Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Deutsches Zentrum
für Luft- und Raumfahrt
German Aerospace Center

Fachbereich Elektrotechnik, Maschinenbau
und Technikjournalismus (EMT)
Studiengang Maschinenbau - Produktentwicklung (B. Eng.)

Bachelor-Thesis

Parametrische CFD-Netzgenerierung im Flugzeugvorentwurf

Vorgelegt von:
Paul Putin
Bambergstraße 7
53721 Siegburg
Tel. +49 (0) 1590 / 13 74 663
paul.putin@smail.emt.h-brs.de
Matr.-Nr. 9023970

Erstgutachter: Dr. Jan Kleinert
Zweitgutachter: Prof. Dr. Olaf Bruch

Sankt Augustin, den 25. Januar 2019

Erklärung zur Bachelor-Thesis

„Ich versichere hiermit, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.
Mir ist bewusst, dass sich die Hochschule vorbehält, meine Arbeit auf plagiierte Inhalte hin zu überprüfen und dass das Auffinden von plagiierten Inhalten zur Nichtigkeit der Arbeit, zur Aberkennung des Abschlusses und zur Exmatrikulation führen können.“

Ort, Datum

Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BRep	Boundary Representation
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CFD	Computional Fluid Dynamics
CGNS	CFD General Notation System
CPACS	Common Parametric Aircraft Configuration Schema
DLR	Deutsches Zentrum für Luft- und Raumfahrt
DNS	Direct Numerical Simulation
FLTK	Fast Light Tool Kit
Flucs	FLexible Unstructured CFD Software
FVM	Finite Volume Method
GUI	Graphical User Interface
IGES	Initial Graphics Exchange Specification
MATLAB	MATrix LABoratory
MDO	Multi-Disciplinary Optimization
NetCDF	Network Common Data Format
ONELAB	Open Numerical Engineering LABoratory
OpenFOAM	Open Field Operation And Manipulation
PDE	Partial Differential Equation
RANS	Reynolds-Averaged Navier-Stokes
STEP	Standard for the Exchange of Product Model Data
STL	Standard Triangulation Language
TAU	Triangular Adaptive Upwind
TiGL	TiGL Geometry Library
UNV	Universal File Format

Symbolverzeichnis

Symbol	Beschreibung	Einheit
A_i	Flächeninhalt einer Tetraederseite	m^2
A_{tri}	Flächeninhalt eines Dreiecks	m^2
$\hat{a}$	Innerer Winkel eines Dreiecks	°
$\hat{b}$	Innerer Winkel eines Dreiecks	°
$\hat{c}$	Innerer Winkel eines Dreiecks	°
$DistMax$	Parameter: Entfernung Oberfläche − $LcMax$-Elemente	m
$DistMin$	Parameter: Mindesthöhe der $LcMin$-Element-Schicht	m
Div_{lin}	Parameter: Divisor für Feinheitsgradjustierung	-
e	Spezifische innere Energie	J/kg
E_{tet}	Qualitätsmaß für Tetraeder bei NETGEN	-
E_{tri}	Qualitätsmaß für Dreiecke bei NETGEN	-
γ_{tet}	Radius Ratio für Tetraeder	-
γ_{tri}	Radius Ratio für Dreiecke	-
$\vec{g}$	Gravitationsvektor im 3D Raum	m/s^2
h	Spezifische Enthalpie	J/kg
I	Einheitsmatrix	-
k	Vom Benutzer eingestellte Kantenlänge eines Elements	m
KV	Kontrollvolumen in der FVM	m^3
λ	Wärmeübergangskoeffizient	W/(m K)
l_c	Parameter: Charakteristikum Element-Zielkantenlänge	-
$LcMax$	Parameter: obere Grenze für l_c	-
$LcMin$	Parameter: untere Grenze für l_c	-
l_i	Kantenlängen eines Elements	m
$Lmax_{xyz}$	Parameter: $max(l_x, l_y, l_z)$	m
l_{min}	Kürzeste Kante eines Elements	m
l_x	Parameter: max. Flugzeugausdehnung in x	m
l_y	Parameter: max. Flugzeugausdehnung in y	m
l_z	Parameter: max. Flugzeugausdehnung in z	m
$\vec{\nabla}$	Divergenz im 3D Raum	-
p	Druck	N/m^2
ϕ	Erhaltungsgröße (Masse bzw. Impuls bzw. Energie)	kg bzw. N s bzw. J
Ψ	Fluss der Erhaltungsgröße ϕ	ϕ/t

Symbol	Beschreibung	Einheit
P_i	Zentrum einer Voronoï Region	-
P_j	Zentrum einer Voronoï Region	-
$\dot{q}_S$	Wärmestrahlung	W/kg
Q_{tet}	Verhältnis von $max(l_i)$ zu $r_{Inkugel}$ für Tetraeder	-
ρ	Dichte	kg/m^3
ρ_{tet}	Radius Edge Ratio für Tetraeder	-
$r_{Inkreis}$	Radius des Dreieck-Inkreises	m
$r_{Inkugel}$	Radius der Tetraeder-Inkugel	m
$r_{Umkreis}$	Radius des Dreieck-Umkreises	m
$r_{Umkugel}$	Radius der Tetraeder-Umkugel	m
T	Temperatur	K
t	Zeit	s
τ	Spannungsmatrix im 3D Raum	N/m^2
θ_{ij}	Innere Winkel eines Tetraeders (Diederwinkel)	$^\circ$
$\vec{u}$	Geschwindigkeitsvektor im 3D Raum	m/s
V_i	Voronoï Region	-
V	Volumen	m^3
V_{tet}	Volumen eines Tetraeders	m^3

1 Einleitung

> I hate meshes. I cannot believe how hard this is. Geometry is hard.
>
> David Baraff
> Senior Research Scientist
> Pixar Animation Studios

Die Vernetzung von Geometrien ist eine lohnenswerte Herausforderung, ob es sich dabei um Clownfische und sprechende Automobile für die Kinoleinwand, um Blutgefäße und Gehirne für die Medizin oder um Flugzeuge für die Luftfahrt handelt. Die Begründung liegt im Ergebnis, das schön anzusehen ist, sogar unterhaltsam sein kann, andererseits aber Leben rettet oder technischen Fortschritt und Effizienz fördert. Letzteres inspiriert diese Arbeit.

In dieser Arbeit geht es grundsätzlich um den Machbarkeitsnachweis von parametrischer, automatisierbarer Netzgenerierung für Strömungssimulationen. Das Ziel ist die Erweiterung der bereits vorhandenen, parametrisierten Geometriebibliothek TiGL um einen solchen Netzgenerator. Mit der Erweiterung um die Vernetzung eröffnet sich das Potential auf künftige, automatisierbare Aerodynamikberechnung von Flugzeugvorentwürfen in einer Optimierungsumgebung.

Warum und wofür die Vernetzung im vorliegenden Fall wichtig ist, auf welcher Geometriegrundlage sie aufbaut und welche Anforderungen im Raum stehen, darüber gibt diese Einleitung Aufschluss. Ebenso findet die Beschreibung der Vorgehensweise dieser Arbeit hier statt.

1.1 Motivation

Die Motivation für die Vernetzung von Geometrien gründet sich auf dem Bestreben nach Verbesserung der kritischen Eigenschaften eines Flugzeugs. Diese Charakteristika beziehen sich bspw. auf den Auftrieb, den Luftwiderstand, den Kraftstoffverbrauch, das Gewicht, die Steifigkeit, die Manövrierfähigkeit, die Schadstoff- und Lärmemissionen, den Nutzwert, etc. Diese Merkmale stammen aus verschiedenen, teilweise widersprüchlichen Bereichen. Der gemeinsame Einflussfaktor ist jedoch stets die geometrische Formgebung. Auf der Suche nach höherer Effizienz kann die Flugzeuggeometrie iterativ variiert werden. Für die Korrelation zwischen Formgebung und Aerodynamik bieten sich Windkanaltests oder Strömungssimulationen an. Im Sinne der Entwicklungskosten werden reale Tests nur für ausgereifte Konzepte durchgeführt. In der Findungsphase basieren Designentscheidungen zunächst auf Simulationsergebnissen.

Die Betrachtung einer kritischen Eigenschaft (in diesem Fall der Aerodynamik) kann nicht losgelöst von den anderen Attributen geschehen. In diesem Zusammenhang ist beim Flugzeugvorentwurf die sog. „Multi-Disciplinary Optimization“ (MDO) im Einsatz. Verschiedene Ingenieurdisziplinen (z. B. Thermodynamik, Aerodynamik, Aeroakustik, Elastostatik, etc.) sind in diesem Prozess aufeinander angewiesen. Insbesondere aufgrund der gegenseitigen Einflussnahmen der Disziplinen und der teilweisen Gegenläufigkeit der oben genannten Flugzeugcharakteristika kann keine getrennte Optimierung stattfinden.

Ein Beispiel soll grob die Vorgehensweise der MDO verdeutlichen:
Eine Iteration beginnt mit der automatisierten Generierung einer Flugzeuggeometrie, bspw. durch TiGL. Anschließend erfolgt die Bestimmung der gesuchten Größen für jede in die Optimierung involvierte Disziplin. Dies beinhaltet u. a. auch die Simulation der aerodynamischen Eigenschaften der Flugzeuggeometrie. Damit geht die Notwendigkeit von vernetzten Flugzeuggeometrien für die numerische Diskretisierung einher. An diesem Punkt befindet sich die potentielle Anwendung des Netzgenerators, der in dieser Arbeit vorgestellt wird. Durch Erzeugung von Geometrie und CFD Netz in *einer* Software ergibt sich der Vorteil, dass die ohnehin komplexe MDO eine zu realisierende Softwareschnittstelle weniger hat. Mit der errechneten aerodynamischen Druckverteilung kann eine weitere, bspw. strukturmechanische Simulation gestartet werden. Gleichzeitig könnte parallel eine weitere Disziplin mit den Ergebnissen der Strömungssimulation als Eingangsgröße abgearbeitet werden. So wird verfahren, bis alle gesuchten Größen für diese Iteration vorliegen. Anhand der Ergebnisse leitet der Optimierer neue Parameter für die Geometriegenerierung ab. Eine weitere Schleife mit der Vernetzung der neuen Geometrie als Teilaspekt kann beginnen.
Vor Beginn der MDO muss der Ingenieur das Optimierungsproblem inkl. der zu optimierenden Größen formulieren. Zusätzlich benötigen alle involvierten Variablen sinnvolle Ober- und Untergrenzen. Für die Lösung der Problematik gibt es gradientenbasierte (lokale) und gradientenfreie (globale) Optimierungstechniken, die wiederum in vielfältigen Optimierungsarchitekturen eingebettet sein können. [24, 28]

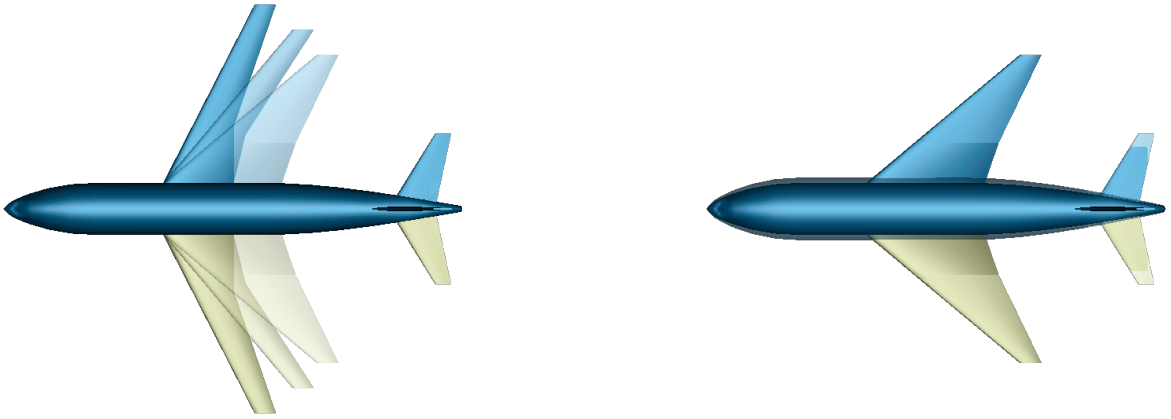
1.2 Anforderungen im Flugzeugvorentwurf mit MDO

Jede Iteration des Suchalgorithmus' stellt eine komplette Simulation und Berechnung aller gesuchten Variablen dar. Umso wichtiger ist in der Fluidsimulation eine adäquate Vernetzung der Geometrie, die das Ergebnis nicht numerisch verfälscht, aber trotzdem eine hohe Zeiteffizienz fördert. Ein Netzgenerator kann die Laufzeit der MDO auf dreifache Weise beeinflussen. Zunächst benötigt die Netzgenerierung eine gewisse Dauer. Die erreichte Elementanzahl und Netzqualität beeinflusst den numerischen Aufwand und somit ebenfalls die CFD Simulationszeit. Damit eine Optimierung grundsätzlich zu einem Ergebnis kommen kann, ist eine hohe Robustheit aller beteiligten Komponenten unabdingbar. Die Netzerzeugung für die Strömungssimulation spielt für die Robustheit des CFD Lösers eine entscheidende Rolle. Die Variation der Geometrie nach jeder Iteration erfordert auf der Seite der Gittererzeugung eine robuste Lösung. Für die Anwendung des Netzes in einer automatisierten Optimierungsumgebung ist entsprechend eine parametrische Vernetzung unerlässlich.

Die Relevanz der Zeitkomponente spiegelt sich auch darin wieder, dass die Optimierungen in die Auflösungsstufen „low-fidelity“, „mid-fidelity“ und „high-fidelity“ eingeteilt werden [28]. Diese Einteilungen richten sich mit fließenden Grenzen nach dem zeitlichen Aufwand der Simulationsmethoden in einer MDO. Der Aufwand resultiert aus der Netzfeinheit in einer Simulation (FEM, CFD, etc.). Unter diesen drei Kategorien haben „low-fidelity“-Optimierungen insbesondere bei Konzeptstudien die größte Verbreitung. Doch auch „mid-fidelity“-Optimierungen sind häufig in Verwendung. Im Aerodynamikanteil der Optimierung mittlerer Feinheit werden die Euler Gleichungen berechnet oder grobe RANS Simulationen (siehe Kapitel 2.1.2) durchgeführt. [28]
In dieser Arbeit wird den Anforderungen des Flugzeugentwurfs entsprechend eine „mid-fidelity“-Vernetzung für die zugehörigen CFD Methoden angestrebt.

1.3 TiGL als Ausgangspunkt

Diese Arbeit bewegt sich zwischen zwei Schnittstellen. Die Netzgenerierung baut auf der Geometrie aus der TiGL Bibliothek auf und knüpft an eine Berechnungssoftware für Fluidsimulation an. Geometrieerzeugung, Vernetzung und Berechnung sind in eine Optimierungsumgebung für den Flugzeugvorentwurf eingebettet. An dieser Stelle soll TiGL als Ausgangspunkt kurz vorgestellt werden.
TiGL ist eine Open Source Bibliothek zur Geometriemodellierung von Flugzeugen und Hubschraubern, wobei es CPACS Dateien (Common Parametric Aircraft Configuration Schema) interpretiert, die u. a. auch parametrische Definitionen der entsprechenden Geometrie enthalten. Aus CPACS Informationen können mit Hilfe von TiGL 3D Geometrien in alle gängigen CAD Formate exportiert werden. Mit Hilfe von Open CASCADE[1] und proprietären Algorithmen zur Oberflächendarstellung steht in TiGL ein breites Spektrum von CAD Funktionalitäten zur Verfügung. TiGL wird hauptsächlich vom DLR entwickelt und fokussiert MDO im Flugzeugvorentwurf als Anwendungsgebiet. Der Einsatz in Optimierungsschleifen wird durch die Parametrisierung erleichtert, weil sich so mit der Änderung weniger Einstellungswerte gänzlich neue Geometrien generieren lassen (siehe Abbildung 1.3-1). Die Parametersteuerung lässt sich in einer vollautomatisierten MDO Umgebung einbinden, um viele verschiedene Konzepte von der Optimierungsschleife testen zu lassen. TiGL befindet sich in permanenter Weiterentwicklung und ein naheliegender nächster Schritt ist die Erweiterung um einen automatisierbaren Gittergenerator aus dem Open Source Bereich. [40]



(a) Variierende Tragflächen einer TiGL Geometrie

(b) Variierende Höhenleitwerke und Rümpfe einer TiGL Geometrie

Abbildung 1.3-1: TiGl für den Optimierungseinsatz [40]: Visualisierung der Möglichkeiten einer automatischen Geometrieerzeugung für eine Optimierungsschleife. Die Geometrievarianten entstehen durch Änderung spezifischer Parameter der CPACS Datei.

[1] Diese Softwarebibliothek ist ein weiteres Open Source Werkzeug für die 3D CAD Modellierung.

1.4 Vorgehensweise und Zielsetzung der Kapitel

Das Ziel dieser Arbeit ist die Entwicklung des „TiGL Meshers“. Dies ist ein Prototyp, der aus einer TiGL Geometrie (bestenfalls automatisiert) ein Netz für eine numerische Strömungssimulation erstellen kann. Dabei liegt der Fokus auf sog. „mid-fidelity“-Gittern, die für die Lösung von RANS oder Euler Gleichungen geeignet sind. Diese Arbeit zielt nicht auf eine fertige, ausgereifte Implementierung eines Netzgenerators ab. Sie umfasst auch nicht die Untersuchung der Aussagekraft von Simulationen auf Gittern aus dem Prototyp. Stattdessen geht es um den Nachweis der grundsätzlichen Machbarkeit einer Vernetzungsfunktion für TiGL, die Potential für den Einsatz in einer Optimierungsschleife hat. Die dafür relevanten Schritte sind wie folgt strukturiert:

Zu Beginn der Arbeit stehen die Navier Stokes Gleichungen im Vordergrund, weil sie die Grundlage für jegliche Strömungssimulation bilden. Mit geeigneten Vereinfachungen und Modellierungen ist der Einsatz dieser Erhaltungsgleichungen auch in der zeitkritischen Umgebung einer Vorentwurf-Optimierungsschleife möglich. Als Werkzeug für die numerische Lösung der vereinfachten Navier Stokes Gleichungen wird im Folgenden die Finite Volumen Methode grob charakterisiert. Deswegen ist anschließend ein kurzer Überblick über existierende Gitterkategorien für Finite Volumen Methoden nützlich. So können auch die verwendeten triangulierten, unstrukturierten Netze eingeordnet werden. Es folgt die Beschreibung von drei grundlegend verschiedenen Ansätzen der triangulierenden Gittergenerierung, die der Benutzer des TiGL Meshers zur Verfügung hat. Dabei handelt es sich um den Advancing Front Algorithmus NETGEN, die Delaunay Algorithmus Familie und den Mesh Adapt Algorithmus. Die Funktionsweise dieser Algorithmen führt den Leser zu ihren Stärken und Schwächen und ermöglicht einen gezielten Einsatz bei der Vernetzung mit dem TiGL Mesher. Bei der Netzerzeugung ist sowohl für interne Berechnungen im Softwareprototyp, als auch für den Benutzer von großer Bedeutung, die Eigenschaften des Gitters bewerten zu können. Hierbei ist der letzte Teil des zweiten Kapitels behilflich. Mit diesen Hintergrundinformationen ist die Verständnisgrundlage für den Einsatzzweck, die Bedienung und die Funktionalitäten des TiGL Meshers gelegt.

Das dritte Kapitel beschäftigt sich mit der Auswahl einer Open Source Software, die als Grundlage für den TiGL Mesher geeignet ist. Nach der Definition eines Anforderungskataloges folgt eine Vorauswahl und eine Gegenüberstellung von Gemeinsamkeiten und Unterschieden, die zwischen den Favoriten Gmsh und SALOME vorliegen. Die Begründung der Entscheidung für Gmsh bildet den Schluss dieser Einheit.

Nachdem die theoretische Grundlage gelegt, und die Entscheidung für ein Softwareprodukt getroffen ist, kann die praktische Umsetzung in den Mittelpunkt rücken. Die folgende Programmierung eines Prototyps stellt den Kern und das Resultat dieser Arbeit dar. Das Kapitel vier beschreibt zunächst allgemein die Programmierschnittstelle von Gmsh, an die der TiGL Mesher angebunden ist. Anschließend werden die Parameter des TiGL Meshers eingeführt, an denen sich die (automatisierte) Vernetzung orientiert. Der in der Gmsh API programmierte, exemplarische Ablauf der Gittergenerierung dient der Übersicht über die Vorkommen der Parameter. Im Zentrum dieses Kapitels steht die Automatisierung der Parametereinstellungen. Zwar kann der Benutzer die Parameter z. T. auch manuell definieren, doch ist für eine Optimierungsschleife vor allem eine automatische Gittererzeugung relevant. Diese Arbeit bemüht einen Ansatz für eine Vernetzungsautomatik. Mit der hier vorgestellten heuristisch entwickelten Formel soll erreicht werden, dass der TiGL Mesher für verschiedene Skalierungen einer Geometrie stets gleichwertige Netze erzeugt. Damit ist die Anzahl, Verteilung und Dichte der Elemente (Dreiecke und Tetraeder) gemeint. Das Einsetzen der manuell und / oder automatisch justierten Parameter in die Programmierschnittstelle rundet dieses Kapitel

zusammen mit einigen nützlichen Nebenfunktionen ab.

Das fünfte Kapitel zeigt auf anschauliche Weise, wie sich die Einstellung der Parameter auf das resultierende Netz auswirken können. Für die Benutzung des TiGL Meshers können aus diesem Teil wichtige Schlüsse gezogen werden. Dazu gehören bspw. die Steuerung des Zellenwachstums und der Zellenanzahl, aber auch Wechselwirkungen zwischen den vernetzenden Algorithmen und den optionalen Optimieren. Gleichzeitig liefert dieses Kapitel einen Eindruck zur Vernetzungsdauer und der entsprechenden Zellenanzahl bei verschiedenen Feinheitsstufen des Gitters. In den Testläufen zeigt sich, inwieweit der Automatikmodus (basierend auf der heuristischen Formel) bereits unabhängig von der Skalierung einer Geometrie gleichwertige Netze erzeugen kann. Im Sinne der Machbarkeitsstudie erfolgt auch eine Demonstration des Workflows von TiGL über den TiGL Mesher hin zu einer Strömungssimulation mit OpenFOAM.

An diesem Punkt angelangt, kann das Résumé aus dieser Arbeit gezogen werden, um einen Ausblick auf die nächsten Schritte in der Entwicklung des TiGL Meshers zu geben.

Einige allgemeine Bemerkungen zu Konventionen in dieser Arbeit:
Referenzen innerhalb eines Satzes beziehen sich auf den betreffenden Satz. Referenzen außerhalb eines Satzes, bzw. am Ende eines Abschnittes beziehen sich auf den vorherigen Gedankengang bzw. vorherigen Abschnitt. Die zahlreichen Angaben zur Anzahl von Elementen in einem Netz sind immer auf Tausend gerundet.

2 Vernetzung für Finite-Volumen-Methoden

In diesem Kapitel werden die Grundlagen für den späteren Verlauf der Arbeit gelegt. Bei der Vernetzung laufen die Fäden aus den physikalischen Vorgängen, den numerischen Berechnungen und der Geometrie zusammen. Die Anforderungen an die Vernetzung stammen aus der Strömungssimulation und der darauf aufbauenden Optimierungsschleife. Vor einer Klassifizierung von Gittern und der Beschreibung relevanter Vernetzungsalgorithmen werden die Navier-Stokes-Gleichungen und ihre numerische Lösung mit der Finite-Volumen-Methode kurz umrissen. Zum Schluss dieses Kapitels werden verschiedene Möglichkeiten vorgestellt, Netze zu evaluieren.

2.1 Navier-Stokes-Gleichungen

In der Strömungsmechanik wird die Erhaltung der physikalischen Größen in einem Fluid (Masse, Impuls, Energie) mit den Navier-Stokes-Gleichungen mathematisch beschrieben:

- Massenerhaltung
- Impulserhaltung in x-Richtung
- Impulserhaltung in y-Richtung
- Impulserhaltung in z-Richtung
- Energieerhaltung

2.1.1 Vollständige Navier-Stokes-Gleichungen

Diese Gleichungen bilden ein gekoppeltes, nichtlineares Gleichungssystem, das analytisch nur für Spezialfälle niedriger Komplexität gelöst werden kann. Es folgt ohne eine Herleitung die Notation in Divergenzform, die unabhängig von einem Koordinatensystem geschrieben wird und gleichzeitig mehr Übersicht bietet, als die Skalarform. Gleichungen 2.1.1-1 bis 2.1.1-4 und die entsprechenden Erklärungen stammen aus Kapitel 2.3 bei [22]. Für die Divergenzschreibweise kommt der Geschwindigkeitsvektor $\vec{u}$, der Gravitationsvektor $\vec{g}$, die Divergenz $\vec{\nabla}$, die Einheitsmatrik I und die Spannungsmatrix τ zur Anwendung:

$$\vec{u} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad \vec{g} = \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} \qquad \vec{\nabla} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} \qquad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \tau = \begin{bmatrix} \tau_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \tau_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \tau_{zz} \end{bmatrix}$$

Nach Einführung dieser Variablen wird die Massenerhaltung zu:

$$\frac{\partial}{\partial t}(\rho) + \vec{\nabla} \cdot (\rho \cdot \vec{u}) = 0 \tag{2.1.1-1}$$

Die drei Impulserhaltungsgleichungen (abgeleitet vom zweiten Newtonschen Gesetz) ergeben sich entsprechend zu:

$$\frac{\partial}{\partial t}(\rho \cdot \vec{u}) + \vec{\nabla} \cdot (\rho \cdot \vec{u} \times \vec{u} + p \cdot I - \tau) = \rho \cdot \vec{g} \tag{2.1.1-2}$$

Die Energieerhaltungsgleichung (abgeleitet vom ersten Hauptsatz der Thermodynamik) lautet:

$$\frac{\partial}{\partial t}\left[\rho \cdot \left(e + \frac{1}{2} \cdot \vec{u}^2\right)\right] + \vec{\nabla} \cdot \left[\rho \cdot \vec{u} \cdot \left(h + \frac{1}{2} \cdot \vec{u}^2\right) - \tau \cdot \vec{u} - \lambda \cdot \vec{\nabla} T\right] = \rho \cdot \vec{g} \cdot \vec{u} + \rho \cdot \dot{q}_S \quad \text{(2.1.1-3)}$$

Zusammengefasst in einem vollständigen Navier-Stokes-Gleichungssystem:

$$\frac{\partial}{\partial t}\begin{bmatrix} \rho \\ \rho \cdot \vec{u} \\ \rho \cdot \left(e + \frac{1}{2} \cdot \vec{u}^2\right) \end{bmatrix} + \vec{\nabla} \cdot \begin{bmatrix} \rho \cdot \vec{u} \\ \rho \cdot \vec{u} \times \vec{u} + p \cdot I - \tau \\ \rho \cdot \vec{u} \cdot \left(h + \frac{1}{2} \cdot \vec{u}^2\right) - \tau \cdot \vec{u} - \lambda \cdot \vec{\nabla} T \end{bmatrix} = \begin{bmatrix} 0 \\ \rho \cdot \vec{g} \\ \rho \cdot \vec{g} \cdot \vec{u} + \rho \cdot \dot{q}_S \end{bmatrix} \quad \text{(2.1.1-4)}$$

Erklärung der übrigenen Variablen:

- $\vec{u}^2 = u^2 + v^2 + w^2$
- e ist die spezifische innere Energie [J/kg]
- h ist die spezifische Enthalpie [J/kg] mit $h = e + p/\rho$
- ρ ist die Dichte des Fluids [kg/m^3]
- $\rho \cdot \vec{u}$ ist der Impuls
- $\rho \cdot \left(e + \frac{1}{2} \cdot \vec{u}^2\right)$ ist die totale Energie
- p ist der Druck [N/m^2]
- $\dot{q}_S$ ist die Wärmestrahlung, die auf das Volumen wirkt [W/kg]
- λ ist der Wärmeleitkoeffizient [W/(m K)]
- T ist die Temperatur [K]

Insgesamt sind im Gleichungssystem 17 gesuchte Variablen. Aus diesem Grund werden 12 weitere Gleichungen benötigt, um zu einer Lösung zu kommen. In den zusätzlichen Gleichungen sind drei Zustandsgleichungen für das Fluid und neun Stokessche Beziehungen für die Normal- und Schubspannungen vertreten. [22]

2.1.2 Variationen der Navier-Stokes-Gleichungen

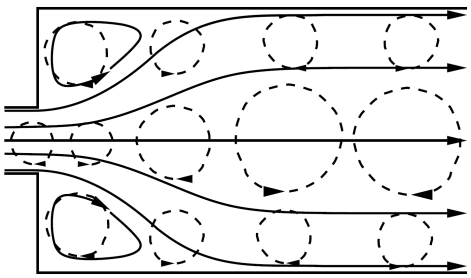


Abbildung 2.1.2-1: RANS-Simulation [34]: Durchgezogen: Mittelung; Gestrichelt: nicht aufgelöste hochfrequente Turbulenzen

Eine Simulation erfordert nicht in jedem Anwendungsfall die Berechnung aller oben gelisteter Größen. Die Komplexität des Gleichungssystems kann selbst im dreidimensionalen Raum stark dezimiert werden, wenn bei der Modellierung physikalische Vereinfachungen angenommen werden können, bspw. eine konstante Temperatur oder eine viskositätsfreie Strömung. Ebenso wirkt sich eine Vernachlässigung der Kompressibilität des Fluides positiv auf den Simulationsaufwand aus. Sofern nicht *jede* Turbulenz aufgelöst werden braucht wie bei der *Direct Numerical Simulation* (DNS), bietet sich eine Vergröberung des Netzes und eine Einführung von Turbulenzmodellen an. Diese approximieren bzw. mitteln das Verhalten und die Auswirkung hochfrequenter Turbulenzen mit einer Modellierung

(*Reynolds-Averaged Navier Stokes* (RANS) Simulation siehe Abbildung 2.1.2-1). Für die Berechnung der Turbulenzen im mittleren und niederen Frequenzbereich ist dann eine signifikant niedrigere Netzfeinheit bereits ausreichend. Für hauptsächlich laminare Strömungen können darüber hinaus auf nochmals vergröberten Simulationsnetzen Reibung und Ablösungen / Turbulenzen vernachlässigt werden (*Euler* Gleichungen), wobei die Aussagekraft über den Auftrieb eines Flugzeugs erhalten bleibt [17]. Eine Auswahl möglicher Simplifikationen und daraus resultierender Gleichungen ist in Abbildung 2.1.2-2 dargestellt. [22, 34]

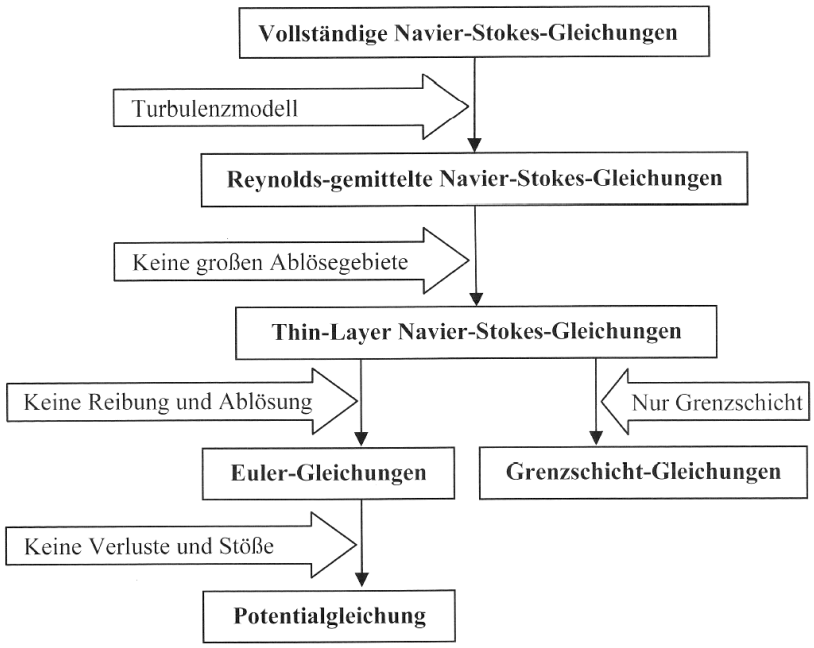


Abbildung 2.1.2-2: Mögliche Vereinfachungen der Navier-Stokes-Gleichungen [22]

2.2 Finite-Volumen-Methode für Strömungssimulationen

Im Bereich der numerischen Strömungsmechanik ist die Finite-Volumen-Methode am häufigsten zur Lösung von PDEs verbreitet [34]. Anders, als bei der Finite-Differenzen- oder Elemente-Methode werden hier Flüsse zwischen einzelnen Zellen des Rechengitters auf den Zellwänden explizit bilanziert. Daraus ergeben sich praktische Vorteile bei der CFD Simulation. Die Erhaltungsgrößen der Strömungsmechanik, Masse, Impuls und Energie, werden aufgrund der lokalen und globalen Bilanzen dieser Größen durch numerische Einflüsse nicht verringert oder vergrößert. Ein weiterer positiver Aspekt ist die Möglichkeit, sowohl strukturierte, als auch unstrukturierte, triangulierte Gitter für die Diskretisierung zu verwenden. [34, 19]
Bei Anwendung der FVM für CFD Simulationen muss eine Diskretisierung des Strömungsgebiets in eine endliche Zahl von nicht überlappenden Kontrollvolumina erfolgen. Jede dieser Zellen bekommt einen charakteristischen Punkt, der in der Zellenmitte angesiedelt ist und Startwerte für die zu erhaltenden Größen enthält. Die anschließende Bilanzierung des Flusses in eine Zelle und aus einer Zelle kann folgendermaßen ausgedrückt werden:

$$\frac{\partial}{\partial t}\phi = \vec{\nabla} \cdot \Psi(\phi) \qquad \Leftrightarrow \qquad \int_{KV} \frac{\partial}{\partial t}\phi \, dV \; = \int_{KV} \vec{\nabla} \cdot \Psi(\phi) \, dV \tag{2.2-1}$$

Die Änderung der Erhaltungsgröße ϕ (Masse, Impuls oder Energie) über die Zeit ist gleich der Divergenz des Flusses $\Psi(\phi)$ der Erhaltungsgröße. Die Erhaltungsgleichung (siehe auch Kapitel 2.1) wird über die einzelnen Volumina KV integriert. Mit dem Gauß'schen Integralsatz wird das Volumenintegral in einzelne Flächenintegrale vereinfacht. Vor der Lösung der

Flächenintegrale erfolgt eine Extrapolation des ϕ aus dem Zellmittelpunkt auf die Randflächen von *KV*. Hierbei wird in den meisten Fällen zwischen den charakteristischen Punkten benachbarter Zellen interpoliert. Mit diesen diskretisierten Erhaltungsgleichungen kann in Kombination mit geeigneten Randbedinungen ein algebraisches System mit je einer Gleichung pro Kontrollvolumen entstehen. Die Lösung des Gleichungssystems wird iterativ für alle Erhaltungsgrößen angenähert. Die jeweilige Erhaltungsgröße ϕ wird bei jeder Iteration als Ergebnis einer Gleichung wieder im Zellmittelpunkt gespeichert. [18, 19, 34]

2.3 Kategorisierung von Netzen für Finite-Volumen-Methoden

Für die Aufteilung des Strömungsgebiets in eine endliche Anzahl von Kontrollvolumina existieren verschiedene Ansätze, die zugrunde liegende Geometrie zu vernetzen:

- Strukturierte Gitter
- Blockstrukturierte Gitter
- Hybride Gitter
- Unstrukturierte Gitter

Strukturierte Netze bestehen im einfachsten Fall aus Quadraten oder Würfeln. Doch auch wenn die Elemente verzerrt sind und ihre Innenwinkel von 90° abweichen (siehe Abbildung 2.3-1), besteht weiterhin eine wichtige Gemeinsamkeit: Die Topologie des Netzes kann über die Indizes der Elemente immer in eine zwei- oder dreidimensionale Matrix ohne leere Einträge übertragen werden. Auf diese Weise sind die Nachbarschaftsbeziehungen für jede Zelle bekannt und eindeutig. Auch besitzt jede Zelle im Inneren des Netzes die gleich Anzahl von Nachbarzellen. Geometrien, die zwar gekrümmte Flächen, jedoch insgesamt eine geringe Komplexität aufweisen, können mit sogenannten O-Gittern (für stumpfe Körper), C-Gittern (für halbschlanke Körper) oder H-Gittern (bspw. für Innendurchströmung einer Düse) beschrieben werden.

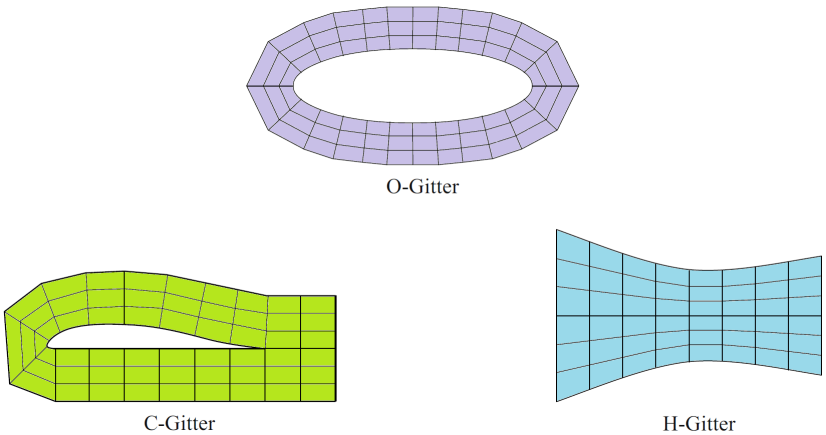


Abbildung 2.3-1: Strukturierte, schiefwinklige Rechennetze [34]

Eine Abschwächung dieser Form der Gitter stellen blockstrukturierte Netze dar (siehe Abbildung 2.3-2). Hier ist die Gittertopologie global unstrukturiert, jedoch sind *alle* lokalen Blöcke / Unterteilungen für sich genommen strukturiert. Hybride Netze sind ebenfalls in Blöcke eingeteilt, jedoch mit dem Unterschied, dass nur *manche* von ihnen noch eine strukturierte Topologie aufweisen. Mit hybriden Netzen wird der Vernetzungsaufwand minimiert, wo das Gitter für die CFD Simulation und zu modellierende physikalische Vorgänge eine untergeordnete Rolle spielen.

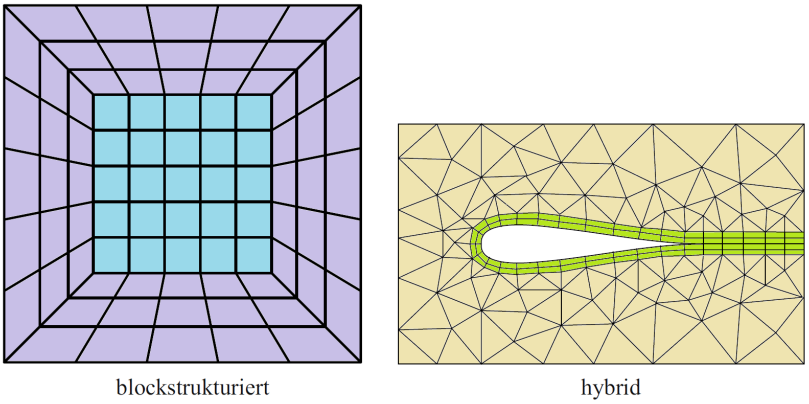


Abbildung 2.3-2: Gemischte Gitter [34]: Blockstrukturiertes Netz (links) zusammengesetzt aus einem O- und H-Gitter; Hybrides Netz (rechts) zusammengesetzt aus einem C-Gitter und einem triangulierten, unstrukturierten Gitter.

Die größte Flexibilität für komplexe Geometrien und lokale Verfeinerungen bieten unstrukturierte Gitter, die ein Volumen in Tetraeder, Hexaeder, Prismen oder andere Polyeder aufteilen. Häufig werden mehrere dieser geometrischen Körper in einem Netz kombiniert. Die zahlreichen Einsatzmöglichkeiten ziehen jedoch im Vergleich zu strukturierten Gittern einen erhöhten Speicherbedarf und Rechenaufwand in der Strömungssimulation mit sich. Trotzdem ist dieser Ansatz für automatische Netzgenerierung arbiträrer Geometrien weit verbreitet und findet auch in dieser Arbeit Anwendung. [34]

2.4 Algorithmen für triangulierte Netze

Im Open Source Segment exitieren zahlreiche Programmcodes, mit denen unstrukturierte, triangulierte Gitter generiert werden können, um diese anschließend für Finite Elemente und Finite Volumen Methoden zu nutzen. An dieser Stelle soll darum ein Überblick über alle relevanten Vernetzungsmethoden entstehen. Dieser umfasst einerseits die Advancing / Moving Front Methode und die Delaunay Vernetzung andererseits. Zusätzlich wird aufgrund des Bezuges zu Gmsh der proprietäre Mesh Adapt Algorithmus behandelt. Der Schwerpunkt liegt im Folgenden auf dem NETGEN Algorithmus, der an diese Stelle exemplarisch die Advancing / Moving Front Algorithmen vertritt. Er ist u. a. auch wichtiger Bestandteil des Vergleichs in Kapitel 3.4. Die Familie der Delaunay Algorithmen wird weniger detailliert beleuchtet. Durch die überaus weite Verbreitung dieses Ansatzes in der Literatur [27] und den entsprechenden Softwarecodes können die einzelnen Vertreter in ihren Einzelheiten an dieser Stelle nicht dargestellt werden. Stattdessen bietet sich eine Übersicht über die drei grundlegenden Ansätze bei Delaunay Algorithmen an.

2.4.1 Advancing Front Algorithmen

Der NETGEN Algorithmus basiert auf der der sog. Advancing / Moving Front Methode. Die ursprüngliche Variante von J. Alan George (1971) [12] fand zunächst keinen großen Anklang, wurde aber später wiederentdeckt und hat über die Jahre in zahlreichen Abwandlungen eine weite Verbreitung gefunden [41].
Unabhängig davon, ob eine 2D oder 3D Variante vorliegt, werden die Elemente nach gewissen Regeln der Reihe nach erzeugt. Damit wird immer an den begrenzenden Außenlinien / Außenflächen einer Geometrie begonnen. Das Stichwort „Advancing“ steht für ein fortschreitendes Vernetzen von den Rändern zur Mitte hin. Die sogenannte „Front“ ist die aktuelle Grenze zwischen vernetzten und unvernetzten Bereichen. Hier prüft der Algorithmus die Anwendbarkeit

von Regeln und fügt ggf. neue Elemente hinzu. Mit fortschreitendem Verlauf verkürzt sich die „Front“, bis sie sich selbst in der Mitte trifft. Charakteristisch für diese Verfahrensvorschrift ist eine sehr hohe Elementqualität an den besagten Rändern, den Startorten der Vernetzung. Dieses Vorgehen hat einen immanenten Nachteil in den Regionen, in denen die „Front“ sich selbst trifft (siehe Abbildung 5.1-1a). Für die Anwendung von Regeln wird ein gewisses Maß an freier Fläche benötigt, die noch nicht vernetzt ist. Deswegen tauchen verschlechterte Elementqualitäten dort auf, wo die Fronten sich begegnen. Insbesondere im 3D Bereich gibt es in der Literatur keine nachweislich robusten Algorithmen, die unter jeglichen Umständen bzw. aus *jeder* Geometrie ein Netz mit gleichmäßigen Elementen erzeugen können. Deswegen wird im weiteren Verlauf die u. U. notwendige Optimierung dieses Advancing Front Gitters behandelt. Trotz allem findet diese Methode im CFD Bereich weite Verbreitung, weil dort der Fokus auf einer adäquaten Vernetzung der Ränder einer Geometrie liegt. Eine erfolgreiche Modellierung der viskosen Randschichten ist hierbei gewährleistet durch die Möglichkeit, anisotrope Zellen von hoher Güte mit dem Anvancing Front Algorithmus zu erzeugen. [36]

Die allgemeine Vorschrift dieses Algorithmus' für ein 2D Beispiel [41] lautet:

- Gegeben: eine rechteckige 2D Fläche mit einem mittigen kreisförmigen Ausschnitt.
- Die äußeren vier Linien und der innere Kreis werden mit Punkten versetzt.
- Die Punkte auf den Außenkanten werden mit Linien verbunden, eine Front entsteht.
- Die Punkte auf dem Kreis werden mit Linien verbunden, eine zweite Front entsteht.
- Vernetzung: Auf einer der Fronten werden zwei beliebige, jedoch benachbarte Punkte zu einer Strecke $\overline{AB}$ verbunden. Rechtwinklig zu $\overline{AB}$ wird ein dritter Punkt C eingeführt. Der Abstand von C zu $\overline{AB}$ beruht auf einem Parameter (Höhe h), einzugeben durch den Benutzer. $\triangle ABC$ bilden ein Element.
- Anschließend wird wieder aus zwei benachbarten Punkten eine Strecke $\overline{AB}$ gebildet und wie zuvor ein potentieller Punkt C bestimmt, jedoch unter folgender Bedingung: In einem vom Benutzer festgelegten Radius um C liegt *kein* weiterer Punkt D. Existiert in diesem Radius ein Punkt D, heißt das neue Dreieck nicht $\triangle ABC$, sondern $\triangle ABD$.
- Dieser Prozess wird wiederholt, bis die Fronten sich allerorts selbst treffen.

2.4.1.1 NETGEN Implementierung

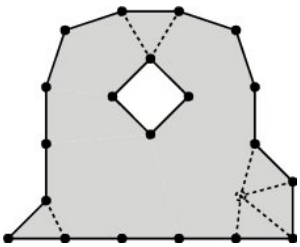


Abbildung 2.4.1.1-1: Vorgehensweise NETGEN Algorithmus [33]

NETGEN Algorithmus [33]

Load boundary mesh (starting front)
Initialize quality classes
While front is not empty
    Choose base-element from front
    Get environment of base-element
    Transform to local coordinate system
    Test for applicable rules
    If a rule is applicable
        Generate new points in local coordinates
        Transform new points to global coordinates
        Store new inner elements
        Update front
    Else
        Decrease quality class for base-element

Bei Gmsh und SALOME findet der NETGEN Algorithmus für Volumenvernetzung Anwendung. Die Kernbegriffe dieser spezifischen Implementierung / Variante des Frontal Algorithmus':

- **„front“** ist ein Vektor aus Punktkoordinaten der Ränder der (noch) zu vernetzenden Geometrie. So lange der Vektor Einträge enthält, fährt der Algorithmus fort.
- **„quality class“** ist eine Zahl $\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$, die die Qualität eines Elements beschreibt, wobei 1 für die höchste Qualität steht und beim Initialisieren allen Randelementen zugewiesen wird. Die „quality class“ wird aber auch herangezogen, um die Toleranz für die Anwendbarkeit einer Regel zu definieren.
- **„base-element“** sind zwei Punkte, die eine Linie (2D) oder drei Punkte, die ein Dreieck (3D) auf der „front“ bilden. Von diesem Element aus werden neue Elemente gebildet.
- **„choose“** steht für die geschickte Wahl beim nächsten zu betrachtenden „base-element“. Die Kriterien hierfür stellen sicher, dass die „front“ gleichmäßig aus allen Richtungen zur Mitte hin wächst und gleichzeitig zunächst die Elemente der „front“ behandelt werden, die am besten zu den „rules“ passen (siehe Abbildung 2.4.1.1-1).
- **„local coordinates“** sind hilfreich, um die geometrischen Operation an einem bestimmten Ort durchzuführen und hierfür einen allgemeingültigen Ablauf in einem einheitlichen Koordinatensystem zu implementieren. Mit der Koordinatentransformation kann der Algorithmus Ebenen im $\mathbb{R}^2$ und gekrümmte Flächen im $\mathbb{R}^3$ mit den gleichen „rules“ behandeln. Die „local coordinates“ ermöglichen auch lokal gewünschte, anisotrope Elemente. Darüber hinaus wird mit diesem Hilfsmittel auch das „graded mesh“ realisiert, das entsprechend der Benutzereingabe mit einem definierten Faktor graduell feiner wird.
- **„rules“** sind geometrische Regeln, nach denen NETGEN vorgeht. Das Vorkommen eines bestimmten geometrischen Musters / Problems wird immer gleich behandelt. Ein Beispiel (siehe Abbildung 2.4.1.1-2): Zwei Linien, beschrieben durch drei Punkte an der Front, stehen in einem 120° Winkel zueinander. Erkennt der Algorithmus so eine Situation, generiert er einen neuen, vierten Punkt, der mit den drei vorhandenen Punkten zwei gleichseitige Dreiecke bildet. In der Praxis werden solche exakte Winkel und Längen nicht angetroffen. Auch der verfügbare Raum für das Setzen des vierten Punktes kann begrenzt sein. Es reicht allerdings schon eine hinreichende Ähnlichkeit (definiert durch die „quality class“) mit dem Idealfall, damit nach dieser Regel zwei annähernd kongruente Dreiecke entstehen. Die „rules“ definieren zusätzlich die notwendigen Aktualisierungen im Vektor der „advancing front“, nachdem neue Elemente entstanden sind.
- **„decrease“** ist das Herabsetzen der „quality class“ und führt dazu, dass die Toleranz der „rules“ für dieses Element steigt und in einem späteren Anlauf eine weniger passende Regel trotzdem angewendet werden kann. Es kann aber auch vorkommen, dass die Anwendung einer Regel auf ein benachbartes Element dazu führt, dass das erstere aufgrund seiner niedrigen „quality class“ gänzlich aufgelöst wird.

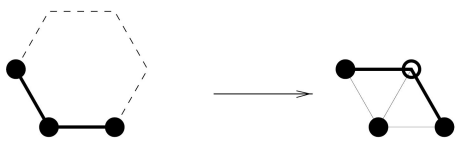


Abbildung 2.4.1.1-2: Eine von neun NETGEN 2D Regeln [33]: Muster, nach dem neue Punkte entstehen. Gestrichelt ist der noch unvernetzte Raum angedeutet.

Das Alleinstellungsmerkmal des NETGEN Algorithmus' im Vergleich zu anderen Advancing Front Algorithmen besteht in der Implementierung der geometrischen Regeln. Dieser Ansatz trennt die Definition der Regeln vom Programmcode, der diese Regeln anwendet. Die wiederkehrend zu überprüfenden Regeln sind in bestimmten, separaten Datenstrukturen abgelegt. Auf diese Weise bleibt die Komplexität des Programmcodes unabhängig von der Regelanzahl.

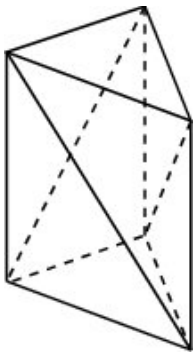


Abbildung 2.4.1.1-3: Pentaeder als Problemfall für NETGEN [33]: „boundaries“ können nicht robust in Tetraeder aufgeteilt werden. Im Inneren ist *ein* neuer Punkt notwendig, der zu acht „base-elements“ passt.

Die asymptotische Komplexität wird ausschließlich vom geometrischen Suchprozesses zur Findung passender Regeln bestimmt. Eine Quadtree-basierte (2D) bzw. eine Octree-basierte (3D) Strukturierung der Daten / Punkte reduziert die Komplexität der Suche auf ein logarithmisches Verhalten. Die Terminiertheit des Algorithmus' ist im 2D Bereich durch neun verschiedene Regeln für alle denkbaren Fälle gewährleistet. Für die Vernetzung von Volumina gibt es keine grundsätzlich robuste / terminierende Formulierung der NETGEN Regeln. Für einen 3D Anwendungsfall ist ein Deadlock somit nicht ausgeschlossen. Allerdings ist dafür gesorgt, dass so eine Situation von NETGEN erkannt wird, um anschließend auf einen alternativen Algorithmus umzuschalten. Dieser ist in der Lage, einen neuen Punkt (siehe Abbildung 2.4.1.1-3) zu berechnen und einzufügen, der die Regeln für gleichzeitig mehrere „base-elements“ erfüllt. Diese Punkte werden aus gegebenem Grund „star points“ genannt. Mit dieser Maßnahme bekommt NETGEN eine erhöhte Praxistauglichkeit und terminiert laut Urheber so für die „meisten Sonderfälle“. [33]

2.4.1.2 NETGEN Optimierung

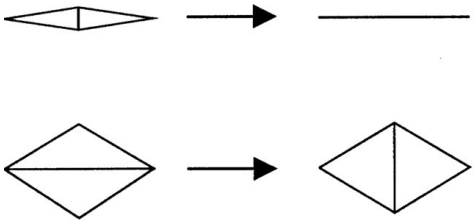


Abbildung 2.4.1.2-1: Topologische Optimierung mit NETGEN [33]: „point collapsing“ (oben) und „edge swapping“ (unten)

Das bis hier entstandene Netz kann optional durch eine Kombination von zwei verschiedenen Methoden optimiert werden, die NETGEN mitliefert. So kommt das Gitter näher an den Idealzustand von ausschließlich gleichseitigen Elementen. Die „metric optimization“ verschiebt iterativ einzelne Punkte auf der Fläche bzw. in dem Volumen, um die Abstände von Punkten zu Element-Schwerpunkten zu minimieren („vertex re-positioning“). Nachdem dieser Vorgang konvergiert ist, setzt die „topological optimization“ (siehe Abbildung 2.4.1.2-1) ein, um Punkte auf eine ganz neue Weise miteinander zu verbinden und einige von ihnen gänzlich zu löschen. Hierbei werden wiederum zwei Vorgehensweisen gebraucht. Beim „point collapsing“ wird geprüft, ob ein Zusammenlegen von zwei Punkten auf eine gemeinsame Linie das Netz an dieser Stelle verbessert. Die andere Methode evaluiert bspw. im 2D Bereich, ob vier Punkte, die zwei angrenzende Dreiecke beschreiben, in einer anderen Verbindung gleichmäßiger verteilte Winkel ergeben. Für die Optimierung ist eine interne Bewertung der Elemente notwendig. Dies geschieht nicht mit der allgemein üblichen Berechnung des Verhältnisses von maximaler Kantenlänge eines Elements zum Radius des inneren Kreises im 2D / der inneren Kugel im 3D (Q_{tet} siehe Gleichung 2.5-6). Für diese Aufgabe wird stattdessen jeweils für den 2D und 3D Bereich eine zweiteilige Funktion herangezogen:

$$E_{tri} = \frac{\sqrt{3}}{36} \cdot \frac{\left(\sum_i l_i\right)^2}{A_{tri}} + \sum_i \left(\frac{l_i}{k} + \frac{k}{l_i} - 2 \right) \tag{2.4.1.2-1}$$

$$E_{tet} = \frac{1}{6^4 \cdot \sqrt{2}} \cdot \frac{\left(\sum_i l_i\right)^3}{V_{tet}} + \sum_i \left(\frac{l_i}{k} + \frac{k}{l_i} - 2 \right) \tag{2.4.1.2-2}$$

Die Längen l_i stehen für die drei bzw. sechs Kantenlängen des Elements. Der vom Benutzer eingegebene Parameter für die gewünschte Kantenlänge ist k. Der erste Term dieser Funktionen bestraft zu flache Elemente. Der zweite Term ahndet Elemente, die zu klein oder zu groß ausfallen. Mit sinkender Elementqualität gilt: $E_{tri} \to \infty$ bzw. $E_{tet} \to \infty$. Die Berechnung dieser Funktion ist zweckdienlicher, als das oben genannte geometrische Verhältnis und zusätzlich günstiger in der Berechnung. [33]

2.4.2 Delaunay Algorithmen

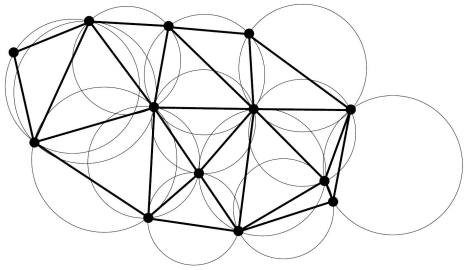


Abbildung 2.4.2-1: Delaunay Triangulierung in der Ebene [36]

Die Grundlage für diese Algorithmusfamilie wurde 1934 vom russischen Mathematiker Boris Nikolaevich Delaunay gelegt, der auch unter dem Namen Delone publiziert hat. Seine Abhandlung bemüht eine vereinfachte Anwendung des Theorems von Voronoï, um dieses in der Geometrie populärer zu machen [8]. Dabei geht es um das sogenannte Delaunay-Kriterium. Dieses wurde 1981 von Bowyer [5] und Watson [45] zum ersten Mal in einem Algorithmus verwendet, um aus einer Punktmenge möglichst gleichseitige Dreiecke und Tetraeder zu generieren. Das Kriterium maximiert automatisch die kleinsten Innenwinkel der entstehenden Elemente. Die Delaunay-Bedingung besagt im Fall eines zweidimensionalen Raums, in dem die Punktmenge liegt, dass der Kreis durch die drei Punkte eines Dreiecks niemals Punkte von benachbarten Dreiecken einschließen darf (siehe Abbildung 2.4.2-1). Aus den vorhandenen Punkten werden nur Dreiecke erzeugt, die diese Bedingung erfüllen. Im 3D Bereich gilt diese Bedingung analog für die Sphäre, die durch die vier Punkte des Tetraeders beschrieben wird und keinen fünften Punkt enthalten darf.

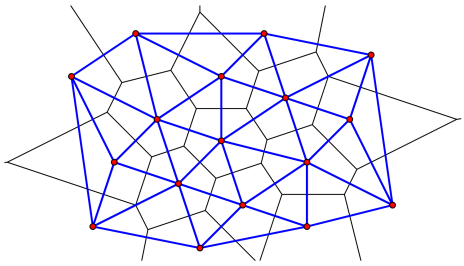


Abbildung 2.4.2-2: Voronoï Diagramm (schwarz), Delaunay-Netz (blau) und Voronoï-Zentren (rot) [23]

Die Delaunay Triangulierung ist eng verwandt mit Voronoï Diagrammen (siehe Abbildung 2.4.2-2). Jede Voronoï Region V_i ist ein konvexes Polygon (im $\mathbb{R}^2$) und gehört zu einem Voronoï-Zentrum, einem Punkt P_i aus der Punktemenge. In diesen Diagrammen liegt jeder Punkt aus einer Region V_i näher an P_i als an einem anderen Punkt P_j. Verbindet man alle Punkte von aneinander grenzenden Voronoï Regionen, entsteht ein Delaunay-konformes Netz („Dirichlet Tesselierung“) [29].

Es gibt drei grundsätzlich verschiedene Strömungen innerhalb der Algorithmen, die mit der Delaunay-Bedingung arbeiten. Im Folgenden werden ihre jeweiligen Stärken und Schwächen kurz ausgeführt, die sich aus der Herangehensweise ergeben. Je nach Anwendung des Gitters kann die eine oder andere Methode die Diskretisierung geeigneter vornehmen. Eine ausführliche Übersicht hierzu ist zu finden bei [11] und in [38] (mit zahlreicher weiterführender Literatur).

Boundary Constrained Delaunay Algorithmen fokussieren ein exaktes Beibehalten der initialen Grenzlinien und -flächen der zugrunde liegenden Geometrie. Bei diesem Ansatz wird die Geometrie nicht durch die Vernetzung vereinfacht und passt zu vielen Anwendungen im Ingenieurwesen, wo die Netze der numerischen Diskretisierung dienen. Mit diesem Anspruch, bzw. mit der Bindung an die „Boundary“ gehen folgende Nachteile einher: Die Netzqualität hängt unmittelbar mit der Güte der Geometrie zusammen. Ferner sind bei vielen Vertre-

tern heuristische Methoden in den Algorithmen involviert, die kein theoretisch erfolgreiches Terminieren garantieren können. Nichtsdestotrotz ist dieser Ansatz in der Praxis sogar für automatisierte Netzgenerierung relevant [43, 35], weil relevante Geometrien bspw. in der Strukturanalyse nicht grundsätzlich arbiträre Sonderfälle sind. [38]

Delaunay Refinement Algorithmen stellen die zweite Grundrichtung dar. Hier steht die theoretisch beweisbare Qualität des Netzes im Vordergrund, die unter gewissen Bedingungen stets erreicht wird. Zu diesem Zweck ist es jedoch unvermeidbar, das durch die initialen Grenzlinien und -flächen der Geometrie definierte „Randnetz“ zu modifizieren. Dafür gibt es verschiedene Ansätze:

- Linien und Dreiecke des initialen Randnetzes werden durch zusätzliche Unterteilung verfeinert. Dem Randnetz werden neue Punkte hinzugefügt.
- Entfernen der Elemente, die zu kleine Winkel aufweisen. Damit wird die ursprüngliche Geometrie vereinfacht, wobei trotzdem eine starke Abhängigkeit von der Qualität der Ausgangsgeometrie bestehen bleibt.
- Das durch die initialen Grenzlinien und -flächen der Geometrie definierte „Randnetz“ wird nicht weiterverwendet. Es wird stattdessen ein neues, vom Algorithmus selbst berechnetes Netz als Ausgangslage gewählt. Damit findet auch in diesem Fall eine Vereinfachung zu Gunsten der höheren Unabhängigkeit von der Geometrie statt.

Die Problematik dieser drei Methoden ist ihnen inhärent. Die Wiedergabe der Geometrie durch das Netz ist nicht exakt gegeben. Zusätzlich sind die theoretisch garantierten, hochwertigen Netze nur für Geometrien garantiert, deren Oberflächen keine spitzen Winkel unterhalb einer kritischen Grenze aufweisen. [38]

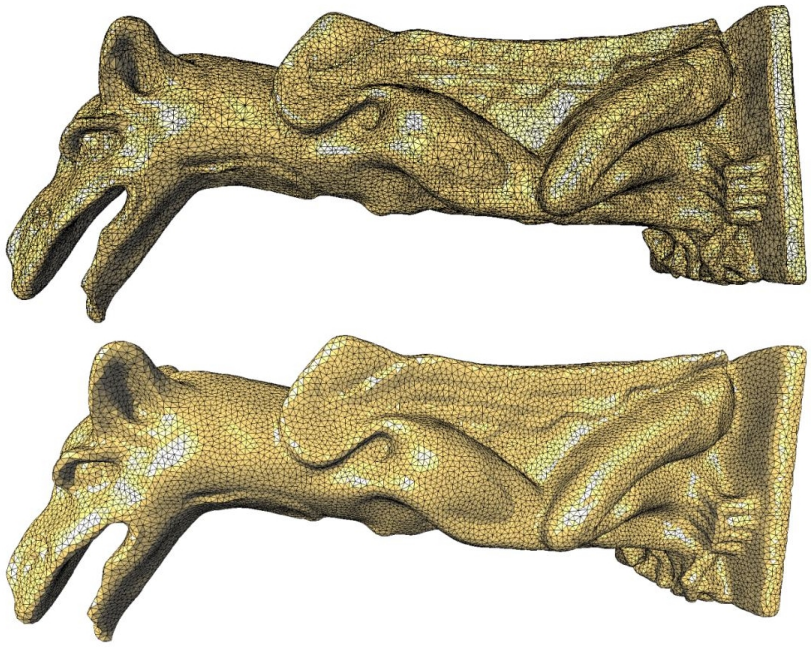


Abbildung 2.4.2-3: Vergleich von Variational Voronoï und Delaunay Refinement Methodik [1]: Das Delaunay Refinement Netz (oben) aus 200.000 Tetraedern [1] zielt laut der Autoren explizit auf Reduktion von schlecht konditionierten Tetraedern und auf Minimierung der Elementanzahl ab [7]. Das Variational Voronoï Netz (unten) hat 260.000 Tetraeder. Es hat eine vergleichsweise gleichmäßigere Form und Anordnung der Elemente, jedoch werden die gut konditionierten Tetraeder auf Kosten der Genauigkeit erstellt. Die ursprüngliche Form des Gargoyle wird vom Delaunay Refinement Netz präziser wiedergegeben.

Variational Voronoï Methoden sind das dritte Mitglied der Delaunay Familie. Hier findet die Optimierung des Netzes schon während der Generierung desselben statt. Dies geschieht mittels sogenannter Energiefunktionale, die lokal oder global minimiert werden. Dabei werden die Eigenschaften des Gitters mit der Energie in einem System dargestellt. Für dieses System wird der Zustand mit der geringsten Energie gesucht. Dafür gibt es verschiedene Ansätze, die teilweise mehrere Energiefunktionale kombinieren. Bei [1] wird die Energie beispielsweise minimiert, indem Punkte verschoben *und* Alternativen zu bestehenden Verbindungen von Punkten gesucht werden.
Populäre Vertreter der Variational Voronoï Methode ist die „Centroidal Voronoï Tesselation (CVT)“ [9] und die gewichtete Delaunay Triangulierung bei [10]. Diese Methoden bedürfen aufgrund der auszuwertenden (u. U. globalen) Energiefunktionale von allen Delaunay Vertretern die höchste Rechendauer [1]. Ähnlich den Delaunay Refinement Algorithmen wird auch bei diesem Ansatz das Netz von der Ausgangsgeometrie stellenweise losgelöst. Die Ergebnisse hingegen sind ausgesprochen gleichmäßig geformte und angeordnete Elemente, deren Anzahl vergleichsweise gering bleiben kann, um eine Geometrie wiederzugeben (siehe Abbildung 2.4.2-3). Diese Eigenschaften stellen im Bereich von Simulationen wichtige Kriterien dar. [38]

Die Grenzen zwischen den drei vorgestellten Ansätzen sind fließend, in der Literatur werden immer wieder neue Kombinationen vorgenommen, um unvermeidliche Nachteile „aufzufangen“. Ein passendes Beispiel hierfür ist der TetGen Netzgenerator [38, 39]. Diese Software verbindet die ersten beiden Ansätze „Boundary Constrained“ und „Delaunay Refinement“.

2.4.3 Mesh Adapt

Der Mesh Adapt Algorithmus wird bei Gmsh für die Netzgenerierung von Flächen angeboten. Er ist in der Literatur weniger weit verbreitet und weder mit Delaunay Algorithmen, noch mit Advancing Front Methoden verwandt. Entwickelt wurde er u. a. von Jean-François Remacle, der gleichzeitig ein Hauptautor von Gmsh ist. Um nachzuvollziehen, weswegen Mesh Adapt unter Gmsh die höchste Robustheit bei Oberflächenvernetzung aufweist (siehe Tabelle 3.4.2.5-1), ist ein kurzer Exkurs in die Gmsh 2D-Algorithmen und die Grundlagen von Gittererzeugung auf gekrümmten Oberflächen sinnvoll. Dabei bieten sich zweierlei Möglichkeiten:

- Vernetzung der Fläche im dreidimensionalen Raum $\mathbb{R}^3$
- Vernetzung der Fläche in einem zweidimensionalen parametrischen Raum

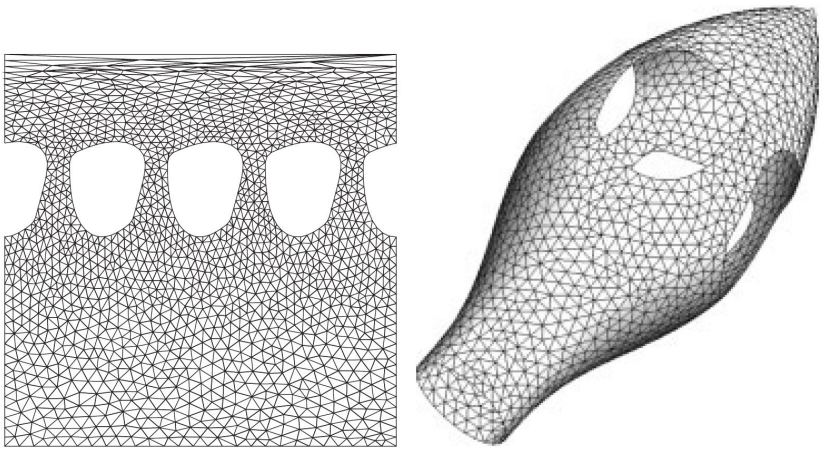


Abbildung 2.4.3-1: Oberflächenvernetzung im Parameterraum [16]: „Abwicklung“ des Netzes aus dem $\mathbb{R}^3$ (rechts) in einen zweidimensionalen Parameterraum (links) mit hoch anisotropen Elementen nahe der Spitze

Der Großteil aller Vernetzungsalgorithmen konzentriert sich auf die erste Variante. Der offensichtliche Vorteil ist der direkte Weg von der Geometrie zum Netz. Es ist kein zusätzlicher Schritt für die Parametrisierung notwendig. Wie bereits erläutert, besitzen Delaunay Algorithmen im $\mathbb{R}^3$ jedoch keine uneingeschränkte Robustheit oder garantierte Terminiertheit, sondern gründen sich z. T. auf Heuristik. (Bei Gmsh ist kein reiner Advancing Front Ansatz für Oberflächengitter vorgesehen, wobei NETGEN eine garantierte Terminiertheit für Oberflächennetze im $\mathbb{R}^3$ hat.)
Mesh Adapt erlangt seine Robustheit durch Vernetzung der Geometrie im parametrischen Raum[1]. Sofern eine Parametrisierung der Oberflächen verfügbar ist, kann eine Triangulierung in der Ebene in jedem Fall erreicht werden. Aus der Abbildung 2.4.3-1 ist ersichtlich, dass für diese Aufgabe nur Algorithmen in Frage kommen, die für anisotrope, ungleichmäßige Netze ausgelegt sind. Nur so kann das Äquivalent im dreidimensionalen Raum isotrop werden.
Gmsh gestaltet die Vernetzung von Oberflächen grundsätzlich in einem Parameterraum. Hierfür existiert eine Implementierung des Delaunay Algorithmus auf Grundlage der Arbeit von [13]. Außerdem steht dem Benutzer von Gmsh eine Abwandlung der Delaunay Methode zur Verfügung, die einige Komponenten der Advancing Front Familie beinhaltet [29]. Der Vorteil dieser Kombination liegt darin, dass keine Suche nach topologischen Ähnlichkeiten notwendig ist. Die Dreiecke in der Nähe von Netzrändern haben jedoch wie bei Moving Front Methoden sehr gute Eigenschaften. Der proprietäre Mesh Adapt Algorithmus hat gegenüber den Ansätzen von [13] und [29] jedoch den Vorteil, dass er ohne Ableitungen der Parametrisierung bzw. des Mappings auskommt. Dies ist ein entscheidender Vorteil im Sinne der Robustheit, wenn die Parametrisierung singulär wird (siehe Spitze der Geometrie in Abbildung 2.4.3-1).

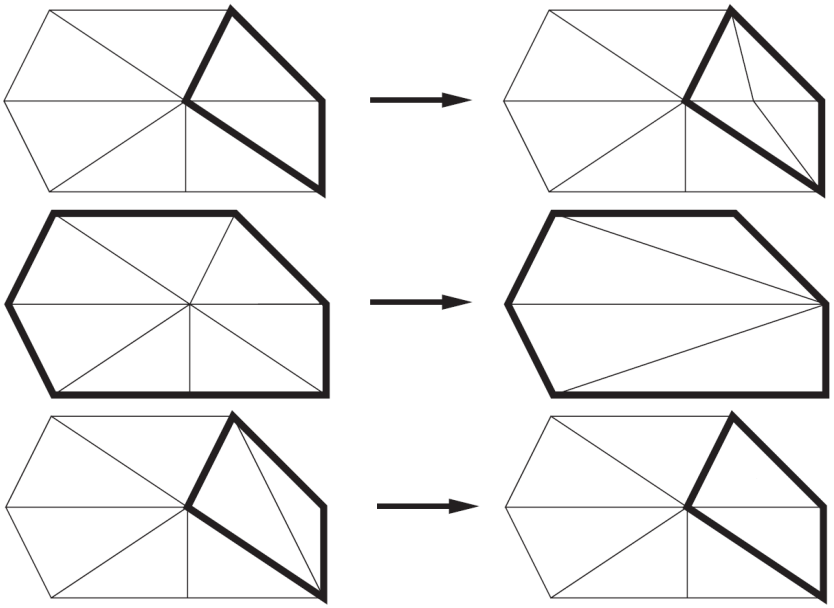


Abbildung 2.4.3-2: Netzoptimierung im Mesh Adapt Algorithmus [16]: oben: „edge splitting“, mittig: „edge collapsing“, unten: „edge swapping“

Der Mesh Adapt Algorithmus funktioniert wie folgt:

- Ein initiales Netz wird im Parameterraum aufgebaut. Dieses enthält alle Punkte, die die Linien und Flächen im $\mathbb{R}^3$ beschreiben.
- Alle Ecken und Kanten aus dem $\mathbb{R}^3$ werden im Parameterraum wiederhergestellt.
- In einem dritten Schritt wird das Netz iterativ auf vierfache Weise modifiziert und optimiert (siehe Abbildung 2.4.3-2). Dabei beinhaltet jede Iteration folgende Operationen:

[1] TiGL arbeitet intern ausschließlich mit Oberflächen im parametrischen Raum [40].

- „edge splitting“: Zu lange Kanten werden zweigeteilt.
- „edge collapsing“: Zu kurze Kanten werden aufgelöst.
- „edge swapping“: Neue Verbindungen für optimalere Dreieckswinkel werden gesucht.
- „vertex re-positioning“: Kleinster an einen Punkt grenzender Winkel wird optimiert.

Alle Gmsh Netzgeneratoren speichern für jeden Punkt des Gitters die realen $(x,\ y,\ z)$ und die parametrischen $(f,\ g)$ Koordinaten. Dies ist notwendig, weil im $\mathbb{R}^3$ bewertet wird, ob sich für eine der (in diesem Fall) vier Modifikationen eine *Gelegenheit* bietet. Anschließend wird die betreffende Modifikation im Parameterraum auf ihre *Validität* geprüft. In der Praxis konvergiert der Mesh Adapt Algorithmus nach bereits 6-8 Iterationen und stellt eine schnelle, robuste Alternative zu Delaunay, bzw. Frontal-Delaunay dar. [16]

2.4.4 Zusammenfassung

Die Generierung von Netzen, wie sie in diesem Kapitel dargestellt ist, befindet sich immer im Spannungsfeld von Robustheit, Elementqualität und Wiedergabe der Geometrieoberfläche.
NETGEN als Vertreter der Advancing Front Algorithmen priorisiert die Geometrieoberfläche, hat aber im dreidimensionalen Raum einen Schwachpunkt in der Robustheit. Dieser Umstand wird durch einen alternativen Algorithmus aufgefangen. So bleiben in der Praxis nur eine zu vernachlässigende Anzahl von Sonderfällen eine Gefahr für die Konvergenz.
Delaunay Algorithmen sind in der Fachwelt weiter verbreitet und mehr diversifiziert, bieten darum für jede der drei widersprüchlichen Ziele eine Lösung. Aber auch hier bleibt die Spannung stets erhalten.
Mesh Adapt bietet eine sehr robuste Lösung für die 3D-Oberflächenvernetzung an, macht aber Kompromisse bei der Geometrietreue.
Ein Teil der Spannungen kann und wird bereits aufgelöst, wenn verschiedene Ansätze kombiniert werden und diverse Optimierer zum Einsatz kommen. Neben dem NETGEN Optimierer in Kapitel 2.4.1.2 eignet sich hierfür das klassische und erweiterte „Laplace Smoothing“ [26].

2.5 Bewertungsmethoden für triangulierte Netze

Für eine aussagekräftige CFD Simulation oder eine effektive Radarerkennung ist neben einer hochwertigen Geometrie auch ein *qualitatives* Netz notwendig. Die bisher beschriebenen Methoden sind darum auf eine adäquate Überwachung des Vernetzungsprozesses angewiesen. Insbesondere die Generierung von volumetrischen Gittern ist eine nichttriviale Aufgabe. Bei diesem Vorgang wird die Grundlage für die numerische Berechnung und die Auflösung physikalischer Vorgänge gelegt. Dafür darf das Netz bspw. im Hinblick auf das gewählte CFD-Turbulenzmodell an bestimmten Flächen nicht zu fein *oder* zu grob sein, weil die Modellierung der turbulenten Grenzschicht sonst nicht mehr valide ist [30]. Abgesehen von den Abmaßen der Tetraeder muss auch eine hinreichende Gleichmäßigkeit in der Formgebung („aspect ratio“) und dem Zellenwachstum („growth ratio“) gegeben sein. Ist dies nicht der Fall, können erhöhte Diskretisierungsfehler in der numerischen Lösung auftreten. Neben fehlerhaften Ergebnissen für die PDEs ist auch eine verlängerte Berechnungsdauer eine potentielle Konsequenz. Unter Umständen wird die Konvergenz gar nicht erreicht, weil deformierte oder zu große Zellen in Bereichen mit hohen Gradienten das Ergebnis oszillieren lassen. Degenerierte Elemente in einem Netz können darüber hinaus sogar einen Abbruch einer Simulation herbeiführen. [20]
Die direkte Korrelation zwischen Gittern und Numerik ist ein eigenes, wichtiges Forschungsfeld. Dabei liegt das Augenmerk u. a. auf der netzabhängigen Kondition der Matrizen und der Größe ihrer Eigenwerte. In der Literatur bieten Knupp [20] und Shewchuk [37] jeweils Metriken für Netzbewertung nach numerischen Kriterien an. Im Folgenden werden jedoch die

geometrischen Bewertungsmethoden beleuchtet, die bei Netzoptimierern und automatischen Gittergeneratoren weit verbreitet sind.

Die Bewertung eines Netzes, sei es im $\mathbb{R}^3$ oder $\mathbb{R}^2$, kann in SI-Einheiten oder einheitenlos geschehen. Die zweite Möglichkeit findet insbesondere bei internen Berechnungen einer Vernetzungssoftware ihre Anwendung. Optimierungsalgorithmen benötigen ein Maß, das unabhängig von der tatsächlichen Größe eines Elements Aussagen über dessen Deformierungsgrad und den Optimierungsbedarf treffen kann (siehe auch Kapitel 2.4.1.2). Beispiele für ungünstige Tetraederformen sind in der Abbildung 2.5-1 zu finden.

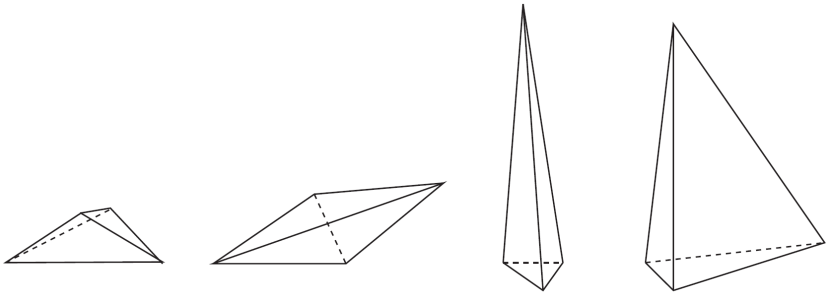


Abbildung 2.5-1: Tetraeder mit optimierbaren Eigenschaften [4]: Von links nach rechts: stumpfe, große Winkel in einem Simplex; Splitter-Element; nadelähnlicher Tetraeder; keilförmiges Element

Bei den Metriken, die auf physikalische Größen setzen, seien Winkel, minimale / maximale Kantenlängen, Radien, Flächen und Volumina als prominente Vertreter genannt. In vielen Fällen werden diese Einheiten nicht nur einzeln gemessen, sondern zueinander in Relation gestellt, um eine fallspezifische Bewertung einer bestimmten Eigenschaft vorzunehmen.
Damit entstehen meistens einheitenlose Kriterien, weil die Einheiten sich kürzen. Allgemein gibt es für diese Kriterien in der Literatur übliche Vereinbarungen, um die Vergleichbarkeit zu fördern und zu gewährleisten [37]:

- Degenerierte Elemente haben die Qualität „0“.
- Das Maß bleibt unabhängig von der Skalierung der Geometrie.
- Die Bewertung wird normiert, sodass ideale Tetraeder / Dreiecke die Qualität „1“ haben.
- Invertierten Elementen wird ein negativer Wert zugewiesen. (nicht immer realisierbar)

An ein Netz werden in manchen Fällen annähernd widersprüchliche Ansprüche gestellt. Typischerweise sind gleichseitige und gleichwinklige Elemente der Idealfall. Darüber hinaus gibt es aber auch den Bedarf an hoch anisotropen Zellen, die gut geeignet sind, um besondere physikalische Vorgänge abzubilden (siehe Abbildung 3.4.2.6-1). Hierbei wird entsprechend eine bestimmte Orientierung der Elemente gefordert [36]. Dies verdeutlicht, weswegen die Netzgeneratoren in der Literatur dutzende verschiedene Bewertungskriterien verwenden, abhängig vom Algorithmus und den speziellen Anforderungen. Eine umfassende Übersicht über die üblichen Metriken für Tetraeder und Dreiecke findet sich inkl. individueller Visualisierungen, Einschätzung zur Aussagekraft und einer Beschreibung der Stärken und Schwächen bei [37].

Beispielhaft werden nun zusätzlich zu den NETGEN Optimierungskriterien aus Kapitel 2.4.1.2 vier weitere einheitenlose Bewertungsmethoden vorgestellt. Die *Radius Edge Ratio* ρ_{tet} ist das Kriterium, das von Delaunay Refinement Algorithmen optimiert wird, um das dreidimensionale Netz in eine bessere Struktur zu überführen. Dafür wird die Umkugel des Tetraeders mit

seiner kürzesten Kante in Relation gebracht [37]:

$$\rho_{tet} = \frac{\sqrt{6}}{4} \cdot \frac{l_{min}}{r_{Umkugel}} \tag{2.5-1}$$

Für gleichseitige Tetraeder wird ρ_{tet} minimal. Das Maß ist aufgrund seiner zu großen Toleranz gegenüber splitterähnlichen Simplizes (siehe Abbildung 2.5-1) nur bedingt geeignet, schlecht konditionierte Elemente zu erkennen, wird aber im Delaunay Refinement angewendet [44, 37].

Als zweites Beispiel wird die weit verbreitete, *Radius Ratio* γ_{tri} bzw. γ_{tet} [6, 44] vorgestellt. Diese Maß wird u. a. intern in Gmsh verwendet, kann dort aber auch visualisiert werden. Einfach ausgedrückt spiegelt es folgende Verhältnisse wieder:

$$\gamma_{tri} = \frac{r_{Inkreis}}{r_{Umkreis}} \tag{2.5-2}$$

$$\gamma_{tet} = \frac{r_{Inkugel}}{r_{Umkugel}} \tag{2.5-3}$$

Die spezifische Formulierung für Dreiecke lautet in Gmsh [16]:

$$\gamma_{tri} = 4 \cdot \frac{\sin\hat{a} \cdot \sin\hat{b} \cdot \sin\hat{c}}{\sin\hat{a} + \sin\hat{b} + \sin\hat{c}} \tag{2.5-4}$$

Hierbei sind $\hat{a}$, $\hat{b}$ und $\hat{c}$ die inneren Winkel des bewerteten Dreiecks. Bei dieser Metrik hat ein gleichseitiges Dreieck die Qualität $\gamma_{tri} = 1$. Ein degeneriertes Element läuft mit schwindender Fläche gegen $\gamma_{tri} = 0$. Die spezifische Formulierung für Tetraeder lautet in Gmsh [16]:

$$\gamma_{tet} = \frac{6 \cdot \sqrt{6} \cdot V_{tet}}{(\sum_{i=1}^{4} A_i) \cdot max_{i=1,\,...,\,6} l_i} \tag{2.5-5}$$

Hier ist V_{tet} das Volumen des Tetraeders, A_i die Fläche der i-ten Seite und l_i die physikalische Länge der i-ten Kante des Tetraeders. Wiederum bewegt sich das Ergebnis von γ_{tet} im Intervall $[0, 1]$, wobei $\gamma_{tet} \approx 0$ einen splitterförmigen Tetraeder beschreibt (siehe Abbildung 2.5-1).

Das dritte Bewertungskriterium ist das *Minimum Sine Measure* (untersucht von [26]). Dabei wird der kleinste Sinus unter den Diederwinkeln θ_{ij} im Tetraeder gesucht. Sowohl besonders stumpfe ($\sim 180°$), als auch sehr spitze ($\sim 0°$) Winkel werden mit einer Qualität nahe 0 bemessen. Für ein perfekt gleichseitiges Element liegt das theoretisch erreichbare Maximum der Qualität bei $2 \cdot \sqrt{2}/3 \approx 0,943$ [3]. Diese Metrik stellt aufgrund der intuitiven geometrischen Bedeutung in [44] den Favoriten dar.

Zuletzt soll die Metrik vorgestellt werden, die für Visualisierungen in dieser Arbeit gewählt wurde (Abbildungen 3.4.1-1, 3.4.1-2, 5.6-1 und 5.6-2). Dieses Kriterium steht in der SALOME GUI für die Bewertung von Tetraedern zur Verfügung [31]. Ursprünglich wurde es in der Literatur von Baker [2] vorgeschlagen. Es setzt die längste Tetraederkante mit dem Radius der Inkugel in Relation. Der Faktor $2\sqrt{6}$ normiert Q_{tet} für ideale Tetraeder auf 1.

$$Q_{tet} = \frac{max_{i=1,\,...,\,6} l_i}{2 \cdot \sqrt{6} \cdot r_{Inkugel}} \tag{2.5-6}$$

3 Auswahl des Vernetzungscodes

An dieser Stelle wird der Auswahlprozess mit den zu Grunde liegenden Kriterien dargestellt. Zwei weitverbreitete Softwareprodukte, die viele der Kriterien erfüllen können, werden im Anschluss detailliert verglichen.

3.1 Anwendungsgebiete der Netze

Die vernetzten Flugzeuggeometrien werden grundsätzlich in frühen Entwicklungsstadien angewendet. An dieser Stelle hat der Ingenieur noch viele Freiheiten und kann die unterschiedlichsten Konfigurationen testen. Es lassen sich drei spezifische Anwendungen differenzieren:

- Generierung Volumennetz
  - Berechnung der aerodynamischen Eigenschaften (u. a. Auftrieb, Widerstand)
  - Vereinfachung der Geometrie ist akzeptabel
  - Potentielle numerische Löser sind z. B. OpenFOAM, TAU (DLR)
- Generierung Oberflächennetz
  - Anwendung in der Radar Erkennung
  - Detaillierte Wiedergabe der Oberflächen im Netz, keine Vereinfachungen
- Weiterverwendung des Netzes als „Starter“ für:
  - Weiterführende, kommerzielle Vernetzungssoftware
  - Sogenannte „overset meshes“

3.2 Anforderungen an die auszuwählende Software

Die folgende Auflistung zeigt in Anlehnung an die Vorgaben des DLR Projektes „Diabolo“ die Bewertungskriterien und -eigenschaften in absteigender Priorität, nach denen unter existierenden Gittergeneratoren gesucht wird:

- Potential für parametrisierte Vernetzung für vereinfachte Bedienung in TiGL
- Open Source
- Tetraeder (3D) und Dreiecke (2D)
- Robuster Algorithmus für Vernetzung
- Zeiteffizienz bei Netzgenerierung
- Plattformunabhängigkeit (Linux / Windows)
- Programmiersprache C++

- Kompatibilität mit TiGL Datenformaten (bestenfalls BRep)
- Keine oder wenige Abhängigkeiten zu anderen Softwarebibliotheken
- Optional: Open CASCADE als CAD Kernel
- Optional: Funktion zur Geometriemanipulation
- Optional: Funktion zur Bestimmung der Netzqualität in SI Einheiten

3.3 Vorauswahl

Eine Eingrenzung des Angebots an Vernetzungssoftware auf die Open Source Produkte liefert dennoch mehr als 100 Möglichkeiten[1] zur Auswahl. Viele kommen trotzdem wegen der folgenden Gründe nicht in Betracht:

- Ein vergleichsweise geringes Angebot an Dokumentation, oder aber eine Dokumentation, die nicht in Englisch verfügbar ist (Méfisto).
- Sehr spezifisch zugeschnittener Einsatz der Software: beispielsweise Vernetzung in der Geografie (Globegen) oder für magnetische Felder (xprob).
- Mehrfachverwendung einer zugrunde liegenden Bibliothek: So existieren mehrere Vernetzungsprogramme, die alle auf dem NETGEN Algorithmus basieren. Damit besteht nicht mehr die Notwendigkeit, jedes dieser Softwareprodukte in Betracht zu ziehen. In solch einem Fall trägt der allgemeine Funktionsumfang der Software zur Entscheidung bei.
- Weiterentwicklung ist seit mehr als 8 Jahren eingestellt (delaundo oder GridTool).
- Keine triangulierten, unstrukturierten Netze (SnappyHexMesh von OpenFOAM).
- Ebenso sind nicht alle Codes dafür geeignet, dreidimensionale Gitter zu erstellen.
- Die Programmiersprache des Source Codes ist nicht C oder C++.

Folgende Gittergeneratoren / Codes / Algorithmen stehen anschließend in der Vorauswahl, weil sie über eine Dokumentation verfügen, triangulierte Netze erzeugen und in C++ geschrieben sind:

- NETGEN
- TetGen
- Gmsh
- SALOME
- enGrid
- DelPSC
- OpenMesh
- OpenVolumeMesh
- MeshGenC++
- NWGrid
- MeshLab
- GNU Triangulated Surface Library (GTS)

Aus dieser Menge stechen Gmsh und SALOME heraus, weil die Dokumentation überdurchschnittlich umfassend ist, weil sie in der Open Source Community sehr weit verbreitet sind und entsprechend viel Hilfestellung verfügbar ist. Beide Softwarepakete werden gepflegt und aktuell gehalten, die Datenformate sind in der Theorie kompatibel und es bestehen Funktionalitäten für CAD Manipulation, Gittervisualisierung und -bewertung. Weil ein Vergleich unter 12 potentiellen Produkten einen hohen Zeitaufwand bedeutet, beschränkt sich diese Arbeit auf Gmsh und SALOME.

[1] Die Auflistungen auf den Websites [32] und [42] erheben nicht den Anspruch auf Vollständigkeit, liefern aber kombiniert einen umfassenden Katalog.

3.4 Gegenüberstellung zwischen Gmsh und SALOME

SALOME und Gmsh eignen sich potentiell beide für die die Einbettung in TiGL. Dafür sind die individuellen Fähigkeiten der Programmierschnittstellen und deren Zugänglichkeit von höchster Bedeutung. Ein detaillierter Vergleich auf Grundlage der jeweiligen API soll nun begründen, welche Software weiterverfolgt wird. Die Einarbeitung in die Vernetzung mit diesen Programmen impliziert, dass im Anschluss detaillierte Aussagen über Funktionsumfang, Dokumentation, etc. getroffen werden können.
Auf den Abbildungen 3.4-1 und 3.4-2 sind jeweils Vernetzungen eines Geometriebeispiels abgebildet, das mit dem NETGEN Algorithmus in der jeweiligen GUI erstellt und anschließend optimiert ist. Bei der Optimierung werden Zellen teilweise neu angeordnet, sodass die Tetraeder verbesserte Seitenverhältnisse erhalten. Dies ist später der Numerik bei der CFD Simulation zuträglich. Gleichzeitig reduziert sich die Anzahl der Zellen, weil manche Tetraeder mit anderen verschmelzen. Beide Ergebnisse wurden ohne Parallelisierung auf einem Intel Core i7-3520M Prozessor mit 3,6 GHz berechnet, weil Gmsh und SALOME keine parallelisierten Vernetzungsalgorithmen aufweisen.

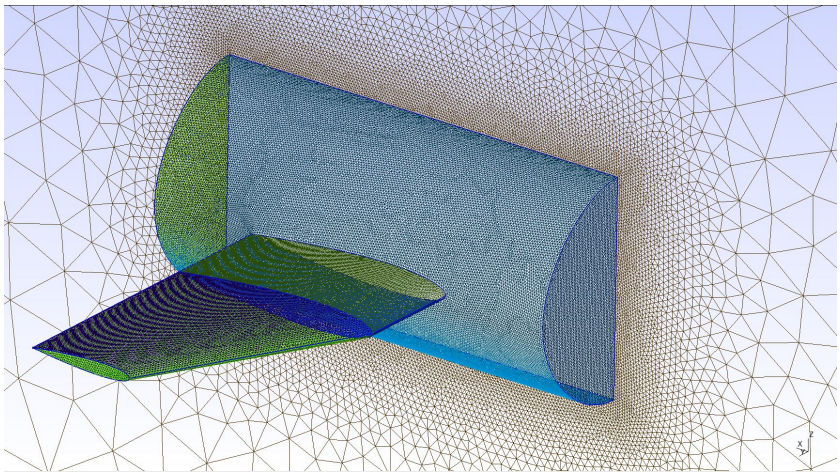


Abbildung 3.4-1: Gmsh Vernetzung des Geometriebeispiels: Gmsh Gitter mit 682.000 Tetraedern (904.000 vor der Optimierung); NETGEN Algorithmus und anschließende NETGEN Optimierung; Dauer: 131s in der GUI; Dargestellt ist das Oberflächennetz.

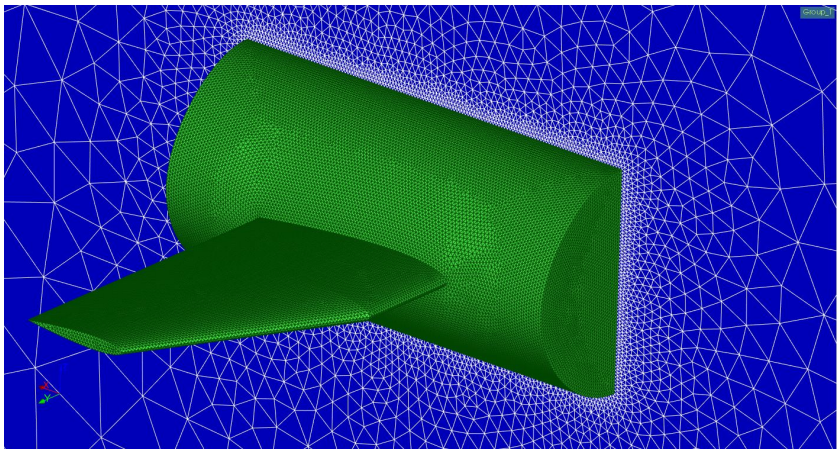


Abbildung 3.4-2: SALOME Vernetzung des Geometriebeispiels: SALOME Gitter mit 662.000 Tetraedern (956.000 vor der Optimierung); NETGEN Algorithmus und anschließende NETGEN Optimierung; Dauer: 145s in der GUI; Dargestellt ist das Oberflächennetz.

3.4.1 Gemeinsamkeiten

In den folgenden Aspekten bestehen zwischen Gmsh und SALOME Gemeinsamkeiten:

- NETGEN als Vernetzungsalgorithmus
- Open CASCADE als CAD Basis
- Ausführen und Verfassen von Skripten

Für die eigentliche Vernetzung ist in beiden Tools u.a. NETGEN implementiert. Die Elementqualität ist bei den resultierenden Gittern in den Abbildungen 3.4-1 und 3.4-2 durch die Optimierung relativ hoch (kein Seitenverhältnis ≥ 9) und gleichwertig. Dies kann aus den unten stehenden Diagrammen in den Abbildungen 3.4.1-1 und 3.4.1-2 abgeleitet werden.

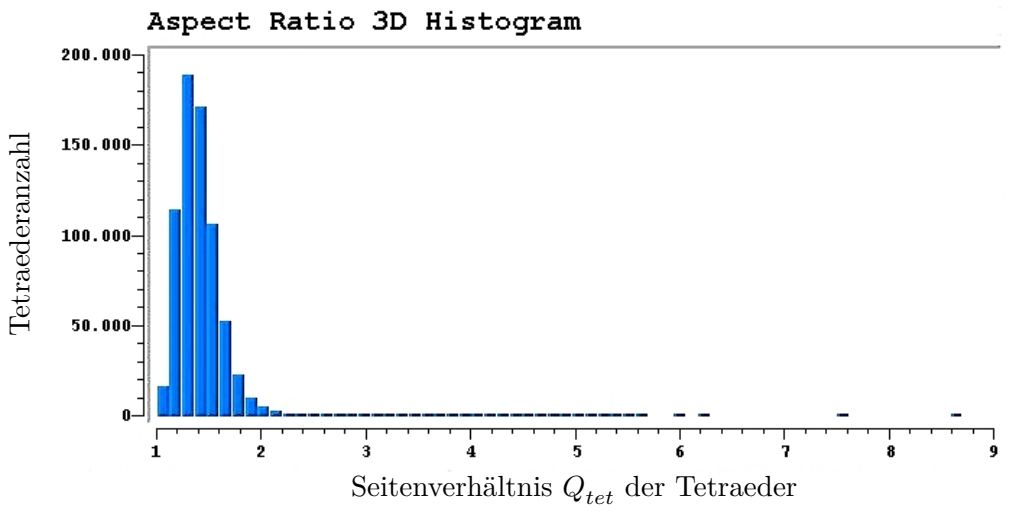


Abbildung 3.4.1-1: Statistische Verteilung der Seitenverhältnisse eines Gmsh Gitters: Gmsh Gitter der Beispielgeometrie mit 682.000 Zellen (erstellt mit NETGEN inkl. NETGEN Optimierung): Verteilung der Seitenverhältnisse (Q_{tet} siehe Gleichung 2.5-6) über die Zellenanzahl

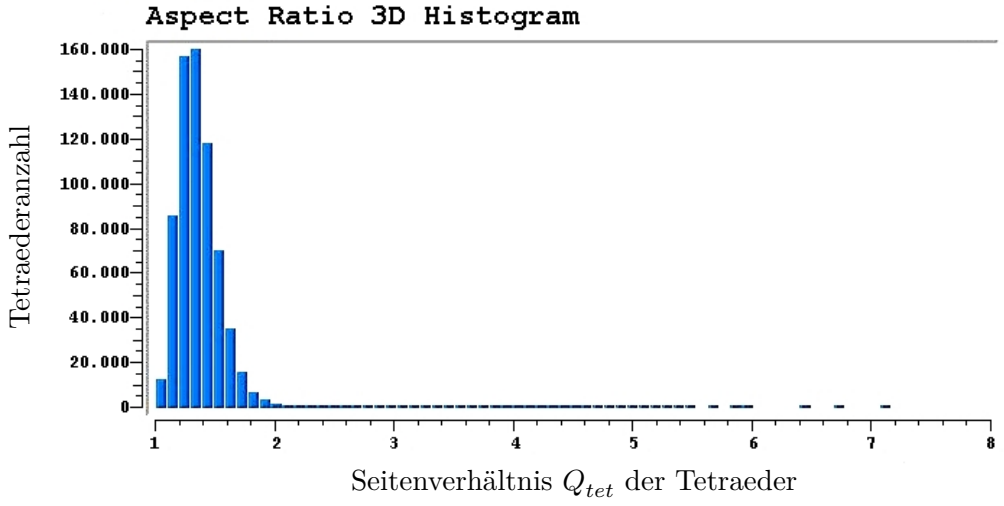


Abbildung 3.4.1-2: Statistische Verteilung der Seitenverhältnisse eines SALOME Gitters: SALOME Gitter der Beispielgeometrie mit 662.000 Zellen (erstellt mit NETGEN inkl. NETGEN Optimierung): Verteilung der Seitenverhältnisse (Q_{tet} siehe Gleichung 2.5-6) über die Zellenanzahl

Zusätzlich zur Vernetzung bieten SALOME und Gmsh Funktionalitäten zur CAD Manipulation an. Dies beinhaltet die Erstellung von neuen geometrischen Objekten (1D, 2D, 3D), Transformationen, Extrusionen und Boolesche Operationen. Die CAD Bearbeitung wird in beiden Programmen auf der Basis von Open CASCADE (ein Open Source CAD Kernel) ausgeführt. Dies wird sich zu späterem Zeitpunkt als vorteilhaft erweisen, da somit uneingeschränkte Kompatibilität[2] zu TiGL gegeben ist.
In beiden Softwareprodukten können die Ergebnisse aus der Arbeit in der Benutzeroberfläche in Form von Skripten automatisiert festgehalten bzw. exportiert werden. So entsteht bei SALOME ein Python Skript. Gmsh schreibt in einer simpel aufgebauten, proprietären Skriptsprache. Diese bedarf nur einer vergleichsweise kurzen Einarbeitung (wenige Stunden). Natürlich können diese Skripte auch manuell weiterverarbeitet und modifiziert werden. So lassen sich eine Vielzahl von Netzen zeiteffizient erzeugen.

3.4.2 Unterschiede

In den folgenden Aspekten bestehen zwischen Gmsh und SALOME markante Unterschiede:

- Entwicklungszweige
- Datenformate
- Zeiteffizienz
- Dokumentation und Support
- Verfügbare Algorithmen
- Steuerung der Netzeigenschaften
- Informationen zur Gitterqualität

3.4.2.1 Entwicklungszweige

Bei SALOME bestehen in der Open Source Gemeinschaft verschiedene Entwicklungszweige. Für die Festlegung auf einen dieser SALOME Zweige stellen sich drei Bedingungen. Unter Berücksichtigung des Vergleichs zwischen den NETGEN und MEFISTO Algorithmen (siehe Abbildung 3.4.2.5-1) wird die Verfügbarkeit von NETGEN zur festen Voraussetzung für die Entscheidung zwischen Gmsh und SALOME gemacht. Ebenso gehört die Ansteuerung der Softwarefunktionen über eine C++ API zu den zwingenden Bedingungen. Idealerweise existiert zu der API auch eine Dokumentation. Diese drei Kriterien erfüllt Gmsh in allen Details. Die SALOME Entwicklungszweige erfüllen einzelne der drei Bedingungen, jedoch niemals alle gemeinsam. Entwicklungszweige von SALOME:

- Offizielle GUI mit MEFISTO[3] und NETGEN Algorithmen inkl. Dokumentation und einer Ansteuerung mittels Pythonsktipten
- Python API mit MEFISTO Algorithmus und Dokumentation[4]
- C++ API mit MEFISTO und NETGEN Algorithmen ohne Dokumentation[5]

Die dritte Variante kommt den geforderten Ansprüchen am nächsten und wird für den Vergleich mit der Gmsh API herangezogen.

3.4.2.2 Datenformate

Bei den Datenformaten (siehe Tabelle 3.4.2.2-1) gibt es Überschneidungen (hervorgehoben) und auch Unterschiede. Der Import von Geometrien stellt bei beiden Programmen keine

[2] IGES, STEP und vor allem BRep als gemeinsame Nenner für Geometriedarstellung, siehe Tabelle 3.4.2.2-1.
[3] MEFISTO ist in SALOME der Standard für triangulierte Vernetzung.
[4] `https://github.com/tpaviot/smesh`.
[5] `https://github.com/LaughlinResearch/SMESH`.

Schwierigkeit dar, weil TAU, Gmsh und SALOME auf Open CASCADE aufbauen. Ausschlaggebend für die Workflow Einbindung und die Simultationen mit OpenFOAM oder TAU sind also die Export Datenformate, insbesondere .cgns[6] und .unv[7].

SALOME ⇒ OpenFOAM
Im Zuge der Workflow Tests wurde eine TiGL Geometrie erfolgreich mit der SALOME GUI in das .unv Format exportiert, um anschließend mit OpenFOAM in einer RANS Simulation gerechnet zu werden. Aus später angeführten Gründen („submeshes“ in Kapitel 3.4.2.6), die nicht mit dem Format zusammenhängen, ist dieser Vorgang aus der SALOME API heraus nicht möglich.

SALOME ⇒ TAU
Mit einem Zwischenschritt in Tecplot kann ein SALOME .cgns Netz von TAU interpretiert werden. Jedoch werden die enthaltenen, untergeordneten Netze auf einzelnen Flächen nicht erkannt. Damit ist die Definition von Randbedingungen und somit das Simulieren ohne zusätzlichen Aufwand (noch) nicht möglich (siehe Kapitel 3.4.2.6).

Gmsh ⇒ OpenFOAM
Mit einigen für das Ziel dieser Arbeit unrelevanten Einschränkungen (siehe Kapitel 3.4.2.5) hat Gmsh (sowohl API als auch GUI) eine allgemeine Kompatibilität zu OpenFOAM (siehe Kapitel 5.4).

Gmsh ⇒ TAU
Das .cgns Format befindet sich laut der Gmsh Entwickler noch in einem Alphastadium. Der Workflow mit TAU als Solver für ein Netz aus Gmsh konnte bis zu diesem Zeitpunkt nicht getestet werden. Er wird voraussichtlich gesonderten Programmieraufwand benötigen, weil dieses Format die genuine Schnittstelle zu TAU darstellt.

Gmsh Randnotizen
Das aus Gmsh exportierte .cgns Gitter ist im Gegensatz zum SALOME Resultat nicht auf Anhieb in Paraview visualisierbar. Auch das .unv Format von Gmsh lässt sich nicht ohne manuelle Modifikation in einer anderen Software öffnen. Dieses Umstand lässt sich jedoch durch Einfügen einiger weniger Zeilen[8] in die ASCII Datei beheben (erfolgreich getestet).

Tabelle 3.4.2.2-1: Eingangs- und Ausgangsdatenformate von Gmsh und SALOME; kursiv: proprietäre Formate; fett: Überschneidungen

	Eingangsformat (Geometrien)	Ausgangsformat (Netze)
SALOME	**.brep, .step, .iges,** .stl, .xao	*.hdf (SALOME),* **.stl, .dat, .med, .unv, .cgns (Boeing/NASA)**, .sauv, .gmf, Konverter Plugins
Gmsh	**.brep, .step, .iges**	*.msh (Gmsh),* **.stl, .dat, .med, .unv, .cgns (Alphastadium)**, .inp (Abaqus), .vtk, .key, .celum, .diff, .ir3, .mesh, .mail, .m, .bdf, .p3d, .wrl, .ply2, .su2, .neu

[6] CGNS basiert auf HDF 5 und ist prinzipiell kompatibel mit TAU.
[7] UNV kann in OpenFOAM nach Ausführung der gmshToFoam Funktion weiterverwendet werden.
[8] Diese enthalten u. a. ein globales Koordinatensystem.

3.4.2.3 Zeiteffizienz

Die Gitter (erzeugt in der jeweiligen Benutzeroberfläche) in den Abbildungen 3.4-1 und 3.4-2 sind bei Vernetzungszeit und Zellenanzahl nur marginal verschieden. Für die sequenzielle Vernetzung der Beispielgeometrie benötigt Gmsh 131 Sekunden auf einem Intel Core i7-3520M Prozessor mit 3,6 GHz. SALOME braucht auf der gleichen Hardware bei gleichen Bedingungen +10,6% mehr Zeit.
Die Tests zur Vernetzungszeit der Beispielgeometrie in der jeweiligen API ergeben jedoch hohe Zeitdifferenzen zwischen Gmsh und SALOME. Alle Zeitmessungen in den APIs beziehen sich auf einem Intel Xeon E5530 Prozessor mit 2,40 GHz. SALOME bedurfte viele Anläufe, um bei der Netzgenerierung zu konvergieren und ein Ergebnis liefern zu können. Selbst mit sehr hohem Aufwand bleibt die Rechendauer für die Beispielgeometrie in der API um den Faktor zehn höher, als in der GUI. Dieser Anstieg der Laufzeit ist nicht ausschließlich durch die verschiedenen Prozessoren und Taktraten erklärbar. Die Rechenzeit pro Tetraeder steigt nochmals signifikant an, wenn statt der Beispielgeometrie die Passagierflugzeugkonfiguration D150 (siehe Abbildung 5.4-1) in der API vernetzt wird.
Gmsh verursacht unter der API keinen deutlich erhöhten Zeitbedarf im Vergleich zur GUI. Der Aufwand für ein Netz mit ~ 2 Millionen Tetraedern und einfacher Optimierung beträgt ≤ 4 Minuten, unabhängig davon, ob die Beispielgeometrie oder ein Passagierflugzeug vernetzt wird.

Der hohe zeitliche Aufwand in der SALOME API und die Sensitivität gegenüber verschiedenen Geometrien legt den Schluss einer falschen Benutzung der API aufgrund der fehlenden Dokumentation nahe. Im Rahmen dieser Arbeit war es jedoch nicht möglich, den Grund für diese Problematik abschließend aufzudecken und ggf. Programmierarbeit zu leisten.

Bei einer potentiellen Verwendung in einer Optimierungsschleife ist der Zeitaufwand von Gmsh vorhersehbar und vertretbar. Die Dauer der Vernetzung in SALOME ist nicht in jedem Fall planbar und kann eine Optimierungsschleife bei Nichtkonvergenz zum Erliegen bringen.

3.4.2.4 Dokumentation und Support

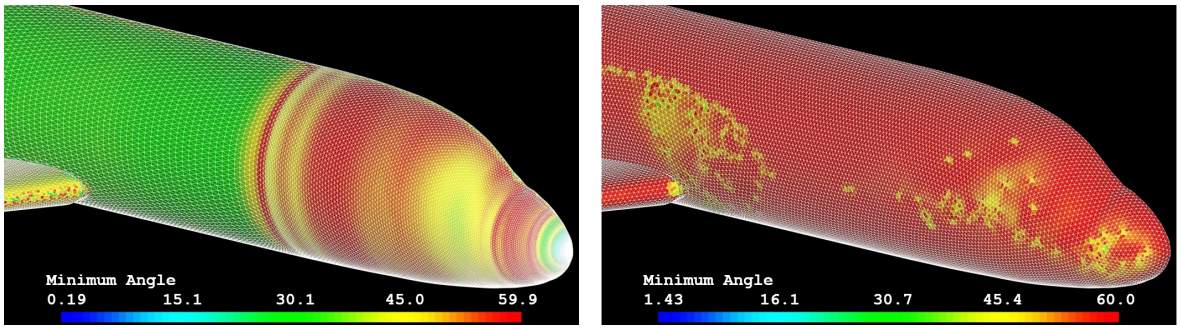
Für die erste Einarbeitung, aber auch für die spätere Implementierung der Software in TiGL bedarf es einer guten Dokumentation und einer aktiven Open Source Community.
Support von SALOME und Gmsh
Beide Programme erfreuen sich einer aktiven Open Source Community in den einschlägigen CFD Foren. SALOME hat darüber hinaus ein eigenes Forum. Im Falle von Gmsh besteht das Forum aus einer sehr empfehlenswerten Emailliste.[9] Die Entwickler und andere erfahrene Benutzer teilen hier täglich Antworten und Lösungen zu gestellten Fragen.
Für beide Plattformen existiert im Internet auch umfangreiches Videomaterial, das den Einstieg in die Bedienung erleichtert. Ebenso existieren zahlreiche Beispiele in Form von Skripten, die in Python bzw. der Gmsh Skriptsprache vorliegen.
GUI von SALOME und Gmsh
In dieser Bewertungskategorie zeigt sich, dass SALOME für die GUI eine vergleichsweise umfangreichere Dokumentation aufweist. Dabei hängt dies auch mit dem größeren Funktionsumfang der Software zusammen. SALOMEs offizielle Dokumentation differenziert zwischen Benutzern der GUI und Entwicklern (Doxygen Dokumentation).
Die Dokumentation von Gmsh enthält eine Unterteilung für die GUI und die Skripte. Hier wird jede Funktion und jede Methode ausführlich mit allen Optionen dargestellt.
API von SALOME und Gmsh
Für die Anwendung in einem Prototypen ist eine vorhandene API inklusive Dokumentation ein

[9] `http://onelab.info/mailman/listinfo/gmsh/`.

wichtiger Punkt. Bei SALOME besteht jedoch die Einschränkung, dass es keine Dokumentation der C++ API für die Vernetzung mit dem NETGEN Plugin gibt (siehe Übersicht der Entwicklungszweige in Kapitel 3.4.2.1). Hier besteht die Dokumentation in weiten Teilen nur aus der Namensgebung der C++ Klassen und ihrer Methoden. Dies birgt Potential für Verwechslungen und Verlust der Übersicht.
Wenngleich Gmsh insgesamt minimalistischer aufgebaut ist und bei der GUI Dokumentation einen geringeren Umfang hat / bedarf, so hebt es sich bei der API und deren Dokumentation von den SALOME Entwicklungszweigen ab: Es hat seit Mai 2018 eine neue API für C, C++, Python und Julia. Die grundlegenden Methoden und Klassen sind sehr ausführlich dokumentiert. Bei allen bisherigen Untersuchungen traten hierbei im Gegensatz zu SALOME jedoch keine Diskrepanzen zwischen API und GUI auf. Da Gmsh nur zwei aktive Entwickler hat und minimalistisch aufgebaut ist, hat es im Laufe der Zeit eine gute Übersichtlichkeit und Struktur beibehalten.
Der Einstieg gelingt in beiden Fällen mit den vorhandenen Tutorials und Dokumentationen erfolgreich. Für die Einbindung einer API in TiGL erweist sich Gmsh jedoch vorteilhafter.

3.4.2.5 Verfügbare Algorithmen

Die Algorithmen für triangulierte Vernetzung werden im Folgenden beleuchtet. Bei SALOME existieren verschiedene Entwicklungszweige mit je einer API für Python und C++. Die Pythonvariante stellt keine Anbindug an NETGEN bereit. Dieser Entwicklungszweig unterstützt ausschließlich MEFISTO für die Triangulierung der Oberflächen. Die Abbildung 3.4.2.5-1 demonstriert anhand der Vernetzung eines Flugzeugs die resultierende Elementqualität von NETGEN und MEFISTO. NETGEN generiert im Gegensatz zu MEFISTO größtenteils gleiseitige Dreicke, auch an der Rumpfspitze des Flugzeugs, wo MEFISTO eine erhebliche Anzahl spitzwinkliger Elemente einführt. Dagegen sind suboptimale Oberflächenelemente bei NETGEN die Ausnahme.



(a) MEFISTO: Die Minderheit der Elemente sind perfekte gleichseitige Dreiecke. Kleinster Winkel (zahlreiche Vorkommen) ist 0,19°.

(b) NETGEN (siehe 2.4.1): Die Mehrheit der Elemente sind perfekte gleichseitige Dreiecke. Kleinster Winkel (Einzelfall) ist 1,43°.

Abbildung 3.4.2.5-1: Oberflächennetze mit verschiedenen SALOME Algorithmen:Gleichseitige Dreiecke: rot, Elenente mit spitzen Winkeln: blau

Aufgrund der fehlenden Funktionalität für Volumenvernetzung bei MEFISTO ist die C++ API eine geeignete Ausweichmöglichkeit, um den NETGEN Algorithmus für Oberflächen und Volumina anzuwenden. In der API Variante von SALOME ist eine spezielle angepasste Version (4.9.13 mit einem Patch) des NETGEN Algorithmus' eingebunden, in der Benutzeroberfläche eine neuere Variante (5.3.1). Bei vermeintlich identisch eingestellten Optionen fallen die Gitter verschieden aus. Um den Ursprung dieses Phänomens aufzudecken, fehlt jedoch ein Vergleich mit einer aktualisierten Version von NETGEN in der API. Der hierfür notwendige Aufwand konnte im Rahmen dieser Arbeit nicht erbracht werden. Die Funktionalität der C++ API erstreckt sich nicht über alle Möglichkeiten der GUI und ist zudem nicht dokumentiert.

Für die Oberflächengitter stellt Gmsh drei bewährte und robuste, unstrukturierte Algorithmen für die Triangulierung zur Verfügung:

Tabelle 3.4.2.5-1: Charakterisierung / Rangordnung der 2D Algorithmen in Gmsh [15]: Erklärung: Mesh Adapt ist bspw. robuster als Delaunay. Delaunay ist wiederum robuster als Frontal.

Algorithmus	Robustheit	Zeiteffizienz	Elementqualität
MeshAdapt [16]	1	3	2
Delaunay [11]	2	1	2
Frontal [29]	3	2	1

Über diese Auflistung hinaus existiert eine vierte Option: Im „Automatic“ Modus werden plane Flächen mit Delaunay vernetzt, alle anderen mit MeshAdapt.
Für die Volumenvernetzung steht eine größere Anzahl von Algorithmen für die Triangulierung zur Verfügung. Jedoch sind einige von ihnen noch experimenteller Natur und werden hier nicht namentlich erwähnt. Zu den robusten Vertretern zählen (Gmsh-interne Namensgebung hervorgehoben):

- 3D Adapted **Delaunay** mit initialem Netz basierend auf dem TetGen Algorithmus [38] von Hang Si mit anschließender Anwendung eines angepassten 2D Delaunay Algorithmus'.
- **New Delaunay** ohne das oben genannte initale Netz
- **Frontal** Algorithmus alias NETGEN [33] von Joachim Schoeberl

Innerhalb dieser drei Methoden für die Volumenvernetzung sind die Delaunay Algorithmen bei der Robustheit und der Zeiteffizienz dem NETGEN Algorithmus überlegen. Gewisse Kombinationen von den hier *nicht* genannten 2D und 3D Gmsh Algorithmen können unter Umständen zu nicht näher definierten Fehlern bei der Vernetzung selbst oder beim Import in OpenFOAM führen [21]. Diese potentiell fehlerhaften Kombinationen werden in dieser Arbeit nicht weiter berücksichtigt, da es 12 Verbindungen von 2D und 3D Algorithmen gibt, deren Ergebnis OpenFOAM importieren kann.

3.4.2.6 Steuerung der Netzeigenschaften

Die Steuerung der Netzeigenschaften erfordert bei SALOME individuelle Eingaben[10], je nachdem, welcher Algorithmus bzw. welche Diskretisierung im 1D, 2D, 3D angewendet wird. Da bei SALOME im 2D und 3D Bereich jedoch nur NETGEN in Frage kommt, würde dieser Umstand eine Parametrisierung nicht erschweren. (Es braucht keine Abstrahierung der Eingabeparameter, damit diese z.B. auf NETGEN *und* auf MEFISTO passen.)
Gmsh bietet dem Benutzer bei der Diskretisierung im 1D Bereich keine Auswahl. Für die Flächen und Volumina stehen bei Gmsh verschiedene valide Kombinationen von Algorithmen bereit, die jedoch (fast) alle mit den gleichen Parametern angesprochen werden.[11] Dies ist hilfreich für eine allgemeingültige, parametrisierte Vernetzung beliebiger Flugzeuge. Es impliziert jedoch auch, dass die Entwickler von Gmsh eine Abstrahierung der Eingabewerte vorgenommen haben, damit die eingegebenen Werte später zu der internen, individuellen Arbeitsweise eines jeden Algorithmus' passen.
In Gmsh gibt es anders als bei SALOME keine explizite Steuerung der Zellenwachstumsrate.

[10] Ein Beispiel mit NETGEN im 2D und 3D:
1D: Abstand der Punkte für Diskretisierung der Linien und die Genauigkeit bei der Ausführung
2D: Minimale und maximale Kantenlänge der Dreiecke, Wachstumsrate, optional: lokale Verfeinerung
3D: Minimale und maximale Kantenlänge der Tetraeder, Wachstumsrate, optional: lokale Verfeinerung

[11] Eine Ausnahme bildet der 3D Frontal / NETGEN Algorithmus. Er ist bei der Netzdefinition für einige der sogenannten „fields“ nicht sensitiv [15].

Trotzdem kann ein kontrolliertes Zellenwachstum ohne Sprünge erreicht werden (siehe Abbildung 3.4-1 und 3.4-2).
Eine feine Vernetzung des Volumens nahe einer Oberfläche wird bei Gmsh nur fehlerfrei ausgeführt, wenn diese Feinheit sich für ≥ 10 Zellenschichten fortsetzt (siehe Abbildung 3.4-1) und die Vergröberung nicht direkt beginnt. Dies kann unter Umständen den späteren Rechenaufwand erhöhen, ohne dass sich die Ergebnisgüte entsprechend verbessert.

Nach derzeitigem Kenntnisstand existiert nur in der SALOME GUI die Option, das Netz auf gewünschten Flächen mit graduell wachsenden Prismenschichten aufzubauen (siehe Abbildung 3.4.2.6-1). Diese Möglichkeit erhöht die potentielle Berechnungsgenauigkeit und schafft Zeitgewinn bei der CFD Simulation durch reduzierte Zellenanzahl in Richtung der niedrigen Gradienten. Eine Prismenschicht konnte im Zuge der Untersuchungen in der SALOME API auch mit erhöhtem Aufwand nicht generiert werden.

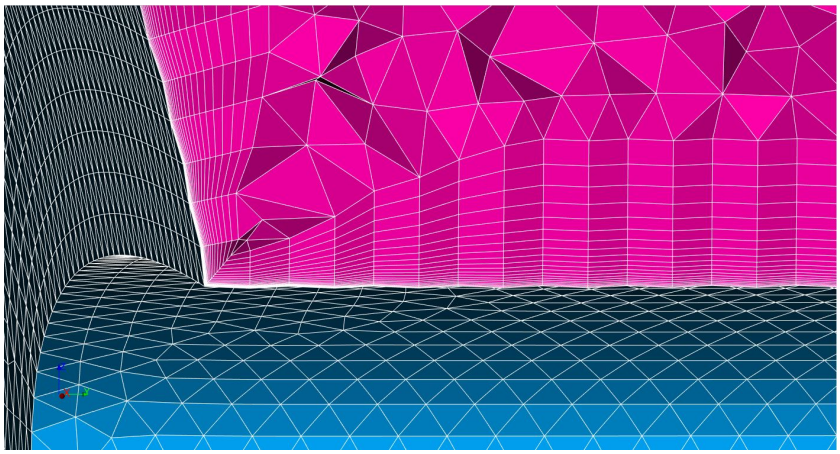


Abbildung 3.4.2.6-1: Vernetzung der Randschichten mit SALOME: Schnitt durch vernetztes Geometriebeispiel: 20 Prismenschichten formen sich zu einer Randschicht mit einer Dicke bis zu 0.03m, Wachstumsfaktor der Zellen: 120%

Gmsh liefert keine so zahlreichen Funktionalitäten zur Prismengenerierung in seiner GUI oder API. Die existierenden Funktionen für Extrusion von Prismen aus einem Oberflächennetz bedürfen in Gmsh tiefergehende Untersuchungen, um sie allgemeingültig für eine parametrisierte, automatische Vernetzung nutzbar zu machen. Diese Problemstellung kann gut in einer separaten weiterführenden Arbeit behandelt werden.

Anders verhält es sich mit der Erstellung von untergeordneten Netzen und der Gruppierung dieser innerhalb des Netzes. Diese sogenannten „submeshes“ sind zwingend notwendig, um später in der Simulation je nach Fläche verschiedene Randbedingungen zu setzen. Die untergeordneten Oberflächennetze werden dann z. B. als „inlet“ / „outlet“ der Strömung oder als reibungsbehaftete Oberflächen definiert. Diese „submeshes“ konnten im Rahmen dieser Arbeit nur in der SALOME GUI erstellt werden, nicht jedoch in der API. Diese Tatsache schließt die geplante automatisierte Erstellung von Gittern aus. Ein SALOME Gitter aus der GUI lässt sich außerdem nicht ohne zusätzlichen Aufwand korrekt in TAU berechnen, weil die generierten „submeshes“ in TAU nicht ansprechbar sind. Die Lösung wäre ein Konverter, für den es in MATLAB fertige Funktionen aus der NetCDF Bibliothek gibt. Dieser Ansatz wurde nicht weiter verfolgt, weil in der SALOME API noch keine „submeshes“ erzeugt werden konnten.

In der Gmsh API hingegen ist die Erstellung solcher Oberflächennetze und die Adressierung dieser durch OpenFOAM grundsätzlich gegeben. In einer Simulation mit OpenFOAM konnte dies getestet werden (siehe Kapitel 5.4). Inwieweit auch TAU in der Lage ist, ein Gmsh Gitter mit Randbedingungen zu belegen, ist noch zu untersuchen.

3.4.2.7 Informationen zur Gitterqualität

Im Prozess der Parametrisierung und Automatisierung der Vernetzung ist immer wieder eine Überprüfung der Zellenqualität notwendig. Der Anforderungskatalog umfasst darum den Punkt einer Quantifizierung der Güte in Form von physikalischen Größen (z.B. Kantenlängen, Flächen, Volumina, Winkel). SALOME bietet zu diesem Zweck dutzende passende Möglichkeiten an, die entweder zahlenmäßig, in einem Graphen oder farblich / visuell einen Eindruck von der Netzqualität vermitteln.[12]
Gmsh kann die Gitterqualität intern nur mit drei verschiedenen einheitenlosen Größen beschreiben. Durch eine leichte Modifikation des Schreibprozesses beim .unv Format in Gmsh können die Netze in SALOME importiert und auf ihre Qualität hin untersucht werden.

3.5 Zusammenfassung

Von allen Open Source Softwareprodukten passt die Gmsh API nach diesem Vergleich am besten zu den Anforderungen in der Auflistung aus Kapitel 3.2. Der Vergleich zwischen der Gmsh API und der SALOME API fand unter der Voraussetzung statt, dass kein zusätzlicher Programmieraufwand für eine evtl. notwendige Modifikation einer API geleistet werden kann. Die Unterschiede zwischen Gmsh und SALOME sind insbesondere bei essentiellen Eigenschaften deutlich. Gmsh ist nicht frei von Nachteilen, zeigt sich jedoch nach bisherigen Untersuchungen als Favorit (siehe Tabelle 3.5-1) für den weiteren Verlauf dieser Arbeit und die Einbindung in TiGL. Vor allem bildet Gmsh ein Basis, auf der selbst jetzt noch vorhandene Kritikpunkte in Zukunft mit absehbarem Aufwand ausgebessert werden können.

Tabelle 3.5-1: Zusammenfassender Vergleich zwischen Gmsh und SALOME

	Gmsh	SALOME
Nutzbarkeit der API	++	−−
Datenformate / Kompatibilität	o	−
Zeiteffizienz	+	−−
Robustheit bei verschiedenen Geometrien	++	−−
Dokumentation (der API) / Support	++	o
Verfügbare Algorithmen	++	o
Intuitive Vernetzungseinstellungen	o	++
Erstellung Prismenschichten	−	++
Erstellung von Gruppen / „submeshes“	++	−−
Information zur Gitterqualität	o	++

[12] Beispiele hierfür sind die Abbildungen 3.4.2.5-1a und 3.4.2.5-1b. Die Plots in den Abbildungen 3.4.1-1 und 3.4.1-1 sind ebenfalls mit SALOME erstellt, enthalten allerdings keine SI Einheiten.

4 Parametrische, automatisierte Vernetzung

In diesem Kapitel geht es um die konkrete Umsetzung in Form des TiGL Meshers. Dieser ist ein Software Prototyp und greift auf die APIs von Gmsh und Open CASCADE zurück. Gmsh nutzt intern zwar ebenfalls Open CASCADE Klassen, diese sind aber nicht in der Gmsh API ansprechbar. Für die automatisierte Parameterfindung in diesem Prototyp sind jedoch Open CASCADE Funktionalitäten zur Bestimmung geometrischer Größen notwendig.

Nun folgt eine Beschreibung der Gmsh API, der Vernetzungsparameter, des Programmablaufs, der automatischen Parameterberechnung und der verwendeten API Funktionen für die parametrische Netzgenerierung. Parallel fließen erprobte Best Practice Einstellungen für die Automatisierung ein. Zusätzlich werden nützliche Nebenfunktionen des Prototyps gezeigt.

4.1 Grundsätzlicher Aufbau der Gmsh API

Die Gmsh API wird in diesem Fall in C++ angewendet. Es existieren aber auch Versionen in C, Python und Julia. Die Programmierschnittstelle umfasst alle Funktionalitäten der Standardbibliothek von Gmsh. Die API ist in folgende Module gegliedert [14]:

Tabelle 4.1-1: Module der Gmsh API, die insgesamt 189 Methoden umfassen

Modul	Funktion
`module::gmsh`	Grundlegende Funktionen: z.B. Dateiimport, Dateiexport, Initialisierung der API
`module::gmsh::option`	Eingabe und Rückgabe von Parametern
`module::gmsh::model`	Verwaltung eines bzw. mehrerer geometrischer Modelle und der 1D- / 2D- / 3D-Geometriegruppen, Namensgebung
`module::gmsh::model::mesh`	Netzgenerierung, Ableitung eines Netzes von spezifischen Geometriegruppen, Adressierung einzelner Netzelemente
`module::gmsh::model::mesh::field`	Netzverfeinerung mittels „mesh size fields“
`module::gmsh::model::geo`	Internes GEO CAD: Erzeugung von Punkten, Linien, Flächen, Extrusionen, etc.
`module::gmsh::model::geo::mesh`	GEO-spezifische Vernetzungsbedingungen
`module::gmsh::model::occ`	Internes Open CASCADE CAD: deutlich umfangreicher, als internes GEO CAD
`module::gmsh::view`	Visualisierung im Post-Processing
`module::gmsh::plugin`	Adressierung von Plugins (CAE Software)
`module::gmsh::graphics`	Grafische Anzeige
`module::gmsh::fltk`	Initialisierung der FLTK-basierten GUI
`module::gmsh::onelab`	Anbindung an die FEM-Software ONELAB
`module::gmsh::logger`	Protokollierung der Ausgabe

4.2 Übersicht der Vernetzungsparameter

Tabelle 4.2-1: Übersicht der Vernetzungsparameter im TiGL-Mesher-Prototyp

Parameter [Einheit]	Inhalt / Berechnung	Art der Definition
format	msh = Gmsh Format unv = CAE Format stl = STL Format	man.
dim	2 = Oberflächennetz 3 = Volumennetz	man.
automatedSetup	0 = deaktiviert 1 = aktiviert	man.
Dim_Tags	IDs Flugzeugoberflächen	autom.
l_x, l_y, l_z, $Lmax_{xyz}$ [m]	Flugzeugabmessungen	autom.
lc [ ]	Gleichung 4.4-1 oder $\mathbb{R}^{>0}$	autom. / man.
LcMin [ ]	$LcMin = lc$	autom.
LcMax [ ]	$LcMax = (lc \cdot 200)^{1.2}$	autom.
DistMin [m]	$DistMin = lc \cdot 5$ oder $\mathbb{R}^{>0}$	autom. / man.
DistMax [m]	$DistMax = lc \cdot 150$	autom.
algo2D	1 = Mesh Adapt 2 = Automatic 5 = Delaunay 6 = Frontal	autom. / man.
algo3D	1 = Delaunay 2 = New-Delaunay 4 = Frontal (NETGEN)	opt. / autom. / man.
optimizeGmsh	0 = deaktiviert 1 = aktiviert	opt. / autom. / man.
optimizeNETGEN	0 = deaktiviert 1 = aktiviert	opt. / autom. / man.
lcFromCurvature	0 = deaktiviert 1 = aktiviert	prog.
Sigmoid	0 = deaktiviert 1 = aktiviert	prog.
Div_{lin}	Divisor aus $\mathbb{R}^{>0}$	prog.

Diese Tabelle gibt alle Parametereinstellungen wieder, die derzeit im TiGL Mesher möglich sind. In der Spalte „Art der Definition“ der Tabelle 4.2-1 lässt sich ablesen, auf welche Weise sich die Parameter ändern lassen / geändert werden. Für Parameter mit der Kennzeichnung „man.“ ist eine Eingabe des Benutzers im Terminal notwendig (siehe Abbildung A.1-2). Daneben gibt es „autom.“-Parameter, die stets oder zumindest im Automatikmodus von der Software übernommen werden. Wahlweise werden im Falle einer Volumenvernetzung die „opt.“-Parameter wichtig. Die letzten drei „prog.“-Parameter sind im aktuellen Prototyp fest einprogrammiert, weil sie bisher nur selten verändert werden mussten. Sie können im Initialisierungsbereich am Kopf des Hauptprogramms deklariert werden. Die ganzen Zahlen bei *algo2D* und *algo3D* sind die internen Gmsh Kürzel für die Algorithmen. Der Zahlenwert von *lc* wird innerhalb des TiGL Meshers grundsätzlich mit dem Parameter *LcMin* gleichbedeutend verwendet. Dunkelblau und unterstrichen sind die Voreinstellungen bestimmter Parameter im Automatikmodus. Weitere Ausführungen zu allen Parametern (*kursiv* gekennzeichnet) befinden sich in den folgenden Unterkapiteln.

4.3 Programmablauf des Prototyps

Grundsätzlich ist die Vorgehensweise des Hauptprogramms wie folgt (siehe Anhang A.1):

- Eingaben des Benutzers
- Einstellung der Vernetzungsparameter (automatisch oder anhand der Eingaben)
- Vernetzung
- Datenexport

Zu Beginn wird in der Konsole beim Nutzer abgefragt in welches Format das Gitter später exportiert werden soll (*format*). Darauffolgend wird festgelegt, ob das Netz nur die Oberflächen oder auch das Volumen der Geometrie abbilden soll (*dim*). Im nächsten Schritt bekommt der Anwender die Möglichkeit, automatisch errechnete Vernetzungsparameter zu verwenden (*automatedSetup*). Sollen stattdessen selbstgewählte Zahlenwerte Anwendung finden, wird dem Benutzer als Hilfestellung trotzdem der automatisch errechnete Wert für *lc* ausgegeben. Anschließend erwartet die Software Eingaben zu den Parametern *lc* und *DistMin*. Mit Hilfe von *lc* werden intern zwei weitere Parameter (*DistMax* und *LcMax*) festgelegt, die der Einfachheit halber niemals frei einstellbar sind. Im manuellen Modus werden auch die Algorithmen für die Vernetzung der Oberflächen (*algo2D*) und ggf. Volumina (*algo3D*) ausgesucht (verfügbare Möglichkeiten siehe Kapitel 3.4.2.5). Ist ein Volumennetz gewünscht, stehen an dieser Stelle je ein Optimierer von Gmsh[1] (*optimizeGmsh*) und / oder NETGEN[2] (*optimizeNETGEN*) zur Verfügung, die die Tetraeder nach der Netzgenerierung in einen gleichmäßigeren Zustand führen. Die letzte Benutzereingabe betrifft die Namensgebung der später entstehenden Datei. Das Hauptprogramm greift bei diesen Vorgängen auf „gmsh.h“ und auf eine eigenständig entwickelte Klasse zu, die in „TiGL_Mesher_Tools.h“ definiert ist und nach aktuellem Stand 23 Methoden beinhaltet. Diese sind in die vier Kategorien „Hintergrundaufgaben“, „Parametrisierung“, „Zeitmessung“ und „Schreibprozesse“ gegliedert. Desweiteren verwendet „TiGL_Mesher_Tools.h“ seinerseits acht Klassen aus Open CASCADE.

4.4 Automatische Parameterberechnung und -einstellung

Die Vernetzungsparameter sind im Softwareprototyp alle in Abhängigkeit von l_c gestellt. Es ist eine einheitenlose Zahl, die die Zielkantenlänge der Elemente charakterisiert [16]. Bei konstant gesetztem *lc* können sich je nach gewähltem Vernetzungsalgorithmus und Geometrieskalierung stark voneinander abweichende Gitter ergeben. Ein Wechsel von der NETGEN- zur Delaunay-Methode hat beispielsweise zur Folge, dass die Anzahl der generierten Zellen bei sonst identischen Parametern um den Faktor 2 ansteigen kann (siehe Tabelle 5.6-2). Die charakteristische Länge wird im Softwareprototyp für jede Geometrie errechnet, um dem Benutzer eine Orientierung zu bieten, selbst wenn dieser selbstgewählte Werte für *lc* und *DistMin* verwenden möchte. Das Ziel der automatischen Berechnung von *lc* liegt darin, für gänzlich verschiedene Größenskalen einer Geometrie trotzdem gleichwertige Netze erzeugen zu können (siehe Abbildung 5.7-3). Die relative Größe der Zellen und ihre Verteilung müssen dafür einander möglichst nahe kommen (siehe auch Abbildungen 5.7-2 und 5.7-1). Dies stellt eine Herausforderung dar, weil *lc* nicht linear von der Skalierung einer Geometrie abhängt. Ein zweites Ziel liegt darin, dem Anwender ein „schnelles“ erstes Ergebnis bei der Vernetzung seiner Geometrie zu liefern. Gmsh ist in der Lage, in einer Zeit von $\sim$ 2 Minuten (inkl. der einfachen Optimierung) ein Gitter aus $\sim$ 1 Million Tetraedern zu generieren (siehe Tabelle 5.5-2). Im

[1] Der Gmsh Optimierer vollzieht „edge swapping“ und „vertex re-positioning“ [16] (siehe Kapitel 2.4.3).

[2] Der NETGEN Optimierer vollzieht „edge swapping“, „vertex re-positioning“ und „point collapsing“ [33](siehe Kapitel 2.4.1.2).

Verlauf der Entwicklung wurde ein Netz dieser Größenordnung und dieses Zeitaufwandes als Ziel der automatischen Parameterberechnung angesetzt.

Als Grundlage der Berechnung von *lc* wird mittels der Open CASCADE Klassen automatisiert die maximale Ausdehnung der Flugzeuggeometrie [m] entlang der drei Koordinatenachsen bestimmt (l_x, l_y, l_z). Von diesen drei Größen ist $Lmax_{xyz}$ das Maximum. Für die automatische Bestimmung der physikalischen Längen gilt beim TiGL Mesher die Voraussetzung, dass die Flugzeuggeometrie von einem quaderförmigen Raum (Fernfeld) umgeben ist. Das Flugzeug ist dabei an der Symmetrieebene halbiert und liegt als Negativabdruck im Quader vor. Im nächsten Schritt wird mit diesen vier Längenangaben und einer eigenständig heuristisch entwickelten Formel der gesuchte Wert für *lc* bestimmt:

$$lc = \ln\left(\ln\left(\frac{\sqrt{2 \cdot Lmax_{xyz} \cdot \frac{1}{\mathrm{m}}}}{5} + 3\right) \cdot \sqrt{\left(\sum_{i=x}^{z} l_i\right) \cdot \frac{1}{\mathrm{m}}}\right) \cdot \frac{\sqrt{\left(\sum_{i=x}^{z} l_i\right) \cdot \frac{1}{\mathrm{m}}}}{Div_{lin}} \tag{4.4-1}$$

Erklärung der Gleichung:
Der innere Logarithmus ergibt für alle in Frage kommenden Geometrien ($0,001\mathrm{m} \leq Lmax_{xyz} \leq 200\mathrm{m}$) für die darauffolgende Wurzel einen Faktor zwischen 1,1 und 2,0. Dieser Faktor dient dazu, *lc* grundsätzlich anzuheben, wobei der Zuwachs für vergleichsweise kleine Geometrien prozentual absichtlich höher ausfällt. Dieser Ausdruck kann entsprechend genutzt werden, um die Entwicklung des *lc* bei verschiedenartigen Geometrien weiter zu optimieren. Der Faktor $\sqrt{\left(\sum_{i=x}^{z} l_i\right) \cdot \frac{1}{\mathrm{m}}}$ glättet den Einfluss der Größe eines Objekts. Aber auch Flugzeugformen, deren physikalische Ausdehnung in eine Dimension verhältnismäßig hoch ist, beeinflussen das *lc* mit dieser Maßnahme nicht überproportional. Es bietet sich an, die Anzahl der generierten Zellen annähernd linear (siehe Tabelle 5.7-1) und unabhängig von physikalischen Längenmaßen über den Divisor Div_{lin} zu steuern, falls die automatische Parametergewinnung grundsätzlich gröbere oder feinere Netze herbeiführen soll. Für Div_{lin} befinden sich erprobte, sinnvolle Werte im Intervall [50, 250]. In der aktuellen Version des TiGL Meshers ist $Div_{lin} = 150$.
Mit Hilfe der Gleichung 4.4-1 kann nun für die meisten getesteten Geometrien ein angemessener Wert für *lc* bestimmt werden, bei dessen Benutzung in der gewünschten Zeit ein Netz mit ≤ 1 Millionen Tetraedern entsteht. Eine Ausnahme bilden vergleichsweise kleine Geometrien, die ohne Gegenmaßnahme wegen des Logarithmus' zu einem unerwünschten, negativen Ergebnis der Formel führen würden. Fälle, in denen $\sqrt{\left(\sum_{i=x}^{z} l_i\right) \cdot \frac{1}{\mathrm{m}}} \leq 1,2$ ist, werden von der Software darum aufgefangen. Eine künstliche Erhöhung der Summe um 1m ermöglicht anschließend für jegliche Ausmaße des zu vernetzenden Objekts eine Gittererzeugung. Unter diesen Umständen kann eine manuelle Einstellung des *lc* das gewünschte Ergebnis jedoch besser treffen.

Das berechnete *lc* wird im Anschluss verwendet, um drei weitere Vernetzungsparameter zu bestimmen. Dabei handelt es sich um $DistMin$, $DistMax$ und $LcMax$. $DistMin$ ist der Mindestabstand [m] von einer verfeinerten Oberfläche, in dem sich die Feinheit der Dreiecke auch bei den Tetraedern fortsetzt. $DistMax$ ist die Entfernung [m] von einer verfeinerten Oberfläche, ab der die Referenz-Elementgröße nicht mehr von *lc*, sondern von $LcMax$ bestimmt wird. $LcMax$ ist ebenso wie *lc* einheitenlos und charakterisiert die Kantenlängen der Elemente. Im Automatikmodus wird $DistMin = lc \cdot 5\mathrm{m}$, $DistMax = lc \cdot 150\mathrm{m}$ und $LcMax = (lc \cdot 200)^{1.2}$ gesetzt. $LcMax$ hat neben einem Faktor auch eine Potenz, damit verschieden skalierte Geometrien nach der Vernetzung ähnlichere Zellenanzahlen aufweisen.
Diese Werte haben sich in zahlreichen Tests bewährt, sind natürlich je nach künftigen Anforderungen aber entsprechend anzupassen. Der Einfachheit halber werden beim Prototyp selbst im manuellen Modus die automatisch berechneten Zahlen für $DistMax$ und $LcMax$ verwendet.

Neben diesen zu berechnenden Parametern erfolgt im Automatikmodus auch die Einstellung für die Algorithmen und Optimierer ohne eine Benutzereingabe. Für Oberflächen ($algo2D = 5$) ist die Delaunay Methode voreingestellt. Für Volumina ($algo3D = 2$) kommt zusätzlich der sog. New-Delaunay Algorithmus (siehe Kapitel 3.4.2.5) zum Einsatz, dessen resultierende Tetraeder zum Schluss mit dem einfachen Gmsh Optimierer ($optimizeGmsh = 1$) überarbeitet werden. Diese Einstellungen beruhen auf der Begründung, dass die Delaunay Methode im Vergleich zum Frontal Ansatz robustere Eigenschaften und vielseitigere Einstellungsmöglichkeiten hat [15]. Zusätzlich generiert Delaunay mehr Tetraeder pro Sekunde (in einem Beispielfall 16.900 Tetraeder/s zu 10.800 Tetraeder/s). Bei den Optimierern bekommt die Gmsh Variante aufgrund ihrer Effizienz (siehe Kapitel 5.6) den Vorzug ($optimizeNETGEN = 0$).
Mit diesen Berechnungen und Einstellungen bleibt dem Benutzer für eine automatisierte Diskretisierung nur zu definieren, in welches Format (*format*) das Gitter geschrieben wird und ob eine Oberfläche oder ein Volumen (*dim*) zu vernetzen ist (siehe Abbildung A.1-1).

4.5 Verwendete Gmsh API Methoden für Gittergenerierung

Für die bisher berechneten, eingegebenen oder programmierten Parameter erfolgt das Einsetzen in einige der zahlreichen Methoden von „gmsh.h“.

Vor Beginn des Prozesses sammelt der Prototyp mit `gmsh::model::getEntities` automatisch die IDs aller Flächen der Flugzeuggeometrie. Diese werden anschließend dem Vektor bzw. dem Parameter *Dim_Tags* zugewiesen. Auf diese Weise entsteht die Grundlage für die Geometriegruppen und die Voraussetzung für gezielte Verfeinerung bestimmter Oberflächen und deren angrenzenden Volumina.

Für die spätere CFD Simulation sind Gruppierungen der Geometrieoberflächen und entsprechende Sub-Netze zwingend erforderlich. Nur so können die Symmetrieebene, der Einlass, der Auslass, die Flugzeuggeometrie und die übrigen drei Flächen des Fernfeldquaders für die Simulation mit unterschiedlichen Randbedingungen versetzt werden. Die Erstellung und Benennung der Geometriegruppen geschieht mit `gmsh::model::addPhysicalGroup` und `gmsh::model::setPhysicalName`.

Im weiteren Verlauf des Programms wird auf die „FacesList“ der Flugzeuggeometrie-Gruppe durch `gmsh::model::mesh::field::add` ein „Attractor“ gesetzt. Dieser erhält wiederum mit `gmsh::model::mesh::field::add` einen „Threshold“.

Anschließend erhält der TiGL Mesher die Zahlenwerte für die Parameter *LcMin*, *LcMax* *DistMin* und *DistMax* durch `gmsh::model::mesh::field::setNumber`.

Optional kann an diesem Punkt im „Threshold“ die Interpolationsmethode für das Elementwachstum zwischen *LcMin* und *LcMax* bestimmt werden. Im Prototyp ist für das Zellenwachstum standardmäßig kein linearer Zusammenhang, sondern eine Sigmoidfunktion ($Sigmoid = 1$) festgelegt (Auswirkungen dieser Einstellung siehe Kapitel 5.3).

Bei den Algorithmen (2D / 3D) und den Optimierern (Gmsh / NETGEN) findet die Übergabe der entsprechenden Parameter mit `gmsh::option::setNumber` statt.

Auf die gleiche Weise kann an dieser Stelle der Parameter *lcFromCurvature* aktiviert werden, damit zusätzlich zu *lc* auch die Krümmung einer Oberfläche Einfluss auf die Elementkantenlänge nimmt (siehe Kapitel 5.2). Existieren für ein Element mehrere zu erfüllende Netzfeinheiten, wird in Gmsh stets die höhere Feinheit gewählt.

Für eine erfolgreiche Vernetzung benötigt Gmsh einen Wert für das „Hintergrundnetz“, der sich aus dem „Threshold“ ableitet (`gmsh::model::mesh::field::setAsBackgroundMesh`).

Damit OpenFOAM .msh Dateien problemlos lesen kann, muss die „Mesh.MshFileVersion“ durch `gmsh::option::setNumber` auf den Wert 2 gesetzt werden.

Der eigentliche Netzgenerierung passiert beim Aufruf von `gmsh::model::mesh::generate`, wobei der Übergabeparameter die Dimension des gewünschten Netzes (2D oder 3D) ist.

Zuletzt soll die `gmsh::write` Funktion erwähnt werden, die das generierte Netz in eines der drei Dateiformate überführt, die im Prototyp freigegeben sind. Dabei wird die Endung des zu übergebenden Strings automatisch von der Methode erkannt.

4.6 Nützliche Nebenfunktionen des Prototyps

Für eine erleichterte Bedienung des Prototyps existiert in „TiGL_Mesher_Tools.h“ eine Methode, die die physikalischen Längenausdehnungen der Flugzeuggeometrie erkennt und ausgibt. Die Ausmaße des Fernfelds bzw. des Quaders sind ein Nebenprodukt und werden ebenfalls angezeigt. (Der Prototyp setzt voraus, dass die Flugzeuggeometrie an ihrer Symmetrieebene halbiert und als Negativabdruck in einem Quader vorliegt.)

Unter den TiGL Mesher Werkzeugen befindet sich auch die Möglichkeit, die Zeit der eigentlichen Vernetzung inklusive der optionalen Optimierung zu messen und diese auszugeben. Schreibvorgänge sind hierbei bewusst nicht berücksichtigt.

In dem Fall, dass .unv als Endung ausgewählt ist, kann das in Gmsh generierte Netz zunächst nicht in SALOME visualisiert werden. Um diesen Umstand zu beheben, fügt eine Funktion u. a. ein globales Koordinatensystem in die ASCII-Datei ein. Dies funktioniert in diesem Stadium des Prototyps allerdings nur mit einer Kopie der Datei und zusätzlichem Zeitaufwand.

Für den Dateinamen, unter dem das Gitter abgespeichert werden soll, setzt ein Namengenerator folgende Bausteine zusammen:

- Datumstempel und Zeitstempel des Vernetzungsbeginns
- *lc* (manuell oder automatisch definiert)
- *DistMin* (manuell oder automatisch definiert)
- Kürzel für den 2D Algorithmus
- Kürzel für den 3D Algorithmus (sofern Volumennetz)
- Boolescher Ausdruck für Gmsh Optimierer (sofern Volumennetz)
- Boolescher Ausdruck für NETGEN Optimierer (sofern Volumennetz)
- Netzbeschreibung des Benutzers
- Dateiendung entsprechend der Benutzereingabe zu Beginn

Auf diese Weise bleibt grade bei zahlreichen erstellten Gittern der Überblick über die vorgenommenen Netzparameter erhalten. Ein Netzresultat könnte entsprechend bspw. folgenden Dateinamen erhalten:
2018-11-31_12-30-45_lc_0.1_DistMin_0.7_Algo2D_5_Algo3D_2_GmshOpt_1_NetgenOpt_0_TestV1.unv

4.7 Zusammenfassung

In diesem Kapitel wurde die Grundlage für spätere Weiterentwicklung des Prototyps gelegt. Schon jetzt ist er in der Lage, die Geometrien aus TiGL zu analysieren und mit einfachen Mitteln parametrisch Netze aus Oberflächen und Volumina zu erzeugen. Zu diesem Zweck existiert ein vollautomatischer Ansatz für die Parameterbestimmung, die auf einer eigenständig heuristisch entwickelten Formel basiert. Entspricht das generierte Netz damit noch nicht den Ansprüchen, gibt es aber auch die Möglichkeit, einen Teil der Parameter eigenhändig einzupflegen. Die Auswahl von verschiedenen Algorithmen eröffnet dem Benutzer die Freiheit, zwischen Elementqualität, Laufzeit und Robustheit Prioritäten zu setzen. Bei Bedarf können zusätzlich auch Optimierer zugeschaltet werden.

Die Möglichkeiten der Gmsh API sind allerdings noch nicht ausgeschöpft, sodass Potential besteht, gezielter zu verfeinern (z. B. an den Anströmkanten der Flügel), weitere Parameter einzuführen und den Automatikmodus zu optimieren.
Für die Verfeinerung bieten sich u. a. automatisch generierte Quader an, die jeweils eine Tragfläche, ein Leitwerk oder den Flugzeugrumpf enthalten. Innerhalb der Quader besteht die Möglichkeit, individuelle Werte für *lc* anzunehmen.
Um den Automatikmodus zu optimieren, kann der Parameter *DistMin* (bisher linear von *lc* abhängig) direkt von den Eigenschaften einer Geometrie abgeleitet werden. Auf diese Weise lässt sich bei unterschiedlichen Skalierungen die Differenz in der Dicke der verfeinerten Zellschicht (siehe Abbildung 5.7-1) ausgleichen.
Die Berechnung des Parameters *lc* kann für skalierte Geometrien verbessert werden, indem eine stetige, jedoch nur abschnittsweise differenzierbare Funktion für annähernd identisch skalierte Netze sorgt.
Einen großen Vorteil für numerische Simulationen würde auch die gezielte Generierung von Prismenschichten an bestimmten Oberflächen bringen. Im Rahmen dieser Arbeit konnte ein solches Prismengitter bisher nicht erzeugt werden, obwohl Gmsh Werkzeuge hierfür bereitstellt.

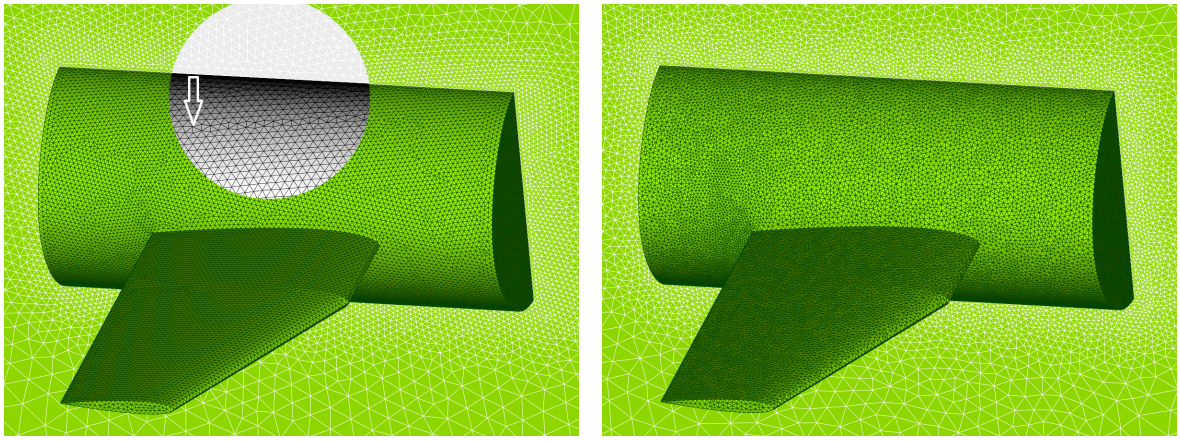
Die Kompatibilität der Ergebnisse zu OpenFOAM ist durch das .msh Format gegeben und erfolgreich getestet (siehe Kapitel 5.4). Für andere Solver müssen ggf. Anpassungen am Dateiexport und bei der Definition der Geometriegruppen (für CFD Randbedingungen) vollzogen werden.

5 Prüfung und Anwendung des Prototyps

Dieses Kapitel dient der bildhaften Darstellung der aktuellen Fähigkeiten des TiGL Meshers in seinem Prototyp Stadium. Zahlreiche Testläufe verdeutlichen den Einfluss von einzelnen Parametern aus dem vorherigen Kapitel. Dabei werden Laufzeiten gemessen, Tetraeder gezählt, und insbesondere die Wirkung der Parameter *lc*, Div_{lin}, *algo2D*, *algo3D*, *optimizeGmsh*, *optimizeNETGEN*, *lcFromCurvature* und *Sigmoid* vorgestellt. Auch eine Präsentation der automatischen Vernetzung und des Workflows von TiGL über Gmsh zu OpenFOAM findet hier statt.
Aus den Ergebnissen kann der Benutzer des Prototyps ableiten, welche Auswirkung die Parameter haben und so schneller zu einem gewünschten Netz kommen.

5.1 Delaunay- und Frontal-Oberflächengitter

Oberflächengitter werden in Gmsh auf verschiedene Weisen angelegt. In jedem Fall bleibt dieses Netz jedoch in seiner ursprünglichen Form bestehen und kann durch keinen der Optimierer nachträglich verbessert werden. Darum stellt es eine wichtige Vorstufe des Volumengitters dar, der genügend Aufmerksamkeit geschenkt werden muss, um bestimmte Qualitäten zu erreichen. Eine bildliche Darstellung der Vorgänge in Kapitel 2.4.1 und 2.4.2 verdeutlicht an dieser Stelle nochmals die Eigenschaften des Frontal Algorithmus' und der Delaunay Methodik (siehe Abbildung 5.1-1). Aufgrund der grundlegend verschiedenen Herangehensweise wurde für die Testläufe in dieser Arbeit bei der Wahl der 2D- und 3D-Algorithmen auf Konsistenz geachtet. Der nicht abgebildete Mesh Adapt Algorithmus stellt vom optischen Eindruck her eine Mischform der beiden anderen Vorgehensweisen dar. Beim Mesh Adapt wechseln sich unstrukturierte (wie bei Delaunay) und gleichmäßige Muster (wie bei Frontal) auf der Oberfläche ab.



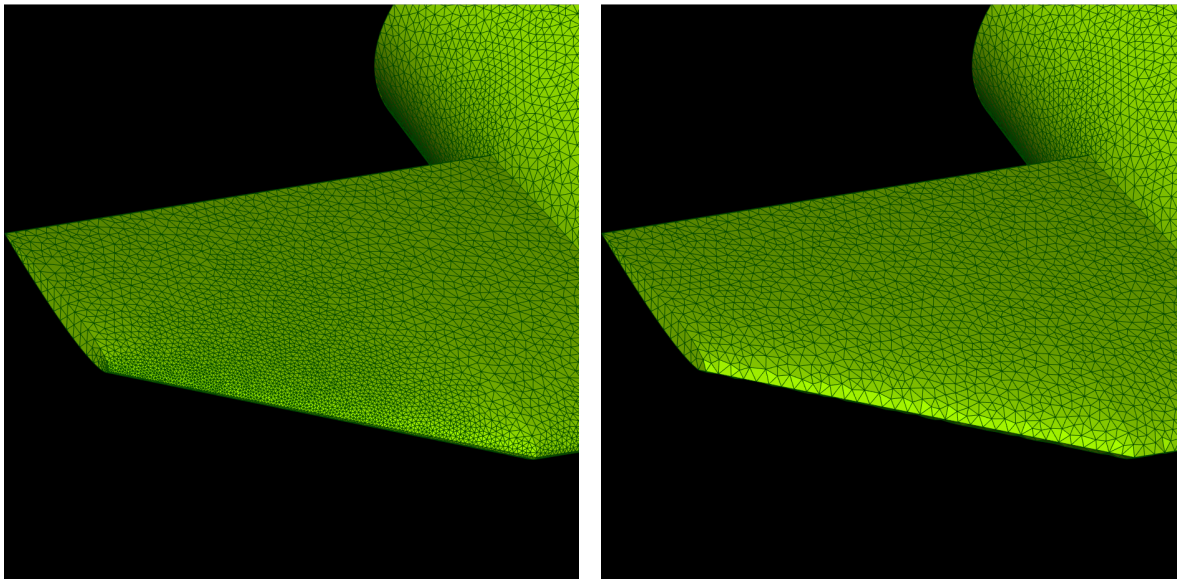
(a) Frontal Oberflächengitter: unstrukturiertes Netz mit Gleichmäßigkeiten und erkennbaren Fronten in dem vergrößerten Bildausschnitt

(b) Delaunay Oberflächengitter: unstrukturiertes Netz ohne Gleichmäßigkeiten

Abbildung 5.1-1: Delaunay- und Frontal-Oberflächengitter: Direkter visueller Vergleich der Algorithmen, angewendet auf die Beispielgeometrie

5.2 Vernetzung gekrümmter Oberflächen

Die Oberflächenalgorithmen Mesh Adapt, Frontal und Delaunay bieten die Option, die Netzfeinheit nicht nur statisch mit *lc* zu steuern, sondern auch eine dynamische, lokale Anpassung der Zellengröße an die Krümmung von Flächen vorzunehmen. Für eine genauere Wiedergabe der Krümmungen reduzieren sich so die zusätzlich notwendigen Elemente auf ein Minimum, weil das globale *lc* konstant bleibt. Das lokale *lc* wird intern von Gmsh ausgerechnet und justiert.
Die automatische Netzanpassung konnte in dieser Arbeit erfolgreich an der Beispielgeometrie getestet werden (siehe Abbildung 5.2-1). Für andere Geometrien (siehe Abbildung 5.8-1) konnten die Oberflächenalgorithmen in den durchgeführten Tests bisher nicht terminieren. Aus diesem Grund beinhaltet die automatische Vernetzung diese Option nicht.



(a) Automatische Anpassung des lokalen *lc* an die Krümmung einer Oberfläche

(b) Festgesetzter Wert für *lc* wird keinerorts unterschritten

Abbildung 5.2-1: Vernetzung gekrümmter Oberflächen: Die automatische Anpassung des lokalen *lc* fördert eine genauere Wiedergabe der Geometrieoberfläche.

5.3 Sigmoid Interpolation für Zellenwachstum

Die Interpolation der Zellengröße zwischen den charakteristischen Größen $LcMin = lc$ und *LcMax* spielt eine wichtige Rolle bei der Gestaltung eines gleichmäßigen Netzes. Die Beispiele auf den Abbildungen 5.3-1 und 5.3-2 wurden bis auf die Sigmoid Option mit identischen Einstellungen vernetzt. Abgebildet ist nur das Sub-Netz der Symmetrieebene aus der Beispielgeometrie. Die Interpolation mit Hilfe einer Sigmoidfunktion bewirkt, dass das Zellenwachstum gleichmäßiger und kontrollierter verläuft. Andererseits werden unvorteilhafte Elemente mit Winkeln deutlich über oder unter 60° vermieden. In der Abbildung 5.3-1 zeigt die Vergrößerung Elemente mit stark variierenden Winkeln, die zwangsläufig durch den „harten“ Übergang zwischen feinem und grobem Netz entstehen. Testläufe haben auch ergeben, dass die Konfiguration in der Abbildung 5.3-2 zeitlich effizienter funktioniert.

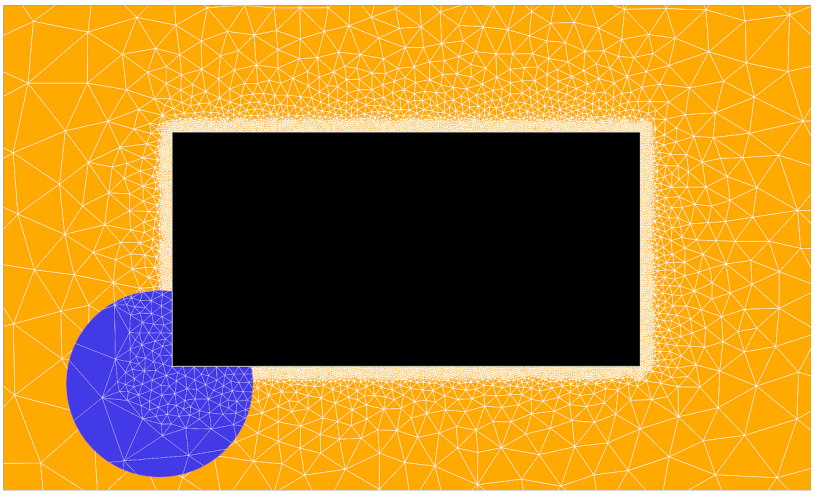


Abbildung 5.3-1: Vernetzte Symmetrieebene ohne Sigmoid Interpolation: Die blaue Vergrößerung in der Ecke zeigt den „harten“ Übergang zwischen grobem / feinem Netz und entsprechend verzerrte Elemente.

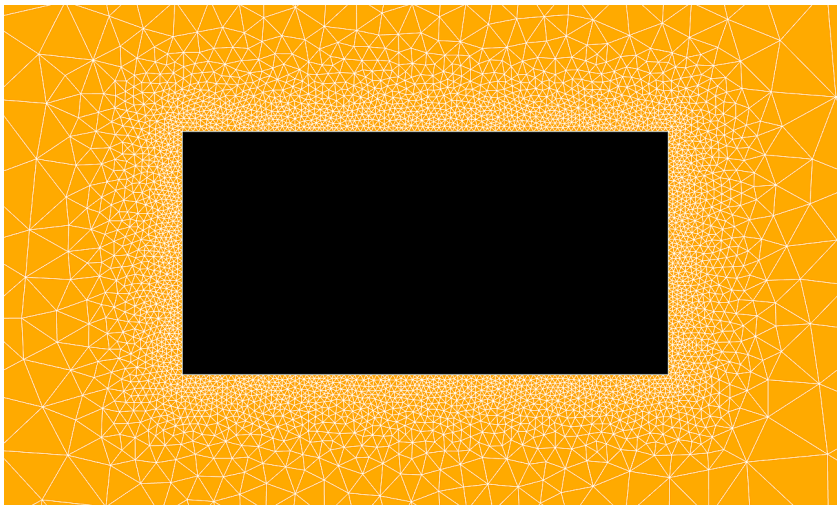
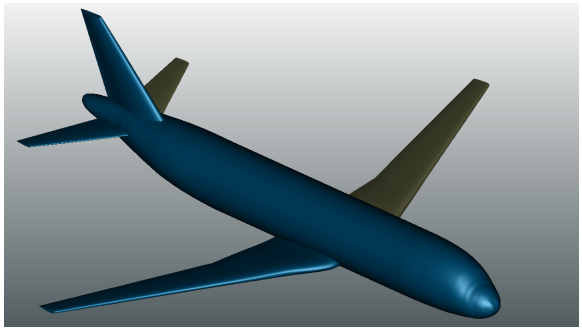


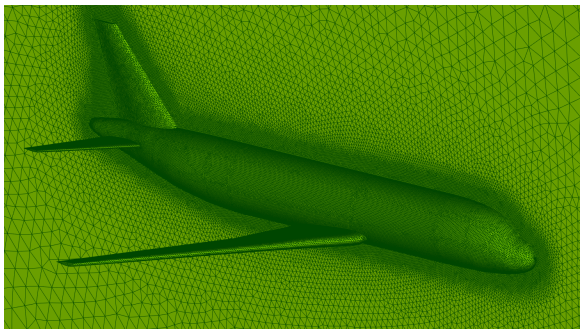
Abbildung 5.3-2: Vernetzte Symmetrieebene mit Sigmoid Interpolation

5.4 Machbarkeit des Workflows

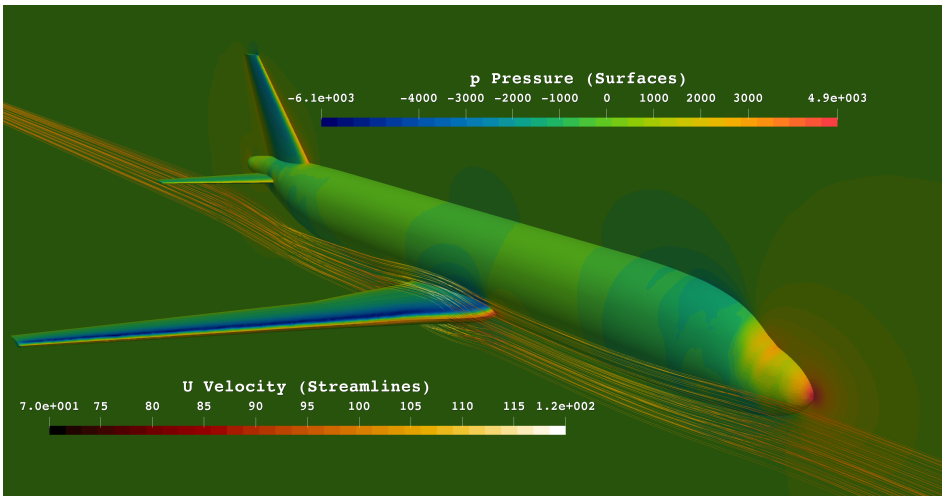
Das Ziel dieser Arbeit besteht u. a. darin, die grundsätzliche Machbarkeit für eine Benutzung der erzeugten Gitter in einer automatisierten Optimierungsumgebung nachzuweisen. Dafür sind vor allem die Schnittstellen zwischen den Softwareprodukten relevant. Die Kompatibilität von Gmsh und TiGL ist durch die gemeinsame Open CASCADE Basis gegeben. Für eine testweise Erprobung der Netze in einem Solver kommt an dieser Stelle OpenFOAM zum Einsatz. Für diesen Transfer gehen die Anforderungen über das Datenformat und vorhandene Sub-Netze für die Randbedingungen hinaus. An dieser Schnittstelle gewinnt die Netzqualität stark an Bedeutung. In den Testläufen für die Simulation in Abbildung 5.4-1c wurde festgestellt, dass die Netzqualität und die Realisierbarkeit einer Simulation auch von der Darstellung der Oberflächen in TiGL abhängt. Dabei besteht ein kausaler Zusammenhang mit der Anzahl der „guide curves“ und „profiles“ in der Geometriedefinition [40]. Die Simulation einer vernetzten Geometrie mit einem Flugzeugrumpf aus 16 „guide curves“ und 60 „profiles“ führte aufgrund der Netzqualität in OpenFOAM zu einer Division durch null und schlug fehl.
Trotzdem ist mit dem Workflow in Abbildung 5.4-1 die grundsätzliche Tauglichkeit des TiGL Meshers für den ihm zugedachten Zweck bewiesen. Die Simulation eines Gmsh Netzes im DLR Solver TAU konnte im Rahmen dieser Arbeit nicht abschließend ausgeführt werden.



(a) Von TiGl generierte Geometrie aus einer CPACS Datei als Grundlage des Netzes in Abbildung 5.4-1b, visualisiert im TiGL Viewer



(b) Vom TiGL Mesher (Gmsh) erzeugtes NETGEN Netz als Grundlage der Simulation in Abbildung 5.4-1c, visualisiert in SALOME



(c) Von OpenFOAM durchgeführte CFD (RANS) Simulation eines Flugzeugs, visualisiert in Paraview: Druckverteilung auf dem Flugzeug: $[m^2/s^2]$ (Druck geteilt durch die Dichte); Geschwindigkeiten der Stromlinien: [m/s] ausgehend von 100 m/s Einlass-Strömung

Abbildung 5.4-1: Workflow Machbarkeit: Alle notwendigen Schritte ausgehend von der CPACS Datei bis zur Simulation in einem Open Source Solver wurden testweise vollzogen, um potentielle Fehler in den jeweiligen Schnittstellen aufzudecken.

5.5 Vernetzungsdauer

Im Kern dieses Unterkapitels steht der Einfluss der charakteristischen Länge *lc* und der Optimierer auf die Laufzeit der Vernetzungsvorgänge. Für die folgenden Diagramme und Tabellen wurden stets Netze mit dem Frontal Algorithmus auf den Oberflächen und dem NETGEN Algorithmus im Volumen erstellt. Dabei wurde *lc* über drei Konfigurationen variiert:

- Frontal & NETGEN Algorithmen + Gmsh[1] & NETGEN[2] Optimierer
- Frontal & NETGEN Algorithmen + Gmsh Optimierer
- Frontal & NETGEN Algorithmen + kein Optimierer

Die Grundlage für die Netze bildet die Passagierflugzeug-Geometrie D150 des DLR inklusive eines Fernfeldes (siehe Abbildung 5.8-1b). Die Berechnung der Netze fand auf einem Intel Xeon E5530 Kern (2,40 GHz) statt.

[1] Der Gmsh Optimierer vollzieht „edge swapping“ und „vertex re-positioning“ [16] (siehe Kapitel 2.4.3).
[2] Der NETGEN Optimierer vollzieht „edge swapping“, „vertex re-positioning“ und „point collapsing“ [33](siehe Kapitel 2.4.1.2).

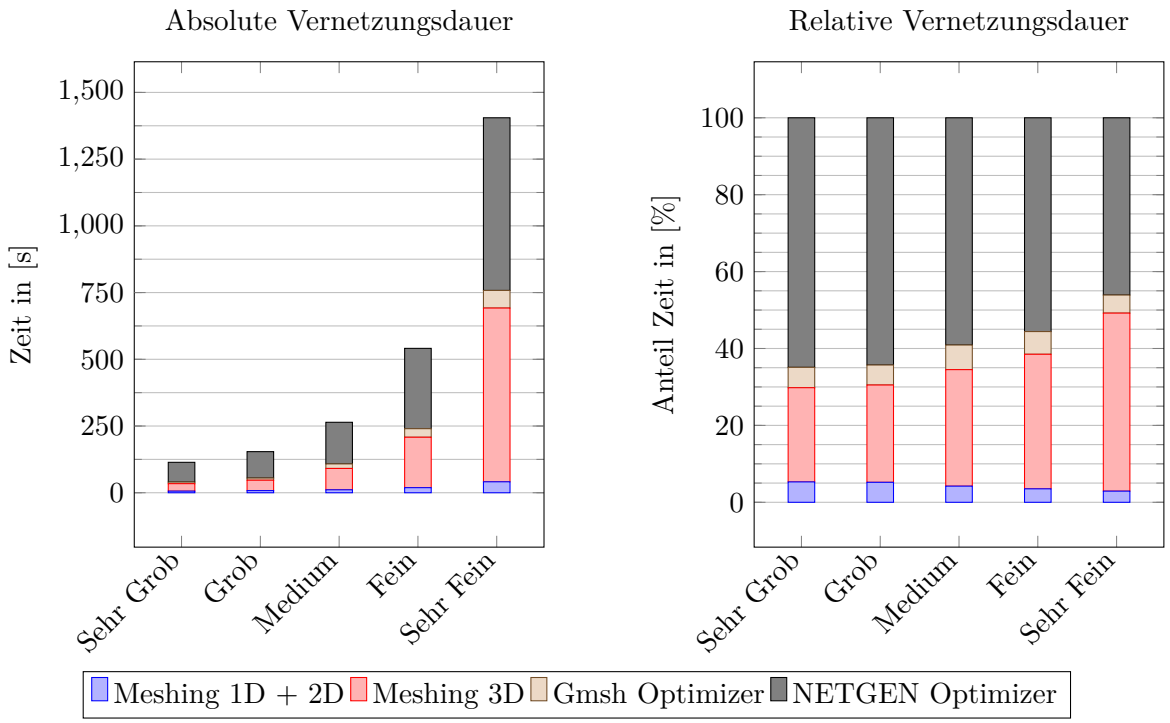


Abbildung 5.5-1: Vergleich absoluter und relativer Laufzeiten für fünf Netzfeinheitstufen

Tabelle 5.5-1: Übersicht zu Laufzeiten der Vernetzung und entstehenden Tetraedern bei verschiedenen Feinheitsstufen; Gmsh und NETGEN Optimierung inbegriffen

Netz	*lc*	Anzahl Tetraeder	Laufzeit [s]
Sehr Grob	0,08	449.000	114
Grob	0,07	571.000	154
Medium	0,05	903.000	264
Fein	0,04	1.724.000	541
Sehr Fein	0,03	3.407.000	1405

Tabelle 5.5-2: Übersicht zu Laufzeiten der Vernetzung und entstehenden Tetraedern bei verschiedenen Feinheitsstufen; Gmsh Optimierung inbegriffen

Netz	*lc*	Anzahl Tetraeder	Laufzeit [s]
Sehr Grob	0,08	592.000	40
Grob	0,07	753.000	55
Medium	0,05	1.261.000	113
Fein	0,04	2.275.000	243
Sehr Fein	0,03	4.708.000	762

Tabelle 5.5-3: Übersicht zu Laufzeiten der Vernetzung und entstehenden Tetraedern bei verschiedenen Feinheitsstufen; Keine Optimierung inbegriffen

Netz	*lc*	Anzahl Tetraeder	Laufzeit [s]
Sehr Grob	0,08	606.000	34
Grob	0,07	774.000	47
Medium	0,05	1.303.000	105
Fein	0,04	2.330.000	208
Sehr Fein	0,03	4.857.000	708

Aussagen, die mit der vorliegenden Algorithmuskombination aus den Diagrammen in Abbildung 5.5-1 und den Tabellen 5.5-1, 5.5-2 und 5.5-3 abgeleitet werden können:

- Je kleiner *lc* ausfällt, desto sensibler reagiert die Zahl der generierten Elemente.
- Die Vernetzungsdauer verhält sich nicht linear zu der entstehenden Zellenzahl. Die Laufzeit wächst überproportional an.
- Der prozentuale Anteil des Rechenaufwandes für die Volumenvernetzung steigt zusammen mit der Feinheit.
- Der NETGEN Optimierer benötigt immer ein Vielfaches der Zeit, die der Gmsh Optimierer in Anspruch nimmt.
- Mit steigender Feinheit sinkt der prozentuale Anteil des NETGEN Optimierers an der Berechnugsdauer.

5.6 Gmsh und NETGEN Optimierer

Für die Optimierung der Volumennetze stehen zwei Möglichkeiten zur Verfügung: die proprietäre Lösung von Gmsh[3] und die Variante von NETGEN[4]. An dieser Stelle sei nochmals erwähnt, dass die Oberflächennetze in beiden Fällen nicht optimiert oder verändert werden können. Die Oberflächengitter stellen somit potentiell auch den limitierenden Faktor dar. Es werden nun die Ergebnisse aus folgenden Testläufen vorgestellt und die entsprechenden Rückschlüsse aus dem Vergleich gezogen.

Tabelle 5.6-1: Versuchsaufbau zum Vergleich der Netzoptimierer

Optimierer / Algorithmus	Kein	Gmsh	NETGEN	Beide
Delaunay + New Delaunay	Run 1	Run 2	Run 3	Run 4
Frontal + NETGEN	Run 5	Run 6	Run 7	Run 8

Tabelle 5.6-2: Versuchsergebnisse zum Vergleich der Netzoptimierer: Die letzten zwei Spalten beinhalten Angaben zu den anteiligen Zeiten der Optimierer an der Laufzeit.

Run	Anzahl Tetraeder	Laufzeit [s]	Gmsh Opt. [s]	NETGEN Opt. [s]
1	1.762.000	143	-	-
2	1.731.000	155	12	-
3	1.377.000	413	-	270
4	1.519.000	441	12	286
5	849.000	66	-	-
6	827.000	75	10	-
7	602.000	169	-	104
8	623.000	181	10	105

Die Grundlage für die Netze bildet die Beispielgeometrie mit den Abmaßen $32\text{m} \times 32\text{m} \times 16\text{m}$ inklusive eines Fernfeldes (siehe Abbildung 5.1-1b). Die Berechnung der Netze fand auf einem Intel Xeon E5530 Kern mit 2,40 GHz statt. Für alle Durchläufe wurde $lc = 0,12$ und $DistMin = 1\text{m}$ gewählt.

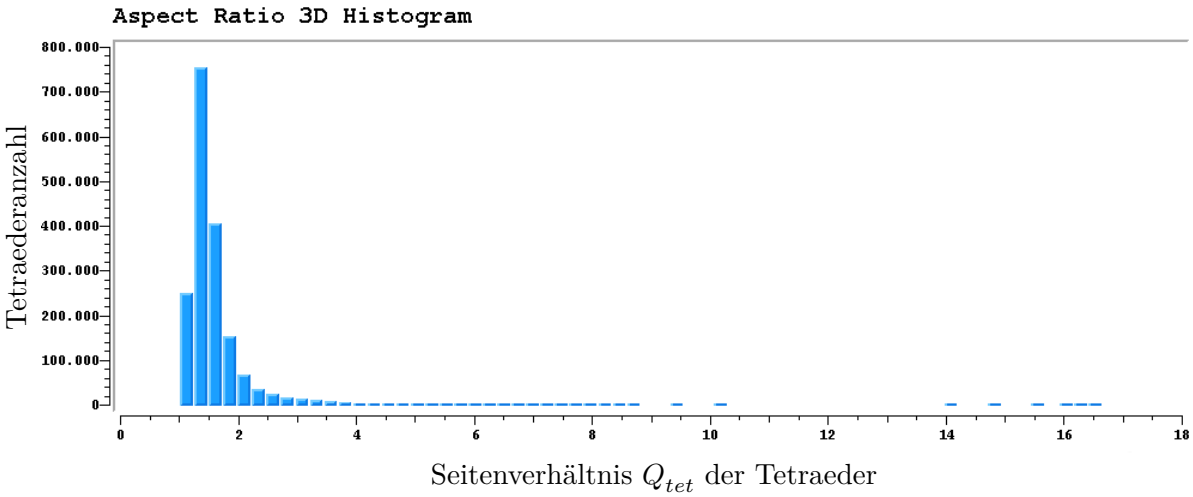
[3] Der Gmsh Optimierer vollzieht „edge swapping“ und „vertex re-positioning“ [16] (siehe Kapitel 2.4.3).

[4] Der NETGEN Optimierer vollzieht „edge swapping“, „vertex re-positioning“ und „point collapsing“ [33](siehe Kapitel 2.4.1.2).

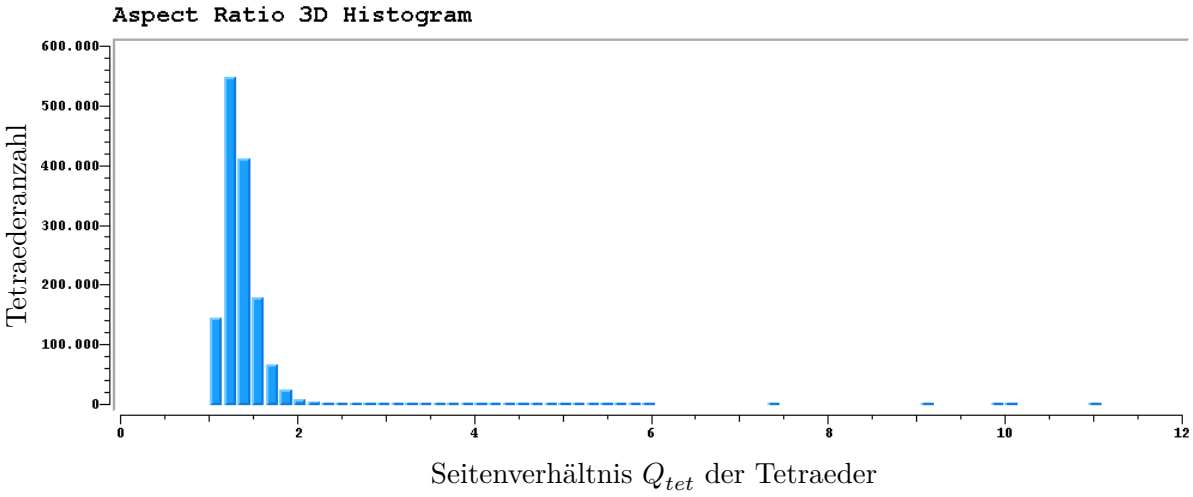
Die Ergebnisse aus der zweiten Spalte der Tabelle 5.6-1 werden nicht in den Abbildungen 5.6-1 und 5.6-2 visualisiert, da die höchsten 3D-Seitenverhältnisse bei 2200 (Run 1) und 1.200.000 (Run 5) liegen. Der Plot verliert durch eine solche Spreitzung der Werte seine Aussagekraft / Lesbarkeit. Aus dieser Tatsache lässt sich bereits der erste Schluss folgern: Grundsätzlich ist eine Optimierung des Netzes sehr empfehlenswert, weil ansonsten mit beiden Algorithmuskombinationen einzelne, stark verzerrte Elemente auftreten. Zu beachten ist bei dem folgenden Vergleich, dass die kombinierten Algorithmen ohne Optimierung bereits Unterschiede in der Zellenzahl aufweisen: 1.762.000 [Delaunay (2D) + New Delaunay (3D)] und 849.000 [Frontal (2D) + NETGEN (3D)].

Aussagen, die mit den zwei vorliegenden Algorithmuskombinationen aus der Tabelle 5.6-2 und den Abbildungen 5.6-1 und 5.6-2 abgeleitet werden können:

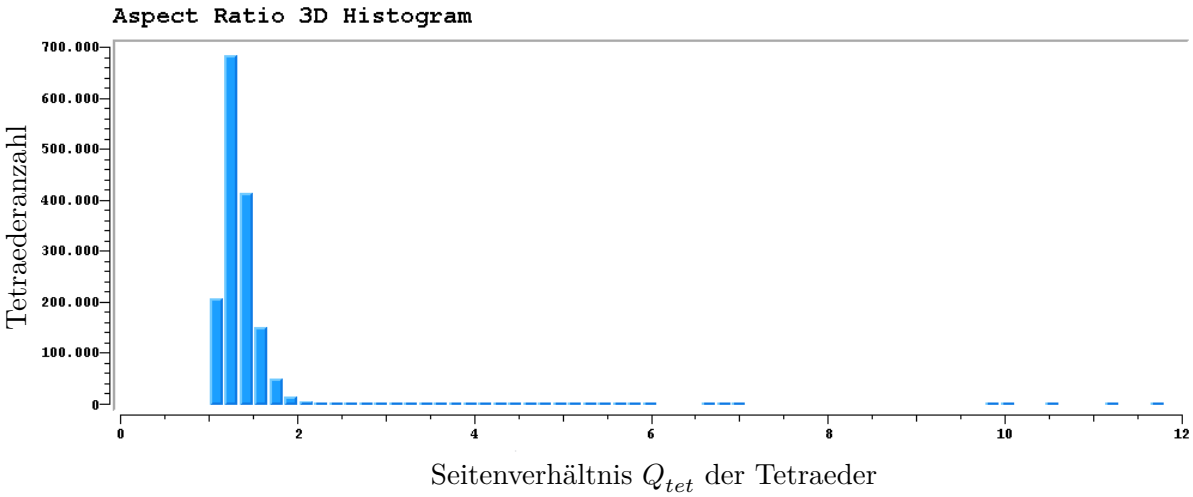
- Der Gmsh Optimierer arbeitet effizienter, wenn die aufgewendete Zeit und die entstehenden maximalen Seitenverhältnisse in Betracht gezogen werden.
- Ein Frontal / NETGEN Netz passt am besten zum NETGEN Optimierer.
- Der NETGEN Optimierer ist nur empfehlenswert, wenn der Gmsh Optimierer noch kein zufriedenstellendes Ergebnis erzeugen kann. Jedoch gibt es keine positive, sondern teilweise negative Effekte (auf einem Delaunay Netz), wenn man beide Optimierer kombiniert.
- Wenn beide Optimierer zur Anwendung kommen, ist die resultierende Zellenzahl höher, als wenn nur NETGEN die Optimierung übernimmt.
- Ohne Optimierer erzeugt die Kombination [Delaunay (2D) + New Delaunay (3D)] ein besseres Netz.
- Ein hoher Zeitaufwand bedeutet beim NETGEN Optimierer keine zwingende Verbesserung der Tetraederqualitäten, wenn zuvor der Gmsh Optimierer angewendet wurde. In solch einem Fall reduziert sich hauptsächlich nur die Tetraederanzahl durch „point collapsing“ (siehe Abbildung 2.4.1.2-1).
- Der Gmsh Optimierer führt eine vergleichsweise schnelle Korrektur der Tetraeder herbei, ohne die Zellenanzahl drastisch zu reduzieren.
- Der Gmsh Optimierer erreicht auf einem Frontal Netz eine schlechtere Verteilung der Seitenverhältnisse Q_{tet}, als auf einem Delaunay Netz.
- Im Hinblick auf die statistische Verteilung von Q_{tet}, die Laufzeit und die entstehende Tetraederanzahl stellt sich die Wahl der Algorithmen Delaunay (Oberfläche) und New Delaunay (Volumen) in Kombination mit dem Gmsh Optimierer für die Beispielgeometrie als effizienteste Lösung heraus.



(a) Run 2: Delaunay-Netz mit Gmsh Optimierung

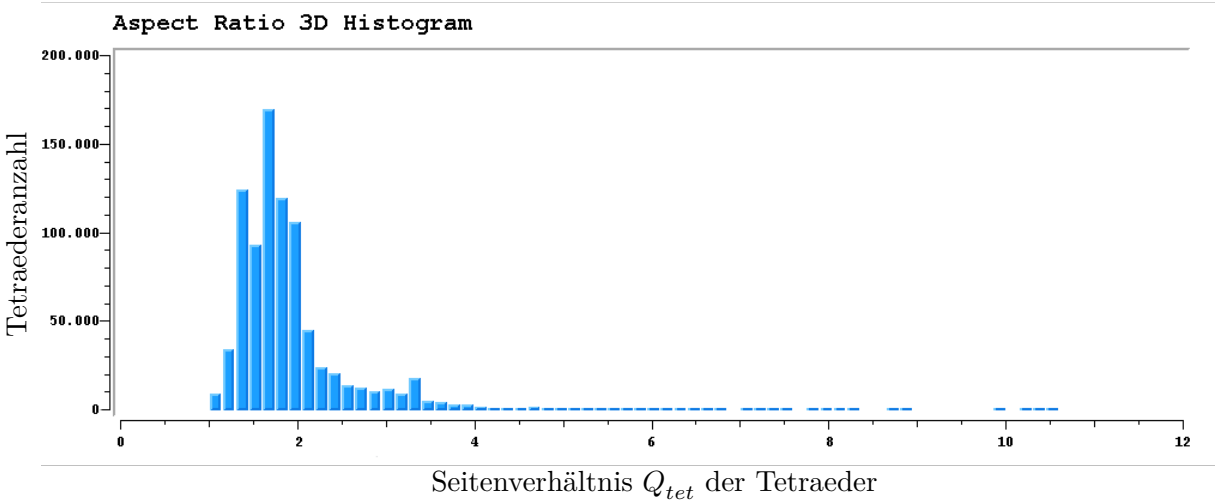


(b) Run 3: Delaunay-Netz mit NETGEN Optimierung

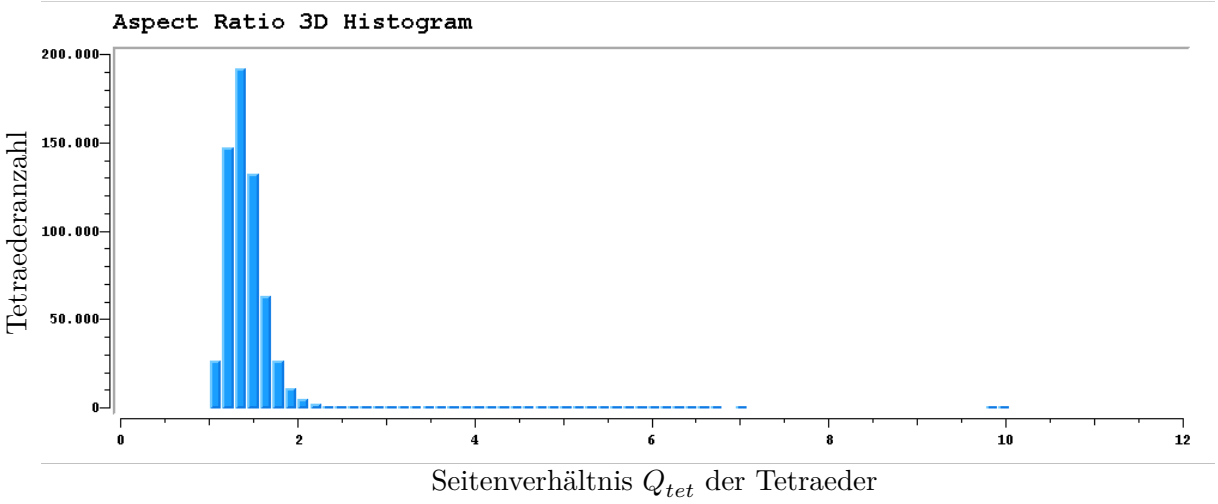


(c) Run 4: Delaunay-Netz, zunächst mit Gmsh, dann mit NETGEN optimiert

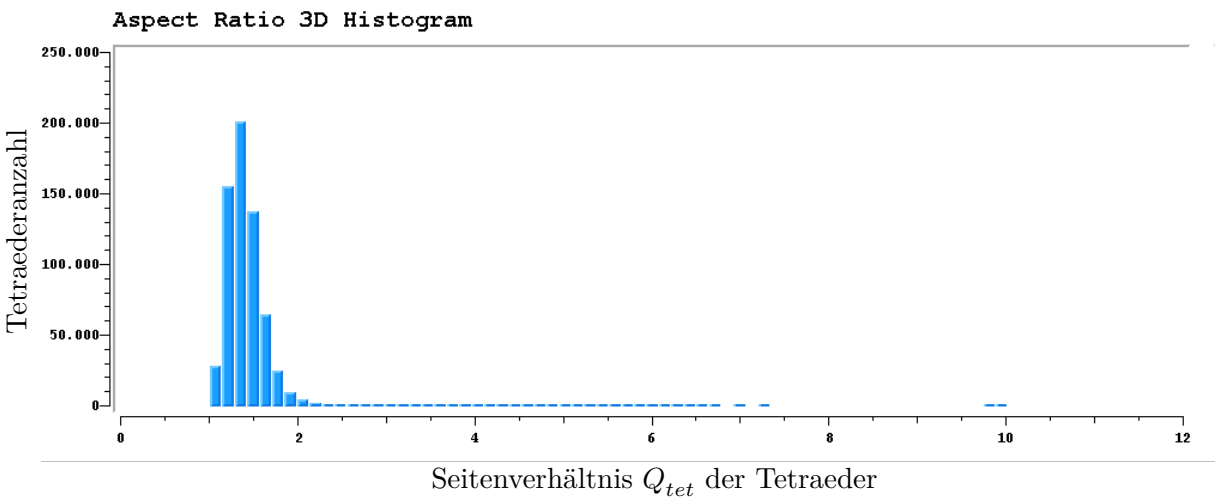
Abbildung 5.6-1: Optimierungen auf Delaunay-Netzen: Vergleich zwischen Delaunay-Netzen mit Gmsh Optimierung, NETGEN Optimierung und der Kombination beider Optimierer; Darstellung: Verteilung der Seitenverhältnisse (Q_{tet} siehe Gleichung 2.5-6) über die Zellenanzahl



(a) Run 6: Frontal-Netz mit Gmsh Optimierung



(b) Run 7: Frontal-Netz mit NETGEN Optimierung



(c) Run 8: Frontal-Netz, zunächst mit Gmsh, dann mit NETGEN optimiert

Abbildung 5.6-2: Optimierungen auf Frontal-Netzen: Vergleich zwischen Frontal-Netzen mit Gmsh Optimierung, NETGEN Optimierung und der Kombination beider Optimierer; Darstellung: Verteilung der Seitenverhältnisse (Q_{tet} siehe Gleichung 2.5-6) über die Zellenanzahl

5.7 Automatische Gittererzeugung

In diesem Abschnitt werden die Möglichkeiten und bisherigen Grenzen des TiGl Meshers im Sinne der Automatisierung visualisiert. Hierbei stellt sich die Frage, inwieweit die automatisch erzeugten Netze von skalierten Geometrien noch Ähnlichkeit miteinander haben. Die vom Benutzer benötigten Eingaben zur automatisierten, parametrisierten Vernetzung sind in der Abbildung A.1-1 nochmals dargestellt.

Zunächst soll der Einfluss des Parameters Div_{lin} auf die Netzgröße tabellarisch aufgezeigt werden (siehe Tabelle 5.7-1). Hierfür wurde die Beispielgeometrie auf sechs verschiedene Größen skaliert und anschließend automatisch zweimal mit verschiedenen Werten für Div_{lin} vernetzt. Die erste Spalte zeigt die Skalierung der Geometrie. Die zweite und dritte Spalte verdeutlichen, dass ein vergrößerter Divisor Div_{lin} eine konstante prozentuale Verminderung des lc für alle Skalierungen hervorruft. Bei Betrachtung der vierten und fünften Spalte fällt auf, dass die resultierende Anzahl von generierten Elementen für große Geometrien einen markanten Anstieg erfährt. In einem Vergleich zwischen der dritten und fünften Spalte wird deutlich, dass der prozentuale Zuwachs der Tetraederanzahl nicht in einem linearen Verhältnis zur prozentualen Abnahme des lc steht. Der erhöhte Zuwachs der Tetraederanzahl für die größte getestete Geometrie hängt mit der hohen Sensibilität des lc zusammen. Anders ausgedrückt schafft es die Gleichung 4.4-1 nicht, das Wachstum von Geometrie und lc für jede Skalierung in Einklang zu bringen. Die Anzahl generierter Tetraeder befindet sich trotz dieses Sprungs jedoch in der gleichen Größenordnung. Für die ersten fünf Skalierungen lässt sich das Wachstum des Gitters wie gewünscht nahezu linear über Div_{lin} steuern.

Tabelle 5.7-1: Anpassung der Formel für lc mittels Div_{lin}

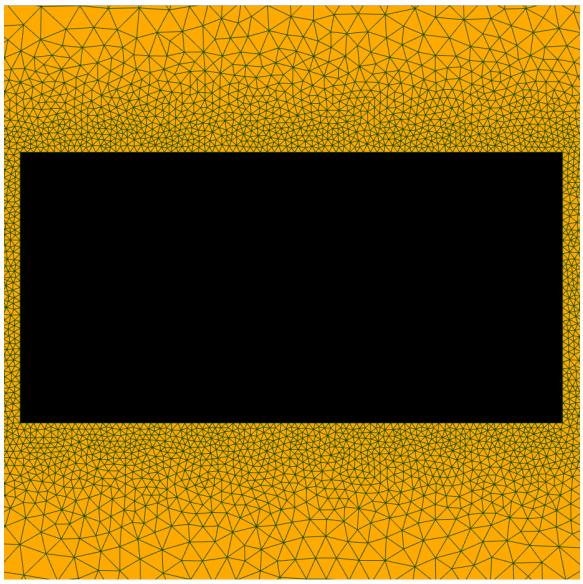
Abmaße [m]	lc für Div_{lin} = 150	180	Abweichung lc [%]	Anzahl Tetraeder		Abweichung Tetraeder [%]
$2 \times 2 \times 1$	0,01528	0,01273	$-16,7$	694.000	1.088.000	$+56,8$
$4 \times 4 \times 2$	0,02982	0,02485	$-16,7$	585.000	909.000	$+55,4$
$8 \times 8 \times 4$	0,05403	0,04503	$-16,7$	570.000	878.000	$+54,0$
$16 \times 16 \times 8$	0,09375	0,07813	$-16,7$	618.000	963.000	$+55,8$
$32 \times 32 \times 16$	0,15784	0,13153	$-16,7$	716.000	1.119.000	$+56,3$
$64 \times 64 \times 32$	0,25991	0,21659	$-16,7$	893.000	1.507.000	$+68,8$

Um die Sensibilität des lc zu verdeutlichen, wurde eine weitere Versuchsreihe aufgebaut. Die Berechnung der Netze für sechs skalierte Geometrien fand auf einem Intel Xeon E5530 Kern (2,40 GHz) statt. In der folgenden Tabelle 5.7-2 wird dargestellt, welche Werte der Benutzer manuell für das lc wählen *müsste*, um eine nahezu identische Anzahl von Elementen für alle sechs Skalierungsstufen zu erreichen. Als Ziel der Vernetzung in der Testreihe ist ein Gitter mit $\sim$ 620.000 Tetraedern gesetzt. Dies entspricht bereits dem vierten Durchlauf in der Übersicht. Der Versuchsreihe ist entnehmbar, dass in Abhängigkeit von den Geometrieausmaßen bereits kleine Änderungen des lc relevante Abweichungen in der Elementzahl erzeugen.

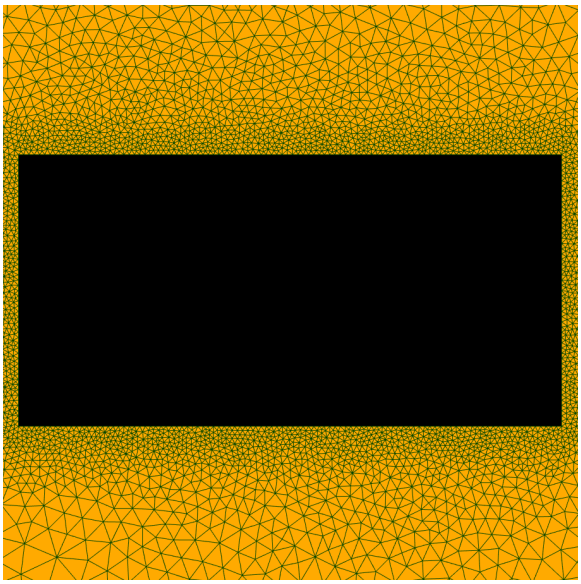
Tabelle 5.7-2: Sensibilität des Parameters lc

Abmaße [m]	lc autom.	man.	Anzahl Tetraeder		Laufzeit [s]	
$2 \times 2 \times 1$	0,01528	0,016	694.000	625.000	36	29
$4 \times 4 \times 2$	0,02982	0,029	585.000	623.000	26	29
$8 \times 8 \times 4$	0,05403	0,052	570.000	619.000	31	31
$16 \times 16 \times 8$	0,09375		618.000		29	
$32 \times 32 \times 16$	0,15784	0,169	716.000	619.000	40	32
$64 \times 64 \times 32$	0,25991	0,302	893.000	618.000	82	38

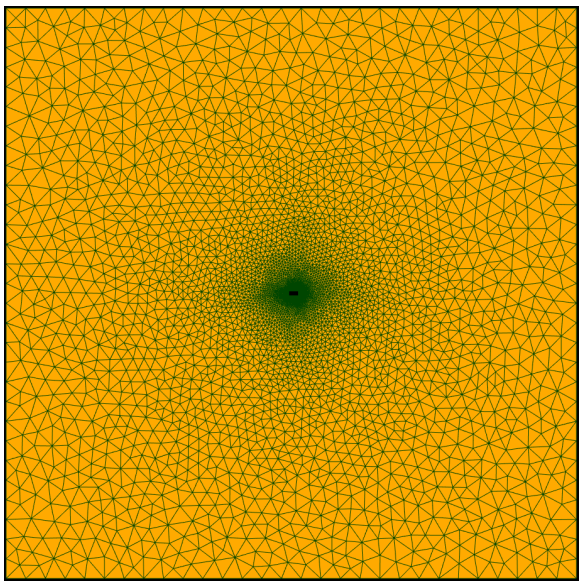
Für einen visuellen Vergleich der automatischen Vernetzung von skalierten Geometrien folgen die Abbildungen 5.7-1, 5.7-2 und 5.7-3. Weitere Eindrücke des automatischen Modus' für tatsächliche Flugzeuggeometrien sind in der Abbildung 5.8-1 zu finden.



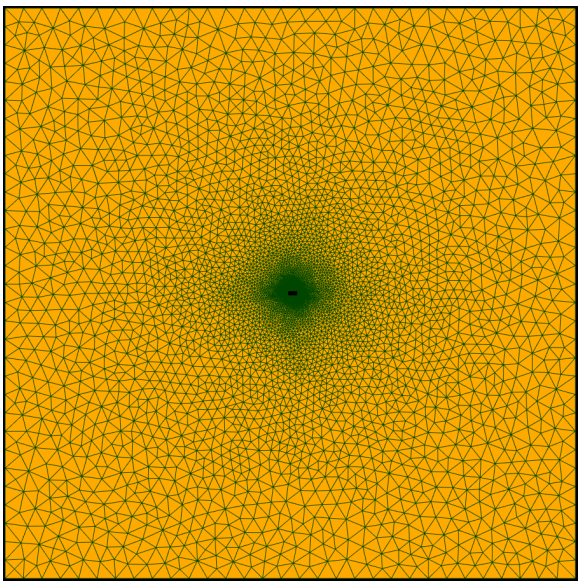
(a) Die Maße des schwarzen Ausschnitts betragen 2m × 4m. Das gesamte Netz besteht aus 50.000 Dreiecken und 585.000 Tetraedern.



(b) Die Maße des schwarzen Ausschnitts betragen 16m × 32m. Das gesamte Netz besteht aus 74.000 Dreiecken und 716.000 Tetraedern.



(c) Die Maße der Symmetrieebene betragen 240m × 240m.



(d) Die Maße der Symmetrieebene betragen 1920m × 1920m.

Abbildung 5.7-1: Automatisch vernetzte Beispielgeometrie: Dargestellt ist die Symmetrieebene aus zwei verschiedenen Skalierungen der Geometrie. Jeweils rechts ist die höhere Skalierung abgebildet. Relativ zu den Abmaßen des Ausschnitts hat das linke Beispiel (a) eine dickere Schicht feiner Elemente. Die Abbildungen (b) und (d) weisen einen höheren allgemeinen Feinheitsgrad auf.

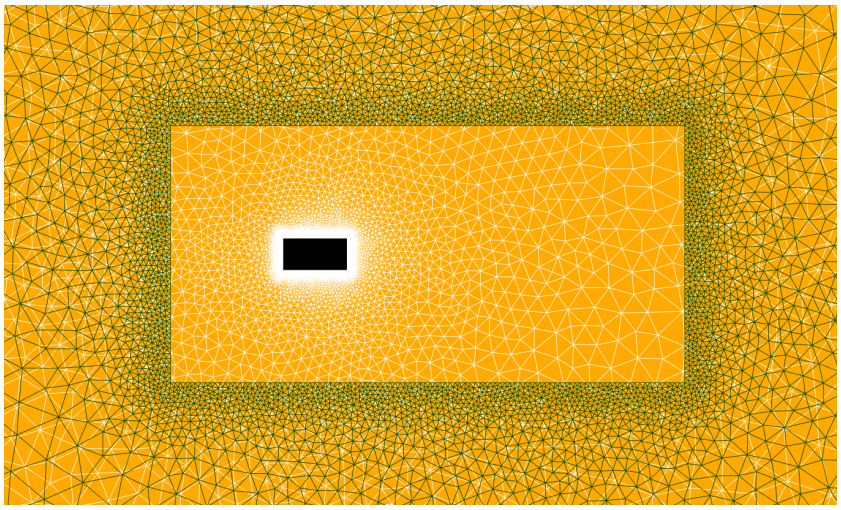


Abbildung 5.7-2: Kombinierte Ansicht der Vernetzungsläufe in Abbildung 5.7-1 verdeutlicht das Größenverhältnis: Gezeigt sind übereinander gelegte Symmetrieebenen mit den jeweiligen Ausschnitten (dunkles Netz 16m × 32m bzw. helles Netz 2m × 4m) für die eigentliche Geometrie.

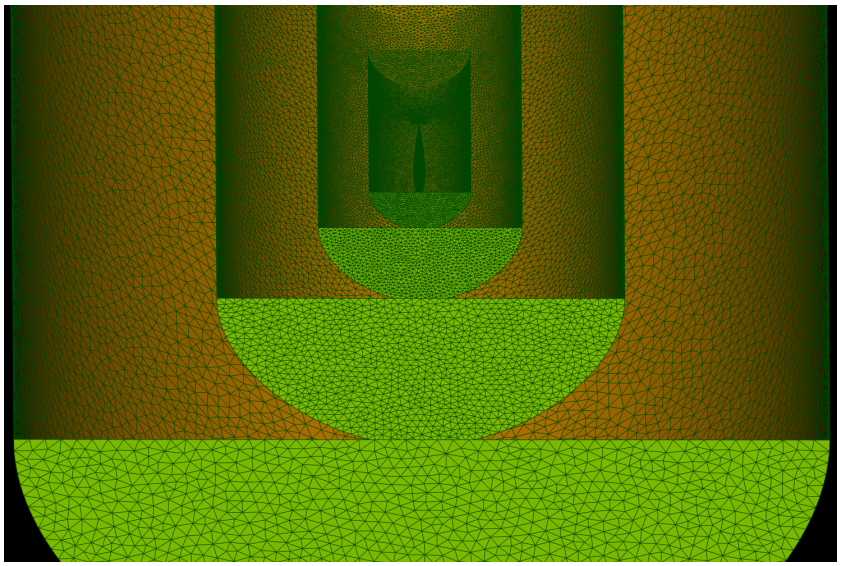
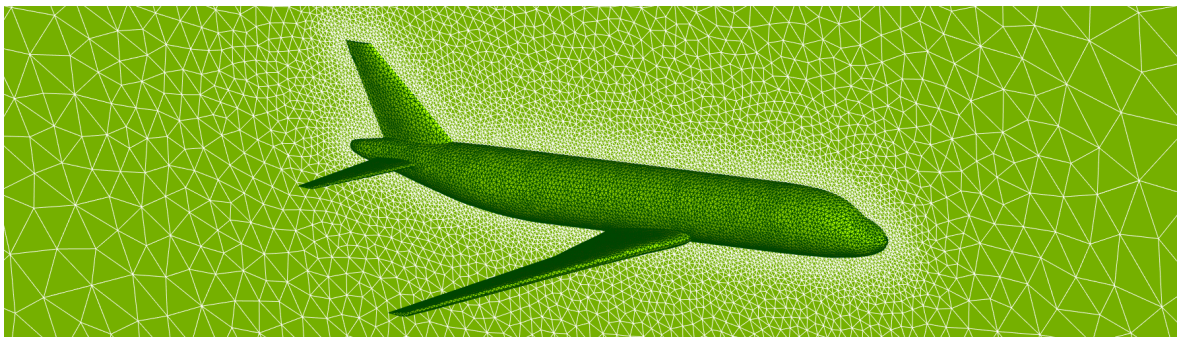


Abbildung 5.7-3: Automatische Vernetzung vier skalierter Beispielgeometrien: Vergleich der vier automatischen Vernetzungsdurchläufe (basierend auf der Gleichung 4.4-1) zwischen unterschiedlich großen Geometrien. Die Radien der abgebildeten Halbkreise betragen 8, 4, 2 und 1 Meter. Sie gehören jeweils zu einer der skalierten Varianten der Beispielgeometrie.

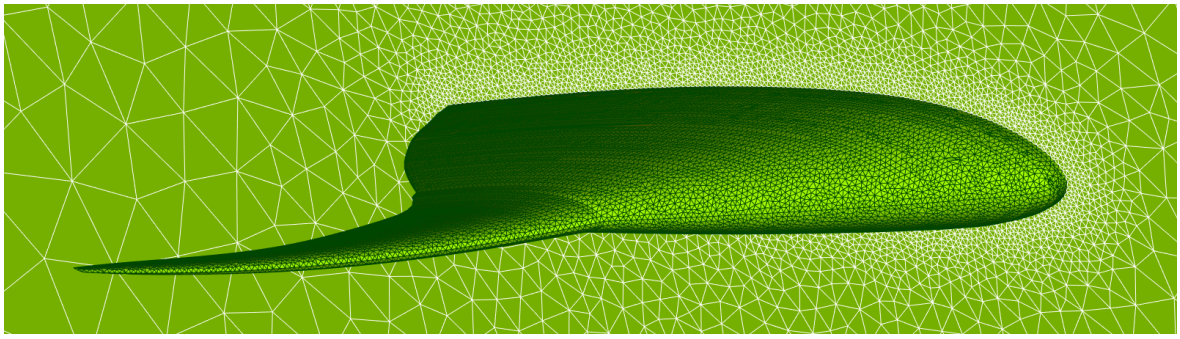
5.8 Robustheit automatischer Vernetzung

Als Ergänzung wurden im Rahmen dieser Arbeit drei verschiedene Flugzeugkonfigurationen (siehe Abbildung 5.8-1) aus TiGL exportiert und testweise im automatischen Modus vernetzt. Anhand dieser drei Geometrien ist eine grundsätzliche Robustheit noch nicht abzuleiten. Die Konfigurationen konnten jedoch erfolgreich vernetzt werden und liegen in dem Spektrum, das für die Anwendung des TiGL Meshers in einer Optimierungsumgebung für den Flugzeugvorentwurf zu erwarten ist.

(a) Drohne mit ausgefahrener Klappe, Abmaße: $11,4\text{m} \times 7,6\text{m} \times 1,3\text{m}$, 50.000 Dreiecke, 547.000 Tetraeder



(b) Passagierflugzeug D150, Abmaße: $37,8\text{m} \times 17,0\text{m} \times 9,6\text{m}$, 34.000 Dreiecke, 320.000 Tetraeder



(c) Nurflügelflugzeug / Flying Wing, Abmaße: $40,0\text{m} \times 32,1\text{m} \times 8,1\text{m}$, 41.000 Dreiecke, 414.000 Tetraeder

Abbildung 5.8-1: Automatisch vernetzte Flugzeugkonfigurationen

6 Fazit und Ausblick

Dieses abschließende Kapitel präsentiert nochmals die wichtigsten, bereits unternommenen Schritte, um eine parametrisierte und somit potentiell automatisierbare Vernetzung von TiGL Geometrien für CFD Simulationen im Flugzeugvorentwurf zu erreichen. Gleichzeitig erfährt der Leser, in welche Richtung weitere Schritte gehen können, um den TiGL Mesher Prototyp in seinem jetzigen Status weiterzuentwickeln.

In dieser Arbeit konnte ein Softwareprototyp vorgestellt werden, der grundsätzlich geeignet ist, TiGL um eine einfach zu bedienende, manuelle oder wahlweise automatische Vernetzungsfunktion zu bereichern. Als Basis für den TiGL Mesher dient die Gmsh API, die sich gegenüber der SALOME C++ API für diese Anwendung als geeigneter herausgestellt hat (siehe Kapitel 3.5). Für die Applikation einer höher entwickelten Version des TiGL Meshers in MDOs enthält der vorliegende Prototyp den Ansatz einer parametrisierten Vernetzungsautomatik (siehe Kapitel 4.4). Zusammen mit der bereits vorhandenen parametrisierten, automatisierbaren Geometriegenerierung in TiGL ist damit eine Grundlage für die Durchführung von automatisierten Euler- oder groben RANS-Simulationen (siehe Kapitel 2.1.2) in einer Optimierungsschleife gelegt. Die grundsätzliche Realisierbarkeit dieses Vorhabens und die Eignung der ausgewählten Gmsh API wurde demonstriert. Dafür vernetzte der TiGL Mesher eine Passagierflugzeugkonfiguration aus TiGL, die anschließend mittels der RANS Gleichungen in OpenFOAM erfolgreich simuliert werden konnte (siehe Kapitel 5.4).

Für zukünftige Weiterentwicklung ist diesbezüglich noch zu prüfen, inwieweit die erzeugten Gitter valide Ergebnisse in der Strömungssimulation hervorbringen. Dafür ist eine Sensitivitätsstudie des Netzes und der Vergleich mit anderen validierten Simulationen oder Windkanalresultaten notwendig. Im jetzigen Stadium des Softwareprototyps bietet sich auch eine vertiefende Beschäftigung mit den Anforderungen der Simulationsmethoden (Euler / RANS) an das Gitter an. Ggf. ist der TiGL Mesher anschließend um weitere Netzverfeinerungsmöglichkeiten und die Funktionalität für Prismenschichten (für RANS Rechnungen) zu erweitern. Der bisherige Ansatz einer einheitlichen Netzfeinheit auf Rumpf *und* Flügeln bedeutet einen ungleich erhöhten Zeitaufwand im Vergleich zum Informationsgewinn, wenn bspw. an einem Flügel lokal eine höhere Auflösung des Netzes / physikalischer Effekte gefordert wird. Mit der gezielten lokalen Verfeinerung des Netzes an den Flügelanströmkanten und den Flügelhinterkanten könnte wertvolle Vernetzungs- und Simulationszeit eingespart werden. Die Anpassung des Netzes an die Oberflächenkrümmung (siehe Kapitel 5.2) bietet einen vielversprechenden Ansatz für die Vernetzung der Anströmkanten. Die Anwendung dieser Funktion über die Beispielgeometrie hinaus bleibt ein Thema für nachfolgende Entwicklungsschritte.

Ebenso bedarf es im Hinblick auf eine robuste Anwendung in einer Optimierungsumgebung weiterer Untersuchung der bereits erwähnten Problematik bei einer erhöhten Anzahl von „guide curves“ und „profiles“ in der parametrischen Geometriedefinition. Beim Test des Workflows brachte ein Flugzeugrumpf aus 16 „guide curves“ und 60 „profiles“ den OpenFOAM Solver zum Erliegen (siehe Kapitel 5.4). Eine Geometrie, die im Prototyp eine robuste Vernetzung ermöglicht, führt nicht zwingend auch zu einer robusten Simulation. Hierfür gilt es, die Voraussetzungen besser zu verstehen, unter denen eine Geometrie zuverlässig vernetzt *und* simuliert werden kann, damit unter allen Umständen Vorkommnisse dieser Art in einer zukünftigen Optimierungsschleife unterbunden bleiben.

Der TiGL Mesher wurde ausgiebig getestet, um den Einfluss der Vernetzungsparameter und die Ergebnisse des Automatikmodus' zu präsentieren. Bei den Parametern zur Auswahl der Algorithmen und Netzoptimierer sind in Bezug auf Netzqualität und Laufzeit pro Tetraeder die Delaunay Algorithmen in Kombination mit dem Gmsh Optimierer die effizienteste Wahl. Für die automatische Netzgenerierung bestand in der Entwicklung u. a. die Herausforderung, Geometrien verschiedener Skalierungen gleichwertig zu vernetzen. Aus den Abbildungen in Kapitel 5.7 ist zu entnehmen, dass die relative Größe der Zellen, ihre Verteilung und Dichte sich bereits gut an die Skalierung anpasst. Grund dafür ist die eigenständig heuristisch entwickelte Formel für den zentralen Parameter *lc* in Gleichung 4.4-1. Trotzdem bleibt noch Verbesserungspotential (siehe Kapitel 4.7) für diese Formel, die den Automatikmodus möglich macht. Der Schlüssel für die Weiterentwicklung des Automatikmodus' ist die Sensibilität des Parameters *lc* (siehe Tabelle 5.7-2). Positive Eindrücke im Sinne der Vernetzungsrobustheit hinterlässt die automatisierte Vernetzung von verschiedenartigen Flugzeuggeometrien in Kapitel 5.8.

Der entwickelte Prototyp ist zwar theoretisch kompatibel zu TiGL, jedoch noch nicht integriert. Bei der Integration des TiGL Meshers bietet es sich an, neue entsprechende Optionen zur manuellen und automatischen Vernetzung in der GUI des TiGL Viewers einzuführen. Für eine erweiterte Solverkompatibilität außerhalb von OpenFOAM sollten auch die Schnittstellen zu den DLR CFD Solvern TAU und Flucs weiter untersucht werden.

Insgesamt kann der TiGL Mesher die zu Beginn dieser Arbeit gestellten Anforderungen erfüllen. Selbst ohne Prismenschichten auf der Flugzeugoberfläche ist das Netz grundsätzlich für Euler Rechnungen im Flugzeugvorentwurf tauglich [17, 25]. Passend zur parametrisierten Definition und automatisierbaren Änderung der Geometrien in TiGL bietet der Prototyp eine der Geometrie angepasste, parametrisierte Vernetzung an. Der TiGL Mesher bildet eine Basis für zukünftige Aktivitäten in Richtung einer Integration in eine MDO für Flugzeugkonzepte.

A Anhang 1

A.1 Automatische und manuelle Programmabläufe des Prototyps

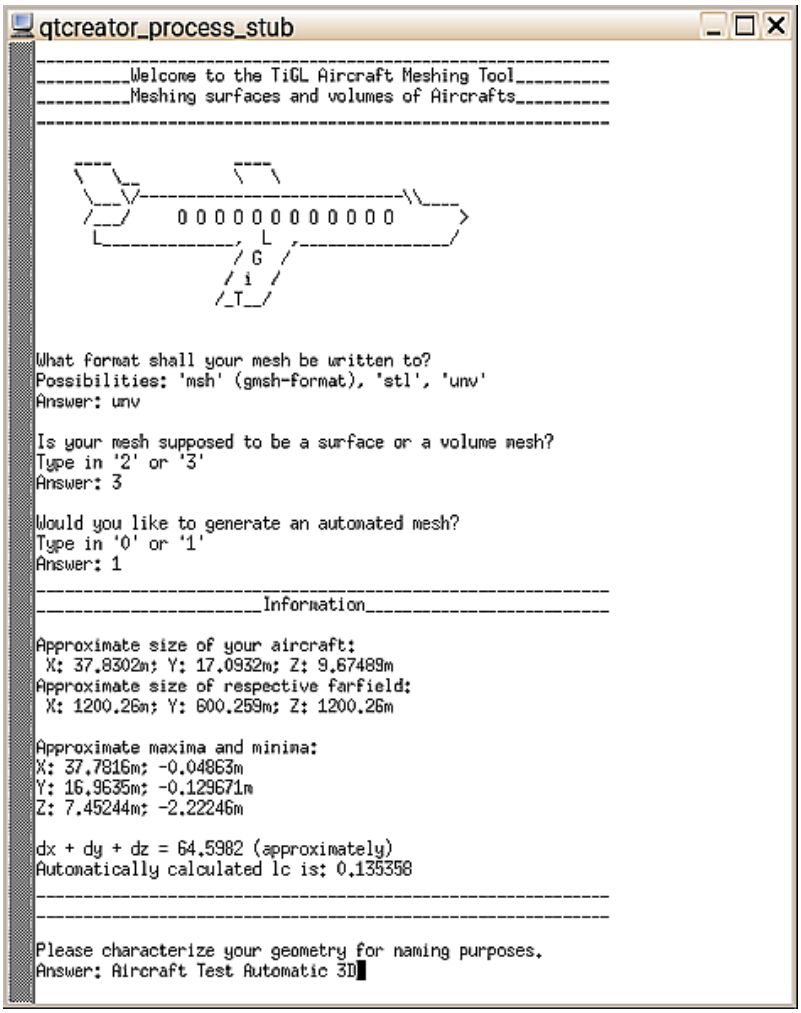


Abbildung A.1-1: Beispielablauf einer automatischen Volumenvernetzung

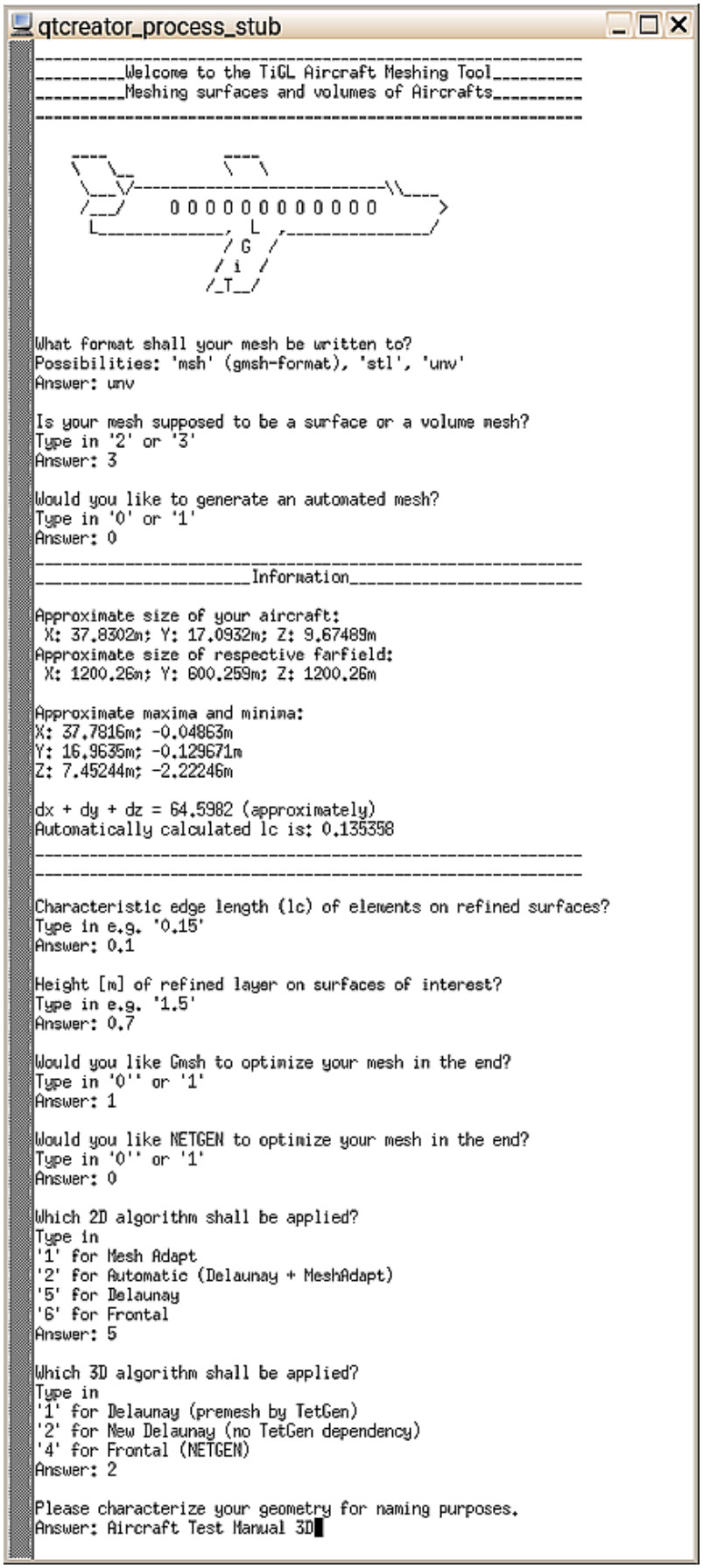


Abbildung A.1-2: Beispielablauf einer manuellen Volumenvernetzung

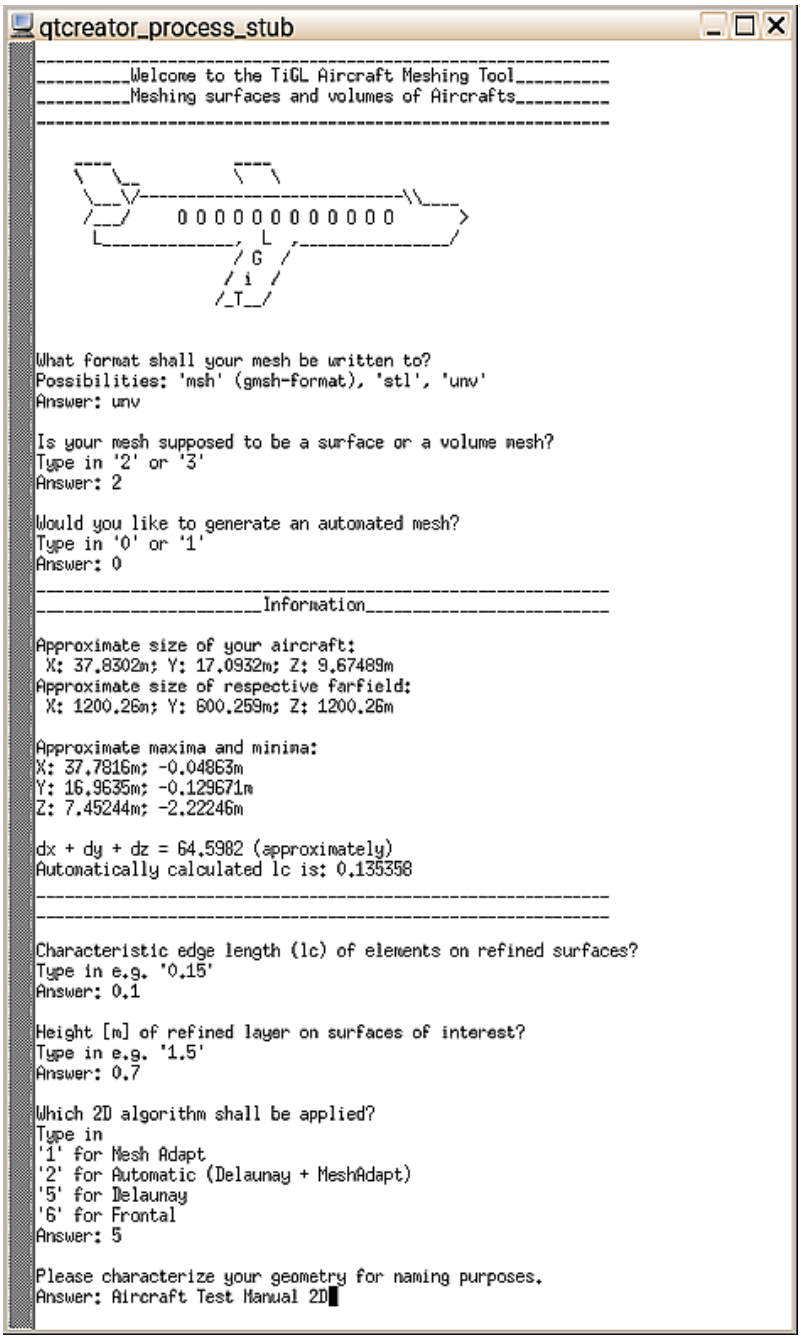


Abbildung A.1-3: Beispielablauf einer manuellen Oberflächenvernetzung

Literatur

[1] Alliez, Pierre u. a. „Variational Tetrahedral Meshing“. In: *ACM Transactions on Graphics,* 24.3 (Juli 2005), S. 617–625. DOI: 10.1145/1073204.1073238.

[2] Baker, Timothy J. „Element Quality in Tetrahedral Meshes“. In: *Proceedings of the 7th International Conference on Finite Element Methods in Flow Problems*. 1989, S. 1018–1024.

[3] Bhagavatula, Phanindra. „Variational Delaunay Triangulation“. Masterth. State University of New York at Stony Brook, Mai 2012. URL: https://ir.stonybrook.edu/xmlui/bitstream/handle/11401/71155/Bhagavatula_grad.sunysb_0771M_10922.pdf.

[4] Blazek, Jiri. „Computational Fluid Dynamics: Principles and Applications“. In: 3. Aufl. Chapter 11. Principles of Grid Generation. Dez. 2015, S. 357–393. ISBN: 978-0080999951. DOI: 10.1016/B978-0-08-099995-1.00011-7.

[5] Bowyer, Adrian. „Computing Dirichlet Tesselations“. In: *The Computer Journal* 24 (Feb. 1981), S. 162–166. DOI: 10.1093/comjnl/24.2.162.

[6] Cavendish, James, Field, David und Frey, William. „An approach to automatic three-dimensional mesh generation“. In: *International Journal for Numerical Methods in Engineering* 21 (Feb. 1985), S. 329–347. DOI: 10.1002/nme.1620210210.

[7] Cutler, Barbara, Dorsey, Julie und McMillan, Leonard. *Simplification and Improvement of Tetrahedral Models for Simulation*. Eurographics Symposium on Geometry Processing. Juli 2004. DOI: 10.1145/1057432.1057445.

[8] Delaunay, Boris Nikolaevich. „Sur la sphère vide. A la mémoire de Georges Voronoï“. In: *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na* 6 (1934), S. 793–800. URL: https://zbmath.org/?q=an:0010.41101%7C60.0946.06.

[9] Du, Qiang, Faber, Vance und Gunzburger, Max. „Centroidal Voronoï Tessellations: Applications and Algorithms“. In: *SIAM Review* 41.4 (1999), S. 637–676. URL: http://www.jstor.org/stable/2653198.

[10] Edelsbrunner, Herbert. *Geometry and Topology for Mesh Generation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2001. ISBN: 9780511530067. DOI: 10.1017/CBO9780511530067.

[11] Frey, Pascal Jean und George, Paul-Louis. *Mesh Generation: Application to Finite Elements*. 2. Aufl. Wiley, Hoboken, 2010. ISBN: 9781848210295. DOI: 10.1002/9780470611166.

[12] George, John Alan. „Computer Implementation of the Finite Element Method“. Technical Report STAN-CS-71-208. Diss. Stanford University, März 1971.

[13] George, Paul-Louis und Borouchaki, Houman. *Delaunay Triangulation and Meshing: Application to Finite Elements*. 1. Aufl. Hermes, 1998. ISBN: 2866016920. URL: https://de.scribd.com/doc/200328710/.

[14] Geuzaine, Christophe. *gmsh.h*. Aufgerufen am: 27.12.2018. URL: https://gitlab.onelab.info/gmsh/gmsh/blob/master/api/gmsh.h.

[15] Geuzaine, Christophe und Remacle, Jean-François. *Gmsh - Choosing the right unstructured algorithm*. Aufgerufen am: 25.10.2018. 2009. URL: http://gmsh.info/doc/texinfo/gmsh.html#Choosing-the-right-unstructured-algorithm.

[16] Geuzaine, Christophe und Remacle, Jean-François. „Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities“. In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), S. 1309–1331. DOI: 10.1002/nme.2579.

[17] Ghoreyshi, M. u. a. „Automated CFD Analysis for the Investigation of Flight Handling Qualities“. In: *Mathematical Modelling of Natural Phenomena* 6.3 (2011), S. 166–188. DOI: 10.1051/mmnp/20116307.

[18] Hickel, Stefan. „Angewandte Strömungssimulation“. Unveröffentlichtes Skript, Technical University of Munich, Aufgerufen am: 03.01.2019. 2014. URL: https://www.aer.mw.tum.de/fileadmin/tumwaer/www/pdf/lehre/angewandte_cfd/V6.pdf.

[19] Kleinert, Jan. „Modellbildung und Simulation 2“. Unveröffentlichtes Skript, University of Applied Science Sankt Augustin. 2018.

[20] Knupp, Patrick. „Remarks on mesh quality“. In: *46th AIAA Aerospace Sciences Meeting and Exhibit* (Jan. 2007). URL: https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/reno07paper.pdf.

[21] Langheinrich, Maximilian. „Evaluation of Gmsh Meshing Algorithms in Preparation of High-Resolution Wind Speed Simulations in Urban Areas“. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2 (2018), S. 559–564. DOI: 10.5194/isprs-archives-XLII-2-559-2018.

[22] Lecheler, Stefan. *Numerische Strömungsberechnung*. 2. Aufl. Springer Vieweg, 2011. ISBN: 9783834815682.

[23] Lembach, Stefan. „Voronoi and Delaunay diagrams“. Bachelorth. Technical University of Munich, Juli 2010. URL: https://www-m10.ma.tum.de/foswiki/pub/Lehrstuhl/StefanKranich/bachelor-thesis.pdf.

[24] Martins, Joaquim und Lambe, Andrew. „Multidisciplinary Design Optimization: A Survey of Architectures“. In: *AIAA Journal* 51 (Sep. 2013), S. 2049–2075. DOI: 10.2514/1.J051895.

[25] Navrátil, Jan. „Aerodynamic shape optimization of an airliner elastic wing“. In: *ITM Web of Conferences* 14 (Jan. 2017), S. 6. DOI: 10.1051/itmconf/20171400006.

[26] Ollivier-Gooch, Carl und Freitag, Lori. „Tetrahedral Mesh Improvement Using Swapping and Smoothing“. In: *International Journal for Numerical Methods in Engineering* 40 (Nov. 1997), S. 3979–4002. DOI: 10.1002/(SICI)1097-0207(19971115)40:21<3979::AID-NME251>3.0.CO;2-9.

[27] Owen, Steven. „A Survey of Unstructured Mesh Generation Technology“. In: *7th International Meshing Roundtable (IMR)* 3 (Mai 2000), S. 239–267. URL: https://www.researchgate.net/publication/2632219.

[28] Papageorgiou, Athanasios u. a. „Multidisciplinary Design Optimization of Aerial Vehicles: A Review of Recent Advancements“. In: *International Journal of Aerospace Engineering* (Mai 2018). Artikel ID: 4258020, S. 21. DOI: 10.1155/2018/4258020.

[29] Rebay, Stefano. „Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm“. In: *Journal of Computational Physics* 106.1 (1993), S. 125–138. DOI: 10.1006/jcph.1993.1097.

[30] Salim, Salim Mohamed und Cheah, Siew Cheong. „Wall y+ Strategy for Dealing with Wall-bounded Turbulent Flows“. In: Bd. 2. März 2009, S. 1–6. URL: http://www.iaeng.org/publication/IMECS2009/IMECS2009_pp2165-2170.pdf.

[31] *SALOME Documentation - Aspect Ratio 3D*. Aufgerufen am: 06.01.2019. URL: http://docs.salome-platform.org/7/gui/SMESH/aspect_ratio_3d_page.html.

[32] Schneiders, Robert. *Meshgeneration Software*. Aufgerufen am: 26.10.2018. URL: http://www.robertschneiders.de/meshgeneration/software.html.

[33] Schöberl, Joachim. „NETGEN An advancing front 2D/3D-mesh generator based on abstract rules“. In: *Computing and Visualization in Science* 1.1 (1997), S. 41–52. DOI: 10.1007/s007910050004.

[34] Schwarze, Rüdiger. *CFD-Modellierung: Grundlagen und Anwendungen bei Strömungsprozessen*. Springer Vieweg, 2012. ISBN: 9783642243776. DOI: 10.1007/978-3-642-24378-3.

[35] Shewchuk, Jonathan Richard. „Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery“. In: *Proceedings of the 11th International Meshing Roundtable (IMR)* (Sep. 2002). URL: https://imr.sandia.gov/papers/imr11/shewchuk1.pdf.

[36] Shewchuk, Jonathan Richard. „Lecture Notes on Delaunay Mesh Generation“. Unveröffentlichtes Skript, University of California at Berkeley, Aufgerufen am: 09.12.2018. Feb. 2012. URL: http://www.wias-berlin.de/people/si/course/files/delnotes.pdf.

[37] Shewchuk, Jonathan Richard. „What Is a Good Linear Finite Element? Interpolation, Conditioning, Anisotropy, and Quality Measures“. Unveröffentlichtes Skript, University of California at Berkeley, Aufgerufen am: 09.12.2018. Dez. 2002. URL: https://people.eecs.berkeley.edu/~jrs/papers/elemj.pdf.

[38] Si, Hang. „TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator“. In: *ACM Trans. Math. Softw.* 41.2 (2015), S. 1–36. DOI: 10.1145/2629697.

[39] Si, Hang. *TetGen: A quality tetrahedral mesh generator and a 3D Delaunay triangulator (Version 1.5 — User's Manual)*. Techn. Ber. 13. Weierstrass Institute for Applied Analysis und Stochastics (WIAS), Jan. 2013. URL: http://wias-berlin.de/software/tetgen/1.5/doc/manual/manual.pdf.

[40] Siggel, Martin u. a. „TiGL - An Open Source Computational Geometry Library for Parametric Aircraft Design“. In: *CoRR* abs/1810.10795 (Okt. 2018). URL: https://arxiv.org/pdf/1810.10795.pdf.

[41] Singh, Lokesh. „A Review on Mesh GenerationAlgorithms“. In: *International Journal of Research in Aeronautical and Mechanical Engineering* 5.5 (2017), S. 7–19. ISSN: 2321-3051. URL: http://www.ijrame.com/wp-content/uploads/2018/02/V5i503.pdf.

[42] Smit, Matthijs Sypkens. *Open Source Meshing Software*. Aufgerufen am: 26.10.2018. URL: http://www.cg.its.tudelft.nl/~matthijss/oss_meshing_software.html.

[43] *TetMesh-GSH3D. A powerful isotropic tet-mesher. Version 4.2*. Aufgerufen am: 14.12.2018. 2010. URL: https://team.inria.fr/gamma3/gamma-software/ghs3d.

[44] Tournois, Jane, Srinivasan, Rahul und Alliez, Pierre. „Perturbing Slivers in 3D Delaunay Meshes“. In: *Proceedings of the 18th International Meshing Roundtable, (IMR)* (Okt. 2009). DOI: 10.1007/978-3-642-04319-2_10.

[45] Watson, David F. „Computing the n-dimensional Delaunay tessellation with application to Voronoï polytopes“. In: *The Computer Journal* 24.2 (Feb. 1981), S. 167–172. DOI: 10.1093/comjnl/24.2.167.