dieser Radius definiert einen Kreis, innerhalb dessen das Gitter nicht
26     // manipuliert wird. Dies ermöglicht eine gleichbleibend hohe Gitterqualität
27     // an der Oberfläche des zu untersuchenden Objekts.
28     innerRadius 2.0;/5.0
29
30     // dieser Radius definiert den maximalen Bereich innerhalb dessen
31     // die Gitterzellen manipuliert werden dürfen
32     outerRadius 10.0;
33
34     polynomials true; // dürfen Polynome zur interpolation eingesetzt werden?
35
36 // hier werden die Parameter der oben ausgewählten Funktion definiert.
37 W2Coeffs
38 {
39     radius      1.0;
40 }
41 TPSCoeffs
42 {
43     radius      5.0;
44 }
45 GaussCoeffs
46 {
47     radius      0.1;
48 }
49 IMQBCoeffs
50 {
51     radius      0.001;
52 }
53 }
54
55 // ****

```

Des Weiteren mussten die zur Bewegung benötigten Initial- und Randbedingungen in der Datei *controlDict* festgelegt werden. Listing 2.3 zeigt im Auszug aus der Datei *controlDict* die benötigten Parameter.

Listing 2.3: Einstellungen *controlDict* für RBF-Simulationen

```
1 functions
```

```

2 (
3   RBFMotion
4   {
5     type RBFMotion;
6     functionObjectLibs ("libRBFMotionFunction.so");
7     rotationAmplitude      -0.707; // Maximale Rotation in rad / s
8     rotationFrequency       1;    // Rotationsfrequenz
9     translationAmplitude   (0.0 0.0 1.0); // Maximale Translation
10    translationFrequency   (0.0 0.0 1.0); // Translationsfrequenz
11    initialRotationOrigin  (0.25 0.0 0.0); // Rotationsursprung
12  }
13 );

```

Im Rahmen dieser Arbeit wurde die Bewegung eines einfachen, zweidimensionalen NACA-Profil (NACA-0006) untersucht. Es wurden zunächst zwei verschiedene kinematische Modelle genutzt; das erste kinematische Modell entstammte den Beispielen aus *foam-extended* und ist in Listing 2.4 gelistet. Das zweite kinematische Modell wurde vom Autor neu geschrieben um den Ansprüchen an die Simulationen zu genügen. Listing 2.5 listet den Quellcode des zweiten kinematischen Modells. Es wurden für jedes kinematische Modell jeweils zwei exemplarische Simulationen durchgeführt. Es kam allerdings nur eine Simulation zur Auswertung. In erster Linie war das Ziel dieser Simulationen die Funktionsfähigkeit der Gittermanipulation mit Hilfe von Radialen Basisfunktionen zu demonstrieren. Des Weiteren sollten für den Schlagflug im niedrigen Reynoldszahlbereich typische Wirbelstrukturen identifiziert werden.

2.2.1 Kinematische Modelle

Das erste kinematische Modell beschreibt eine zweidimensionale, harmonische Schwingung, sowohl translativ als auch rotativ. Die Bewegung findet dabei in der X-Z-Ebene statt. Translativ wird nur in Richtung der Z-Achse bewegt, rotativ wird um die Y-Achse rotiert. Die translativen Bewegung genügt dabei folgender Gleichung:

$$z(t) = A_t \cdot \sin(2\pi \cdot f_t \cdot t) \quad (2.6)$$

Mit A_t = Amplitude der Translation und f_t = Frequenz der Translation.

Die Rotation wird durch folgende Gleichung beschrieben:

$$\alpha(t) = A_r \cdot \sin(2\pi \cdot f_r \cdot t) \quad (2.7)$$

Mit A_r = Amplitude der Rotation und f_r = Frequenz der Rotation.

Die Werte für $z(t)$ und $\alpha(t)$ wurden zu jedem Zeitschritt berechnet. Die Werte des vorhergegangenen Zeitschritts wurden von den neuen Werten abgezogen und aus dem Ergebnis wurde die Rotationsmatrix, respektive der Translationsvektor gebildet.

Listing 2.4: einfaches kinematisches modell

```

1 vectorField motion(ms.movingPoints().size(), vector::zero);
2 vectorField oldPoints = ms.movingPoints();
3 scalar oldTime = time_.value() - time_.deltaT().value();
4 scalar curTime = time_.value();
5 scalar alphaOld = 0.0;
6 scalar alphaCur = 0.0;
7 scalar pi=3.141592;
8
9 alphaOld = rotationAmplitude_*Foam::sin(2*pi*rotationFrequency_*oldTime);
10 alphaCur = rotationAmplitude_*Foam::sin(2*pi*rotationFrequency_*curTime);
11
12 vector translationVector
13 (
14     translationAmplitude_[0]*
15     (
16         Foam::sin(2*pi*translationFrequency_[0]*curTime)
17         - Foam::sin(2*pi*translationFrequency_[0]*oldTime)
18     ),
19     0,
20     translationAmplitude_[1]*
21     (
22         Foam::sin(2*pi*translationFrequency_[1]*curTime)
23         - Foam::sin(2*pi*translationFrequency_[1]*oldTime)
24     )
25 );
26
27 tensor RyOld
28 (
29     Foam::cos(alphaOld), 0, -Foam::sin(alphaOld),
30     0, 1, 0,
31     Foam::sin(alphaOld), 0, Foam::cos(alphaOld)
32 );
33
34 tensor RyCur
35 (
36     Foam::cos(alphaCur), 0, -Foam::sin(alphaCur),
37     0, 1, 0,
38     Foam::sin(alphaCur), 0, Foam::cos(alphaCur)
39 );
40
41
42 vectorField rotationField
43 (
44     (RyCur - RyOld)

```

```

45     & (statPoints_ - initialRotationOrigin_);
46 );
47
48 motion = translationVector + rotationField;

```

Da die Bewegungen eines vereinfachten Flügels nach biologischem Vorbild nicht dem einfachen kinematischen Modell genügte, wurde ein zweites kinematisches Modell entwickelt, welches zunächst nur eine Translation beinhaltet und erst beim Erreichen einer festgelegten relativen Amplitude zu rotieren beginnt. Die Translative Bewegung genügte dabei Gleichung 2.6. Listing 2.5 listet den Quellcode des zweiten kinematischen Modells.

Listing 2.5: komplexes kinematisches Modell

```

1 vectorField motion(ms.movingPoints().size(), vector::zero);
2 vectorField oldPoints = ms.movingPoints();
3 scalar oldTime = time_.value() - time_.deltaT().value();
4 scalar curTime = time_.value();
5 scalar alphaOld = 0.0;
6 scalar alphaCur = 0.0;
7 scalar rotThreshold = 0.8 * translationAmplitude_.z();
8 Info << "rotation threshold is: " << rotThreshold << endl;
9 vector position (0.0, 0.0, 0.0);
10 scalar pi = 3.141592;
11
12 vector translationVectorXZ
13 (
14     translationAmplitude_[0]*
15     (
16         Foam::sin(2*pi*translationFrequency_[0]*curTime)
17         - Foam::sin(2*pi*translationFrequency_[0]*oldTime)
18     ),
19     0,
20     translationAmplitude_[2]*
21     (
22         Foam::sin(2*pi*translationFrequency_[2]*curTime)
23         - Foam::sin(2*pi*translationFrequency_[2]*oldTime)
24     )
25 );
26
27 position.x() = translationAmplitude_.x() * Foam::sin(2 * pi *
28     translationFrequency_.x() * curTime);
29 position.y() = translationAmplitude_.y() * Foam::sin(2 * pi *
30     translationFrequency_.y() * curTime);
31 position.z() = translationAmplitude_.z() * Foam::sin(2 * pi *
32     translationFrequency_.z() * curTime);

```

```

31 if (fabs(position.z()) >= rotThreshold) {
32   Info << "ROTATING: position is: " << fabs(position.z()) << endl;
33   if (rotating_ == false) {
34     rotStartTime_ = time_.value();
35     rotating_ = true;
36     Info << "started to rotate at time: " << rotStartTime_ << endl;
37   }
38
39   if (position.z() > 0) {
40     Info << "rotating in positive direction" << endl;
41
42     alphaOld = rotationAmplitude_ * (oldTime - rotStartTime_);
43     alphaCur = rotationAmplitude_ * (curTime - rotStartTime_);
44
45   } else if (position.z() < 0) {
46     Info << "rotating in negative direction" << endl;
47     alphaOld = (rotationAmplitude_ * -1) * (oldTime - rotStartTime_);
48     alphaCur = (rotationAmplitude_ * -1) * (curTime - rotStartTime_);
49   } else if (position.z() == 0 && rotThreshold == 0) {
50     Info << "no translation" << endl;
51     alphaOld = rotationAmplitude_ * (oldTime - rotStartTime_);
52     alphaCur = rotationAmplitude_ * (curTime - rotStartTime_);
53   }
54 } else {
55
56   Info << "setting rotation to false" << endl;
57   rotating_ = false;
58   alphaOld = 0.0;
59   alphaCur = 0.0;
60 }
61
62 tensor RyOld
63 (
64   Foam::cos(alphaOld), 0, -Foam::sin(alphaOld),
65   0, 1, 0,
66   Foam::sin(alphaOld), 0, Foam::cos(alphaOld)
67 );
68
69 tensor RyCur
70 (
71   Foam::cos(alphaCur), 0, -Foam::sin(alphaCur),
72   0, 1, 0,
73   Foam::sin(alphaCur), 0, Foam::cos(alphaCur)
74 );
75

```

```
76 // rotation in XZ plane
77 vectorField rotationFieldXZ
78 (
79     (RyCur - RyOld)
80     & (statPoints_ - initialRotationOrigin_)
81 );
82 // translation in XZ plane
83 motion = translationVectorXZ + rotationFieldXZ;
```

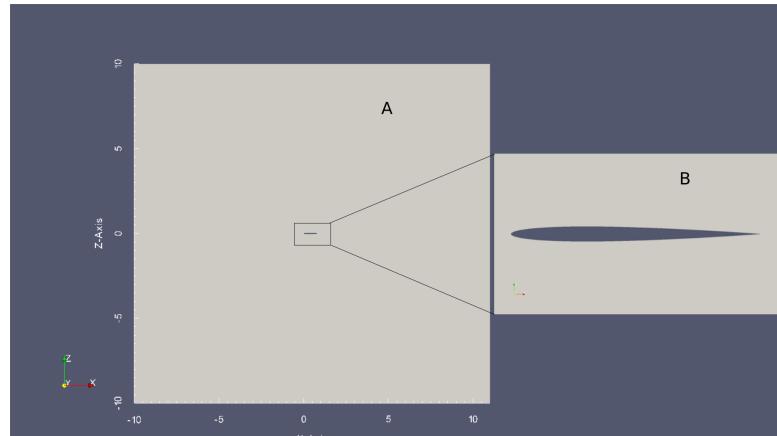


Abbildung 2.7: Die für die RBF-Rechnungen verwendete Geometrie. Teilabbildung A zeigt die Gesamtübersicht. Die Kantenlänge in X-Richtung (grüner Pfeil) beträgt 11 LE, die Kantenlänge in Z-Richtung beträgt 10 LE. Die Sehnenlänge des Flügels ist eine LE lang.

2.2.2 Numerischer Aufbau

Im Nachfolgenden wird der Aufbau der mit *RBF* durchgeführten Simulationen aufgeführt.

Geometrie

Die Geometrie wurde zunächst mit einem Matlab-Skript (siehe Listing 2) generiert. Bei der Geometrie handelte es sich um ein einfaches, spiegelsymmetrisches NACA-Flügelprofil (NACA-0006). Der Flügel hat eine Sehnenlänge von 1 m. Das umgebende Kontrollvolumen ein Quader mit den Abmessungen 11 m x 10 m x 0.01 m. Abbildung 2.7 zeigt die Geometrie und das Kontrollvolumen.

Entdimensionierung

Da in keiner Simulationen eine vorgegebene Anströmung existiert, sondern das Strömungsfeld durch die Bewegung des Flügels induziert wird, wird zur Berechnung der Reynoldszahl die mittlere Geschwindigkeit des Flügels und die Sehnenlänge genutzt. Für das biologische Vorbild Kolibri ergibt sich mit einer kinematischen Viskosität von $\nu = 1.5 \cdot 10^{-5} \text{ m}^2 \text{s}^{-1}$, einer Sehnenlänge von $l_s = 0.015 \text{ m}$ und einer Geschwindigkeit von $v = \omega \cdot r = 2\pi \cdot 50 \text{ s}^{-1} \cdot 0.04 \text{ m} = 12.56 \text{ ms}^{-1}$ eine Reynoldszahl von 12058.

Für die zweidimensionalen Simulationen wird die charakteristische Länge des Flügels auf $l_c = 1 \text{ m}$ gesetzt. Die Geschwindigkeit des Modells wurde ausgehend von Gleichung 2.6 auf $2 l_c s^{-1}$ festgelegt. Um eine Reynoldszahl von etwa 13.000 zu erreichen, ergibt sich daraus eine kinematische Viskosität von $1.5 \cdot 10^{-4} \text{ m}^2 \text{s}^{-1}$

Charakterisierung der Strömung

Bei der vorliegenden Strömung handelt es sich um eine Strömung eines newtonischen Fluids. Die Machzahl liegt mit $Ma = 0.0116$ deutlich unter der Kompressibilitätsgrenze von 0.3 für Luft. Es gilt die Kontinuumshypothese, da die Knudsenzahl mit $Kn << 0.001$ kleiner als der maximal zulässige Wert ist. Da für Umströmungen eine kritische Reynoldszahl von $Re_{krit} = 5 \cdot 10^5$ gilt, wird die hier vorliegende Strömung als laminar angesehen. Des Weiteren werden keine Wärmeübertragung, keine Mehrphasen und keine chemischen Reaktionen betrachtet. Weitere zu erwartende Phänomene sind Wirbelbildung und Scherschichten.

Modellgleichungen

Aufgrund der Inkompressibilität ergibt sich aus der Massenerhaltung eine vereinfachte Kontinuitätsgleichung:

$$\nabla \cdot \mathbf{u} = 0 \quad (2.8)$$

Für konstante Dicht und Inkompressibilität ergeben sich die inkompressiblen Navier-Stokes-Gleichungen:

$$\underbrace{\frac{\partial \mathbf{u}}{\partial t}}_{\text{instationärer Term}} + \underbrace{(\mathbf{u} \cdot \nabla) \mathbf{u}}_{\text{konvektiver Term}} = \underbrace{-\nabla \bar{p}}_{\text{Druckgradient}} + \underbrace{\nu \Delta \mathbf{u}}_{\text{diffusiver Term}} + \underbrace{\bar{f}}_{\text{Volumenkräfte}} \quad (2.9)$$

Mit $\bar{p} = \frac{p}{\rho}$ und $\bar{f} = \frac{f}{\rho}$.

Diskretisierung

OpenFOAM nutzt zur Diskretisierung grundsätzlich die Finite-Volumen Methode. Tabelle 2.5 listet die eingesetzten Diskretisierungsverfahren für die Terme der Navier-Stokes-Gleichungen.

Tabelle 2.5: Diskretisierung der einzelnen Terme der inkompressiblen Navier-Stokes-Gleichungen

Term	openFoam Bezeichnung	Diskretisierungsmethode
instationär	backward	drei-Zeitebenen Verfahren
konvektiv	Gauss upwind	Aufwind-Interpolation Gauß 1. Ordn.
Gradienten	Gauss linear	lin. Gaußsches Eliminierungsverf.
diffusiv	Gauss linear corrected	lin. Gaußsches Eliminierungsverf. mit Korr.-Term
interpolation	linear	lineares Interpolationsverfahren

Anfangs- und Randbedingungen

Die Anfangs- und Randbedingungen sind für alle Simulationen gleich gewählt. Tabelle 2.6 listet die Initial- und Randbedingungen auf.

Tabelle 2.6: Initial- und Randbedingungen der mit *RBF* durchgeführten Simulationen

Bedingung	Geschwindigkeit	Druck
Dirichlet (fester Wert)	$wing$ $u = 0$	$left, right, top, bottom$ $p = 0$
Neumann (Nullgradient)	$left, right, top, bottom$	$wing$

Da es sich um eine zweidimensionale Rechnung handelt, OpenFOAM jedoch nur dreidimensionale Gitter akzeptiert, wurde für die Ganzflächen *front* und *back* die option *empty* gesetzt. Diese Option teilt OpenFOAM mit, die Komponenten der Strömungsgrößen welche in Richtung der Normalvektoren der *empty*-Flächen liegen nicht zu berechnen.

Numerisches Gitter

Die numerischen Gitter zur Simulation wurden mit *snappyHexMesh* erstellt, einem hybriden Gittergenerator welcher zu OpenFOAM gehört. Der Gittergenerator baut auf einen blockstrukturiertem Gitter auf, in welches ein Oberflächengitter im *Stereolithographie*-Format eingebettet wird. *snappyHexMesh* generiert aus dem blockstrukturierten Gitter ein hybrides Gitter in dem die Zellen welche das Oberflächengitter schneidet, verfeinert und geglättet werden. Tabelle 2.7 führt die wichtigsten Gitterparameter auf.

Lösungsalgorithmus und Gleichungslöser

Als Lösungsalgorithmus wird *pimpleDyMFoam* eingesetzt. Dieser Lösungsalgorithmus für bewegte Gitter basiert auf dem *merged-PISO-SIMPLE* Verfahren. Es werden pro Zeitschritt drei PIMPLE-Iterationen und pro PIMPLE-Iteration zwei Druckiterationen gerechnet. Um die nicht-Orthogonalität des bewegten Gitters auszugleichen, werden pro Druckiteration 10 nicht-Orthogonalitätskorrekturen ausgeführt. Es werden keine Relaxationsfaktoren eingesetzt. Um das Geschwindigkeitsfeld zu lösen wird ein präkonditioniertes, bi-konjugiertes Gradientenverfahren eingesetzt. Der Druck wird mittels geometrisch-algebraischem Mehrgitter-Verfahren gelöst.

Auswertung

Die Auswertung erfolgte mittels Paraview (Los Almos National Laboratory, New Mexico, USA; Sandia National Laboratories Neu Mexiko und Kalifornien, USA; Kitware, New York, USA) und den mit OpenFOAM mitgelieferten Post-Processing Tools.

Tabelle 2.7: Gitterparameter des Gitters welches in Kombination mit *RBF* eingesetzt wurde

Parameter	Wert
Gitterauflösung	
Zellanzahl	90.918
Verblockung	< 1 %
Gitterqualität (MAX)	
Nicht-Orthogonalität	max 57,8465
Schiefe	1,706
Gitterstruktur	
Gittertyp	hybrides Gitter
Zelltypen	Hexaeder 89.036, Prismen 50, Polyeder 1832
snappyHexMesh	
Grundzellgröße	x: 0.1909 LE y: 0.1 LE z: 0.2 LE
Zellen zw. Verfeinerungsleveln	5
Verfeinerungslevel	6 - 8
Prismenschichten	5
Wachstumsfaktor Prismenschichten	1.5
relative Dicke Prismenschicht	100%
Verfeinerungsbox 1	min (-0.25 -0.05 -0.3) max (1.25 0.05 0.3) Verfeinerungslevel: 5
Verfeinerungsbox 2	min (-1.0 -0.05 -1.0) max (3.0 0.05 1.0) Verfeinerungslevel: 4
Verfeinerungsbox 3	min (0.0 -0.05 -4.0) max (4.0 0.05 4.0) Verfeinerungslevel: 1
zeitliches Gitter	
	instationär

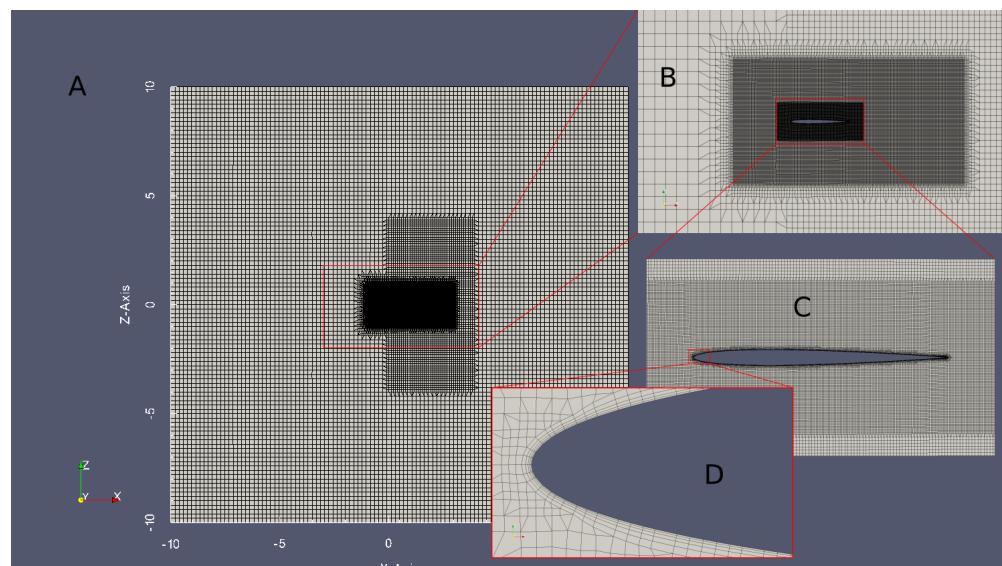


Abbildung 2.8: Schematische Übersicht über das RBF-Gitter. A: Kontrollvolumen und Verfeinerungsboxen. B: Tragflügel innerhalb der zweiten Verfeinerungsbox. C: Detaillierte Darstellung des NACA 0006 Profils. D: Detailansicht der Prismenschichten an der Oberfläche des Tragflügels

2.3 Arbitrary Mesh Interface

Arbitrary Mesh Interface (kurz: *AMI*) bezeichnet die in der Einleitung beschriebene Technik des Auftrennens des Gitters in zwei oder mehrere Sub-Gitter an deren Grenzflächen interpoliert wird. *AMI* bietet im Vergleich zu *dynamicTopoFvMesh* und *RBF* den Vorteil, dass die Gitterqualität an der Oberfläche des bewegten Objekts konstant bleibt. Durch das Auf trennen des Gitters ist es außerdem möglich unbegrenzte Rotationen zu rechnen. Um *AMI* in OpenFOAM nutzen zu können, muss das Gitter zu Beginn der Simulationen aufgetrennt werden und die aneinander liegenden Grenzflächen als *cyclicAMI*-patches definiert werden. Damit wird eine zyklische Randbedingung zwischen den Flächen hergestellt. Um die Bewegungen zu steuern, stehen verschiedene Eingabemöglichkeiten zur Verfügung, welche in der Datei *dynamicMeshDict* festgelegt werden müssen. Zur Verfügung steht dabei u.A. auch die Möglichkeit eine Bewegung mit sechs Freiheitsgraden festzulegen. Dabei werden zu festen Zeitpunkten zu erreichende Winkel und Auslenkung in allen drei Raumrichtungen festgelegt. Zwischen diesen Werten wird über die Zeit auf jeden Zeitschritt linear interpoliert. Dies bietet Bedienungskomfort, allerdings auch die Gefahr von Interpolationsfehlern. Listing 2.6 listet die zur Durchführung benötigten Einstellungen in der Datei *dynamicMeshDict*.

Listing 2.6: *dynamicMeshDict* für *AMI*-Simulationen

```

1 // Gittertyp festlegen
2 dynamicFvMesh solidBodyMotionFvMesh;
3 // den Lösungsalgorithmus zur Bewegung des Gitters festlegen
4 motionSolverLibs ( "libfvMotionSolvers.so" );
5 // Im nachfolgenden Subdict werden die Bewegungsparameter definiert
6 solidBodyMotionFvMeshCoeffs
7 {
8     // hier wird das zu bewegende Teilgitter definiert.
9     cellZone      sphere;
10    // Die Bewegungsart festlegen
11    solidBodyMotionFunction tabulated6DoFMotion;
12    tabulated6DoFMotionCoeffs
13    {
14        CofG          ( 0 0 0 ); // Nullpunkt der Bewegung
15        // die Datei im nachfolgenden Pfad enthält die zu bestimmten
16        // Zeitpunkten zu erreichenden Auslenkungen und Rotationen
17        timeDataFileName "\$FOAM_CASE/constant/6DoF.dat";
18    }
19 }
```

Die genauen Winkelwerte über die Zeit sind in Tabelle 2.10 gelistet.

Obwohl die Technik kaum translative Bewegungen zulässt, kann mit ihr die (vereinfachte) Flügelbewegung eines Kolibris abgebildet werden. Tobalske u. a. 2007 stellt fest, dass die Flügellänge eines Kolibris um nur etwa 5 - 10 % schwankt. Damit haben diese Organismen

äusserst starre Flügel. Die Trajektorie der Flügelspitze kann durch die Verläufe der drei Raumwinkel zwischen Ursprung und Flügelspitze, sowie eines dritten, beliebigen Punkts im Flügel beschrieben werden.

2.3.1 Numerischer Aufbau

Geometrie

Für die Simulationen wird aus Kruyt u. a. (2014) die Flügelform von *Colibri coruscans* ausgewählt. Die Vorlage wurde in Rhino 5 (McNeal Europe, Barcelona, Spanien) geladen und mit Hilfe einer Spline-Interpolation erfasst. Da keinerlei Daten zu den Profilformen vorlagen, wird ein NACA 0006 Profile als Profil angenommen. Abbildung 2.9 zeigt in Teilabbildung A das biologische Vorbild (veränderte Darstellung nach Kruyt u. a. (2014)). Teilabbildung B zeigt die aus der Vorlage erstellte CAD-Geometrie.

Entdimensionierung

Das CAD-Modell (Abbildung 2.9, Teilabbildung B) hat eine Spanne von 0.4 m und eine max. Sehnenlänge von 0.135 m. Als charakteristische Länge l_c wurde die maximale Sehnenlänge von 0.135 m ausgewählt. Aus Tabelle 2.10 ergibt sich eine mittlere Winkelgeschwindigkeit von $\frac{7}{9} \cdot \pi \text{ s}^{-1}$. Um die der Literatur (Tobalske u. a. 2007) entnommene Reynoldszahl von etwa 13.000 zu erreichen, wurde eine kinematische Viskosität von $1.5 \cdot 10^{-4} \text{ m}^2\text{s}^{-1}$ eingestellt.

Da die Strömungsgeschwindigkeit nur durch die Flügelbewegung induziert wird, wird zur Entdimensionierung der Geschwindigkeit die mittlere Winkelgeschwindigkeit genutzt.

Charakterisierung der Strömung

Bei der Strömung handelt es sich um eine inkompressible, dreidimensionale Strömung. Das Fluid ist newtonsch. Es findet keine Wärmeübertragung und keine chemische Reaktion statt. Es wird erwartet, dass sich durch die komplexe Bewegung ein starkes Wirbelfeld ausbildet, induziert durch das ablösen der Scherströmungen an der Flügeloberfläche.

Modellgleichungen

Es gelten Analog zu den RBF-Rechnungen die gleichen Gleichungen (siehe Gleichung 2.8 und Gleichung 2.9).

Diskretisierung

Die zur Diskretisierung genutzten Verfahren werden in Tabelle 2.8 gelistet.

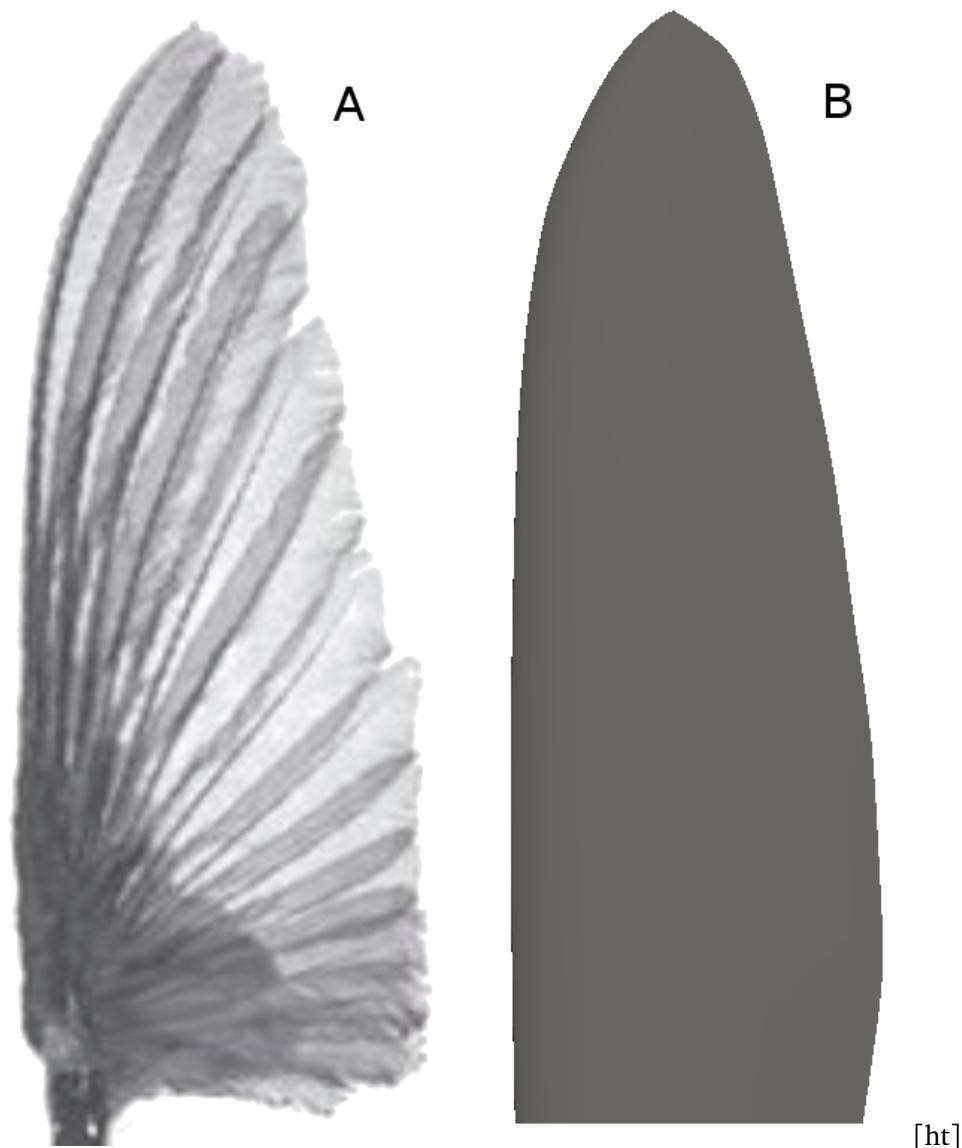


Abbildung 2.9: Vorlage und Geometrie welche zur dreidimensionalen Simulation mit Hilfe von AMI durchgeführt wurden. Teilabbildung A zeigt die Vorlage (verändert nach Kruyt u. a. 2014). Bei dem präparierten Flügel handelt es sich um ein männliches Exemplar von *Colibri coruscans*. Teilabbildung B zeigt die mit Hilfe von Spline-Interpolation erzeugte CAD-Geometrie.

Tabelle 2.8: Diskretisierung der einzelnen Terme der inkompressiblen Navier-Stokes-Gleichungen

Term	openFoam Bezeichnung	Diskretisierungsmethode
instationär	backward	drei-Zeitebenen Verfahren
konvektiv	Gauss linearUpwindV grad(U)	hybride linear-Aufwind-Interpolation Gauß 2. Ordn.
Gradienten	Gauss linear	lin. Gaußsches Eliminierungsverf.
diffusiv	Gauss linear corrected	lin. Gaußsches Eliminierungsverf. mit Korrig.-Term
interpolation	linear	lineares Interpolationsverfahren

Anfangs und Randbedingungen

Tabelle 2.9 listet die verwendeten Initial- und Randbedingungen.

Tabelle 2.9: Initial- und Randbedingungen für die mit *AMI* durchgeführten Simulationen

Bedingung	Geschwindigkeit	Druck
Dirichlet (fester Wert)	$wing$ $u = 0$	$walls$ $p = 0$
Neumann (Nullgradient)	$walls$	$wing$

Für die dreidimensionale Simulation wurden zusätzlich die Winkelverläufe über die Zeit benötigt. Tabelle 2.10 listet diese.

Tabelle 2.10: Initial- und Randbedingungen für die Gitterbewegung. Es findet keinerlei Translation statt. Die Bewegung wird aus den nachfolgend gelisteten Werten linear interpoliert.

Zeit	Translation	Rotation
0	(0 0 0)	(0 0 0)
0.25	(0 0 0)	(0 -45 0)
0.5	(0 0 0)	(0 -70 0)
0.55	(0 0 0)	(0 -70 -90)
0.8	(0 0 0)	(0 -45 -90)
1.05	(0 0 0)	(0 0 -90)
1.3	(0 0 0)	(0 45 -90)
1.55	(0 0 0)	(0 70 -90)
1.6	(0 0 0)	(0 70 0)
1.85	(0 0 0)	(0 45 0)
2.1	(0 0 0)	(0 0 0)
2.35	(0 0 0)	(0 -45 0)
2.6	(0 0 0)	(0 -70 0)
2.65	(0 0 0)	(0 -70 -90)
2.9	(0 0 0)	(0 -45 -90)
3.15	(0 0 0)	(0 0 -90)
3.4	(0 0 0)	(0 45 -90)
3.65	(0 0 0)	(0 90 -90)

Numerisches Gitter

Das räumliche Gitter wurde ebenfalls mit *snappyHexMesh* generiert. Das zugrunde liegende blockstrukturierte Gitter hat Kantenlängen von 2,75 LE x 2,75 LE x 3,375 LE. Um die Rotation des Flügels um die entsprechenden Raumwinkel abzubilden, wurde der Flügel in eine Kugel mit dem Radius 1,25 LE eingebettet. Der Ursprung der Kugel lag im Koordinatenursprung. Tabelle 2.11 listet die wichtigsten Gitterparameter. In Abbildung 2.10 zeigt das eingesetzte räumliche Gitter.

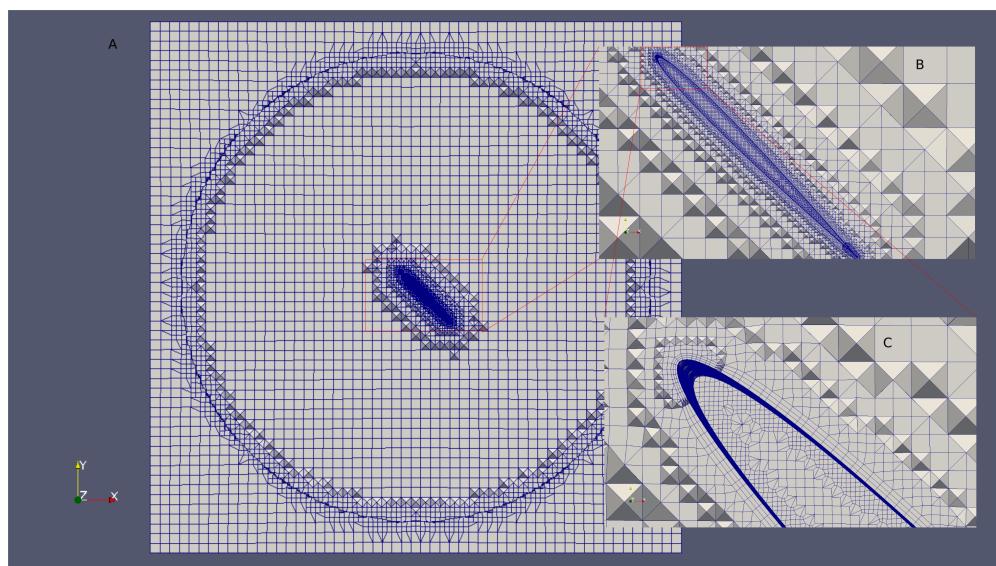


Abbildung 2.10: BLABLABLA

Tabelle 2.11: Gitterparameter für die AMI-Rechnungen

Parameter	Wert
Gitterauflösung	
Zellanzahl	1.936.594
Verblockung	zwischen 0.1 und 10 %
Gitterqualität (MAX)	
Nicht-Orthogonalität	max 64.92 durchschnittlich 10,83
Schiefe	max 7,842 (zwei Zellen)
Gitterstruktur	
Gittertyp	hybrides Gitter
Zelltypen	Hexaeder 1.489.715, Prismen 184789, Polyeder 262028, flache Tetraeder 62
snappyHexMesh	
Grundzellgröße	x: 0.05 LE y: 0.05 LE z: 0.0499 LE
Zellen zw. Verfeinerungsleveln	2
Verfeinerungslevel	5 - 6
Prismenschichten	3
Wachstumsfaktor Prismenschichten	1.1
relative Dicke Prismenschicht	50%
zeitliches Gitter	instationär

Lösungsalgorithmus

Analog zu den *RBF*-Rechnungen wurde ebenfalls *pimpleDyMFoam* eingesetzt.

Auswertung der Ergebnisse

Zur Auswertung der Ergebnisse kam Paraview (paraview.org, Los Almos National Laboratory, New Mexico, USA; Sandia National Laboratories Neu Mexiko und Kalifornien, USA; Kitware, New York, USA) zum Einsatz. Des Weiteren kamen zu OpenFOAM gehörende Post-Processing Werkzeuge zum Einsatz, u.A. *Q*, *vorticity*, *wallShearStress*, *wallBoundedStreamlines* und *execFlowFunctionObjects*. Listing 2.7 listet die zur Erstellung von wandnahen Stromlinien benötigte Inputdatei.

Listing 2.7: Dictionary zum erstellen von wallBounded Streamlines

```

1 near // Das Strömungsfeld muss auf die Oberfläche des Patches interpoliert werden
2 {
3     functionObjectLibs ("libfieldFunctionObjects.so");
4     type nearWallFields;
5     outputControl    outputTime;
6     fields
7     (
8         (U UNear)
9     );
10    patches (wing);
11    distance 0.01;
12 }
13 wallBoundedStreamLines
14 {
15     functionObjectLibs ("libfieldFunctionObjects.so");
16     type          wallBoundedStreamLine;
17     outputControl    outputTime;
18     setFormat      vtk; //gnuplot; //xmgr; //raw; //jplot;
19     UName UNear;
20     trackForward   true;
21     interpolationScheme cellPoint;
22     fields (p U UNear);
23     lifeTime       500;
24     cloudName      wallBoundedParticleTracks;
25     seedSampleSet  patchSeed; //cloud;//triSurfaceMeshPointSet;
26     uniformCoeffs
27     {
28         type      uniform;
29         axis      x; //distance;
30         start    (0.0035 0.0999 0.0001);
31         end      (0.0035 0.0999 0.0099);

```

```
32     nPoints      20;
33 }
34 cloudCoeffs
{
35     type        cloud;
36     axis        x; //distance;
37     points     ((0.351516548679288 -0.0116085375585099 1.24));
38 }
39 patchSeedCoeffs
{
40     type        patchSeed;
41     patches    (wing);
42     axis        x; //distance;
43     maxPoints  10000;
44 }
45 }
46 }
47 }
```

3 Ergebnisse

3.1 dynamicTopoFvMesh

3.1.1 Testszenario A

Im ersten Testszenario wurde das Verhalten von Translation und Rotation getrennt untersucht. Abbildung 3.1 (a - h) zeigt einen Querschnitt durch den Mittelpunkt, normal zur X-Achse. Um den Umfang zu begrenzen wurden nur acht Zeitschritte betrachtet.

Rotation um die X-Achse

In der ersten Abbildung ist deutlich zu erkennen, dass das Gitter im Vergleich zu Abbildung 2.4 bereits stark manipuliert wurde. An der Oberfläche sind die Zellen feiner, während die weiter entfernten Zellen ein größeres Zellvolumen haben. Mit dem Fortschreiten der Rotation werden die Zellen immer stärker gestreckt, bzw. gestaucht. Übersteigt die Deformation das festgelegte Größenverhältnis der benachbarten Zellen, so werden diese entweder durch den Glättungsmechanismus angepasst, zerteilt oder kollabiert. Es wurde eine vollständige Umdrehung abgebildet. Jede Abbildung entspricht einem zusätzlichen Winkel von 45° . Um sicher zu stellen, dass es nicht zu einem Anwachsen der Fehler kommt, wurden insgesamt 6 Rotationen gerechnet.

Die Gitterqualität bleibt dabei relativ konstant, die maximale nicht-Orthogonalität schwankt mit einer Abweichung von etwa 10% um einen Wert von 65° .

Translation in Richtung der X und Z-Achse

Ausgehend von dem gleichen Gitter wie im Fall der Rotation wurde die Platte zunächst in die positive X-Richtung (Abbildung 3.2, (a - d)) bewegt. Als die Wand nahezu erreicht war, wurde die Bewegungsrichtung in die positive Z-Achsenrichtung geändert (Abbildung 3.2 (e - f)). Kurz vor dem Erreichen der nächsten Wand wurde die Bewegungsrichtung wiederum, in die negative X-Achsenrichtung (Abbildung 3.2 (g)) geändert. Beim Erreichen einer etwa mittigen Position wurde noch einmal die Bewegungsrichtung in Richtung der negativen X-Achsenrichtung geändert um die Ausgangsposition zu erreichen (Abbildung 3.2 (h)).

Als die ungefähre Ausgangsposition erreicht war, wurde die Simulation angehalten. Auch hier wurde, um zu überprüfen, ob es zu einem Anwachsen der Fehler kommt in einem weitere Szenario weitaus mehr Bewegungsänderungen eingefügt.

3.1.2 Testszenario B

Abbildung 3.3 zeigt die Translation der Kugel durch das Gitter in acht ausgewählten Zeitschritten. Auch hier ist deutlich zu sehen, dass das ursprüngliche Gitter bereits zwischen den ersten beiden Zeitschritten stark verändert wurde, obwohl noch keine Beschleunigung der Kugel erfolgte. Dies ist den Einstellungen der Parameter geschuldet, da *dynamicTopoFvMesh* zunächst das gegebene Gitter den eingestellten Parameter anpasst. Mit jedem weiteren Zeitschritt werden die Zellen vor der Kugel weiter verfeinert, während die Zellen hinter der Kugel mit steigendem Abstand vergrößert werden. Deutlicher noch als in Testszenario A wird das dynamische Regenerieren der Gitterzellen deutlich. Nachdem die Kugel um etwa 1.5 LE bewegt wurde, musste die Simulation jedoch aufgrund von zu schlechter Gitterqualität abgebrochen werden.

3 Ergebnisse

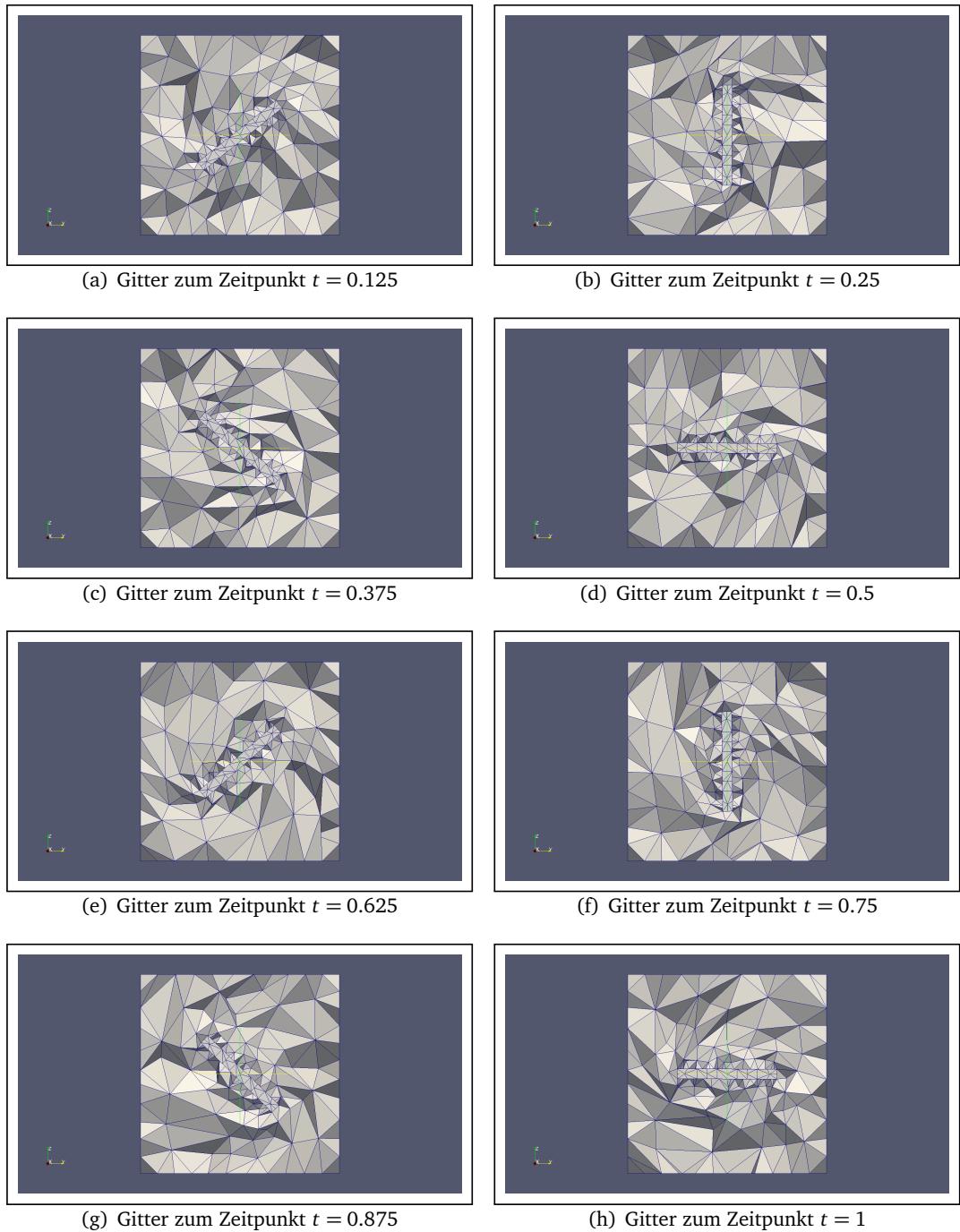


Abbildung 3.1: Rotation der Platte im Kontrollvolumen. Deutlich ist zu erkennen, dass die Zellen an der Oberfläche der Platte über alle Zeitschritte etwa die Gleiche Form und Größe behalten. Die umgebenden Zellen werden teils stark gestaucht, bzw. gestreckt.

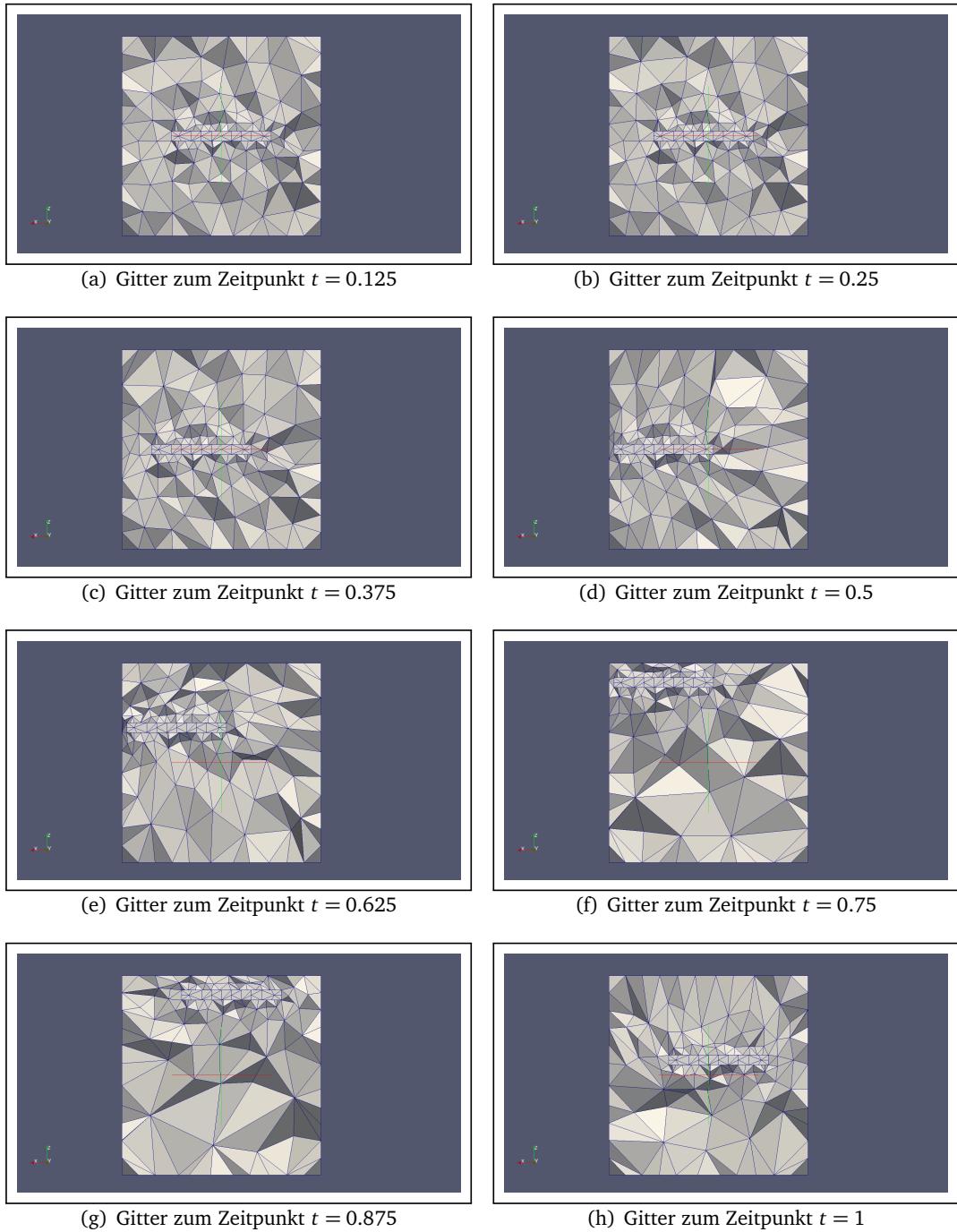


Abbildung 3.2: Translation der Platte im Kontrollvolumen. Deutlich zu erkennen ist, dass die Zellen an der Vorderkante der Platte zunächst gestaucht werden um schließlich zu kollabieren. Durch den fortschreitenden Kollaps der Zellen nimmt die Anzahl der Zellen kontinuierlich ab (a - c), sodass schließlich nur eine Zellschicht verbleibt (d). Diese eine Zellschicht verbleibt an der linken Kante der Platte bis die Anzahl der Zellen schließlich durch dehnen und aufteilen wieder zunimmt (e - h). Für die Zellen hinter der Platte gilt leider, dass das *bisectionRatio* sehr groß ist und die Zellen daher kaum zerteilt werden.

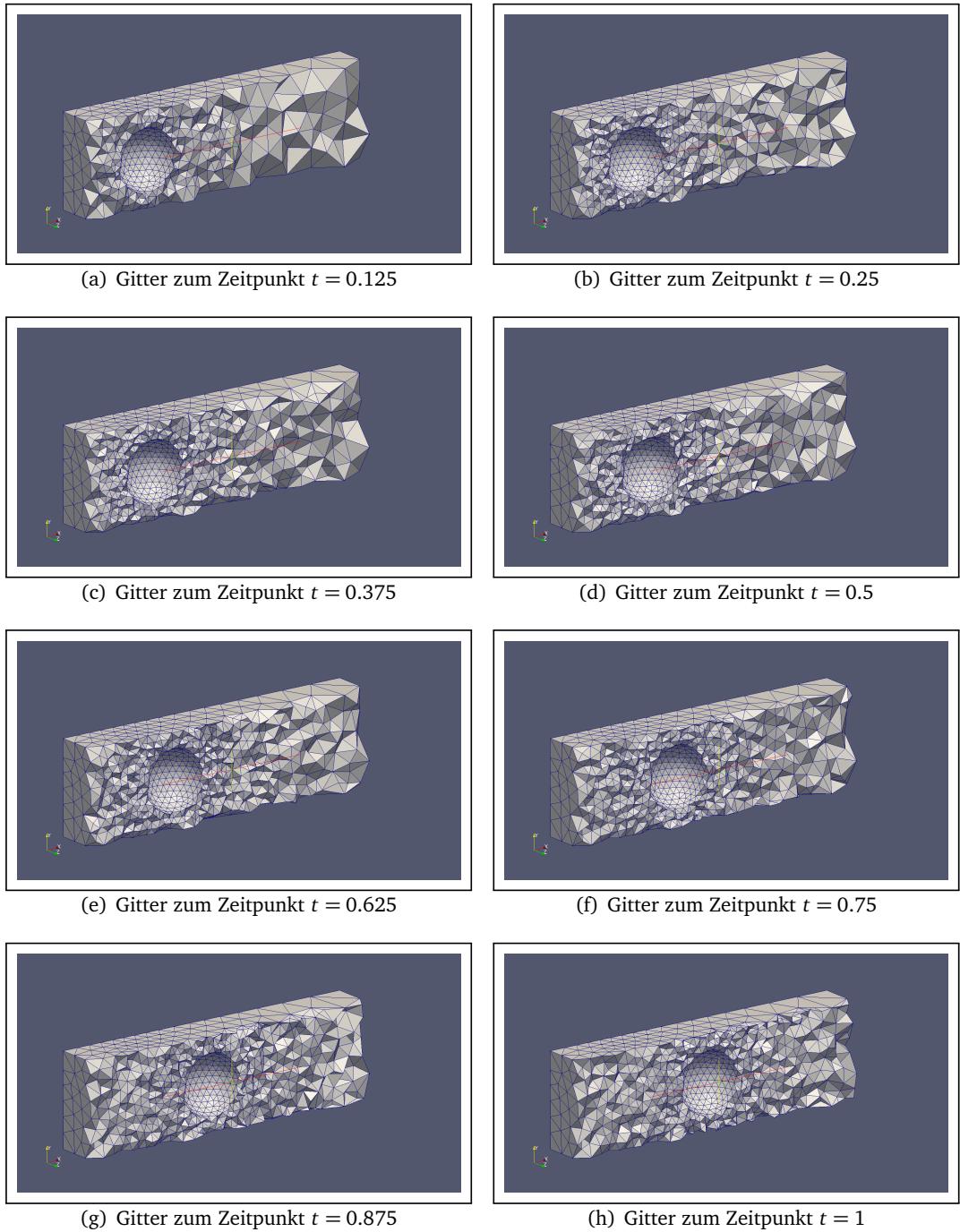


Abbildung 3.3: Translation einer Kugel im Kontrollvolumen. Deutlich erkennbar verfeinert *dynamicTopoFvMesh* zunächst die die Kugel umgebenden Zellen (Vergleich (a) zu (b)). Danach erfolgt die Beschleunigung der Kugel in positive X-Richtung. Dabei bleiben die Zellgrößen an der Oberfläche der Kugel erhalten, im Nachlauf kommt es mit steigendem Abstand zur Wand zu einer Vergrößerung der Zellen.

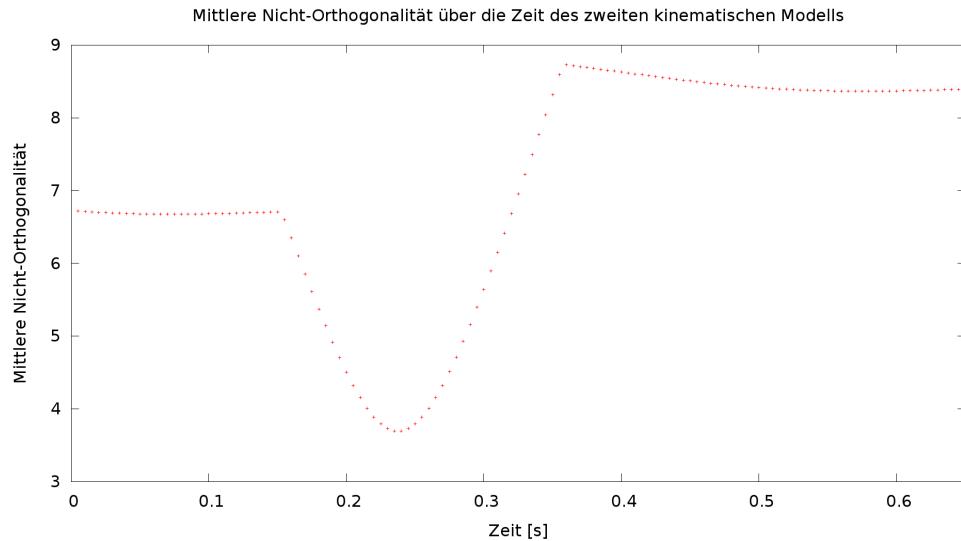


Abbildung 3.4: Mittlere Nicht-Orthogonalität des Gitters über die Zeit für das zweite kinematische Modell

3.2 Radiale Basisfunktionen

Es wurden für alle Simulationen Residuen von mindestens 10^{-5} für den Druck und 10^{-6} für das Geschwindigkeitsfeld erreicht. Zwar traten deutliche Spitzen in den Druckresiduen auf, welche jeweils mit Beginn und Ende der Rotation korrespondierten, diese erreichte jedoch innerhalb kürzester Zeit ihre alten Werte.

3.2.1 Gitterqualität & Gitterdeformation

Abbildung 3.4 zeigt die mittlere Nicht-Orthogonalität des Gitters über die Simulationszeit. Da das Gitter zum Zeitpunkt $t = 0\text{s}$ bereits gedreht ist, ist die mittlere Nicht-Orthogonalität zu Beginn bereits erhöht (6.72). Mit dem Beginn der Rotation, sinkt die Orthogonalität auf $3,694^\circ$ zum Zeitpunkt $t = 0.24\text{s}$ ab und erhöht sich danach bis auf einen Wert von $8,726^\circ$ zum Zeitpunkt $t = 0.36\text{s}$. Dieser Zeitpunkt entspricht auch dem Ende der Rotation. Bis zum Ende der Simulation sinkt die Nicht-Orthogonalität langsam auf etwa $8,3^\circ$ ab.

Abbildung 3.5 zeigt die Gitterdeformation, welche durch das zweite kinematische Modell berechnet wird.

In Abbildung 3.5(a) und Abbildung 3.5(b) erfolgt zunächst eine rein translativen Bewegung. In Abbildung 3.5(c) beginnt zusätzlich zur Translation auch die Rotation, welche bis zum Zeitpunkt $t = 0.375$ (Abbildung 3.5(d)) fortgeführt wird. Abbildung 3.5(e) und 3.5(f) zeigen den Rest der wieder rein translativen Bewegung. Es wird insgesamt eine Rotationsamplitude von 90° und eine Translationsamplitude von 1 LE erreicht.

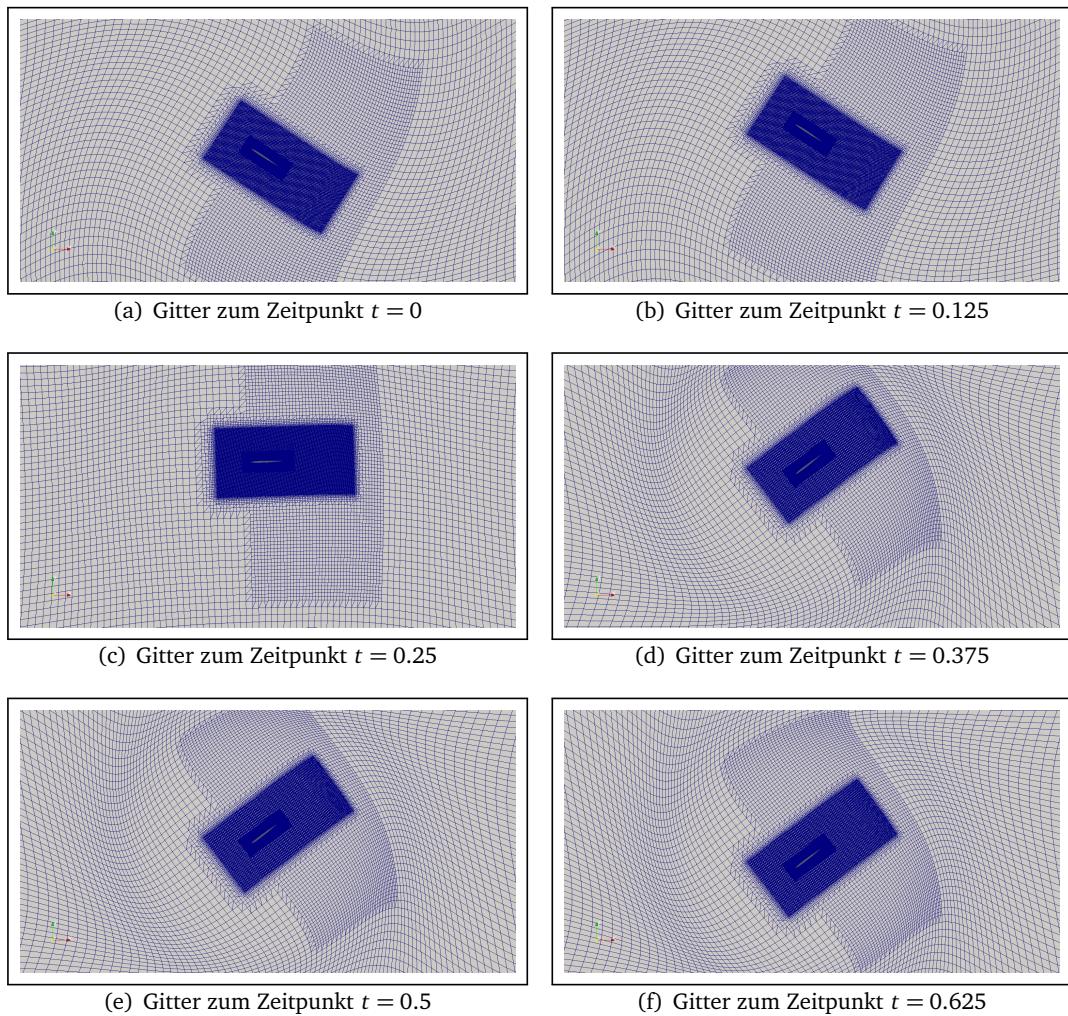


Abbildung 3.5: Unterschrift

3.2.2 Wirbelstärke

Abbildung 3.6 ist die Wirbelstärke abgebildet. Stark hellblaue Bereiche entsprechen dabei einer Wirbelstärke von -250s^{-1} und einem gegen den Uhrzeigersinn drehenden Strömungsfeld. Hellrote Bereiche entsprechen einem im Uhrzeigersinn rotierenden Strömungsfeld und einer maximalen Wirbelstärke von 250s^{-1} .

Die identifizierten Strömungstrukturen tragen in den Abbildungen folgende Abkürzungen und sind in ihrer Schreibweise an Birch und Dickinson (2003) angelehnt.

LEV: *Leading Edge Vortex*. Ein großer, instationärer Wirbel, welcher sich hinter der angeströmten Kante ausbildet und dieser Kante folgt.

TSV: *Tranlation Starting Vortex.* Ein Wirbel welcher durch den Beginn der Translation ausgebildet wird und sich schnell von der Hinterkante ablöst.

RSV: *Rotation Starting Vortex.* Ein Wirbel welcher durch den Beginn der Rotation abgelöst wird und ebenfalls schnell vom Flügel ablöst.

SBL: *Shedding Boundary Layer.* Durch die Rotation wird die Grenzschicht und die damit verbundene Scherschicht über die Vorderkante des Flügels weggedrückt. Dieses Phänomen wird in der Literatur so nicht beschrieben, der Begriff wurde vom Autor selbst eingeführt.

USL: *Underwing Shear Layer.* Durch die Bewegung durch das ruhende, umgebende Fluid bildet sich eine große Scherschicht an der angeströmten Unterseite des Flügels aus.

Im folgenden werden die identifizierbaren Wirbel mit ihrem Typ (bspw: *Leading Edge Vortex, LEV*) und einer Nummer, welche mit dem Auftreten korrespondiert bezeichnet (bspw: *LEV1* für den ersten identifizierten *Leading Edge Vortex*).

In Abbildung 3.6(a) ist die Wirbelstärke zum Zeitpunkt $t = 0.125$ dargestellt. Deutlich zu erkennen ist der sich ausbildende *Leading Edge Vortex, LEV1*, der in der Ablösung befindliche *Translation Starting Vortex, TSV1* und die als *Underwing Shear Layer, USL1* bezeichnete Scherschicht, wobei sich *Underwing* auf die Bewegungsrichtung und damit auf die angeströmte Flügelfläche bezieht.

In Abbildung 3.6(b) ($t = 0.25$) ist der *TSV1* bereits vollständig abgelöst und der *Rotational Starting Vortex, RSV1* befindet sich bereits im Ablösestadium. Der *LEV1* ist näher an die Flügeloberfläche gewandert, was der verlangsamten Translation und der eigenen Geschwindigkeit des *LEV1* geschuldet ist. Ein Weiteres Phänomen in Abbildung 3.6(b) ist die vom Autor als *Shedded Boundary Layer, SBL1* bezeichnete Struktur. Es scheint sich dabei um eine Scherschicht zu handeln, welche sowohl durch die Umfangsgeschwindigkeit des *LEV1* als auch die gerade begonnene Rotation des Flügels über die Vorderkante abgeschieden wird.

In Abbildung 3.6(c) ($t = 0.375$) ist die Rotation bereits abgeschlossen und der Flügel befindet sich bereits in der Abwärtsbewegung. Deutlich zu erkennen ist ein zweiter *TSV2*, welcher sich bereits in der Ablösung befindet. Der in der ersten Hälfte des Schlagzyklus generierte *LEV1* (Abbildungen 3.6(a) - 3.6(b)) trifft auf den mittlerweile um 90° gedrehten Flügel und wird dadurch sowohl entlang des Flügels als auch über die Vorderkante deformiert. Die Reste des *SBL1* sind als dünner, roter Bereich mit positiver Wirbelstärke erkennbar. Ebenfalls deutlich zu erkennen ist das erneute Ausbilden eines *LEV2*. Der erste *TSV1* und *RSV1* befinden hinter der Hinterkante des Flügels und wandern mit relativ konstanter Geschwindigkeit durch den Nachlauf.

In Abbildung ?? ($t = 0.5$) ist der *LEV2* bereits voll ausgebildet, allerdings scheint der Wirbel durch die Reste des ersten *LEV1* in die Länge gezogen zu werden. Dies mag der entgegengesetzten Rotationsgeschwindigkeit des ersten *LEV1* geschuldet sein. *RSV1* und

3 Ergebnisse

TSV1 zeigen deutliche Anzeichen von Dissipation, wobei *TSV1* bereits deutlich degenerierter scheint. *TSV2* hat sich zu diesem Zeitpunkt bereits vom Flügel abgelöst und ist nur noch durch eine dünne, hohe Wirbelstärke aufweisende Schicht mit dem Flügel verbunden. An der angeströmten Seite des Flügels ist deutlich eine weitere Scherschicht zu erkennen, welche sich jedoch auch mit den Resten des *LEV1* vermischen dürfte. *TSV1* ist zu diesem Zeitpunkt fast vollständig degeneriert. *RSV1* wandert weiterhin, mit deutlich verminderter Wirbelstärke durch den Nachlauf.

Abbildung 3.6(e) ($t = 0.625$) zeigt einen vollausgebildeten *LEV2*. *TSV2* hat sich vollständig vom Flügel abgelöst. *TSV1* dagegen ist vollständig degeneriert und auf der Darstellung nicht mehr zu identifizieren. *RSV1* ist immer noch zu erkennen, jedoch mit deutlich verminderter Wirbelstärke.

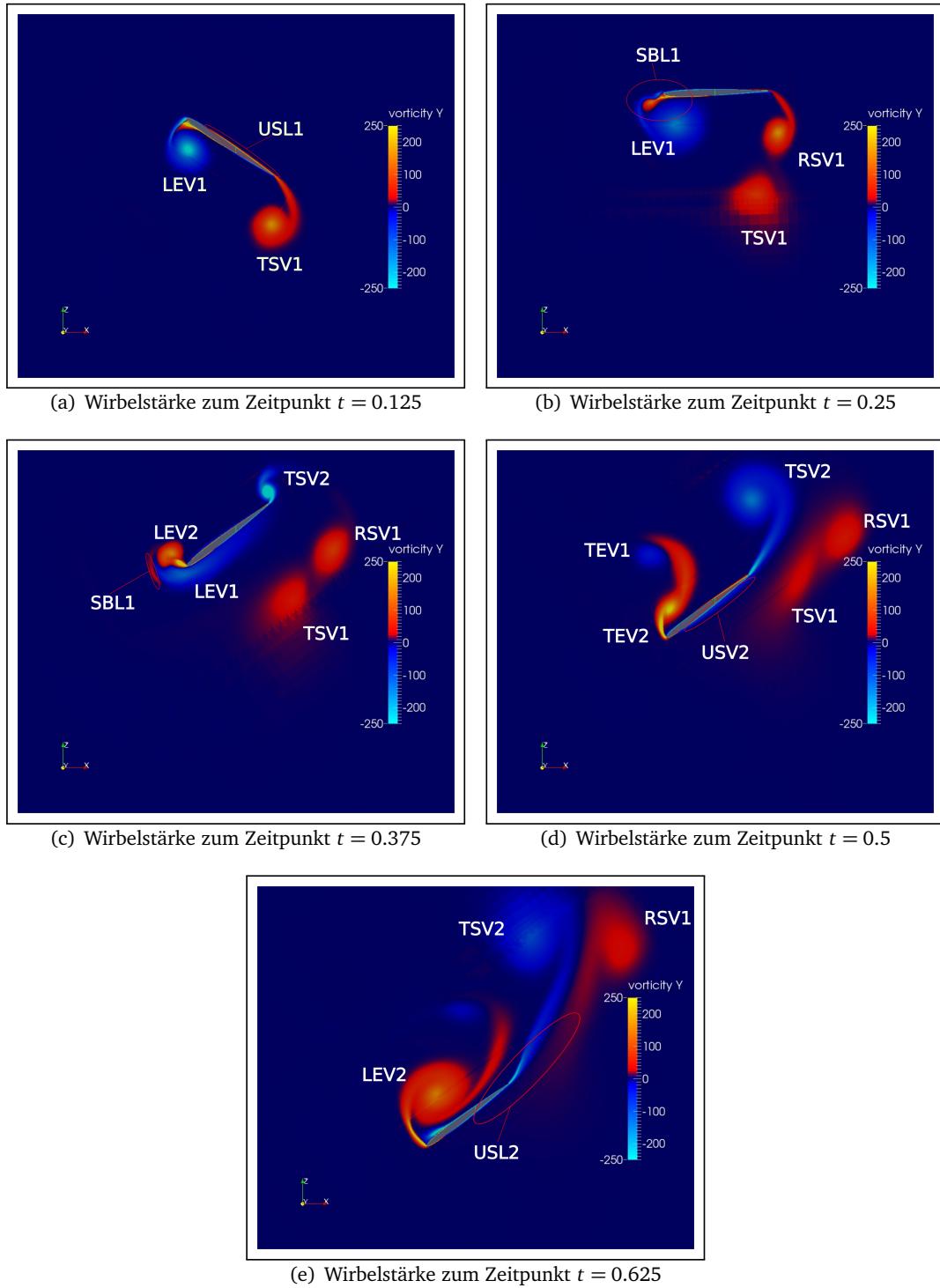


Abbildung 3.6: Unterschrift

3.3 Arbitrary Mesh Interface

Für die durchgeführte Rechnung wurden Residuen von min. 10^{-5} für das Druckfeld und 10^{-6} für das Geschwindigkeitsfeld erreicht. Auch hier traten Spitzen in den Residuen des Druckfelds auf, welche jeweils mit Beginn und Ende der Rotationsbewegung zusammen fielen.

3.3.1 Gitterbewegung

Abbildung 3.7 zeigt die Simulierte Bewegung des Flügels im Kontrollvolumen.

Abbildung 3.7(a) zeigt das Gitter zum Zeitpunkt $t = 0s$. In Abbildung 3.7(b) ($t = 0.5s$) hat der Flügel bereits die maximale Amplitude von 70° erreicht. In der darauf folgenden Abbildung (Abbildung 3.7(c), $t = 1.05s$) ist der Flügel wieder in Ruhelage, allerdings ist die angeströmte Kante (*Leading Edge, LE*) um 90° gedreht. Abbildung 3.7(d) ($t = 1.55s$) zeigt den Flügel wieder bei maximaler Amplitude. Zum Zeitschritt $t = 1.6s$ (Abbildung 3.7(e)) ist die *LE* wieder um 90° gedreht und somit in der ursprünglichen Ausrichtung. In Abbildung 3.7(f) ($t = 2.1s$) ist die Ausgangslage wieder erreicht.

3 Ergebnisse

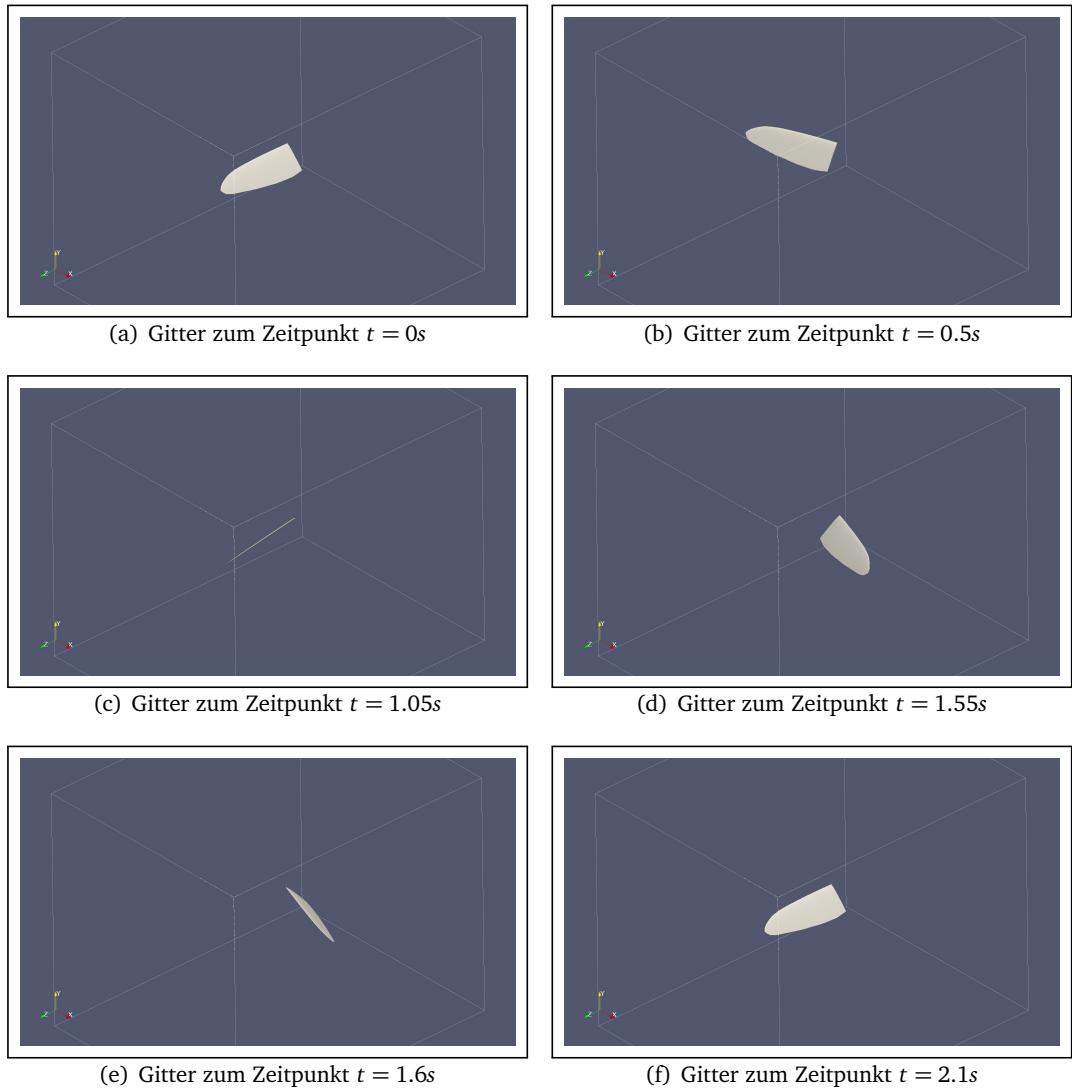


Abbildung 3.7: Die Bewegung des Flügels in der Rechendomäne. Abbildung 3.7(a) bis Abbildung 3.7(b) zeigt den *upstroke*, Abbildung 3.7(c) bis Abbildung 3.7(d) den darauf folgenden *downstroke*. Abbildung 3.7(e) zeigt den Flügel direkt nach der als *Supination* bezeichneten Rotation um die Flügelachse. In Abbildung 3.7(f) ist schließlich der Ursprungszustand wieder erreicht.

3.3.2 Strömungsfeld & Strukturen

Abbildung 3.8 zeigt die Visualisierung des Strömungsfelds mit Hilfe des Q-Kriteriums. Gezeigt werden Flächen gleichen Wertes für Q mit einem Wert von 2,5. Die identifizierbaren Wirbelstrukturen sind nach dem Schema von Birch und Dickinson (2003) beschrieben.

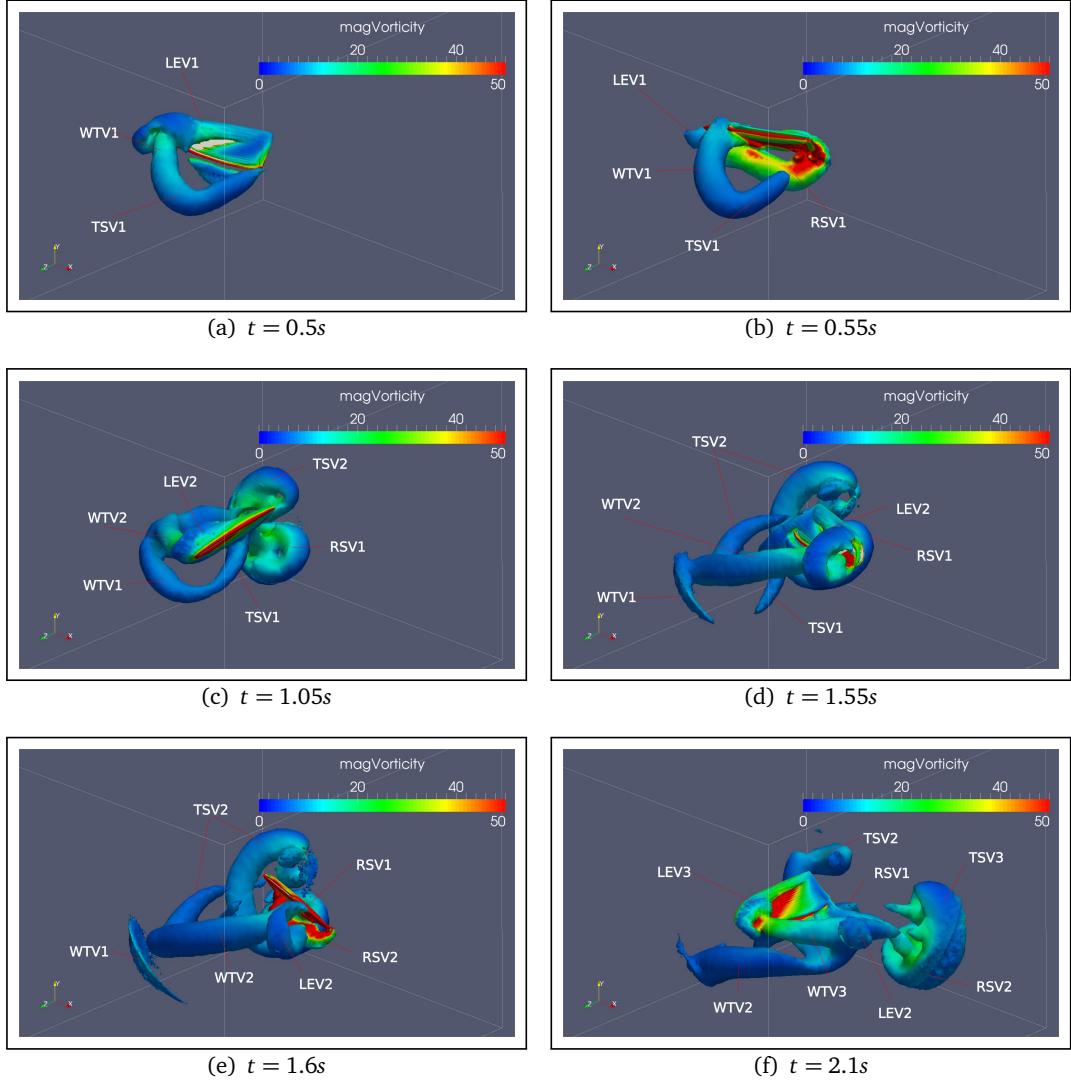


Abbildung 3.8: Q-Kriterium + Wirbelstärke AMI Rechnung

In Abbildung 3.8(a) ist das bereits voll ausgebildet ($t = 0.5s$). Die Bewegung ist exakt einen Zeitschritt vor Beginn der Rotation. Eingefärbt ist das Q-Kriterium mit dem Betrag der Wirbelstärke, wobei dunkelblaue Bereiche einer Wirbelstärke von fast 0s^{-1} und rote Bereiche eine Wirbelstärke von maximal 50 s^{-1} visualisieren. Deutlich zu erkennen ist der *Leading*

3 Ergebnisse

Edge Vortex (LEV1 - LEV3) und der *Wing Tip Vortex (WTV1 - WTV3)*. Der sog. *Translational Starting Vortex (TSV1 - TSV3)* ist in dieser Abbildung mit dem *WTV1* verschmolzen.

Zum Zeitpunkt $t = 0.55\text{s}$ (Abbildung 3.8(b)) ist die Rotation bereits beendet. Deutlich ist der *Rotational Starting Vortex (RSV1 - RSV2)* unterhalb der Unterkante des Flügels zu erkennen. Der *LEV1* ist bereits stark degeneriert. Der *WTV1* ist vollständig vom Flügel abgelöst. Auch der *TSV1* ist nicht mehr mit dem Flügel verbunden.

Abbildung 3.8(c) zeigt den Flügel im Nulldurchgang der Translation. Der Flügel ist 90° umd die Z-Achse rotiert. Im Nachlauf des Flügels sind die Reste des *TSV2* und des *RSV1* zu erkennen. Der Großteil dieser beiden Wirbel bewegt sich in negativer Z-Achsenrichtung fort. Der *WTV1* ist noch vorhanden und bewegt sich mit dem in negative Y-Achsenrichtung beschleunigten Fluid mit. Es bildet sich bereits ein zweiter *Trailing Edge Vortex (TEV2)* aus. Auch der zweite *WTV2* ist im Aufbau. Die Reste des *TSV1* sind noch zu erkennen, dieser Wirbel wird aber langsam dissipiert.

Zum Zeitpunkt $t = 1.55\text{s}$ ist die wieder die maximale Translationsamplitude erreicht. Der Flügel hat zu diesem Zeitpunkt seinen eigenen Nachlauf durchquert. Die Reste des *TSV2* sind noch gut zu erkennen, wobei dieser Wirbel sich dieser Wirbel in der Auflösung befindet. Auch *WTV1* und *WTV2* sind deutlich zu erkennen. *WTV1* ist jedoch bereits stark deformiert und geschrumpft. Der Wirbel ist stark zusammengedrückt und weist eine kantige Struktur, welche sich längs durch den Wirbel zieht auf. *TSV1* ist fast vollständig dissipiert. Aufgrund der Perspektive wird der Eindruck erweckt, dass der *RSV1* in Verbindung mit dem Flügel getreten ist. Dies ist nicht der Fall, der Wirbel bewegt sich langsam, zusammen mit *TSV2* etwa an der Position der maximalen positiven Schlagamplitude in negativer Z-Achsenrichtung fort. Der *LEV2* ist ebenfalls deutlich zu erkennen. An der Spitze des Flügels scheinen der *WTV2* und der *LEV2* miteinander zu interagieren.

Zum Zeitpunkt $t = 1.6\text{s}$ (Abbildung 3.8(e)) ist die zweite Rotation vollständig abgeschlossen, der Flügel hat seinen ursprünglichen Anstellwinkel von 45° wieder erreicht. Der *LEV2* ist vom Flügel abgelöst, wobei ein Teil des Wirbels in negativer Y-Achsenrichtung die Flügelfläche hinabgewandert ist und ein weiterer Teil (an der Flügelspitze) erhalten geblieben ist. Durch die abgeschlossene Rotation hat sich ein weiterer *Rotational Starting Vortex (RSV2)* ausgebildet. *WTV1* ist fast vollständig dissipiert und bewegt sich stark deformiert in positiver Z-Achsenrichtung fort. *WTV2* ist noch vollständig erhalten. *TSV2* und *RSV1* sind noch zu erkennen.

In Abbildung 3.8(f) ist der Flügel wieder in seiner Ausgangsposition angelangt. Auch der Anstellwinkel von 45° entspricht der ursprünglichen Einstellung. *TSV2* und *RSV1* sind fast vollständig dissipiert. Auch *WTV1* ist nicht mehr zu erkennen. *WTV2* ist teils durch die Flügelbewegung, teils durch seine eigene Bewegung in negative Y-Achsenrichtung verschoben. Deutlich zu erkennen ist das erneute Ausbilden eines *Leading Edge Vortex (LEV3)* und eines *Wing Tip Vortex (WTV3)*. *TSV3* und *RSV2* sind als Wirbelpaar deutlich zu erkennen. *TSV2* und *RSV1* sind zwar deutlich deformiert aber immer noch sichtbar.

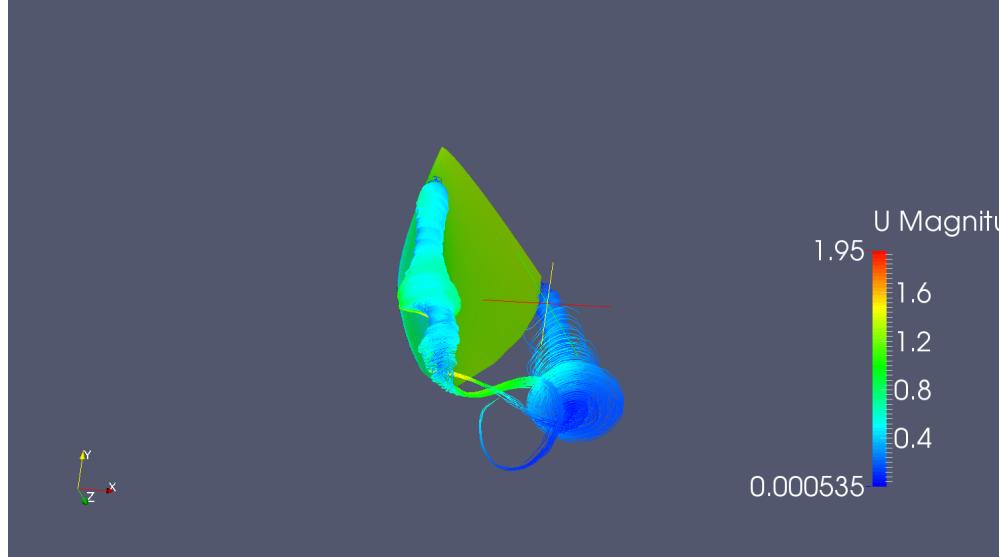


Abbildung 3.9: Detailansicht von *Leading Edge Vortex* und *Translational Starting Vortex* zum Zeitpunkt $t = 0.1s$. Man beachte den helixförmigen Charakter des *LEV*.

Abbildung 3.9 zeigt im Detail den *Leading Edge Vortex* und den *Translational Starting Vortex* zum Zeitpunkt $t = 0.1s$. Deutlich zu erkennen ist, dass der *LEV* noch am Flügel anhaftet, der Flügelvorderkante folgt, während der zweite Wirbel (*TSV*) bereits vom Flügel abgelöst ist. Des Weiteren ist der *WTV* als dünner Streifen erkennbar, welcher mit dem *TSV* verbunden ist. Besondere Beachtung sollte dem helixförmigen Charakter des *LEV* geschenkt werden. Während der *TSV* einen geringen Anteil an Strömung entlang der Achse des Wirbels aufweist, ist dies beim *LEV* wesentlich ausgeprägter. Dieser Aspekt des *LEV* ist von eklatanter Bedeutung.

3.3.3 Sekundärströmungen

Abbildung 3.10 zeigt nochmals den *Leading Edge Vortex* zum Zeitpunkt $t = 0.1s$ im Detail. Des Weiteren sind sog. *wall-bounded Streamlines*, Stromlinien an der Flügeloberfläche dargestellt (siehe Listing 2.7). Deutlich ist der Einfluss des *LEV* auf die Strömung an der Oberfläche erkennbar. Der *LEV* selbst nimmt in Längsrichtung des Flügels an Durchmesser zu. Der Betrag der Wirbelstärke an den Rändern des Wirbels sinkt mit steigendem Durchmesser.

3.3.4 Kräfte

Im zeitlichen Mittel ergeben sich für die generierten Kräfte im Zeitraum von $t = 0s$ bis $t = 2.1s$ folgende Werte:

- Kraft in X-Achsenrichtung: $9.1 \times 10^{-4} \text{ kg} \cdot \text{m} \cdot \text{s}^{-2}$

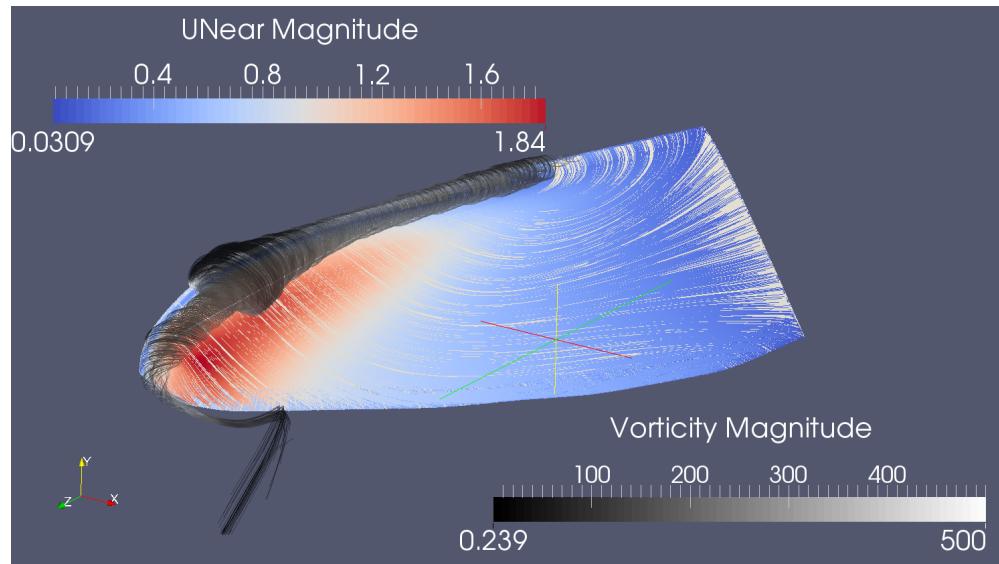


Abbildung 3.10: Sekundärströmungen an der Oberfläche des Flügels, visualisiert mit Hilfe von Stromlinien (siehe Listing 2.7). Zusätzlich ist der *Leading Edge Vortex* mit Hilfe von Stromlinien dargestellt.

- Kraft in Y-Achsenrichtung: $1.45 \times 10^{-2} \text{ kg} \cdot \text{m} \cdot \text{s}^{-2}$
- Kraft in Z-Achsenrichtung: $-4.9 \times 10^{-3} \text{ kg} \cdot \text{m} \cdot \text{s}^{-2}$

Bei dieser Mittlung wurde allerdings die Rotation um die Längsachse des Flügels nicht beachtet.

4 Diskussion

4.1 dynamicTopoFvMesh

Grundsätzlich konnte gezeigt werden, dass die von Menon 2011 entwickelte Software funktioniert. Im Nachfolgenden soll zunächst auf die Testszenarien eingegangen werden um dann *dynamicTopoFvMesh* im Detail zu diskutieren.

4.1.1 Testszenario A

Die in diesem Testszenario erreichten Ergebnisse lassen hoffen, dass es tatsächlich möglich ist, mit *dynamicTopoFvMesh* starke Deformationen wie zum Beispiel das Schlagen von Insektenflügeln numerisch zu untersuchen. Allerdings muss für eine erfolgreiche fluidodynamische Simulation die Gitterqualität verbessert sowie die Gitterglättung stärker genutzt werden. Die Stauchung und Dehnung der Zellen in den Rechnungen von Testszenario A ist zu groß und es kommt noch zu zu stark deformierten Zellen. Ein limitierender Faktor hierbei war jedoch, dass der von Autor gesetzte Parameter *minLengthScale* kleinere Zellen im Bereich zwischen Platte und Wand verhinderte. Eine weitere Schlussfolgerung aus den Berechnungen dieses Szenarios ist, dass das Glättungsintervall größer als das Regenerationsintervall sein sollte. Denn durch das gleichzeitige Regenerieren und Glätten konnte keine optimale Gitterqualität erreicht werden.

4.1.2 Testszenario B

Durch die Erfahrungen in Testszenario A konnte zunächst ein uniformeres Gitter erhalten bleiben. Jedoch brach die Simulation nach etwa der Hälfte der vorgesehenen Bewegung ab, da einige Zellen negative Zellvolumina aufwiesen, obwohl die theoretisch durch *dynamicTopoFvMesh* unterbunden werden sollte. Ein weiterer Faktor für das frühe Abbrechen der Simulationen war ein Programmierfehler in dem Codeteil, der für Patches mit einem festen Wert für die lokale Zelllänge zuständig war. Auch die gewählten Abmessungen der die Kugel umgebenden Box durften zu wenig Raum für das Ausbilden eines optimalen Gitters gelassen haben. Daraus lässt sich schließen, dass für erfolgreiche Simulationen mit *dynamicTopoFvMesh* zunächst ein besseres Verständnis der Parameter nötig ist, sowie das Testen unter parallelisierten Bedingungen. Laut Menon 2011 ist *dynamicTopoFvMesh* bereits in Teilen parallelisiert, jedoch erschwert das parallele Rechnen die Fehlersuche im Code zur Laufzeit.

4.1.3 *dynamicTopoFvMesh*

Grundsätzlich sind die Ergebnisse als positiv zu bewerten. Natürlich ist das Programm nicht nur für die hier umgesetzten Szenarien denkbar, insbesondere auch frei fallende Objekte könnten auf diese Weise gut untersucht werden. In der Kombination mit einer Finite-Elemente-Methode könnten so auch Fluid-Gitter-Interaktionen besser berechnet werden.

Als ein Teilproblem wird vom Autor das als *h-refinement* bezeichnete Verfeinern der Zellen mit Hilfe der lokalen Vergleichslänge betrachtet. Dies kann nicht vom allgemeinen Kollabieren und Teilen der Zellen getrennt werden, sodass bei jeder Simulation ein Umgang mit dieser Verfeinerungstechnik von Nöten ist. Dies hat einen eklatanten Nachteil: die Rechenzeit pro Rechenschritt kann durch ein Anwachsen der Gittergröße, bedingt durch das *h-refinement* stark fluktuieren. Wird die Anzahl der manipulierbaren Zellen begrenzt, unterscheidet *dynamicTopoFvMesh* nicht zwischen den Zellen welche aufgrund des *h-refinements* oder aufgrund schlechter Gitterqualität manipuliert werden soll, so dass ein 'halbfertiges' Gitter entsteht.

Der Status der Bibliothek ist leider nach wie vor als experimentell zu bezeichnen. Während aller durchgeführten Berechnungen kam es zu Speicherzugriffsfehlern, welche das Durchführen von Simulationen erheblich erschweren. Ein Fehler konnte nach dem Durcharbeiten des Quellcodes vom Autor selbst korrigiert werden, im Laufe des Projekts sind dennoch weitere Programmierfehler im Code des Programms entdeckt worden.

Der kritischste Punkt bei der allgemeinen Betrachtung ist aber ein gänzlich anderer. Es existiert eine als rudimentär zu bezeichnende Dokumentation und es gibt kaum Beispiele oder Erfahrungen anderer Wissenschaftler mit dem Programm. Dies macht es extrem schwierig Simulationen zu Implementieren, da durch die fehlende Dokumentation lediglich durch Codeanalyse oder Ausprobieren die für die korrekte Nutzung von *dynamicTopoFvMesh* benötigten Parameter ermittelt werden können.

Des Weiteren ist die Beschränkung auf Tetraedergitter als ebenfalls kritisch zu beurteilen. Tetraedergitter bieten zwar den Vorteil, dass sie (relativ) einfach zu generieren sind und komplexe Geometrien gut abbilden können, jedoch geht jedes Tetraedergitter mit einer großen Nicht-Orthogonalität einher, was beim Lösen der Navier-Stokes-Gleichungen Korrekturschleifen nötig macht.

Auch die fehlende Unterstützung von Prismenschichten an den Oberflächen von Geometrien zum besseren Auflösen des Druckgradienten ist ein erheblicher Nachteil im Gegensatz zu kommerziell verfügbarer Programm. Das Fehlen dieser Option kann durch extrem kleine Zellen an der Oberfläche der Geometrie ausgewogen werden, jedoch muss hier beachtet werden, dass selbst kleinste Tetraederzellen nicht die Qualität von orthogonalen Prismenschichten erreichen können. Auch die allgemein hohe Anzahl von Zellen im Vergleich zu orthogonalen Hexaedergittern ist ein Nachteil.

Aus diesen Gründen gelang es im Rahmen dieser Arbeit nicht mit Hilfe dieser Technik Flügelschlag zu simulieren. In den Augen des Autors ist diese Technik jedoch nach wie

4 Diskussion

vor numerisch attraktiv, so dass unter allen Umständen die Weiterentwicklung der Software betrieben werden sollte. Ohne eine solche numerische Technik, wird es zumindest mit OpenFOAM auf absehbare Zeit nicht möglich sein ein vollständiges Modell einer Libelle zu simulieren.

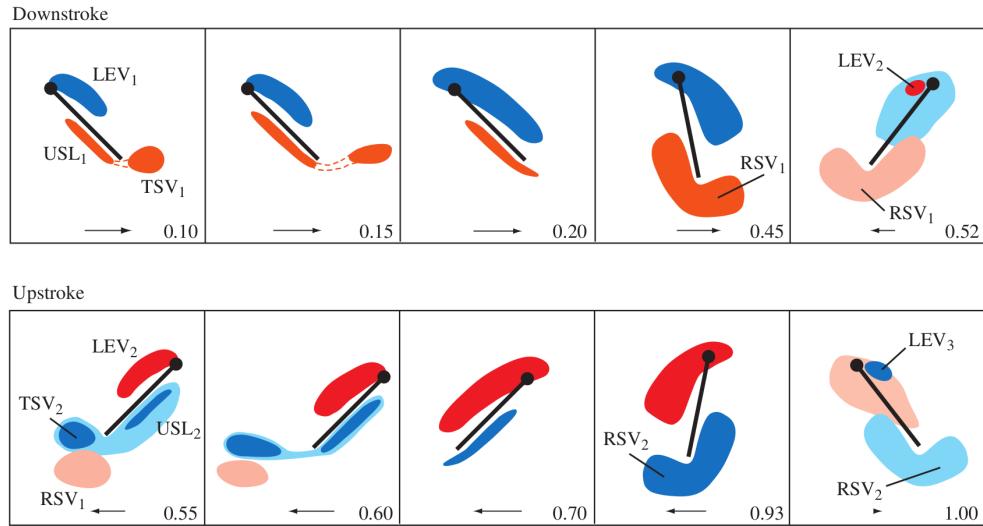


Abbildung 4.1: Die zu erwartenden Strömungsstrukturen im Schlagflug. Verändert nach Birch und Dickinson (2003)

4.2 Radiale Basisfunktionen

Die mit Hilfe der *Radialen Basisfunktionen* erzielten Ergebnisse sind zunächst sehr positiv zu bewerten. Insbesondere im Hinblick auf die Gitterqualität der Simulationen in Relation zu den erreichten Translations und Rotationsamplituden. Obwohl die Initial und Randbedingungen nicht optimal gewählt sind und auch die Zahlenwerte der Simulation mit Vorsicht interpretiert werden sollten, so ist es doch bemerkenswert wie gut dieser Ansatz (im zweidimensionalen) funktioniert. Nichtsdestotrotz muss beachtet werden, dass keinerlei Turbulenzmodellierung eingesetzt wurde. Dies kann bei diesem Reynoldszahlbereich bereits als unzureichend angesehen werden. Im Vergleich mit Birch und Dickinson (2001), Birch und Dickinson (2003), Altshuler, Dudley und Charles P. Ellington (2004), Altshuler, Dudley und McGuire (2004) und F. M. Bos, Oudheusden und Bijl (2013) zeigt sich, dass alle Strömungsstrukturen, welche bei dieser Art von Kinematik zu erwarten sind identifiziert werden konnten (Siehe Abbildung 4.1). Auch sind kaum Artefakte, welche sich auf schlechte Gitterqualität oder Interpolationsfehler zurück führen lassen erkennbar. Es bleibt abzuwarten ob sich diese Technik auch in der dreidimensionalen Anwendung bewährt. F. Bos u. a. (2007) zeigte aber bereits, dass sich *RBF* auch für dreidimensionale Anwendungen eignet. Nachteilig sind zu bewerten ist, dass es zu Hystereseffekten innerhalb des Gitters kommen kann. Da innerhalb dieser Arbeit keine Parameterstudie zu den zugrunde liegenden radialen Basisfunktionen durchgeführt wurde, liegt die Vermutung nahe, dass insbesondere Hystereseffekte durch die Wahl geeigneter Funktionen und Parameter kontrolliert werden können. Als weiterer Nachteil könnte allerdings auch aufgefasst werden, dass zur Umsetzung einer Simulation mit *RBF* Programmierkenntnisse in C++ benötigt werden, da ein individuelles

4 Diskussion

kinematisches Modell benötigt wird. Dies dürfte den Zugang zu dieser numerischen Technik für Einsteiger im CFD-Bereich deutlich erschweren. Die Modellierungsfreiheiten die jedoch durch die individuelle Programmierung gegeben werden, gleichen in den Augen des Autors diesen Nachteil vielfach aus. Ein weiterer Vorteil dieser numerischen Technik ist es, dass zweidimensionale Gitter numerisch bei weitem nicht so rechenintensiv sind wie dreidimensionale Gitter. Sollen mit Hilfe von *RBF* besonders große Bewegungsamplituden umgesetzt werden, kann das umgebende Gitter stark vergrößert werden und die Zellmanipulation in diesen Bereich verlagert werden. So dass zum Einen eine konstante Gitterqualität an der Oberfläche und der näheren Umgebung des bewegten Objekts gewährleistet ist und durch die Verlagerung der Zellmanipulation in die äußeren Bereich des Gitters mit einer (angenommenen) größeren mittleren Zellgröße wesentlich größere absolute Zellbewegungen bei gleicher (relativer) Gitterqualität vorgenommen werden können.

Diese Methode ist insbesondere für komplexe zweidimensionale Bewegungen, insbesondere durch ihre freie Programmierbarkeit gut geeignet.

4.3 Arbitrary Mesh Interface

Auch die mit der in OpenFOAM bewährten Technik *Arbitrary Mesh Interface* erhaltenen Ergebnisse sind als positiv zu bewerten. Obwohl die Bewegungen, welche dem Schlagflug eines Kolibris nachempfunden sind, stark abstrahiert wurden, konnte erfolgreich Auftrieb generiert werden. Eine einfache Rechnung, ausgehend von $5 \cdot 10^{-3} \text{ kg}$ Körpergewicht und einer Erdbeschleunigung von $9.81 \text{ kg} \cdot \text{m}^{-1} \cdot \text{s}^{-2}$ zeigt, dass, obwohl keine Kräfte während der Rotation beachtet wurden, mit $2 \times 1.45 \times 10^{-2} \text{ kg} \cdot \text{m} \cdot \text{s}^{-2}$ knapp 80 % des benötigten Auftriebs generiert werden konnte. Ein großer Vorteil dieser Technik ist, dass, obwohl die Implementierung einfach ist, bereits sehr komplexe Strömungen simuliert werden können. Insbesondere dreidimensionale Effekte, wie das Verhalten des *Leading Edge Vortex* und des *Wing Tip Vortex* seien hier genannt. Aber auch Sekundärströmungen, wie zum Beispiel die Querströmungen auf der Flügeloberfläche oder die Interaktion des Flügels mit seinem eigenen Nachlauf, können sehr gut abgebildet werden. Die Ergebnisse zeigen alle zu erwartenden Strukturen des Schlagflugs auf. Überraschend an den Ergebnisse ist die Interaktion zwischen *Translational Starting Vortex* und *Rotational Starting Vortex*, welche sich zu einem ringförmigen Wirbelpaar verbinden. Dieses Wirbelsystem bleibt sehr lange (im Vergleich zu bspw. dem *LEV*, siehe Abbildung 3.8) erhalten und deutet damit auf einen hohen Gehalt an kinetischer Energie hin. Quantitativ sind auch diese Ergebnisse zunächst mit Skepsis zu betrachten. Obwohl, wie vorher aufgeführt, die beobachteten Phänomene weitestgehend den Erwartungen entsprechen, muss in weitergehenden Untersuchungen der Frage nach Turbulenzmodellierung nachgegangen werden. Obwohl die Reynoldsnummern im niedrigen fünstelligen Bereich angesiedelt sind, kann nicht ausgeschlossen werde, dass, insbesondere durch den Charakter der Strömung, (induziert durch den Flügelschlag, wiederholte Interaktion des Flügels mit dem eigenen Nachlauf, hohe Winkelgeschwindigkeiten während der Supination des Flügels) die Strömung als laminar angesehen werden kann. Gleichzeitig ist jedoch davon auszugehen, dass der Gehalt an kinetischer Energie im System zu gering ist um mit klassischen High-Reynolds-Stressmodellen, wie beispielsweise dem $k - \varepsilon$ oder dem $k - \omega - SST$ Modell die eventuell auftretenden Turbulenzen zu modellieren. Ein mögliches Turbulenzmodell, welches in der Lage sein könnte auch bei geringen Reynoldsnummern akkurate Ergebnisse zu generieren, ist das von Walters und Leyland (2005) entwickelte $k - k_L - \omega$ Turbulenzmodell. Leider unterstützt die Implementierung dieses Modells in OpenFOAM die hier genutzte *AMI*-Technik jedoch nicht. Ein großer Nachteil der *AMI*-Technik ist, dass nur sehr bedingt translativ Bewegungen simuliert werden können. Der zugrunde liegende Lösungsalgorithmus, in OpenFOAM als *sixDOFSolver* bezeichnet, erlaubt zwar sowohl rotative als auch translativ Bewegungen, das Gitter jedoch nicht. Da das zugrunde liegende Gitter aus zwei Gittern besteht, an deren Grenzflächen interpoliert wird, führt ein Verschieben des inneren Gitters sehr schnell zu Überlappungen. Sobald diese Überlappungen in ihrer Größenordnung die Größenordnung einer Zelle überschreiten, ist keine Translation mehr möglich. Auch die Simulation der Flügel-Körper-Interaktionen sind mit dieser numerischen

4 Diskussion

Technik nicht möglich, da ein Körper mit dem Flügel mit bewegt werden würde. Als größte Schwierigkeit bei der Simulation mit *AMI* haben sich die Randbedingungen erwiesen. Da die Kombination von *AMI* und *6DOFSolver* in diesem Sinne wahrscheinlich nicht vorgesehen ist, hat es sich als ein äußerst mühsames Unterfangen herausgestellt die Randbedingungen für die Rotation zu berechnen, da zwischen einem globalen Koordinatensystem und einem lokalem Koordinatensystem hin- und hergerechnet wird. Dies erfordert, dass die Positionen von min. zwei Punkten auf oder innerhalb der bewegten Geometrie zu jedem Zeitpunkt bekannt sind. Im Rahmen dieser Arbeit wurde diese Problematik zum einen durch ein Programm gelöst, welches die erforderlichen Koordinatentransformationen und Rotationen berechnet. Zum Anderen, da mit dem Fortschreiten des Projekts klar wurde, dass die Bewegungen Abstraktionen unterzogen werden müssen, mit Hilfe eines einfachen Kniffs. Die Koordinaten des Flügels wurden so gewählt, dass durch die Supination des Flügels lediglich zwei Achsen des lokalen Koordinatensystems, die X- und Y-Achse vertauscht wurden. Dies vereinfachte das Festlegen der Randbedingungen (siehe Tabelle 2.10).

5 Fazit & Ausblick

Im Rahmen dieser Arbeit konnten erfolgreich die nach der Literatur (u.A. Birch und Dickinson (2001), Birch und Dickinson (2003), F. Bos u. a. (2007), F. M. Bos, Oudheusden und Bijl (2013) und Dickinson, Lehmann und Sane (1999)) für die dynamische Auftriebsgenerierung verantwortlichen strömungsmechanischen Phänomene simuliert werden.

Leider ist es nicht gelungen die elegante und ressourcensparende Technik von *dynamicTopoFvMesh* zur Simulation des Schlagflugs um- und einzusetzen. Obwohl gezeigt werden konnte, dass die Technik grundsätzlich funktioniert, haben sich die in der Implementierung befindlichen Programmierfehler als zu systemisch herausgestellt, als das *dynamicTopoFvMesh* im großen Maßstab hätten eingesetzt werden können. Auch die Limitierung der Zellen auf Tetraederzellen hat sich als sehr problematisch erwiesen, da dieser Zelltypus insbesondere für Scherschichten ungeeignet scheint. Nichtsdestotrotz erachtet der Autor die von Menon (2011) implementierte Technik als elegantesten und sinnvollsten Ansatz um komplexe Bewegungen zu simulieren; aufbauend auf dieser Arbeit müsste in zukünftigen Arbeiten zunächst eine Re-Implementierung und Dokumentation der Software angefertigt werden. Das Ausweichen auf bereits etablierte Techniken in OpenFOAM hat sich dennoch als äußerst erfolgreich herausgestellt. Obwohl *RBF* noch nicht in der allgemeinen OpenFOAM-Distribution enthalten ist, scheint *RBF* bereits so stabil, dass es ohne weitere Bedenken zur Simulation eingesetzt werden kann. In weiteren Arbeiten sollte *RBF* insbesondere die Eignung von *RBF* dreidimensionalen Simualtionen durchzuführen untersucht werden. Sollten diese Untersuchungen positive Ergebnisse bringen, stände damit ein weiteres robustes und vielfältiges Werkzeug zur Simulation komplexer Bewegungen zur Verfügung. Für zweidimensionale Simulationen konnte im Rahmen dieser Arbeit bereits gezeigt werden, dass *RBF* hinreichen stabil ist um vielfältige Fragestellungen zu untersuchen. Insbesondere das Koppeln von bewegten Objekten könnte mit *RBF* gut umgesetzt werden können. Als hervorragende Technik zur Simulation starren Schlagflugs hat sich *AMI* herausgestellt. Aus der Notwendigkeit eine Alternative zu *dynamicTopoFvMesh* zu finden, konnte mit dieser Technik sehr erfolgreich der dem Schlagflug eines Kolibris nachempfundene Bewegungsablauf simuliert werden. In weiterführenden Arbeiten sollte die Kopplung von *RBF* und *AMI* untersucht werden. Sollte es möglich sein auf diese Weise auch Translationen zu simulieren, wäre die Notwendigkeit für eine Software wie *dynamicTopoFvMesh* nicht mehr so groß.

Bis dahin bleibt es jedoch sehr wünschenswert, wenn nicht notwendig, Techniken wie *dynamicTopoFvMesh* soweit weiter zu entwickeln, das sie nicht nur für Spezialfälle nutzbar sind.

6 Danksagung

Mein größter Dank gilt zunächst Frau Prof. Dr. A. B. Kesel und Herrn Prof. Dr. Albert Baars. Ohne Ihre Unterstützung wäre diese Projektarbeit in diesem Rahmen sicher nicht möglich gewesen. Des Weiteren gilt mein Dank all meinen Studienkollegen und Freunden die mich bei meinem täglichen Kampf mit OpenFOAM unterstützt haben. Last, but not least, geht ein großes Dankeschön an meine Eltern, ohne deren Unterstützung ich dieses Studium niemals hätte beginnen können.

Anhang

Listing 1: *writeLengthScale*-Tool zum berechnen der mittleren Vergleichlänge aller patches eines Gitters. Die Vergleichlänge wird nach Gleichung 2.3 berechnet

```
1  *---------------------------------------------------------------------*\\
2  ======          |
3  \\\   /  F ield      | foam-extend: Open Source CFD
4  \\\  /  O peration   |
5  \\\ /  A nd         | For copyright notice see file Copyright
6  \\\/  M anipulation |
7  -----
8  License
9  This file is part of foam-extend.
10 foam-extend is free software: you can redistribute it and/or modify it
11 under the terms of the GNU General Public License as published by the
12 Free Software Foundation, either version 3 of the License, or (at your
13 option) any later version.
14 foam-extend is distributed in the hope that it will be useful, but
15 WITHOUT ANY WARRANTY; without even the implied warranty of
16 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
17 General Public License for more details.
18 You should have received a copy of the GNU General Public License
19 along with foam-extend. If not, see <http://www.gnu.org/licenses/>.
20 Description
21 Write the three components of the cell centres as volScalarFields so
22 they can be used in postprocessing in thresholding.
23 *---------------------------------------------------------------------*/
24 #include "argList.H"
25 #include "timeSelector.H"
26 #include "objectRegistry.H"
27 #include "Time.H"
28 #include "fvMesh.H"
29 #include "vectorIOField.H"
30 #include "volFields.H"
31 using namespace Foam;
32 // * * * * *
33 // Main program:
34 int main(int argc, char *argv[])
35 {
```

```

36   timeSelector::addOptions();
37   # include "setRootCase.H"
38   # include "createTime.H"
39   instantList timeDirs = timeSelector::select0(runTime, args);
40   # include "createMesh.H"
41
42   forAll(timeDirs, timeI)
43   {
44     runTime.setTime(timeDirs[timeI], timeI);
45     Info<< "Time = " << runTime.timeName() << endl;
46     // Check for new mesh
47     mesh.readUpdate();
48
49     // get the boundary mesh
50     const polyBoundaryMesh& boundary = mesh.boundaryMesh();
51
52     // iterate over the boundary patches
53     forAll(boundary,patchI)
54     {
55       // get the current boundary patch
56       const polyPatch& bdyPatch = boundary[patchI];
57
58       // Pout << "iterating over faces of patch" << bdyPatch.name() << endl;
59
60       // get the starting cell of the boundary patch
61       //label pStart = bdyPatch.start();
62
63       scalar sumLengthScale = 0.0, nFacesPatch = 0, avgLengthScale = 0.0;
64
65       // iterate over the faces of the boundary patch
66       forAll(bdyPatch,faceI)
67       {
68         sumLengthScale += Foam::sqrt(2.0 * mag(mesh.faceAreas()[faceI]));
69         nFacesPatch++;
70       }
71
72       avgLengthScale = sumLengthScale / nFacesPatch;
73       Pout << "patch " << bdyPatch.name() << " has an average Length Scale of "
74           << avgLengthScale << endl;
75
76       avgLengthScale = 0.0;
77       sumLengthScale = 0.0;
78       nFacesPatch = 0.0;
79     }
  }
```

```
80     Info<< "\nEnd" << endl;
81     return 0;
82 }
83 // *****
```

Listing 2: naca2stl.m - Der zur Generierung von NACA-Profilen eingesetzte Code. Verändert nach Hakon Strandenes

```
1 #!/usr/bin/octave -qf
2
3 % ----- START OF INPUT PARAMETER REGION -----
4
5 % Foil geometry
6 c = 1; % Geometric chord length
7 s = 2; % Span (along y-axis)
8 alpha = 0.0; % Angle of attack (in radians)
9 NACA = [0 0 0 6]; % NACA 4-digit designation as a row vector;
10
11 % Surface resolution parameters
12 Ni = 1000; % Number of interpolation points along the foil
13
14 % ----- END OF INPUT PARAMETER REGION -----
15
16 % ----- LICENCE -----
17 %
18 % Copyrighted 2011, 2012 by Hakon Strandenes, hakostra@stud.ntnu.no
19 %
20 % This program is free software: you can redistribute it and/or modify
21 % it under the terms of the GNU General Public License as published by
22 % the Free Software Foundation, either version 3 of the License, or
23 % (at your option) any later version.
24 %
25 % This program is distributed in the hope that it will be useful,
26 % but WITHOUT ANY WARRANTY; without even the implied warranty of
27 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
28 % GNU General Public License for more details.
29 %
30 % You should have received a copy of the GNU General Public License
31 % along with this program. If not, see <http://www.gnu.org/licenses/>.
32 %
33
34 % Create a vector with x-coordinates, camber and thickness
35 beta=linspace(0,pi,Ni);
36 x = c*(0.5*(1-cos(beta)));
37 z_c = zeros(size(x));
38 z_t = zeros(size(x));
39 theta = zeros(size(x));
40
41 % Values of m, p and t
42 m = NACA(1)/100;
43 p = NACA(2)/10;
```

```

44 t = (NACA(3)*10 + NACA(4))/100;
45
46 % Calculate thickness
47 % The upper expression will give the airfoil a finite thickness at the trailing
48 % edge, which might cause trouble. The lower expression is corrected to give
49 % zero thickness at the trailing edge, but the foil is strictly speaking no
50 % longer a proper NACA airfoil.
51 %
52 % See http://turbmodels.larc.nasa.gov/naca4412sep_val.html
53 %      http://en.wikipedia.org/wiki/NACA_airfoil
54
55 %z_t = (t*c/0.2) * (0.2969.* (x/c).^0.5 - 0.1260.* (x/c) - 0.3516.* (x/c).^2 +
56 %      0.2843.* (x/c).^3 - 0.1015.* (x/c).^4);
56 z_t = (t*c/0.2) * (0.2969.* (x/c).^0.5 - 0.1260.* (x/c) - 0.3516.* (x/c).^2 +
57 %      0.2843.* (x/c).^3 - 0.1036.* (x/c).^4);
57 % Calculate camber
58 if (p > 0)
59 % Calculate camber
60 z_c = z_c + (m.*x/p^2) .* (2*p - x/c) .* (x < p*c);
61 z_c = z_c + (m.*(c-x)/(1-p)^2) .* (1 + x/c - 2*p) .* (x >= p*c);
62 % Calculate theta-value
63 theta = theta + atan( (m/p^2) * (2*p - 2*x/c) ) .* (x < p*c);
64 theta = theta + atan( (m/(1-p)^2) * (-2*x/c + 2*p) ) .* (x >= p*c);
65 end
66 % Calculate coordinates of upper surface
67 Xu = x - z_t.*sin(theta);
68 Zu = z_c + z_t.*cos(theta);
69 % Calculate coordinates of lower surface
70 Xl = x + z_t.*sin(theta);
71 Zl = z_c - z_t.*cos(theta);
72 % Rotate foil to specified angle of attack
73 upper = [cos(alpha), sin(alpha); -sin(alpha), cos(alpha)] * [Xu ; Zu];
74 lower = [cos(alpha), sin(alpha); -sin(alpha), cos(alpha)] * [Xl ; Zl];
75 % Merge upper and lower surface (NB: Assume that the trailing edge is sharp)
76 % (see comments w.r.t. thickness calculation above)
77 X = [ upper(1,:) lower(1,Ni-1:-1:2) ];
78 Z = [ upper(2,:) lower(2,Ni-1:-1:2) ];
79 N = length(X);
80 % Triangulate the end surface
81 tri = [1, 2, N];
82 for i=2:Ni-1
83 tri = [tri; i, i+1, N-i+2];
84 end
85 for i=Ni+1:N-1
86 tri = [tri; i, i+1, N-i+2];

```

```

87 end
88 % Make it 3D
89 X = [X X];
90 Z = [Z Z];
91 Y = [(s/2)*ones(1,N) -(s/2)*ones(1,N)];
92 % Triangulate the second end surface
93 tri = [tri; tri(:,2)+N, tri(:,1)+N, tri(:,3)+N];
94 % Triangulate the top and bottom
95 for i=1:N-1
96 tri = [tri; i, N+i, i+1];
97 end
98 tri = [tri; N, 2*N, 1];
99 for i=N+1:(2*N-1)
100 tri = [tri; i, i+1, i-N+1];
101 end
102 tri = [tri; 2*N, N+1, 1];
103 % Open file
104 fo = fopen('airfoil.stl', 'w');
105 % Write file
106 fprintf(fo, 'solid airfoil\n');
107 for i=1:length(tri)
108 % Calculate normal vector
109 AB = [X(tri(i,2)) - X(tri(i,1)), Y(tri(i,2)) - Y(tri(i,1)), Z(tri(i,2)) - Z(tri(i,1))];
110 AC = [X(tri(i,3)) - X(tri(i,1)), Y(tri(i,3)) - Y(tri(i,1)), Z(tri(i,3)) - Z(tri(i,1))];
111 n = cross(AB, AC) ./ norm(cross(AB, AC));
112 % Write facet
113 fprintf(fo, ' facet normal %e %e %e\n', n);
114 fprintf(fo, '     outer loop\n');
115 fprintf(fo, '         vertex %e %e %e\n', X(tri(i,1)), Y(tri(i,1)), Z(tri(i,1)));
116 fprintf(fo, '         vertex %e %e %e\n', X(tri(i,2)), Y(tri(i,2)), Z(tri(i,2)));
117 fprintf(fo, '         vertex %e %e %e\n', X(tri(i,3)), Y(tri(i,3)), Z(tri(i,3)));
118 fprintf(fo, '     endloop\n');
119 fprintf(fo, ' endfacet\n');
120 end
121 fprintf(fo, 'endsolid airfoil\n');
122 % Close file
123 fclose(fo);

```

Literaturverzeichnis

- Altshuler, Douglas L., Robert Dudley und Charles P Ellington (2004). „Aerodynamic forces of revolving hummingbirdwings and wing models“. In: *Journal of Zoology* 264, S. 327–332.
- Altshuler, Douglas L., Robert Dudley und Jimmy A. McGuire (2004). „Resolution of a paradox: Hummingbird flight at high elevation does not come without a cost“. In: *Proceedings of the National Academy of Sciences of the United States of America* 101, S. 17731–17736.
- Birch, James M und Michael H Dickinson (2001). „Spanwise flow and the attachment of the leading-edge vortex on insect wings“. In: *Nature* 412.6848, S. 729–733.
- (2003). „The influence of wing-wake interactions on the production of aerodynamic forces in flapping flight“. In: *The Journal of Experimental Biology* 206, S. 2257–2272.
- Boer, Aukje de, Martijn S. van der Schoor und Hester Bijl (2006). „New Method for Mesh moving based on Radial Basis Function interpolation“. In: *ECCOMAS CFD 2006*.
- Bos, FM u. a. (2007). „Numerical study of kinematic wing models of hovering insect flight“. In: *AIAA Aerospace Sciences Meeting and Exhibit*, S. 1–18.
- Bos, Frank M, Bas W van Oudheusden und Hester Bijl (2013). „Wing performance and 3-D vortical structure formation in flapping flight“. In: *Journal of Fluids and Structures* 42, S. 130–151.
- Chai, Peng, Johnny S. C. Chen und Robert Dudley (1997). „Transient Hovering Performance of Hummingbirds under conditions of maximal loading“. In: *Journal of experimental Biology* 200, S. 921–929.
- Dickinson, Michael H, Fritz-Olaf Lehmann und Sanjay P Sane (1999). „Wing rotation and the aerodynamic basis of insect flight“. In: *Science* 284.5422, S. 1954–1960.
- Ellington, Charles P. (2006). „Insects versus birds: the great divide“. In: *American Institute of Aeronautics and Astronautics: Aerospace Sciences Meeting and Exhibit* 35, S. 1–6.
- Ellington, Charles P u. a. (1996). „Leading-edge vortices in insect flight“. In:
- Ferziger, Joel H. und Milovan Perić (2008). *Numerische Strömungsmechanik*. Springer Verlag Berlin Heidelberg.
- Hedrick, Tyson L. u. a. (2011). „Morphological and kinematic basis of the hummingbird flight stroke: scaling of flight muscle transmission ratio“. In: *Proceedings of the Royal Society B* 1, S. 1–7.
- Jasak, Hrvoje und Henrik Rusche (2009). „Dynamic mesh handling in OpenFOAM“. In: *Proceeding of the 47th Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Orlando, FL*.
- Jasak, Hrvoje und Zeljko Tukovic (2006). „Automatic mesh motion for the unstructured finite volume method“. In: *Transactions of FAMENA* 30.2, S. 1–20.

- Kassiotis, Christophe (2008). „Which strategy to move the mesh in the Computational Fluid Dynamic code OpenFOAM“. In: *Report 'Ecole Normale Sup'erieure de Cachan. Available online: http://perso. cnrs. org/kassiotis/openfoam/movingmesh. pdf*.
- Kruyt, Jan W. u. a. (2014). „Hummingbird wing efficacy depends on aspect ratio and compares with helicopter rotors“. In: *Journal of The Royal Society Interface* 11.99. doi: 10.1098/rsif.2014.0585.
- McIntosh, Sen H., Sunil K. Agrawal und Zaeem Khan (2006). „Design of a Mechanism for a Biaxial Rotation of a Wing for a Hovering Vehicle“. In: *IEEE/ASME Transactions on Mechatronics* 11, S. 145–153.
- Menon, Sandeep (2011). „A numerical study of droplet formation and behavior using interface tracking methods“. Diss. University of Massachusetts.
- Möller, Sebastian (2011). *Starrkörpersimulation mit sechs Freiheitsgraden. Teil I: Implementierung von Kraft und Bewegungsberechnungen*. Techn. Ber. Hochschule Bremen.
- Raney, David L. und Eric C. Slominski (2004). „Mechanization and Control Concepts for Biologically Inspired Micro Air Vehicles“. In: *Journal of Aircraft* 41, S. 1257–1265.
- Spurk, Joseph H. und Nuri Aksel (2010). *Strömungslehre*. Springer Verlag Berlin Heidelberg.
- Styles, F.Gary, Douglas L. Altshuler und Robert Dudley (2005). „Wing morphology and flight behaviour of some north american hummingbird species“. In: *The Auk* 872, S. 872–886.
- Tobalske, Bret W. u. a. (2007). „Three-Dimensional kinematics of hummingbird flight“. In: *Journal of Experimental Biology* 210, S. 2368–2382.
- Walters, D. Keith und James H. Leylek (2005). „Computational Fluid Dynamics Study of Wake-Induced Transition on a Compressor-Like Flat Plate“. In: *Journal of Turbomachinery* 127, S. 52–63.
- Warrick, Douglas R., Bret W. Tobalske und Donald R. Powers (2005). „Aerodynamic of the hovering hummingbird“. In: *Letters to Nature* 435, S. 1094–1097.