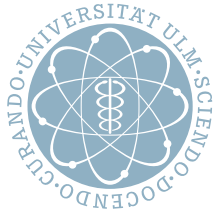

ulm university universität
uulm

Universität Ulm
Fakultät für Mathematik und
Wirtschaftswissenschaften

Numerische Optimierung von Schiffen mit VSP: Automatische Gittergenerierung

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Mathematik und Wirtschaftswissenschaften
der Universität Ulm

vorgelegt von

Michael Hopfensitz

aus Dinkelsbühl

2010

Amtierender Dekan:

Prof. Dr. Werner Kratz

Erstgutachter:

Prof. Dr. Karsten Urban

Zweitgutachter:

Prof. Dr. Stefan Funken

Tag der Promotion:

29.10.2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

1D	Eindimensional
2D	Zweidimensional
3D	Dreidimensional
AD	Automatisches Differenzieren
CAD	Computer Aided Design
CD	Central Differencing
CFD	Computational Fluid Dynamics (numerische Strömungsmechanik)
COMET	Continuum Mechanics Engineering Tool
FDM	Finite-Differenzen-Methode
FEM	Finite-Elemente-Methode
FVM	Finite-Volumen-Methode
KV	Kontrollvolumen
UD	Upwind Differencing
VSP	Voith-Schneider-Propeller

Zusammenfassung

Diese Dissertation entstand im Rahmen des Projektes SimuVSP in enger Kooperation der Universität Ulm mit Voith Turbo Schneider Propulsion & Co. KG und wurde gefördert vom Bundesministerium für Wirtschaft und Technologie. Das Projekt beschäftigt sich mit der numerischen Optimierung, wobei Schiffsgeometrien optimiert werden sollen, die mit einem Voith-Schneider-Propeller ausgestattet sind. Das Gesamtproblem der Optimierung teilt sich dabei in vier Bereiche auf. Die parametrische Modellierung der Schiffskörpergeometrie, die automatische Netzgenerierung für diese Art von Schiffen, die Strömungsberechnung und die Optimierung. Diese Dissertation konzentriert sich auf die automatische Netzgenerierung, welche für die Optimierung solcher Geometrien unentbehrlich ist.

Zunächst werden Grundlagen und verschiedene Algorithmen zur Gittergenerierung beschrieben, wobei besonders auf die Gitterqualität, den sogenannten Paving-Algorithmus, Algorithmen zur Tetraedervernetzung und bereits existierende Vernetzungsverfahren zur Erzeugung von Hexaedernetzen eingegangen wird. Bei der numerischen Strömungssimulation liegt das Augenmerk im Besonderen auf der Finite-Volumen-Methode. Dabei werden vornehmlich die Zusammenhänge der Ergebnisse einer solchen Simulation mit dem zugrunde liegenden Gitter herausgearbeitet.
Die detaillierte Beschreibung des entwickelten automatischen Netzgenerierers **vmesh** beinhaltet zum einen die neu entstandenen Algorithmen und das entworfene Vernetzungskonzept und zum anderen die Bedienung des Programms. Das Vernetzungstool **vmesh** ist in der Lage für komplexe Geometrien vollautomatisch ein hybrides Qualitätsnetz für Strömungsberechnungen zu generieren. Das Netz für die Strömungsberechnung wird für das Vollmodell der Geometrie erzeugt. Dem Vernetzungskonzept liegt ein Schichtenmodell und die Idee der Verwendung eines hybriden Netzes, bestehend aus Hexaedern, Tetraedern und Pyramiden, zugrunde. Für die Grenzschicht der zu vernetzenden Geometrie wurde ein Offset-Algorithmus entwickelt, der es ermöglicht aus einem bestehenden Oberflächennetz aus Vierecken ein Volumennetzetz aus Hexaedern für diesen Bereich zu generieren. Algorithmen hierzu sind aus der Literatur nicht bekannt.
Im Anschluß wird die geometriebasierte Optimierung und die in **vmesh** umgesetzte netzbasierte Formoptimierung vorgestellt. Zuletzt werden die von **vmesh** generierten Netze für verschiedene Geometrien gezeigt und die erzielten Ergebnisse aus den Vergleichsrechnungen und der Optimierung veranschaulicht.

Abstract

This dissertation was produced within the SimuVSP project, in close cooperation with Ulm University and Voith Turbo Schneider Propulsion & Co. KG, with support from the Federal Ministry for Economy and Technology. The project concentrates on the numerical optimization of ship geometries, equipped with a Voith-Schneider-Propeller. The optimization problem is divided into four parts; the parametric modeling of ship hull geometries, the automatic grid generation of these types of vessels, the CFD-computation and the optimization itself. This dissertation focuses on the automatic grid generation, which is essential for the optimization of such geometries.

Firstly, foundations and various algorithms for grid generation are described. In particular, the grid quality, paving-algorithm, algorithms to obtain tetrahedrons and meshing methods to obtain hexahedrons are addressed. In the numerical flow simulation, the focus is specifically on the Finite-Volume-Method, which primarily demonstrates the relationship of the results of such a simulation with the underlying grid.
The detailed description of the developed mesh generation tool, **vmesh**, includes the meshing algorithms, the meshing concept and the handling of the program. This tool automatically generates a hybrid quality grid for CFD calculations for complicated geometries. The grid for the CFD-computation is generated for the complete model of the geometry. The concept for the grid is based on a layer model and the idea of using a hybrid grid, consisting of hexahedrons, tetrahedrons and pyramids. To make this possible, an offset-algorithm was developed which enables the generation of a volume grid of hexahedrons for the boundary layer out of a surface mesh of quadrangles. Moreover, at this time, there are no documented algorithms.
Subsequently, the geometry-based optimization and the mesh-based optimization, which are implemented in **vmesh** are presented. Lastly, the grids generated by **vmesh** for various geometries and the achieved results from the comparative calculations and the optimization are demonstrated.

Kapitel 1

Einleitung

Die Bedeutung der Simulation von physikalischen oder chemischen Vorgängen hat in den letzten Jahren kontinuierlich zugenommen. Diese Tatsache ist nicht zuletzt auf die rasante Entwicklung von Rechnerkapazitäten und die immer preiswerter werdenden Speicherkapazitäten zurückzuführen. Der Forschung und Entwicklung in Industrie und Wirtschaft dienen Simulationen in den verschiedensten Bereichen, wie z.B. beim Produktdesign im Maschinen-, Auto-, Schiffs- und Flugzeugbau oder im Bereich der Chemie, des Elektromagnetismus sowie zur Berechnung von Strömungen im Wasser oder in der Luft. Dabei werden sehr verschiedene Vorgänge, wie z.B. Formgebung, thermische Druckentwicklung, elastische und plastische Verformungen oder Strömungen, simuliert. Solche Vorgänge können durch partielle Differentialgleichungen beschrieben werden, die ihrerseits wiederum numerisch mit der Finite-Elemente-Methode oder der Finite-Volumen-Methode gelöst werden. Dafür werden entsprechend angepasste Netze benötigt, [30].

An der Vielzahl der Anwenungsmöglichkeiten kann man schon erkennen, dass es nahezu unmöglich ist, einen universellen Vernetzer zu implementieren, der allen Anforderungen in höchstem Maße gerecht wird. Die Dimension der Netze spielt ebenfalls eine wichtige Rolle. Für einige Anwendungen reicht die Vernetzung der Oberfläche aus, für andere ist ein dreidimensionales Netz nötig. Auch ist die Netzstruktur und der Zelltyp oft von entscheidender Bedeutung, wenn es um das Konvergenzverhalten solcher Simulationen geht. Um für ein entsprechendes Problem bei der dazugehörigen Simulation aussagekräftige und relevante Ergebnisse zu erhalten, muss eine Vielzahl von Bedingungen an das Netz erfüllt sein. Deshalb wurde eine Vielzahl von Netzgeneratoren entwickelt.

Bereits Anfang der 70er Jahre wurden die ersten Netzgeneratoren entwickelt. Die ständige Verbesserung, Weiterentwicklung und Forschung auf diesem Gebiet dauert bis jetzt an. Dabei wurden einige Methoden entwickelt, wie die Delaunay-Triangulation (Kapitel 2.3), das Advancing-Front-Verfahren (Kapitel 2.2), das Medial-Axis-Verfahren und das Octree-Verfahren (Kapitel 2.4.2), die sich bis zum heutigen Tage bewährt haben, [30]. Ob eine Vernetzungsmethode für bestimmte Problemstellungen einsetzbar ist, hängt oft davon ab, ob die Methode automatisiert abläuft.

Wie notwendig die automatische Gittergenerierung in der Stömungsmechanik und der numerischen Optimierung ist, erkennt man an den folgenden zwei Punkten.
Zum einen an der Zeit, die für eine manuelle Netzgenerierung benötigt wird. Ein erfahrener Ingenieur braucht für die manuelle Generierung eines Gitters für komplexe Geometrien mehrere Tage. Im Vergleich dazu liegt die Rechenzeit der CFD-Rechnung, die auf mehreren Prozessoren gerechnet wird, im Stundenbereich. Die automatische Netzgenerierung liegt ebenfalls im Minuten- bis Stundenbereich.
Die Notwendigkeit der automatischen Gittergenerierung erkennt man zum anderen daran, dass eine numerische Optimierung, bei der bei jedem Durchlauf neu vernetzt werden muss, ohne die automatische Gittergenerierung nahezu unmöglich ist.

1.1 Projekt

Das Projekt SimuVSP wurde vom Bundesministerium für Wirtschaft und Technologie gefördet. Dieses wurde in enger Kooperation des Institutes der Numerischen Mathematik der Universität Ulm mit Voith Turbo Schneider Propulsion durchgeführt. In diesem Rahmen entstand diese Dissertation.
In Zusammenarbeit mit Voith Turbo Schneider Propulsion sollen die Rümpfe von Schiffen, die mit Voith-Schneider-Propeller (VSP) ausgerüstet sind, hinsichtlich ihrer Strömungsverluste optimiert werden. Dabei teilt sich das Gesamtproblem der Optimierung in vier Teilprobleme auf. Die parametrische Modellierung der Schiffskörpergeometrie, die automatische Netzgenerierung für diese Art von Schiffen, die Strömungsberechnung und die Optimierung.
Ein Teil des Projekts bestand in der Aufgabe einen automatischen Vernetzer für VSP-getriebene Schiffe zu entwickeln und zu implementieren. Das dabei entstandene Vernetzungstool **vmesh** soll in dieser Promotionsarbeit vorgestellt werden. Dabei wurden neue Vernetzungsmethoden und Algorithmen entwickelt. Darüber hinaus werden auch die Möglichkeiten der Optimierung, die sich dadurch ergeben, untersucht und erste Ergebnisse gezeigt.
Diesem Projekt sind bereits eine Reihe von anderen Projekten mit Voith Turbo Schneider Propulsion vorangegangen, die jeweils zu signifikanten Verbesserungen des Wirkungsgrades geführt haben. Zum einen die Optimierung der Flügelwinkelkurve des VSP und zum anderen die Optimierung der Flügelblätter des VSP.
In der Dissertation von J. C. Matutat [47] ist das, ebenfalls in diesem Projekt entstandene, Modellierungstool *PaShiMo* beschrieben.

1.2 Der Voith-Schneider-Propeller

Der Voith-Schneider-Propeller (VSP) (Abb.: 1.2), auch Zykloidalpropeller genannt, ist ein einzigartiger Schiffsantrieb, welcher auf einer Erfindung von Ernst Schneider von 1925 basiert.
Beim VSP rotieren senkrecht im Wasser stehende Flügel in einer Kreisbahn um eine

vertikale Drehachse. Neben der Rotation auf dem sogenannten Flügelkreis führen die Flügel dabei gleichzeitig eine zusätzliche Schwingbewegung aus, bei der diese so gesteuert werden, dass ein gerichteter Schub entsteht (Abb.: 1.1), [36]. So kann bei konstanter Drehzahl der Betrag und die Richtung des Schubs stufenlos variiert werden. Der VSP ist somit Antrieb und Steuerungsorgan zugleich, [39].

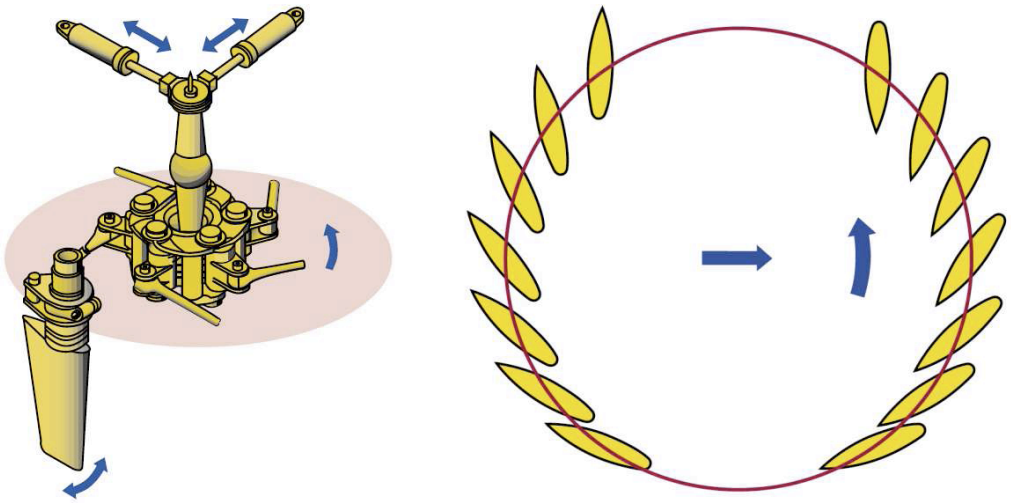

Abbildung 1.1: Bewegung eines Flügels (links), Flügelstellung bei der Schuberzeugung (rechts), [17]

Abbildung 1.2: VSP (links), VSP am Schiffsrumpf (rechts) [14]

Seit über 75 Jahren werden VSP's für Spezialschiffe verwendet, die hohe Anforderungen an Sicherheit und Manövrierfähigkeit stellen. VSP's sind heute bei Schleppfahrzeugen, Doppelendfähren, Minenbekämpfungsschiffen, Fahrgastschiffen, Tonnenlegern und Schwimmkränen im Einsatz, [61].

Die Funktionalitäten des VSP können auch auf der Homepage von Voith interaktiv getestet werden, [15].

1.3 Kapitelübersicht

Kapitel 2 soll als Grundlage für die Arbeit dienen und dem Leser ein besseres Verständnis ermöglichen. Dabei werden grundlegende Begriffe erläutert, Kriterien für die Gitterqualität von Tetraeder- und Hexaedernetzen angegeben und Möglichkeiten der Verbesserung der Netzqualität aufgezeigt. Zusätzlich werden Vernetzungsverfahren, welche in den Vernetzungszyklus aufgenommen wurden, erklärt. Dieses Kapitel beinhaltet zudem eine Beschreibung existierender Verfahren zur Generierung von Hexaedernetzen.
Im Kapitel 3 wird die Theorie zur benötigten Strömungsmechanik herausgearbeitet. Dabei wird im speziellen der Zusammenhang zwischen den Ergebnissen der Strömungssimulation mit der Finiten-Volumen Methode und dem zugrundeliegenden Gitter untersucht.
Der aus dieser Arbeit entstandene Vernetzer **vmesh** wird detailliert in Kapitel 4 vorgestellt. Die Beschreibung des entwickelten automatischen Netzgenerierers beinhaltet zum einen die neu entstandenen Algorithmen und das entworfene Vernetzungskonzept und zum anderen die Bedienung des Programms. Unter anderem wurde ein Offset-Algorithmus entwickelt, der es ermögicht aus einem bestehenden Oberflächennetz aus Vierecken ein Volumennetzetz aus Hexaedern für die Grenzschicht der zu vernetzenden Geometrie zu generieren.
Kapitel 5 beinhaltet zwei Methoden der numerischen Optimierung. Zum einen die geometriebasierte Optimierung und zum anderen die in **vmesh** realisierte netzbasierte Formoptimierung.
In Kapitel 6 werden die von **vmesh** generierten Netze für verschiedene Geometrien gezeigt und die erzielten Ergebnisse aus den Vergleichsrechnungen und der Optimierung veranschaulicht.
Ein Ausblick ist in Kapitel 7 zu finden.

Kapitel 2

Gittergenerierung

Durch die stetig ansteigende Verfügbarkeit von Computerkapazitäten, die es erlauben, komplexe physikalische oder chemische Vorgänge auf der Grundlage kontinuumstheoretischer Bilanzgleichungen (Euler-, Navier-Stokes-Gleichungen, usw.) zu erfassen, erlangte die automatische Generierung von Rechengittern in den letzten Jahren eine immer wichtigere Bedeutung, [12].

Im ersten Teil des Kapitels wird eine Einführung in die Gittergenerierung gegeben, wobei wichtige Begriffe erklärt, Gitterkriterien für Hexaeder- und Tetraedernetze dargestellt und Optionen zur Verbesserung der Gitterqualität aufgezeigt werden. Der anschließende Abschnitt beinhaltet eine detaillierte Beschreibung verwendeter Vernetzungsalgorithmen, z.B. des Paving-Algorithmus oder Algorithmen zur Tetraedervernetzung, welche auf der Delaunay-Triangulierung basieren. Anschließend werden bereits existierende Methoden zur Hexaedergenerierung erläutert und auf deren Automatisierbarkeit untersucht.

2.1 Einführung in die Gittergenerierung

Die Einführung beinhaltet eine Erläuterung wichtiger Begriffe aus der Gittergenerierung. Dabei werden verschiedene Gitterarten und Elementtypen vorgestellt, ihre Besonderheiten aufgezeigt, Gitterkriterien für Tetraeder- und Hexaedernetze dargestellt und Wege zur Verbesserung der Netzqualität beschrieben.
Eine Unterscheidung der Begriffe Gitter und Netz findet in dieser Dissertation nicht statt.
An dieser Stelle wird noch auf eine sehr interessante, deutschsprachige Internetseite zur Gittergenerierung hingewiesen. Es handelt sich um die Homepage von Robert Schneiders, [10].

2.1.1 Rechengitter

Um eine partielle Differentialgleichung numerisch lösen zu können, muss zuerst das Berechnungsgebiet in Teilgebiete unterteilt werden. Diesen Prozess bezeichnet man als

Diskretisierung. Dabei wird die unendliche Anzahl von Freiheitsgraden der kontinuierlichen Gleichung durch eine endliche ersetzt. Diese kann anschließend von einem Rechner bearbeitet werden. Eine feinere Unterteilung des Berechnungsgebiets führt dazu, dass sich die Approximationsfehler auf den Untergebieten nicht so stark auswirken. In der numerischen Mathematik wird eine diskrete Zerlegung des Raumes auch als Rechengitter bezeichnet. Die partielle Differentialgleichungen, die auf einem solchen Gitter gelöst werden, sind in unserem Falle die Navier-Stokes-Gleichungen (siehe Kapitel 3). Als Knoten wird der Schnittpunkt zweier Gitterlinien bezeichnet. Die Zellen werden in Finite-Elemente-Verfahren auch als Elemente und in Finite-Volumen-Verfahren als Volumen bezeichnet. Für eine Strömungssimulation müssen an den Rändern des Gitters Randbedingungen vorgegeben werden. Solche Gitter können räumlich invariant sein oder sich zeitlich verändern, [5] [13].

2.1.2 Strukturierte und unstrukturierte Netze

Rechengitter lassen sich in strukturierte und unstrukturierte Netze unterteilen.
Ein strukturiertes Netz wird durch die folgende Definition beschrieben.

Definition 2.1.1: *Ein Elementnetz wird als strukturiert bezeichnet, wenn es an allen inneren Knoten des Netzes regulär ist, [66].*

Reguläre Knoten werden dabei wie folgt definiert:

Definition 2.1.2: *Ein Knoten wird als regulär bezeichnet, wenn die Anzahl der an ihn angrenzenden Elemente die optimale Anzahl ist. Die optimale Anzahl von Elementen je Knoten ist dabei für ein Dreieck* 6*, für ein Viereck* 4*, für Tetraeder* 24 *und für Hexaeder* 8*, [66].*

Bei strukturierten Netzen ist die Anzahl der an einen Knoten angrenzenden Elemente für jeden inneren Knoten gleich und entsprechend der optimalen Anzahl.
Ihr Vorteil ist sicher, dass auf die Gittereigenschaften mehr Einfluß genommen werden kann. Darüber hinaus sind die numerischen Simulationsergebnisse oft genauer als bei unstrukturierten Netzen. Ein großer Nachteil von strukturierten Netzen besteht darin, dass diese nur in geometrisch einfachen Lösungsgebieten einsetzbar sind, [33]. Darüber hinaus lassen sich solche Netze oft nicht automatisch generieren, und schon gar nicht, wenn die zu vernetzende Geometrie komplex ist.
Strukturierte Netze können weiter unterteilt werden in reguläre, orthogonale und krummlinige Netze, [13]. Reguläre Netze sind Netze, bei denen die Zellenlängen in die jeweiligen Raumrichtungen immer konstant sind. Bei orthogonalen Netzen stehen die Gitterlinien immer senkrecht aufeinander und krummlinige Netze zeichnen sich dadurch aus, dass die Gitterlinien beliebigen Raumkurven folgen können.
Schwieriger sind unstrukturierte Netze zu generieren. Diese weisen demgegenüber aber eine höhere Flexibilität (gerade auch für sehr komplexe Geometrien) auf. Solche Gitter lassen sich auch automatisch mit bereits existierenden Algorithmen generieren, [33]. Bei einem unstrukturierten Netz ist die Anzahl der Nachbarn einer Zelle variabel. Somit ist es nicht mehr möglich, eine bestimmte Zelle durch zwei bzw. drei Indizes (2D

bzw. 3D) zu identifizieren. Außerdem sind die Knoten, die eine Zelle bilden nicht a priori gegeben sondern müssen definiert werden. Ein unstrukturiertes Netz muss zum einen die Informationen über sämtliche Knoten und deren Koordinaten beinhalten und zum anderen die Information über sämtliche Zellen und die Information, welche Knoten jeweils Bestandteil dieser Zelle sind. Diese Netze werden vor allem bei der Finite-Elemente-Methode verwendet. Auch für die Finite-Volumen-Methode werden teilweise unstrukturierte Gitter gebraucht, allerdings verbunden mit dem Nachteil von langsamerer Rechengeschwindigkeit, [13].

2.1.3 Hybride Gitter

Definition 2.1.3: *Ein Gitter heißt hybrid, wenn in dem Gitter strukturierte und unstrukturierte Gitter vorkommen, [71].*

Dabei werden die guten Eigenschaften der beiden Gittertypen kombiniert. Dies sind bei strukturierten Gittern die bessere Handhabung von Gittereigenschaften und die genaueren numerischen Simulationsergebnisse. Unstrukturierte Gitter weisen dagegen eine höhere Flexibilität bei komplexen Geometrien auf.

Definition 2.1.4: *Ein Gitter heißt gemischt, wenn unterschiedliche Elementtypen in einem Gitter vorkommen, [71].*

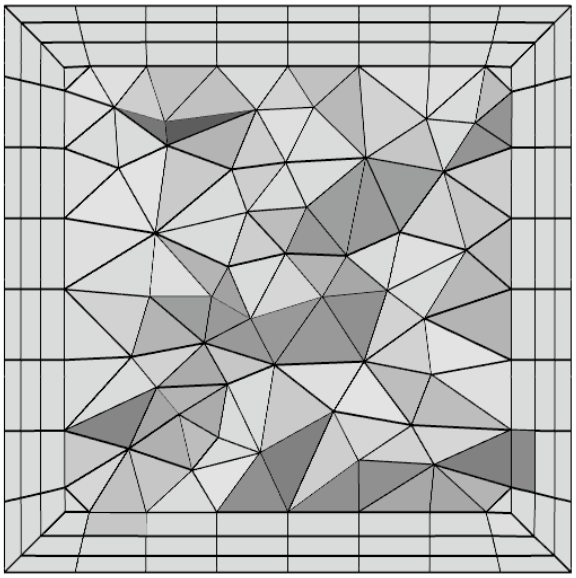

Abbildung 2.1: Schnitt durch ein hybrides Gitter, welches gleichzeitig gemischt ist [71]

Bei der automatischen Gittergenerierung in Kapitel 4 wird auch ein hybrides Volumennetz für den Strömungsraum erzeugt.

2.1.4 Elementtypen

Bei der Oberflächenvernetzung werden hauptsächlich Dreiecke und Vierecke verwendet. Dreiecks- und Tetraedernetze werden üblicher Weise mit der Delaunay-Triangulierung (Abschnitt 2.3) erzeugt. Diese Netze sind für gewöhnlich unstrukturiert. Bei der Generierung von Volumennetzen werden Hexaedernetze häufiger verwendet als Tetraedernetze, da diese für gewöhnlich bei den Simulationen bessere Ergebnisse liefern. Pyramiden dienen häufig als Übergang von Hexaeder- zu Tetraedernetzen. Somit entstehen beim Übergang keine hängenden Knoten. Hängende Knoten sind Gitterpunkte, die nicht für alle angrenzenden Elemente Eckpunkte sind.

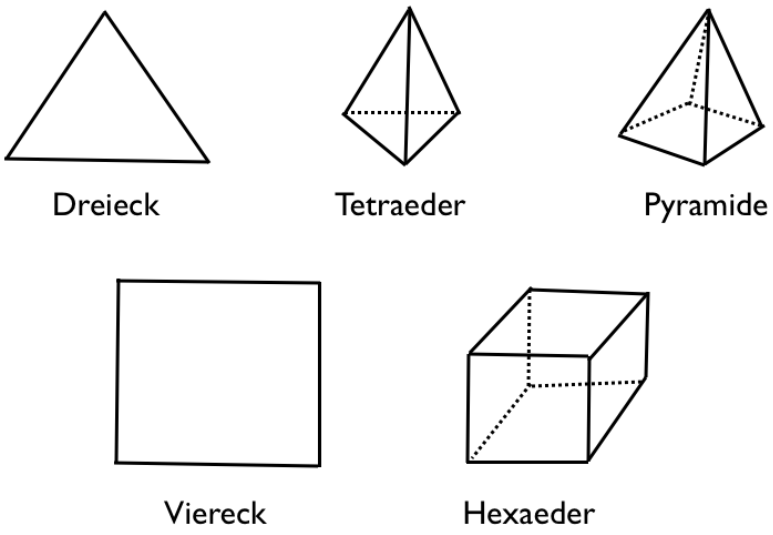

Abbildung 2.2: Verschiedene Elementtypen

2.1.5 Gitterqualität

In [42] definiert Knupp den Begriff der Gitterqualität für Simulationen, die auf der Lösung von partiellen Differentialgleichungen beruhen. Dort heißt es:

"Mesh Quality concerns the characteristics of a mesh that permit a particular numerical PDE simulation to be efficiently performed, with fidelity to the underlaying physics, and with the accuracy required for the problem."

Das Gitter muss gemeinsam mit der ausgewählten Diskretisierungsmethode garantieren, dass sich der Fehler, der durch die Problemdiskretisierung entstanden ist, in einem annehmbaren Bereich befindet, [71].
Im Folgenden werden für Tetraeder- und Hexaedernetze verschiedene und unkomplizierte Gitterkriterien definiert, [64]. Auf diese Kriterien lassen sich die von **vmesh** generierten Netze dann auch untersuchen.

Kriterien für Tetraeder:

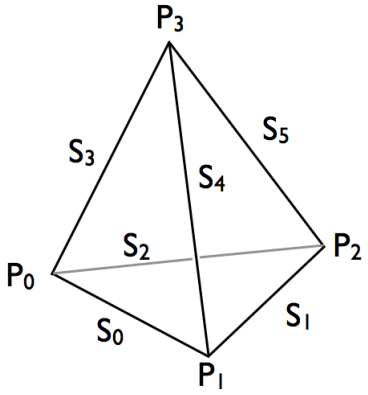

Abbildung 2.3: Tetraeder

Es gilt nach Abbildung 2.3:

$$\begin{aligned} \vec{S}_0 &= \vec{P}_1 - \vec{P}_0 & \vec{S}_3 &= \vec{P}_3 - \vec{P}_0 \\ \vec{S}_1 &= \vec{P}_2 - \vec{P}_1 & \vec{S}_4 &= \vec{P}_3 - \vec{P}_1 \\ \vec{S}_2 &= \vec{P}_0 - \vec{P}_2 & \vec{S}_5 &= \vec{P}_3 - \vec{P}_2 \end{aligned}$$

Dann gilt für die Seitenlängen:

$$S_0 = \|\vec{S}_0\|,\ S_1 = \|\vec{S}_1\|,\ S_2 = \|\vec{S}_2\|,\ S_3 = \|\vec{S}_3\|,\ S_4 = \|\vec{S}_4\|,\ S_5 = \|\vec{S}_5\|$$

Dann ist:

$$S_{min} = \min\{S_0, S_1, S_2, S_3, S_4, S_5\},\ S_{max} = \max\{S_0, S_1, S_2, S_3, S_4, S_5\}$$

Für das Volumen des Tetraeders ergibt sich:

$$V = \frac{(\vec{S}_2 \times \vec{S}_0) \cdot \vec{S}_3}{6}$$

Für die Oberfläche des Tetraeders gilt dann:

$$A = \frac{1}{2}\left(\|\vec{S}_2 \times \vec{S}_0\| + \|\vec{S}_3 \times \vec{S}_0\| + \|\vec{S}_4 \times \vec{S}_1\| + \|\vec{S}_3 \times \vec{S}_2\|\right)$$

Daraus ergibt sich der Inkugelradius r:

$$r = \frac{3V}{A}$$

Im Folgenden werden die wichtigsten Kriterien für Tetraedernetze definiert.

Kantenverhältnis:

Definition 2.1.5: *Als Kantenverhältnis [64] (engl.: Edge Ratio) definiert man bei Tetraedern den Quotient*

$$\frac{S_{max}}{S_{min}}.$$

Ein akzeptables Intervall für diese Größe ist $[1, 3]$.

Aspektverhältnis:

Definition 2.1.6: *Als Aspektverhältnis [64] (engl.: Aspect Ratio) definiert man bei Tetraedern das Verhältnis*

$$\frac{S_{max}}{2\sqrt{6}r}.$$

Ein akzeptables Intervall für diese Größe ist ebenfalls $[1, 3]$.

Jacobi-Determinante:

Definition 2.1.7: *Als Jacobi-Determinante (engl.: Jacobian) J eines Tetraeders wird der Ausdruck*

$$J = (\vec{S}_2 \times \vec{S}_0) \cdot \vec{S}_3$$

definiert, [64].

Mindestwinkel:

Der Winkel (in Grad) zwischen zwei benachbarten Flächen eines Tetraeders mit gemeinsamer Kante i $(0 \leq i \leq 5)$ wird berechnet mit

$$\alpha_i = \frac{180^\circ}{\pi} \arccos(\vec{n}_{i1} \cdot \vec{n}_{i2}),$$

wobei $\vec{n}_{i1}$ und $\vec{n}_{i2}$ die Normalenvektoren der jeweils betrachteten Tetraederflächen sind.

Definition 2.1.8: *Als Mindestwinkel (engl.: Minimum Angle) α_{min} eines Tetraeders wird dann*

$$\alpha_{min} = \min_{i \in \{0,1,2,3,4,5\}} \alpha_i$$

definiert, [64].

Dieser Winkel sollte im Bereich von $[40°, \frac{180°}{\pi} \arccos \frac{1}{3}]$ liegen.

Skalierte Jacobi-Determinate:

Zuerst definieren wir

$$\begin{aligned} \lambda_1 &= \|\vec{S}_0\|\|\vec{S}_2\|\|\vec{S}_3\| \qquad \lambda_2 = \|\vec{S}_0\|\|\vec{S}_1\|\|\vec{S}_4\| \\ \lambda_3 &= \|\vec{S}_1\|\|\vec{S}_2\|\|\vec{S}_5\| \qquad \lambda_4 = \|\vec{S}_3\|\|\vec{S}_4\|\|\vec{S}_5\| \\ \lambda_{max} &= \max \lambda_1, \lambda_2, \lambda_3, \lambda_4, J \end{aligned}$$

Definition 2.1.9: *Als skalierte Jacobi-Determinante (engl.: Scaled Jacobian) J eines Tetraeders wird der Ausdruck*

$$J_s = \frac{J\sqrt{2}}{\lambda_{max}}$$

definiert, [64].

Diese Größe sollte im Intervall $[\frac{1}{2}, \frac{\sqrt{2}}{2}]$ liegen.

Shape:

Definition 2.1.10: *Als Shape q eines Tetraeders wird der Ausdruck*

$$q = \frac{3\left(J\sqrt{2}\right)^{\frac{2}{3}}}{\frac{3}{2}\left(\vec{S}_0 \cdot \vec{S}_0 + \vec{S}_2 \cdot \vec{S}_2 + \vec{S}_3 \cdot \vec{S}_3\right) - \left(\vec{S}_0 \cdot (-\vec{S}_2) + \vec{S}_0 \cdot \vec{S}_3 + (-\vec{S}_2) \cdot \vec{S}_3\right)}$$

definiert, [64].

Ein akzeptables Intervall für diese Größe ist $[0.3, 1]$.

Kriterien für Hexaeder:

Hier gilt nach Abbildung 2.4:

$$\begin{array}{lll} \vec{S}_0 = \vec{P}_1 - \vec{P}_0 & \vec{S}_4 = \vec{P}_4 - \vec{P}_0 & \vec{S}_8 = \vec{P}_5 - \vec{P}_4 \\ \vec{S}_1 = \vec{P}_2 - \vec{P}_1 & \vec{S}_5 = \vec{P}_5 - \vec{P}_1 & \vec{S}_9 = \vec{P}_6 - \vec{P}_5 \\ \vec{S}_2 = \vec{P}_3 - \vec{P}_2 & \vec{S}_6 = \vec{P}_6 - \vec{P}_2 & \vec{S}_{10} = \vec{P}_7 - \vec{P}_6 \\ \vec{S}_3 = \vec{P}_3 - \vec{P}_0 & \vec{S}_7 = \vec{P}_7 - \vec{P}_3 & \vec{S}_{11} = \vec{P}_7 - \vec{P}_4 \end{array}$$

Weiter ist

$$S_i = \|\vec{S}_i\|, \quad \forall\, i \in \{0, ..., 11\}$$

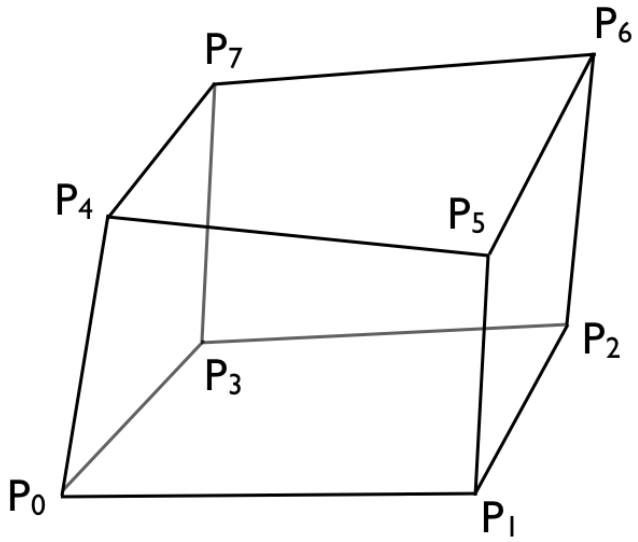

Abbildung 2.4: Hexaeder

und

$$S_{min} = \min\{S_0, ..., S_{11}\},\ S_{max} = \max\{S_0, ..., S_{11}\}, \quad \forall\ i \in \{0, ..., 11\}.$$

Für die vier Diagonalen des Hexaeders gilt:

$$\begin{array}{ll} \vec{D}_0 = \vec{P}_6 - \vec{P}_0 & \vec{D}_2 = \vec{P}_4 - \vec{P}_2 \\ \vec{D}_1 = \vec{P}_7 - \vec{P}_1 & \vec{D}_3 = \vec{P}_5 - \vec{P}_3 \end{array}$$

$$\begin{aligned} D_{min} &= \min\{\|D_0\|, \|D_1\|, \|D_2\|, \|D_3\|\} \\ D_{max} &= \max\{\|D_0\|, \|D_1\|, \|D_2\|, \|D_3\|\} \end{aligned}$$

Die Hauptachsen des Hexaeders sind:

$$\begin{aligned} \vec{Y}_0 &= \left(\vec{P}_1 - \vec{P}_0\right) + \left(\vec{P}_2 - \vec{P}_3\right) + \left(\vec{P}_5 - \vec{P}_4\right) + \left(\vec{P}_6 - \vec{P}_1\right) \\ \vec{Y}_1 &= \left(\vec{P}_3 - \vec{P}_0\right) + \left(\vec{P}_2 - \vec{P}_1\right) + \left(\vec{P}_7 - \vec{P}_4\right) + \left(\vec{P}_6 - \vec{P}_5\right) \\ \vec{Y}_2 &= \left(\vec{P}_4 - \vec{P}_0\right) + \left(\vec{P}_5 - \vec{P}_1\right) + \left(\vec{P}_6 - \vec{P}_2\right) + \left(\vec{P}_7 - \vec{P}_3\right) \end{aligned}$$

Es werden neun Jacobi-Matrizen mit den Hexaederkanten definiert:

$$\begin{array}{lll} A_0 = \left(\vec{S}_0,\ \vec{S}_3,\ \vec{S}_4\right) & & A_5 = \left(-\vec{S}_8,\ \vec{S}_9,\ -\vec{S}_5\right) \\ A_1 = \left(\vec{S}_1,\ -\vec{S}_0,\ \vec{S}_5\right) & & A_6 = \left(-\vec{S}_9,\ \vec{S}_{10},\ -\vec{S}_6\right) \\ A_2 = \left(\vec{S}_2,\ -\vec{S}_1,\ \vec{S}_6\right) & & A_7 = \left(-\vec{S}_{10},\ -\vec{S}_{11},\ -\vec{S}_7\right) \\ A_3 = \left(-\vec{S}_3,\ -\vec{S}_2,\ \vec{S}_7\right) & & A_8 = \left(\vec{Y}_1,\ \vec{Y}_2,\ \vec{Y}_3\right) \\ A_4 = \left(\vec{S}_{11},\ \vec{S}_8,\ -\vec{S}_4\right) & & \end{array}$$

Dann sei $\alpha_i := \det(A_i)$ für alle $i \in \{0, ..., 8\}$.

Sei mit A eine der obigen Matrizen gemeint und $\vec{v}_1$, $\vec{v}_2$ und $\vec{v}_3$ seien die Spaltenvektoren dazu. Dann gilt $A = [\vec{v}_1, \vec{v}_2, \vec{v}_3]$. Wir definieren

$$\hat{A} := \left(\frac{\vec{v}_1}{\|\vec{v}_1\|}, \frac{\vec{v}_2}{\|\vec{v}_2\|}, \frac{\vec{v}_3}{\|\vec{v}_3\|} \right).$$

Weiter sei

$$\hat{\alpha}_i := \det \hat{A}_i.$$

Im Folgenden werden die wichtigsten Kriterien für Hexaedernetze definiert.

Kantenverhältnis:

Definition 2.1.11: *Als Kantenverhältnis [64] (engl.: Edge Ratio) definiert man bei Hexaedern den Quotient*

$$\frac{S_{max}}{S_{min}}.$$

Diagonalenverhältnis:

Definition 2.1.12: *Als Diagonalenverhältnis [64] (engl.: Diagonal) definiert man bei Hexaedern den Quotient*

$$\frac{D_{max}}{D_{min}}.$$

Diese Größe sollte im Intervall $[0.65, 1]$ liegen.

Jacobi-Determinante:

Definition 2.1.13: *Als Jacobi-Determinante (engl.: Jacobian) J eines Hexaeders wird der Ausdruck*

$$J = \min \left\{ \{\alpha_i\}_{i=0}^{7}, \frac{\alpha_8}{64} \right\}$$

definiert, [64].

Shape:

Definition 2.1.14: *Als Shape q eines Hexaeders wird der Ausdruck*

$$q = 3 \min_{i \in 0,1,...,8} \left\{ \frac{\alpha_i^{\frac{3}{2}}}{|A_i^2|} \right\}$$

definiert, [64].

Ein akzeptables Intervall für diese Größe ist $[0.3, 1]$.

Scherung:

Definition 2.1.15: *Als Scherung (engl.: Shear) q eines Hexaeders wird der Ausdruck*

$$q = \min_{i \in 0,1,\ldots,8} \{\hat{\alpha}_i\}$$

definiert, [64].

Ein akzeptables Intervall für diese Größe ist ebenfalls $[0.3, 1]$.

Mit all diesen Kriterien kann die Qualität der erzeugten Netze untersucht werden.

2.1.6 Verbesserung der Gitterqualität

Es gibt mehrere Möglichkeiten die Netzqualität zu verbessern. Einige davon sollen im Folgenden aufgezeigt werden.

Netzoptimierung [66]:

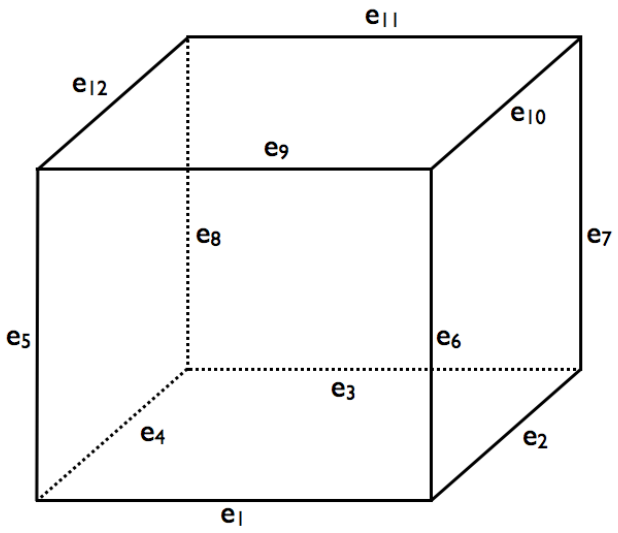

Abbildung 2.5: Beispielzelle

Bei dieser Methode wird ein Funktional optimiert, welches gewichtet zusammengesetzt sein soll aus verschiedenen Kriterien, die für wichtig erachtet werden.
Um zum Beispiel die Orthogonalität einer Zelle mit einfließen zu lassen betrachten wir das Orhogonalitätskriterium:

$$\sigma^c_{ortho} = \sum_{\substack{1\leq i<j\leq 12,\\ e_i\cap e_j\neq\emptyset}} (e_i \cdot e_j)^2 \tag{2.1.1}$$

Für die Regularität würde man folgendes Kriterium auswählen:

$$\sigma^c_{reg} = \sum_{1\leq i\leq 12} \left[\frac{|e_i|}{\sum\limits_{1\leq j\leq 12} |e_j|} \right]^2 \tag{2.1.2}$$

Das Funktional hat dann die Gestalt:

$$\sigma = \sum_{c \in Mesh} \left(\lambda \sigma^c_{ortho} + (1 - \lambda) \sigma^c_{reg} \right); \quad \lambda \in [0, 1] \tag{2.1.3}$$

Kriterien für das Gitter, die speziell bei der Finiten-Volumen Methode sinnvoll sind, werden in Kapitel 3.3 untersucht.

Netzglättung:

Einer der einfachsten und zugleich gängigsten Glättungsmethoden ist die Laplace-Glättung. Bei diesem Algorithmus wird jede ursprüngliche Punktposition durch eine neue Punktposition p_i des Punktes i ersetzt, der sich wie folgt jeweils durch den Mittelwert seiner Nachbarpunkte berechnen läßt.

$$p_i := \frac{1}{|N(i)|} \sum_{j \in N(i)} q_j \tag{2.1.4}$$

Dabei wird $N(i)$ die Menge der Nachbarpunkte und mit q_j die Nachbarpunkte des Punktes i bezeichnet. Ausgehend von dieser Idee gibt es natürlich eine Vielzahl von Verbesserungen und Erweiterungen des Laplace-Glättungsalgorithmus, [22].

Es werden auch andere Glättungsmethoden verwendet, z.B. die Equipotential-Glättung, Glättungsmethoden, die z.B. das Aspekten- oder Kantenverhältnis einer Zelle verbessern. Auf diese soll aber nicht näher eingegangen werden.

2.2 Paving-Algorithmus: Vernetzung von dreidimensionalen Oberflächen mit Vierecken

Der Paving-Algorithmus ist ein Vernetzungsalgorithmus zur Generierung von Vierecken auf dreidimensionalen Oberflächen. Ein solcher Algorithmus kommt auch bei der Generierung des Oberflächennetzes der zu vernetzenden Schiffgeometrien zum Einsatz (Kapitel 4). Dazu wurde der Paving-Algorithmus verwendet, welcher Bestandteil des Vernetzungstools CUBIT (Abschnitt 2.5) ist.
Dieser Abschnitt beruht hauptsächlich auf dem Paving-Algorithmus von Blacker und White, [19][54].

2.2.1 Einführung

Seit die automatisierte Vernetzung auf großes Interesse gestoßen ist, gibt es ausgedehnte Forschungen auf diesem Gebiet. Der Großteil der Forschung spezialisierte sich dabei auf die 2D Quadtree und 3D Octree Triangulierung. Im Allgemeinen generieren diese Methoden Dreieckselemente. Viele kommerzielle Tools verwenden zur Viereckserzeugung die Mapping-Methode ([25], [34]). Diese Methode läuft jedoch nicht automatisiert ab. Der Nutzer muss das zu vernetzende Gebiet zuerst in Regionen zerlegen. Obwohl die Mapping-Methode mühsam und nur mit viel Sachkompetenz anwendbar ist, ziehen sie viele Nutzer den anderen Methoden aus folgenden Gründen vor:

- Die Netzlinien folgen den Konturen der Geometrie.
- Die Netztopologie ändert sich nicht, wenn die zu vernetzende Geometrie zuvor rotiert oder verschoben wird.
- Wenige innere Knoten sind mit mehr oder weniger als vier Elementen verbunden.

Die Paving-Methode generiert vollautomatisch ein Netz, welches komplett aus Vierecken besteht, mit ähnlicher Qualität wie es die Mapping-Methode liefert, jedoch ohne dabei die zu vernetzende Geometrie zu unterteilen. Der Paving-Algorithmus pflastert die Geometrie mit Reihen von Viereckselementen vom Rand der Geometrie zu ihrem Inneren zu. Diese Methode ist relativ schnell und arbeitet sehr robust, [19].

2.2.2 Konventionen und Definitionen

Die Paving-Methode basiert auf iterativem Pflastern von Reihen mit Vierecken vom Rand zum Inneren des Gebiets. Wie in Abbildung 2.6 zu sehen, füllen diese Reihen das zu vernetzende Gebiet komplett innerhalb aus. Wenn Reihen beginnen sich im Inneren der Geometrie zu überlappen oder zusammenfallen, werden sie so miteinander verbunden, dass ein gültiges Netz aus Vierecken entsteht.
Die Paving-Methode beginnt mit dem Einlesen von einer oder mehreren geschlossenen Schleifen von Knoten, welche auf dem Rand der zu vernetzenden Geometrie liegen, wie in Abbildung 2.7 zu sehen ist.

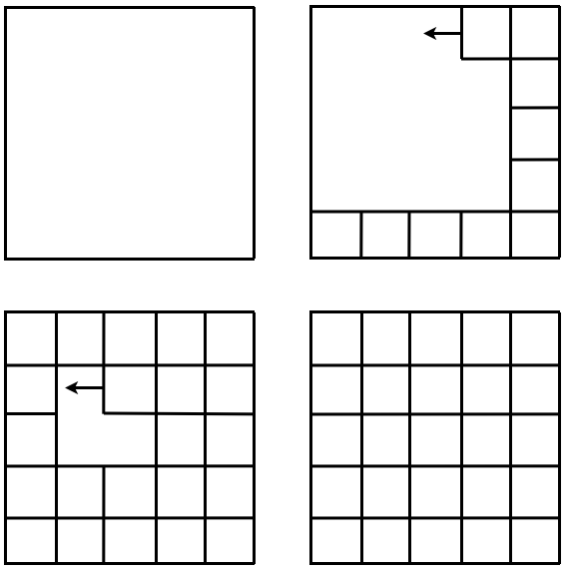

Abbildung 2.6: Einfaches Beispiel zur Paving-Methode

Definition 2.2.1: *Diese Startschleifen werden auch als fester Rand bezeichnet (siehe Abbildung 2.7), [19].*

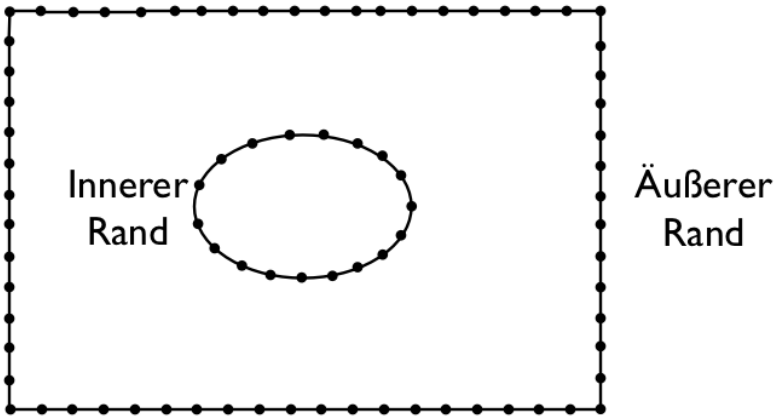

Abbildung 2.7: Beispiel von festen Rändern als Input für die Paving-Methode

Der Zusammenhang und der Ort der Startknoten dürfen während des Vernetzens nicht geändert werden. Dies sichert die Kompatibilität des Netzes zu angrenzenden Regionen. Feste Ränder sind unterteilt in innere und äußere Ränder. Jedoch gibt es nur einen äußeren Rand pro zu vernetzendem Gebiet. Diese Schleife darf sich nicht schneiden und muss das Gebiet komplett umschließen. Die Knoten sind dabei gegen den Uhrzeigersinn angeordnet. Innere Ränder definieren Löcher innerhalb des Gebietes. Hier sind die Knoten im Uhrzeigersinn angeordnet. Wenn mehrere innere Ränder existieren, dürfen

diese sich nicht schneiden und müssen jeweils komplett von dem äußeren Rand eingeschlossen sein.

Definition 2.2.2: *Als Paving-Rand werden Knoten bezeichnet, auf denen die Paving-Methode operiert (siehe Abbildung 2.8), [19].*

Anfangs ist jeder feste Rand ein Paving-Rand. Paving-Ränder werden unterteilt in innere und äußere Paving-Ränder. Äußere Paving-Ränder werden gegen den Uhrzeigersinn gepflastert und entwickeln sich von dem äußeren Rand einwärts. Innere Paving-Ränder werden im Uhrzeigersinn gepflastert und entwickeln sich von dem inneren Rand nach außen.

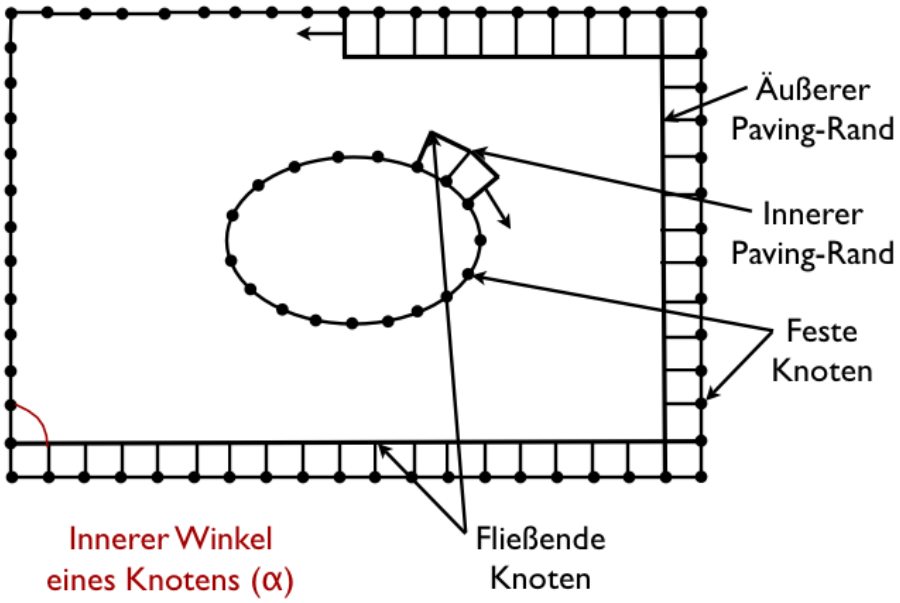

Abbildung 2.8: Beispiel für innere und äußere Ränder bei der Paving-Methode

Definition 2.2.3: *Ein Knoten auf einer Paving-Rand wird als Paving-Knoten bezeichnet, [19].*

Definition 2.2.4: *Ein Knoten auf einem festen Rand wird als fester Knoten bezeichnet, [19].*

Definition 2.2.5: *Ein Knoten der nicht auf einem festen Rand liegt, wird als fließender Knoten bezeichnet, [19].*

Definition 2.2.6: *Jeder Paving-Knoten besitzt einen inneren Winkel, welcher durch die Strecke des Knotens zum vorausgehenden Knoten und der Strecke des Knotens zum nächsten Paving-Knoten beschrieben wird, [19].*

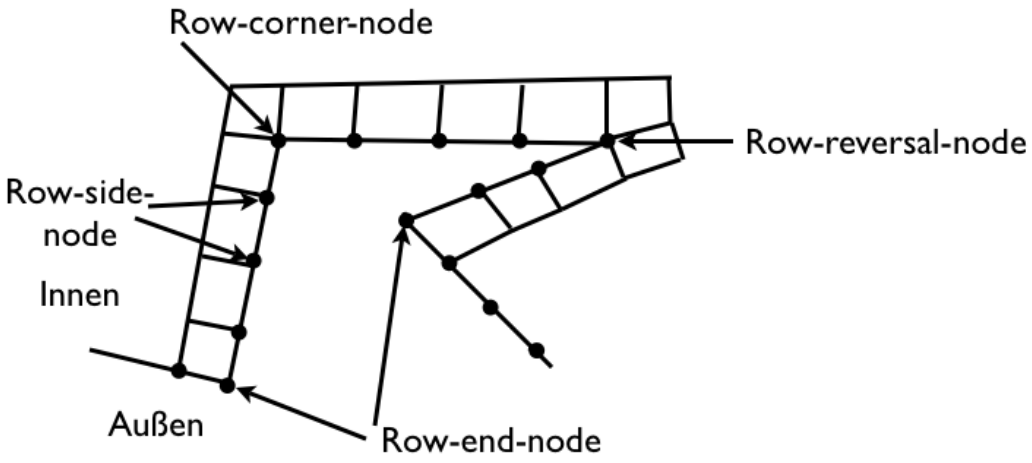

Abbildung 2.9: Beispiel für die Kategorisierung der Knoten

Definition 2.2.7: *Eine Reihe wird definiert durch entsprechende Start- und End-Paving-Knoten, [19].*

Die Paving-Knoten werden kategorisiert über ihren inneren Winkel α und Winkeltoleranzen $\sigma_1, ..., \sigma_6$. Dabei werden folgende Fallunterscheidungen gemacht:

- $\alpha \leq \pi/2 + \sigma_1$: Ein Row-end-node ist ein Knoten bei dem eine Reihe endet. An einem Row-end-node kann nur ein neues Reihenelement generiert werden.
- $\pi/2 + \sigma_1 < \alpha \leq \pi - \sigma_2$: Ein Row-end-node oder ein Row-side-node.
- $\pi - \sigma_2 < \alpha \leq \pi + \sigma_3$: Ein Row-side-node ist ein Knoten bei dem die Reihe normal verläuft. An einem solchen Knoten werden zwei Viereckselemente generiert.
- $\pi + \sigma_3 < \alpha \leq 3\pi/2 - \sigma_4$: Ein Row-side-node oder ein Row-corner-node.
- $3\pi/2 - \sigma_4 < \alpha \leq 3\pi/2 + \sigma_5$: Ein Row-corner-node ist ein Knoten bei dem drei neue Elemente generiert werden.
- $3\pi/2 + \sigma_5 < \alpha \leq 2\pi - \sigma_6$: Ein Row-corner-node oder ein Row-reversal-node. Dieser Knoten hat drei oder vier Elemente.
- $\alpha > 2\pi - \sigma_6$: Ein Row-reversal-node ist ein Knoten bei dem vier neue Elemente generiert werden.

In Abbildung 2.9 und 2.10 wird die Kategorisierung der Knoten und der Winkelstatus nochmal veranschaulicht.

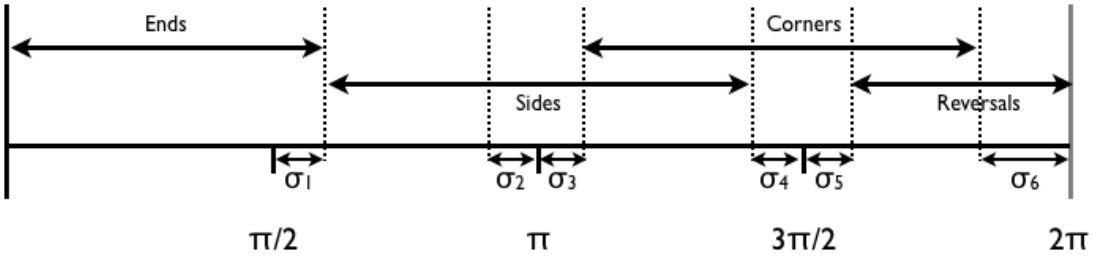

Abbildung 2.10: Übersicht über den Winkelstatus der inneren Winkel

2.2.3 Ablauf des Paving-Algorithmus

Der Ablauf des Paving-Algorithmus ist in Abbildung 2.11 als Flussdiagramm dargestellt.

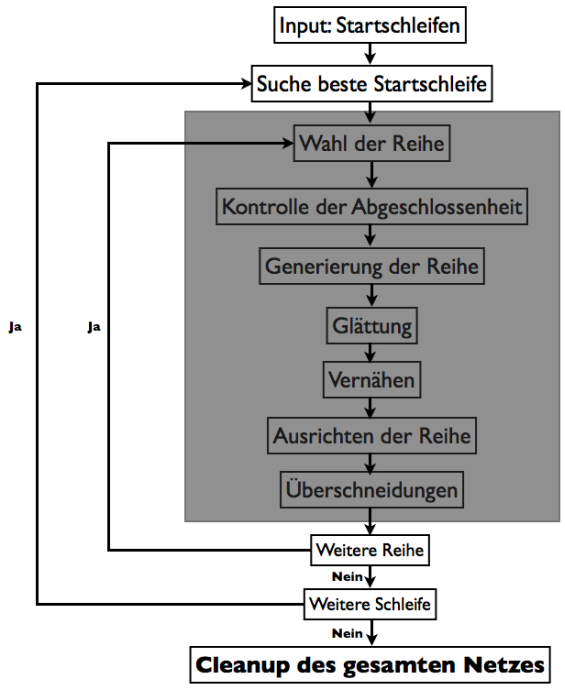

Abbildung 2.11: Ablauf: Paving-Algorithmus

Im Folgenden werden die in Abbildung 2.11 einzeln aufgeführten Schritte beschrieben.

Input: Startschleifen:

Bevor die Vernetzung mit dem Paving-Algorithmus gestartet wird, werden Listen von Knoten, welche Punkte auf der diskretisierten Grenze (Kante) der zu vernetzenden Oberfläche sind, dem Algorithmus übergegeben. Diese Listen beinhalten Netzknoten

einer gegebenen Schleife von Kanten, welche die Oberfläche begrenzen. In Abbildung 2.12 ist die zu vernetzende Oberfläche ein Gebiet mit einem Loch. Hier werden zwei Knotenlisten dem Algorithmus übergeben. Die Knotenlisten werden vorab, durch eine Kurvenvernetzung generiert. Diese Knotenlisten sind sozusagen der Ursprung des inneren Oberflächennetzes, [19].

Suche beste Startschleife:

Der Algorithmus beginnt mit der äußeren Schleife. Anschließend wird der Startknoten berechnet. Gesucht wird der Knoten, bei welchem es am einfachsten ist, ein Viereckselement zu generieren. An diesem Startknoten wird das erste Element generiert. Ausgehend von diesem Element, wird dann die erste Reihe von Elementen entlang der Schleife erzeugt. Durch die neuen Knoten wird eine neue Schleife bestimmt, [54].

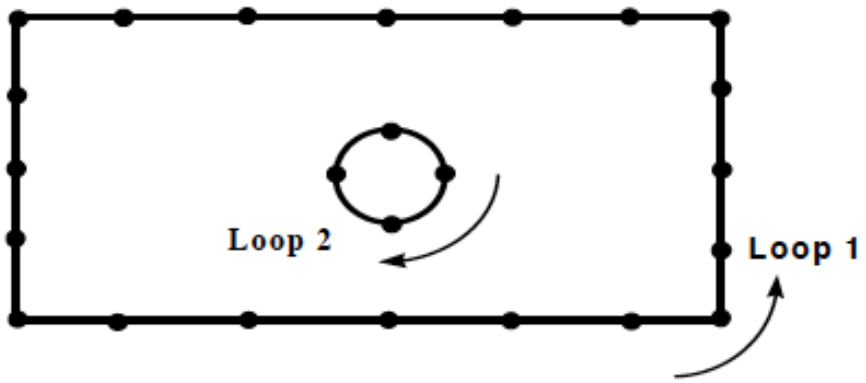

Abbildung 2.12: Beispiel: Startschleifen beim Paving-Algorithmus [54]

Wahl der Reihe:

Der Anfangs- und Endknoten der nächsten Reihe von Elementen, die hinzugefügt werden soll, wird gesucht, [19].

Kontrolle der Abgeschlossenheit:

Es wird kontrolliert, ob mehr als sechs Knoten im Paving-Rand enthalten sind. Spezielle Techniken werden angewendet, wenn dies sechs oder weniger Knoten sind, [19].

Erzeugung von Reihen:

Die nächste Reihe von Elementen wird nach und nach hinzugefügt, [19].

Sobald die Enden einer Reihe bestimmt sind, beginnt die Generierung der Reihenelemente. Dies wird durch Projektion der Randknoten erreicht. Diese Projektion basiert auf der Knotenklassifizierung aus Unterabschnitt 2.2.2. Es werden folgende Fallunterscheidungen gemacht:

Projektion eines Row-side-node (Abb.: 2.13):
Sei N_i ein existierender Knoten. Dann wird der neue Knoten N_j an der Spitze des Vektors $\mathbf{V}$ positioniert, dessen Ursprung in N_i ist. $\mathbf{V}$ ist so gerichtet, dass er den inneren Winkel α von N_i gerade halbiert. Weiter ist $d_1 = \|\overrightarrow{N_{i-1}, N_i}\|$ und $d_2 = \|\overrightarrow{N_i, N_{i+1}}\|$. Dann gilt:

$$|\mathbf{V}| = \frac{(d_1 + d_2)/2}{\sin(\alpha/2)} \tag{2.2.1}$$

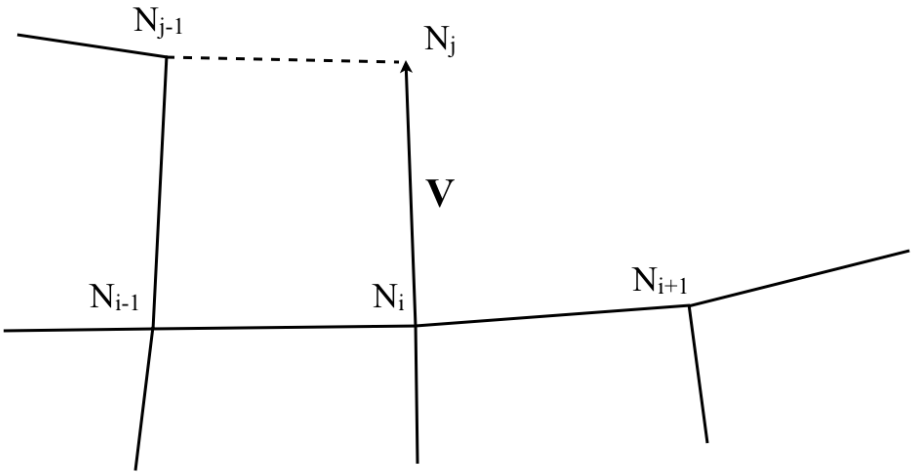

Abbildung 2.13: Projektion eines neuen Knoten von einem Row-side-node

Projektion eines Row-corner-node (Abb.: 2.14):
Auch hier sei N_i ein existierender Knoten. Dann werden drei neue Knoten N_j, N_k und N_l an den Spitzen der entsprechenden Vektoren $\mathbf{V_j}$, $\mathbf{V_k}$ und $\mathbf{V_l}$ positioniert, deren Ursprung in N_i ist. Die drei Vektoren sind dabei gerade so gerichtet, dass sie den inneren Winkel α von N_i im Uhrzeigersinn im Verhältnis 1/3, 1/2 und 2/3 teilen. Auch hier ist wieder $d_1 = \|\overrightarrow{N_{i-1}, N_i}\|$ und $d_2 = \|\overrightarrow{N_i, N_{i+1}}\|$. Dann gilt in diesem Fall:

$$|\mathbf{V_j}| = \frac{(d_1 + d_2)/2}{\sin(\alpha/3)} \tag{2.2.2}$$

$$|\mathbf{V_k}| = \sqrt{2}\,|\mathbf{V_j}| \tag{2.2.3}$$

$$|\mathbf{V_l}| = |\mathbf{V_j}| \tag{2.2.4}$$

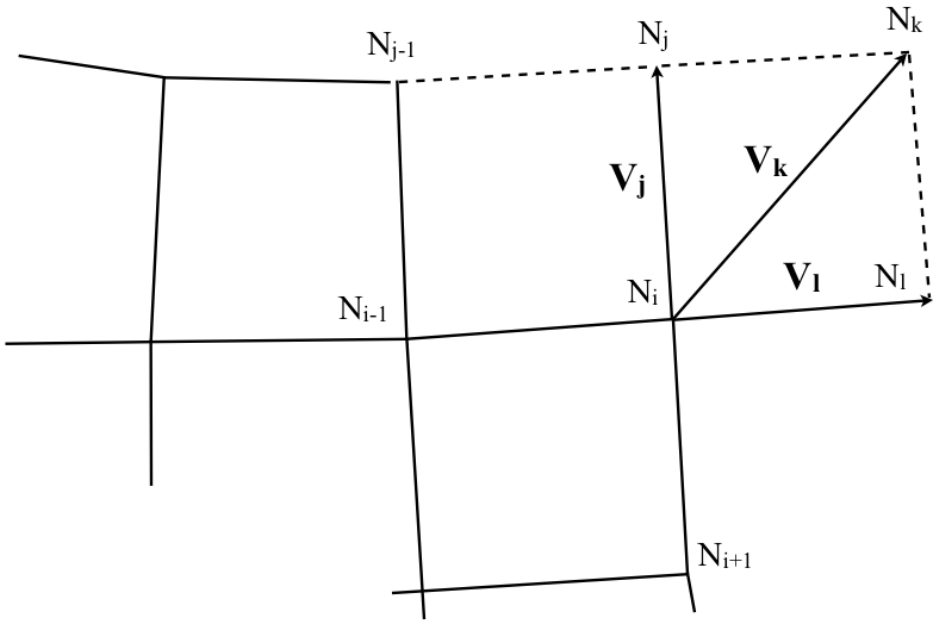

Abbildung 2.14: Projektion eines neuen Knoten von einem Row-corner-node

Projektion eines Row-reversal-node (Abb.: 2.15):
Auch hier sei N_i ein existierender Knoten. Dann werden fünf neue Knoten N_j, N_k, N_l, N_m und N_n an den Spitzen der entsprechenden Vektoren $\mathbf{V_j}$, $\mathbf{V_k}$, $\mathbf{V_l}$, $\mathbf{V_m}$ und $\mathbf{V_n}$ positioniert, deren Ursprung in N_i ist. Die fünf Vektoren sind so gerichtet, dass sie den inneren Winkel α von N_i gerade im Uhrzeigersinn wie folgt 1/4, 3/8, 1/2, 5/8 und 3/4 teilen. Dabei ist wieder $d_1 = \|\overrightarrow{N_{i-1}, N_i}\|$ und $d_2 = \|\overrightarrow{N_i, N_{i+1}}\|$. Dann gilt in diesem Fall:

$$|\mathbf{V_j}| = \frac{(d_1 + d_2)/2}{\sin(\alpha/4)} \tag{2.2.5}$$

$$|\mathbf{V_k}| = \sqrt{2}\,|\mathbf{V_j}| \tag{2.2.6}$$

$$|\mathbf{V_l}| = |\mathbf{V_j}| \tag{2.2.7}$$

$$|\mathbf{V_m}| = |\mathbf{V_k}| \tag{2.2.8}$$

$$|\mathbf{V_n}| = |\mathbf{V_j}| \tag{2.2.9}$$

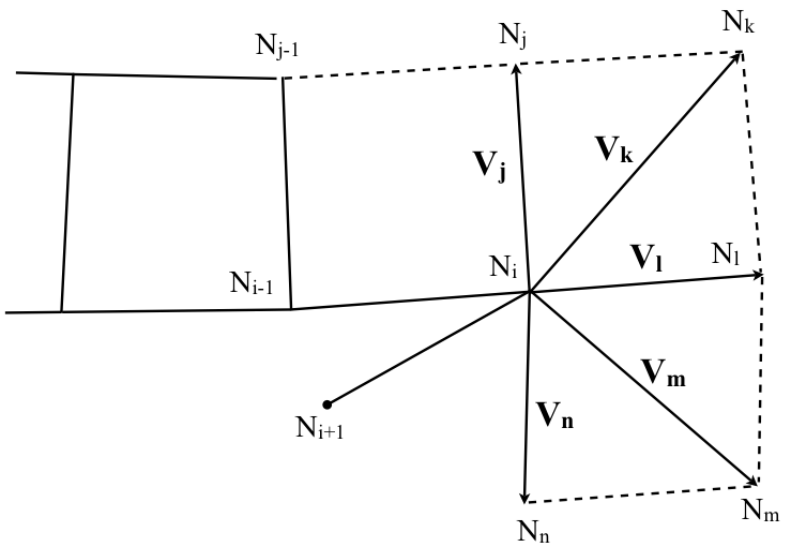

Abbildung 2.15: Projektion eines neuen Knoten von einem Row-reversal-node

Projektion eines All-row-side-node (Abb.: 2.16):
Auch hier sei N_i der beginnende Knoten. Dann werden zwei neue Knoten N_j und N_k an den Spitzen der entsprechenden Vektoren $\mathbf{V_j}$ und $\mathbf{V_k}$ positioniert. $\mathbf{V_j}$ hat seinen Ursprung in N_i. $\mathbf{V_k}$ hat seinen Ursprung in N_{i+1}. $\mathbf{V_j}$ ist so gerichtet, dass er den inneren Winkel α von N_i halbiert. Und $\mathbf{V_k}$ ist so gerichtet, dass er den inneren Winkel β von N_{i+1} halbiert. Auch hier ist wieder $d_1 = \|\overrightarrow{N_{i-1}, N_i}\|$, $d_2 = \|\overrightarrow{N_i, N_{i+1}}\|$ und $d_3 = \|\overrightarrow{N_{i+1}, N_{i+2}}\|$. Dann gilt:

$$|\mathbf{V_j}| = \frac{(d_1 + d_2)/2}{\sin(\alpha/2)} \tag{2.2.10}$$

$$|\mathbf{V_k}| = \frac{(d_2 + d_3)/2}{\sin(\beta/4)} \tag{2.2.11}$$

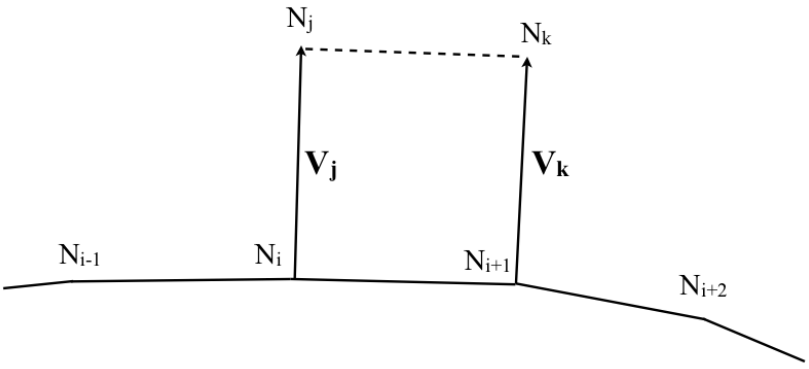

Abbildung 2.16: Projektion eines neuen Knoten von einem All-row-side-node [19]

Glättung:

Fließende Knoten werden ausgerichtet um die Netzqualität und die Glattheit des Randes zu verbessern, [19].

Vernähen:

Kleine Innenwinkel im Paving-Rand werden von angrenzenden gegenüberliegenden Elementen vernäht oder verschlossen (Abb.: 2.17 und 2.18).[19]

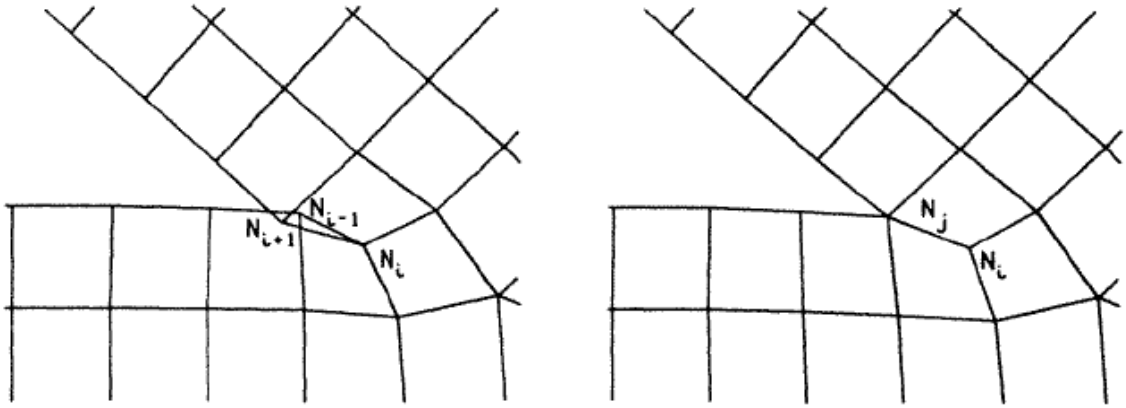

Abbildung 2.17: Beispiel: Verschließen von zu kleinen Innenwinkeln (1) [19]

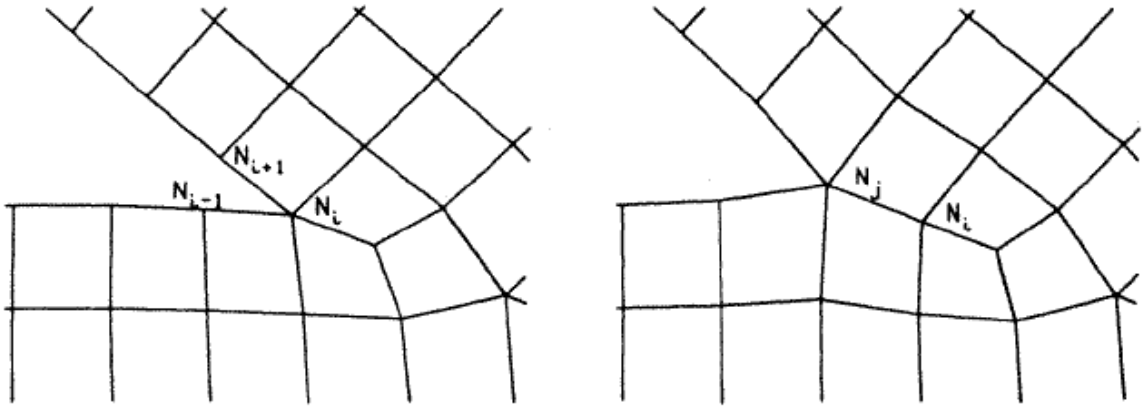

Abbildung 2.18: Beispiel: Verschließen von zu kleinen Innenwinkeln (2) [19]

Ausrichten der Reihen:

Die neue Reihe wird ausgerichtet indem Keile oder Falten eingefügt werden um zu große oder zu kleine Elemente zu vermeiden (Abb.: 2.19), [19].

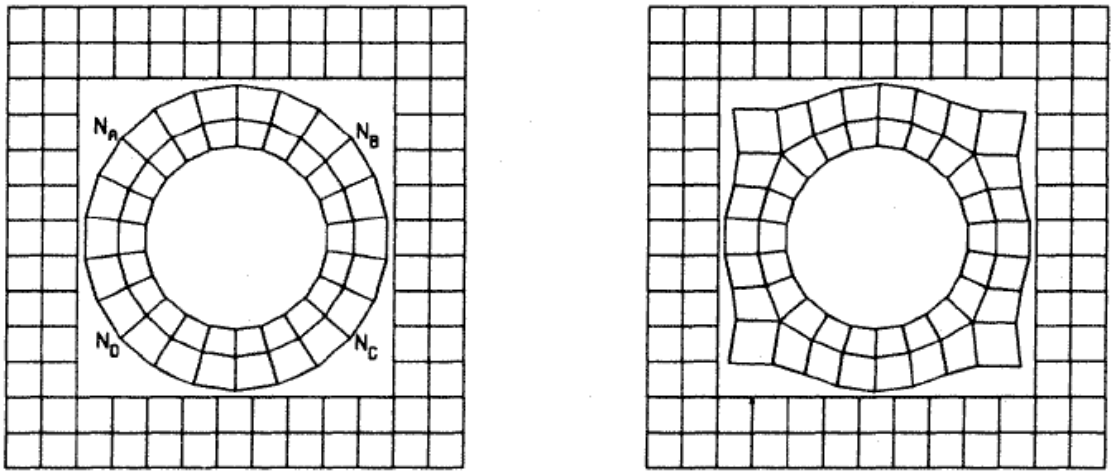

Abbildung 2.19: Beispiel: Einfügen eines Keils [19]

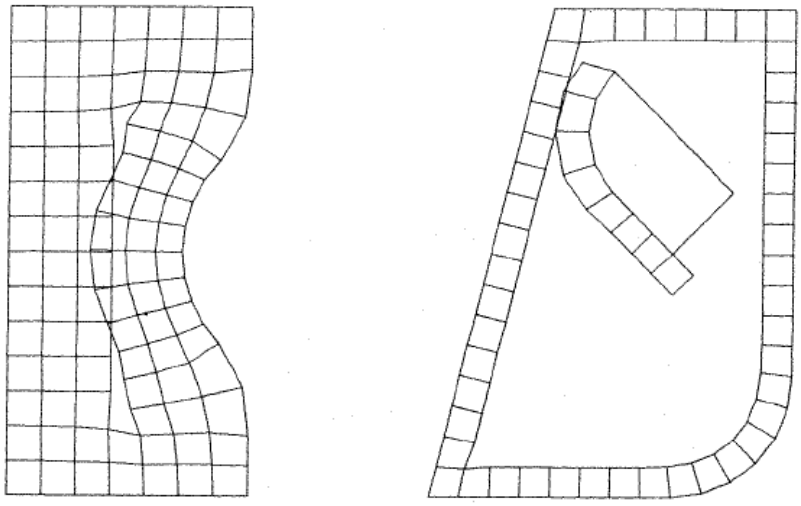

Abbildung 2.20: Beispiel: Überschneidungen [19]

Überschneidungen:

Der Paving-Rand wird nach Überschneidungen mit sich selbst oder anderen Paving-Rändern untersucht (Abb.: 2.20), [19].

Cleanup:

Das gesamte Netz wird dort neu ausgerichtet, wo das Entfernen oder Hinzufügen von Elementen eine Verbesserung der gesamten Netzqualität führt. Dabei spielen das Aspektverhältnis, irreguläre Knoten oder schlechte innere Winkel eine wichtige Rolle, [19].

2.3 Delaunay-Triangulierung

Auf der Delaunay-Triangulierung ([3], [45], [60], [21], [70]) basiert ein Teil der in Kapitel 4 beschriebenen Vernetzung. Deswegen wird im Folgenden genauer darauf eingegangen.

Definition 2.3.1: *Genau dann ist eine Triangulierung einer 3D Punktmenge eine Delaunay-Triangulierung, wenn die Umkugel um jeden Tetraeder keinen weiteren Punkt mehr enthält, [3].*

Die Summe der kleinsten Winkel der Tetraeder wird dabei als Nebenbedingung maximiert. Die Delaunay-Triangulierung ist der duale Graph des Voronoi-Diagramms. Die Ecken der Voronoizellen sind die Umkugelmittelpunkte der Tetraeder der Delaunay-Triangulation. Die Voronoi-Zellen erhält man (anschaulich in 2D in Abbildung 2.21) , wenn von allen Dreiecksseiten die Mittelsenkrechten bis zum gemeinsamen Schnittpunkt mit den anderen beiden Mittelsenkrechten desselben Dreiecks eingezeichnet werden, [3]. Im $\mathbb{R}^3$ lassen sich die Voronoizellen auch wie folgt definieren. $\{P_k\}$ sei die zu

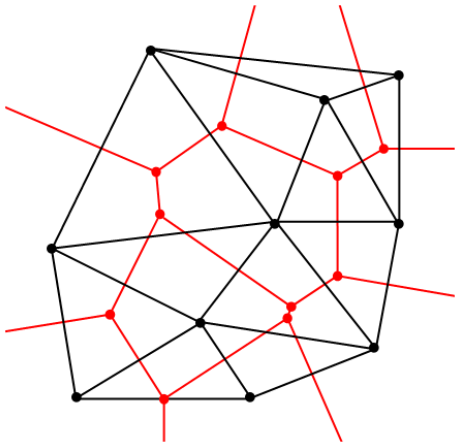

Abbildung 2.21: Zusammenhang: Delaunay-Triangulierung (schwarz) und Voronoi-Diagramm (rot) in 2D [1]

triangulierende Punktmenge in $\mathbb{R}^3$. Dann ist jede Voronoizelle folgendermaßen definiert:

$$V_i = \{P \in \mathbb{R}^3 \mid \|P - P_i\| \leq \|P - P_j\|,\ \forall\ i \neq j\} \tag{2.3.1}$$

2.3.1 Liegt ein Punkt innerhalb einer Kugel?

Um im Mehrdimensionalen zu entscheiden, ob ein Punkt innerhalb einer Kugel liegt, wird hier oft eine Projektion zur Hilfe genommen, [45]. Im zweidimensionalen läßt sich diese Idee sehr schön veranschaulichen. In Abbildung 2.22 liegen die Punkte P_2, P_3 und P_4 auf einem Kreis. Der Punkt P_1 liegt innerhalb des Kreises. Durch die beschriebene Projektion werden alle Punkte der Ebene auf ein Paraboloid abgebildet. Die Punkte P_2', P_3' und P_4' liegen natürlich immer noch auf einem Kreis und beschreiben eine Ebene. P_1' liegt nicht auf dieser Ebene, sondern "rechts" davon. Was dies bedeutet wird

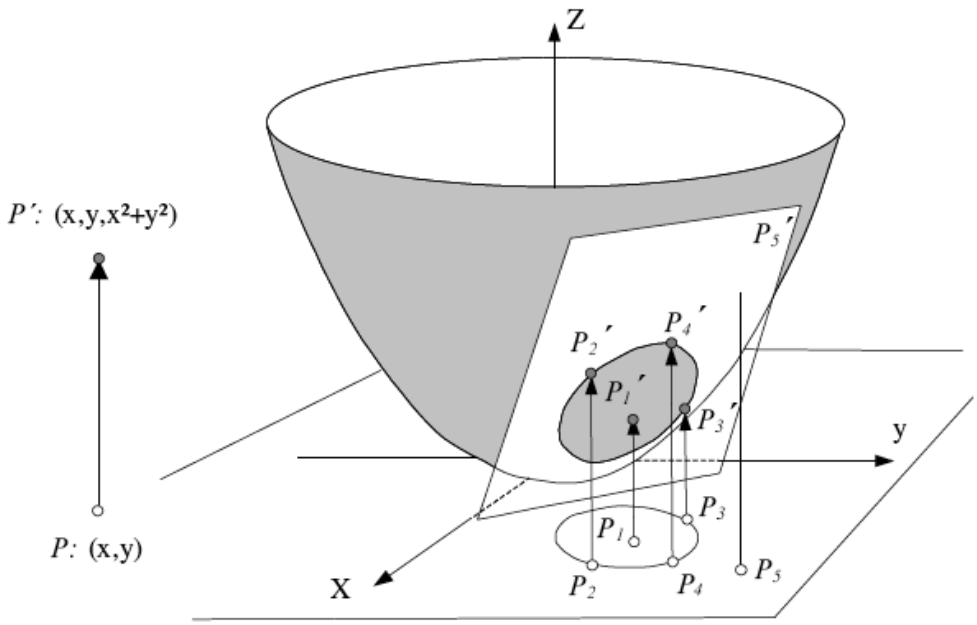

Abbildung 2.22: Idee: Projektion in 2D [45]

nachfolgend angegeben. Genau diese Eigenschft macht man sich zunutze.

Gegeben seien $d+1$ Punkte $P_1, ..., P_{d+1}$ im $\mathbb{R}^d$ mit $d \in \mathbb{N}$. Dann gilt folgendes Lemma:

Lemma 2.3.2: *P_1 liegt genau dann im rechten Halbraum, abgegrenzt durch die Hyperebene, welche durch die linear unabhängigen Punkte $P_2, P_3, ..., P_{d+1}$ aufgespannt wird, wenn* $\det(P_2 - P_1, P_3 - P_1, ..., P_{d+1} - P_1) < 0$ *ist, [45].*

Somit kann im d-dimensionalen Raum für die Punkte $P_1, P_2, ..., P_{d+1}$ entschieden werden, ob der Punkt P_1 im Inneren der Kugel liegt, deren Oberfläche die linear unabhängigen Punkte $P_2, ..., P_{d+1}$ enthält.
Als Hilfsmittel definieren wir die Projektion $\mathbf{P} : \mathbb{R}^d \to \mathbb{R}^{d+1}$ mit

$$\mathbf{P}(x_1, x_2, ..., x_d) = (x_1, x_2, ..., x_d^2 + x_1^2 + x_2^2 + ... + x_d^2). \tag{2.3.2}$$

Es gilt dann folgender Satz:

Satz 2.3.3: *Der Punkt P_1 liegt genau dann im Inneren der Kugel, deren Oberfläche die linear unabhängigen Punkte $P_2, ..., P_{d+1}$ enthält, wenn* $\det(P_2 - P_1, P_3 - P_1, ..., P_{d+1} - P_1) < 0$ *ist, [45].*

2.3.2 Steiner-Triangulierung, Steiner-Punkte

Definition 2.3.4: *Eine Triangulierung wird als Steiner-Triangulierung bezeichnet, wenn zusätzlich Punkte hinzugefügt wurden, [37].*

Definition 2.3.5: *Als Steiner-Punkte werden Punkte bezeichnet, die zu den ursprünglich vorhandenen Punkten hinzugefügt wurden, [37].*

Steiner-Punkte werden hinzügefügt, um die Netzqualität zu steigern. Explizit wird darauf in [21] und [43] eingegangen.

2.3.3 Durchführung einer Delaunay-Triangulierung

Es gibt verschiedene Möglichkeiten eine Delaunay-Triangulierung zu realisiern, [3].

Bei der *Incremental Construction* (Inkrementelle Konstruktion) wird dem Netz immer ein Tetraeder (Dreieck) so hinzugefügt, dass die Delaunay-Bedingung erfüllt bleibt. Der Rechenaufwand liegt bei $\mathcal{O}(n^2)$ und bei angepasster Datenstruktur bei $\mathcal{O}(n \log n)$, [45].

Beim *Sweep-Algorithmus* wird ebenfalls immer ein Tetraeder (Dreieck) so hinzugefügt, dass die Delaunay-Bedingung erfüllt bleibt. Es handelt sich jedoch hierbei um ein benachbarten Tetraeder (Dreieck). Bei der inkrementellen Konstruktion kann der Tetraeder beliebig angehängt werden. Der Rechenaufwand liegt auch bei $\mathcal{O}(n \log n)$, [45].

Beim *Voronoi*-Ansatz wird der Voronoi-Graph der Punktmenge bestimmt, [3]. Das Tetraedernetz (Dreiecksnetz) ergibt sich aus der Dualität des Voronoi-Graphen zur entsprechenden Delaunay-Triangulierung.

2.4 Möglichkeiten zur Erzeugung von Hexaedernetzen

Der folgende Abschnitt soll einen Überblick darüber geben, welche die gängigsten Vernetzungsverfahren zur Erzeugung von Hexaedernetzen sind. Dabei werden die einzelnen Algorithmen nur schemenhaft dargestellt.

2.4.1 Sweeping

Der zu vernetzende Körper wird dabei zuerst in (ggfs. gebogene) zylinderförmige Teilstücke zerlegt (Abb.: 2.23). Jedes Teilstück besteht aus zwei Oberflächen und einer Manteloberfläche. Für eine der Oberflächen (Startoberfläche) eines Teilstücks wird dann ein Viereckснetz generiert. Ausgehend von der Startoberfläche wird innerhalb eines Teilstücks durch ein spezielles Projektionsverfahren, das die Geometrie des zu vernetzenden (Teil-)Körpers berücksichtigt, ein Hexaedernetz erzeugt. Das 2D-Netz auf der Oberfläche (Zieloberfläche) besitzt dann die gleiche Topologie wie das 2D-Netz der Startoberfläche.

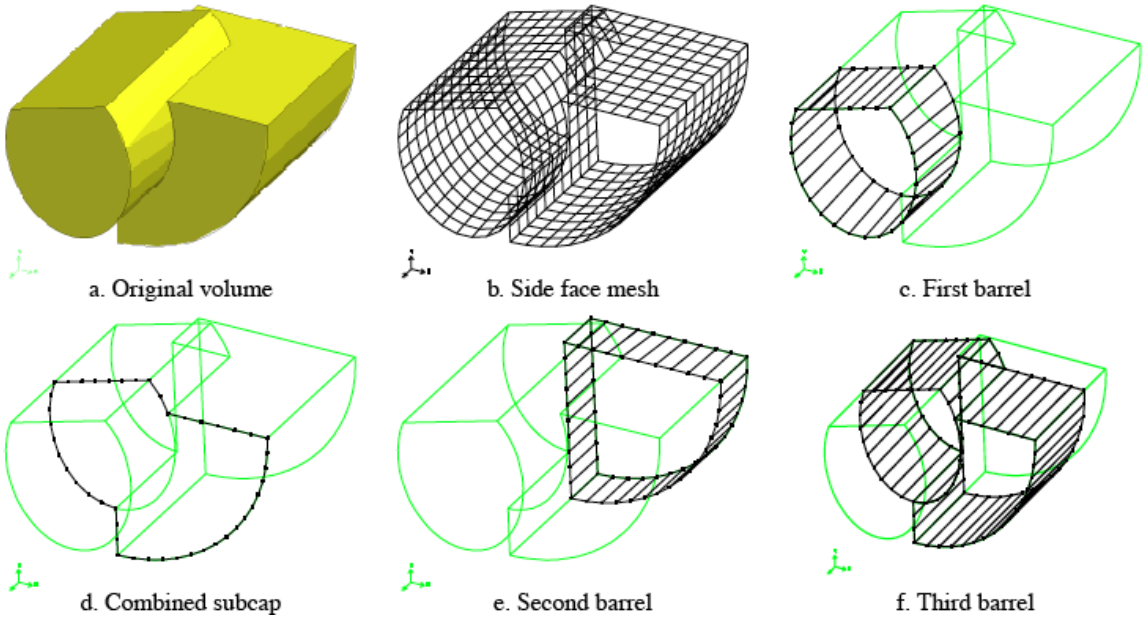

Abbildung 2.23: Beispiel zum Sweeping-Algorithmus: Zerlegung in Teilstücke [18]

Bemerkung 2.4.1: *Dieses Verfahren läßt sich nur sehr schwer automatisieren. Die Zerlegung in Teilstücke muss bei etwas schwierigeren Geometrien manuell vorgenommen werden. Detaillierter wird dieses Verfahren in [18], [56], [63], [23] und [55] beschrieben.*

2.4.2 Octree-Verfahren

Beim Octree-Verfahren (analog zur Quadtree-Technik) wird zuerst ein grobes strukturiertes Hexaedernetz generiert (Abb.: 2.4.2). Oft wird dafür der Begriff Box verwendet. Dann wird die Box solange in 8 Boxen (3D) zerlegt, bis innerhalb einer Box nur noch ein geometrie-definierender Punkt (Kontourpukt) liegt. Danach wird eine Verfeinerung solange vorgenommen, bis sich der Verfeinerungslevel benachbarter Boxen nur noch um 1 unterscheidet. Innere Boxen werden dabei nach einem festen Schema zerlegt. In Boxen, die einen Teil des Randes überdecken, verschiebt man Eckpunkte der Box auf die Kontourpunkte. Boxen, die außerhalb der Kontour liegen werden entfernt.

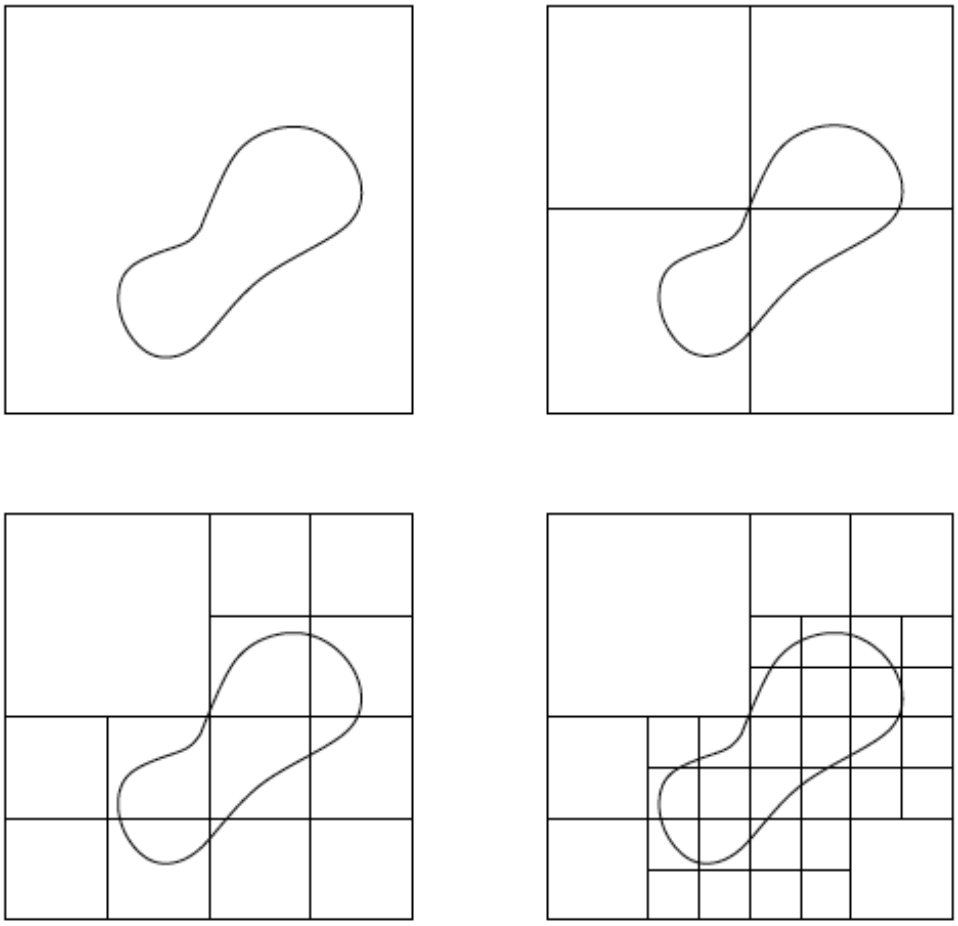

Abbildung 2.24: Octree-Verfahren (2D) [unb.]

Bemerkung 2.4.2: *Diese Vernetzungsmethode läßt sich relativ gut automatisieren. Am Rand der Geometrie ist es jedoch schwierig mit dieser Methode orthogonale Netze zu erzeugen.*

2.4.3 Gitter-basierte Verfahren

Hier wird zuerst der zu vernetzende Körper mit einem orthogonalen Hexaedernetz überdeckt. Dann werden alle Zellen, die nicht vollständig innerhalb des Körpers liegen eliminiert. Zuletzt findet ein Randanpassungsprozess statt. Der grobe Verlauf ist in Abbildung 2.25 veranschaulicht.

Bemerkung 2.4.3: *Diese Vernetzungsmethode läßt sich auch relativ gut automatisieren. Auch hier ist es am Rand der Geometrie jedoch schwierig mit dieser Methode orthogonale Netze zu generieren.*

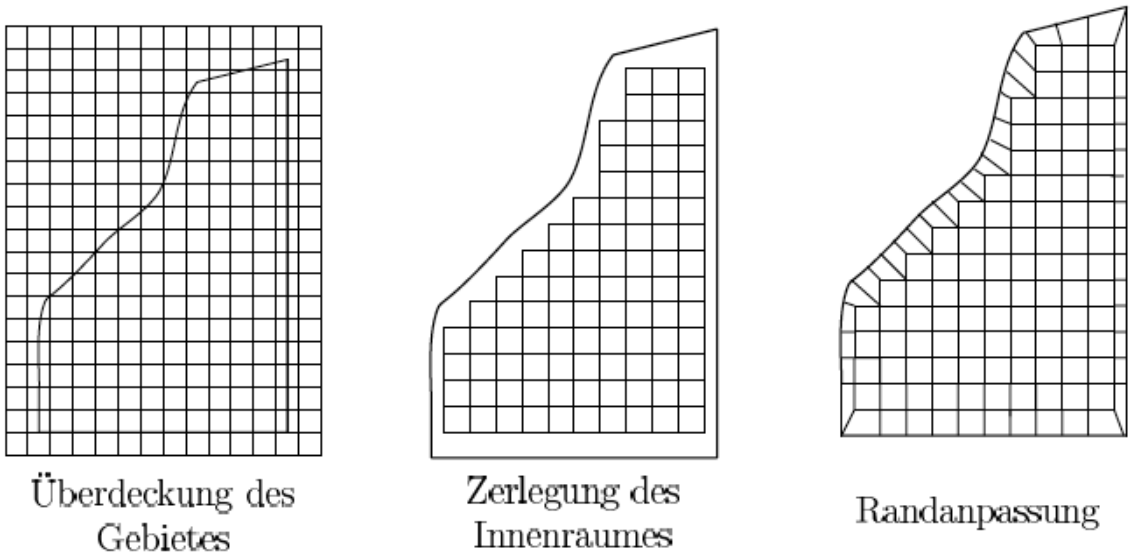

Abbildung 2.25: Gitter-basierte Verfahren (1) [unb.]

Bemerkung 2.4.4: *Weitere Methoden zur Erzeugung von Hexaedernetzen sind die Advancing-Front-Methode ([58], [37], [46], [66], [48]) und das Medial-Axis-Verfahren. Diese sollen an dieser Stelle aber nur erwähnt werden.*

2.5 CUBIT

CUBIT ist ein kommerzielles Vernetzungstool von den Sandia National Laboratories, [2]. Die Software bietet robuste zwei- und dreidimensionale Vernetzungsalgorithmen. Darüber hinaus ist die Möglichkeit der geometrischen Modellierung gegeben. Es sind für die Oberflächenvernetzung Vierecks- und Dreiecksvernetzungsalgorithmen (Paving), 2D und 3D Mapping-Methoden, Sweepingalgorithmen und Tetraedervernetzer (TetMesh) implemeniert. Manche dieser Algorithmen lassen sich bereits automatisch anwenden (z.B. Paving, TetMesher), wobei andere manuelle Hilfe des Benutzers benötigen. Außerdem sind eine Vielzahl von Smoothing-Algorithmen bereitgestellt.
Das Ziel von CUBIT ist es, die Zeit zu reduzieren, welche ein Benutzer für die Modellierung und Vernetzung benötigt. CUBIT lässt sich zum einen über eine GUI und zum anderen über eine Shell bedienen, [57].

Ein Oberflächenvernetzer (Paving-Algorithmus, Abschnitt 2.2) und ein Tetraedervernetzer (TetMesher) wurden in den automatischen Vernetzungszyklus integriert (siehe Kapitel 4).

Kapitel 3

Strömungssimulation

Die Berechnung des Strömungsverhaltens mit numerischen Methoden benötigt Differentialgleichungen und geeignete Anfangs- und Randbedingungen. Dabei werden die physikalischen Vorgänge durch die Differentialgleichungen mathematisch formuliert. Um die Eindeutigkeit der Lösung zu gewährleisten, werden passende Anfangs- und Randbedingungen gebraucht. Als Basis für die Beschreibung einer Strömung dienen die zwei physikalischen Erhaltungsprinzipien für Masse und Impuls. Aus diesen Prinzipien lassen sich die Massen- und Impulserhaltungsgleichung herleiten. Diese beiden Hauptgleichungen können, entspechend der behandelnden Strömung, durch zusätzliche Erhaltungsgleichungen ergänzt werden. [62]

Der erste Abschnitt dieses Kapitels beinhaltet eine Herleitung der generischen Navier-Stokes Gleichung. Im nächsten Abschnitt wird auf die Finite-Volumen-Methode eingegangen. Danach wird detailliert aufgezeigt, wie wichtig die Gitterqualität für aussagekräftige Ergebnisse bei der Finite-Volumen-Methode ist und welche Gittereigenschaften vermieden werden sollten. Anschließend soll ein einfaches Beispiel den Zusammenhang zwischen der Methode und dem zugrunde liegenden Gitter darstellen. Der nächste Abschnitt enthält eine Berechnung des Diskretisierungsfehler bei der Finite-Volumen-Methode. Darauffolgend wird die Finite-Volumen-Methode bei dreidimensionalen Gittern erläutert. Eine Vorstellung des Strömungslösers Comet findet im letzten Abschnitt statt.

3.1 Herleitung der generischen Navier-Stokes Gleichung

Im Folgenden wird immer von *Kontrollvolumen* (KV) gesprochen. Mit KV ist ein vorgegebener Raum, in welchem die vorliegende Strömung untersucht wird, gemeint. Die daraus resultierende Untersuchungsmethode wird als *Kontrollvolumenmethode* bezeichnet. Eine Menge Materie wird als *Kontrollmasse* (KM) bezeichnet. Eine solche Menge Materie besitzt *extensive* Eigenschaften, wie Masse, Impuls und Energie. Als *intensive* Eigenschaften werden die Dichte ρ und die Geschwindigkeit $\mathbf{v}$ bezeichnet. Sie sind un-

abhängig von der Menge der betrachteten Materie.[33]

Nun schauen wir uns die Erhaltungsprinzipien für die Masse und den Impuls an. Da in einer Strömung Masse weder erzeugt noch vernichtet wird, kann die Masse als konstant angesehen werden. Es gilt dann

$$\frac{dm}{dt} = 0, \tag{3.1.1}$$

wobei m für die Masse einer KM und t für die Zeit steht.
Der Impuls einer KM muss aber nicht konstant sein. Dieser kann durch den Einfluss von Kräften verändert werden. Nach dem zweiten Newtonschen Bewegungsgesetz gilt:

$$\frac{dm\mathbf{v}}{dt} = \sum \mathbf{f} \tag{3.1.2}$$

Dabei bezeichnet $\mathbf{v}$ die Geschwindigkeit und $\mathbf{f}$ die Kraft, welche auf die betrachtete KM wirkt.
Für den Zusammenhang einer beliebigen intensiven Erhaltungseigenschaft ϕ (Massenerhaltung: $\phi = 1$, Impulserhaltung: $\phi = \mathbf{v}$) mit der entsprechenden extensiven Eigenschaft Φ gilt:

$$\Phi = \int_{V_{KM}} \rho\phi dV \tag{3.1.3}$$

V_{KM} steht dabei für das Volumen der KM.
Mit Gleichung (3.1.3) lassen sich dann die Massenerhaltungsgleichung ($\phi = 1$)

$$\frac{\partial}{\partial t}\int_V \rho\mathbf{v}dV + \int_S \rho\mathbf{v}\cdot\mathbf{n}dS \;=\; 0 \tag{3.1.4}$$

und die Impulserhaltungsgleichung ($\phi = \mathbf{v}$) herleiten [33]. Die generische Form der Navier-Stokes Gleichungen ergeben sich analog für eine skalare generische Variable ϕ:

$$\frac{\partial}{\partial t}\int_V \rho\phi dV + \int_S \rho\phi\mathbf{v}\cdot\mathbf{n}dS = \sum f_\phi \tag{3.1.5}$$

Dabei beschreibt f_ϕ den Transport von ϕ durch alle Mechanismen außer Konvektion sowie jegliche Quellen oder Senken des Skalars. Auch in ruhenden Fluiden ist diffusiver Transport immer vorhanden und oft mit dem Fick-Gesetz beschrieben. Dabei ist

$$f_\phi^d = \int_S \Gamma\nabla\phi\cdot\mathbf{n}dS, \tag{3.1.6}$$

wobei Γ den Diffusionskoeffizient für die Größe ϕ darstellt. Aus Gleichung (3.1.5) und (3.1.6) ergibt sich dann unmittelbar die generische Transportgleichung der generischen Variablen ϕ:

$$\frac{\partial}{\partial t}\int_V \rho\phi dV + \int_S \rho\phi\mathbf{v}\cdot\mathbf{n}dS = \int_S \Gamma\nabla\phi\cdot\mathbf{n}dS + \int_V q_\phi dV, \tag{3.1.7}$$

wobei q_ϕ Senken oder Quellen von ϕ bezeichnet. Die zeitliche Änderung wird dabei durch den ersten Term beschrieben. Term zwei steht für die Konvektion. Die Diffusion wir durch den Term drei dargestellt und Term vier ist der Quellterm der generischen Variablen ϕ.[33]

3.2 Finite-Volumen-Methode

Die Erhaltungsgleichung in Integralform ist die Basis der Finite-Volumen Methode. Die generische Erhaltungsgleichung hat die Form:

$$\frac{\partial}{\partial t}\int_V \rho\phi dV + \int_S \rho\phi\mathbf{v}\cdot\mathbf{n}dS = \int_S \Gamma\nabla\phi\cdot\mathbf{n}dS + \int_V q_\phi dV \tag{3.2.1}$$

Zuerst wird das Lösungsgebiet dabei in eine endliche Anzahl von KVs unterteilt, auf welchen dann die Erhaltungsgleichung angewendet wird. Der Rechenknoten, in dem die Variablenwerte berechnet werden liegt im Schwerpunkt in dem entsprechenden KV. Somit ergeben sich Bilanzgleichungen für die einzelnen KVs. Für benachbarte Zellen hat das zur Folge, dass was aus dem KV hinausfließt, auch wieder in das benachbarte Kontrollvolumen hineinfließen muss. Um die Variablenwerte auf der KV-Oberfläche mittels der KV-Zentren auszudrücken, wird interpoliert. Geeignete Quadraturformeln approximieren die Oberflächen- ind Volumenintegrale. Somit ergeben sich für jedes KV eine algebraische Gleichung, in welcher die Variablenwerte aus dem eigenen KV-Zentrum und auch aus verschiedenen benachbarten KVs enthalten sind. [33]

3.2.1 Approximation von Flächenintegralen

Die Summe der Flüsse über alle Flächen eines Kontrollvolumens entspricht dem Fluss des KVs über den Kontrollvolumenrand. Der Nettofluss durch den KV-Rand läßt sich demnach darstellen:

$$\int f dS = \sum_k \int_{S_k} f dS, \tag{3.2.2}$$

wobei f die Kompnente des diffusiven ($\Gamma\nabla\phi\cdot\mathbf{n}$) oder des konvektiven ($\rho\phi\mathbf{v}\cdot\mathbf{n}$) Vektors in Richtung der Normalen zur KV-Seite bezeichnet. Das Geschwindigkeitsfeld und die Fluideigenschaften können als bekannt angenommen werden. ϕ hingegen ist unbekannt. Um das Obeflächenintegral in Gleichung (3.2.2) genau berechnen zu können, müsste der Integrand überall auf der Oberfläche bekannt sein. Es werden jedoch nur die Knotenwerte von ϕ im Zentrum vom KV berechnet. Das Oberflächenintegral muss also durch eine Approximation bestimmt werden.[62] Die Mittelpunktregel liefert dabei die einfachste Approximation des Integrals:

$$\int_{S_k} f dS \approx f_k S_k \tag{3.2.3}$$

3.2.2 Approximation der Volumenintegrale

Auch hier erfolgt die Approximation wieder nach der Mittelpunktregel. Dabei wird das Volumenintegral durch das Produkt aus dem Volumen des KV und dem Wert des Integranden im KV-Zentrum approximiert. Somit gilt:

$$\int_V q dV = \overline{q}\triangle V \approx q_P \triangle V, \tag{3.2.4}$$

wobei mit P Werte im KV-Zentrum bezeichnet werden und $\overline{q}$ der Mittelwert von q im KV ist.[62]

3.2.3 Approximation der Quellterme

Ein Volumenintegral kann mit der Mittelpunktregel durch das Produkt aus dem Wert des Integranden im KV-Zentrum und des KV-Volumens approximiert werden. Wir erhalten dann:

$$Q_P^{\phi} = \int_V q_{\phi} dV \approx q_{\phi,P} \triangle V \tag{3.2.5}$$

Wenn der Rechenpunkt P im Schwerpunkt des KV liegt, dann hat die Approximation eine Genauigkeit 2. Ordnung und ist nicht von der KV-Form abhängig.[33]

3.2.4 Interpolationsverfahren

Die lineare Interpolation ist das einfachste Verfahren den Integrand vom KV-Zentrum zu interpolieren. Dabei gilt für einen beliebeigen Punkt x, der ausgehend von dem KV-Zentrum P approximiert werden soll:

$$\Phi(x) = \Phi_P + \left(\frac{\partial \Phi}{\partial x}\right)_P (x - x_P) \tag{3.2.6}$$

Die Berechnung der Oberflächenintegrale kommt nicht ohne die Werte an den KV-Seiten aus. Die Gleichung (3.2.6) liefert für unterschiedliche KV-Zentren jedoch oft unterschiedliche Werte. Abhilfe kann da ein symmetrischer Ausdruck, der als *Zentrale Differenzen* (Central Differencing - CD) gekennzeichnet wird, schaffen. Es gilt

$$\Phi_{\lambda} = \lambda_x \Phi_P + (1 - \lambda_x)\Phi_N, \tag{3.2.7}$$

wobei N das KV-Zentrum einer Nachbarzelle und λ der Zellflächenmittelpunkt ist. Der Interpolationsfaktor λ_x wird dann definiert als

$$\lambda_x := \frac{x_{\lambda} - x_N}{x_P - x_N} \tag{3.2.8}$$

Setzen wir in die Taylorreihenentwicklung von Φ_N um den Punkt x_P die Taylorreihenentwicklung von Φ_{λ} um den Punkt x_P ein, fällt die erste Ableitung weg und wir erhalten:

$$\Phi_{\lambda} = \lambda_x \Phi_P + (1 - \lambda_x)\Phi_N + \left(\frac{\partial^2 \Phi}{\partial x^2}\right)_P \frac{(x_{\lambda} - x_P) - (x_{\lambda} - x_N)}{2} + H \tag{3.2.9}$$

Dieses Interpolationsschema 2. Ordnung kann jedoch bei der Berechnung von konvektionsdominaten Problemen unphysikalische Oszillationen hervorrufen und die Beschränktheit der Lösung kann nicht gesichert werden. [49].

Alternativ gibt es noch das *Upwind*-Schema (Upwind Differencing - UD). Dieses Verfahren ist 1. Ordnung und die Strömung wird dabei berücksichtigt. Aus dem Skalarprodukt der Geschwindigkeit $\mathbf{U}$ mit der Normalen $\mathbf{n}$ der KV-Seite kann die Richtung der Strömung bestimmt werden. Die Strömung flïeßt über die KV-Seite aus der Zelle heraus, wenn $(\mathbf{U}\cdot\mathbf{n})_\lambda > 0$ ist und in das KV hinein, wenn $(\mathbf{U}\cdot\mathbf{n})_\lambda < 0$ ist. Wir erhalten also:

$$\Phi_\lambda = \begin{cases} \Phi_P, & \text{wenn } (\mathbf{U}\cdot\mathbf{n})_\lambda > 0 \\ \Phi_N, & \text{wenn } (\mathbf{U}\cdot\mathbf{n})_\lambda < 0 \end{cases} \tag{3.2.10}$$

Diese Schema garantiert die Beschränktheit der Lösung, allerdings wird eine nicht unerhebliche Menge Diffusuion eingebracht [49].

Damit man sich den Vorteil der Beschränktheit des UD-Verfahrens sowie die höhere Ordnung des CD-Verfahrens zu eigen machen kann, wird das *Blended*-Diskretisierungsschema vorgeschlagen [33]. Es wird dabei eine Linearkombination beider Verfahren betrachtet:

$$\Phi_\lambda = (1-\gamma)(\Phi_\lambda)_{UD} + \gamma(\Phi_\lambda)_{CD}, \tag{3.2.11}$$

wobei $\gamma,\ 0 \leq \gamma \leq 1$ als Blendingfaktor bezeichnet wird und den Anteil der numerischen Diffusion bestimmt.

Allgemein lassen sich folgende Aussagen machen [33]:
Oszillationen treten bei Verfahren höherer Ordnung bei groben Gittern auf. Bei einer Gitterverfeinerung kommen diese Verfahren allerdings schneller zu einer genauen Lösung als Verfahren niedriger Ordnung.
Die Upwind-Approximation ist ungenau. Es wird dabei ein diffusiver Fehler in Stömungsrichtung als auch quer zu ihr eingeführt.
Die lineare Interpolation mit der Genauigkeit 2. Ordnung bietet einen guten Kompromiss zwischen Genauigkeit, Einfachheit und Effizienz.

3.3 Gitterqualität bei der Finite-Volumen-Methode

Die Prinzipien der Finite-Volumen Methode sind gitterunabhängig. Bei unstrukturierten und nichtorthogonalen Gittern treten aber Besonderheiten auf. Diese Besonderheiten werden im folgenden Abschnitt untersucht.
Eine Verfeinerung des Gitters bewirkt normalerweise eine Reduzierung des Diskretisierungsfehlers. Im Vergleich dazu kann eine Optimierung des Gitters bei gleichbleibender Knotenzahl einen größeren Beitrag zur Minimierung des Diskretisierungsfehlers hervorbringen, wie eine Gitterverfeinerung. Deswegen soll in diesem Abschnitt untersucht werden, wie sich die Gitterqualität auf die Ergebnisse der FVM auswirken.
Bei der FVM soll die Optimierung des Gitters die Genauigkeit der Approximationen für Flächen- und Volumenintegrale sowie Gradienten erhöhen.[33]

3.3.1 Konvektiver Fluss:

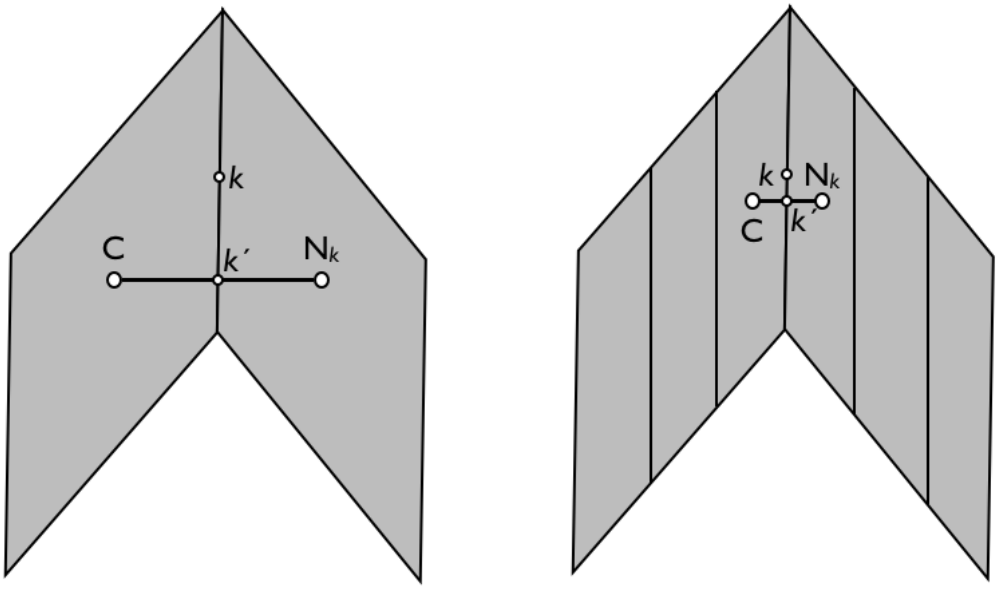

Abbildung 3.1: Verbeserung der nach einer Verfeinerung bezüglich der approximation konvexer Flüsse

Die Linie, welche durch die beiden benachbarten Kontrollvolumen(KV)-Zentren verläuft, sollte durch den Mittelpunkt der Seite verlaufen (siehe Abb.: 3.1). Dadurch erhält man die maximale Genauigkeit der Approximation des konvektiven Flusses durch ein KV-Seite, wenn die lineare Interpolation und die Mittelpunktregel für Integralapproximation verwendet wird. In Abbildung 3.1 wird deutlich, dass eine angepaßte Verfeinerung schon zu einer Verbesserung führen kann. Das Ziel ist es den Abstand $d := \|k - k'\|$ zu verkleinern. In einem Gitteroptimierungsprozess kann es sinnvoll sein, diesen Wert zu

minimieren. In ungünstigen Fällen können zu große Werte für d zu Konvergenzschwierigkeiten und unphysikalischen Lösungen führen.[33]

3.3.2 Diffusiver Fluss:

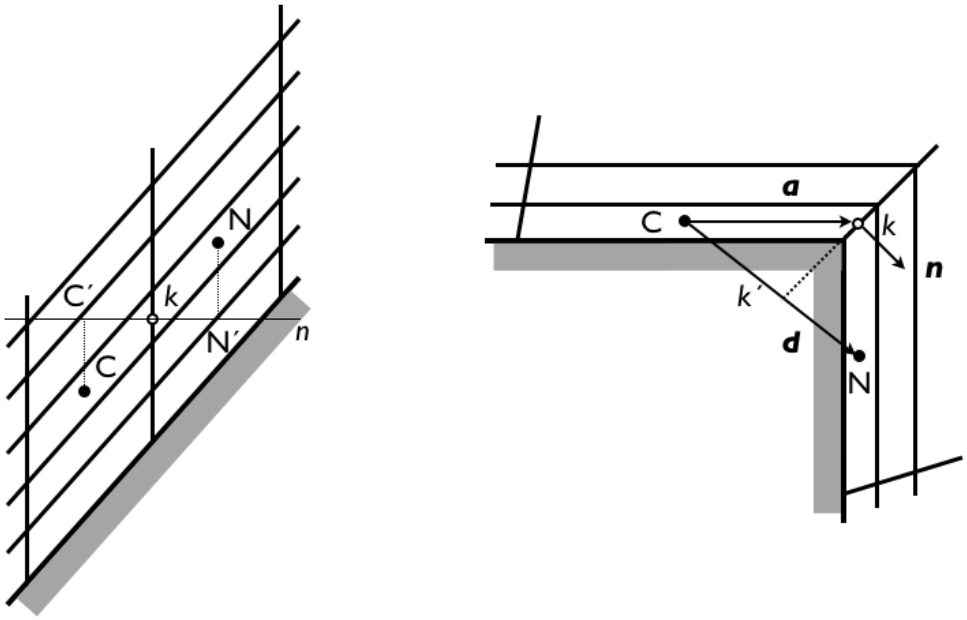

Abbildung 3.2: Beispiele für die Beeinträchtigung der Gitterqualität durch hohe Nichtorthogonalität

Beim diffusiven Fluss wird maximale Genauigkeit dadurch erreicht, dass die Linie, welche die beiden benachbarten KV-Zentren verbindet, orthogonal zur Seite steht und durch den Mittelpunkt der Seite verläuft. Die einfache Zentraldifferenz-Approximation (3.3.1) ergibt eine gute Näherung für die Ableitung in Richtung der Normalen zur KV-Seite, wenn die Othogonalität gegeben ist.[33]

$$\left(\frac{\partial \phi}{\partial n}\right)_{k'} \approx \frac{\phi_{N_k} - \phi_C}{|\mathbf{r}_{N_k} - \mathbf{r}_C|} \tag{3.3.1}$$

Beispiele für Nichtorthogonalität sind im Abbildung 3.2 aufgezeigt. Im linken Bild erkennt man, wie wichtig es ist, dass die Gitterlinien orthogonal zur Oberfläche verlaufen. Das verdeutlicht dann auch die Notwendigkeit des Offset-Algorithmus in Kapitel 4. Dort wurde ein Algorithmus implementiert, der solche Grenzschichten generiert, aber unter der Nebenbedingung, dass die Gitterlinien orthogonal zur Oberfläche sind. Solche Gitter können auch hier wieder zu Konvergenzproblemen und unphysikalischen Lösungen führen.[33]

3.3.3 Unebenheiten:

Gegeben seien die drei Vektoren (siehe Abbildung 3.3):

- **a**: Verbindung vom KV-Zentrum mit dem Zentrum der KV-Seite
- **d**: Verbindung vom KV-Zentrum mit dem Zentrum der Zelle jeneits der betreffenden KV-Seite
- **n**: Flächenvektor der KV-Seite

Dabei gilt:
Sind diese drei Vektoren kolinear, so ist das Gitter orthogonal. Es sollte also

$$\mathbf{a} \cdot \mathbf{n} = 1 \ \wedge \ \mathbf{a} \cdot \mathbf{d} = 1 \ \wedge \ \mathbf{d} \cdot \mathbf{n} = 1 \tag{3.3.2}$$

gelten. Sinnvoll ist eine Optimierung, die eine Maximierung der drei Skalarprodukte bewirkt.[33]

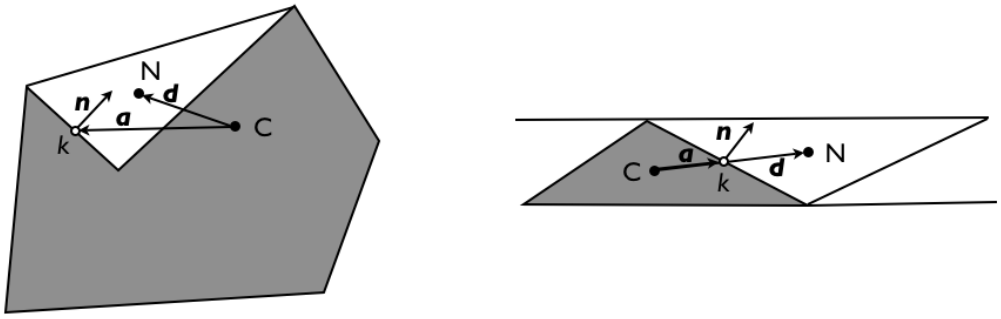

Abbildung 3.3: Gitterqualität bezüglich der Orthogonalität, Konkavität und Unebenheit der KV-Seiten

Bei Tetraedern sind alle vier Seiten eben. Hier kann sich aber die Lage der benachbarten Zellzentren bei starker Nichtorthogonalität negativ auswirken, so dass die Berechnung der Gradienten ungenau wird. Da ein Tetraeder nur vier Nachbarn besitzt, ist dies nicht selten der Fall. Es kann auch noch zu dem Problem kommen, dass ein KV in Wandnähe nur ein oder zwei Nachbarn hat, was unter anderem zu Konvergenzschwierigkeiten führen kann. Das ist unter anderem der Grund, wieso Tetraeder nicht für die Wandnähe verwendet werden sollen. Das ist unter anderem der Grund, wieso für die innere Schicht Hexaeder verwendet werden (siehe Kapitel 4).
Bei Hexaedern kann zusätzlich noch das Problem von unebenen Seiten auftauchen, was aber normalerweise bei geringen Unebenheiten zu nur sehr kleinen Fehlern führt.[33]

3.4 Einfaches Beispiel

Anhand eines Beispiels werden im Folgenden einige Eigenschaften der vorgestellten Diskretisierungsmethoden dargestellt. Dieses Beispiel ist in [33] zu finden.
Dabei wird das Problem des Transports einer skalaren Größe in einem vorgegebenen Geschwindigkeitsfeld betrachtet (siehe Abb.: 3.7.1). Dieses Geschwindigkeitsfeld ist durch $u_x = x$ und $u_y = -y$ gegeben. Dadurch wird eine Strömung nahe einem Staupunkt beschrieben. Durch $xy = const.$ sind die Stromlinien gegeben. Diese ändern ihre Richtung relativ zum kartesischen Gitter. Die normale Geschwindigkeitskomponente ist auf jeder KV-Seite unverändert.

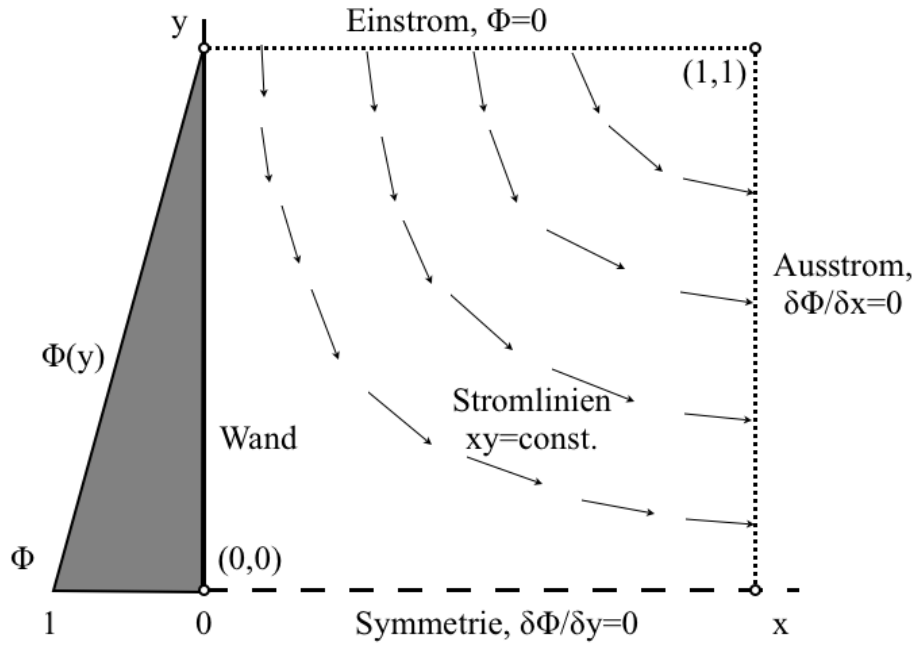

Abbildung 3.4: Randbedingungen und Geometrie für den Skalartransport in einer Staupunktströmung

Die zugehörige Transportgleichung lautet:

$$\int_S \rho\phi\mathbf{v}\cdot\mathbf{n}dS = \int_S \Gamma\nabla\phi\cdot\mathbf{n}dS \tag{3.4.1}$$

Dabei sind folgende Randbedingungen zu erfüllen.

- Einstrom: $\phi = 0$
- Wand: $\phi(y) = 1 - y$
- Südrand: Nullgradient normal zum Rand
- Ausstrom: Nullgradient in Strömungsrichtung

Mehr Details zur Diskretisierung finden sich in [33]. Wir wollen nur auf die Ergebnisse dieses Tests eingehen.

Ergebnisse:

Zuerst betrachten wir die Isolinien von ϕ, welche auf einem äquidistanten Gitter mit $40 \cdot 40$ Kontrollvolumen anhand linearer Interpolation für die konvektiven Flüsse für $\Gamma = 0.001$ und $\Gamma = 0.01$ berechnet wurden (Abb.: 3.5). Der Wert ρ wurde dabei auf 1 gesetzt. Als Resultat ergibt sich, dass der diffusive Transport von ϕ quer zur Strömungsrichtung beim größeren Γ deutlich stärker ist, als beim kleineren Wert. Im Folgenden

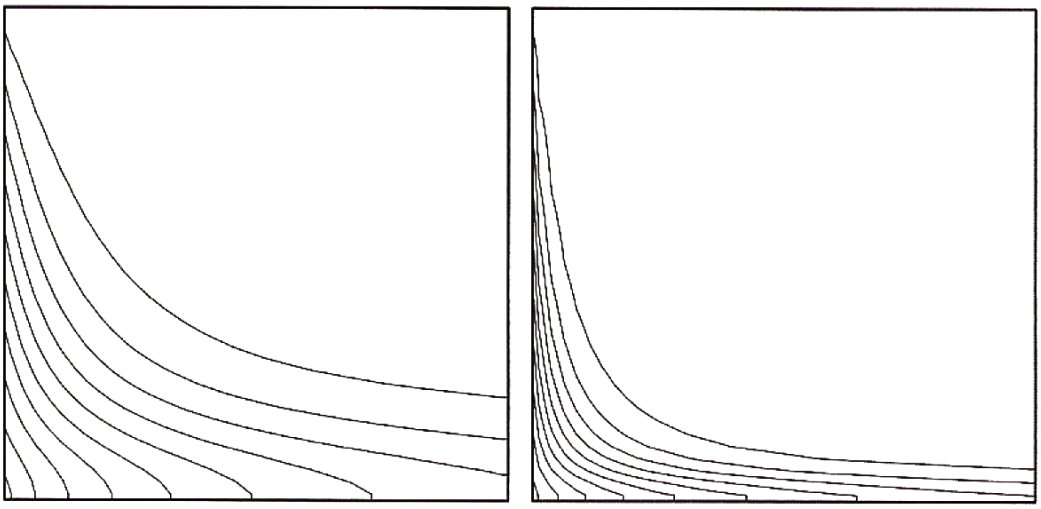

Abbildung 3.5: Isolinien von ϕ ab 0.05 bis 0.95 mit Schritten von 0.1 (von oben nach unten) für $\Gamma = 0.01$ (links) und $\Gamma = 0.001$ (rechts) [33]

wird der gesamte Fluss von ϕ durch den Westrand, an dem ϕ vorgegeben ist, betrachtet um die Genauigkeit der Vorhersage zu beurteilen. Der gesamten Fluss ergibt sich, indem die diffusiven Flüsse über alle KV-Seiten entlang dem Westrand aufsummiert werden.
In Abbildung 3.6 (links) ist die Änderung dieses Flusses mit der Verfeinerung des Gitters für die Aufwind-Approximation und die lineare Interpolation bei der Berechnung der konvektiven Flüsse dargestellt. Dabei wurden die diffusiven Flüsse mit Zentraldifferenzen diskretisiert. Die Verfeinerung des Gitters wurde von $10 \cdot 10$ bis $320 \cdot 320$ Kontrollvolumina vorgenommen.
Die gitterunabhängige Lösung wurde mittels Richardson-Extrapolation [33], ausgehend von einer Konvergenz 2.Ordnung bei der Berechnung mit linearer Interpolation, abgeschätzt. Daraus ergibt sich dann der Fehler in jeder Lösung. Die Fehler sind in Abbildung 3.6 (rechts) als Funktion eines normierten Gitterabstandes dargestellt. $\triangle x = 1$ steht dabei für das gröbste Gitter.

Lineare Interpolation:
Mit linearer Interpolation wurde dabei auf dem gröbsten Gitter keine sinnvolle Lösung erzielt. Durch eine Verfeinerung des Gitters konvergieren die Ergebnisse jedoch stetig gegen die gitterunabgängige Lösung. Die Konvergenz verläuft monoton (Abb.: 3.6 (links)).

In Abbildung 3.6 (rechts) stimmt die Fehlerkurve für lineare Interpolation mit dem Verlauf für ein Verfahren 2. Ordnung überein. Die lineare Interpolation liefert ein genaueres Ergebnis schon auf einem Gitter mit $80 \cdot 80$ Kontrollvolumen.

Aufwind-Approximation:
Die Lösungen weisen auf keinem Gitter Oszillationen auf. Die Konvergenz ist hier nicht monoton (Abb.: 3.6 (links)).

In Abbildung 3.6 (rechts) zeigt die Fehlerkurve auf den ersten drei Gittern ein unregelmäßiges Verhalten. Die erwartete Lösung nähert sich die Fehlerkurve ab dem vierten Gitter. Auf dem Gitter mit 320·320 Kontrollvolumina hat die Lösung immer noch einen Fehler von über einem Prozent.

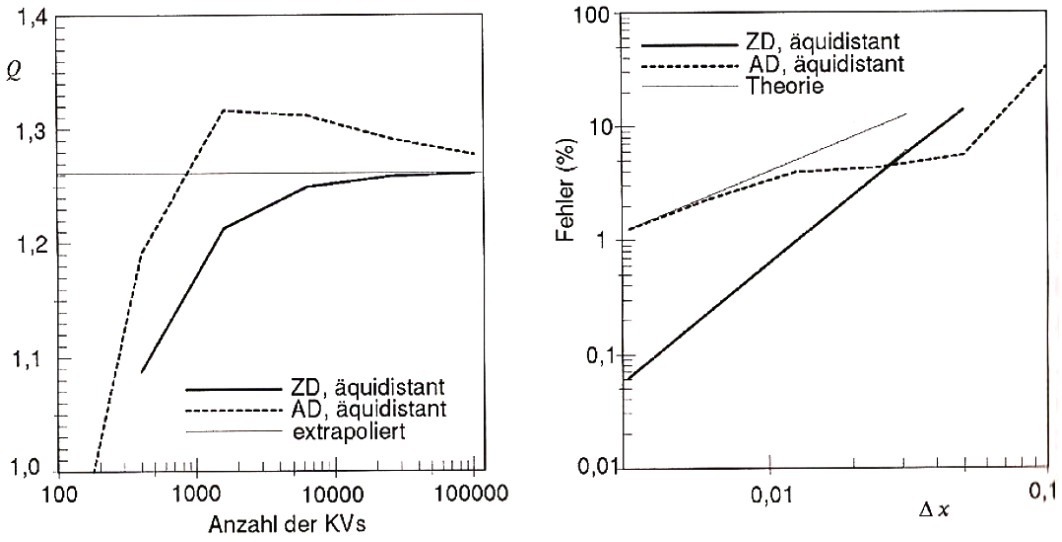

Abbildung 3.6: Links: Konvergenz des Gesamtflusses von ϕ durch den Westrand; Rechts: Fehler im berechneten Fluss als eine Funktion des Gitterabstandes für $\Gamma = 0.01$; AD - Aufwind-Approximation, ZD - lineare Interpolation

Fazit:
Die Ordnung des Verfahrens ermöglicht nur dann eine zuverlässige Abschätzung des Fehlers, wenn die Gitter so fein sind, dass die Lösung monoton konvergiert.

3.5 Dreidimensionale Gitter bei der FVM

Die Diskretisierung an 3D-Gittern soll nur schemenhaft dargestellt werden. Dabei wird nur auf die Berechnung des Volumens eines KV und die Berechnung des Flächenvektors einer KV-Seite eingegangen. Die Techniken, die im 2D-Fall beschrieben wurden, sind auf 3D-Gitter erweiterbar. Dabei ist die größere Komplexität nur geometrischer Natur. [33]

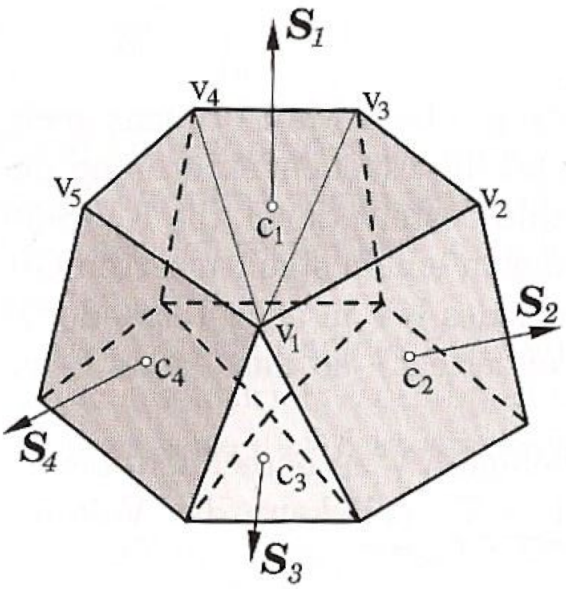

Abbildung 3.7: Beliebiges KV zur Berechnung des Volumens und des Flächenvektors [33]

Berechnung des Volumens eines KV:

Dabei macht man sich die Identität $1 = \nabla \cdot (x\mathbf{i})$ zu nutze. Mit dem Gauß-Theorem gilt dann für das Volumen:

$$\nabla V = \int_V dV = \int_V \nabla \cdot (x\mathbf{i}) dV = \int_S x\mathbf{i} \cdot \mathbf{n} dS \approx \sum_k x_k S_k^x \tag{3.5.1}$$

Wie in Abbildung 3.7 ersichtlich, kennzeichnet k die KV-Seiten und mit S_k^x sind die x-Komponenten des Flächenvektors der KV-Seite gemeint.[33]

Berechnung des Flächenvektors einer KV-Seite:

Hier wird jede KV-Seite in Dreiecke mit einem gemeinsamen Eckpunkt zerlegt (Abb.: 3.7). Der Flächenvektor für die ganze KV-Seite läßt sich dann aus den Flächenvektoren aller Dreiecke berechnen mit:

$$\mathbf{S}_c = \frac{1}{2} \sum_{i=3}^{N_v} [(\mathbf{r}_{i-1} - \mathbf{r}_1) \times (\mathbf{r}_i - \mathbf{r}_1)] \tag{3.5.2}$$

N_v steht dabei für die Anzahl der Eckpunkte am Rand der KV-Seite. Und $\mathbf{r}_i$ bezeichnet den Positionsvektor des Eckpunktes i. [33]

3.6 Comet

Comet (Continuum Mechanics Engineering Tool) ist der Strömungslöser, der in diesem Projekt eingesetzt wird. Dieser CFD-Löser basiert auf der Finite-Volumen-Methode. Dieses kommerzielle Softwarepaket der Firma CD-adapco ist in den prozeduralen Programmiersprachen C und Fortran77 implementiert. In diesem Projekt wurde die Version 2.360.Z eingesetzt.
In diesem Programm sind das Berechnungsprogramm Comet und der Prä- und Postprozessor *Cometpp* enthalten, mit welchen Gitter generiert, Ergebnisse visualisiert und Fälle aufgesetzt werden. Das *Usercoding* bietet dem Benutzer die Option, das Programm für seine Anwendung zu ergänzen. Die Eingabe der Daten für den Berechnungsfall wird in Comet entweder über eine GUI oder anhand von Befehlen in Cometpp realisiert. Es können so Parameter für

- das Gitter,
- die Stoffeigenschaften,
- die Gleichungslöser
- die zu lösenden Gleichungen
- die Randbedingungen und
- die Paralleleisierung

eingegeben werden. Nach dieser Eingabe wird Cometpp beendet und der Strömungslöser über eine Kommandozeile gestartet.[62]
In [38] befindet sich eine detaillierte Beschreibung der Bedienung von Comet.

3.7 Berechnung des Diskretisierungsfehlers bei der Finite-Volumen Methode

Im Folgenden schauen wir uns die Abweichung nur in Richtung ξ, welche durch die Verbindung der Nachbarpunkte $\overrightarrow{P_0, P_j}$ gegeben ist (Abb.: 3.8), an [38].

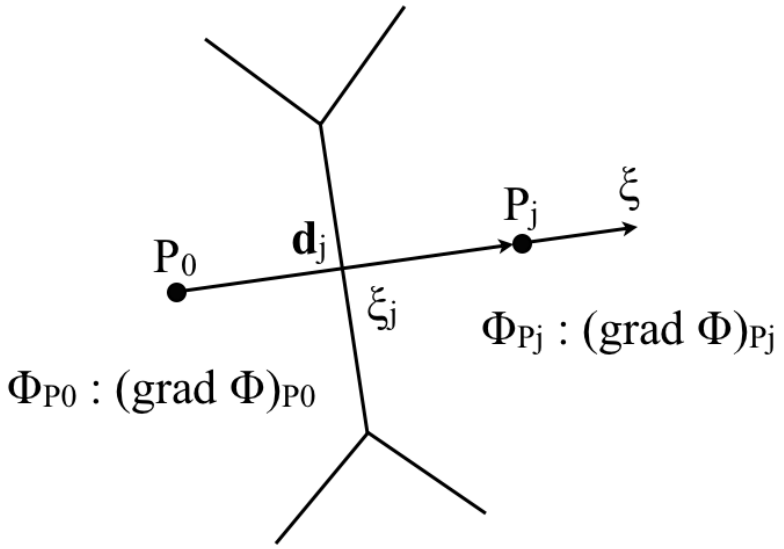

Abbildung 3.8: Eindimensionale Abbruchfehleranalyse für die Variable ϕ

Für $\phi(\xi)$ betrachten wir dabei:

$$\phi(\xi) = c_0 + c_1\xi + c_2\xi^2 + c_3\xi^3 \tag{3.7.1}$$

Dabei müssen die Koeffizenten c_i die folgenden Bedingungen

$$\xi = 0: \quad \phi = \phi_{P_0}; \tag{3.7.2}$$
$$\xi = 0: \quad (\nabla\phi)^{\xi} = (\nabla\phi)^{\xi}_{P_0}; \tag{3.7.3}$$
$$\xi = |\mathbf{d}_j|: \quad \phi = \phi_{P_j}; \tag{3.7.4}$$
$$\xi = |\mathbf{d}_j|: \quad (\nabla\phi)^{\xi} = (\nabla\phi)^{\xi}_{P_j}; \tag{3.7.5}$$

erfüllen, wobei $(\nabla\phi)^{\xi}_{P_j}$ der Gradient in Richtung ξ ist. Somit gilt folgende Gleichung:

$$(\nabla\phi)^{\xi}_{P_j} = (\nabla\phi)_{P_j} \cdot \mathbf{d}_j \tag{3.7.6}$$

Für die Koeffizienten erhalten wir:

$$c_0 = \phi_{P_0}, \tag{3.7.7}$$
$$c_1 = (\nabla\phi)^{\xi}_{P_0}, \tag{3.7.8}$$
$$c_2 = 3\frac{\phi_{P_j} - \phi_{P_0}}{|\mathbf{d}_j|^2} - \frac{(\nabla\phi)^{\xi}_{P_j} + 2(\nabla\phi)^{\xi}_{P_0}}{|\mathbf{d}_j|}, \tag{3.7.9}$$
$$c_3 = -2\frac{\phi_{P_j} - \phi_{P_0}}{|\mathbf{d}_j|^3} + \frac{(\nabla\phi)^{\xi}_{P_j} + (\nabla\phi)^{\xi}_{P_0}}{|\mathbf{d}_j|^2}. \tag{3.7.10}$$

Mit Hilfe des Ausdrucks 3.7.1 können die Werte der abhängigen Variablen und ihrer Gradienten in Richtung d_j für die KV-Seite j, geschätzt werden.

$$\widetilde{\phi_j} = c_0 + c_1\xi_j + c_2\xi_j^2 + c_3\xi_j^3 \tag{3.7.11}$$
$$\widetilde{\nabla\phi_j} \cdot \frac{\mathbf{d}_j}{|\mathbf{d}_j|} = c_1 + 2c_2\xi_j + 3c_3\xi_j^2 \tag{3.7.12}$$

Somit ergeben sich für die Masse und die konvektiven und diffusiven Flüsse:

$$\widetilde{\dot{m}_j} = \rho_j\widetilde{\mathbf{v}}_j \cdot s_j, \tag{3.7.13}$$
$$\widetilde{C_j} = \widetilde{\dot{m}_j}\widetilde{\phi_j}, \tag{3.7.14}$$
$$\widetilde{D_j} = -\lambda_{\phi_j}\left(\widetilde{\nabla\phi_j} \cdot \frac{\mathbf{d}_j}{|\mathbf{d}_j|}\right)\frac{\mathbf{d}_j}{|\mathbf{d}_j|} \cdot s_j. \tag{3.7.15}$$

Summieren wir die Differenzen der konvektiven und diffusiven Flüsse über alle Flächen auf, ergibt sich der Ausdruck für den Diskretisierungsfehler [38]:

$$\widetilde{\tau}_{P_0} = \sum_{j=1}^{n_f}\left[\left(\widetilde{C_j} - C_j\right) + \left(\widetilde{D_j} - D_j^N\right)\right] \tag{3.7.16}$$

Kapitel 4

vmesh: Automatische Gittergenerierung

Im folgenden Kapitel werden die theoretischen Hintergründe des Vernetzungstools **vmesh** erklärt, welches im Rahmen des Forschungsprojekts SimuVSP entstand. Dabei wird zuerst auf das Gesamtkonzept der Vernetzung eingegangen. Die zuvor ausgearbeiteten Bedingungen, die das Netz dabei zu erfüllen hat, fließen in das Gesamtkonzept mit ein. Die einzelnen Vernetzungsmethoden werden detailliert erläutert und die entwickelten Algorithmen erklärt.
Außerdem wird die Benutzung dieses Programms dargelegt, was den Einstieg zur Anwendung von **vmesh** erleichtert. Gewisse Teile des nachfolgenden Kapitels können deswegen auch als Handbuch verstanden werden. Ein Abschnitt beinhaltet eine Beschreibung wichtiger CAD-Formate. Darüber hinaus wird dargelegt, wie die Ordnerstruktur aussieht. Der nächste Abschnitt umschließt die Exportmöglichkeiten des Vernetzers. Des Weiteren werden die Einstellungsmöglichkeiten besprochen, welche dieses Programm bietet und wo diese vorzunehmen sind. Der letzte Abschnitt enthält eine Untersuchung des Rechenaufwandes des gesamten Vernetzungsablaufs.

4.1 Was ist vmesh?

Im Rahmen des Projektes SimuVSP sollen, in einer Kooperation mit Voith Turbo Schneider Propulsion, die Rümpfe von Schiffen, die mit Voith-Schneider-Propeller (VSP) (siehe Kapitel 1.2) ausgerüstet sind, hinsichtlich ihrer Strömungsverluste optimiert werden. Dabei teilt sich das Gesamtproblem der Optimierung in vier Teilprobleme auf. Die parametrische Modellierung der Schiffskörpergeometrie, die automatische Netzgenerierung für diese Art von Schiffen, die Strömungsberechnung und die Optimierung. Den sehr anspruchsvollen Teil der automatischen Vernetzung wurde mit dem C++ Programm **vmesh** realisiert.
Ziel war die Erstellung eines automatischen Vernetzers, welcher ein Elementnetz für den CFD-Solver generiert und dabei insbesondere die Formspezifika VSP- getriebener Schiffe berücksichtigt und automatisiert abläuft. Um das Schiff soll dabei eine Box gelegt und das Gebiet zwischen Schiffskörper und Boxberandung mit Kontrollvolumen

(Hexaedern) gefüllt werden.
Damit für den Strömungslöser COMET [1] (Kapitel 3.6) Qualitätsnetze generiert werden konnten, mussten vorab Kriterien entwickelt werden, die das Volumennetz erfüllen muss (Abschnitt 4.2). Das Netz soll bei der späteren Strömungsberechnung aussagekräftige und realistische Ergebnisse liefern.
Um all diese Kriterien erfüllen zu können, wurde ein Gesamtkonzept für das Volumennetz entwickelt, welches in Abschnitt 4.4 genauer erläutert wird. Darüber hinaus wurden auch neue Vernetzungsalgorithmen entwickelt und implementiert. Unter anderem wurde ein Offset-Algorithmus entwickelt, der es ermögicht aus einem bestehenden Oberflächennetz aus Vierecken ein Volumennetzetz aus Hexaedern für die Grenzschicht zu generieren (Abschnitt 4.6). Algorithmen hierzu sind aus der Literatur nicht bekannt. So entstand ein Vernetzungstool **vmesh**, welches es ermöglicht, für sehr komplexe Geometrien vollautomatisch Qualitätsnetze für Strömungsberechnungen zu generiern. Dies soll in diesem Kapitel aufgezeigt werden.

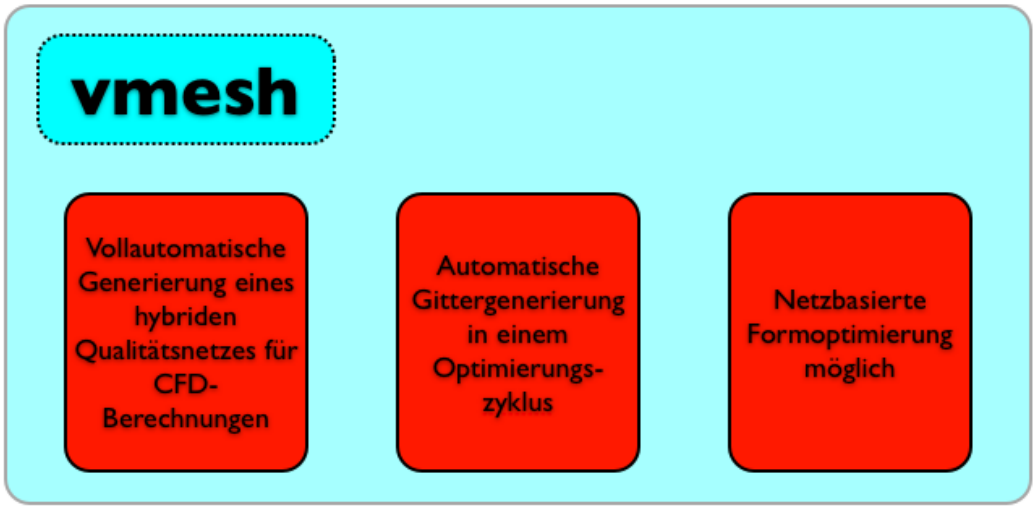

Abbildung 4.1: Anwendungsmöglichkeiten des automatischen Vernetzungstools **vmesh**

Dieser automatische Vernetzer wurde einerseits bei der Optimierung von Schiffen in die Optimierungskette (Abb.: 5.1) integriert (Kapitel 5.1). Darüber hinaus ist in **vmesh** die Möglichkeit der netzbasierten Formoptimierung realisiert (Kapitel 5.2). In Abbildung 4.1 sind die möglichen Anwendungsgebiete des automatischen Vernetzers **vmesh** dargestellt.

[1] COMET: CFD-Solver

4.2 Voraussetzngen an das Gesamtvolumennetz

Zu Beginn des Projektes wurden bisher existierende Möglichkeiten der Vernetzung herausgearbeitet und darüber diskutiert, welche für unser Projekt in Frage kommen (Kapitel 2.4). Hier entschloss man sich für ein hybrides Netz, um die Vorteile von Hexaeder- und Tetraedernetzen abgreifen zu können. Anschließend wurden folgende Kriterien, welche das Netz erfüllen sollte, festgelegt. Diese Kriterien beruhen unter anderem auf der Grundlage bisheriger Erfahrungswerte mit COMET. In Kapitel 3.3 (und 3.4) wird der Zusammenhang der Gitterqualität mit den Ergebnissen aus der Strömungssimulation der Finite-Volumen Methode herausgearbeitet. Die Tatsache, dass ein hochwertiges Gitter für aussagekräftige Ergebnisse notwendig ist, wird dort fundiert dargestellt und gleichzeitig werden die folgenden Gitterkriterien dadurch untermauert.

Allgemeine Gitterkriterien:

- Die Zellen sollen nach Möglichkeit die Form von Hexaedern haben.
- Das Verhältnis der Volumina zweier benachbarter Zellen soll maximal 1:2 sein.
- Die Innenwinkel einer Zelle sollten das Minimum von 10° nicht unterschreiten und das Maximum von 170° nicht überschreiten.

Gitterkriterien für Zellen mit geringem Abstand zum Schiffsrumpf:

- Es soll eine äquidistante Schicht geben, welche dem Rumpf direkt anliegt. Äquidistant heißt hier, dass alle äußeren Punkte dieser Schicht in etwa den gleichen Abstand vom Rumpf haben.
- Die Gitterlinien dieser Schicht sollen entweder orthogonal bzw. parallel zum Rumpf verlaufen.
- Für jeden Knick des Schiffes soll es eine Gitterlinie geben, welche exakt auf diesem Knick verläuft.

Gitterkriterien für Zellen mit goßem Abstand zum Schiffsrumpf:

- Die Gitterlinien an der Wasseroberfläche sollen horizontal verlaufen.
- Hängende Knoten sind erlaubt.
- Das Gesamtgitter soll Quaderform haben.

Zur Umsetzung all dieser Kriterien wurde ein Modell herausgearbeitet, welches aus drei Schichten besteht. Dem Schiffsrumpf soll eine innere Schicht direkt anliegen und die Gitterkriterien für die Zellen mit geringem Abstand zum Schiffsrumpf erfüllen. Eine äußere Schicht soll an die Boxberandung angrenzen und den größten Teil des Netzes ausmachen und die Gitterkriterien für die Zellen mit großem Abstand zum Schiffsrumpf erfüllen. Im Bereich zwischen der inneren und der äußeren Schicht soll eine mittlere Schicht erzeugt werden (siehe Abschnitt 4.4).

4.3 Formate

Bei der automatischen Vernetzung spielen die im Folgenden aufgefürten Formate beim Import und Export von Dateien eine wichtige Rolle. Deswegen wird ein Überblick über die in dieser Arbeit verwendeten Formate gegeben.
Um die Formate besser vegleichen zu können, soll veranschaulicht werden, wie sich ein Dreieck mit den Eckpunkten $(0, 0, 0)$, $(1, 0, 0)$ und $(0, 1, 1)$ bzw. der Einheitswürfel in dem jeweiligen Format darstellt.

Abbildung 4.2: Einheitswürfel und Dreieck

4.3.1 STL-Format

```
solid Body_1
  facet normal 0.000000e+00 -0.000000e+00 1.000000e+00
    outer loop
      vertex 0.000000e+00 0.000000e+00 0.000000e+00
      vertex 1.000000e+00 0.000000e+00 0.000000e+00
      vertex 0.000000e+00 1.000000e+00 0.000000e+00
    endloop
  endfacet
endsolid Body_1
```

Abbildung 4.3: STL-Format eines Dreiecks

Beim STL-Format werden die Oberflächen von 3D-Körpern mit Hilfe von Dreiecksfacetten (englisch tesselation = 'Parkettierung') beschrieben. Dabei wird jede Dreiecksfacette durch drei Eckpunkte und die zugehörige Flächennormale des Dreiecks charakterisiert. Durch die Tatsache, dass mehrere Dreiecke einen gemeinsamen Eckpunkt besitzen, wird also jeder Punkt auch mehrmals aufgelistet. Zusätzlich wird die Flächennormale abgespeichert. Dies führt dazu, dass die Datenmenge enorme Größen annehmen kann. Ein Nachteil dieses Formats ist, dass gekrümmte Oberflächen durch die Dreiecke nur angenähert werden.
Dabei gilt: Je geringer die Anzahl der Dreiecke, desto größer sind die Abweichungen und je genauer die Annäherung sein muss, desto mehr Einzeldreiecke sind nötig. Somit steigt die Datenmenge mit höherer Genauigkeit stark an. [7]

In Abbildung 4.3 wird veranschaulicht, wie sich ein Dreieck mit den Eckpunkten $(0, 0, 0)$, $(1, 0, 0)$ und $(0, 1, 1)$ im STL-Format darstellt.

4.3.2 IGES-Format

IGES (Initial Graphics Exchange Specification) ist ein ANSI-kompatibles, nicht herstellergebundenes Public Domain-Dateiformat. Dieses Format gilt als internationales Standardformat zum Austausch von Produktdefinitionsdaten zwischen den verschiedenen CAD/CAM-Systemen. Es wurde vorwiegend für die Übertragung von Geometriedaten entwickelt. Vor allem betrifft dies 2D- und 3D-Flächenmodelle (inklusive Bezier- und NURBS-Flächen). [11] [9]

```
3D InterOp IGES (Version 17 0 2), Spatial Corp. Copyright (c) 1999-2007.S      1
1H,,1H;,6HNoname,12Htriangle.igs,13HSpatial Corp.,20H3D InterOp ACIS/IGEG      1
S,32,38, 6,308,15,6HNoname,1.000,2,2HMM,1,1.000,15H20100316.211015,     G      2
1.0e-06,0.00,5Hmichi,6HNoname,11,0,15H20100316.211015;                 G      3
     110       1       0       0       0       0       0       000010001D      1
     110       0       0       1       0       0       0               0D      2
     122       2       0       0       0       0       0       000010001D      3
     122       0       0       1       0       0       0               0D      4
     110       3       0       0       0       0       0       000010001D      5
     110       0       0       1       0       0       0               0D      6
     110       4       0       0       0       0       0       000010001D      7
     110       0       0       1       0       0       0               0D      8
     110       5       0       0       0       0       0       000010001D      9
     110       0       0       1       0       0       0               0D     10
     102       6       0       0       0       0       0       000010001D     11
     102       0       0       1       0       0       0               0D     12
     142       7       0       0       0       0       0       000010001D     13
     142       0       0       1       0       0       0               0D     14
     144       8       0       0       0       0       0       000000001D     15
     144       0       0       1       0       0       0               0D     16
110,0.,0.,0.,1.,0.,0.;                                               1P      1
122,1,0.,1.,0.;                                                      3P      2
110,0.,0.,0.,1.,0.,0.;                                               5P      3
110,1.,0.,0.,0.,1.,0.;                                               7P      4
110,0.,1.,0.,0.,0.,0.;                                               9P      5
102,3,5,7,9;                                                        11P      6
142,1,3,0,11,2;                                                     13P      7
144,3,1,0,13;                                                       15P      8
S      1G      3D     16P      8                                        T      1
```

Abbildung 4.4: IGES-Format eines Dreiecks

4.3.3 SAT-Format

ACIS bezeichnet einen Volumenkernel, der zu einem Standard in der CAD-Welt geworden ist. ACIS ist in der objektorientierten Programmiersprache C++ geschrieben und zeichnet sich durch eine offene Systemarchitektur aus. In diesem Format können auch Draht- und Flächenmodelle verwaltet werden. In den neueren Versionen wird die Funktion des toleranten Modellierens unterstützt. Somit können importierte Modelle mit ungenauer Geometriebeschreibung korrekt weiterverarbeitet werden. [11]

```
1700 0 4 0
10 Cubit 11.1 19 ACIS 17.0.2 Unknown 24 Tue Mar 16 21:09:55 2010
1 9.9999999999999995e-07 1e-10
body $-1 -1 -1 $-1 $4 $-1 $-1 T 0 0 0 1 1 0 #
vertex $-1 -1 -1 $-1 $-1 $5 #
vertex $-1 -1 -1 $-1 $-1 $6 #
vertex $-1 -1 -1 $-1 $-1 $7 #
lump $-1 -1 -1 $-1 $-1 $8 $0 T 0 0 0 1 1 0 #
point $-1 -1 -1 $-1 0 0 0 #
point $-1 -1 -1 $-1 1 0 0 #
point $-1 -1 -1 $-1 0 1 0 #
shell $-1 -1 -1 $-1 $-1 $-1 $9 $-1 $4 T 0 0 0 1 1 0 #
face $-1 -1 -1 $-1 $-1 $10 $8 $-1 $11 forward double out T 0 0 0 1 1 0 F #
loop $-1 -1 -1 $-1 $-1 $12 $9 T 0 0 0 1 1 0 periphery $11 F #
plane-surface $-1 -1 -1 $-1 0.35355339059327384 0.35355339059327373 0 0 0 1 1 0 0 forward_v I I I I #
coedge $-1 -1 -1 $-1 $13 $14 $-1 $15 forward $10 $-1 #
coedge $-1 -1 -1 $-1 $14 $12 $-1 $16 forward $10 $-1 #
coedge $-1 -1 -1 $-1 $12 $13 $-1 $17 forward $10 $-1 #
edge $-1 -1 -1 $-1 $18 0 $19 1 $12 $20 forward @7 unknown T 0 0 0 1 0 0 #
edge $-1 -1 -1 $-1 $19 0 $21 1.4142135623730949 $13 $22 forward @7 unknown T 0 0 0 1 1 0 #
edge $-1 -1 -1 $-1 $21 0 $18 1 $14 $23 forward @7 unknown T 0 0 0 0 1 0 #
vertex $-1 -1 -1 $-1 $15 $24 #
vertex $-1 -1 -1 $-1 $15 $25 #
straight-curve $-1 -1 -1 $-1 0 0 0 1 0 0 I I #
vertex $-1 -1 -1 $-1 $16 $26 #
straight-curve $-1 -1 -1 $-1 1 0 0 -0.70710678118654746 0.70710678118654746 0 I I #
straight-curve $-1 -1 -1 $-1 0 1 0 0 -1 0 I I #
point $-1 -1 -1 $-1 0 0 0 #
point $-1 -1 -1 $-1 1 0 0 #
point $-1 -1 -1 $-1 0 1 0 #
End-of-ACIS-data
```

Abbildung 4.5: ACIS-Format eines Dreiecks

4.3.4 VTK-Format

Das Visualization Toolkit (VTK) ist eine Open-Source-C++-Klassenbibliothek. Diese ist besonders geeignet für die 3D-Computergrafik mit einem Augenmerk für die wissenschaftliche Visualisierung. Anfangs diente VTK zur Visualisierung von Bilddaten bildgebender Verfahren wurde aber schon bald in seinem Anwendungsgebiet erweitert. [8] [16]

```
# vtk DataFile Version 3.0
Cube example
ASCII
DATASET UNSTRUCTURED_GRID
POINTS            8            float
                0              0              0
                1              0              0
                1              1              0
                0              1              0
                0              0              1
                1              0              1
                1              1              1
                0              1              1
CELLS            1          9
8        0       1       2       3       4       5       6       7
CELL_TYPES           1
12
```

Abbildung 4.6: VTK-Format des Einheitswürfels

4.3.5 Comet-Format

Der Strömungslöser Comet benötigt als Import der Volumennetze zwei Dateien. In einer Datei (.vrt) sind die Knoten des Netzes nummeriert mit den jeweiligen Raumkoordinaten abgespeichert. In einer zweiten Datei, werden die entsprechenden Zellinformationen (.cel) hinterlegt. Dabei werden die Zellen durchnummeriert und mit den entsprechenden Knoten der Zellecken versehen. In Abbildung 4.7 ist die Knoten- und Zelldatei des Einheitswürfels dargestellt.

```
1       0     0     0
2       1     0     0
3       1     1     0
4       0     1     0
5       0     0     1
6       1     0     1
7       1     1     1
8       0     1     1
```

```
1        1   2   3   5   6   7   8   17   1   1
```

Abbildung 4.7: Knoten- (links) und Zelldatei (rechts) des Einheitswürfels im Comet-Format

4.3.6 Abaqus-Format

```
*HEADING
cubit(/Users/michi/Desktop/cube.inp): 03/17/2010: 15:02:39
*NODE
       1,   0.000000e+00,   0.000000e+00,   1.000000e+00
       2,   0.000000e+00,   0.000000e+00,   0.000000e+00
       3,   0.000000e+00,   1.000000e+00,   0.000000e+00
       4,   0.000000e+00,   1.000000e+00,   1.000000e+00
       5,   1.000000e+00,   0.000000e+00,   1.000000e+00
       6,   1.000000e+00,   0.000000e+00,   0.000000e+00
       7,   1.000000e+00,   1.000000e+00,   0.000000e+00
       8,   1.000000e+00,   1.000000e+00,   1.000000e+00
*ELEMENT, TYPE=C3D8R, ELSET=EB1
       1,       1,       2,       3,       4,       5,       6,       7,       8
```

Abbildung 4.8: Abaqus-Format des Einheitswürfels

Das Abaqus-Format wird oft als Exportformat für 2D und 3D Netze verwendet. In unserem Fall dient dieses Format zum Abspeichern von 2D Oberflächennetzen. In Abbildung 4.8 ist der Einheitswürfel in diesem Format dargestellt.

4.4 Gesamtkonzept der Netzgenerierung für vmesh

4.4.1 Vernetzungskonzept - Schichtenmodell

Im Teilbereich der Netzgenerierung wurde zuerst ein Vernetzungskonzept entwickelt, welches eine Vernetzung fast aller Schiffsgeometrien erlaubt. Dabei werden auch die zuvor festgelegten Gitterkriterien (Abschnitt 4.2) berücksichtigt.
Das so erzeugte hybride Volumennetz besteht aus drei Schichten (Abb.: 4.9). Eine innere Schicht (Abschnitt 4.7) liegt dem Schiffsrumpf direkt an, welche aus Hexaedern besteht. Eine äußere Schicht (Abschnitt 4.8) aus Hexaedern (strukturiert) deckt den größten Bereich von der Strömungsbox zum Schiffrumpf hin ab. Der übrige Bereich wird mittlere Schicht (Abschnitt 4.9) genannt und mit Tetraedern gefüllt. Durch diese Tetraederschicht ist man in der Lage auch sehr komplexe Geometrien automatisch und robust zu vernetzen, da die vorhandenen Algorithmen der automatischen Tetraedervernetzung schon sehr ausgereift und robust sind. Jedoch hätte man für die CFD-Berechnung, die auf der Finiten-Volumen-Methode beruht, gerne ausschließlich Hexaeder, aber die bisher vorhandenen Algorithmen der Hexaedervernetzung sind für solch komplexe Geometrien, welche bei diesem Projekt optimiert werden sollen, nicht ausreichend.
Um an der Grenze zwischen den Hexaedern und den Tetraedern keine hängenden Knoten zu bekommen, wird dort auf die Hexaeder noch eine Pyramidenschicht eingearbeitet. Hängende Knoten sind Gitterpunkte, die nicht für alle angrenzenden Elemente Eckpunkte sind.
Für das Oberflächennetz wird ein Paving-Algorithmus (siehe Abschnitt 2.2) und für die mittlere Schicht TetMesh (beides von CUBIT) verwendet. Um aus dem Oberflächennetz die innere Schicht generieren zu können, wurde ein Offsetalgorithmus (Abschnitt 4.6) entwickelt und implementiert (**vmesh**, C++). Die äußere Schicht wird auch von **vmesh** erzeugt.
Abbildung 4.9 soll das Schichtenmodell veranschaulichen.

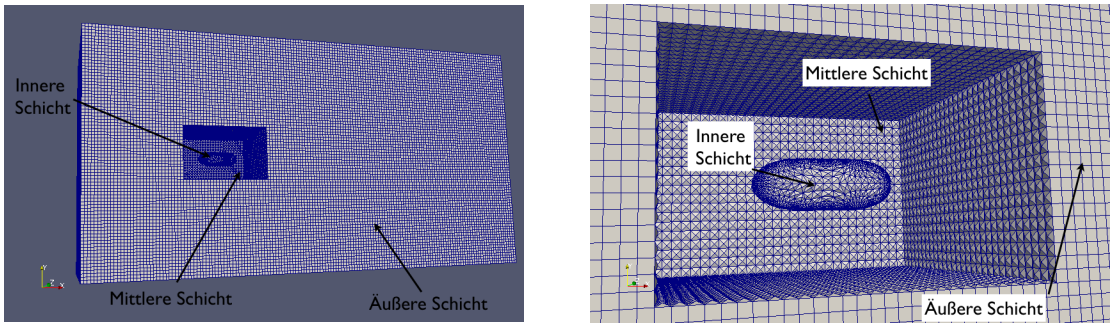

Abbildung 4.9: Schichtenmodell

4.4.2 Ablauf des Vernetzungsprozesses

Zuerst soll ein grober Überblick über den Ablauf gegeben werden, so dass die nachfolgenden Abschnitte, in denen das Programm detailliert beschrieben wird, einfacher zu verstehen sind (Abb.: 4.10).

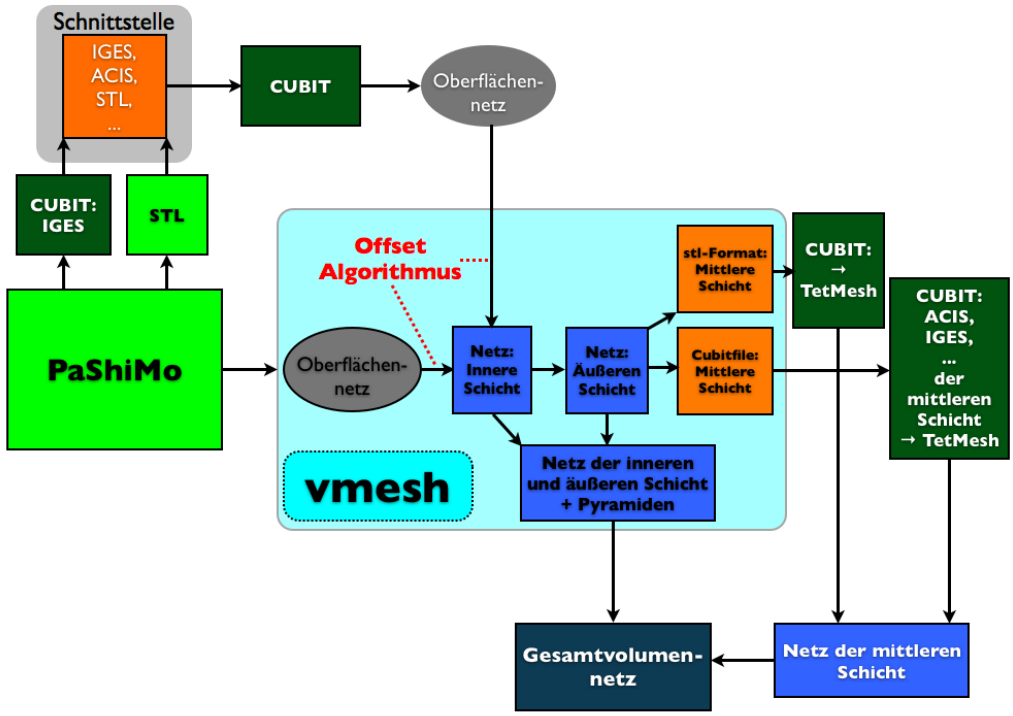

Abbildung 4.10: Weg zum Gesamtvolumennetz

Vernetzbar mit **vmesh** sind Geometrien (Schiffe, Autos, Flugzeuge) die entweder im STL-, IGES- oder ACIS-Format vorliegen. Ist dies erfüllt, so kann **vmesh** ein sehr hochwertiges Volumennetz für Strömungsberechnungen generieren.
Eine solche Geometrie im STL-, IGES- oder ACIS-Format besteht normalerweise aus mehreren Surfaces. Diese werden zuerst 'gestitcht' bzw. 'gemergt' (vernäht), [57]. Dies hat den Grund, da sonst die einzelnen Surfaces unabängig voneinander vernetz werden und dann das Netz nicht in sich geschlossen ist. Hat man die Surfaces miteinander vernäht so wird mit dem Paving-Algorithmus (Cubit) (siehe Abschnitt 2.2) ein geschlossenes Oberfächennetz generiert. Dabei wird berücksichtigt, dass auf den Kanten der einzelnen Surfaces entsprechend Gitterlinien verlaufen. Dieses Netz wird in das Abaqus-Format exportiert. Ausgangspunkt (Import) des Vernetzers ist immer ein solches Oberflächennetz der zu vernetzenden Geometrie.
Mit Hilfe dieses Oberflächennetzes wird die innere Schicht generiert (siehe Abschnitt 4.7), welche ausschließlich aus Hexaedern besteht und der Geometrie direkt anliegt. Extra zu diesem Zwecke wurde ein Offset-Algorithmus entwickelt (siehe Abschnitt 4.6),

welcher sehr robust arbeitet und geometrieunabhängig ist.
Anschließend wird die äußere Schicht erzeugt, welche ebenfalls ausschließlich aus Hexaedern besteht und zudem strukturiert ist (siehe Abschnitt 4.8).
Auf die äußeren Zellen der inneren Schicht und auf die äußeren Zellen des ausgesparten Bereichs der äußeren Schicht werden Pyramiden angebracht, so dass nach der Vernetzung der mittleren Schicht mit Tetraedern keine hängenden Knoten entstehen.
Um die mittlere Schicht noch mit Tetraedern füllen zu können, werden alle Pyramidenseiten und der Deckel im STL-Format generiert. Das STL-File wird direkt von **vmesh** generiert und muss zur weiteren Verarbeitung von Cubit nicht mehr vernäht werden.
Die Netze der einzelnen Schichten werden in einem letzten Schritt zu einem hochwertigen Volumennetz zusammengefügt. Die einzelnen Schritte sind in Abbildung 4.10 dargestellt. Der gesamte Vernetzungsablauf wird dabei durch das nachfolgende Skript (*vmesh_script.cpp*) gesteuert, welches in den folgenden Abschnitten detailliert erläutert wird.

vmesh_script.cpp: (Für Mac OSX)

```
int main()
{
    //*************Generierung des Oberflaechennetzes******************
    system("./vmesh_automation_auto");

    system("/Applications/Cubit-11.1/cubit.command -input
    ./Automation/automation_vmesh_cubit.jou");
    //*****************************************************************

    //*************Generierung der inneren Schicht*********************
    //*************Generierung der aeusseren Schicht*******************
    system("./vmesh_v29_2");
    //*****************************************************************

    //*************Generierung der mittleren Schicht*******************
    system("/Applications/Cubit-11.1/cubit.command -nographics -input
    ./Export/Cubit/middle");
    //*****************************************************************

    //***********Alle Schichten ---> Gesamtvolumennetz*****************
    system("./vmesh_complete");
    //*****************************************************************

    return 0;

}
```

4.5 Generierung des Oberflächennetzes

Die Geometrien (Schiffe, Autos, Flugzeuge), die **vmesh** vernetzen soll, müssen bisher entweder im STL-, IGES- oder ACIS-Format vorliegen. Dabei ist es nicht relevant, ob die Geometrie als Voll- oder Halbmodell vorliegt. Das Vernetzungstool **vmesh** kann für eine solche Geometrie ein sehr hochwertiges Volumennetz für Strömungsberechnungen generieren.

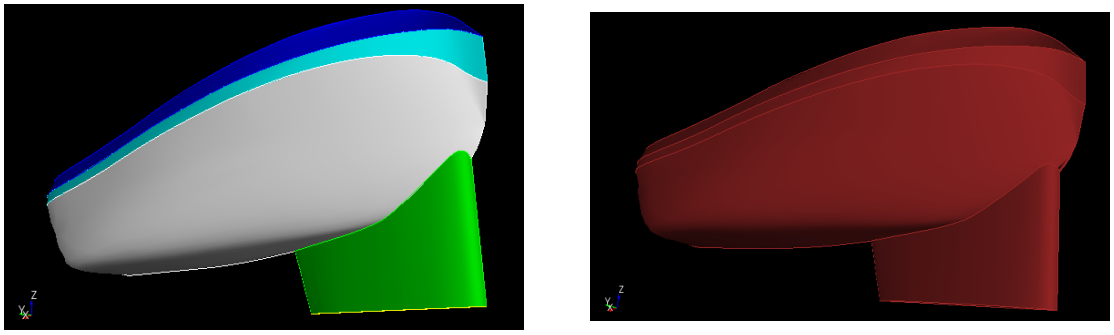

Abbildung 4.11: Surfaces

Da eine solche Geometrie normalerweise aus mehreren Surfaces (siehe Abbildung 4.11) besteht, werden diese dann zuerst 'gestitcht'.

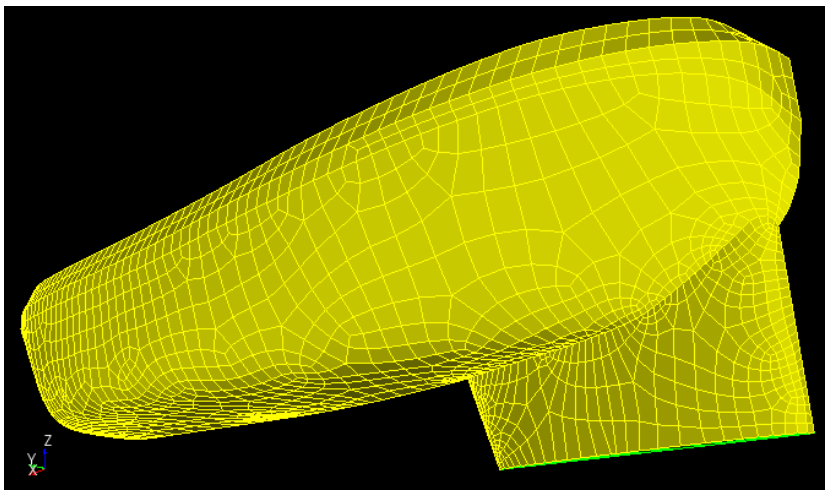

Abbildung 4.12: Oberflächennetz

Wenn die einzelnen Surfaces unabängig voneinander vernetz werden, dann ist das Netz nicht in sich geschlossen. Sind die Surfaces miteinander vernäht so wird mit dem Paving-Algorithmus (Cubit) ein geschlossenes Oberfächennetz generiert. Es wird dabei berücksichtigt, dass auf den Kanten der einzelnen Surfaces entsprechend Gitterlinien verlaufen. Dieses Netz wird in das Abaqus-Format exportiert (siehe Abbildung 4.12). Es besteht die Möglichkeit dieses Netz mit verschiedenen Glättungsmethoden zu bearbeiten. Dieser Teil ist im Programmteil *vmesh_automation_auto.cpp* realisiert.

vmesh_automation_auto.cpp: (Beispiel)

(*vmesh/Automation/vmesh_automation_auto.cpp*)

```
int main()
{
    fstream f;

    string work_dir = getenv("SHIP_HOME");

    f.open("./Automation/automation_vmesh_cubit.jou", ios::out);

    //*****************Importieren der Geometrie**********************
    f << "import stl '" << work_dir << "/vmesh/Import/test_geometry/
    ship_complete_mod_small.stl' feature_angle 135.00 linear merge"
    << endl << endl;

    //*****************Saklierung und Translation*********************
    f << "volume all scale 62.5" << endl << endl;
    f << "volume all move x -62.5 y 0 z 200" << endl << endl;

    //**************Vernaehen der einzelnen Flaechen******************
    f << "merge volume all with volume all" << endl << endl;

    //**************Zellgroesse und Vernetzungsart********************
    f << "surface all size auto factor 2" << endl << endl; // auto
    //f << "surface all sizing function type skeleton scale 2.5
    //time_accuracy_level 2 min_size auto max_size auto max_gradient 1.5"
    //<< endl << endl;  adaptiv
    f << "surface all scheme pave" << endl << endl;
    f << "mesh surface all" << endl << endl;

    //*****************Im Abaqus-Format abspeichern*******************
    f << "block 1 surface all" << endl << endl;
    f << "export abaqus '" << work_dir << "/vmesh/Import/Cubit
    /surface_mesh.inp' block 1 overwrite" << endl << endl;

    //******************Oberflaechennetz loeschen*********************
    f << "delete mesh surface all propagate" << endl << endl;

    //***********************Netzglaettung****************************
    f << "import abaqus mesh geometry '" << work_dir << "/vmesh/Import/
    Cubit/surface_mesh.inp'" << endl << endl;
    f << "surface all smooth scheme laplacian free" << endl << endl;
    f << "smooth surface all" << endl << endl;

    //*****************Im Abaqus-Format abspeichern*******************
    f << "export abaqus '" << work_dir << "/vmesh/Import/Cubit/
    surface_mesh.inp' block all overwrite" << endl << endl;
    f << "exit" << endl << endl;

    f.close();
    return 0;
}
```

4.6 Offsetalgorithmus

Um die Voraussetzungen an das Gesamtvolumennetz, insbesondere der inneren Schicht zu erfüllen, wurde ein Algorithmus zur Generierung einer an den Schiffsrumpf anliegenden Grenzschicht entwickelt und in C++ implementiert. Dieser Algorithmus ist darüber hinaus so allgemein gehalten, dass auch andere Geometrien damit vernetzt werden können, z.B. Autos oder Flugzeuge.
Die Schwiereigkeiten bei der Generierung einer solchen Grenzschicht liegen zum Teil darin die Normalen, wenn sie überhaupt existieren, auch an den Kanten der zu vernetzenden Geometrie zu berechnen. Im Folgenden wird die Normale eines Punktes auf der Oberfläche (mit oder ohne Kante) als "Pseudonormale" bezeichnet.

4.6.1 Voraussetzungen und Ablauf

Um den Algorithmus anwenden zu können, muss für die Geometrie ein Oberflächennetz generiert werden. Dazu wird gegebenenfalls die Geometrie vernäht, falls sie aus mehreren Surfaces besteht und dann mit einem Paving-Algorithmus vernetzt. Das erhaltene Oberflächennetz sollte im Abaqus-Format (4.3.6) vorliegen.

Anschließend werden zu jedem Knoten des Oberflächennetzes bestimmte Nachbarzellen gesucht (Abb.: 4.13) bzw. (Abb.: 4.14). Zu jeder der Nachbarzelle wird dann ihre Normale berechnet. Aus all den Normalen wird danach die Pseudonormale für jeden Knoten berechnet. Auf diese Art wird für jeden Knoten eine Pseudonormale generiert (Abb.: 4.13). Mit Hilfe dieser Pseudonormalen wird dann die Grenzschicht erzeugt.

4.6.2 Algorithmus

Die Menge der Knoten bezeichnen wir mit V und die Menge der Zellen mit C.
Jede Zelle $c \in C$ besteht aus vier Knoten $\{v_1^c, v_2^c, v_3^c, v_4^c\} =: V_c \subseteq V$.
Es werden für alle Knoten $i \in V$ die p_i Nachbarzellen $C_i := \{c_{i,1}, c_{i,2}, ..., c_{i,p_i}\} \subseteq C$ $(p_i \in \mathbb{N})$ gesucht. Dabei soll jeder Knoten i Element von $V_{c_{i,j}}$ sein, wobei $j \in \{1, ..., p_i\}$ ist. Für alle direkten Nachbarzellen $c_{i,j}$ der Knoten i berechnet man die Normale $\mathbf{n}_{i,j}$ (siehe Abbildung 4.13). Mit diesen Normalen läßt sich dann die Pseudonormale n_i mit

$$\mathbf{n}_i := \sum_{j=1}^{p_i} (\mathbf{n}_{i,j})/p_i \tag{4.6.1}$$

berechnen. Mit der Pseudonormale für jeden Knoten, läßt sich dann die innere Schicht generieren (Kapitel 4.7).

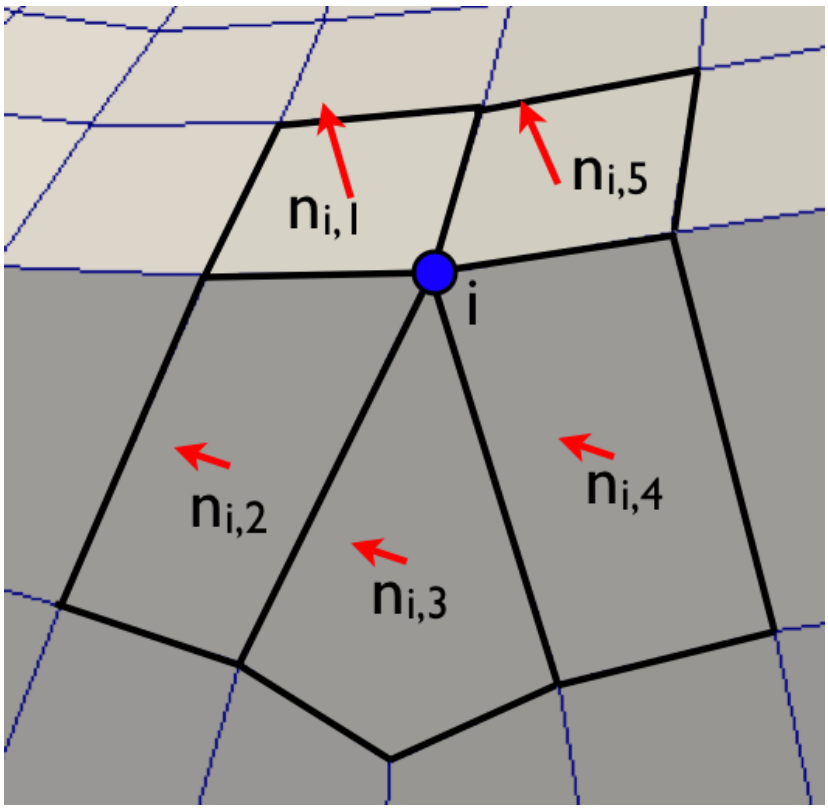

Abbildung 4.13: Offsetalgorithmus

Der Offsetalgorithmus hat die folgende Gestalt.

Algorithm 1 Offsetalgorithmus

for all $i \in V$ **do**
  search C_i
end for
for all $c_{i,j} \in C_i$ **do**
  compute $n_{i,j}$
end for
for all $i \in V$ **do**
  compute $\mathbf{n}_i := \sum_{j=1}^{p_i} (\mathbf{n}_{i,j})/p_i$
end for
for all $c \in C$ **do**
  generate the internal layer
end for

4.6.3 Erweiterung des Algorithmus

Um den Algorithmus zu verbessern, kann es sinnvoll sein, die Anzahl der Nachbarzellen bei der Berechnung der Pseudonormalen, zu vergrößern.
Zuerst wird die Menge der Knoten $V_i^2 := \cup_{j=1}^{p_i} V_{c_{i,j}}$ betrachtet. Danach werden alle Zellen $C_i^2 := \{c_{i,1}, ..., c_{i,q}\} \subseteq C \quad (q \in \mathbb{N})$ gesucht, die mindestens einen Knoten aus V_i^2 besitzen (siehe Abbildung 4.14).

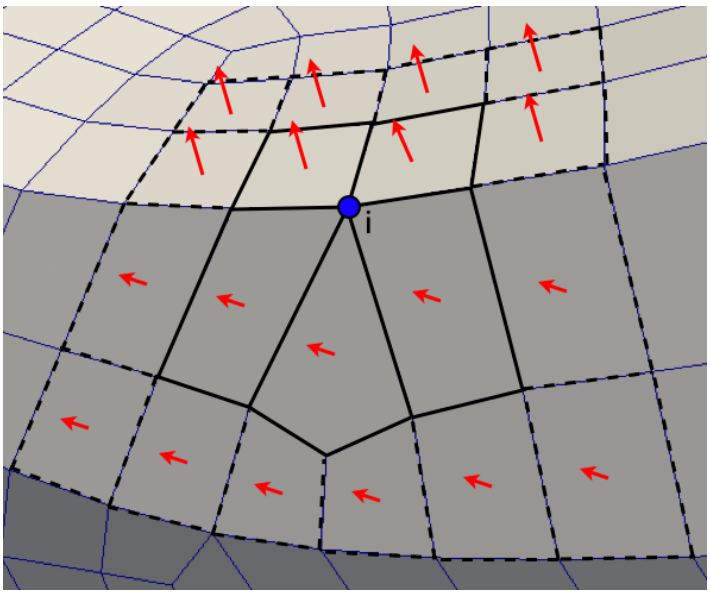

Abbildung 4.14: Verbesserung des Offsetalgorithmus

Der Algorithmus läuft dann wie folgt ab:

Algorithm 2 Offsetalgorithmus

for all $i \in V$ **do**
  search C_i^2
end for
for all $c_{i,j} \in C_i^2$ **do**
  compute $n_{i,j}$
end for
for all $i \in V$ **do**
  compute $\mathbf{n}_i := \sum_{j=1}^{q} (\mathbf{n}_{i,j})/q$
end for
for all $c \in C$ **do**
  generate the internal layer
end for

Mehr Zellen bei der Berechnung der Pseudonormalen für einen Knoten zu nehmen, bietet den Vorteil, dass Gitterlinien, die einem Knick angrenzen, sich diesem anpassen (Abb.: 4.15).
Ohne die Erweiterung des Offset-Algorithmus sind die benachbarten Gitterlinien eines Knicks noch senkrecht zur Oberfläche (linkes Bild von Abbildung 4.15).
Durch die Erweiterung passen sich genau diese Gitterlinien (rote Linien im rechten Bild von Abbildung 4.15) dem Knick an.

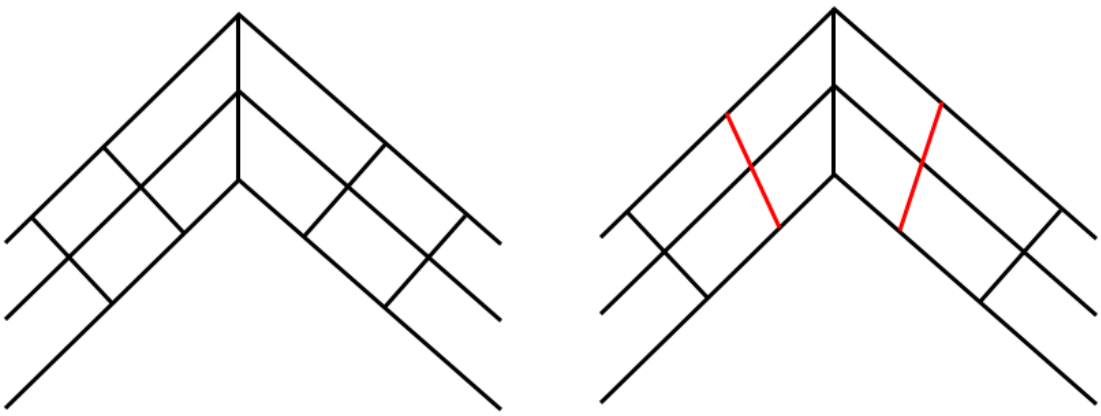

Abbildung 4.15: Veranschaulichung der Verbesserung des Offset-Algorithmus

Der Offset-Algorithmus wurde in folgender Funktion im Programm **vmesh_v29.2.cpp** realisiert. Das Oberflächennetz im Abaqus-Format wird mit den beiden folgenden Funktionen *surfacemesh inp to vmesh* und *mesh2d to matrix* geeignet in die Matrizen *node2d*

und $ele2d$ abgelegt. Im folgenden soll mit $node2d(i, j)$ bzw. $ele2d(i, j)$ das Element der i-ten Zeile und j-ten Spalte gemeint sein, wobei die Nummerierung bei Null beginnt. (Die dazu verwendete Klasse *jmatrix* ist im Modellierungstool *PaShiMo* [47] enthalten.)

4.6.4 Besonderheiten des Offset-Algorithmus

Mit dem Offset-Algorithmus lassen sich für sehr komplexe Geometrien hochwertige Grenzschichten generieren. Grundlegend und neu ist, dass als Basis für die Generierung ein Oberflächennetz genommen wird. Somit lassen sich auch an Kanten "Pseudonormale" berechnen. Algorithmen hierzu sind aus der Literatur nicht bekannt.
Versuche mit kommerziellen Vernetzern für die gleichen Geometrien automatisiert solche Grenzschichten zu generieren sind entweder gescheitert, oder es war nur mit sehr viel mühevoller Handarbeit möglich.
In Kapitel 6.2.3 wird deutlich, wie wichtig die innere Schicht für die Ergebnisse der Strömungsberechnungen sind. Die gute Qualität der inneren Schicht wird durch den Offset-Algorithmus gewährleistet, was durch die Auswertung der Gitterkriterien der inneren Schicht in Unterabschnitt 4.7.1 untermauert wird.
Eine andere Art einen Offset zu erzeugen wird in [40] beschrieben.

4.7 Generierung der inneren Schicht

Bei der Generierung der inneren Schicht sollte ein Algorithmus entwickelt werden, der geometrieunabhängig mehrere Grenzschichten aus Hexaedern generiert. Diese Grenzschichten sind essentiell für das Erhalten aussagekräftiger Ergebnisse der Strömungsberechnungen (Kapitel 3.3 und Kapitel 6.2.3). Hexaeder liefern bei der verwendeten Finite-Volumen-Methode bessere Resultate als Tetraeder bei der Strömungssimulation. Außerdem ist es sehr wichtig in diesem Bereich der Grenzschichten fein auflösen zu können, da es strömungstechnisch in dieser Grenzschicht Wirbelbildung und Turbulenzen gibt. Das Ziel ist es den Diskretisierungsfehler so gering wie möglich zu halten.

Die Generierung der inneren Schicht ist im Programmteil **vmesh_v29.2.cpp** realisiert. Das Oberflächennetz im Abaqus-Format wird mit den beiden folgenden Funktionen *surfacemesh inp to vmesh* und *mesh2d to matrix* geeignet in die Matrizen $node2d$ und $ele2d$ abgelegt. Im folgenden soll mit $node2d(i, j)$ bzw. $ele2d(i, j)$ das Element der i-ten Zeile und j-ten Spalte gemeint sein, wobei die Nummerierung bei Null beginnt.
Für die weitere Vernetzung wird das Oberflächennetz des Vollmodells benötigt. Dadurch kann völlig geometrieunabhängig vernetzt werden. Detailliert wird darauf in Abschnitt 4.9 eingegangen. Um von dem Oberflächennetz für das Halbmodell einer Geometrie das Oberflächennetz für das Vollmodell zu erhalten, wird das Netz zuerst an der Schnittebene gespiegelt. Dadurch sind die Knoten auf der Symmetrieebene doppelt vorhanden, weshalb eine Routine implementiert wurde, die diese Knoten in der Knotendatei aufspürt und dann in der Zelldatei so ändert, dass keine Knoten mehr doppelt sind (siehe Algorithmus 3). Dieser Algorithmus ist in der Funktion *vmerge_full_model*

Algorithm 3 Vmerge-Algorithmus

```
for i = 0 to Anzahl Knoten-1 do
  if node2d(m, 0) nicht markiert then
    for n = m + 1 to Anzahl Knoten do
      if node2d(m, 1) = node2d(n, 1) und
      node2d(m, 2) = node2d(n, 2) und
      node2d(m, 3) = node2d(n, 3) then
        for k = 0 to Anzahl Zellen-1 do
          for l = 1 to 4 do
            if node2d(n, 0) = ele2d(k, l) then
              Ersetze ele2d(k, l) durch node2d(m, 0)
            end if
          end for
        end for
        Markiere node2d(n, 0)
      end if
    end for
  end if
end for
```

umgesetzt. Liegt die Geometrie schon als Vollmodell vor, muss das Oberflächennetz nicht mehr gespiegelt werden. Die entsprechenden Einstellungen dazu finden sich in Abschnitt 4.12. Dass das Oberflächennetz als Vollmodell vorliegt, ist wichtig für den Offset-Algorithmus (Abschnitt 4.6). Ausgangspunkt (Import) des Vernetzers ist immer ein solches Oberflächennetz der zu vernetzenden Geometrie als Vollmodell.

Mit Hilfe dieses Oberflächennetzes wird die innere Schicht generiert, welche ausschließlich aus Hexaedern besteht und der Geometrie direkt anliegt. Zu diesem Zwecke wurde ein Offset-Algorithmus entwickelt, welcher zu jedem Knoten $\mathbf{P}^0(i,k)$ des Oberflächennetzes eine "Pseudonormale" $\mathbf{n}(i,k)$ berechnet und welcher in der Funktion *offset_algo* implementiert ist. $\mathbf{P}^j(i,k)$ steht dabei für den k-ten Knoten der i-ten Zelle der j-ten Schicht. Und $\mathbf{n}(i,k)$ steht dabei für den k-ten Knoten der i-ten Zelle der 0-ten Schicht. Von diesem Oberflächennetz wird die Anzahl der Zellen (Z_2D) in den Funktionen *surfacemesh inp to vmesh* und *mesh2d to matrix* bestimmt. Mit folgendem Algorithmus werden dann die Knoten der inneren Schicht berechnet, wobei num_of_lay für die Anzahl der Schichten der innere Schicht, d für die Dicke der ersten Schicht steht und v ein Verzerrungsfaktor ist.

Algorithm 4 Berechnung der Koordinaten der Knoten der inneren Schicht

for $j = 0$ to $num\_of\_lay - 1$ **do**
  for $i = 0$ to $Z\_2D - 1$ **do**
    for $k = 0$ to 3 **do**
      $\mathbf{P}^j(i,k) = \mathbf{P}^0(i,k) + d \cdot j \cdot \mathbf{n}(i,k) \cdot v^j$
    end for
  end for
end for

Die acht Knoten $K_1, ..., K_8$ einer zugehörigen Zelle werden folgendermaßen berechnet:

Algorithm 5 Berechnung der dazugehörigen Zellinformationen

for $j = 0$ to $num\_of\_lay - 1$ **do**
  for $i = 0$ to $Z\_2D - 1$ **do**
    $K_1 = 4i + 1 + 4j \cdot Z\_2D$
    $K_2 = 4i + 2 + 4j \cdot Z\_2D$
    $K_3 = 4i + 3 + 4j \cdot Z\_2D$
    $K_4 = 4i + 4 + 4j \cdot Z\_2D$
    $K_5 = 4i + 1 + 4(j+1) \cdot Z\_2D$
    $K_6 = 4i + 2 + 4(j+1) \cdot Z\_2D$
    $K_7 = 4i + 3 + 4(j+1) \cdot Z\_2D$
    $K_8 = 4i + 4 + 4(j+1) \cdot Z\_2D$
  end for
end for

Die Umsetzung der Algorithmen 4 und 5 ist in der Funktion *int_lay* zu finden.

Anschließend werden die Pyramiden generiert. Die Koordinaten der Pyramidenspitzen $\mathbf{s}$ berechnen sich aus dem Mittelpunkt $\mathbf{m}$ der Grundflächen, der zugehörigen Normalen $\mathbf{n}$, der Zellgröße zg und dem Streckungsfaktor *scal* wie folgt:

$$\mathbf{s} = \mathbf{m} + zg \cdot scal \cdot \mathbf{n}$$

Der Streckungsfaktor *scal* ist auf 0.2 eingestellt. Die Generierung der Pyramiden ist in der Funktion *pyramide_int* realisiert.

Die Nummerierung der Knoten und der Zellen erfolgt jeweils über entsprechende Zählvariablen.

4.7.1 Auswertung der Gitterkriterien bei der inneren Schicht

Knuckle:

Dazu wurde eine von **vmesh** generierte innere Schicht des Knuckle herangezogen. In Abbildung 6.28 ist dieses Netz dargestellt.

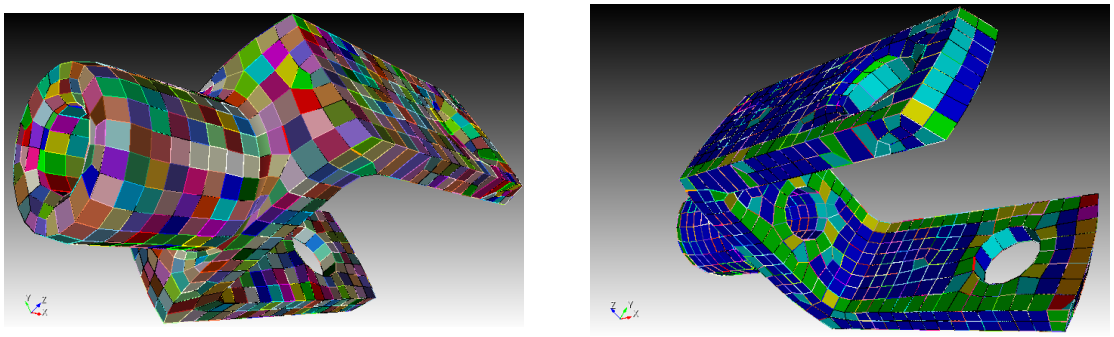

Abbildung 4.16: Netz der inneren Schicht des Knuckle

Dieses Netz wurde auf die verschiedenen Kriterien für Hexaeder, welche in Kapitel 2.1.5 hergeleitet wurden, getestet. Sind diese Kriterien erfüllt, dann ist das ein Hinweis dafür, dass die Zellen nicht entartet sind.

Diagonalenverhältnis:

Für das Diagonalenverhältnis ist $[0.65, 1]$ ein akzeptables Intervall. In Abbildung 6.29 ist das deutlich erfüllt. Wie in Abbildung 4.17 erkennbar ist, ist das Diagonalenverhältnis nahezu aller Zellen der inneren Schicht im Intervall $[0.65, 1]$ enthalten.

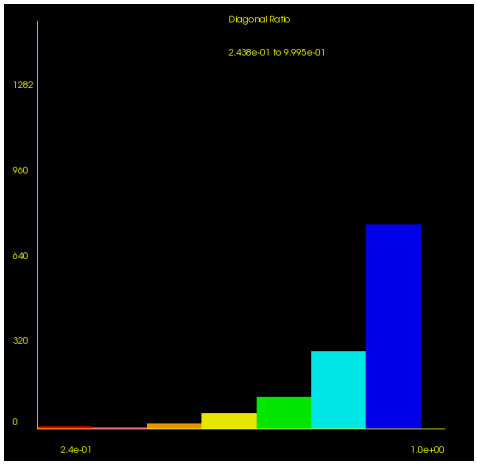

Abbildung 4.17: Säulendiagramm zum Diagonalenverältnis

Shape:

Ein akzeptables Intervall für das Shape ist $[0.3, 1]$. Laut Abbildung 6.30 trifft dies zu.

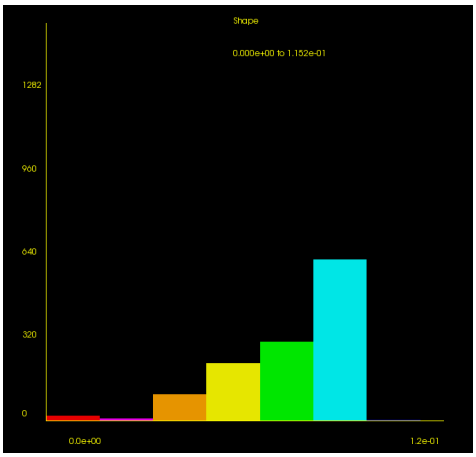

Abbildung 4.18: Säulendiagramm zum Shape

Skalierte Jacobi-Determinante:

Die skalierte Jacobi-Determinante sollte im Intervall $[\frac{1}{2}, 1]$ liegen. In Abbildung 6.31 ist das ebenfalls erfüllt.

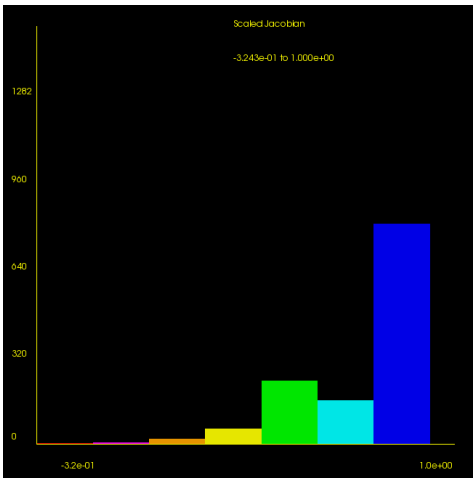

Abbildung 4.19: Säulendiagramm zur skalierten Jacobi-Determinante

Voith Wassertrecker ohne Finne (Voith):

Ein Beispiel der inneren Schicht des Voith Wassertrecker ohne Finne (Voith), welche von **vmesh** generiert wurde, ist in Abbildung 4.20 dargestellt.

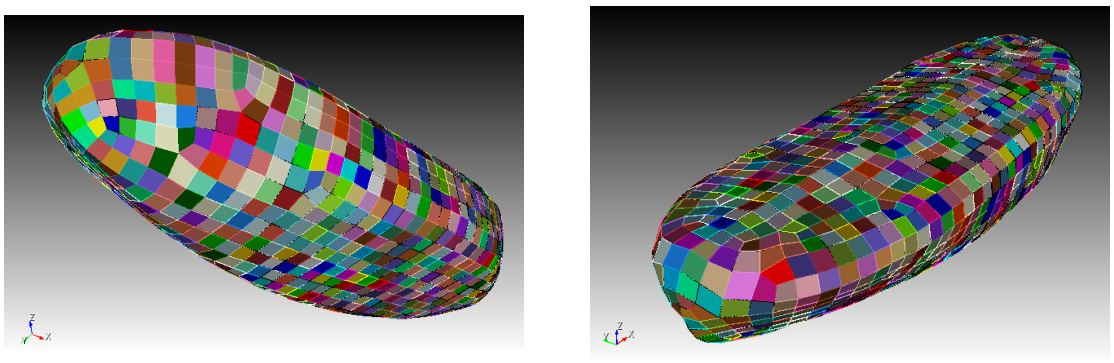

Abbildung 4.20: Netz der inneren Schicht des Voith Wassertrecker ohne Finne (Voith)

Dieses Netz wurde auf die verschiedenen Kriterien für Hexaeder, welche in Kapitel 2.1.5 hergeleitet wurden, getestet.

Diagonalenverhältnis:

Für das Diagonalenverhältnis ist $[0.65, 1]$ ein akzeptables Intervall . In Abbildung 4.21 (links) ist das deutlich erfüllt.

Skalierte Jacobi-Determinante:

Die skalierte Jacobi-Determinante sollte im Intervall $[\frac{1}{2}, 1]$ liegen. In Abbildung 4.21 (rechts) ist das ebenfalls erfüllt.

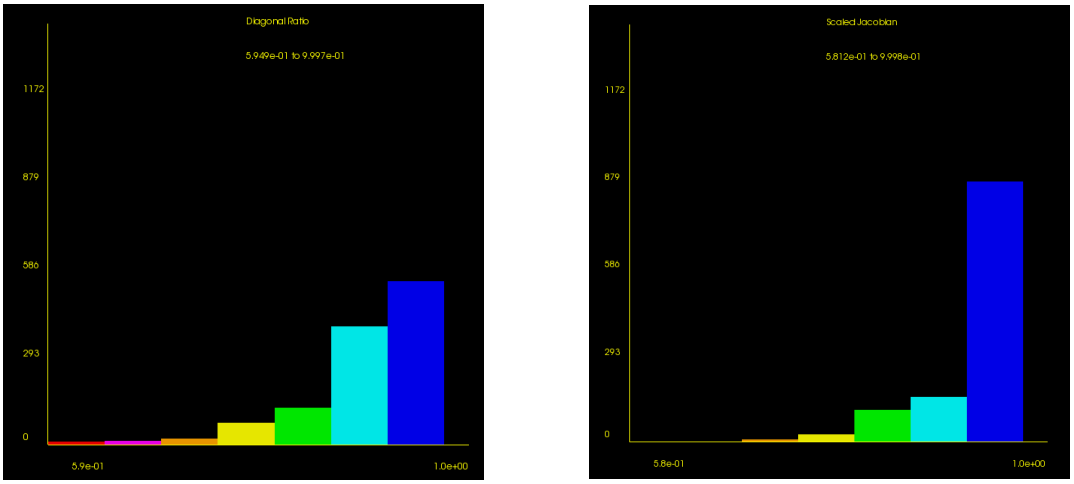

Abbildung 4.21: Links: Säulendiagramm zum Diagonalenverältnis; Rechts: Säulendiagramm zur skalierten Jacobi-Determinante

Voith Wassertrecker mit Finne (Voith):

Ein Beispiel der inneren Schicht des Voith Wassertrecker_v107 (*PAShiMo*), welche von **vmesh** generiert wurde, ist in Abbildung 4.22 dargestellt.

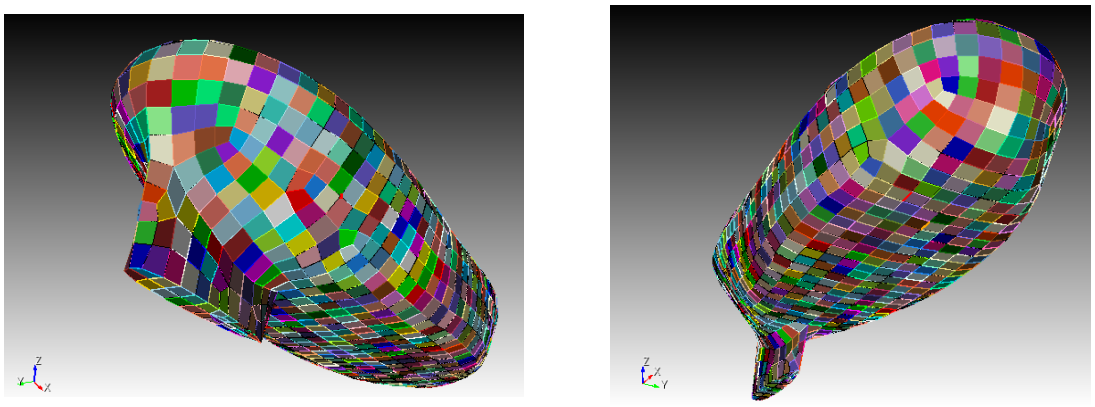

Abbildung 4.22: Netz der inneren Schicht des Voith Wassertrecker mit Finne (Voith)

Dieses Netz wurde auf die verschiedenen Kriterien für Hexaeder, welche in Kapitel 2.1.5 hergeleitet wurden, getestet.

Diagonalenverhältnis:

Für das Diagonalenverhältnis ist $[0.65, 1]$ ein akzeptables Intervall . In Abbildung 4.23 (links) ist das deutlich erfüllt.

Skalierte Jacobi-Determinante:

Die skalierte Jacobi-Determinante sollte im Intervall $[\frac{1}{2}, 1]$ liegen. In Abbildung 4.23 (rechts) ist das ebenfalls erfüllt.

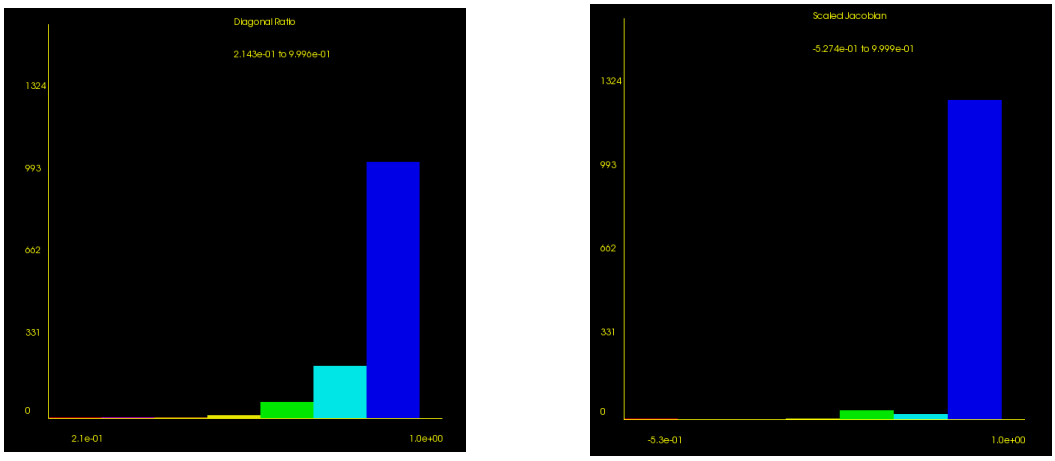

Abbildung 4.23: Links: Säulendiagramm zum Diagonalenverältnis; Rechts: Säulendiagramm zur skalierten Jacobi-Determinante

Voith Wassertrecker_v107:

Ein Beispiel der inneren Schicht des Voith Wassertrecker_v107 (*PAShiMo*), welche von **vmesh** generiert wurde, ist in Abbildung 4.24 dargestellt.

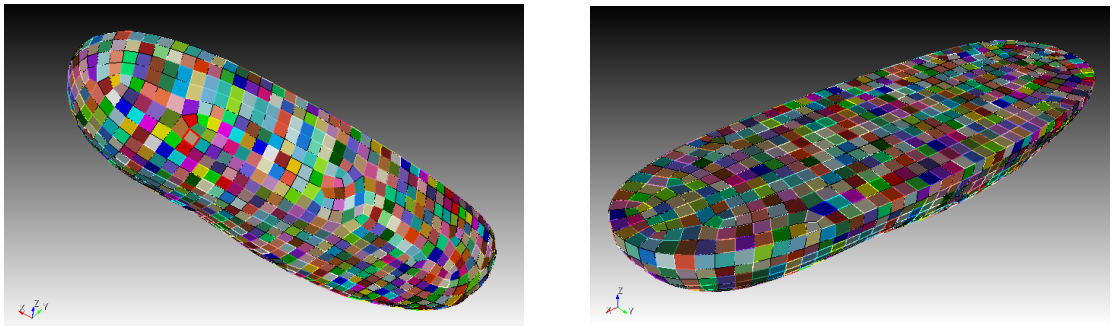

Abbildung 4.24: Netz der inneren Schicht des Voith Wassertrecker_v107

Dieses Netz wurde auf die verschiedenen Kriterien für Hexaeder, welche in Kapitel 2.1.5 hergeleitet wurden, getestet.

Diagonalenverhältnis:

Für das Diagonalenverhältnis ist $[0.65, 1]$ ein akzeptables Intervall . In Abbildung 4.25 ist das deutlich erfüllt.

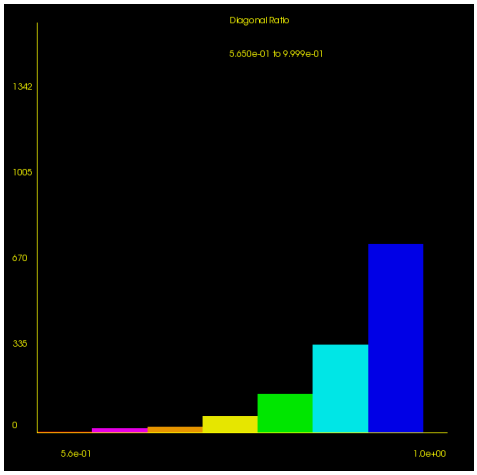

Abbildung 4.25: Säulendiagramm zum Diagonalenverältnis

Shape:

Ein akzeptables Intervall für das Shape ist $[0.3, 1]$. Laut Abbildung 4.26 trifft dies zu.

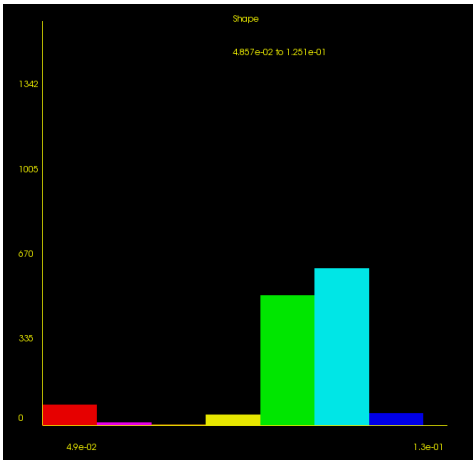

Abbildung 4.26: Säulendiagramm zum Shape

Skalierte Jacobi-Determinante:

Die skalierte Jacobi-Determinante sollte im Intervall $[\frac{1}{2}, 1]$ liegen. In Abbildung 4.27 ist das ebenfalls erfüllt.

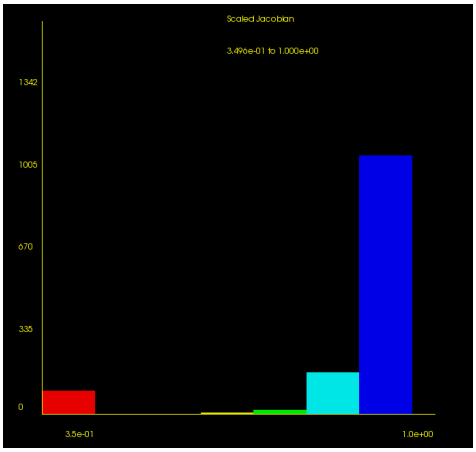

Abbildung 4.27: Säulendiagramm zur skalierten Jacobi-Determinante

Die Zellen, welche in diesem Beispiel außerhalb des Intervalls liegen, sind für die Strömungssimulation nicht relevant, da diese über der Wasseroberfläche liegen.

4.7.2 Weitere visuelle Ergebnisse der inneren Schicht (Offset-Algorithmus)

Das Resultat des Offsetalgorithmus für die Grenzschicht auf der Geometrie ist auch für komplexere Geometrien sehr gut. Das sollen folgende Abbildungen belegen.

Knuckle

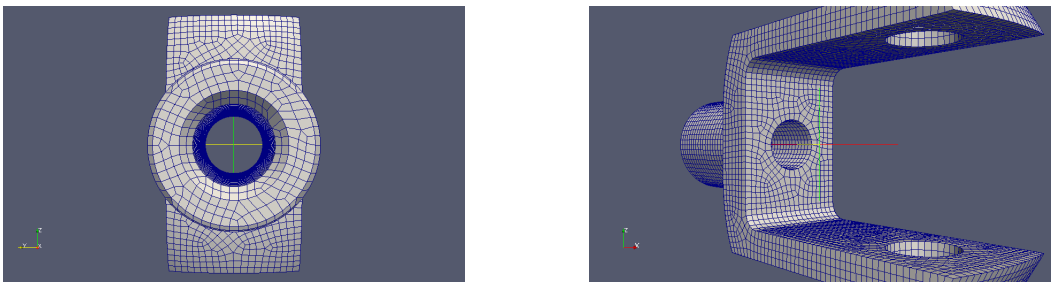

Abbildung 4.28: Knuckle

Voith Wassertrecker

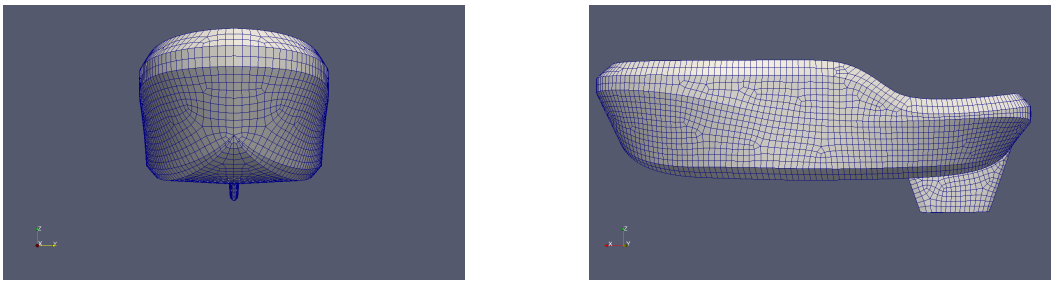

Abbildung 4.29: Voith Wassertrecker

Container-Schiff

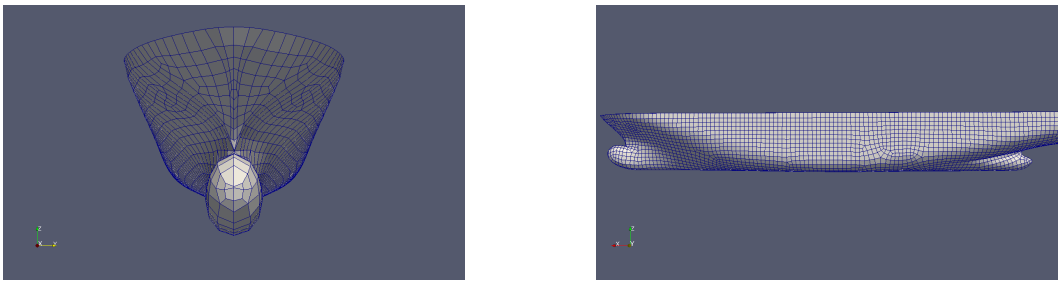

Abbildung 4.30: Container-Schiff

4.8 Generierung der äußeren Schicht

Dieser Abschnitt ist im Programmteil **vmesh_v29.2.cpp** realisiert.
Die äußere Schicht als Halbmodell teilt sich in vier Quader auf (Abb.: 4.31). Die Bezeichnungen der Boxberandung sind in Abbildung 4.32 dargestellt. Dann gilt für die Volumen V_i der einzelnen Quader:

$$\begin{aligned} V_1 &= |(ph - hinten) \cdot seite \cdot (oben - unten)| \\ V_2 &= |(pv - ph) \cdot seite \cdot (pu - unten)| \\ V_3 &= |(pv - ph) \cdot (seite - ps) \cdot (oben - pu)| \\ V_4 &= |(vorne - pv) \cdot seite \cdot (oben - unten)| \end{aligned}$$

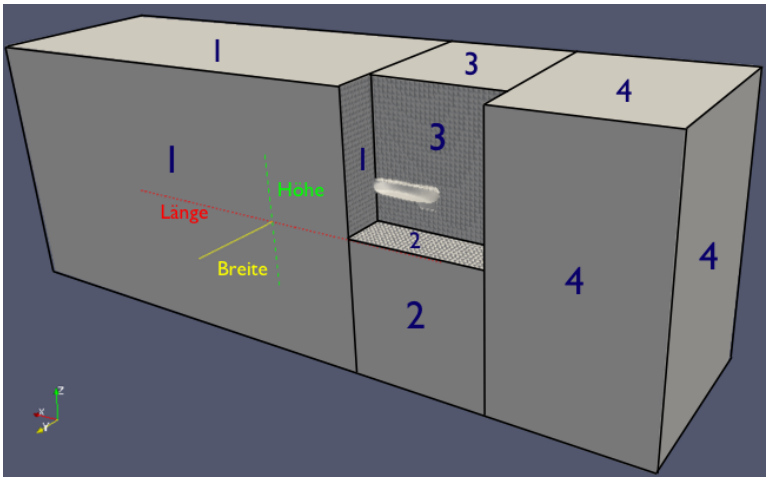

Abbildung 4.31: Aufteilung des Halbmodells in Quader

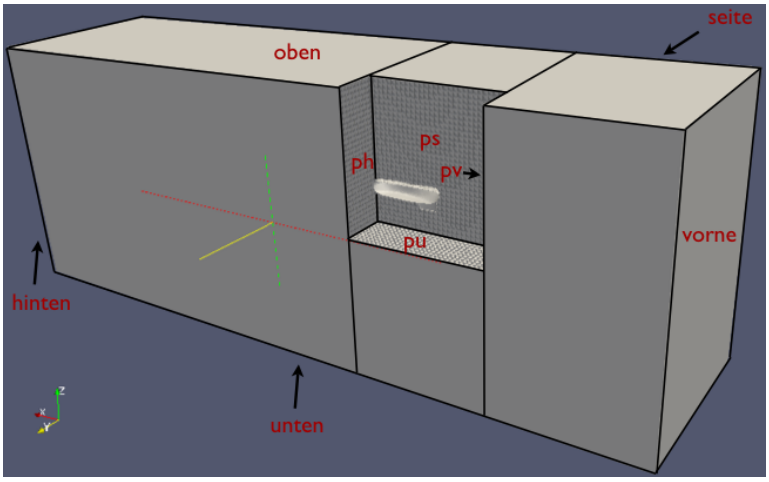

Abbildung 4.32: Bezeichnungen der Boxberandung

Vor der Vernetzung müssen für die Berandung und die Zellgröße (zg) die Einstellungen getroffen werden (Abb.: 4.32) (Abschnitt 4.12). Daraus werden dann folgende Parameter berechnet:

$$\begin{aligned} anzbreite &= \lfloor (seite)/zg + 0.5 \rfloor \\ anzps &= \lfloor (seite - ps)/zg \rfloor \\ anzpu &= \lfloor (pu - unten)/zg \rfloor \\ anzph &= \lfloor (ph - hinten)/zg \rfloor \\ anzpv &= \lfloor (vorne - pv)/zg \rfloor \\ anzmitte &= \lfloor (pv - ph)/zg + 0.5 \rfloor \\ anzhoehe &= \lfloor (oben - unten)/zg + 0.5 \rfloor \end{aligned}$$

Diese Parameter geben die Anzahl der Zellen längs der in Abbildung 4.33 markierten

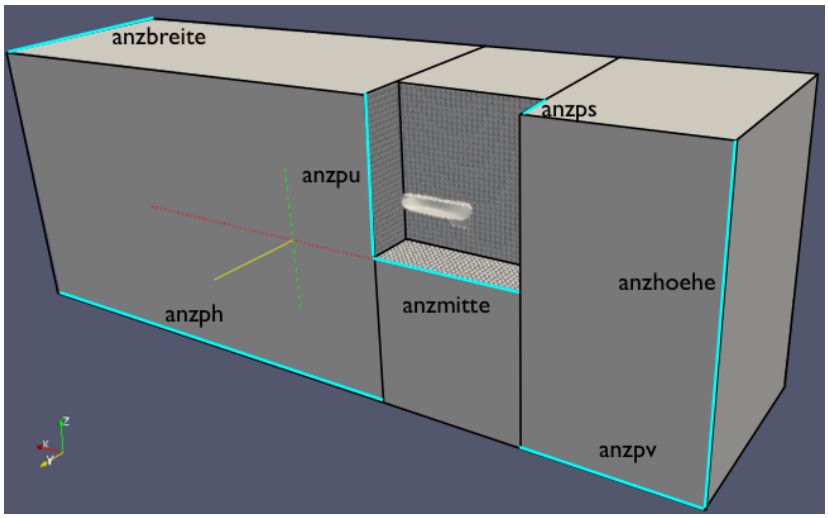

Abbildung 4.33: Parameter für die Anzahl der Zellen längs der markierten Seiten

Seiten an. Daraus lassen sich die Knoteninformationen der einzelnen Quader berechnen.

Berechnung der Koordinaten der Knoten der einzelnen Quader:

Der C++ Code zu folgenden Algorithmen ist in der Funktion *ext_lay* zu finden.

Algorithm 6 Berechnung der Koordinaten der Knoten für den Quader 1

for $l = 0$ to $anzph$ **do**
  for $b = 0$ to $anzbreite$ **do**
    for $h = 0$ to $anzoehe$ **do**
      $x = hinten + l \cdot zg$
      $y = b \cdot zg$
      $z = unten + h \cdot zg$
    end for
  end for
end for

Algorithm 7 Berechnung der Koordinaten der Knoten für den Quader 2

for $l = 0$ to $anzmitte$ **do**
  for $b = 0$ to $anzbreite$ **do**
    for $h = 0$ to $anzoehe$ **do**
      $x = hinten + (anzph + l) \cdot zg$
      $y = b \cdot zg$
      $z = unten + h \cdot zg$
    end for
  end for
end for

Algorithm 8 Berechnung der Koordinaten der Knoten für den Quader 3

for $l = 0$ to $anzmitte$ **do**
  for $b = 0$ to $anzbreite$ **do**
    for $h = 0$ to $anzoehe - anzpu$ **do**
      $x = hinten + (anzph + l) \cdot zg$
      $y = (anzbreite - b) \cdot zg$
      $z = unten + (anzpu + h) \cdot zg$
    end for
  end for
end for

Algorithm 9 Berechnung der Koordinaten der Knoten für den Quader 4

for $l = 0$ to $anzpv$ **do**
  for $b = 0$ to $anzbreite$ **do**
    for $h = 0$ to $anzoehe$ **do**
      $x = hinten + (anzph + anzmitte + l) \cdot zg$
      $y = b \cdot zg$
      $z = unten + h \cdot zg$
    end for
  end for
end for

Berechnung der Zellinformationen für die einzelnen Quader:

Die Umsetzung der folgenden Algorithmen ist in der Funktion *ext_lay* realisiert. Vorab werden noch die Anzahl der Knoten in den einzelnen Quadern bzw. von ausgewählten Seiten der Quader berechnet.

$$\begin{aligned} gesamtquad1 &= (anzbreite + 1)(anzhoehe + 1) \\ gesamtquad13D &= (anzbreite + 1)(anzhoehe + 1)(anzph + 1) \\ gesamtquad2 &= (anzbreite + 1)(anzpu + 1) \\ gesamtquad23D &= (anzbreite + 1)(anzpu + 1)(anzmitte + 1) \\ gesamtquad3 &= (anzps + 1)(anzhoehe - anzpu + 1) \\ gesamtquad33D &= (anzps + 1)(anzhoehe - anzpu + 1)(anzmitte + 1) \\ gesamtquad4 &= (anzbreite + 1)(anzhoehe + 1) \end{aligned}$$

Die Anzahl der Knoten der inneren Schicht $gesamtinnere3D$ wird bei deren Generierung bereits über eine Zählvariable ermittelt.

Algorithm 10 Berechnung der Zellinformationen für den Quader 1

for $l = 0$ to $anzph$ **do**
  for $m = 0$ to $anzbreite$ **do**
    for $n = 0$ to $anzoehe$ **do**
      $z1 = n + (anzhoehe + 1)(m - 1) + (gesamtquad1)(l - 1) + gesamtinnere3D$
      $z2 = z1 + 1$
      $z3 = z2 + anzhoehe + 1$
      $z4 = z1 + anzhoehe + 1$
      $z5 = z1 + gesamtquad1$
      $z6 = z2 + gesamtquad1$
      $z7 = z3 + gesamtquad1$
      $z8 = z4 + gesamtquad1$
    end for
  end for
end for

Algorithm 11 Berechnung der Zellinformationen für den Quader 2

for $l = 0$ to $anzmitte$ **do**
  for $m = 0$ to $anzbreite$ **do**
    for $n = 0$ to $anzpu$ **do**
      $z1 = n + (anzpu + 1)(m - 1) + (gesamtquad2)(l - 1) + gesamtinnere3D + gesamtquad13D$
      $z2 = z1 + 1$
      $z3 = z2 + anzpu + 1$
      $z4 = z1 + anzpu + 1$
      $z5 = z1 + gesamtquad2$
      $z6 = z2 + gesamtquad2$
      $z7 = z3 + gesamtquad2$
      $z8 = z4 + gesamtquad2$
    end for
  end for
end for

Algorithm 12 Berechnung der Zellinformationen für den Quader 3

for $l = 0$ to $anzmitte$ **do**
  for $m = 0$ to $anzbreite$ **do**
    for $n = 0$ to $anzhoehe - anzpu$ **do**
      $z1 = n + (anzhoehe - anzpu + 1)(m - 1) + (gesamtquad3)(l - 1) + gesamtinnere3D + gesamtquad13D + gesamtquad23D$
      $z2 = z1 + 1$
      $z3 = z2 + anzhoehe - anzpu + 1$
      $z4 = z1 + anzhoehe - anzpu + 1$
      $z5 = z1 + gesamtquad3$
      $z6 = z2 + gesamtquad3$
      $z7 = z3 + gesamtquad3$
      $z8 = z4 + gesamtquad3$
    end for
  end for
end for

Algorithm 13 Berechnung der Zellinformationen für den Quader 4

for $l = 0$ to $anzpv$ **do**
  for $m = 0$ to $anzbreite$ **do**
    for $n = 0$ to $anzhoehe$ **do**
      $z1 = n + (anzhoehe + 1)(m - 1) + (gesamtquad3)(l - 1) + gesamtinnere3D + gesamtquad13D + gesamtquad23D + gesamtquad33D$
      $z2 = z1 + 1$
      $z3 = z2 + anzhoehe + 1$
      $z4 = z1 + anzhoehe + 1$
      $z5 = z1 + gesamtquad4$
      $z6 = z2 + gesamtquad4$
      $z7 = z3 + gesamtquad4$
      $z8 = z4 + gesamtquad4$
    end for
  end for
end for

Berechnung der Zellinformationen der Pyramidenschicht:

Nachfolgende Algorithmen sind in der Funktion *pyramide_ext* implementiert.

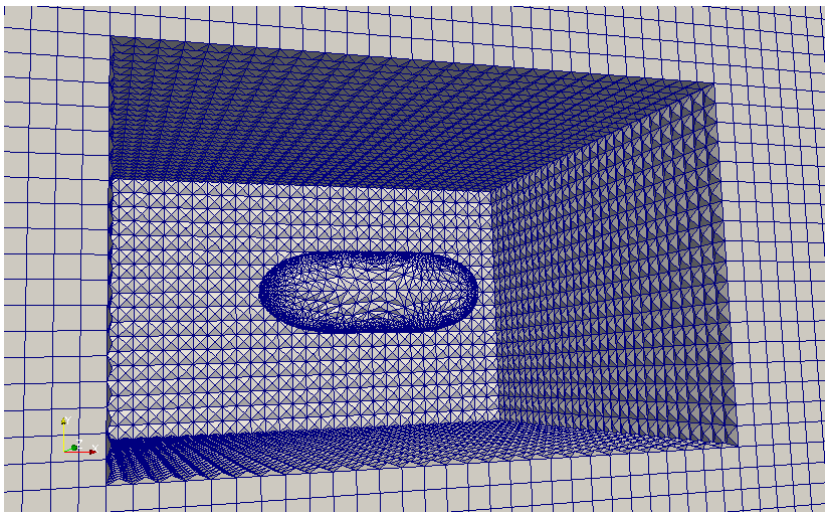

Abbildung 4.34: Netz der äußeren Schicht (2)

Nachdem die äußere Schicht generiert wurde, werden die dazugehörigen Pyramiden erzeugt (Abb.: 4.34). Diese werden auch zuerst für das Halbmodell generiert. Die vier Knotennummeren der Eckpunkte der Pyramidengrundflächen ergeben sich folgendermaßen:

Algorithm 14 Berechnung der Knotennummern der Eckpunkte der Pyramidengrundflächen für Quader 1

for $i = 0$ to $anzbreite - anzps$ **do**
  for $j = 0$ to $anzhoehe - anzpu$ **do**
    $z1 = j + (anzhoehe - anzpu + 1)(i - 1)$
    $z2 = z1 + 1$
    $z3 = z2 + anzhoehe - anzpu + 1$
    $z4 = z1 + anzhoehe - anzpu + 1$
  end for
end for

Algorithm 15 Berechnung der Knotennummern der Eckpunkte der Pyramidengrundflächen für Quader 2

for $i = 0$ to $anzmitte$ **do**
  for $j = 0$ to $anzbreite - anzps$ **do**
    $z1 = j + (anzbreite - anzps + 1)(i - 1) + (anzbreite - anzps + 1)(anzhoehe + 1 - anzpu)$
    $z2 = z1 + 1$
    $z3 = z2 + anzbreite - anzps + 1$
    $z4 = z1 + anzbreite - anzps + 1$
  end for
end for

Algorithm 16 Berechnung der Knotennummern der Eckpunkte der Pyramidengrundflächen für Quader 3

for $i = 0$ to $anzmitte$ **do**
  for $j = 0$ to $anzhoehe - anzpu$ **do**
    $z1 = j + (anzhoehe - anzpu + 1)(i - 1) + (anzbreite - anzps + 1)(anzhoehe + 1 - anzpu) + (anzbreite - anzps + 1)(anzmitte + 1)$
    $z2 = z1 + 1$
    $z3 = z2 + anzhoehe - anzpu + 1$
    $z4 = z1 + anzhoehe - anzpu + 1$
  end for
end for

Algorithm 17 Berechnung der Knotennummern der Eckpunkte der Pyramidengrundflächen für Quader 4

for $i = 0$ to $anzbreite - anzps$ **do**
  for $j = 0$ to $anzhoehe - anzpu$ **do**
    $z1 = j + (anzhoehe - anzpu + 1)(i - 1) + (anzbreite - anzps + 1)(anzhoehe + 1 - anzpu) + (anzbreite - anzps + 1)(anzmitte + 1) + (anzhoehe - anzpu + 1)(anzmitte + 1)$
    $z2 = z1 + 1$
    $z3 = z2 + anzhoehe - anzpu + 1$
    $z4 = z1 + anzhoehe - anzpu + 1$
  end for
end for

Die Koordinaten der Pyramidenspitzen $\mathbf{s}$ berechnen sich aus dem Mittelpunkt der Grundflächen $\mathbf{m}$, der zugehörigen Normalen $\mathbf{n}$, der Zellgröße zg und dem Streckungsfaktor $scal\_a$ wie folgt:

$$\mathbf{s} = \mathbf{m} + zg \cdot scal\_a \cdot \mathbf{n}$$

Der Streckungsfaktor $scal\_a$ ist auf 0.2 eingestellt.

Das Netz der äußeren Schicht und der Pyramidenschicht wird gespiegelt, so dass es als Vollmodell vorliegt (Abb.: 4.35).

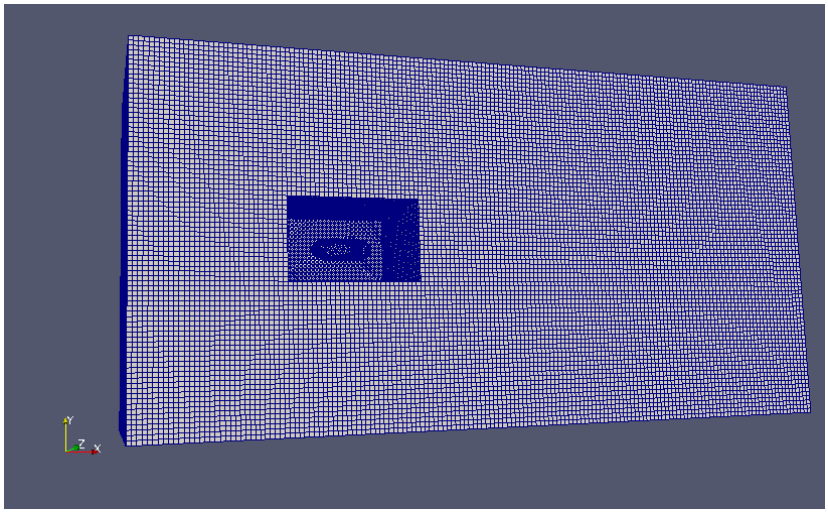

Abbildung 4.35: Netz der äusseren Schicht (1)

Die Gitterkriterien für Hexaeder für diese Schicht sind alle erfüllt, da nur Würfel in dieser Schicht verarbeitet wurden.

Die Nummerierung der Knoten und der Zellen erfolgt auch hier jeweils über entsprechende Zählvariablen.

4.9 Generierung der mittleren Schicht

Das Volumennetz der mittleren Schicht besteht aus Tetraedern. Diese Tetraeder werden mit dem TetMesher von CUBIT (Kapitel 2.5) generiert, welcher auf dem Delaunay-Algorithmus (Kapitel 2.3) basiert. Um das Volumennetz der mittleren Schicht generieren zu können, benötigt der Vernetzer, in unserem Fall CUBIT, die Grenzflächen des zu vernetzenden Volumens in einem Format, welches das Programm lesen und bearbeiten kann. In unserem Fall verwenden wir das STL-Format (Abschnitt 4.3.1). Die Grenzflächen der mittleren Schicht bestehen ausschließlich aus Dreiecken, den Seiten der aufgesetzten Pyramiden und einem Deckel (Abb.: 4.36). Bei der Generierung der Pyramiden der inneren und äußeren Schicht wurden die Eckpunkte der Dreiecke in eine Matrix abgelegt. Für die Generierung des STL-Formats wird nur noch die Normale der einzelnen Dreiecke mit dem Kreuzprodukt berechnet und das ganze wird dann noch in der Datei *stl_middle.stl* im STL-Format abgespeichert. Das Programm **vmesh_v29.2.cpp** erzeugt in der Funktion *acis* eine Datei *middle* aus Befehlen, welche von CUBIT importiert wird. Die Ausführung der Befehlskette führt zur Vernetzung der mittleren Schicht.

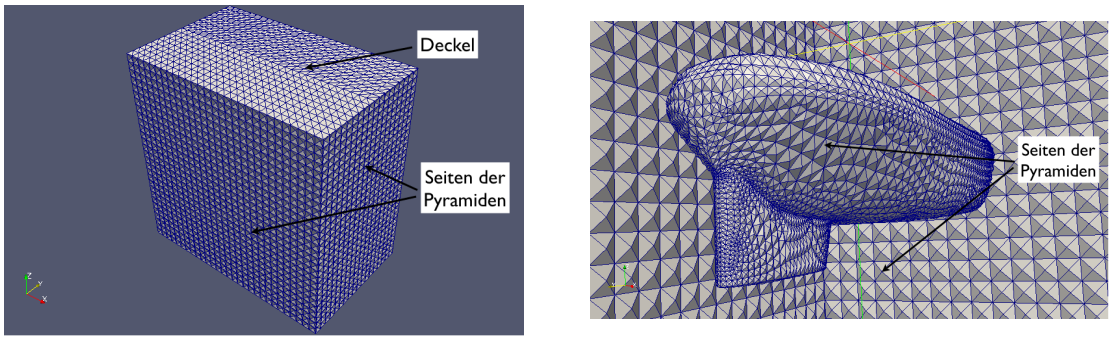

Abbildung 4.36: Mittlere Schicht als Vollmodell im STL-Format

Das Vollmodell ist an dieser Stelle dem Halbmodell überlegen. Damit das Vollmodell der mittleren Schicht ein abgeschlossenes Volumen ist, muss nur zusätzlich ein Deckel aus Dreiecken generiert werden. So bleibt man komplett geometrieunabhängig, da dieser Deckel keinerlei Informationen der Geometrie benötigt. Ganz anders sieht das beim Halbmodell aus. Hier müsste man um eine abgeschlossene Oberfläche der mittleren Schicht zu erhalten, an der Symmetrieebene die Aussparung mit Dreiecken zupflastern (Abb.: 4.37). Dieses Zupflastern ist aber geometrieabhängig und sehr schwer umzusetzen. Dies wäre eine große Einschränkung, da viele Geometrien dann nicht mehr automatisch vernetzbar wären. Durch die Wahl für das Vollmodell ergeben sich sicherlich größere Netze und damit verbunden längere Rechenzeiten bei der CFD-Rechnung.

Dies ist für einen Optimierungslauf ein Nachteil. Jedoch verläuft die Vernetzung komplett geometrieunabhängig. Zudem wird im Vollmodell die Karmansche Wirbelstraße auch aufgelöst, was jedoch fast immer irrelevant ist.

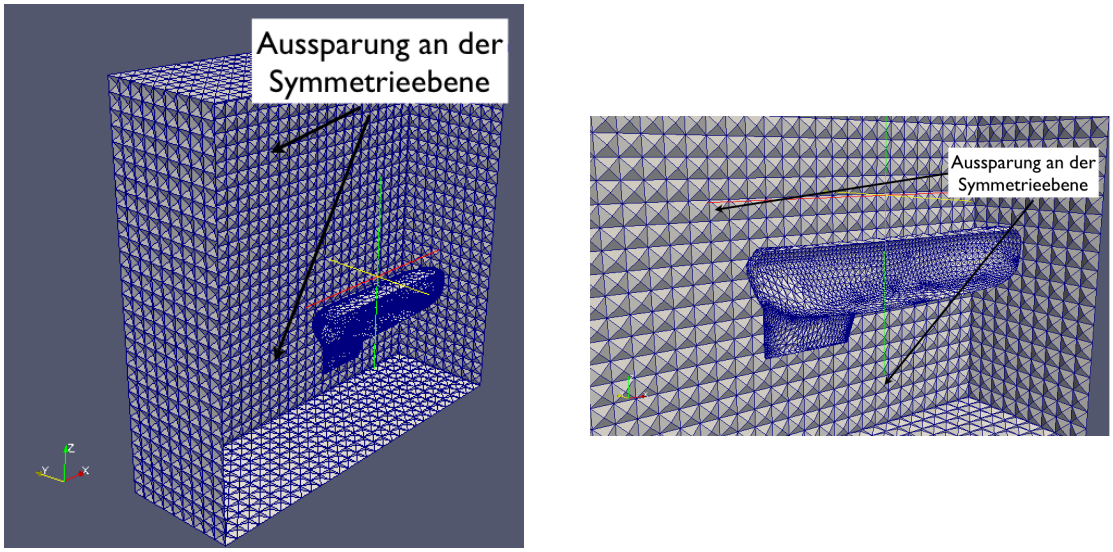

Abbildung 4.37: Mittlere Schicht als Halbmodell (nicht geschlossen) im STL-Format

Funktion *acis*:

(*vmesh/vmesh_v29.2.cpp*)

```
void acis(jmatrix &stl, int laufstl, int ab, int precision, int
    stl_middle_pyramide_on, int acis_on, string path)
{
    if (stl_middle_pyramide_on==1 && acis_on==1)
    {
        fstream f;

        f.open("./Export/Cubit/middle", ios::out);

//**************************Vernetzungsart*******************************

        f << "import stl '" << path << "vmesh/Export/STL/stl_middle.stl'
        feature_angle 135.00 linear merge make_elements" << endl;

        f << "volume all interval 1" << endl;

        f << "volume all scheme tetmesh" << endl;

        f << "mesh volume all" << endl;

//*******************************smoothing*******************************
```

```
        //f << "volume all smooth scheme equipotential" << endl;

        f << "volume all smooth scheme condition number beta 2.0 cpu 20"
           << endl;

//*****************************************************************

        f << "smooth volume all" << endl;

        f << "save as '" << path << "vmesh/Export/Cubit/middle_meshed.cub
            ';
        overwrite" << endl;

        f << "export abaqus '" << path << "vmesh/Import/Cubit/
        volume_mesh_middle.inp' overwrite" << endl;

        f << "exit" << endl;

        f.close();
    }
}
```

Das Netz der mittleren Schicht wird nach seiner Generierung noch mit einem Laplace-Algorithmus geglättet (siehe Kapitel 2.1.6).

Nach der Generierung der mittleren Schicht, werden alle drei Schichten vereinigt. Dies ist im Programmteil **vmesh_complete.cpp** realisiert.

4.9.1 Auswertung der Gitterkriterien bei der mittleren Schicht

Hierzu wurde als Referenznetz das Netz genommen, welches bei den Vergleichsrechnungen (Kapitel 6.2.1) die besten Ergebnisse gab. In Abbildung 4.38 ist ein Netz der mittleren Schicht dargestellt.

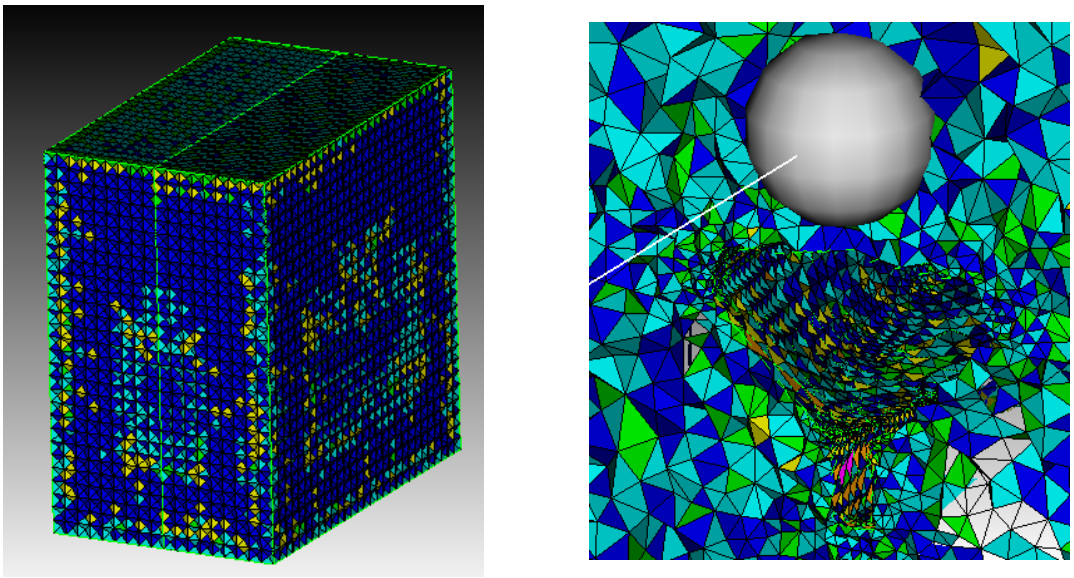

Abbildung 4.38: Netz der mittleren Schicht: Links das komplette Netz; Rechts: Schnitt durch das Netz im Bereich der Finne

Dieses Netz wurde auf die verschiedenen Kriterien für Tetraeder, welche in Kapitel 2.1.5 hergeleitet wurden, getestet.

Aspektverhältnis:

Für das Aspektverhältnis ist $[1, 3]$ ein akzeptables Intervall . In Abbildung 4.39 ist das deutlich erfüllt, da nahezu alle Tetraeder den optimalen Wert 1 für das Aspektverhältnis annehmen.

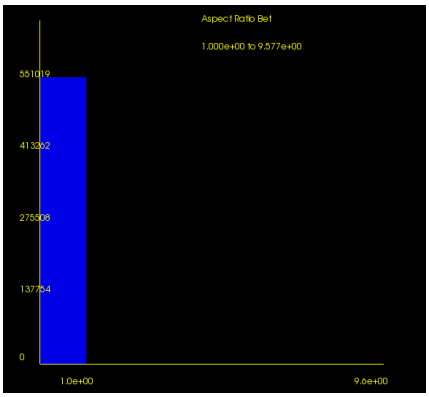

Abbildung 4.39: Säulendiagramm zum Aspektverältnis der mittleren Schicht

Shape:

Ein akzeptables Intervall für das Shape ist $[0.3, 1]$. Laut Abbildung 4.40 trifft dies zu.

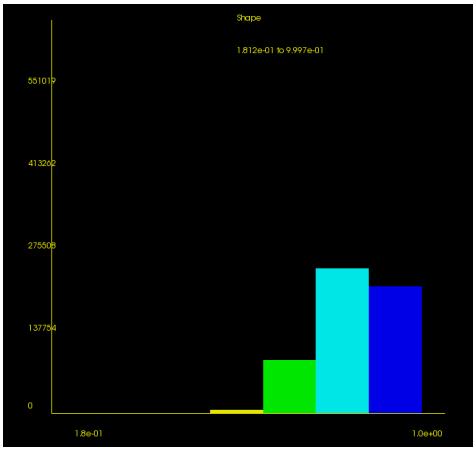

Abbildung 4.40: Säulendiagramm zum Shape der mittleren Schicht

Skalierte Jacobi-Determinante:

Die skalierte Jacobi-Determinante sollte im Intervall $[\frac{1}{2}, \frac{\sqrt{2}}{2}]$ liegen. In Abbildung 4.41 ist auch das deutlich erfüllt.

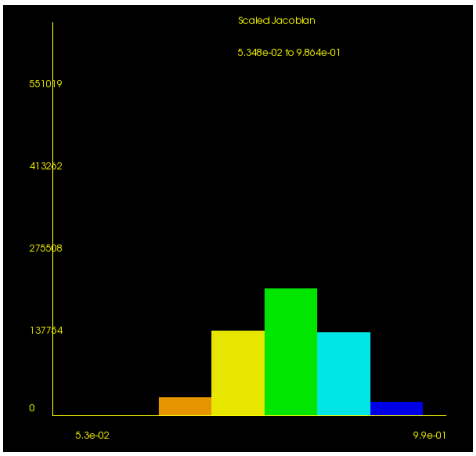

Abbildung 4.41: Säulendiagramm zur skalierten Jacobi-Determinante der mittleren Schicht

4.10 Ordnerstruktur

In Abbildung 4.42 ist die Ordnerstruktur des C++ Programms **vmesh** dargestellt. Das Programm läßt sich dabei in fünf Bereiche unterteilen. Der Unterordner *Automation* beinhaltet die verschiedenen Inputskripte für CUBIT, welche bewirken, dass die zu vernetzende Geometrie importiert wird und das Oberflächennetz generiert wird und im Abaqus-Format abgespeichert wird. In *Comet_stuff* sind Skripte zum automatischen Ablauf von Comet für die netzbasierte Formoptimierung hinterlegt (siehe Kapitel 5.2). Im Unterordner *Export* sind die verschiedenen Ausgaben, welche von **vmesh** erzeugt werden, abgelegt (siehe Abschnitt 4.11). Das Oberflächennetz und die zu vernetzenden Geometrien sind im Unterordner *Import* abgelegt. Neben diesen drei Ordnern finden sich dann noch das *Makefile* und die eigentlichen Programme **vmesh_v29.cpp**, **vmesh_complete.cpp**, **vmesh_script.cpp** der Deformationsalgorithmus und das Hooke-Jeeves-Verfahren.

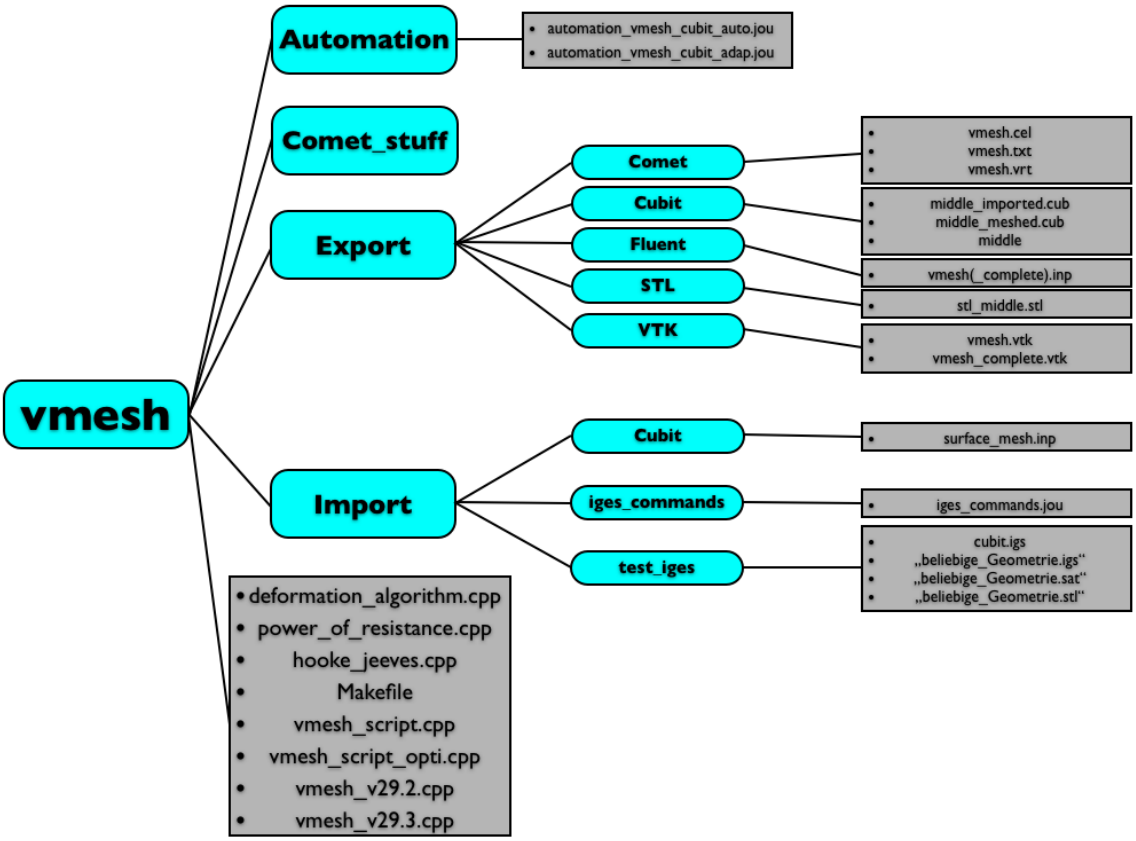

Abbildung 4.42: Ordnerstruktur

4.11 Ausgabeoptionen

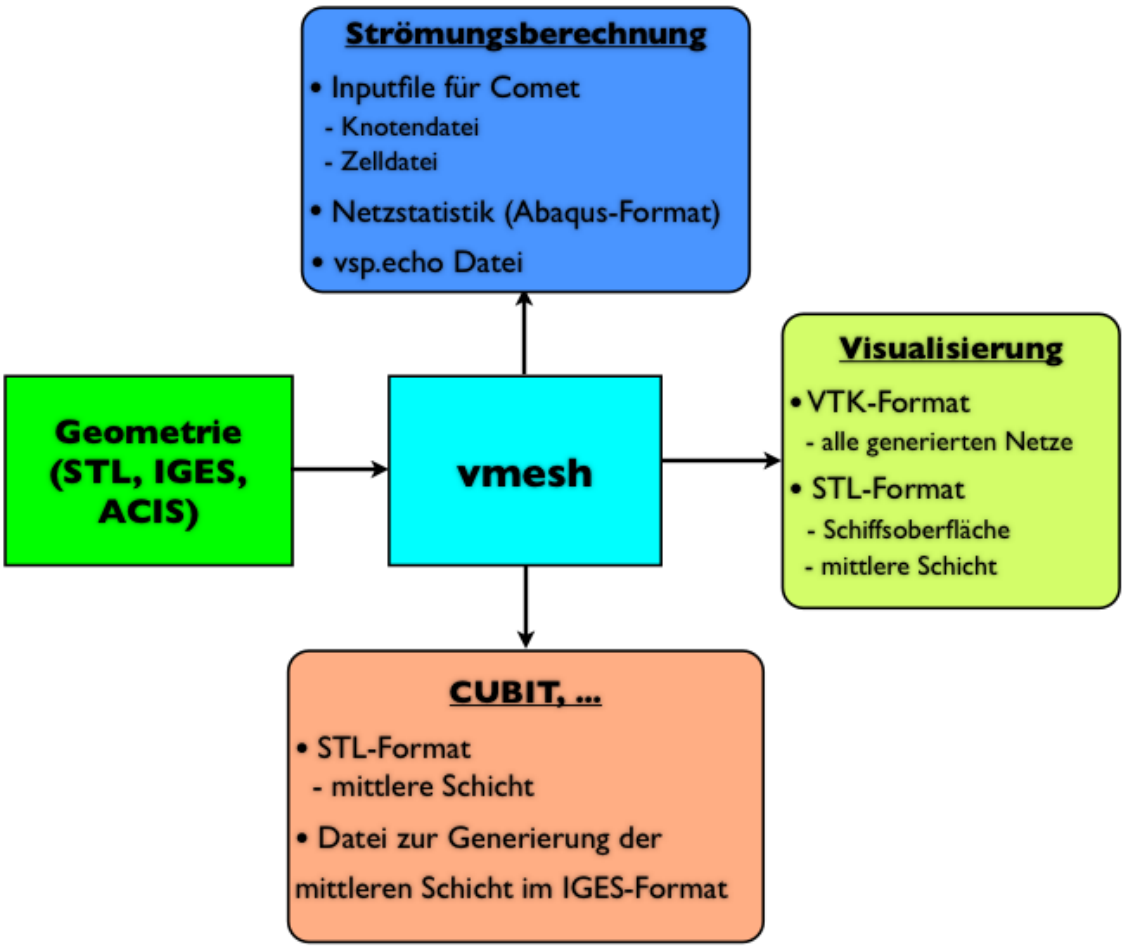

Abbildung 4.43: Ausgabeoptionen

- Strömungsberechnung:
  - Zu diesem Zwecke wird das Netz im Comet-Format benötigt (Abschnitt 4.3.5). Deswegen wird eine Knoten- und Zelldatei in diesem Format exportiert (*vmesh/Export/Comet/vmesh.vrt (vmesh.cel)*).
  - Wichtig für die CFD-Rechnung ist die Netzqualität. Es besteht die Möglichkeit die innere Schicht im Abaqus-Format (Abschnitt 4.3.3) auszugeben. Diese exportierte Datei kann in CUBIT eingelesen werden und die sehr gute Netzstatistik von Cubit kann angewendet werden (*vmesh/Export/Fluent/vmesh.inp*; oder das gesamte Netz mit mittlerer Schicht *vmesh/Export/Fluent/vmesh_complete.inp*).

  - Die vsp.echo Datei (Anhang A) wird optional generiert (*vmesh/Export/Comet/vsp.echo*). In dieser Datei werden die Einstellungen (Randbedingungen, Geschwindigkeit, Turbulenzmodell, uvm) für Comet vorgenommen.

- Visualisierung:

  - Um die Netze auch visuell beurteilen zu können, wurde ein VTK-Export (Abschnitt 4.3.4) verwirklicht. Diese Dateien können mit ParaView eingelesen werden und lassen sich dort anschauen. Dabei sind folgende Optionen realisiert:
    * Gesamtes Volumennetz (*vmesh/Export/VTK/vmesh_complete.vtk*)
    * Nur die innere Schicht (mit und ohne Pyramiden) (*vmesh/Export/VTK/vmesh.vtk*)
    * Nur die äußere Schicht (mit und ohne Pyramiden) (*vmesh/Export/VTK/vmesh.vtk*)
    * Die innere und äußere Schicht (mit und ohne Pyramiden) (*vmesh/Export/VTK/vmesh.vtk*)
  - Die Hülle der mittleren Schicht kann auch im STL-Format (Abschnitt 4.3.1) zur Visualisierung ausgegeben werden (*vmesh/Export/STL/stl_middle.stl*).
  - Die Hülle der mittleren Schicht kann auch im IGES-Format (Abschnitt 4.3.2) zur Visualisierung ausgegeben werden (*vmesh/Export/Cubit/middle_imported.cub*).

- Cubit:

  - Um die mittlere Schicht mit Tetraedern zu füllen, muss diese für Cubit im STL- oder IGES-Format vorliegen. Zu diesem Zwecke wurden für die mittlere Schicht beide Exportmöglichkeiten realisiert.

4.12 Einstellungen

```
//*********Import*************

    int surface_mesh_inp_to_vmesh_on=1;

//*********Layers*************

    int int_lay_on=1;

    int vmerge_int_on=0;

    int full_model_on=1;

    int pyramide_int_on=0;

    int ext_lay_old_on=0;

    int pyramide_ext_old_on=0;

    int vmerge_all_on=1;

//*********Export*************

    int comet_on=1;

    int vtk_on=1;

    int abaqus_on=1;

    int stl_middle_pyramide_on=1;

    int acis_on=1;

    int vspecho_on=1;

//********Algorithm***********

    int deform_on=0;

    int p=1;

    // general parameters
    int laufknoten2d=0;
    int laufzellen2d=0;
    int laufknoten3d=0;
    int laufzellen3d=0;
    int laufpress=0;
    int laufstl=0;
    int ab=17;
    int precision=10;
    double scaling=1;
    double eps=0.05;
```

```
int change=1;

// get path by environment variable
string work_dir = getenv("SHIP_HOME");

// string: path
string path=work_dir+ "/";

// deformation
double x_p=42000;
double y_p=6500;
double z_p=3500;
double dx=0;
double dy=-2000;
double dz=0;
double radius=50000000;
double lambda=1;
double lambda_n=1;

// parameters of the internal layer
double thickness=0.0001;
int num_of_lay=2;
double dilation=1.09;

// parameters of the external layer (old)
double oben=2000;
double unten=-6622;
double seite=7000;
double vorne=20500;
double hinten=-7000;
double pu=-1500;
double ps=1500;
double pv=4000;
double ph=-1500;
double zg=170;
double anzbreite=int((seite-0)/zg+0.5);
double anzps=int((seite-ps)/zg);
double anzpu=int((pu-unten)/zg);
double anzmitte=int((pv-ph)/zg+0.5);
double anzhoehe=int((oben-unten)/zg+0.5);

int lauf_pyramide=0;
int lauf_pyramide2=0;
int lauf_aeusserepolygon=0;
double scal=0.2; // high of pyramide on the ship
double scal_a=0.2; // high of pyramide (normal 0.5)
```

Im Folgenden werden die einzelnen Einstellungsmöglichkeiten erklärt. Dabei steht eine 1 immer für generieren/anwenden und eine 0 für das Gegenteil:

- Import:

  - Als Import wird immer ein Oberflächennetz im Abaqus-Format benötigt. Mit *surface_mesh_inp_to_vmesh_on* kann eingestellt werden, ob ein solches Oberflächennetz eingelesen wird.

- Schichten:
  - Mit *int_lay_on* kann ausgewählt werden, ob die innere Schicht generiert wird.
  - Mit *vmerge_int_on* muss eingestellt werden, ob die zu vernetzende Geometrie als Halb- oder als Vollmodell vorliegt. Beim Halbmodell muss dem Parameter *vmerge_int_on* der Wert 1 zugewiesen werden.
  - Mit *full_model_on* kann ausgewählt werden, ob das generierte Netz als Voll- oder Halbmodell generiert werden soll. Hier wird empfohlen immer das Vollmodell zu nehmen.
  - Mit *pyramide_int_on* kann ausgewählt werden, ob die Pyramiden auf der inneren Schicht generiert werden sollen.
  - Mit *ext_lay_old_on* kann ausgewählt werden, ob die äußere Schicht generiert werden soll.
  - Mit *pyramide_ext_old_on* kann ausgewählt werden, ob die Pyramiden auf der äußeren Schicht generiert werden sollen.

- Export:
  - *comet_on*: Ausgabe der Netze im Comet-Format.
  - *vtk_on*: Ausgabe der Netze im VTK-Format.
  - *abaqus_on*: Ausgabe der inneren Schicht im Abaqus-Format.
  - *stl_middle_pyramide_on*: Ausgabe der Oberfläche der mittleren Schicht im STL-Format.
  - *acis_on*: Ausgabe der Oberfläche der mittleren Schicht im Iges-Format.
  - *vspecho_on*: Ausgabe der vsp.echo Datei für den Strömungslöser Comet.

- Algorithmen:
  - Mit *deform_on* kann ausgewählt werden, ob der Deformierungsalgorithmus aktiviert wird.
  - Mit p kann der Offset-Algorithmus aktiviert werden. $p = 1$ steht dabei für die normale Version (Abschnitt 4.6.2) und $p = 2$ für die erweiterte Option (Abschnitt 4.6.3).

- Allgemeine Parameter:
  - *precision*: Genauigkeit
  - *scaling*: Skalierungsfaktor für das gesamte Netz

  - *eps*: Genauigkeit beim Vmerge-Algorithmus. Wenn zwei Knoten einen Abstand kleiner als *eps* haben, wird ein Knoten in der Zelldatei durch den anderen ersetzt
  - *change*: Richtung der Pyramiden bei der inneren Schicht

- Pfad:
  - Der Pfad des Ordners wird automatisch gesetzt. An der Stelle kann der Pfad aber auch manuell eingestellt werden.

- Parameter der inneren Schicht:
  - *thickness*: Dicke der ersten Lage der inneren Schicht
  - *num_of_lay*: Anzahl der Lagen
  - *dilation*: Streckfaktor für die darauf folgenden Lagen

- Parameter der äußeren Schicht:
  - *oben*: Deckel des gesamten Quaders
  - *unten*: Boden des Quaders
  - *seite*: Seite des Quaders
  - *vorne*: Einlaß des Quaders
  - *hinten*: Auslaß des Quaders
  - *pu*: Boden der Hülle der mittleren Schicht
  - *ps*: Seite der Hülle der mittleren Schicht
  - *pv*: Vorderseite der Hülle der mittleren Schicht
  - *ph*: Hintere Begrenzung der Hülle der mittleren Schicht
  - *zg*: Zellgröße der Zellen der äußeren Schicht
  - *scal*: Höhe der Pyramiden auf der inneren Schicht
  - *scal_a*: Höhe der Pyramiden auf der äußeren Schicht

4.13 Rechenaufwand des gesamten Vernetzungsablaufs

Die Rechnungen zur Bestimmung des Aufwandes für den gesamten Vernetzungsablauf von **vmesh** wurde mit der Cubitversion 12.1 (**vmesh_v29.3.cpp**) auf meinem MacBook (Prozessor: 2.4 GHz Intel Core 2 Duo) durchgeführt. Bei dieser Version laufen auf dem MacBook alle Abschnitte der Vernetzung vollautomatisch ab. Es wurde die Netzfeinheit sukzessive erhöht.

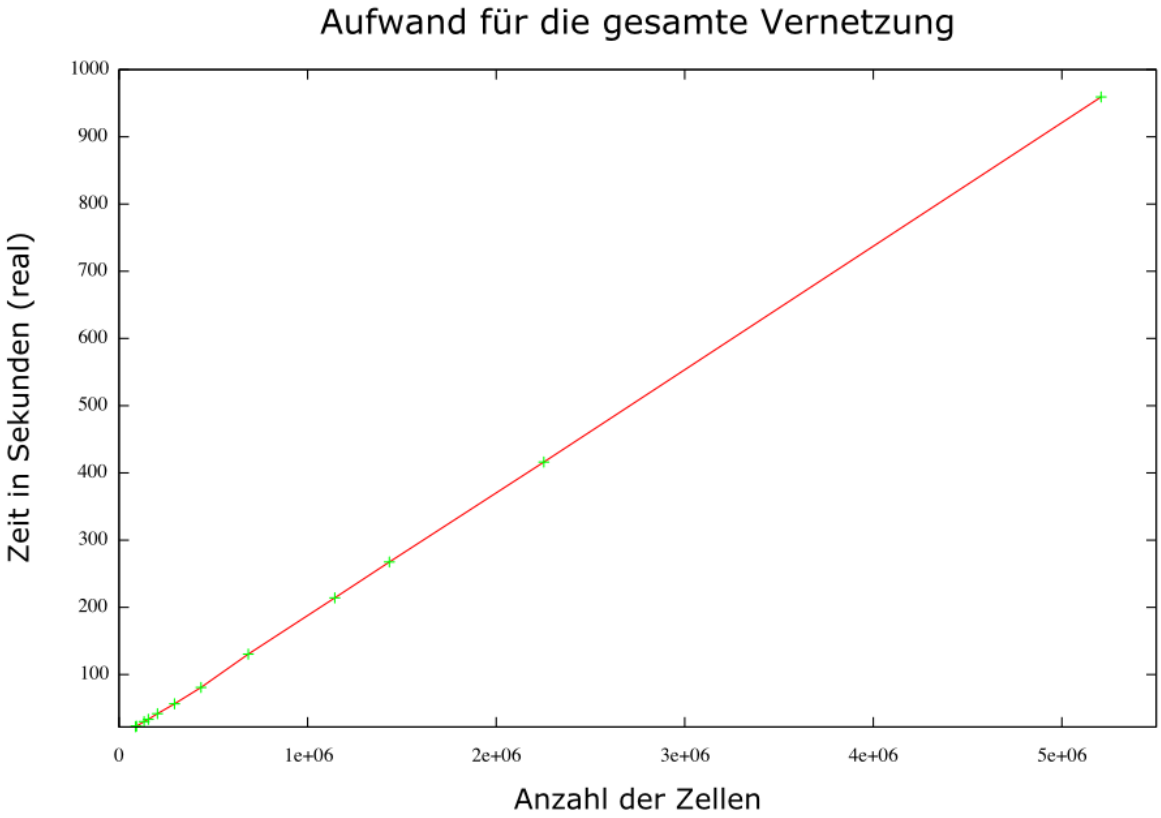

Abbildung 4.44: Diagramm zum Rechenaufwand von **vmesh**

In Abbildung 4.44 kann abgelesen werden, dass die Anzahl der Zellen sich linear zur Zeit der Generierung der Zellen verhält. In den folgenden Tabellen sind die zugehörigen Werte mit den entsprechenden Einstellungen eingetragen.

Anzahl: Zellen	88204	92354	132830	155291	203697	294408
Zeit in Sekekunden (real)	**22.568**	**23.143**	**30.059**	**33.403**	**41.801**	**56.530**
Zeit in Sekunden (user)	17.039	17.572	22.402	25.304	31.553	42.978
Zellgröße: zg	400	380	350	330	300	280
Paving (auto)	10	9	8	7	6	5

Anzahl: Zellen	434829	685614	1144149	1433836	2252854	5208108
Zeit in Sekunden (real)	**80.800**	**130.368**	**214.107**	**267.481**	**416.149**	**959.291**
Zeit in Sekunden (user)	61.859	100.000	165.520	206.156	320.856	729.169
Zellgröße: zg	250	220	190	170	140	100
Paving (auto)	4	3	2	1.7	1.4	1

Die restlichen Parameter wurden konstant gehalten und entsprechen den Einstellungen in Kapitel 6.3.2.

Kapitel 5

Numerische Optimierung

Bei der Formoptimierung wird die Gestalt einer Geometrie durch ein Optimierungsverfahren so variiert, damit das Zielfunktional $g(x) : \mathbb{R}^n \mapsto \mathbb{R}$ minimiert wird. Es gilt das folgende Optimierungsproblem zu lösen

$$\min_{x \in \mathbb{R}^n} g(x).$$

Die möglichen Formvariationen müssen dabei vorab definiert und dem Optimierungsprogramm übergeben werden. Die Formoptimierung ist eine sehr aufwendige und anspruchsvolle Art der Optimierung.

Im Folgenden werden zwei Möglichkeiten der Formoptimierung vorgestellt und beschrieben inwieweit diese realisiert wurden. Es handelt sich um die geometriebasierte Optimierung und die netzbasierte Formoptimierung.

5.1 Geometriebasierte Optimierung

5.1.1 Optimierungskette und Realisierung

In Abbildung 5.1 ist ein solcher Optimierungszyklus dargestellt. Bei der geometriebasierten Optimierung wird zuerst die zu optimierende Geometrie sinnvoll parametrisiert. Diese Parameter werden auch Designvariablen genannt. Mit diesen Parametern lassen sich gezielt Formveränderungen durchführen. Die parametrische Schiffsbeschreibung basierend auf NURBS Kurven und -Flächen [51] wurde in dem C++ Programm *PaShiMo* realisiert [**?**]. Das Modellierungstool entstand ebenfalls im Rahmen des Projektes SimuVSP und es wurde von Juan Carlos Matutat entwickelt. Dieses Tool liefert als Export ein Stl-File der erzeugten Geometrie.
Das STL-File wird dem Vernetzer **vmesh** übergeben und es wird vollautomatisch ein Volumennetz generiert (Kapitel 4).
Das generierte Netz wird dem Strömungslöser *Comet* übergeben, der dann die Widerstandskraft in x-Richtung berechnet. Diese Widerstandskraft dient bei dieser Optimierung als Zielfunktional.

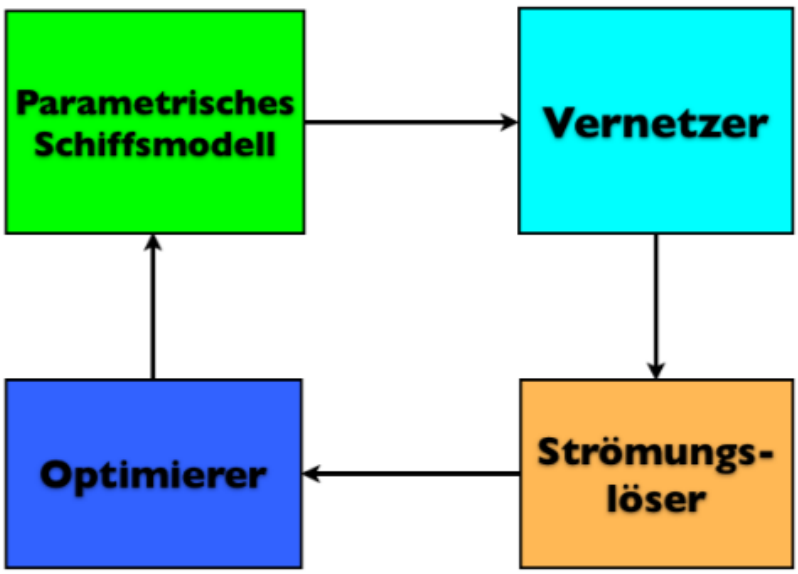

Abbildung 5.1: Optimierungsdurchlauf - modellbasiert

Der Widerstandswert wird dem Optimierer übergeben, der dann entsprechend die Schiffsparameter ändert, mit dem Ziel, dass dieses Zielfunktional kleiner wird.
Mit dem neuen Parametersatz wird das neue Schiff erzeugt und der Durchlauf beginnt von vorne. Die Optimierung läuft so lange, bis der Optimierer den Lauf abbricht.
Die Umsetzung dieser Optimierung ist detailliert in [47] beschrieben.

5.1.2 Vor- und Nachteile und Grenzen der geometriebasierten Formoptimierung

Vorteile:

Einer der gößten Vorteile ist sicherlich, dass bei der geometriebasierten Optimierung die Designvariablen als Optimierungsparameter genutzt werden können. Darüber hinaus wird für eine solche Optimierung bei jedem Durchlauf eine automatische Neuvernetzung benötigt, was eine gute Netzqualität gewährleistet. Zudem liegt die optimale Geometrie nach der Optimierung in einem gängigen CAD-Format (IGES, STL, ...) vor. Des Weiteren läßt sich das Einbinden von Nebenbedingungen (Verdrängung, Volumen, ...) in die Optimierung leichter als bei der netzbasierten Formoptimierung realisiern. Darüber hinaus bietet die geometriebasierte Optimierung mehr Freiheit bei der Formvariation.

Nachteile:

Ein Nachteil ist, dass die zu optimierende Geometrie zuerst mit dem Parametersatz abgebildet werden muss. Dies bedarf oft selber einer aufwendigen Optimierung und ist oft

auch nicht ausreichend realisierbar. Es muss gewährleistet sein, dass durch die Parametriesierung die zu optimierende Geometrie auch abgebildet werden kann. Darüber hinaus kann die automatische Neuvernetzung, die bei jedem Durchlauf stattfinden muss, versagen. Das kann daran liegen, dass durch die Veränderung der Geometrieparameter eine nicht vernetzbare Geometrie entsteht. Es besteht auch die Möglichkeit, dass der Aufbau einer neuen Geometrie zu Beginn eines Durchlaufs scheitert. Und natürlich kann die Auswahl der relevanten Geometrieparameter aus einer großen Liste sehr beschwerlich sein.

5.2 Netzbasierte Formoptimierung

Bei einer netzbasierten Formoptimierung wird zur Beschreibung der zu optimierenden Geometrie nicht die CAD-Beschreibung, sondern das Volumennetz verwendet. Die Optimierung kann deswegen unabhängig von der Geometrie durchgeführt werden. Durch ein Verschieben der Knoten des Netzes wird dabei eine Formvariation erzeugt. Die Netztopologie bleibt hierbei erhalten.

5.2.1 Formoptimierung mit Basisvektoren

Da bei der netzbasierten Formoptimierung nur die Knoten verschoben werden und die Netztopologie nicht verändert wird, kann die Form eines Bauteils mit N Knoten einfach durch einen Vektor $\overrightarrow{R}^{(0)_T}$ der Dimension $3N$, in dem die Koordinaten $\overrightarrow{r}(i),\ i = 1, ..., N$ aller Knoten des Netzes zusammengefasst werden:

$$\overrightarrow{R}^{(0)_T} = \left(\overrightarrow{r}^{(0)}(1), \overrightarrow{r}^{(0)}(2), ..., \overrightarrow{r}^{(0)}(N) \right) \tag{5.2.1}$$

Mit $\overrightarrow{R}^{(0)_T}$ wird die Ausgangsform bezeichnet. Wir erhalten eine Formvariation $\overrightarrow{R}$ indem wir zu $\overrightarrow{R}^{(0)}$ einen Verschiebungsvektor

$$\overrightarrow{T}^T = \left(\overrightarrow{t}(1), \overrightarrow{t}(2), ..., \overrightarrow{t}(N) \right) \tag{5.2.2}$$

hinzuaddieren. $\overrightarrow{T}$ heisst hier Formbasisvektor (Shape Basis Vektor).
Eine Formvariation ergibt sich dann aus

$$\overrightarrow{R} = \overrightarrow{R}^{(0)} + \sum_{i=1}^{M} x_i \overrightarrow{T_i}, \tag{5.2.3}$$

wobei die x_i Designvariabeln sind und Werte zwischen 0 und 1 annehmen können. Diese Designvariablen können als Optimierungsparameter verwendet werden.

5.2.2 Netzdeformierung

Damit die netzbasierte Formoptimierung realisiert werden kann, wurde ein Deformierungsalgorithmus entwickelt, der eine Deformation ausgewählter Knoten des Volumennetzes sinnvoll auf das Volumennetz überträgt.

Deformierungsalgorithmus

Wir betrachten folgende Deformation $\vec{d_0} = \overrightarrow{P_0 P_0'}$, wobei $\vec{d_{n_0}} = \vec{d_0}/\|\vec{d_0}\|$ ist. Die Knoten des Volumennetzes werden mit P_i, $i = 1, 2, ...$ bezeichnet. Darüber hinaus betrachten wir die Vektoren $\vec{d_i} = \overrightarrow{P_0 P_i}$, wobei $\vec{d_{n_i}} = \vec{d_i}/\|\vec{d_i}\|$ ist. Zusätzlich führen wir noch den

Dämpfungsfaktor λ ein. Die Menge der zu deformierenden Punkte wird mit D_P bezeichnet. Der Algorithmus hat dann folgende Gestalt.

Algorithm 18 Deformierungsalgorithmus

if $\vec{P_i'} \in D_P$ **then**

$$\vec{P_i'} = \vec{P_i} + \langle \vec{d_{n_0}}, \vec{d_{n_i}} \rangle \left(1 - e^{-\lambda \frac{\|\vec{d_0}\|}{\|\vec{d_i}\|}}\right) \vec{d_i}$$

else

$$\vec{P_i'} = \vec{P_i}$$

end if

Ein Beispiel einer solchen Deformation ist in Abbildung (5.2) dargestellt.

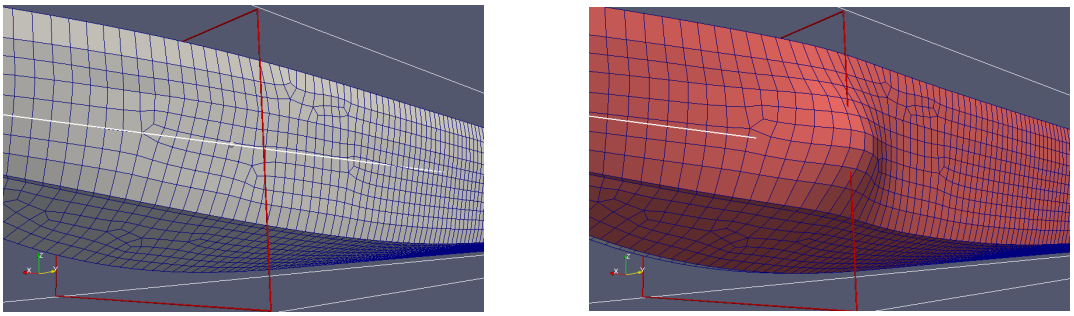

Abbildung 5.2: Beispiel einer Deformation durch den Deformierungsalgorithmus

Durch das Skalarprodukt $\langle \vec{d_{n_0}}, \vec{d_{n_i}} \rangle$ wird die Richtung des Vektors d_{n_i} zur Deformationsrichtung d_{n_0} berücksichtigt. Mit dem Term

$$1 - e^{-\lambda \frac{\|\vec{d_0}\|}{\|\vec{d_i}\|}} \tag{5.2.4}$$

kann eingestellt werden, wie stark die Entfernung des zu deformierenden Knoten P_i von P_0 in die Deformation einfließen soll. Die Variable λ sollte dabei positiv gewählt werden. Bei den ersten Optimierungsläufen war dieser Parameter auf 1 eingestellt.

5.2.3 Optimierungskette und Realisierung in vmesh

Die Realisierung einer netzbasierten Formoptimierung wurde bisher in **vmesh** folgendermaßen umgesetzt (Abb.: 5.3).

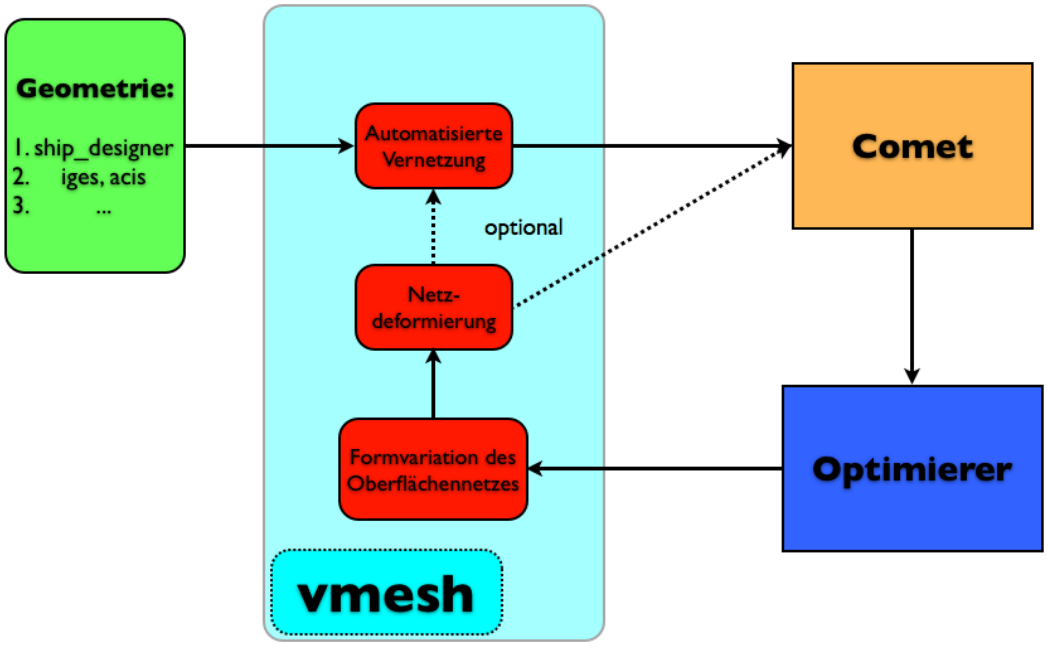

Abbildung 5.3: Optimierungsdurchlauf - netzbasiert

Der Ablauf wird durch das Skript **vmesh_script_opti.cpp** gesteuert.

vmesh_script_opti.cpp: (Für Linux)

```
int main()
{
    //**********meshgeneration**********

    system("./vmesh_automation_auto");

    system("cubit -nogui -nographics -input ./Automation/
        automation_vmesh_cubit.jou");
```

```
    system("./vmesh_v29_2");

    system("cubit -nogui -nographics -input ./Export/Cubit/middle");

    system("./vmesh_complete");

    //**********************optimization**************************

    system("./deform_algorithm");

    double opti1=1;
//  double opti2=0;

    int nvars=1;
    int iteration=0;

    double startpt[nvars];
    startpt[0]=opti1;
//  startpt[1]=opti2;

    double endpt[nvars];

    hooke(nvars, startpt, endpt, 0.5, 0.00001, 500, iteration);

    return 0;
}
```

Zuerst wird die zu optimierende Geometrie entweder von dem Modellierungstool *PaShiMo*[47] erzeugt, oder es wird eine beliebige Geometrie im IGES-Format oder STL-Format eingelesen.
Danach werden die Parameter P_0 (x_p, y_p, z_p) und $\vec{d_{n_0}}$ (dx, dy, dz) vorab eingestellt. Es kann auch noch ein Radius angegeben werden, so dass nur diese Punkte des Volumennetzes deformiert werden, welche innerhalb der Kugel mit angegebenem Radius liegen. Auch müssen die Optimierungsparameter, wie z.B. die minimale Schrittweite, die Dimension der Optimierung und die Abbruchbedingung festgelegt werden (Abschnitt 5.2.4).

```
// deformation
double x_p=42000;
double y_p=6500;
double z_p=3500;
double dx=0;
double dy=-2000;
double dz=0;
double radius=5000;
double lambda=1;
double lambda_n=1;
```

Danach erfolgt die automatische Vernetzung mit **vmesh**. Es kann auch ein manuell erstelltes Netz erzeugt werden. Dieses Netz muss allerdings im Comet-Format vorliegen. Nach der Stömungssimulation erhalten wir für die zu optimierende Geometrie einen Wert für das Zielfunktional $g(x) : \mathbb{R}^n \mapsto \mathbb{R}$. Bei uns ist das die Widerstandskraft in

x-Richtung. Das Programm **power_of_resistance.cpp** berechnet aus einer Datei von Kraftwerten (*./Comet_stuff/calculation/vsp.hisfloat*) aus den einzelnen Zeitschritten einen Wert für die Widerstandskraft in x-Richtung. Dieser wird durch Mittelwertbildung aus den letzten Kraftwerten der auskonvergierten Lösung berechnet.

power_of_resistance.cpp:

```
int main()
{
    int ab=17;

    string infile, s;
    double a1,a,b,c,d,e,e2,g,h,j;

    ifstream in;

    infile = "./Comet_stuff/calculation/vsp.hisfloat";
    in.open(infile.c_str());

    if (!in) {
        cerr << "power_of_resistance: FILE ERROR!\n";
        exit(1);
    }

    double force_sum=0;

    for (int i=0; i<1000; ++i){

        getline(in,s);
        stringstream line(s);
        line >> a >> b >> c >> d >> e >> e2 >> g >> h >> j;

        if(i>=500)
        {
            force_sum=force_sum+j;
        }
    }

    in.close();

    double power_of_resistance=force_sum/double(500);

    infile = "./Comet_stuff/iteration";
    in.open(infile.c_str());

    if (!in) {
        cerr << "power_of_resistance: FILE ERROR!\n";
        exit(1);
    }

    int lauf=0;
```

```
    for (int i=0; i<1; ++i){

        getline(in,s);
        stringstream line(s);
        line >> a1;

        lauf=int(a1);
    }

    in.close();

    fstream f;
    string outfile="./Comet_stuff/calculation/power/force_x_";

    lauf=lauf-1;

    string number= _num_to_str(lauf);
    string outfile2 = outfile + number;

    f.open( outfile2.c_str(), ios::out);

    f << right << setw(ab) << power_of_resistance << endl;

    f.close();
}
```

Dieser Wert wird dem Optimierer gegeben, in unserem Fall dem Hooke-Jeeves Algorithmus (Abschnitt 5.2.4), dieser ändert die Optimierungsparameter. Mit den neuen Parametern wird die Netzdeformation durchgeführt und ein neues Netz entsteht. Dieser Schritt ist in dem Programm **deformation_algorithm.cpp** realisiert.

Funktion deform:

```
void deform(jmatrix &node3d, double x_p, double y_p, double z_p,
double dx, double dy, double dz, int laufknoten3d, int deform_on,
double radius, double lambda, double lambda_n)
{
   if(deform_on==1)
   {
      for (int i=0; i<laufknoten3d; ++i)
      {
         node3d.set_elem(i, 4, node3d.get_elem(i, 1));
         node3d.set_elem(i, 5, node3d.get_elem(i, 2));
         node3d.set_elem(i, 6, node3d.get_elem(i, 3));
      }

      for (int i=0; i<laufknoten3d; ++i)
      {
         double ab=abstand(x_p, y_p, z_p, node3d.get_elem(i, 4), node3d.
             get_elem(i, 5), node3d.get_elem(i, 6));

         if (ab<=radius)
```

```
            {
                double dx_n=dx/(sqrt(dx*dx+dy*dy+dz*dz));
                double dy_n=dy/(sqrt(dx*dx+dy*dy+dz*dz));
                double dz_n=dz/(sqrt(dx*dx+dy*dy+dz*dz));

                double sp=abs(skalarpr(dx_n, dy_n, dz_n,
                norm_x(node3d.get_elem(i, 4), node3d.get_elem(i, 5), node3d.
                    get_elem(i, 6), x_p, y_p, z_p),
                norm_y(node3d.get_elem(i, 4), node3d.get_elem(i, 5), node3d.
                    get_elem(i, 6), x_p, y_p, z_p),
                norm_z(node3d.get_elem(i, 4), node3d.get_elem(i, 5), node3d.
                    get_elem(i, 6), x_p, y_p, z_p)));

                double e_x=1-exp(-lambda*(abs(dx)/ab));
                double e_y=1-exp(-lambda*(abs(dy)/ab));
                double e_z=1-exp(-lambda*(abs(dz)/ab));

                if (node3d.get_elem(i, 5)>=0)
                {
                    node3d.set_elem(i, 1, node3d.get_elem(i, 4)+dx*sp*e_x);
                    node3d.set_elem(i, 2, node3d.get_elem(i, 5)+dy*sp*e_y);
                    node3d.set_elem(i, 3, node3d.get_elem(i, 6)+dz*sp*e_z);
                }

                if (node3d.get_elem(i, 5)<0)
                {
                    node3d.set_elem(i, 1, node3d.get_elem(i, 4)+dx*sp*e_x);
                    node3d.set_elem(i, 2, node3d.get_elem(i, 5)-dy*sp*e_y);
                    node3d.set_elem(i, 3, node3d.get_elem(i, 6)+dz*sp*e_z);
                }
            }
        }
    }
}
```

Das Verfahren bricht ab, wenn für die Differenz zweier aufeinanderfolgender Werte des Ziefunktionals gilt, dass

$$|g(x^{(i+1)}) - g(x^{(i)})| < eps,$$

wobei $x \in \mathbb{R}^n$ und $eps \in \mathbb{R}$ eine vorab eingestellte Abbruchgröße ist.

Bei jeder Iteration wird die Kraft, die aktuellen Optimierungsparameter und das Oberflächennetz der zu vernetzenden Geometrie im Abaqus-Format ausgegeben und abgespeichert, so dass eine sinnvolle Analyse der Optimierung gewährleistet ist.

Bisher wurden erste vielversprechende Testläufe durchgeführt, wobei ein Testlauf eindimensionaler Natur war und der Optimierungsparameter war dy (Ergebnisse in Kapitel 6.4). Andere zweidimensionale Versuche wurden mit dy und y_p als Optimierungsparameter durchgeführt.

5.2.4 Das Verfahren von Hooke und Jeeves

Bei diesem Verfahren ist folgendes Optimierungsproblem gegeben

$$\min_{x \in \mathbb{R}^n} g(x),$$

wobei für das Zielfunktional gilt, dass $g(x) : \mathbb{R}^n \mapsto \mathbb{R}$.

Beginnend mit einem Startpunkt $x^{(0)} \in \mathbb{R}^n$, einer minimalen Schrittweite $\epsilon > 0$ und einem Schrittweitenvektor $h \in \mathbb{R}^n$ wird in jedem Schritt entweder ein neuer Punkt $x^{(i)}$ mit $i \in \mathbb{N}$ erzeugt, wobei

$$g(x^{(0)}) > g(x^{(1)}) > g(x^{(2)}) > ...$$

gilt, oder das Verfahren bricht ab. Nachdem das Verfahren nach m Schritten abgebrochen ist, gilt für den kleinsten Schrittweitenvektor h, der durch Halbierung seiner Elemente erreicht wurde $\max_j h_j \geq \epsilon$. Für den Punkt $x^{(m)}$ gilt:

$$g(x^{(m)}) \leq g(x^{(m)} \pm h_j \epsilon_j), \quad \forall j = 1, 2, ..., n$$

Das Hooke-Jeeves Verfahren arbeitet in zwei Stufen, wie nachfolgend beschrieben. Eine Abstiegsrichtung wird bei der *Erkundungsstufe* gesucht und bei der *Fortschreitungsstufe* bewegt man sich entlang dieser Richtung.

Erkundungsschritt

Beginnend vom Startpunkt x werden entlang der Koordinatenachsen die Funktionswerte $g(x+h_j e_j)$ mit $g(x)$ verglichen, wobei e_j der j-te Einheitsvektor und $j = 1, 2, ..., n$ ist. Wenn $g(x+h_j e_j) < g(x)$ ist, dann wird $x+h_j e_j$ der neue Punkt. Wenn $g(x+h_j e_j) \not< g(x)$ ist, wird zusätzlich die entgegengesetzte Richtung geprüft. Im Erfolgsfall ergibt sich so in maximal $2n$ Schritten eine Abstiegsrichtung.

Fortschreitungsschritt

Nach erfolgreicher Erkundung wird in jeder Iteration nicht der gefundene Punkt x^* als $x^{(i+1)}$ genommen, sondern eine weitere Erkundung mit dem Punkt $2x^* - x^{(i)}$ durchgeführt. Dadurch kann das Verfahren beschleunigt werden, wenn größere Schritte entlang der Abstiegsrichtung gemacht werden. Schlägt die vorangegangene Erkundung fehl, wird die Schrittweite h halbiert. Das Verfahren bricht ab, wenn die minimale Schrittweite unterschritten wird. [61] [68]

5.2.5 Vor- und Nachteile und Grenzen der netzbasierten Formoptimierung

Vorteile:

Einer der größten Vorteil ist sicherlich, dass man geometrieunabhängig ist. Außerdem kann eine beliebige Geometrie entweder automatisch oder auch manuell vernetzt werden und auf dieses Netz läßt sich dann auch eine netzbasierte Formoptimierung durchführen. Oft ist die Anzahl der Parameter auch nicht so groß, so dass einfachere Optimierungsmethoden (Hooke-Jeeves, Nelder-Mead) auch erfolgsversprechend einsetzbar sind. Speziell in unserem Fall ist der C++ Code zur netzbasierten Formoptimierung selbst implementiert, so dass Verfahren zum Automatischen Differenzieren leichter anwendbar sind. Als Folge besteht die Möglichkeit gradientenbasierte Optimierungsverfahren zu verwenden.

Nachteile:

Ein Nachteil ist, dass die Erzeugung der Basisvektoren schwierig sein kann. Außerdem liegt die optimierte Form bei der netzbasierten Formoptimierung dann auch nur in einem Netzformat (Abaqus,...) vor und nicht in einem gängigen CAD-Modell. Darüber hinaus können die verwendeten Netzverzerrungen die Qualität der Simulationsergebnisse deutlich verschlechtern. Des Weiteren sind Nebenbedingungen oft schwieriger zu realisieren.

Vielversprechende Ergebnisse der ersten Optimirungsläufe sind in Kapitel (6.4) aufgezeigt.

5.2.6 Robustheit und Laufzeit

Robustheit

Ergebnisse die Aussagen über die Robustheit beider Methoden der Formoptimierung zulassen sind in jeweils in Kapitel 6.3 und 6.4 aufgezeigt.

Laufzeit

Die Laufzeit eines solchen Optimierungsprozesses hängt von mehreren Faktoren ab.

- Netzfeinheit
- Parallelisierungsgrad der Strömungssimulation
- Abbruchkriterien bei der Optimierung

Die in Kapitel 6.4 durchgeführte Formoptimierung dauerte in etwa eine Woche. Das Netz hatte etwa 1.3 Mio. Zellen und es wurde bei der Strömungssimulation seriell gerechnet.

5.2.7 Mögliche Optimierungsverfahren

Bei unserer Problemstellung ist $g(x)$ nicht zwingend glatt. Numerische Ungenauigkeiten der Strömungssimulation können zu Schwankungen und Störungen in der Zielfunktion

$g(x)$ führen. Die benötigte Glattheit für Gradientenverfahren ist also nicht immer gegeben. In Abbildung 5.4 zeigt eine Einordnung der verschiedenen Optimierungsverfahren.

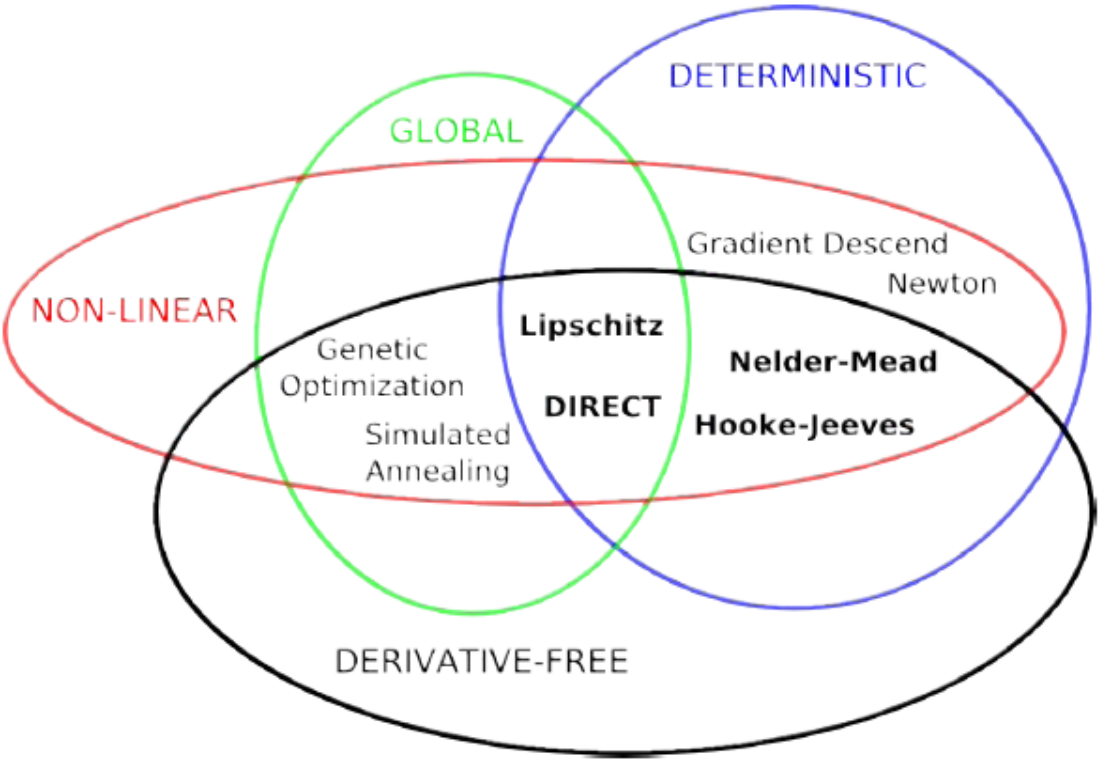

Abbildung 5.4: Einordnung der verschiedenen Optimierungsverfahren [44]

Hooke-Jeeves:

Bisher wurde der Hooke-Jeeves Algorithmus zur Optimierung eingesetzt. Dieses Verfahren benötigt keine Ableitung, arbeitet robust und Nebenbedingungen sind leicht einzubauen. Dieser Algorithmus stößt jedoch bei einer Erhöhung der Optimierungsparameter durch das aufwändige Tasten in alle Raumrichtungen schnell an seine Grenzen.[44]

Nelder-Mead:

Dieser Algorithmus kommt ohne Ableitungen aus, setzt aber die Stetigkeit voraus. Er benötigt nur wenige Funktionsauswertungen und arbeitet effizient und hat oft eine gute Konvergenz. Jedoch hat die Wahl des Startsimplex Einfluß auf das Eintreten und die Schnelligkeit der Konvergenz. Außerdem gibt es keinen Konvergenzbeweis.[44]

Gradientenbasierte Verfahren:

Um gradientenbasierte Verfahren einsetzen zu können, muss mindestens $g(x) \in C^1(\mathbb{R})$ gelten. Dies ist bei unserem Problem jedoch nicht zwingend gegeben. Der Einsatz

des Automatischen Differenzierens (AD) oder der Adjungierten Methode liefern die erwünschten Gradienten und würden somit die gradientenbasierte Optimierung ermöglichen. Aber auch diese Verfahren können das Auffinden des globalen Minimums nicht garantieren. (siehe dazu [68])

Genetische Algorithmen:

Ist die Zielfunktion unstetig und/oder nicht differenzierbar, können zum Beispiel auch Genetische Algorithmen eingesetzt werden.[67] Solche Algorithmen werden vor allem für Probleme eingesetzt, für die eine geschlossene Lösung nicht oder nicht effizient berechnet werden kann, und stehen in Konkurrenz zu klassischen Suchstrategien wie dem Gradientenverfahren.

Simulated Annealing:

Das Simulated Annealing Verfahren wird zum Auffinden einer approximativen Lösung von Optimierungsproblemen eingesetzt, die durch ihre hohe Komplexität das vollständige Ausprobieren aller Möglichkeiten und einfache mathematische Verfahren ausschließen. Dieses Verfahren kann ein lokales Optimum wieder verlassen und ein besseres finden. [6]

Lipschitz-Optimierung:

Diese Optimierung läuft deterministisch ab und benötigt keine Ableitung. Darüber hinaus gibt es zu diesem Verfahren Konvergenzaussagen.[44]

Kapitel 6

Ergebnisse

Im ersten Abschnitt dieses Kapitels werden die generierten Netze für verschiedene Schiffe dargestellt. Um zu zeigen, dass der Vernetzungsprozess unabhängig von der Geometrie ist, beinhaltet dieser Abschnitt zuletzt das Netz von einem Knuckle. Anschließend werden die Ergebnisse von verschiedenen Vergleichsrechnungen vorgestellt. Die letzten beiden Abschnitte enthalten die Ergebnisse der geometriebasierten und der netzbasierten Formoptimierung.

6.1 Von *vmesh* generierte Netze

Im Folgenden werden die Netze vom Voith Wassertrecker, welcher auch für die Vergleichsrechnungen herangezogen wurde (Abschnitt 6.2), der Hard Chine, einem Container-Schiff und einem Knuckle dargestellt.

6.1.1 Voith Wassertrecker mit Finne

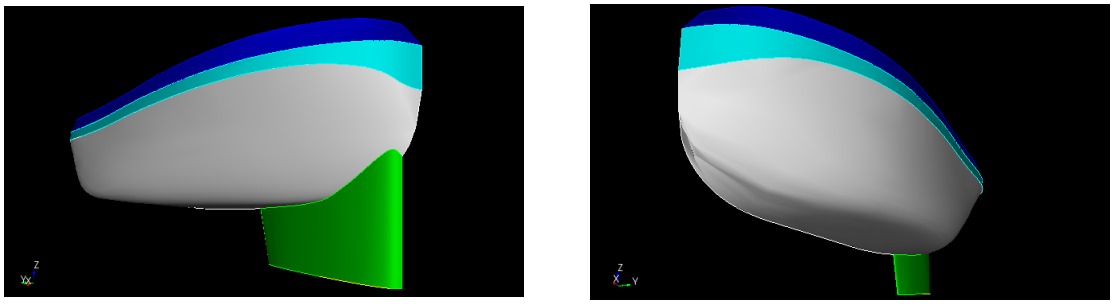

Abbildung 6.1: Voith Wassertrecker im IGES-Format

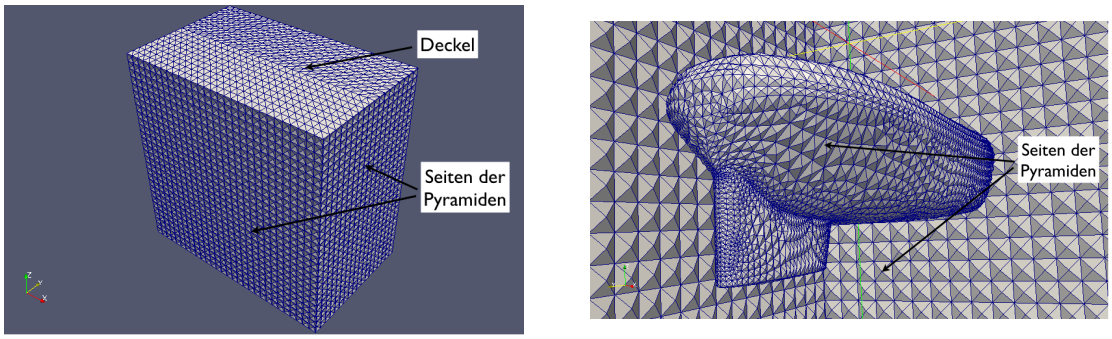

Abbildung 6.2: Mittlere Schicht im STL-Format

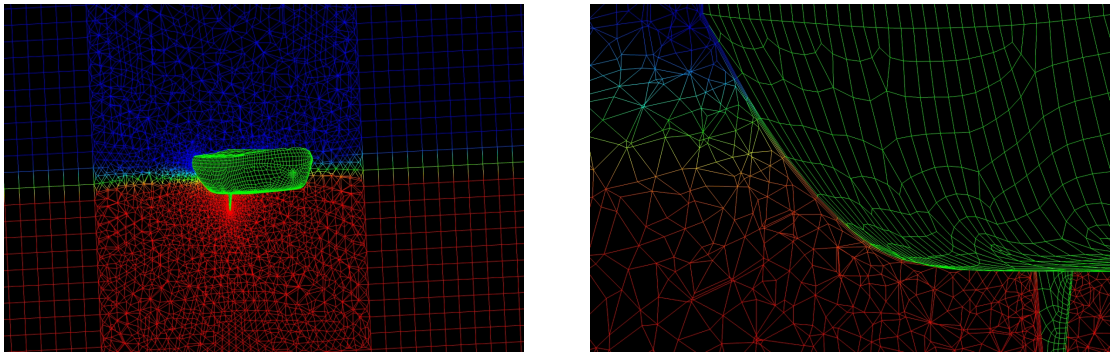

Abbildung 6.3: Einzelne Schichten

6.1.2 HardChine

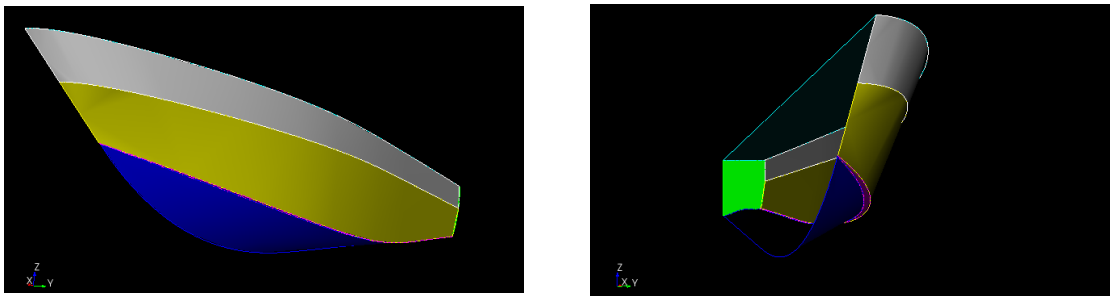

Abbildung 6.4: Hard Chine im IGES-Format

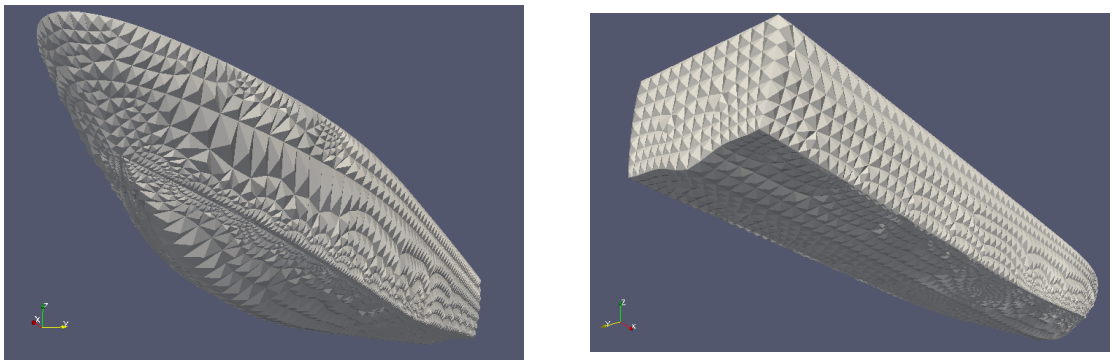

Abbildung 6.5: Hard Chine: Innere Schicht mit Pyramiden

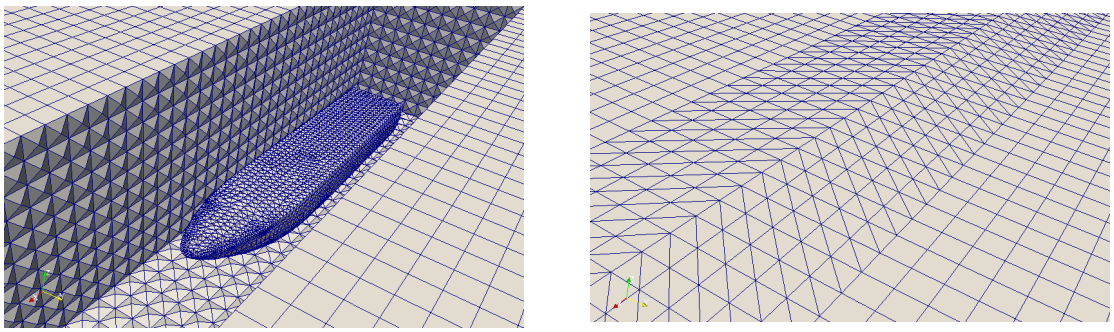

Abbildung 6.6: Hard Chine: Gesamtvolumennetz

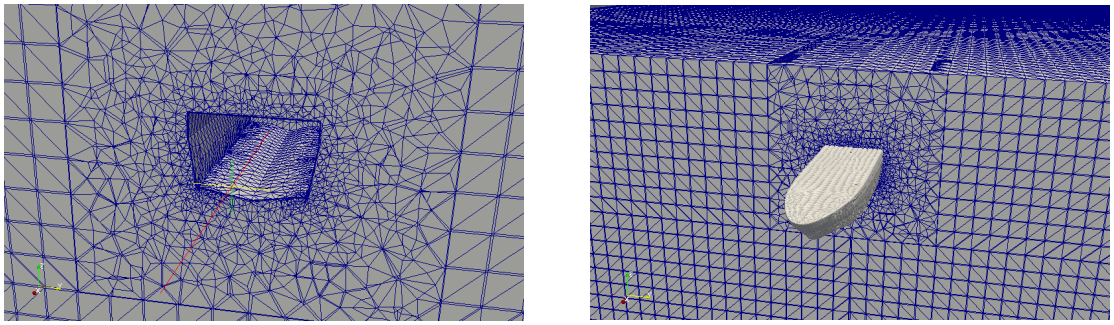

Abbildung 6.7: Hard Chine: Schnitt durch das Gesamtvolumennetz

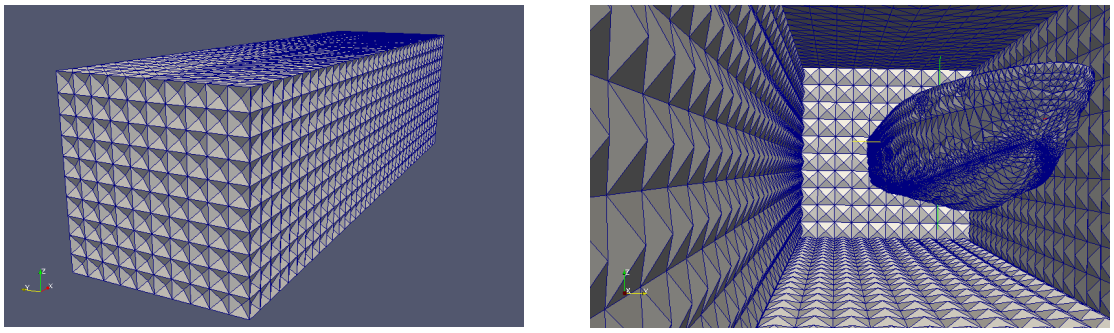

Abbildung 6.8: Hard Chine: STL der mittleren Schicht

6.1.3 Container-Schiff

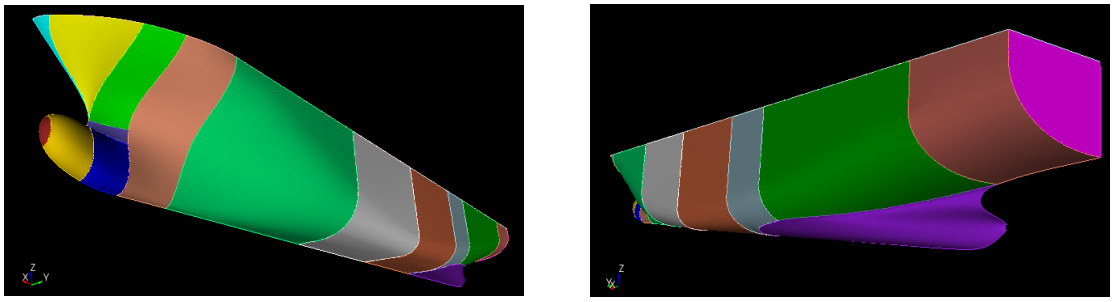

Abbildung 6.9: Container-Schiff im IGES-Format

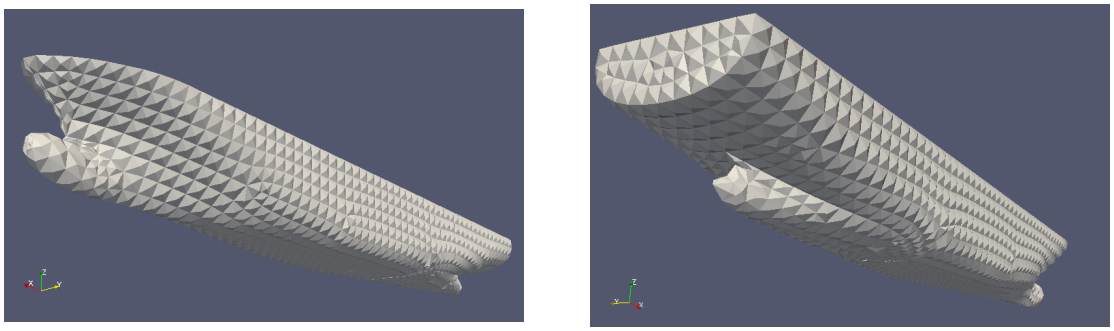

Abbildung 6.10: Container-Schiff: Innere Schicht mit Pyramiden

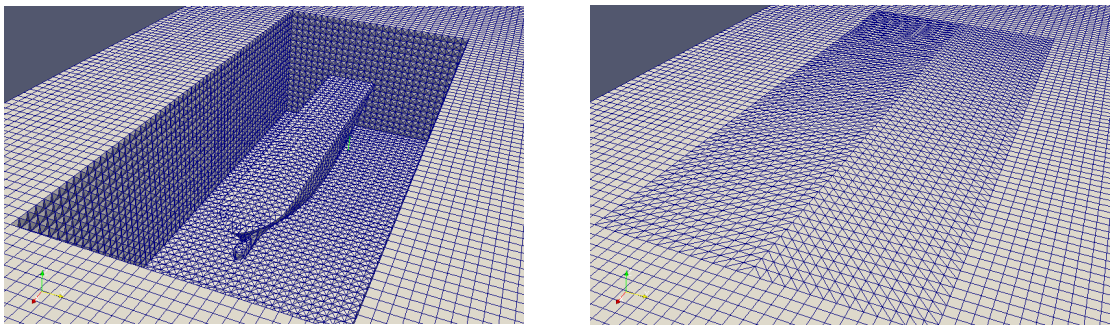

Abbildung 6.11: Container-Schiff: Gesamtvolumennetz

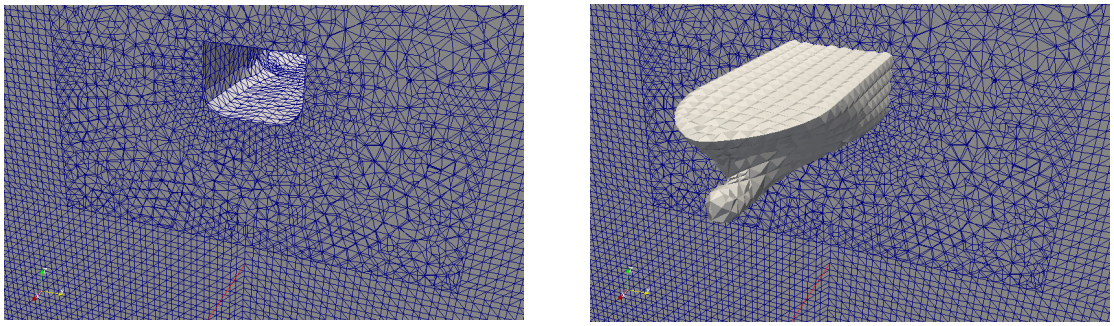

Abbildung 6.12: Container-Schiff: Schnitt durch das Gesamtvolumennetz

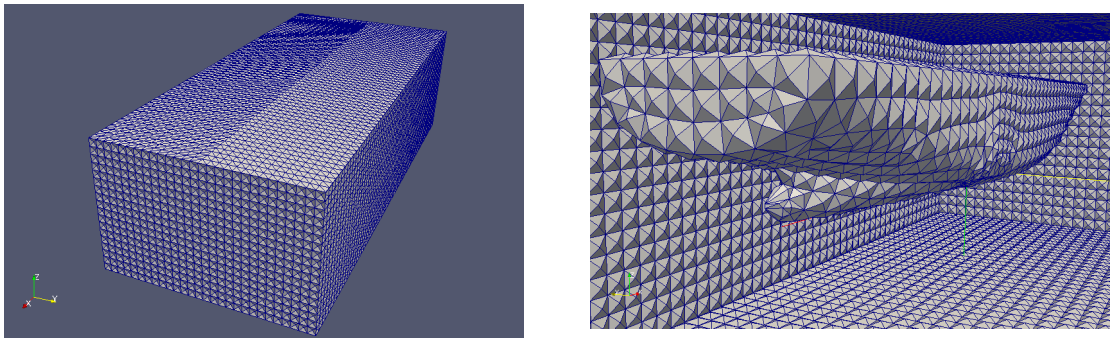

Abbildung 6.13: Container-Schiff: STL der mittleren Schicht

6.1.4 Knuckle

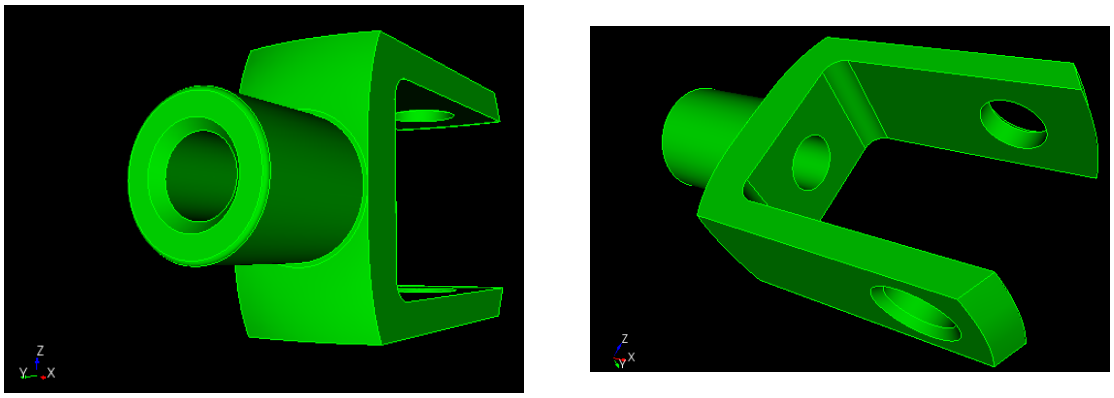

Abbildung 6.14: Knuckle im IGES-Format

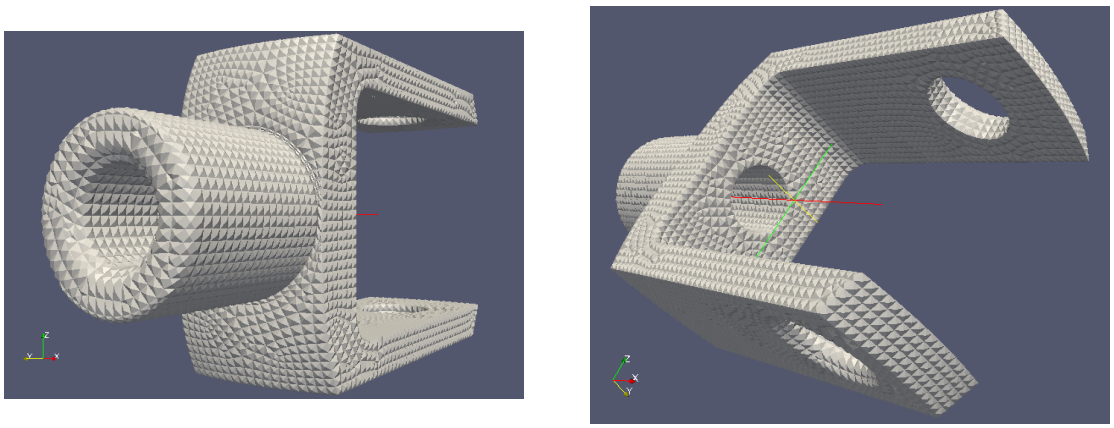

Abbildung 6.15: Knuckle: Innere Schicht mit Pyramiden

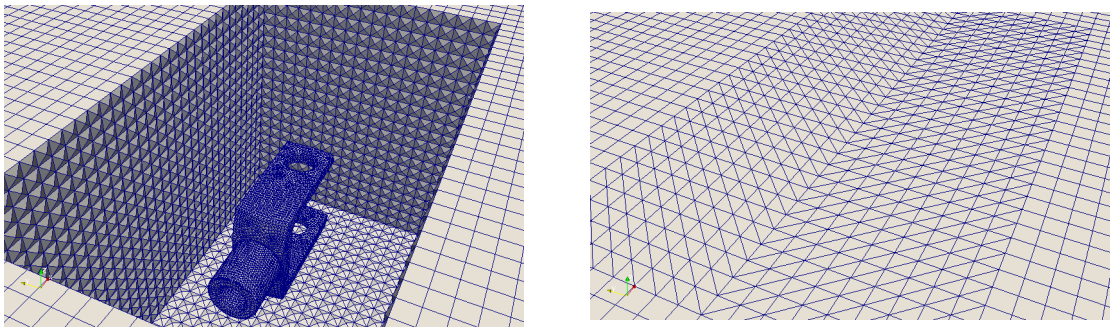

Abbildung 6.16: Knuckle: Gesamtvolumennetz

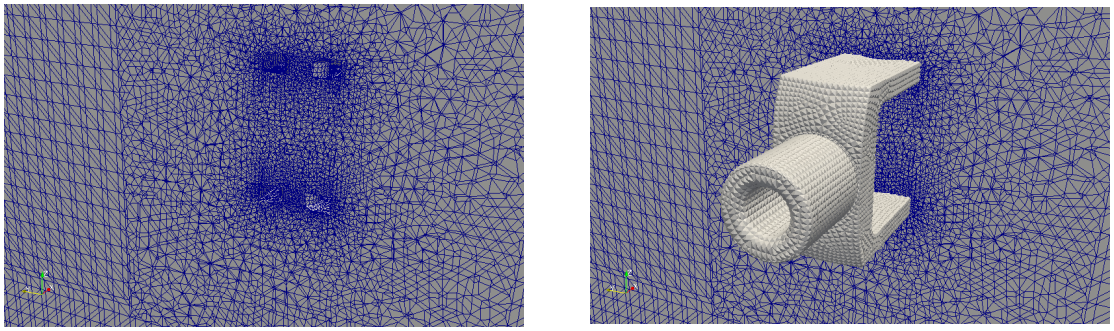

Abbildung 6.17: Knuckle: Schnitt durch das Gesamtvolumennetz

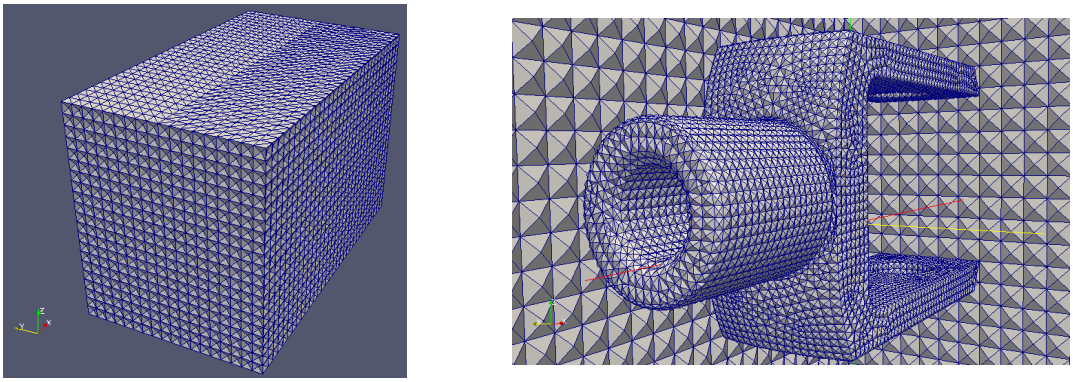

Abbildung 6.18: Knuckle: STL der mittleren Schicht

6.1.5 Mögliche Einstellungen für die verschiedenen Schiffe

Die verwendeten Geometrien wurden zuvor durch eine entsprechende Translation, Spiegelung und Skalierung in Position gebracht. Dies kann entweder in einem CAD-Tool vorab oder im Programm **vmesh_automation_auto.cpp** durchgeführt werden. Die verwendeten Geometrien sind im Ordner *test_geometry* zu finden. Der *Knuckle* liegt als einzige dieser Geometrien als Vollmodell vor.

	Voith Wassertrecker	HardChine	Container-Schiff	Knuckle
Anzahl: Zellen	1474129	808367	929246	460298
Zellgrösse: zg	170	3000	5000	300
Deckel: oben	3378	20000	45000	2000
Boden: unten	-6622	-100000	-150000	-6622
Seite: seite	7000	100000	150000	7000
Vordere Fläche: vorne	20500	200000	350000	20500
Hintere Fläche: hinten	-7000	-300000	-400000	-7000
pu	-2000	-15000	-45000	-1500
ps	2500	20000	70000	1500
pv	4500	80000	250000	4000
ph	-2500	-30000	-70000	-1500
Genauigkeit beim vmerge: eps	0.005	0.005	0.5	0.005
Skalierung: scaling	1	1	1	1
Pyramidenrichtung: change	1	-1	1	1
Dicke erste Zellschicht: thickness	1	20	0.001	0.001
Anzahl von Zellschichten: num of lay	4	4	2	2
Streckung: dilation	1.09	1.09	1.09	1.09
Offset-Algorithmus: p	1	1	1	1
Paving:	3.4 (adaptiv)	3.4 (adaptiv)	3.9 (auto)	3 (auto)
Glättung des Oberflächennetzes:	-	-	Laplace	-

6.2 Vergleichsrechnungen

Um die Qualität der mit **vmesh** vollautomatisch erstellten Netze zu testen, wurden Vergleichsrechnungen durchgeführt.

- **vmesh** im Vergleich mit einem manuell erstellten Netz
- **vmesh** im Vergleich mit einem automatischen Hexaedervernetzer von Cubit
- Vergleichsrechnungen mit und ohne innerer Schicht
- Vergleichsrechnungen für verschiedene Netzfeinheiten

6.2.1 Vegleichsrechnung für den Voith Wassertrecker mit Finne: *vmesh* und manuell erstelltes Netz

Der Voith Wassertrecker mit Finne wurde als Referenzschiff von Voith einmal manuell vernetzt (ca. 3.6 Zellen als Vollmodell) und einmal von **vmesh** automatisch vernetzt (Abb.: 6.19) (ca. 1.3 Mio Zellen als Vollmodell).

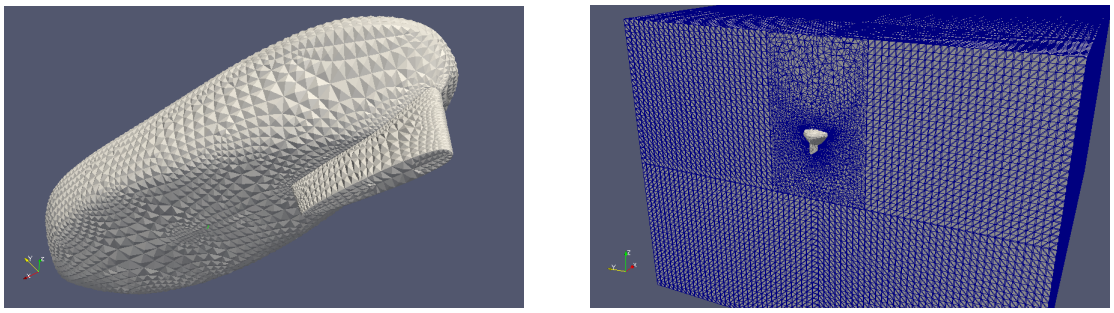

Abbildung 6.19: Von **vmesh** erstelltes Netz für die Vergleichsrechnung

In Abbildung 6.20 sind die Ergebnisse einer solchen Vergleichsrechnung dargestellt. Auf der y-Achse ist dabei die Kraft in x-Richtung dargestellt. Auf der x-Achse sind die Zeitschritte aufgetragen. Ein Zeitschritt entspricht dabei 0.001 Sekunden. Der Voith Wassertrecker wurde zur Berechnung auf Modellgröße skaliert. Die grüne Kurve zeigt dabei die Widerstandskraft für das manuell erstelle Netz mit 3.6 Mio Zellen. Die rote Kurve zeigt den zeitlichen Verlauf der Widerstandskraft des manuell erstellten Netzes, wobei ein Großteil der Zellen zwischen Schiff und Auslass des Strömungsquaders bei der Strömungsberechnung nicht berücksichtigt wurden (Anzahl der Zellen bei der Strömungsberechnung ca. 1.3 Mio Zellen). In blau wurde der Verlauf des von **vmesh** automatisch erstellten Netzes dargestellt. Dabei wird deutlich, dass schon eine deutlich geringere Zellenanzahl zu gleich guten Ergebnissen führt.

Eine Erhöhung der Zellzahl auf 2.2 Millionen Zellen bringt, wie in Abbildung 6.21 ersichtlich ist, keine weitere Verbesserung mehr.

Ein etwas gröberes Netz mit 0.94 Millionen Zellen und ähnlichen Einstellungen liefert jedoch schon deutlich schlechtere Ergebnisse (Abb.: 6.22).

In den Abbildungen 6.23, 6.24 und 6.25 sind die dazugehörigen Wellenbilder (links: manuell, rehts: **vmesh**) dargestellt. Dabei wird ersichtlich, dass das Wellenbild beim manuell erstellten Netz deutlich schöner ist. Dies liegt zum einen daran, dass beim manuell erzeugten Netz, speziell beim Phasenübergang Wasser-Luft, sehr fein aufgelöst wurde. Bei dem mit **vmesh** erstellten Netz wurde das nicht gemacht. Außerdem sind beim manuell erstellten Netz mehr Zellen eingearbeitet. Zum anderen scheinen Tetraeder für solche Zwecke ungeeigneter als Hexaeder zu sein, da es deutlich einfacher ist, bei einem Hexaedernetz die Gitterlinien parallel zur Phasengrenze laufen zu lassen als bei einem Tetraedernetz (Abb.: 6.26).

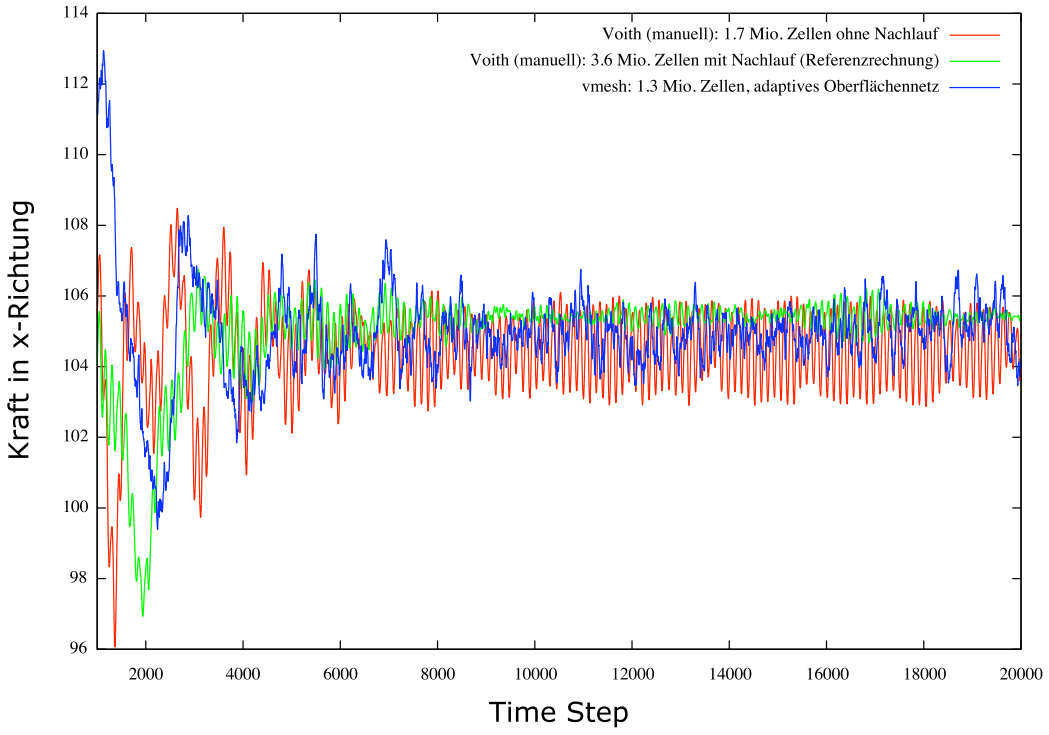

Abbildung 6.20: Vergleichsrechnung zum Voith Wassertrecker mit manuell erstelltem Netz (1)

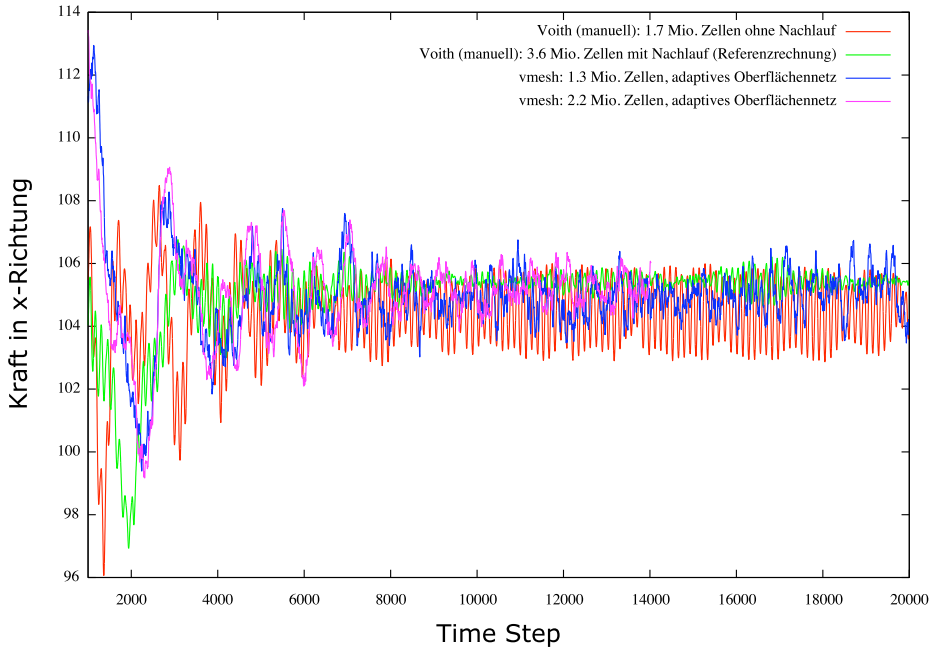

Abbildung 6.21: Vergleichsrechnung zum Voith Wassertrecker mit manuell erstelltem Netz (2)

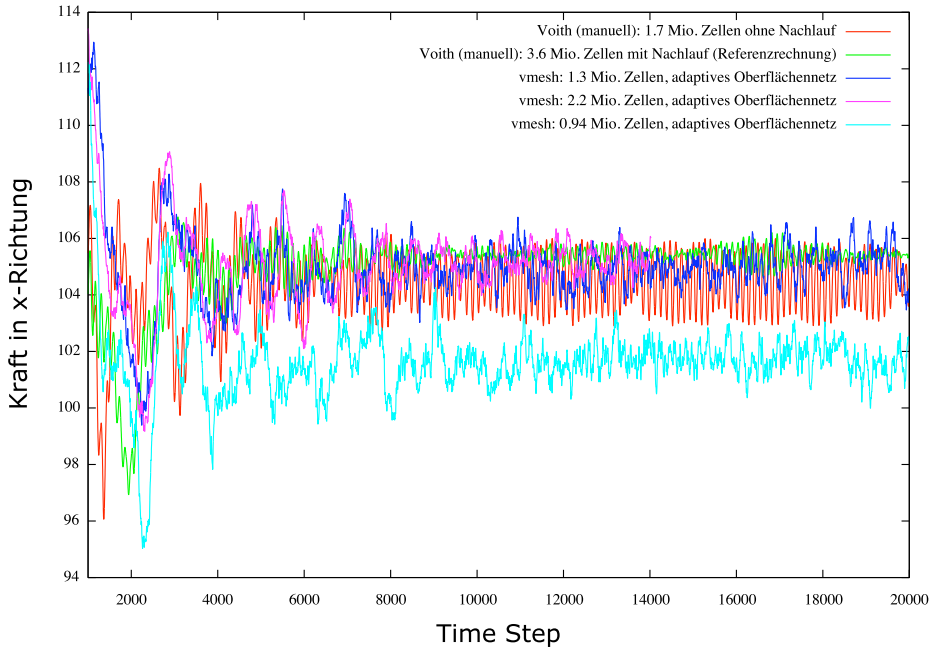

Abbildung 6.22: Vergleichsrechnung zum Voith Wassertrecker mit manuell erstelltem Netz (3)

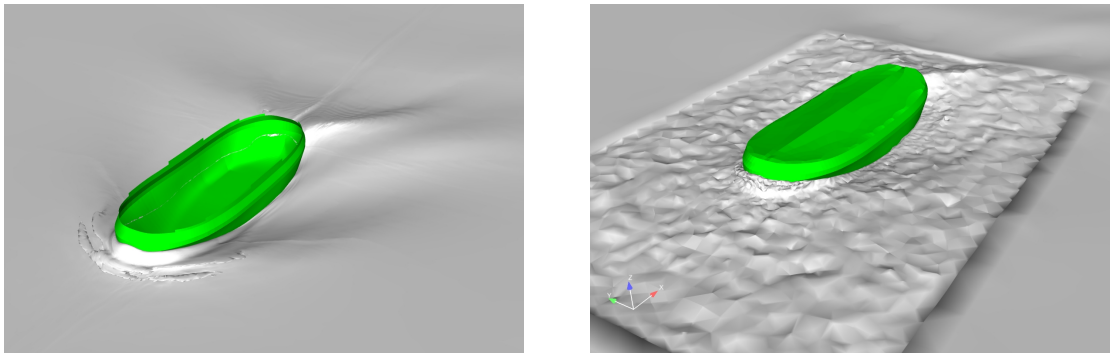

Abbildung 6.23: Strömung am Schiff (1)

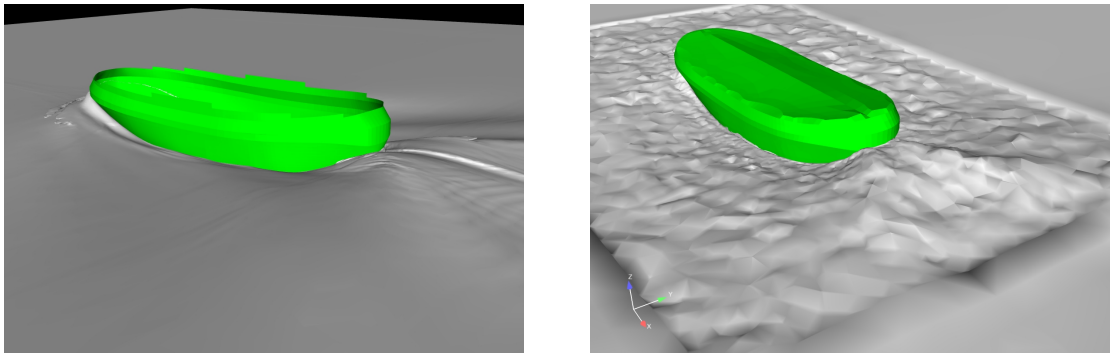

Abbildung 6.24: Strömung am Schiff (2)

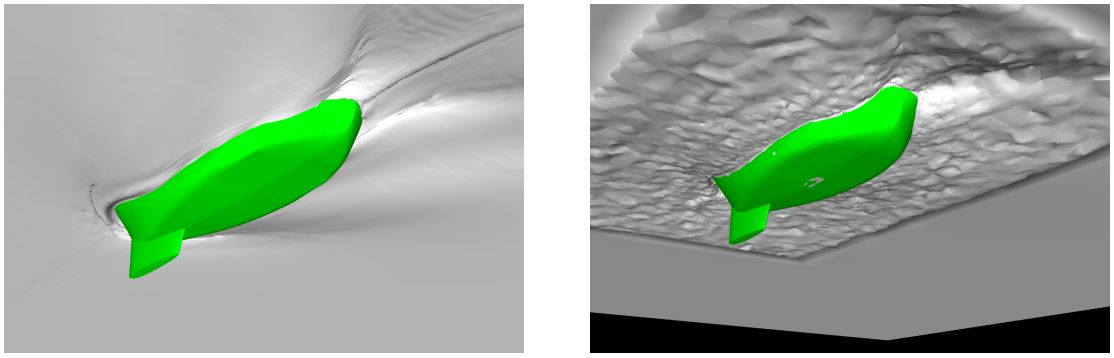

Abbildung 6.25: Strömung am Schiff (3)

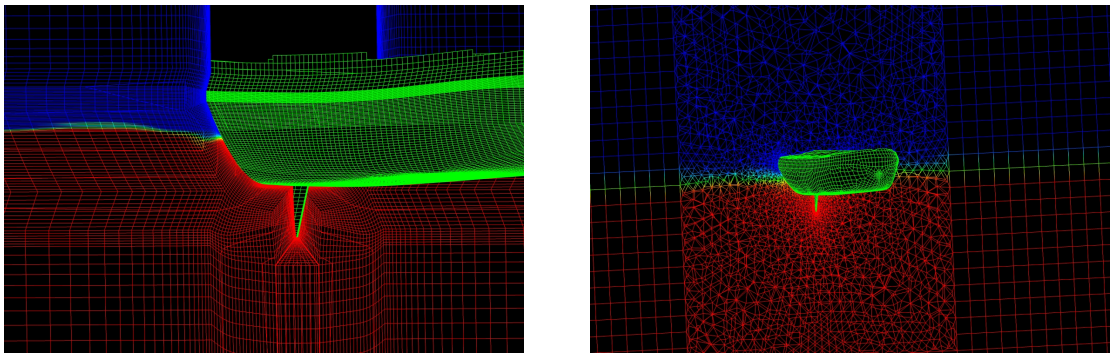

Abbildung 6.26: Generierte Netze: Links manuell erzeugt; Rechts mit **vmesh** erzeugt

6.2.2 Vergleichsrechnung für den Voith Wassertrecker mit Finne: Ein automatisierter TetMesher (THex)

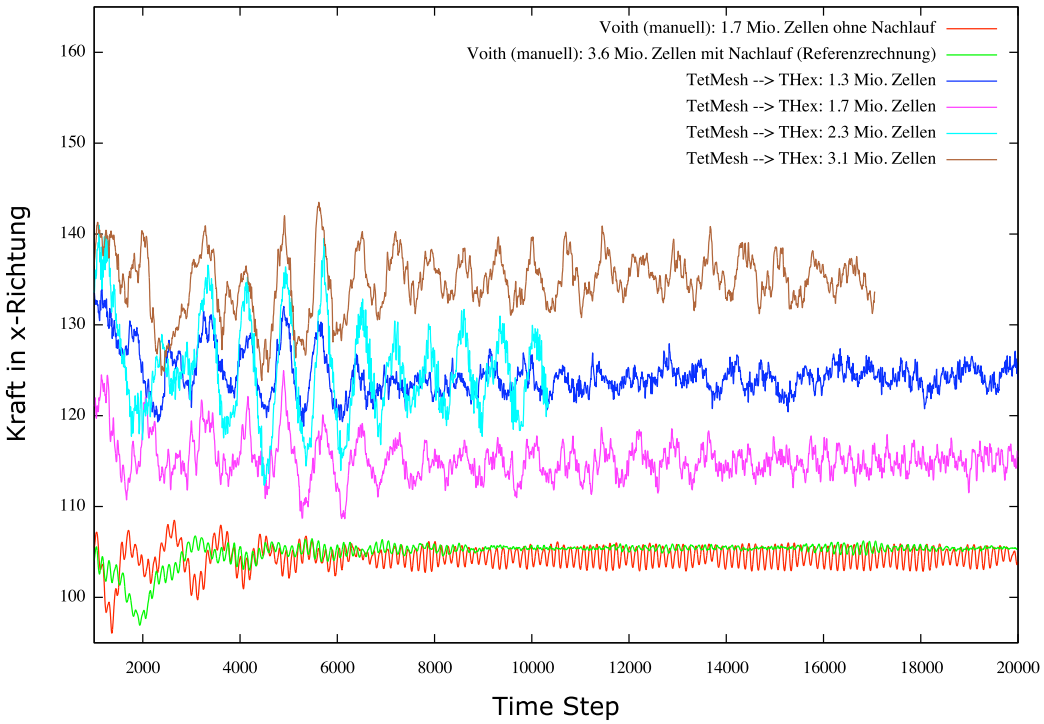

Abbildung 6.27: Vergleichsrechnung mit automatisiertem TetMesher (THex) von CUBIT

Um die Qualität der von CUBIT mit THex (Hexaedervernetzer) generierten Netze zu testen, wurde der komplette Strömungsraum mit Hexaedern gefüllt. Dabei wird der Strömungsraum zuerst mit Tetraedern gefüllt und anschließend wird jedes dieser Tetraeder in vier Hexaeder zerteilt.
Es wurde auch untersucht, ob es eine ausreichende Verfeinerung gibt, die vielleicht schon an die Widerstandswerte des manuell erstellten Netzes und des von **vmesh** erstellten Netzes heranreicht.

Als Ergebnis dieser Vergleichsrechnung kann festgehalten werden, dass dieser THex Vernetzer von CUBIT bei der Finiten-Volumen-Methode nicht ausreicht um vergleichbare Ergebnisse, wie bei der Strömungsberechnungen mit dem von **vmesh** automatisch generierten Netze, zu erzielen.
Dabei wurden Netze von 1.7 Millionen Zellen bis 3.1 Millionen Zellen generiert und dann die gleichen Strömungsberechnungen durchgeführt. Eine Verfeinerung des Netzes hat dabei nicht immer eine Verbesserung hervorgebracht. Bei 3.1 Millionen Zellen waren die Ergebnisse sogar am schlechtesten (siehe Abbildung 6.27).

Auswertung der Gitterkriterien:

Um diese Ergebnisse zu untermauern, werden im Folgenden die Gitterkriterien für Hexaeder ausgewertet. Dazu wurde das Netz mit 2.3 Mio. herangezogen. In Abbildung 6.28 ist dieses Netz dargestellt.

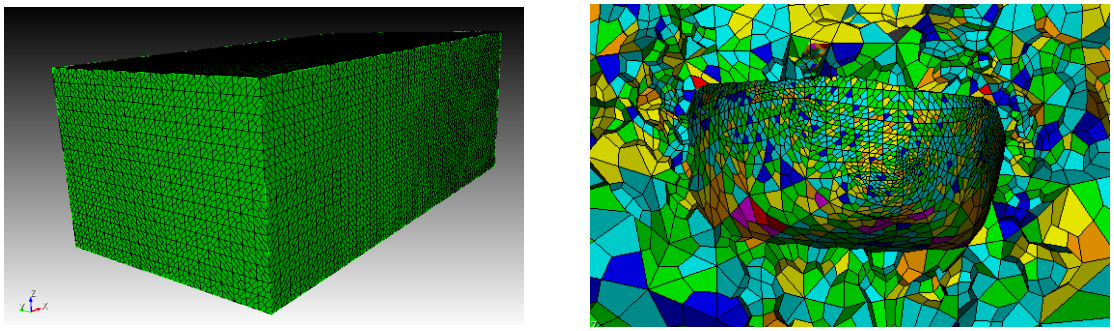

Abbildung 6.28: Links das komplette Netz; Rechts: Schnitt durch das Netz

Dieses Netz wurde auf die verschiedenen Kriterien für Hexaeder (Kapitel 2.1.5) getestet.

Diagonalenverhältnis:

Für das Diagonalenverhältnis ist $[0.65, 1]$ ein akzeptables Intervall . In Abbildung 6.29 ist das deutlich nicht erfüllt. Mehr als die Hälfte der Zellen erfüllt dieses Kriterium nicht.

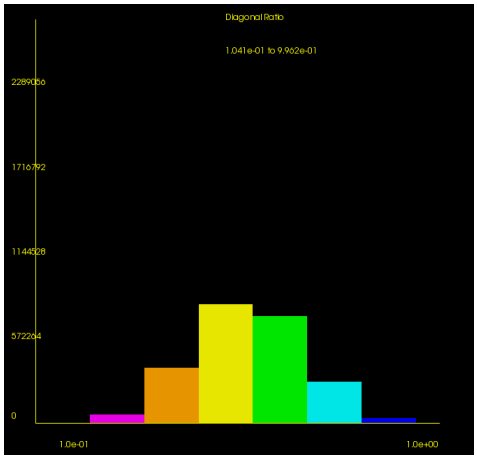

Abbildung 6.29: Säulendiagramm zum Diagonalenverältnis

Shape:

Ein akzeptables Intervall für das Shape ist $[0.3, 1]$. Laut Abbildung 6.30 trifft dies zu.

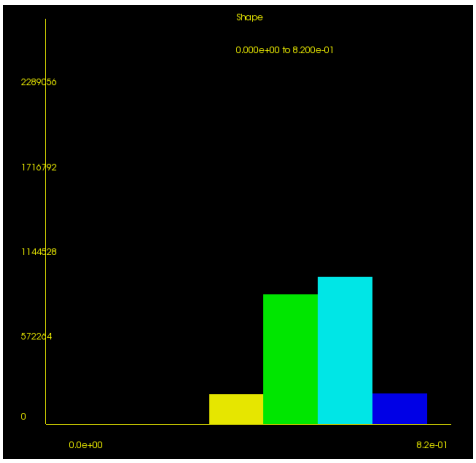

Abbildung 6.30: Säulendiagramm zum Shape

Skalierte Jacobi-Determinante:

Die skalierte Jacobi-Determinante sollte im Intervall $[\frac{1}{2}, 1]$ liegen. In Abbildung 6.31 ist auch das ebenfalls nicht erfüllt. Etwa die Hälfte der Zellen liegt ausserhalb der Range.

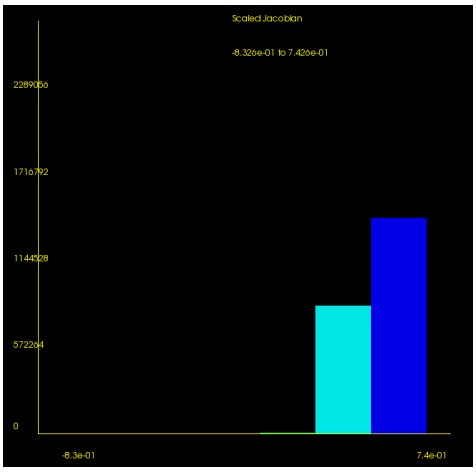

Abbildung 6.31: Säulendiagramm zur skalierten Jacobi-Determinante

Fazit:
Die schlechten Ergebnisse der Strömungsberechnungen spiegeln sich in der Auswertung der Gitterkriterien wieder.

6.2.3 Vegleichsrechnung für den Voith Wassertrecker mit Finne: Mit und ohne innere Schicht

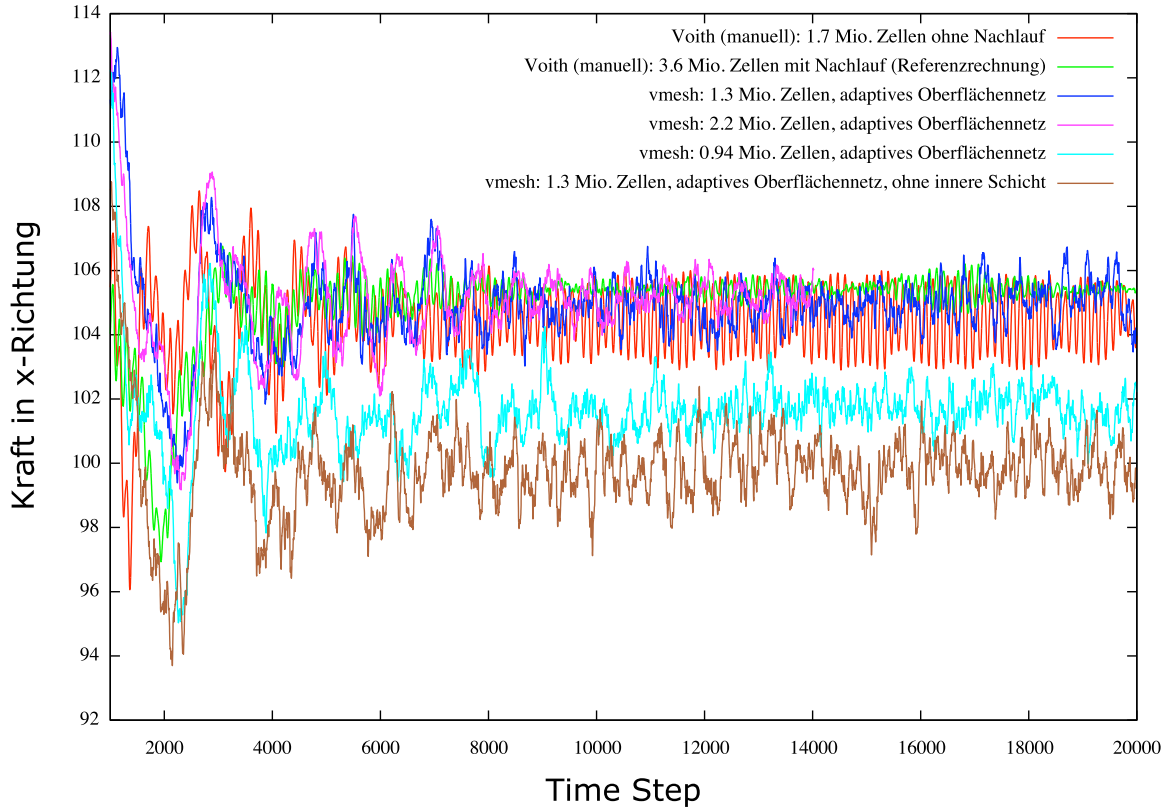

Abbildung 6.32: Vergleichsrechnung für den Voith Wassertrecker mit Finne: Mit und ohne innere Schicht

Wird bei dem Netz mit 1.316 Millionen Zellen (dunkelblau) die innere Schicht weg gelassen und die restlichen Einstellungen beibehalten (braun), so wird in Abbildung 6.32 sofort eine deutliche Verschlechterung der Netzquälitatt beobachtet. Dies zeigt, wie wichtig eine solche Grenzschicht für diese Strömungsberechnungen ist.

6.2.4 Vegleichsrechnung für den Voith Wassertrecker ohne Finne

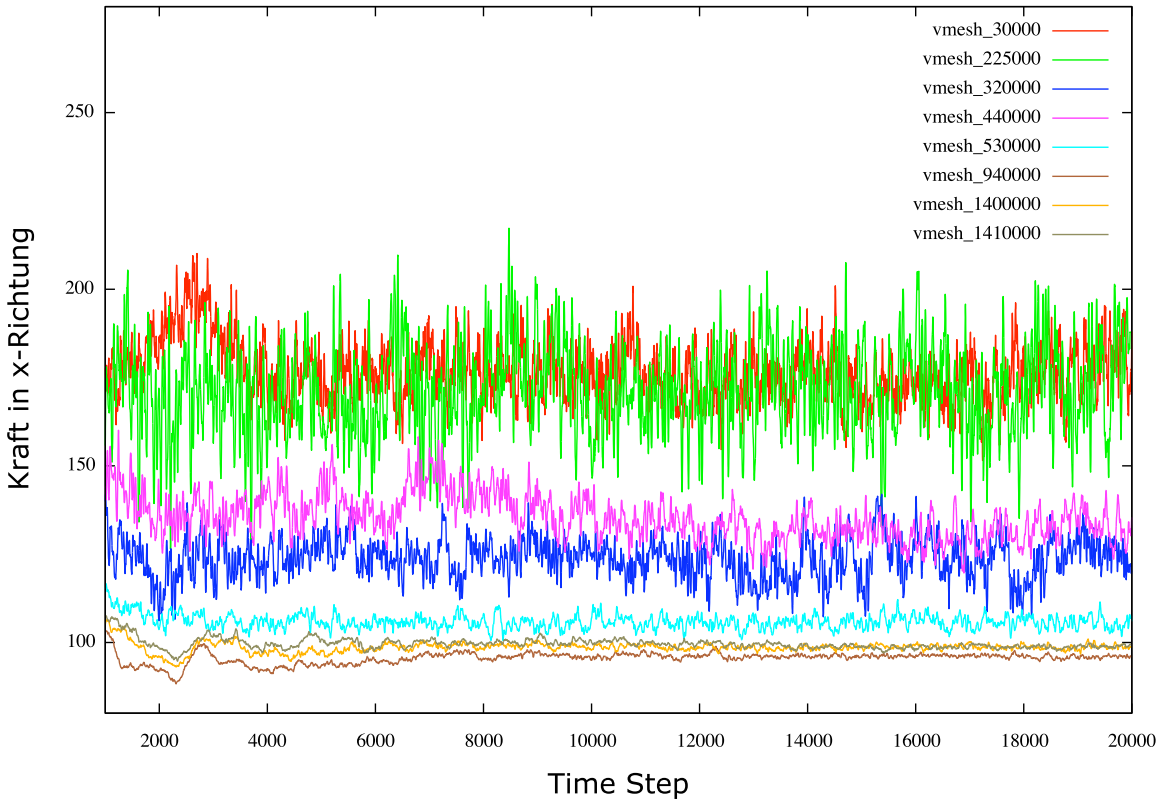

Abbildung 6.33: Vergleichsrechnung für den Voith Wassertrecker ohne Finne: Die dazugehörigen Netze wurden von **vmesh** generiert

In Abbildung 6.33 ist erkennbar, dass eine Verfeinerung deutlich bessere Ergebnisse liefert. Zum einen haben die Kraftwerte deutlich weniger Abstand zur realen Widerstandskraft, und zum anderen sind die Schwingungen bei den jeweiligen Strömungsberechnungen signifikant kleiner, was bei einer Optimierung mit der Kraft als Funktion natürlich unabdingbar ist.

6.3 Ergebnisse aus der geometriebasierten Optimierung

Zuerst werden die dazugehörigen Netze dargestellt. Anschließend sind die Ergebnisse einer Designstudie veranschaulicht.

6.3.1 Voith Wassertrecker-v107

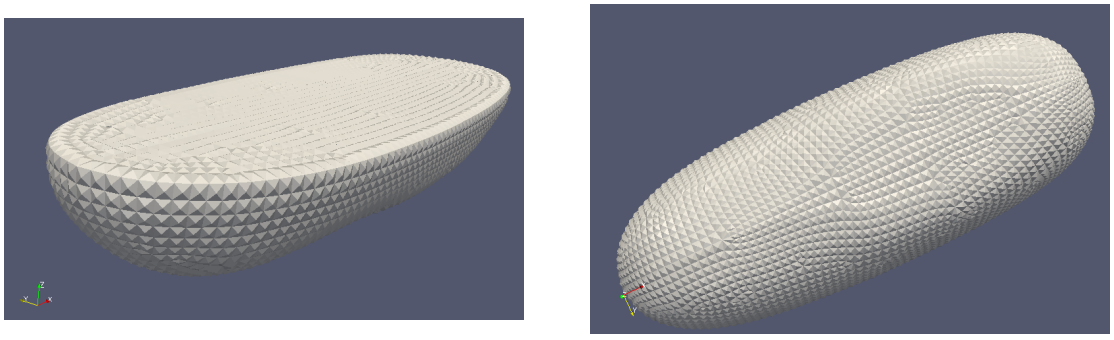

Abbildung 6.34: Voith Wassertrecker-v107: Innere Schicht mit Pyramiden

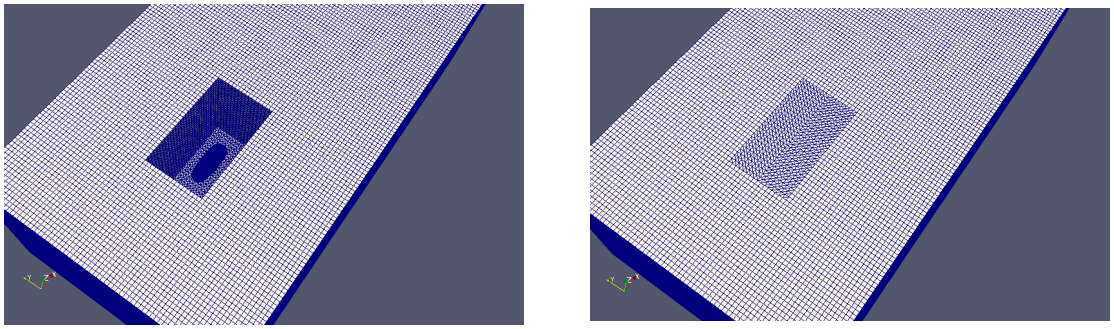

Abbildung 6.35: Voith Wassertrecker-v107: Gesamtvolumennetz

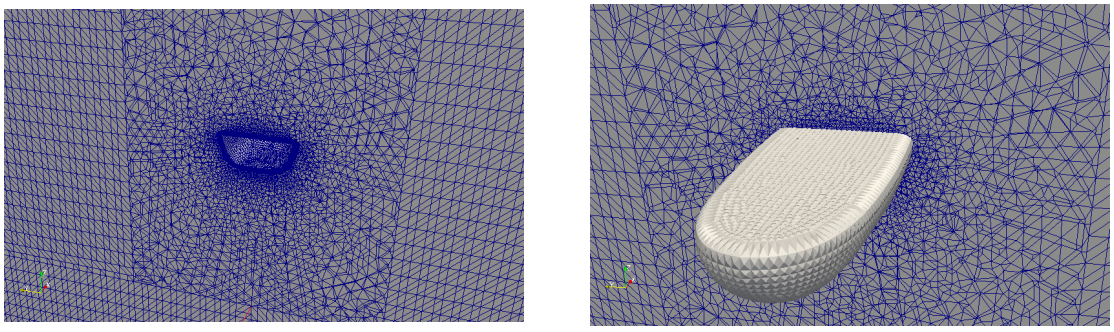

Abbildung 6.36: Voith Wassertrecker-v107: Schnitt durch das Gesamtvolumennetz

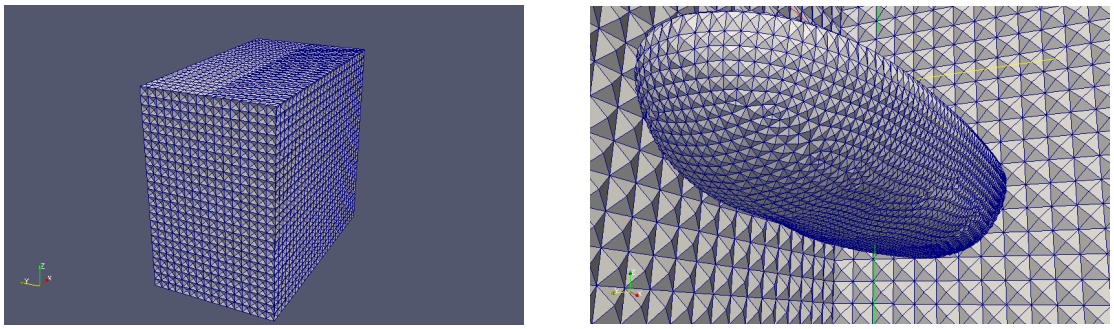

Abbildung 6.37: Voith Wassertrecker-v107: Stl der mittleren Schicht

6.3.2 Einstellung der Netzparameter

	Voith Wassertrecker-v107
Anzahl: Zellen	1432945
Zellgrösse: zg	170
Deckel: oben	3378
Boden: unten	-6622
Seite: seite	7000
Vordere Fläche: vorne	20500
Hintere Fläche: hinten	-7000
pu	-1500
ps	1500
pv	4000
ph	-1500
Genauigkeit beim vmerge: eps	0.5
Skalierung: scaling	1
Pyramidenrichtung: change	-1
Dicke einer Zelle: thickness	1
Anzahl von Zellschichten: num of lay	16
Streckung: dilation	1.09
Offset-Algorithmus: p	1
Paving:	1.7 (auto)
Glättung des Oberflächennetzes	Laplace

6.3.3 Designstudie

Vor dem Beginn der Optimierung, wurden Designstudien durchgeführt. Das Ziel war es die Parameter der Geometrieveränderung den Parametern des Netzes anzupassen. Dabei ist es wichtig die Netzparameter und den Blendingfaktor so einzustellen, dass die Veränderungen der Geometrie in der nachfolgenden Strömungssimulation sichtbar sind. In Abbildung 6.38 (rechts) sind Geometrieänderungen deutlich zu erkennen. Das

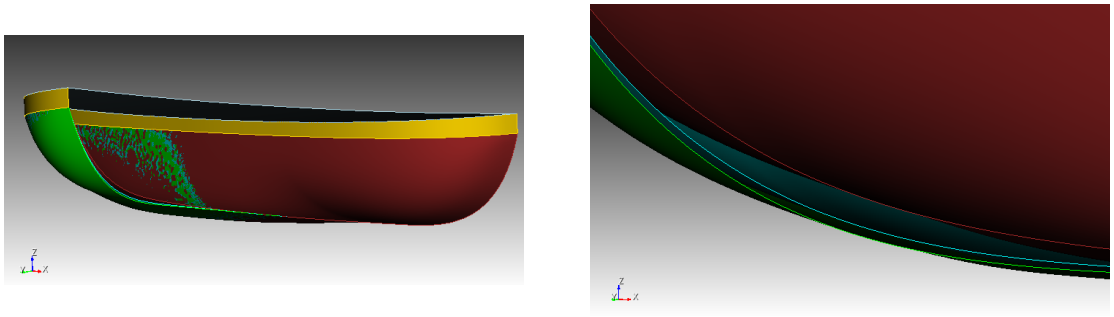

Abbildung 6.38: Voith Wassertrecker-v107: Veranschaulichung der Geometrieveränderung

Ausgangsschiff (grün) wurde am Bug durch den *PaShiMo* durch eine Geometriemodifikation etwas 'schlanker' gemacht. Eine kleine Veränderung ist beim blauen Schiff und eine etwas größere beim roten Schiff zu erkennen. In Abbildung 6.39 sind die ent-

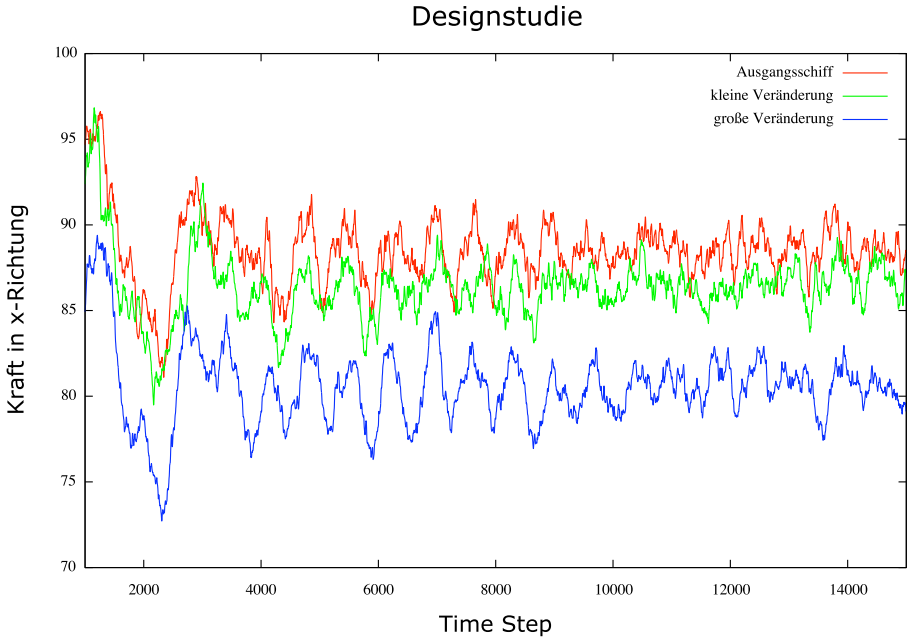

Abbildung 6.39: Veranschaulichung der Ergebnisse der Strömungssimulation der Designstudie (1)

sprechenden Ergebnisse der Strömungssimulationen dargestellt. Die Veränderungen der

Geometrie sind bei der Simulation deutlich wahrgenommen worden.
Eine mittlere Veränderung (zwischen dem blauen und roten Schiff) der Geometrie war jedoch bei der Strömungssimulation (Abbildung 6.40) nicht entsprechend abgebildet.

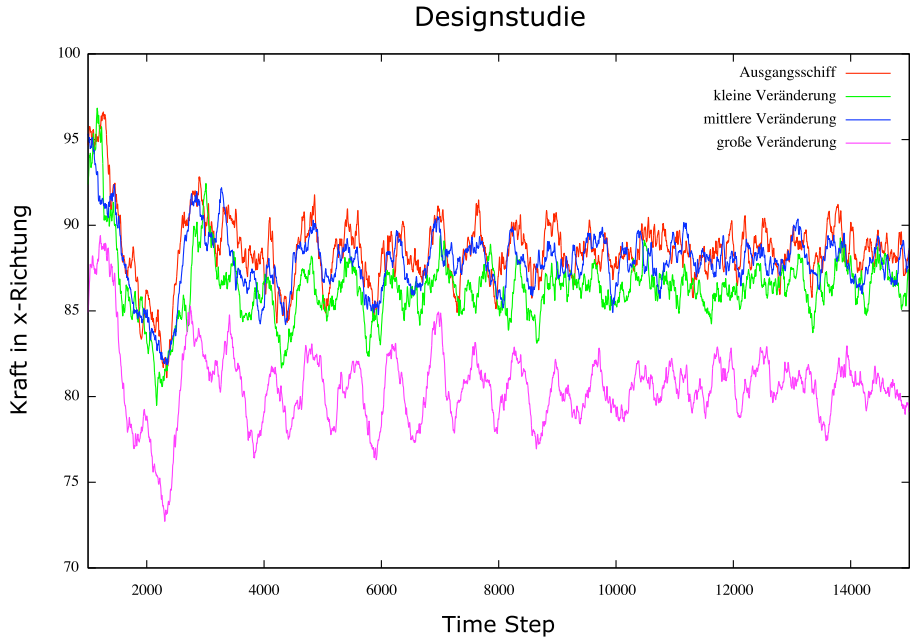

Abbildung 6.40: Veranschaulichung der Ergebnisse der Strömungssimulation der Designstudie (2)

Hier kann eine Verfeinerung des Netzes oder eine Vergößerung des Blendingfaktors (Kapitel 3.2.4) Abhilfe schaffen.

Weitere Ergebnisse der geometriebasierten Optimierung sind in [47] aufgezeigt.

6.4 Ergebnisse aus der netzbasierten Formoptimierung

Bisher wurden erste vielversprechende Testläufe durchgeführt, wobei ein Testlauf eindimensionaler Natur war. Der Optimierungsparameter war *dy* und andere zweidimensionale Versuche wurden mit *dy* und *y_p* als Optimierungsparameter durchgeführt.
Verwendet wurde das von **vmesh** generierte Netz mit 1.3 Mio. Zellen, welches in Abschnitt 6.2.1 die besten Ergebnisse liefert. Im Folgenden sollen die Ergebnisse der eindimensionalen Optimierung dargestellt werden.
Zuerst wurden für den Parameter *dy* Tests durchgeführt, wie sich die Schiffsgeometrie ändert, wenn der Parameter verändert wird. Dadurch ergab sich ein Intervall für diesen Parameter, in welchem gewährleistet ist, dass zum einen keine Überlappungen des Netzes entstehen und zum anderen, dass die Deformationen nicht zu groß sind (Abb.: 6.41).

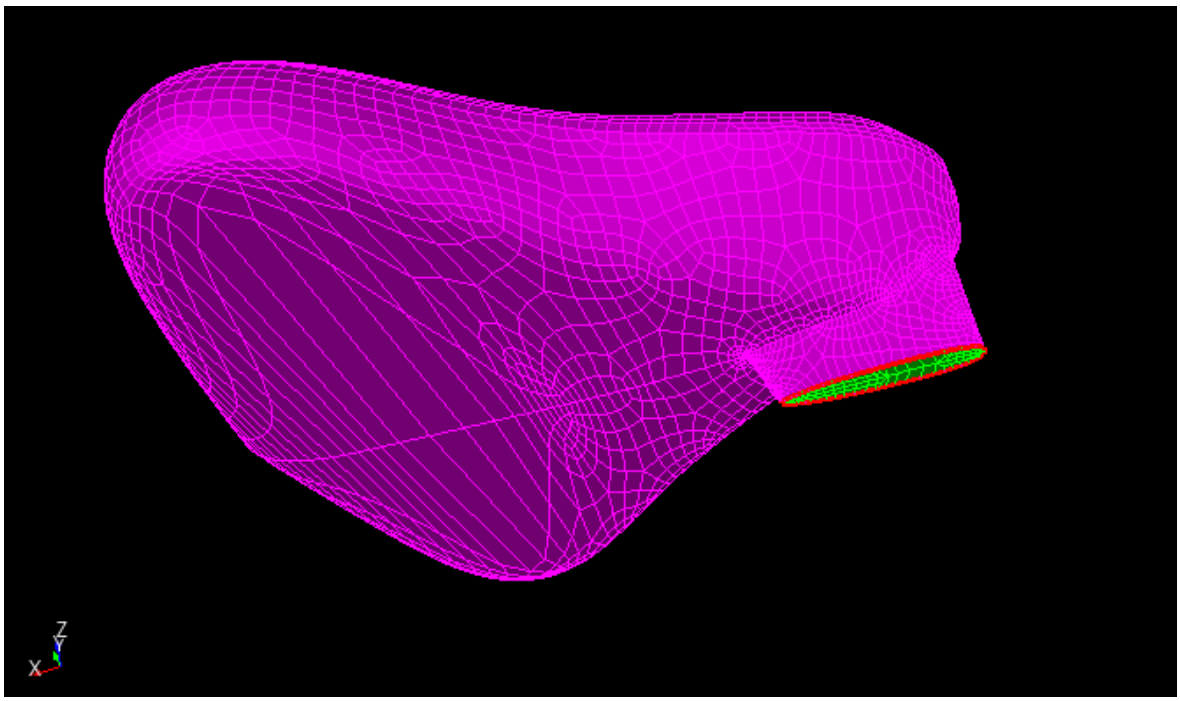

Abbildung 6.41: Oberflächennetz des Schiffes bei zu größer Deformation

Mit diesem Intervall für den Parameter *dy* wurden erste Optimierungen durchgeführt. Ergebnisse sind in Abbildung 6.42, 6.43, 6.44 und 6.45 dargestellt. Das gelbe Oberflächennetz ist das Netz des Ausgangsschiffes und das blaue Netz, das des resultierenden Schiffes.

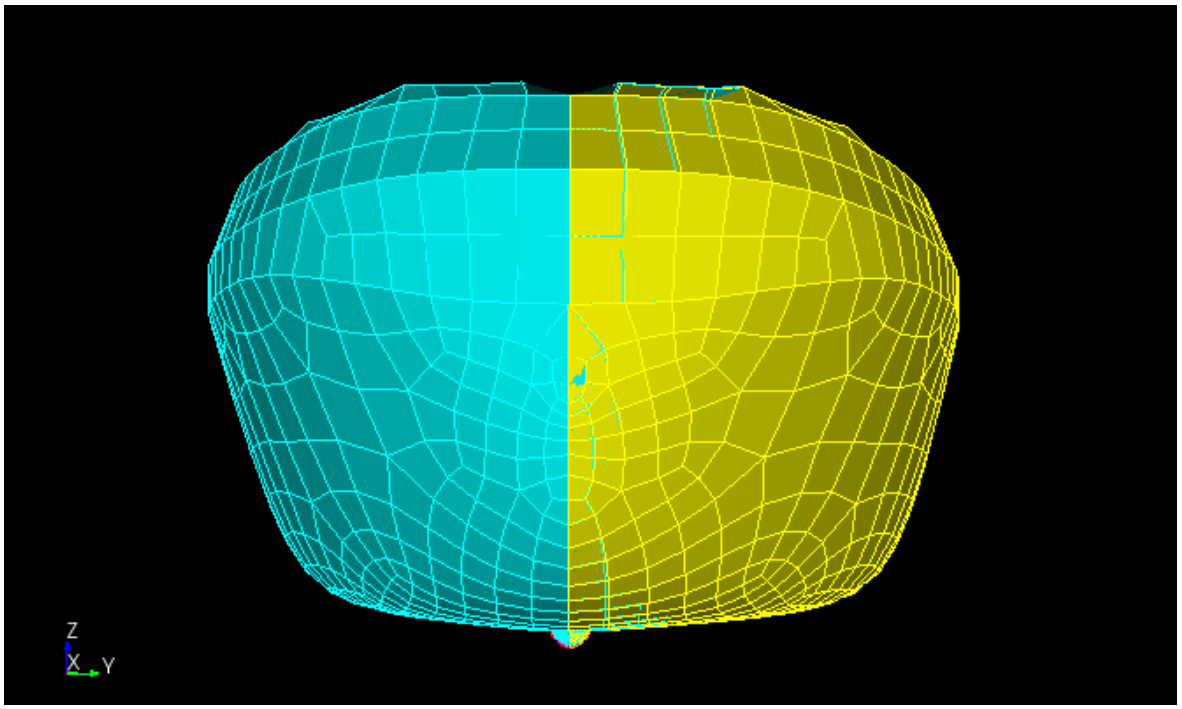

Abbildung 6.42: Optimierungsdurchlauf - netzbasiert (1)

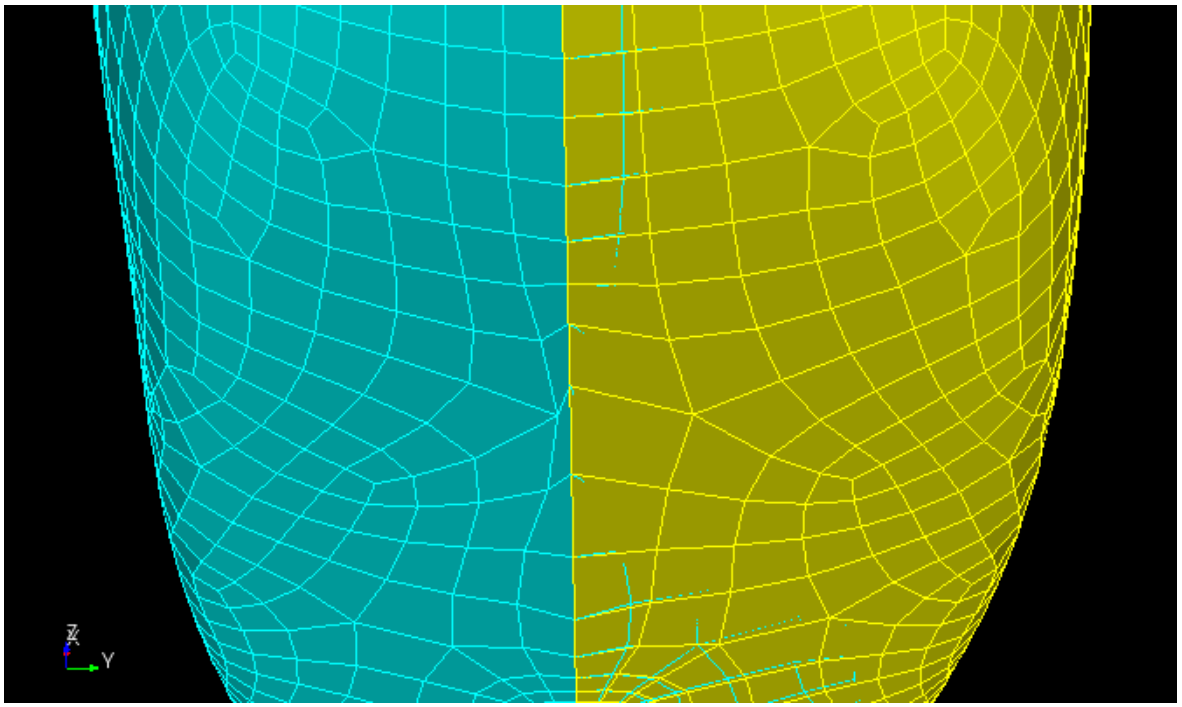

Abbildung 6.43: Optimierungsdurchlauf - netzbasiert (2)

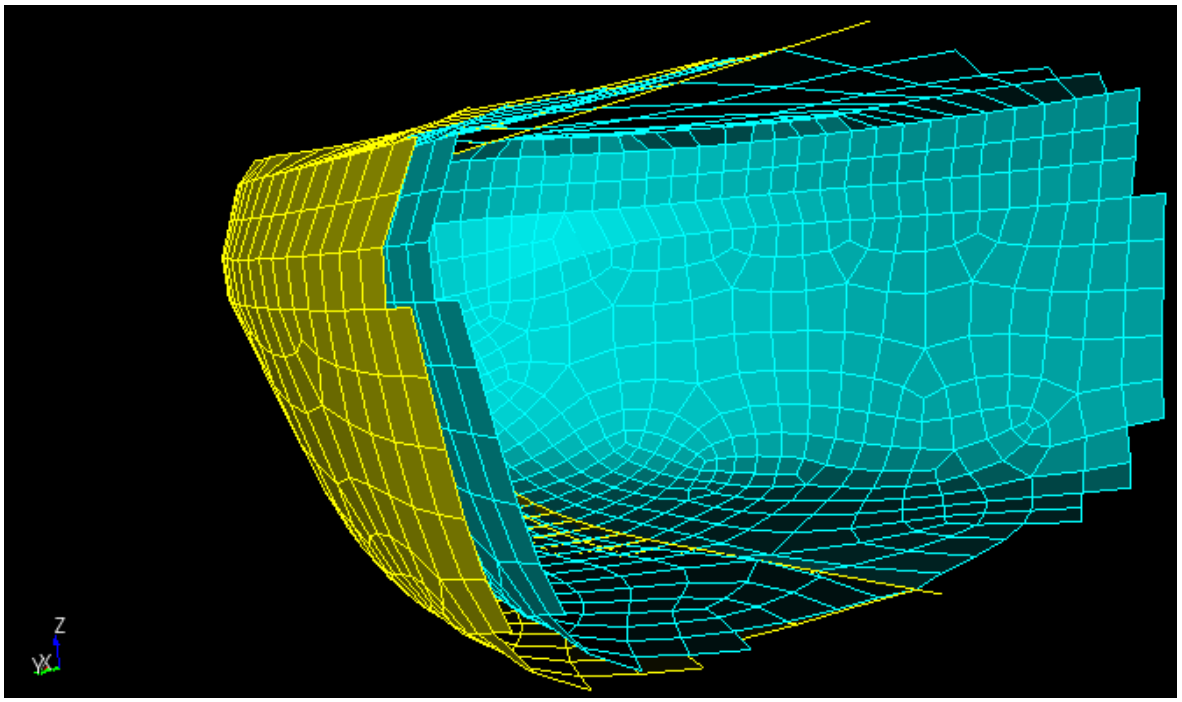

Abbildung 6.44: Optimierungsdurchlauf - netzbasiert (3)

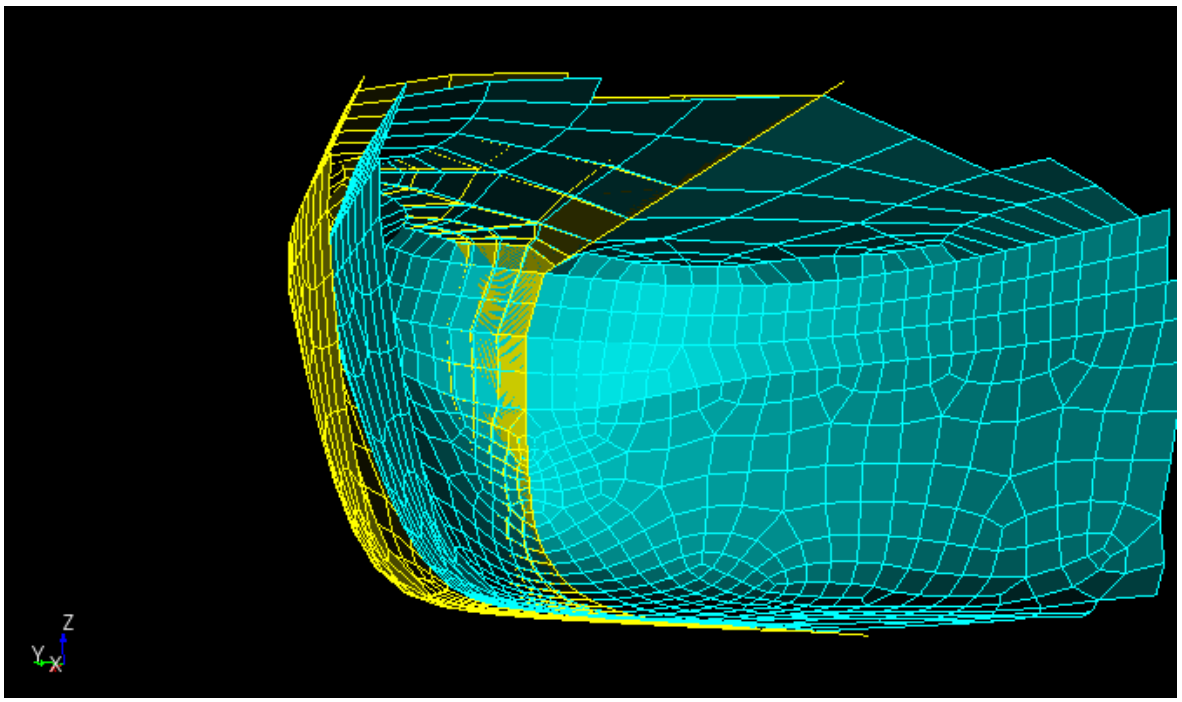

Abbildung 6.45: Optimierungsdurchlauf - netzbasiert (4)

War die Deformation bei der Optimierung zu gering, dann hatte der Optimierer Schwierigkeiten, das Minimum zu finden. Das liegt daran, dass zu jedem Netz eine resultierende Ungenauigkeit (numerische Diffusion, ...) in der Simulation gibt. Ein zu grobes Netz oder zu kleiner Blendingfaktor können diesen Effekt noch verstärken. Wenn diese Ungenauigkeit eine Abweichung vom tatsächlichen Wert des Zielfunktionals hervorruft, die größer als die Veränderung des Zielfunktionals durch die Deformation ist, dann kann der Optimierer die richtige Abstiegsrichtung nicht finden (Abb.: 6.46 und 6.47). Die Erkundungsschritte sind beim Hooke-Jeeves Algorithmus zur Verdeutlichung dieses Phänomens in der Grafik mit aufgenommen.

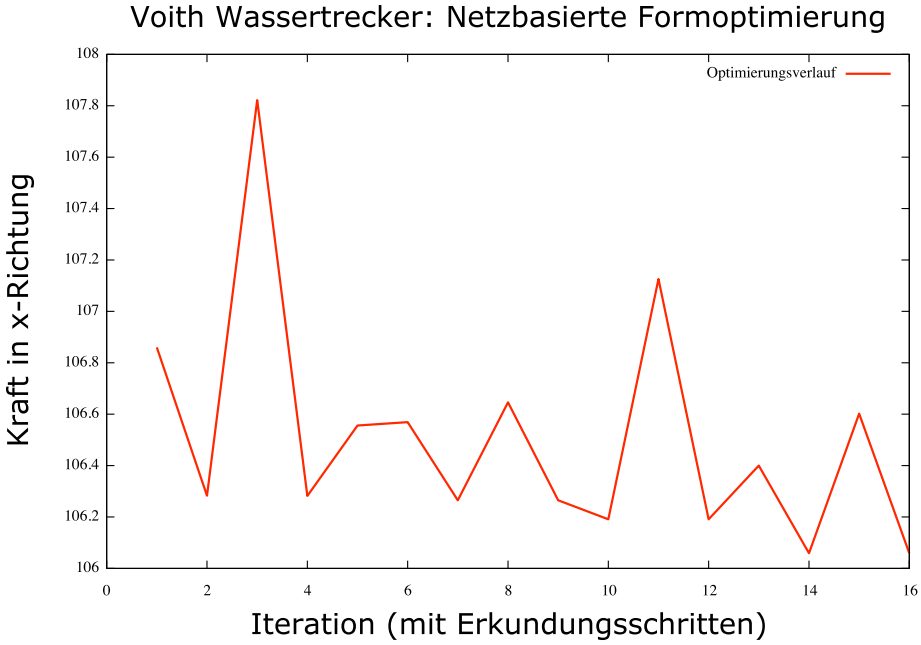

Abbildung 6.46: Optimierungsdurchlauf mit zu geringer Deformation (Startwert: $dy = 0.16m$)

Ist die Deformation entsprechend eingestellt, verläuft die Optimierung reibungslos ab (Abb.: 6.48). Vergleichen wir die Kraftwerte des Zielfunktionals des Ausgangsschiffes mit dem resultierenden Schiff miteinander, so lassen sich doch erhebliche Verbesserungen feststellen.

Bei dieser Optimierung sind bisher noch keine Nebenbedingungen realisiert.

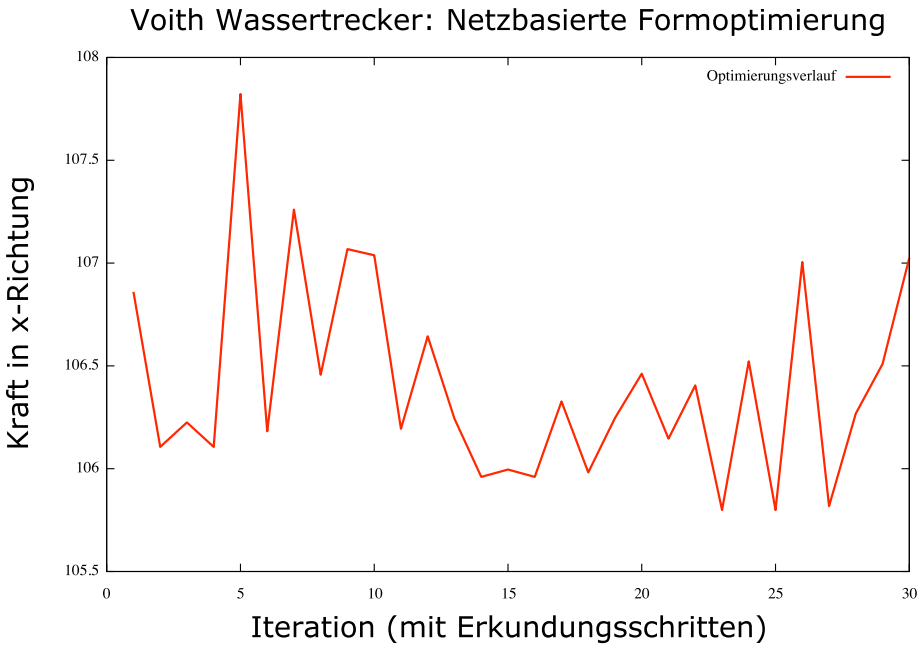

Abbildung 6.47: Optimierungsdurchlauf mit geringer Deformation (Startwert: $dy = 0.32m$)

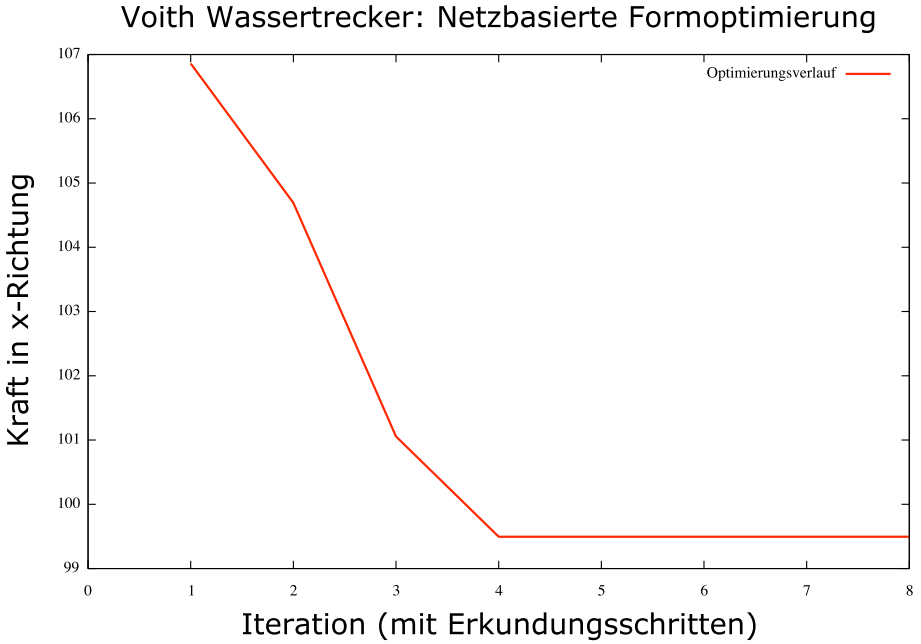

Abbildung 6.48: Optimierungsdurchlauf mit angepaßter Deformation (Startwert: $dy = 0.8m$)

Kapitel 7

Ausblick

Neben den erzielten Fortschritten und Ergebnissen haben sich in der vorliegenden Dissertation zahlreiche neue Fragestellungen und Ansatzpunkte für weitere Arbeiten ergeben.

Um die Anwendung des Vernetzungstools **vmesh** zu erleichtern, ist es angebracht eine benutzerfreundliche Bedienungsoberfläche (GUI) bereitzustellen, bei welcher die Parameter für das Oberflächennetz (Kapitel 4.4.2) und die Netzparameter für das Volumennetz (Kapitel 4.12) nur noch manuell eingegeben werden müssen.
Bei der Vernetzung ist es auch sinnvoll eine anisotrope Netzverfeinerung in z-Richtung am Phasenübergang Wasser-Luft zu realisiern. Hier steckt die Schwierigkeit im hybriden Netz. Für Hexaeder ist dies leicht umzusetzen. Bei der Tetraederschicht ist es allerdings deutlich schwieriger.
Eine Netzoptimierung (Kapitel 2.1.6) des Gesamtvolumennetzes kann im Anschluss an eine Vernetzung angebracht sein. Die Umsetzung ist relativ unaufwendig.

Bei der Optimierung muss der VSP mit all seinen Anbauteilen integriert werden. Neue Verfahren können bei beiden Optimierungsmethoden getestet werden (Kapitel 5.2.7). Durch die Anwendung der adjungierten Methode oder des Automatischen Differenzierens könnte man gradientenbasierte Verfahren anwenden.
Um eine sinnvolle Optimierung zu gewährleisten kann es sinnvoll sein, Zusammenhänge der Netzparameter mit den vorgenommenen Deformationen der Geometrie und des Blendingfaktors (Kapitel 3.2.4), herauszuarbeiten.

Die netzbasierte Formoptimierung kann weiter ausgebaut werden. Es ist sinnvoll Nebenbedingungen, wie zum Beispiel Verdrängung oder Einschränkungen der Optimierungsparameter in die Optimierung mit einzubinden. Außerdem muss die Dimension der Optimierunsparameter erhöht werden. Es kann auch angebracht sein, die Parameter des Netzdeformierungsalgorithmus (Kapitel 5.2.2) auf dessen Auswirkungen bei der Optimierung zu testen.

Anhang A

Beispiel einer vsp.echo Datei

```
star vmesh.vrt vmesh.cel
cset all
cche all righ y
cmod righ cset
cset all

gcol regi

vmer 1 4000000 0.1 low

csac 1
rsur 1 -1 0 0 - - -6999.99
rset all
rsur 2 1 0 0 - 20369.99
rset all
rsur 3 0 -1 0 - - - - -6969.99
rset all
rsur 4 0 1 0 - - - 6969.99
rset all
rsur 5 0 0 1 - - - - - 3407.99
rset all
rsur 6 0 0 -1 - - - - - - -6621.99
rset all

rsur 7 1 0 0 360
rset all

** CSYS1 in Massenschwerpunkt verschieben
vset all
csac 1
vgen 1 0 vset - - -1187.5 0. -481.25
```

```
vset all
vsca vset 0.001 0.001 0.001 1

** General Setup:

** number of iterations per time step and convergence criterion
ITERATION 1 0.001
** AMG rocks:
SOLVER AMG CGSTAB IC
** number of time steps and time step size:
TIME TRANSIENT 70000 0.01 IEULER
** moving grid option every iteration !!!
MGOP ITER

rest n n 10000

** Domain Setup:

** gravity (body forces):
BODY 0 0 -9.81 0 0 0 STANDARD
** switching moving grid on:
MOVGR ON
** setting reference pressure (best at a cell which is
** expected to be never in water):
MACT 0
RPRE 1e+05 126750

** Specifying what to solves:
mome on 0.7 cd 0.0
mass on
TURB KE 0.7
** activating species
SPEC 1 AIR FREE 0.9 HRIC 0.9
** selecting species properties:
MACT 1
SPEC 1 WATER

BDEF 1 INLE STAN 1 NONE 1
BVAL MOME 1
0.0 0.0 0.0 1 0.0 Y
BVAL MASS 1
1.188 N
BVAL ENER 1
```

```
TEMP 293
BVAL TURB 1
TL 0.1 0.001
BVAL SP001 1

BDEF 2 PRES STAN 1 NONE 1
BVAL MOME 2
STAT 0.0
BVAL ENER 2
ZERO
BVAL TURB 2
ZERO
BVAL SP001 2

BDEF 7 WALL STAN 1 NONE 1
BVAL MOME 7
NOSL 0.0 0.0 0.0 1 0.0 90
BVAL ENER 7
ADIA
BVAL TURB 7
WALL 9.0
BVAL SP001 7

BDEF 3 INLE STAN 1 NONE 1
BVAL MOME 3
0.0 0.0 0.0 1 0.0 Y
BVAL MASS 3
1.188 N
BVAL ENER 3
TEMP 293
BVAL TURB 3
TL 0.1 0.001
BVAL SP001 3

BDEF 4 INLE STAN 1 NONE 1
BVAL MOME 4
0.0 0.0 0.0 1 0.0 Y
BVAL MASS 4
1.188 N
BVAL ENER 4
TEMP 293
BVAL TURB 4
TL 0.1 0.001
BVAL SP001 4
```

```
BDEF 5 INLE STAN 1 NONE 1
BVAL MOME 5
0.0 0.0 0.0 1 0.0 Y
BVAL MASS 5
1.188 N
BVAL ENER 5
TEMP 293
BVAL TURB 5
TL 0.1 0.001
BVAL SP001 5

BDEF 6 INLE STAN 1 NONE 1
BVAL MOME 6
0.0 0.0 0.0 1 0.0 Y
BVAL MASS 6
1.188 N
BVAL ENER 6
TEMP 293
BVAL TURB 6
TL 0.1 0.001
BVAL SP001 6

BDEF 8 WALL STAN 1 NONE 1
BVAL MOME 8
NOSL 0.0 0.0 0.0 1 0.0 90
BVAL ENER 8
ADIA
BVAL TURB 8
WALL 9.0
BVAL SP001 8

** Floting body control (no GUI)

** switching on
flbo stat on

** defining number of regions of the body

** to investigate and the region number
** here only one region with region-number 4:
flbo regions 1 7

** motion control + underrelaxation factor for the
```

```
** movement inside the time step
** here URF for translation and rotation is 0.7
** Translation in z-axis and rotation around y-axis
flbo motio 0.7 0.7 0 0 1 0 1 0

** gravity
flbo gravi - - -9.81

** moments of inertia for the body
flbo inert 2. - - - 2. - - - 2.

** mass of the body (400 KG) and releasing the motion (after 500 time steps)
flbo body 120.3 500000

** water plane definition (no GUI)

** plane-ID is 1
** distance is 0.
** normal vector is 0 0 1 in coordinate system 1
cpla 1 -0.10325 0 0 1 1

** Marine modul (no GUI)

** switching on
mare stat on

** hydrostatic boundary condition
** using plane-ID 1
mare hydro 1

** switching moving grid to AUTOMATIC
mare movi 6dof

** species setting for the boundary and initialisation
** using plane-ID 1
mare wate planid 1

** ship speed
mare inve 2.058

ndef oaut 4 - all
paco para
**paco seri
geom 1
```

```
quit save
```

Anhang B

Danksagung

An dieser Stelle möchte ich all jenen ganz herzlich danken, die durch ihre fachliche und persönliche Unterstützung zum Gelingen dieser Doktorarbeit beigetragen haben.

Vor allem danke ich meinem Doktorvater Prof. Karsten Urban für seine exzellente Betreuung und seine sehr hilfreichen Anregungen, sowie Prof. Stefan Funken für seine Bereitschaft, als Zweitgutachter zu fungieren.

Mein Dank für die hilfreiche Unterstützung geht auch an die Mitarbeiter der Abteilung Voith Turbo Schneider Propulsion, im Besonderen an Dirk Jürgens, Michael Palm, Sebastian Singer, und David Bendl. Die offene Atmosphäre und die hervorragende Unterstützung haben wesentlich zum Erfolg dieser Arbeit beigetragen.

Mein Dank gilt auch den vielen Kolleginnen und Kollegen des Instituts für Numerische Mathematik, die mich in vielfältiger Weise bei der täglichen Arbeit unterstützten und begleiteten. Ein besonderer Dank geht dabei an Juan Carlos Matutat für seine Hilfsbereitschaft und Kollegialität.

Meinem Bruder Martin Hopfensitz danke ich nicht zuletzt für die vielen erholsamen, gemeinsamen Mittagspausen.

Für die Unterstützung und den Rückhalt, den ich von meiner Frau Stefanie, meinen Eltern, Schwiegereltern und Geschwistern erfahren habe, möchte ich mich bei meiner ganzen Familie besonders herzlich bedanken.

Index

Literaturverzeichnis

[1] http://commons.wikimedia.org/wiki/File:Delaunay_Voronoi.png.

[2] http://cubit.sandia.gov/.

[3] http://de.wikipedia.org/wiki/Delaunay-Triangulation.

[4] http://de.wikipedia.org/wiki/Genetischer_Algorithmus.

[5] http://de.wikipedia.org/wiki/Rechengitter.

[6] http://de.wikipedia.org/wiki/Simulierte_Abkühlung.

[7] http://de.wikipedia.org/wiki/STL-Format.

[8] http://de.wikipedia.org/wiki/VTK.

[9] http://en.wikipedia.org/wiki/IGES.

[10] http://www-users.informatik.rwth-aachen.de/~roberts/meshgeneration.html.

[11] http://www-wnt.gsi.de/zt/cad-datenformate.htm.

[12] http://www.cfd.tu-berlin.de/lehre/tfd_skript/node70.html.

[13] http://www.ifu.ethz.ch/GWH/education/graduate/Hydraulik_II/Vorlesungen/k5.pdf.

[14] http://www.voith.de.

[15] http://www.voithturbo.de/vt_de_pua_marine_vspropeller.htm.

[16] Vtk file formats.

[17] J.-E. Bartels and D. Jürgens. The voith schneider propeller: current applications and new developments. Technical report, Voith Turbo Marine GmbH, 2006.

[18] T. D. Blacker. The cooper tool. Technical report, Fluid Dynamics International, 1996.

[19] T. D. Blacker and M. B. Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 1991.

[20] M. Bole and B.-S. Lee. Integrating parametric hull generation into early stage design. *Ship Technology Research*, 53, 2006.

[21] H. Borouchaki, F. Hecht, and E. S. P. George. Reasonably efficient Delaunay based mesh generator in 3 dimensions. Technical report, INRIA, 1995.

[22] M. A. Bougharriou. Nerzglättung für Echtzeitanwendungen. Master's thesis, Universität Karlsruhe (TH), 2007.

[23] Brigham Young University. *The Cleave and Fill Tool: An All-Hexahedral Refinement Algorithm for Sweept Meshes.* 9th International Meshing Roundtable, October 2000.

[24] R. Bronsart and G. Knieling. Automated knuckle detection in ship hull forms. *Ship Technology Research*, 54, 2007.

[25] W. A. Cook and W. R. Oakes. Mapping methods for generating three-dimensional meshes. *Comp. Mech. Eng.*, 1983.

[26] M. G. Cox. Cubic spline fitting with convexity and concavity constraints. In *NPL Report NAC 23.* National Physical Laboratory, Teddington, Middlesex, UK, 1973.

[27] C. de Boor and J. R. Rice. Least-squares cubic spline approximation. i: fixed knots. In *Report CSD TR 20.* Purdue University, Lafayette, 1968.

[28] P. Deuflhard and F. Bornemann. *Numerische Mathematik 2.* de Gruyter Lehrbuch, 2002.

[29] P. Deuflhard and A. Hohmann. *Numerische Mathematik 1.* de Gruyter Lehrbuch, 2002.

[30] I. Dittrich. *3D-Netzgenerierung für Fließgewässer.* PhD thesis, RWTH Aachen, 2001.

[31] A. Dudszus and E. Danckwardt. *Schifftechnik, Einführung und Grundbegriffe.* VEB Verlag Technik Berlin, 1982.

[32] J. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics.* Springer Verlag, 1999.

[33] J. Ferziger and M. Peric. *Numerische Strömungsmechanik.* Springer Verlag, 2008.

[34] W. J. Gordon and C. A. Hall. Construction of curvilinear co-ordinate systems and applications to mesh generation. *International Journal for Numerical Methods in Engineering*, 1973.

[35] L. Harzheim. *Strukturoptimierung: Grundlgen und Anwendungen.* Verlag Harri Deutsch, 2007.

[36] A. Helbrich. CFD-Berechnungen mit überlappenden Gittern in Bezug auf den Voith-Schneider-Propeller. Master's thesis, Universität Ulm, 2008.

[37] W. Hoffmann. *Advancing Front Gittergenerierung und apriori Fehlerabschätzungen für elliptische Randwertprobleme mit Singularitäten.* PhD thesis, Universität Stuttgart, 1999.

[38] Institute of Computational Continuum Mechanics GmbH. *COMET.*

[39] D. Jürgens. *Theoretische und experimentelle Untersuchungen instationärer Tragflügelumströmungen und Entwicklung eines Berechnungsverfahrens für Vertikalachsmotoren.* PhD thesis, Rostock, 1994.

[40] S.-J. Kim, D.-Y. Lee, and M.-Y. Yang. Offset Triangular Mesh using the Multiple Normal Vectors of a Vertex. *Computer-Aided Design and Applications*, 2004.

[41] P. Knabner and L. Angermann. *Numerik partieller Differentialgleichungen.* Springer, 2000.

[42] P. M. Knupp. Remarks on Mesh Quality. 45th AIAA Aerospace Sciences Meeting and Exhibit, January 2007.

[43] N. Köckler. Gittergenerierung. Technical report, Universität-GH Paderborn, 2005.

[44] D. Lammert. Direct search algorithm. Technical report, Unversität zu Köln, 2008.

[45] T. Lehner. Digitale Gelände mittels Delaunay-Triangulierung und Abbauplanung in AutoCAD. Master's thesis, Johannes Kepler Universität Linz, 2002.

[46] R. R. Lober, T. J. Tautges, and R. A. Cairncross. The Parallelization of an Advancing-front, All-quadrilateral Meshing Algorithm for Adaptive analysis. Technical report, Sandia National Laboratories, 1995.

[47] J. C. Matutat. *Numerical Modelling, Simulation and Optimization of Ship Hull Geometries.* PhD thesis, Universität Ulm, 2010.

[48] R. J. Meyers, T. J. Tautges, and D. P. M. Tuchinsky. The "Hex-Tet" Hex-Dominant Meshing Algorithm as Implemented in CUBIT. Technical report, Sandia National Laboratories and Ford Research Laboratory, 1998.

[49] S. V. Patankar. Numerical heat transfer and Fluid Flow. Technical report, Hemisphere Publishing Corporation, 1980.

[50] M. Peric. *A Finite Volume Methode for the prediction of three-dimesional Fluid Flow in complex ducts.* PhD thesis, Imperial College, University of London, 1985.

[51] L. Piegel and W. Tiller. *The NURBS book.* Springer, 1997.

[52] R. Ranmacher. *Numerik von Problemen der Kontinuumsmechanik.* Universität Heidelberg, November 2006.

[53] J.-F. Remacle and M. S. Shepard. An algorithm oriented mesh database. *International Journal for Numerical Methods in Engineering*, 58:349–374, 2003.

[54] D. R.White and P. Kinney. Redesign of the paving algorithm: Robustness enhancements through element by element meshing. *6th International Meshing Roundtable*, pages 323–335, October 1997.

[55] Sandia National Laboratories. *Automatic Hexahedral Mesh Generation by Recursive Convex and Swept Volume Decomposition*. 6th International Meshing Roundtable, October 1997.

[56] Sandia National Laboratories. *Hexahedral Mesh Generation using Multi-Axis Cooper Algorithm*. 9th International Meshing Roundtable, October 2000.

[57] Sandia National Laboratories. *CUBIT 11.1 User Documentation*, 2008.

[58] J. Schöberl. NETGEN - AnAdvancing Front 2D/3D-Mesh Generator Based on Abstract Rules. *Computing and Visualization in Science*, 1997.

[59] J. R. Shewchuk. An introduction to the conjugate gradient method. Technical report, Carnegie Mellon University Pittsburgh, 1994.

[60] H. Si. *TetGen - A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator*. Weierstrass Institute for Applied Analysis and Stochastics, January 2006.

[61] S. F. Singer. Numerische optimierung der hydromechanischen parameter des voith-schneider-propellers. Master's thesis, Universität Ulm, 2003.

[62] J. Springer. Vergleichende Analyse numerischer Methoden zu Lösung der inkompressiblen Navier-Stokes Gleichungen in OpenFOAM, Comet und Fresco. Master's thesis, Universität Ulm, 2009.

[63] M. L. Staten, S. A. Canann, and S. J. Owen. BMWSweep: Locating Interior Nodes During Sweeping. *Engineering with Computers*, 15(3):212–218, September 1999.

[64] C. J. Stimpson, C. Ernst, P. Knupp, P. Pebay, and D. Thompson. The Verdict Geometric Quality Library. Technical report, Sandia National Laboratories, 2007.

[65] C. Tapp. *Anisotrope Gitter - Generierung und Verfeinerung*. PhD thesis, Friedrich-Alexander-Universität Erlangen Nürnberg, 1999.

[66] J. F. Thompson, B. K. Soni, and N. P. Weatherill. *Handbook of Grid Generation*. CRC Press, 1999.

[67] M. B. und Boris Lohmann. Optimierung mit Genetischen Algorithmen und eine Anwendung zur Modellreduktion. *at - Automatisierungstechnik*, 2004.

[68] K. Urban. Numerik 2. Vorlesungsskript.

[69] D. Wang, O. Hassan, K. Morgan, and N. Weatherill. Enhanced remeshing from stl files with applications to surface grid generation. *Communications in Numerical Methods in Engineering*, 23:227–239, 2007.

[70] W. Wilke. *Segmentierung und Approximation großer Punktwolken.* PhD thesis, Technische Universität Darmstadt, 2002.

[71] B. Wölk. Generierung hybrider Gitter für die Strömungssimulation auf komplexen anatomischen Geometrien. Master's thesis, Technische Universität Berlin, 2009.

Erklärung

Ich versichere hiermit, daß ich die Arbeit selbständig angefertigt habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe.

Ulm, den _____

(Unterschrift)