TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR ENERGIETECHNIK

Lehrstuhl für Fluidmechanik

Modulare und objektorientierte Implementierung der Finite Volumen Methode (FVM) zur Simulation kavitierender Strömungen in Turbomaschinen

Mathias J. Bogner

Vollständiger Abdruck der von der Fakultät für Maschinenwesen der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. habil. H. Spliethoff
Prüfer der Dissertation:
1. Univ.-Prof. Dr.-Ing. Dr.-Ing. habil. Dr. h. c. R. Schilling
2. Univ.-Prof. Dr.-Ing. habil. B. Stoffel (i.R.)
Technische Universität Darmstadt

Die Dissertation wurde am 18.11.2010 bei der Technischen Universität München eingereicht und durch die Fakultät für Maschinenwesen am 04.05.2011 angenommen.

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Fluidmechanik der Technischen Universität München.

Mein ganz besonderer Dank gilt Herrn Univ.-Prof. Dr.-Ing. Dr.-Ing. habil. Dr. h. c. Rudolf Schilling, der es mir ermöglichte, diese Arbeit anzufertigen und durch seine wissenschaftliche Anleitung und konstruktive Unterstützung wesentlich zum Gelingen beigetragen hat. Seine lehrreichen Anregungen und die zahlreichen fachlichen Diskussionen mit ihm waren stets eine wertvolle Hilfe während meiner Tätigkeit am Lehrstuhl.

Herrn Univ.-Prof. Dr.-Ing. habil. Bernd Stoffel (i.R.) danke ich für die Übernahme des Koreferats. Ebenso bedanke ich mich bei Herrn Univ.-Prof. Dr.-Ing Hartmut Spliethoff dafür, dass er den Vorsitz der Prüfungskommission übernommen hat.

Gerne möchte ich an dieser Stelle auch die Gelegenheit nutzen, all meinen Kollegen für die stets sehr gute Zusammenarbeit und die ausgeprägte Hilfsbereitschaft zu danken. Insbesondere sei hier mein langjähriger Zimmerkollege Benedikt Flurl genannt, mit dem ich so manche interessante und fruchtbare Diskussion geführt habe.

Ganz herzlicher Dank gebührt auch meiner Familie, die mich immer bestärkt hat, meine beruflichen Ziele zu erreichen. Meiner Frau Claudia gilt mein ganz besonderer Dank für die umfassende Unterstützung und vor allem die Rücksichtnahme während meiner gesamten Ausbildungszeit. Ihr ist diese Arbeit gewidmet.

Regensburg, November 2010

Mathias Bogner

Inhaltsverzeichnis

Verwendete Formelzeichen und Abkürzungen

Abkürzungen

2D	zweidimensional
3D	dreidimensional
cout, clog, cerr	Standard-Ausgabeströme der Konsole
CDS	Central-Differencing-Scheme, Zentrale-Differenz
CEV	Constant Enthalpy Vaporisation, Kavitationsmodell des kommerziellen Strömungslösers CFX-TASCflow
CFD	Computational Fluid Dynamics
CFL	Courant Friedrichs Levi Zahl/ Kriterium
CGNS	CFD General Notation System, Format zum Speichern fluiddynamischer Daten
CRS	Compressed-Rowwise-Storage, komprimiertes Zeilenspeicherformat
CRTP	Curiously Recurring Template Pattern
DNS	Direct Numerical Simulation, Direkte Numerische Simulation
ET	Expression Templates
FDM	Finite Differenzen Methode
FEM	Finite Elemente Methode
FLM	Lehrstuhl für Fluidmechanik, Technische Universität München
FST	Lehrstuhl für Fluidsystemtechnik, Technische Universität Darmstadt
FV, FVM	Finite Volumen, Finite Volumen Methode
HDF5	Hierarchical Data Format 5, Format zum Speichern großer und komplexer Datenmengen
HOS	High-Order-Scheme, hochauflösendes Diskretisierungsschema
KV	Kontrollvolumen
LCL	nichtlineares k-ε-Modell
LES	Large Eddy Simulation, Grobstruktursimulation
LGS	Lineares Gleichungs-System
MIMD	Multiple Instruction Multiple Data
MINMOD	MINimum MODulus, Flux-Limiter-Interpolationsverfahren höherer Ordnung
MPI	Message Passing Interface
MPICH2	MPI Implementierung des Argonne National Laboratory

MPMD	Multiple Program Multiple Data
NIST	National Institute of Standards and Technology
NS3D	Navier Stokes 3D Löser
NVD	Normalized Variable Diagramm
NVF	Normalized Variable Formulation
PDG	Partielle Differential-Gleichung
PVM	Parallel Virtual Machine
Q3D	Quasi-3D, besonders im Zusammenhang mit Euler-Q3D-Verfahren
RANS	Reynolds Averaged Navier Stokes, hier auch stellvertretend für Favré Averaged Navier Stokes
RSM	Reynolds-Stress Modell
SCL	Space Conservation Law
SISD	Single Instruction Single Data
SPMD	Single Program Multiple Data
STL	Standard Template Library
TMP	Template Metaprogrammierung
UDS	Upwind-Differencing-Scheme, Stromauf-Interpolation
UML	Unified Modeling Language
XML	Extensible Markup Language, Aufzeichnungssprache für hierarchisch strukturierbare Textdaten

Lateinische Zeichen

$\vec{a}$, a_i	[-]	3-Vektor
A , a_{ij}	[-]	3x3-Matrix
a_P	[-]	Matrixeintrag
$\vec{b}$, b_i	[-]	3-Vektor
b_{ij}	[-]	3x3-Matrix
b_P	[-]	Eintrag der rechten Seite
c_p	[-]	Druckbeiwert
C_μ	[-]	Modellkonstante
$C_{\varepsilon 1}, C_{\varepsilon 2}$	[-]	Modellkonstanten der k-ε-Modelle
d	[-]	Dimension
D	[m]	Laufraddurchmesser

E	[-]	parallele Effizienz
f_i	[-]	Vektorfeld
f_μ , f_1 , f_2	[-]	Dämpfungsfunktionen
g	$[m/s^2]$	Erdbeschleunigung
H	$[m]$	Förderhöhe
I	[-]	Indexraum
k	$[m^2/s^2]$	turbulente kinetische Energie
L	$[m]$	charakteristisches Längenmaß
n	[-]	charakteristische Problemgröße
n	$[1/min]$	Drehzahl
n_0	[-]	Anzahl der Keime pro Einheitsvolumen Flüssigkeit
n_i	[-]	Einheitsnormalenvektor
n_q	$[1/min]$	spezifische Drehzahl
$NPSH$	$[m]$	Net Positve Suction Head, Netto Energiehöhe
$\dot{m}$	$[kg/s]$	Massenstrom
p	$[Pa]$	statischer Druck
p_t	$[Pa]$	Totaldruck
P	[-]	Anzahl der Prozesse, physikalischer Raum
P	$[Pa]$	mittlerer statischer Druck
Q	$[m^3/h]$	Volumenstrom
r_i	$[m]$	Ortsvektor
R	$[m]$	Radius der Dampfblasen
$\dot{R}$	$[m/s]$	Wachstums-/ Kollapsgeschwindigkeit der Dampfblasen
Re_T , Re_y	[-]	turbulente Reynoldszahl
s	[-]	Skalar, Lauflänge
s_i	$[m]$	Zellverbindungsvektor
S	[-]	parallele Beschleunigung
S	$[m^2]$	Flächeninhalt
S_{ij}	$[1/s]$	Deformationstensor
$\dot{S}$	$[kg/(m^3 s)]$	allgemeiner Quellterm
t	[-]	Skalar
t	[°]	Teilungswinkel

t	$[s]$	physikalische Zeit
T	$[s]$	Rechenzeit, Zeitintervall
u	$[m/s]$	Umfangsgeschwindigkeit
$\overline{u}_t$	$[m/s]$	wandtangeniale Geschwindigkeit
u_i	$[m/s]$	Geschwindigkeitsvektor
x_i	$[m]$	Ortskoordinate
y	$[m]$	Wandabstand der ersten Rechenzelle
w	$[m/s]$	Relativgeschwindigkeit
w_i	$[m/s]$	Vektor der Relativgeschwindigkeit

Griechische Zeichen

α	[-]	Dampfvolumenfraktion
β	[-]	Blending-Faktor
Γ	$[kg/(ms)]$	allgemeiner Diffusionskoeffizient
Δ	[-]	Delta, Differenzbetrag
∂	[-]	Differentialoperator
δ_{ij}	[-]	Kronecker-Symbol
ε	$[m^2/s^3]$	Dissipationsrate
λ	[-]	Interpolationsfaktor
μ	$[kg/(ms)]$	dynamische Viskosität
ν	$[m^2/s]$	kinematische Viskosität
μ_T	$[kg/(ms)]$	turbulente dynamische Viskosität
ρ	$[kg/m^3]$	Dichte
$\sigma_k, \sigma_\varepsilon$	[-]	Modellkonstanten
τ_w	$[N/m^2]$	Wandschubspannung
ϕ	[-]	allgemeines Skalar
ω	[-]	Relaxationsfaktor
Ω	$[m^3]$	Volumen
Ω_{ij}	$[1/s]$	Rotationstensor

Hochgestellte Zeichen

´	Schwankungsgröße, geänderte Größe
*	modifizierter Term, Referenz
+	dimensionslose turbulente Größe
c	auf die konvektiven Flüsse bezogen
d	auf die diffusiven Flüsse bezogen
expl	explizit
g	auf den Gradienten bezogen
impl	implizit
new	neue Werte
old	alte Werte
r	Rang
u	auf die Geschwindigkeit bezogen
ϕ	die Variable ϕ betreffend

Tiefgestellte Zeichen

0	Referenz
bnd	Zählindex am Feldrand
c	Zellen-/ Kontrollvolumenindex
C	central, zentral
D	downwind, stromab
f, fc	Flächenindex, Flächenindex des Kontrollvolumens *c*
i, j, k	Zählindex
iC, iF, iN	Zell-, Flächen-, Knotenindex
in	Inlet, Eintritt
incipient	beginnend
k	Komponente
l	Liquid, flüssige Phase
last	vorheriger Index
m	meridional
m	Master-Zellindex der Fläche *f*
max	Maximum
n	normal
n	Zählindex im Feldinneren

n, n_f	Knotenindex, Knotenindex der Fläche f
NB	Nachbareintrag der Matrix
$next$	nächster Index
opt	optimal
P	Diagonaleintrag der Matrix
ref	Referenz
s	Slave-Zellindex der Fläche f
$trans$	transfer
U	upwind, stromauf
v	Vapor, dampfförmige Phase
w	auf die Relativgeschwindigkeit bezogen
x	x-Komponente
y	y-Komponente
z	z-Komponente

Kopfnoten

~	normalisierter Wert, dimensionslose Variable
–	interpolierter Wert, Mittelwert

Zusammenfassung

Die vorliegende Arbeit stellt den Entwurf eines flexiblen Werkzeugs zur Simulation von Strömungsvorgängen mit der FVM vor. Besonderer Wert wird dabei vor allem auf das modulare und objektorientierte Design des Programmpaketes gelegt, wodurch sowohl die Sicherheit als auch die Flexibilität der Implementierung wesentlich gesteigert werden kann.

Im Zuge der Konzepterstellung werden zunächst die Anforderungen, die die unterschiedlichen Anwendergruppen an eine moderne Simulationssoftware stellen, zusammengetragen und analysiert. Es werden fundamentale Softwareeigenschaften wie die Implementierbarkeit neuer physikalischer Modelle oder die Parametrierbarkeit bereits vorhandener Module betrachtet. Das sich ergebende Anforderungsprofil wird in ein Schichtenmodell zur Strukturierung der Software übertragen.

Die Erkenntnisse der vorangehenden Konzeptfindungsphase dienen als Ausgangspunkt für den Entwurf der grundlegenden Datenstrukturen. Insbesondere auf die bequeme Handhabung der in den fluidmechanischen Gleichungen auftretenden Tensorarithmetik sowie die effiziente Programmparallelisierung wird geachtet. Die Detaillierung der verschiedenen Bausteine und Komponenten für die Repräsentation der numerischen Netze und der Erhaltungsgleichungen schließen die programmiertechnische Umsetzung der erarbeiteten Konzepte ab.

Der resultierende Softwareentwurf wird schließlich anhand des klassischen Modellentwicklungsprozesses validiert. Der Prozess bildet die Anforderungen der verschiedenen Anwendergruppen gut ab. Als konkretes Beispiel dient die Entwicklung eines Kavitationsmodells, das die kavitierende Strömung in hydraulischen Strömungsmaschinen für den stationären Fall simulieren kann. Das Modell beruht auf sphärischer Blasendynamik und ist in der Lage kavitierende Strömungen auch anderer Flüssigkeiten als Wasser zu beschreiben. Zudem beschränkt das Modell das Temperaturniveau der Strömung nicht auf die Raumtemperatur. Die Einführung des effektiven Dampfdrucks erweitert die Anwendbarkeit des Modells außerdem auf Flüssigkeiten, die noch gelöste Gase enthalten.

Das Kavitationsmodell wird bei der Simulation der Strömung durch eine Radialpumpe mit spezifischer Schnellläufigkeit $n_q = 26\ \mathrm{1/min}$ getestet. Als zu förderndes Fluid kommt Glykol zum Einsatz, dessen Stoffwerte sich bereits bei Raumtemperatur deutlich von denen des Wassers unterscheiden. Die berechneten $NPSH_{3\%}$-Werte stimmen sehr gut mit der Messung überein.

Die Eignung des Kavitationsmodells für unterschiedliche Temperaturniveaus wird in einem weiteren Testbeispiel, bei dem die kavitierende Strömung durch eine Heizungsumwälzpumpe simuliert wird, untersucht. Als Fördermedium dient hier unbehandeltes Wasser. Auch in diesem Fall zeigt sich wieder eine hervorragende Übereinstimmung zwischen Simulation und Experiment.

1 Einleitung

1.1 Problemstellung

Numerische Verfahren zur Simulation von Strömungsvorgängen werden heute auf breiter Front sowohl in der Wissenschaft als auch in der Industrie entwickelt und eingesetzt. Die rasante Entwicklung der Rechnerkapazitäten und die gleichzeitig günstigen Anschaffungskosten der Computer haben bewirkt, dass immer komplexere Problemstellungen angepackt werden. Dabei zeigt der Trend genauso stark in Richtung detaillierter aufgelöster Geometrien, wie er auch in Richtung genauer modellierter Strömungsphänomene zeigt.

Eine moderne Simulationssoftware sieht sich mit sehr unterschiedlichen Anforderungsprofilen konfrontiert. Dem Forscher zum Beispiel ist oft die Physik des zugrundeliegenden Rechenmodells wichtiger als die anfallende Rechenzeit oder die Stabilität. Ganz anders gewichtet ein Entwicklungsingenieur in der Industrie diese beiden Eigenschaften. So bewertet er tendenziell eher die Effizienz und Stabilität des Rechenmodells höher als die darin enthaltene Physik. Für ihn bildet ein CFD-Code nur das Mittel zum Zweck, mit dem er das Design seines Produktes bewerten will. Nicht zuletzt erzeugt der Programmierer einer Simulationssoftware ein weiteres Anforderungsprofil, in dem die Flexibilität und Sicherheit des Softwaredesigns bei der Implementierung im Vordergrund steht. Dabei ist aber nicht ausgeschlossen, dass die einzelnen Gruppen miteinander interagieren und sich die Gewichtungsfaktoren verändern. So beinhaltet der klassische Entwicklungsprozess von Simulationssoftware bereits alle Profile. Wird ein physikalisches Modell entwickelt, muss dieses implementiert und danach validiert werden. Treten bei der Validierung Fehler auf, wird eine erneute Iterationsschleife durchlaufen. Stellt sich im weiteren Verlauf heraus, dass das Modell unzureichend genau ist, um die Problemstellung abzubilden, so muss das Modell selbst noch einmal verbessert werden. Je sicherer eine Implementierung ist, desto einfacher können unterschiedliche Modellkonfigurationen in den Iterationsschleifen der Entwicklungsschritte ausprobiert werden. Während des gesamten Prozesses wird der Forscher oder Modellentwickler abwechselnd zum Programmierer und Anwender der Software. Natürlich ist er dabei kein Anwender im klassischen Sinne. Das Beispiel zeigt aber trotzdem die Verzahnung der einzelnen Anforderungsgruppen untereinander.

Ein weiteres Beispiel, zur Verzahnung der einzelnen Gruppen lässt sich konstruieren, wenn der Anbieter der Simulationssoftware als die eine und der User der Simulationssoftware als die andere Gruppe betrachtet wird. Beide Gruppen profitieren stark von einer modular aufgebauten und dynamisch konfigurierbaren Software. So kann der Anbieter die Wünsche und Erfordernisse seines Kunden analysieren und eine kundenspezifisch zugeschnittene Distribution erstellen. Programmfeatures, die der Kunde nicht benötigt, muss der somit nicht erwerben. Zusätzlich ist es dem Kunden möglich eine bestehende Installation durch Zukauf neuer Module nachträglich zu erweitern. Der Anbieter kann außerdem Programmupdates modulweise verteilen, sodass nur diejenigen Kunden betroffen sind, deren Distribution das zu erneuernde Modul beinhaltet. Dieses Vorgehen ist in anderen Softwarebereichen nicht neu, hat sich aber im CFD-Bereich noch kaum durchgesetzt.

Dass eine Simulationssoftware nun alle Anforderungen gleichzeitig erfüllt, erscheint kaum realisierbar zu sein. Der Einsatz moderner objektorientierter Programmiertechniken macht dies aber dennoch möglich. Die vorliegende Arbeit beschäftigt sich deshalb mit der modularen und objektorientierten Implementierung der FVM und wählt zur Validierung des Konzepts als strömungstechnisch herausfordernde Problemstellung die Simulation kavitierender Strömungen in Turbomaschinen.

1.2 Stand des Wissens

1.2.1 Diskretisierungstechniken

Die Finite Elemente Methode (FEM) und die Finite Volumen Methode (FVM) stellen zwei weit verbreitete Ansätze zur Lösung der fluidmechanischen Grundgleichungen sowie anderer Transportgleichungen auf komplexen Geometrien dar, vgl. FERZIGER und PERIĆ [23]. Einen weiteren sehr direkten und unkomplizierten Lösungsansatz bietet darüber hinaus die Finite Differenzen Methode (FDM). Sie bleibt allerdings, im Gegensatz zu den beiden Anderen, auf einfache Geometrien und strukturierte Netze beschränkt. Die drei unterschiedlichen Methoden verfolgen den gemeinsamen Ansatz, die betroffenen Differentialgleichungen zunächst zu diskretisieren und anschließend numerisch zu verarbeiten. Ursprünglich ist die FEM in der Strukturmechanik beheimatet, die FVM dagegen in der Fluidmechanik. Beide Methoden sind inzwischen aber erweitert worden und können auch Problemstellungen des jeweils anderen Gebietes lösen. FLURL et al. [25], [26] verwendet beispielsweise die FVM zur Struktursimulation bei der Fluid-Struktur-Interaktion. Umgekehrt verwendet zum Beispiel MÜLLER [60] die FEM zur numerischen Strömungssimulation. Trotz dieser Erweiterungen bleiben beide Methoden aber oft ihren historischen Wurzeln treu.

Bei der FEM wird das Lösungsgebiet zunächst mit einem numerischen Netz versehen, das aus Knoten und Elementen besteht. Die Lösung innerhalb eines jeden Elements wird durch lokale Ansatzfunktionen beschrieben. Die Lösung des gesamten Rechengebietes erhält man durch Superposition der Einzellösungen der Elemente. Die Differentialgleichungen werden mit Testfunktionen multipliziert und über das gesamte Lösungsgebiet integriert. Verschiedene Testfunktionen führen zu unterschiedlichen Diskretisierungen. In der Strömungsmechanik wird oft die Galerkin Methode, auch bekannt als Methode der gewichteten Residuen, verwendet. Die Ansatzfunktionen der FEM entsprechen dann den Wichtungsfunktionen der Galerkin Methode.

Auch bei der FVM wird das Lösungsgebiet zunächst mit einem numerischen Netz versehen, wobei dieses nun aus Knoten und Zellen besteht. Die Zellen definieren Kontrollvolumen (KV), die sich gegenseitig nicht überlappen und das Lösungsgebiet vollständig ausfüllen. Abhängig von der Art, wie die Kontrollvolumen aus dem Netz konstruiert werden, unterscheidet man zwischen zellzentrierten und elementbasierten Verfahren. Zellzentrierte Verfahren verwenden die Zellen des Netzes direkt als KV. Elementbasierte Verfahren konstruieren die KV dagegen um die Knoten des Netzes herum. Elementbasierte Verfahren kommen beispielsweise bei HUURDEMAN [39] oder REXROTH [71] zum Einsatz, zellzentrierte Verfahren bei ZWART [108], SKODA [86] oder EINZINGER [22]. Die FVM nutzt auf natürliche Weise direkt die Erhaltungseigenschaften der fluidmechanischen Grundgleichungen aus. Die Differentialgleichungen werden über

die einzelnen KV integriert, und die Divergenzterme dann mithilfe des Gauß'schen Integralsatzes in Flüsse umgewandelt. Zwei aneinander grenzende KV teilen sich immer eine gemeinsame Grenzfläche. Der Fluss, der das eine KV verlässt, tritt exakt mit gleichem Betrag in das andere KV ein. Die FVM ist deshalb auch lokal massenerhaltend. Die FEM dagegen kann nur globale Massenerhaltung gewährleisten. Die Diskretisierungsgenauigkeit der FVM lässt sich mit relativ einfachen Mitteln erhöhen. So können beispielsweise die konvektiven Flüsse an den KV-Grenzen durch möglichst exakte Ansätze interpoliert werden, vgl. JUSUF [43]. In der Strömungsmechanik ist die FVM sehr weit verbreitet. Viele kommerzielle CFD-Codes, wie CFX, FLUENT oder STAR-CCM+, basieren auf dieser Methode. Auch die vorliegende Arbeit verwendet die zellzentrierte FVM zur Lösung der angegebenen Strömungsprobleme.

Die FDM ist die älteste Methode zur numerischen Lösung partieller Differentialgleichungen. Sie arbeitet direkt auf den Knoten des numerischen Netzes. Mithilfe der Taylor-Reihenentwicklung wandelt die Methode die Differentialgleichungen direkt in Differenzengleichungen um, siehe FORSYTHE und WASOW [28]. Mit der Ordnung der Glieder, die bei der Taylor-Reihenentwicklung berücksichtigt werden, nimmt die Genauigkeit der numerischen Rechnung zu. Gleichzeitig steigt aber die Anzahl der Knotenwerte, die zur Darstellung der einzelnen Terme nötig sind. Das erhöht den numerischen Aufwand, vgl. SMITH [88]. Auf strukturierten Netzen ist die FDM sehr effizient und einfach. Die Netzlinien dienen hier als lokale Koordinatenlinien. Der große Nachteil der FDM ist, dass sie das Erhaltungsprinzip nicht auf natürliche Weise unterstützt und auf einfache Geometrien beschränkt bleibt. Das prinzipielle Vorgehen zur Implementierung der FDM kann unter anderem bei FERZIGER und PERIĆ [23] nachgelesen werden.

1.2.2 Modellierungstechniken

Das physikalische Modell, das für die Beschreibung des Geschwindigkeitsfeldes verwendet wird, liefert ein wichtiges Unterscheidungskriterium bei der Klassifizierung unterschiedlicher numerischer Verfahren zur Strömungssimulation. Abhängig vom Komplexitätsgrad kann das Strömungsfeld entweder mit den Potentialgleichungen, den Euler-Gleichungen oder den Navier-Stokes-Gleichungen modelliert werden. Sowohl die Euler-, wie auch die Navier-Stokes-Gleichungen werden zur Lösung komplexer Strömungsprobleme in hydraulischen Turbomaschinen herangezogen. Die Potentialtheorie ist für diese Art von Strömungsproblemen ungeeignet, da sie zu starke Vereinfachungen beinhaltet.

Die Euler-Gleichungen nehmen die Strömung als reibungsfrei an. Dadurch kann die wandnahe Grenzschicht in der Strömung nicht abgebildet werden. Außerdem ist es unmöglich, mit den Euler-Gleichungen einen turbulenzbedingten Strömungsabriss zu simulieren. Vorteilhaft bei der Euler-Modellierung ist aber, dass vor allem in Wandnähe mit wesentlich gröberen numerischen Netzen gearbeitet werden kann. Für schnelle Abschätzungen und Relativvergleiche bei der Optimierung von Strömungsmaschinen sind Eulerverfahren deshalb sehr gut geeignet. Zunächst kamen aufgrund der fehlenden Rechnerleistung quasi-dreidimensionale (Q3D) Euler-Verfahren zum Einsatz. Die Theorie dazu stammt von WU [105]. SCHILLING [76], [77] wendet das Euler-Q3D-Verfahren erfolgreich bei der Simulation hydraulischer Strömungsmaschinen an. Als effizientes Werkzeug speziell für die Schaufeloptimierung wird es bei THUM und

SCHILLING [94], sowie bei SCHILLING et al. [81] genutzt. RIEDEL [74] verwendet ein voll dreidimensionales Euler-Verfahren und untersucht damit die Rotor-Stator-Wechselwirkung in hydraulischen Maschinen.

Die genaueste Modellierung des Strömungsfeldes ist durch die Navier-Stokes-Gleichungen gegeben. Diese berücksichtigen sowohl Reibung als auch Turbulenz. Da die Strömung in Turbomaschinen generell durch hohe Reynolds-Zahlen bestimmt wird, sind die Navier-Stokes-Gleichungen hier am besten geeignet. Einige Beispiele hierzu finden sich in BADER [5], SKODA [86] und EINZINGER [22]. Je nach Detaillierungsgrad unterscheidet man zwischen der Direkten Numerischen Simulation (DNS), der Grobstruktur-Simulation (LES) sowie der Lösung der Reynolds gemittelten Navier-Stokes Gleichungen (RANS). DNS-Verfahren lösen die Navier-Stokes-Gleichungen unmittelbar und enthalten keinerlei Modellierung für die Turbulenz. Zur Auflösung aller turbulenten Skalen, muss hier eine feine räumliche wie auch zeitliche Diskretisierung gewählt werden, vgl. beispielsweise MOIN und MAHESH [59]. Die DNS stellt deshalb einen extrem hohen Rechenaufwand dar. Die nächste Abstraktionsstufe der Modellierung bilden LES Verfahren. Dabei wird lediglich die instationäre Entwicklung der energietragenden großen Wirbel, durch die das globale Strömungsfeld beeinflusst wird, aufgelöst, die kleinskaligen turbulenten Strukturen werden dagegen modelliert. Die LES zeichnet sich im Vergleich zur DNS durch wesentlich geringere Rechenzeiten aus. Für die meisten realen Anwendungen ist sie aber immer noch zu aufwändig. RANS Verfahren weisen einen noch höheren Abstraktionsgrad auf. Sie beruhen auf einer statistischen Betrachtung der Turbulenz, wodurch auch stationäre Simulationen möglich werden. Durch die Reynolds- bzw. die Favré-Mittelung entsteht in den Impulsgleichungen ein unbekannter Term, der als Reynolds-Spannungstensor bezeichnet wird. Entsprechend der Beschreibung der Reynoldsspannungen, lassen sich RANS basierte Turbulenzmodelle in zwei Hauptgruppen aufteilen, vgl. SCHUSTER [85]. Reynolds-Spannungsmodelle (RSM) berechnen den Reynolds-Spannungstensor mit rein algebraischen Ausdrücken oder differentiell mit zusätzlichen Transportgleichungen. Wirbelviskositätsmodelle dagegen verfolgen den Wirbelviskositätsansatz nach BOUSSINESQ [13] und koppeln die Komponenten des Reynolds-Spannungstensors mit dem mittleren Strömungsfeld. Ein weit verbreitetes und etabliertes lineares Wirbelviskositätsmodell ist das Standard k-ε-Modell von LAUNDER und SPALDING [50], das zwei zusätzliche Transportgleichungen für die turbulente kinetische Energie k und für die Dissipationsrate ε löst. Ein weiteres Wirbelviskositätsmodell ist das nicht-lineare LCL-Modell von LIEN et al. [53]. Beide Modelle zeigen bei der Anwendung auf hydraulische Turbomaschinen gute Resultate und kommen in dieser Arbeit zur Anwendung.

Bei der Lösung der Navier-Stokes-Gleichungen wird zwischen dichte- und druckbasierten Verfahren unterschieden. Erstere berechnen mit der Kontinuitätsgleichung das Dichtefeld, letztere das Druckfeld. Naturgemäß eignen sich dichtebasierte Verfahren besonders zur Simulation hoch kompressibler Strömungen. Im inkompressiblen Fall sind sie dagegen nur bedingt einsetzbar. Pseudokompressibilitätsverfahren begegnen dem Problem der konstanten Dichte, indem sie eine sogenannte Pseudoschallgeschwindigkeit einführen. Sind die Verfahren konvergiert, hat die Pseudoschallgeschwindigkeit dann keinen Einfluss mehr auf das Ergebnis. Durch diesen Ansatz können Pseudokompressibilitätsverfahren dieselben Lösungsalgorithmen einsetzen, die für kompressible Strömungen üblich sind. Druckbasierten Verfahren sind speziell für inkompressible Strömungsprobleme ent-

wickelt worden. Sie bestimmen das Druckfeld direkt über eine sogenannte Druckkorrekturgleichung, die aus der Kontinuitätsgleichung gewonnen wird. Dabei wird das Druckfeld immer so berechnet, dass neben der Impulserhaltung auch die Kontinuität gewahrt bleibt. Die bekannteste Form der Druckkorrekturgleichung wird als SIMPLE-Algorithmus (Semi Implizit Procedure for Pressure-Linked Equations) bezeichnet, siehe PATANKAR [66]. Eine verbesserte und stabilere Variante des SIMPLE-Algorithmus stellt der SIMPLEC-Algorithmus von VAN DOORMAL und RAITHBY [97] dar. Neben der Berechnung inkompressibler Strömungen können druckbasierte Verfahren aber auch für die Berechnung kompressibler Strömungen mit niedrigen bis moderaten Machzahlen eingesetzt werden, vergleiche BADER [5], LILEK [54] und EINZINGER [22]. Hervorzuheben ist hierbei insbesondere, dass der Übergang zwischen inkompressibler und kompressibler Strömung keinerlei zusätzliche Stabilisierungsmaßnahmen erfordert. Im Rahmen dieser Arbeit werden nur dichtebeständige Strömungen untersucht. Im Hinblick auf die kavitierenden Strömungen bedeutet das, dass die beiden Einzelphasen für sich genommen als inkompressibel modelliert werden.

1.2.3 Programmiertechniken

Im Bereich des wissenschaftlichen Hoch- und Höchstleistungsrechnens haben sich die Programmiersprachen C, C++ und FORTRAN etabliert. C und FORTRAN sind beide prozessorientiert, sodass hier die funktionale Aufgliederung der Algorithmen im Fordergrund steht. C++ ist dagegen objektorientiert. Zu den Grundsätzen der objektorientierten Programmierung gehören Abstraktion, Datenkapselung und Vererbung. Die Paradigmen erleichtern die Erstellung von flexibler, modularer sowie sicherer Software. C++-Programme waren in der Vergangenheit meist noch langsamer als vergleichbare C- oder FORTRAN-Programme. Das hat sich aber durch die modernen C++-Compiler stark geändert. Heutzutage stehen C++-Programme in ihrer Laufzeit ihren prozessorientierten Gegenstücken in nichts mehr nach. Das frei verfügbare Simulations-Framework open∇FOAM [64] oder der kommerzielle CFD-Code StarCCM+ sind Beispiele für objektorientiert programmierte Software.

Modernes Programmieren beinhaltet auch die Verwendung statischer und dynamischer Bibliotheken als gekapselte funktionale Module. Statische Bibliotheken werden während des Erstellens fest in das jeweilige Programm eingebunden, dagegen bleiben dynamische Bibliotheken als externe Bestandteile erhalten. Sie werden entweder sofort beim Programmstart oder verzögert erst während der Programmlaufzeit geladen. Insbesondere durch verzögert geladene Bibliotheken erhöht sich die Flexibilität einer Software stark, da diese während der Laufzeit nachkonfiguriert werden kann.

Die Programmiersprache C++ bietet mit der Template-Metaprogrammierung (TMP) ein passendes Werkzeug um die Effizienz und Anwendbarkeit einer Programmbibliothek weiter zu steigern. MEYERS [58] beschreibt TMP als den Vorgang, template-basierte C++-Programe zu schreiben, die wären der Compilezeit ausgeführt werden. Dies wird im englischsprachigen Raum häufig als compile-time computation (CC) bezeichnet. Die Ausgabe eines solchen Metaprogrammes besteht dabei aus gewöhnlichem Quelltext, der dann in einem weiteren Schritt vom Compiler zum ausführbaren Programm übersetzt wird. Die CC nutzt die Tatsache, dass bestimmte Berechnungen schon bei der Erstellung eines Programms durchgeführt werden können. Die zugehörigen Ergebnisse sind dann

fertig im Quelltext integriert. Dadurch fällt zur Laufzeit des Programms kein Rechenaufwand mehr an, und es ist wesentlich schneller als ein vergleichbares Programm ohne TMP. Natürlich erhöht sich der Zeitaufwand zur Programmerstellung bei der TMP, da die Arbeit nun vom Compiler erledigt werden muss. Die Performance unterschiedlicher Compiler in Bezug auf TMP wird zum Beispiel in ABRAHAMS und GURTOVOY [1] analysiert. Im Allgemeinen überwiegt aber natürlich der Gewinn zur Laufzeit den Mehraufwand zur Kompilierzeit ganz wesentlich. Das liegt schon in der Tatsache begründet, dass Software wesentlich öfter ausgeführt als erstellt wird. KLIMANIS [46] nutzt die generische Programmierung exzessiv zur Implementierung der FEM. HÄRDTLEIN [35] beleuchtet ausführlich die unterschiedlichen Aspekte der Expression Template (ET)-Programmierung in Bezug auf deren Einsatz zur Lösung partieller Differentialgleichungen. Hier ermöglicht die ET-Programmierung vor allem eine effiziente und zugleich sichere und einfache Implementierung der auftretenden Tensorarithmetik.

1.2.4 Simulation kavitierender Strömungen in Turbomaschinen

Die numerische Untersuchung kavitierender Strömungen in hydraulischen Maschinen bleibt aufgrund des extrem hohen Rechenaufwands auf Modellierungen im stationären Relativsystem beschränkt. Im Folgenden werden einige Anwendungen zur Simulation kavitierender Pumpenströmungen beschrieben.

ATHAVALE et al. [4] berechnen die kavitierende Strömung in einer Axialpumpe und einer Radialpumpe sowie in einem Inducer zur Förderung von flüssigem Sauerstoff. Sie verwenden hierzu das sogenannte Full Cavitation Model. Das Kavitationsmodell berücksichtigt insbesondere die turbulenten Druckschwankungen bei beginnender Kavitation durch die Reduktion des lokalen statischen Drucks um $\Delta p = 0.39 \cdot \rho \cdot k$. Die Ergebnisse sind allerdings nur qualitativ erfasst und nicht experimentell validiert.

VISSER [102] simuliert die kavitierende Strömung in einer Kreiselpumpe mit und ohne Spirale mit dem Constant Enthalpy Vaporisation (CEV)-Modell des kommerziellen Strömungslösers CFX-TASCflow. Der sich ergebende $NPSH_{3\%}$-Wert für verschiedene Volumenströme zeigt gute Übereinstimmung mit den experimentell verfügbaren Daten. Auch DUPONT und OKAMURA [20] verwenden das CEV-Modell. Sie vergleichen es mit zwei weiteren kommerziellen Kavitationsmodellen. Die einzelnen Ergebnisse weisen hier allerdings erhebliche Unterschiede auf.

WURSTHORN [107] verwendet zur Simulation der kavitierende Strömung in einer 2D-Pumpe spezifischer Drehzahl n_q = 18 $1/min$ das barotrope Kavitationsmodell des kommerziellen CFD-Codes STAR-CD. Das Modell ist ursprünglich von SCHMIDT et al. [82] zur Berechnung von Einspritzdüsen entwickelt worden. WURSTHORN muss die Parameter des Kavitationsmodells anpassen, um sinnvolle und stabile Simulationen durchführen zu können. Der Förderhöhenabfall ist im Vergleich mit dem Experiment reproduzierbar, allerdings fallen die Kavitationszonen zu klein aus.

HOFMANN et al. [38] untersuchen die stationär kavitierende Strömung anhand einer Radialpumpe mit 2D-gekrümmten Schaufeln. Sie verwenden hierbei den kommerziellen Strömungslöser FINE/TURBO. Das verwendete barotrope Kavitationsmodell ist bei COUTIER-DELGOSHA et al. [16] ausführlich beschrieben. Verglichen mit dem Experi-

ment sind die errechneten NPSH3%-Werte generell höher und die Abweichungen nehmen hin zu niedrigen Volumenströmen tendenziell zu.

FROBENIUS et al. [29] bzw. SCHILLING und FROBENIUS [80] simulieren die kavitierende Strömung in einer Kreiselpumpe niedriger spezifischer Drehzahl mit dem auf sphärischer Blasendynamik beruhenden Kavitationsmodell von SAUER und SCHNERR [75] und [84]. Der berechnete Förderhöhenabfall stimmt bei Nennvolumenstrom gut mit den experimentellen Ergebnissen überein. Die Ausdehnung des Kavitationsgebietes ist in der numerischen Simulation allerdings auch hier kleiner als im Experiment.

NOHMI et al. [63] sowie NOHMI und GOTO [62] untersuchen dieselbe Laufradgeometrie sowohl experimentell als auch numerisch. Für die Strömungssimulation ziehen sie CFX-TASCflow mit dem CEV-Kavitationsmodell heran. Ihre Ergebnisse zeigen, dass die gemessenen und berechneten Förderhöhenabfallkurven im Nennpunkt zwar gut passen, bei Überlast aber stark voneinander abweichen. Die Autoren erwähnen außerdem große Konvergenzprobleme bei den Simulationen.

FROBENIUS [30] sowie FROBENIUS und SCHILLING [31] korrigieren für stationäre Simulationen die Terme des Modells von SAUER, die das Blasenwachstum und den Blasenkollaps regeln, und wenden das modifizierte Modell erfolgreich zur Simulation der kavitierenden Strömung in einer Kreiselpumpe spezifischer Schnellläufigkeit n_q = 26 1/min an. Der berechnete Förderhöhenabfall kann mit experimentellen Untersuchungen, die die Druckseitenkavitation als maßgebliche Ursache für den Förderhöhenabfall sehen, bestätigt werden. Zusätzlich zeigt FROBENIUS, dass die Strömung im Radseitenraum der Kreiselpumpe maßgeblichen Einfluss auf die korrekte Vorhersage des Förderhöhenabfalls hat.

SCHILLING und BOGNER [78] erweitern das von FROBENIUS entwickelte Modell weiter. Sie führen den effektiven Dampfdruck ein und verwenden das Kavitationsmodell auch für andere Flüssigkeiten als Wasser. Bei den Simulationen der kavitierenden Strömung wird die bereits von FROBENIUS untersuchte Radialpumpe mit spezifischer Schnellläufigkeit n_q = 26 1/min herangezogen. Als zu förderndes Fluid kommt Glykol zum Einsatz, dessen Stoffwerte sich bereits bei Raumtemperatur deutlich von Wasser unterscheiden. Die errechneten NPSH3%-Werte stimmen sehr gut mit der Messung überein. SCHILLING und BOGNER [79] prüfen die Eignung des Kavitationsmodells zusätzlich für unterschiedliche Temperaturniveaus und mit unbehandeltem Wasser. Zur Validierung wird die kavitierende Strömung durch eine Heizungsumwälzpumpe berechnet. Auch für diesen Fall zeigt sich eine hervorragende Übereinstimmung mit dem Experiment. Die Ergebnisse der Simulationen der beiden Kreiselpumpen sind bei BOGNER et al. [11] zusammengefasst.

1.3 Zielsetzung und Aufbau

Ziel der Arbeit ist die modulare und objektorientierte Implementierung der FVM. Es wird ein Framework, das als CFD-Tool bezeichnet wird, zur numerischen Simulation von Strömungsproblemen entworfen. Der Programmentwurf zielt insbesondere darauf ab, den eingangs erwähnten Spagat zwischen den Anforderungen der unterschiedlichen Anwendergruppen zu schaffen. Dabei wird zur Darstellung der Klassenentwürfe und der Programmabläufe verstärkt auf standardisierte Methoden wie der Unified Modeling Language (UML) zurückgegriffen.

Das zu definierende Konzept für das CFD-Tool soll sowohl Sicherheit als auch Flexibilität bei der Implementierung gewährleistet, da diese Eigenschaften gerade auch im wissenschaftlichen Bereich große Vorteile bei der Generierung neuer mathematischer Modelle darstellen. Der konkrete Entwurf soll deshalb anschließend anhand der Implementierung eines Kavitationsmodells, das die kavitierende Strömung in Strömungsmaschinen für den stationären Fall simulieren kann, validiert werden. Das Modell beruht auf sphärischer Blasendynamik und ist in der Lage kavitierende Strömungen auch anderer Flüssigkeiten als Wasser zu simulieren. Das Temperaturniveau der Strömung ist dabei nicht auf die Raumtemperatur beschränkt. Die Einführung eines effektiven Dampfdrucks erweitert außerdem die Nutzbarkeit des Modells auf Flüssigkeiten, die noch gelöste Gase enthalten.

Kapitel 2 analysiert zunächst die Rahmenbedingungen, die sich für eine Simulationssoftware im Umfeld der FVM ergeben. Es diskutiert die Grundsätze der Methode anhand einer allgemeinen skalaren Transportgleichung. Danach untersucht es die Anforderungen, denen moderne Simulationstools gerecht werden müssen. Der Fokus liegt besonders auf den unterschiedlichen Strategien des Softwareentwurfs, die die Unterteilung numerischer Anwendungen in funktionale Komponenten ermöglichen.

Kapitel 3 beschäftigt sich mit der Grobstruktur des Softwaredesigns. Es stellt das Datenmodell zur Verarbeitung allgemein unstrukturierter oder polyedrischer Netze vor. Außerdem geht das Kapitel auf die Implementierung von Expression Templates zur nativen und bequemen Verarbeitung der bei der Diskretisierung der fluidmechanischen Gleichungen zwangsläufig auftretenden Tensorarithmetik ein. Im Anschluss folgen Modelle zur Parallelisierung.

Kapitel 4 detailliert das zuvor entworfene Grobdesign der Softwarekomponenten. Es kombiniert die einzelnen Klassenbausteine und präsentiert die programmiertechnisch umgesetzte Repräsentation von Geometrie und Gleichungen. Darüber hinaus stellt es Schemataklassen zur Berechnung der geometrischen Größen und zur Diskretisierung der Gleichungen, sowie Algorithmen-Klassen zur Ablaufsteuerung der Iterationsschleifen vor.

Kapitel 5 validiert den Softwareentwurf am konkreten Anwendungsfall. Dies geschieht mithilfe des oben erwähnten Kavitationsmodells. Als Beispiel wird die kavitierende Strömung in einer radialen Förderpumpen mit spezifischer Schnellläufigkeit $n_q = 26$ 1/min, sowie einer Heizungsumwälzpumpe simuliert.

Kapitel 6 fasst die Ergebnisse der Arbeit zusammen.

2 Numerische Simulation mit der FVM

Die FVM ist ein mächtiges Werkzeug zur numerischen Simulation strömungsmechanischer Problemstellungen. Letzten Endes entscheidet aber erst die programmiertechnische Umsetzung der Methode darüber, wie erfolgreich die resultierende Simulationssoftware wirklich ist. Das vorliegende Kapitel beschäftigt sich deshalb mit den Rahmenbedingungen, die sich für eine Simulationssoftware im Umfeld der FVM ergeben. Zunächst diskutiert es die Grundsätze der Methode anhand des Beispiels einer allgemeinen skalaren Transportgleichung. Danach untersucht es die unterschiedlichen Anforderungen, denen moderne Simulationstools gerecht werden müssen. Die beiden Abschnitte über die Strategien beim Softwareentwurf und die Unterteilung der numerischen Anwendungen in funktionale Komponenten schließen das Kapitel ab.

2.1 Diskretisierung partieller Differentialgleichung

Die FVM ist ein weit verbreitetes Verfahren im Bereich der Fluidmechanik. Die Methode basiert auf der Diskretisierung der partiellen Differentialgleichungen (PDG) und deren anschließender numerischer Lösung. Sie kann ohne Weiteres auch auf komplexe Problemstellungen angewandt werden. Die Methode wurde speziell zur Lösung der fluidmechanischen Gleichungen entwickelt und entspricht von ihrem Grundsatz stark den allgemeinen Erhaltungsprinzipien der Strömungsmechanik. Abgesehen davon wird die FVM ebenfalls zur Lösung strukturmechanischer Probleme genutzt, vgl. hierzu FLURL et al. [25] und [26]. Trotzdem bleibt sie bisher aber meist ihren historischen Wurzeln treu.

Eine numerische Lösungsmethode beinhaltet laut FERZIGER und PERIĆ [23] unter Anderem als Bestandteile das mathematische Modell, die Koordinaten- und Basisvektorsysteme, das numerische Netz und die finiten Approximationen. Der folgende Abschnitt widmet sich besonders der Diskretisierung der Gleichungen mit der FVM. Als Koordinaten- und Basisvektorsystem kommt ausschließlich und ohne Einschränkung das kartesische System zum Einsatz.

Bei der Diskretisierung der PDG wird oft zwischen räumlicher und zeitlicher Diskretisierung, sowie der Gleichungsdiskretisierung unterschieden, vgl. open∇FOAM [64]. Die räumliche Diskretisierung bezieht sich auf die Auflösung des Rechengebietes bzw. der Rechengeometrie durch das numerische Netz mit einer finiten Anzahl an Knoten. Korrespondierend dazu beschreibt die zeitliche Diskretisierung die Auflösung des Zeitflusses als finite Anzahl von Zeitschritten. Die Diskretisierung der Gleichungen schließlich behandelt die Abbildung des jeweiligen mathematischen Modells im Rechner als Satz von algebraischen Gleichungssystemen.

Grundsätzlich können die verschiedenen Erhaltungsgleichungen für Masse, Impuls und Energie als spezielle Varianten einer allgemeinen Transportgleichung betrachtet werden. Formuliert mit dem Skalar ϕ als Transportgröße und dem allgemeinen Diffusionskoeffizienten Γ sowie dem volumenbezogenen Quellterm $\dot{S}$, lautet deren differentielle Form:

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{Zeitterm}+\underbrace{\frac{\partial(\rho u_i\phi)}{\partial x_i}}_{Konvektion}=\underbrace{\frac{\partial}{\partial x_i}\left(\Gamma\frac{\partial\phi}{\partial x_i}\right)}_{Diffusion}+\underbrace{\dot{S}}_{Quellterm}. \tag{1.1}$$

In Gleichung (1.1) bezeichnet zudem ρ die Dichte und u_i den Geschwindigkeitsvektor. Die zeitliche Änderung und der geschwindigkeitsabhängige konvektive Transport auf der linken Seite der Gleichung stehen mit dem diffusiven Austausch und dem Quellterm auf der rechten Seite im Gleichgewicht. Der Quellterm fasst dabei sämtliche Einflüsse, die über zeitliche Änderung, Konvektion und Diffusion hinaus wirksam sind, zusammen.

Bei der FVM werden die partiellen Transportgleichungen nun als erstes über ein allgemeines KV Ω integriert. Die skalare Transportgleichung (1.1) schreibt sich somit:

$$\int_\Omega\frac{\partial(\rho\phi)}{\partial t}d\Omega+\int_\Omega\frac{\partial(\rho u_i\phi)}{\partial x_i}d\Omega=\int_\Omega\frac{\partial}{\partial x_i}\left(\Gamma\frac{\partial\phi}{\partial x_i}\right)d\Omega+\int_\Omega\dot{S}d\Omega. \tag{1.2}$$

Gleichung (1.2) kann mithilfe des Gaußschen Integralsatzes weiter umgeformt werden. Dieser besagt, dass die Divergenz der vektoriellen Feldgröße f_i bezogen auf das KV Ω gleich dem Fluss dieser Größe normal zur KV-Oberfläche *S* ist:

$$\int_\Omega\frac{\partial f_i}{\partial x_i}d\Omega=\int_S f_i n_i dS, \tag{1.3}$$

wobei der Oberflächen-Normalenvektor n_i per Definition stets nach außen gerichtet ist. Anwendung des Gaußschen Integralsatzes auf die konvektiven und diffusiven Flüsse der Gleichung (1.2) ergibt:

$$\int_\Omega\frac{\partial(\rho\phi)}{\partial t}d\Omega+\int_S\rho u_i n_i\phi dS=\int_S\Gamma\frac{\partial\phi}{\partial x_i}n_i dS+\int_\Omega\dot{S}d\Omega. \tag{1.4}$$

Eine weitere Vereinfachung von Gleichung (1.4) kann durch Verwendung der Leibnitz-Regel erzielt werden. Bewegt sich die KV-Oberfläche *S* mit der Geschwindigkeit w_i, dann lautet die Leibnitz-Regel für eine skalare Feldgröße f:

$$\frac{d}{dt}\int_\Omega f d\Omega=\int_\Omega\frac{\partial f}{\partial t}d\Omega+\int_S f w_i n_i dS. \tag{1.5}$$

Wird Gleichung (1.5) in Gleichung (1.4) substituiert, so ergibt sich als endgültige integrale Form der Transportgleichung für die skalare Größe ϕ:

$$\frac{d}{dt}\int_{\Omega}\rho\phi\, d\Omega + \int_{S}\rho\phi\left(u_i - w_i\right)n_i dS = \int_{S}\Gamma\frac{\partial\phi}{\partial x_i}n_i dS + \int_{\Omega}\dot{S}d\Omega\,. \tag{1.6}$$

Bis zu diesem Schritt sind keine approximativen Modellierungen herangezogen worden. Die ursprüngliche Gleichung (1.1) ist ausschließlich durch mathematisch exakte Operationen in eine für die FVM günstigere Form gebracht worden. Die umgeformte Gleichung (1.6) entspricht somit genau der Ausgangsgleichung (1.1).

Bei der FVM wird das gesamte Rechengebiet nun mit nicht-überlappenden KV Ω_c vollständig aufgefüllt. Gleichung (1.6) wird dann über die einzelnen KV des Rechengebietes integriert. Die FVM schränkt die Form der KV grundsätzlich nicht ein. Es ist jedoch von Vorteil, wenn diese konvex sind, und der Mittelpunkt dadurch innerhalb eines jeden KV liegt. Die Definition der KV mithilfe des numerischen Netzes wird später in Kapitel 3.1 behandelt.

Bezogen auf die diskreten KV Ω_c und die zugehörigen KV-Oberflächen S_f des numerischen Netzes lautet Gleichung (1.6) dann:

$$\frac{d}{dt}\int_{\Omega_c}\rho\phi\, d\Omega + \sum_{f}\int_{S_f}\rho\phi\left(u_i - w_i\right)n_i dS = \sum_{f}\int_{S_f}\Gamma\frac{\partial\phi}{\partial x_i}n_i dS + \int_{\Omega_c}\dot{S}d\Omega\,, \tag{1.7}$$

wobei *c* den Index des jeweiligen KV und *f* die Indices der zugehörigen KV-Oberflächen spezifizieren.

Die Diskretisierung startet nun mit Anwendung der Mittelpunktsregel. Diese postuliert, dass die einzelnen Integranden durch repräsentative Mittelwerte angenähert werden können. Geht man davon aus, dass die repräsentativen Werte im Mittelpunkt des KV Ω_c bzw. in den Mittelpunkten der zugehörigen KV-Oberflächen S_{f_c} vorliegen, so ergibt sich für Gleichung (1.7):

$$\frac{d\left(\rho\phi\Omega\right)_c}{dt} + \sum_{f_c}\left(\rho\phi\left(u_i - w_i\right)n_i\right)_f S_f = \sum_{f_c}\left(\Gamma\frac{\partial\phi}{\partial x_i}n_i\right)_f S_f + \dot{S}_c\,\Omega_c\,. \tag{1.8}$$

Diese Annahme entspricht grundsätzlich einer Genauigkeit zweiter Ordnung, d. h. dass sich Fehler bei einer Verdoppelung der Auflösung vierteln.

In Gleichung (1.8) kann zusätzlich noch der Massenstrom $\dot{m}$ normal zu den KV-Oberflächen eingesetzt werden:

$$\frac{d(\rho\phi\Omega)_c}{dt} + \sum_{f_c}(\dot{m} - \dot{m}_w)_f \phi_f = \sum_{f_c}\left(\Gamma \frac{\partial \phi}{\partial x_i} n_i\right)_f S_f + \dot{S}_c \Omega_c . \tag{1.9}$$

Gleichung (1.9) beschreibt nun die diskrete Form der skalaren Transportgleichung. Das Problem wurde auf die Bestimmung der einzelnen Größen an den Mittelpunkten der KV sowie den Mittelpunkten der KV-Oberflächen reduziert. Die meisten Werte im Rechengebiet werden bei der FVM in den KV gespeichert. Sind Werte an den KV-Oberflächen nötig, wie das bei den konvektiven und diffusiven Flüssen der Fall ist, so müssen diese von den benachbarten KV interpoliert werden. Dadurch entsteht für jedes Kontrollvolumen eine algebraische Gleichung, die die Abhängigkeit der Lösungsvariablen im KV-Mittelpunkt von den unmittelbaren Nachbar-KV ausdrückt. Werden diese Gleichungen zusammengefasst, so entsteht ein lineares Gleichungssystem (LGS), das mit vorgegebenen Randbedingungen gelöst werden kann. Ob für die Lösung des LGS dabei implizite oder explizite Lösungsalgorithmen zur Anwendung kommen, hängt direkt von der Art der Diskretisierung ab. Bei stationären Problemen verschwindet die zeitliche Ableitung. Deshalb kommen in solchen Fällen üblicherweise implizite Verfahren zum Einsatz. Bei instationären Problemen hingegen können abhängig von den speziellen Anforderungen sowohl implizite als auch explizite Lösungsalgorithmen verwendet werden. Explizite Verfahren sind allerdings hinsichtlich der möglichen Zeitschrittweite an das CFL-Kriterium gebunden. Ist der Zeitschritt zu groß gewählt, sind diese Lösungsverfahren nicht mehr stabil, siehe FERZIGER und PERIĆ [23].

Die einzelnen Interpolationsschemata, die im CFD-Tool für die konvektiven und diffusiven Flüsse zur Verfügung stehen, die Diskretisierungsschemata für die zeitliche Ableitung sowie die Approximationsschemata für die Gradienten werden später behandelt.

2.2 Beurteilung numerischer Verfahren

Für die Beurteilung eines numerischen Lösungsalgorithmus sind verschiedene Kriterien erforderlich, die dessen Eigenschaften möglichst eingängig und fassbar ausdrücken. Im Bereich der Strömungsmechanik haben sich hierfür einige Kriterien etabliert, die im Folgenden aufgelistet werden, vergleiche JUSUF [43]. Die einzelnen Punkte sind teilweise miteinander verwandet und sollten deshalb in ihrer Gesamtheit betrachtet werden.

Konsistenz

Bei der Diskretisierung des mathematischen Modells entstehen Abbruchfehler. Ein numerisches Verfahren wird dann als *konsistent* bezeichnet, wenn bei verschwindenden örtlichen bzw. zeitlichen Schrittweiten die Abbruchfehler ebenfalls verschwinden. Ein numerisches Verfahren, dessen Diskretisierung einen großen Abbruchfehler aufweist, wird auch als diffusiv bezeichnet. Das liegt darin begründet, dass sich der Abbruchfehler durch eine erhöhte Diffusivität der Strömung bemerkbar macht, siehe beispielsweise FERZINGER und PERIĆ [23].

Stabilität

Eine numerische Lösung ist immer mit kleinen Fehlern behaftet. Wenn diese Fehler im Laufe eines Lösungsprozesses nicht verstärkt sondern gedämpft werden, dann heißt das Verfahren *stabil*. Oft werden Verfahren bei Verwendung von Diskretisierungsschemata hoher Ordnung instabil. Eine Reduktion der Ordnung kann die Verfahren stabilisieren, geht jedoch zu Lasten der Lösungsgenauigkeit.

Konvergenz

Ein numerisches Verfahren wird dann als *konvergent* bezeichnet, wenn bei steigender örtlicher bzw. zeitlicher Auflösung die Lösung des diskretisierten Modells gegen die exakte Lösung des mathematischen Modells strebt. Von *Konvergenz* wird auch gesprochen, wenn das Residuum des diskreten LGS mit steigender Iterationszahl des iterativen Lösungsverfahrens abnimmt.

Konservativität

Die mathematischen Modelle für strömungsmechanische Problemstellungen leiten sich von den grundlegenden physikalischen Erhaltungsprinzipien für Masse, Impuls und Energie ab. Kann ein Lösungsverfahren diese Erhaltungsprinzipien sowohl lokal als auch global erfüllen, wird es als konservativ bezeichnet. Die in der vorliegenden Arbeit zum Einsatz kommende FVM folgt direkt dem Erhaltungsprinzip und ist deshalb ein konservatives Verfahren.

Genauigkeit

Numerische Lösungen sind immer mit systematischen Fehlern behaftet. Diese entstehen durch die Modellierung des Strömungsproblems, durch die Ordnung der Diskretisierung und die räumliche und zeitliche Auflösung.

2.3 Anforderungen an Simulationswerkzeuge

Bei der Anforderungsanalyse für ein Simulations-Tool im sowohl wissenschaftlichen als auch industriellen Umfeld ergeben sich drei wesentliche Anwenderrollen, der klassische Simulationsingenieur, der Forscher und der Programmierer. Letzterer ist zwar kein Anwender im herkömmlichen Sinne, wird aber in diesem Zusammenhang zu den Anwendertypen hinzugezählt, da auch er Anforderungen an die Software stellt. Der Simulationsingenieur verwendet das CFD-Programm als reines Werkzeug und ist vor allem an Programmstabilität, Rechenperformance und Modellverfügbarkeit interessiert, wohingegen für den Programmentwickler die Erweiterbarkeit und Wartbarkeit des Quelltextes im Fordergrund stehen. Der Forscher mischt die Anforderungen der beiden genannten Anwendergruppen, da er im Wesentlichen neue Modelle entwickelt und diese deshalb sowohl implementieren als auch validieren muss. Bild 2-1 stellt dar, dass eine CFD-Software von den drei vorgestellten Anwenderklassen unter sehr verschiedenen Gesichtspunkten betrachtet wird, deutet gleichzeitig aber an, dass die Abgrenzung

zwischen den Gruppen verschmieren kann, oder ein Angehöriger einer Gruppe durch Rollentausch auch ein neues Anforderungsprofil an die Software stellen kann.

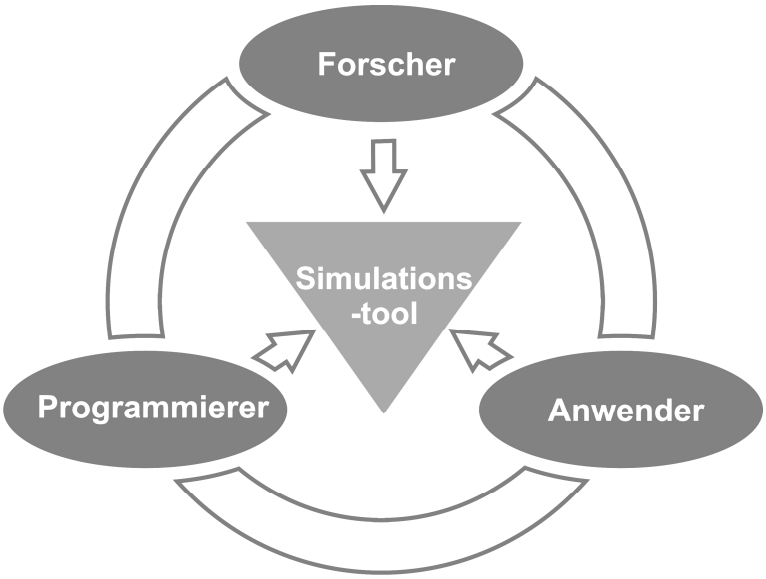


Bild 2-1: Die Simulationssoftware im Fokus unterschiedlicher Anforderungsgruppen

Betrachtet man den Modellentwicklungsprozess in einem CFD-Code, ist der Wechsel des Anforderungsprofils für den Modellentwickler sogar typisch. Denn ausgehend von der Definition des reinen physikalischen Modells tritt der Entwicklungsprozess in eine klassische Iterationsschleife aus Modellimplementierung und Modellvalidierung ein, die schließlich bei erfolgreicher Validierung in der Modellauslieferung endet. Der Modellentwickler durchläuft entsprechen die Rollen Forscher, Softwareentwickler und Anwender, vergleiche Bild 2-2.

Das genannte Szenario zeigt deutlich, dass sehr unterschiedliche Anforderungen an ein Simulations-Tool gestellt werden. Ein starres Softwaredesign kann unmöglich allen sich ergebenden Ansprüchen gerecht werden, weshalb die Auslegung des CFD-Tools als flexibles Framework schon unter diesen grundlegenden Gesichtspunkten als sinnvoll erscheint. Die Interaktionsmöglichkeiten, der einzelnen Anwendergruppen mit den unterschiedlichen Komponenten des CFD-Tools werden in Kapitel 2.5 definiert.

2.4 Aspekte für den Softwareentwurf

Die Implementierung der FVM für die numerische Simulation komplexer strömungstechnischer Problemstellungen stellt, wie die Implementierung aller numerischer Lösungsverfahren, einen enormen programmiertechnischen sowie programmierorganisatorischen Aufwand dar. Die Wartung und Pflege, sowie die Erweiterung der resultierenden Simulationssoftware gestaltet sich wegen ihrer Komplexität und ihrem Umfang sehr aufwändig. Im wissenschaftlichen Bereich sind die prozessorientierten Programmiersprachen FORTRAN und C für die Programmierung numerischer Verfahren weit verbreitet. Dabei stellen diese Sprachen aber nicht die technischen Mittel zur Verfügung,

die für ein modernes Softwaremanagement notwendig wären. Eine strukturierte, modulare und übersichtliche Programmierweise ist in FORTRAN und C nur mit einem hohen Maß an Programmierdisziplin möglich. Das hat zur Folge, dass die bearbeitete Software mit der Zeit meist immer unübersichtlicher wird. Die Lesbarkeit und Übersichtlichkeit ist jedoch für die Wartung, Änderung oder Erweiterung jeder Software immens wichtig. Oft muss dann in regelmäßigen Abständen ein scharfer Schnitt vollzogen werden, der ein komplettes Re-Design der Simulationssoftware bedeutet. Vor diesem Hintergrund erscheint die Verwendung einer Programmiersprache wie C++, die die Implementierung großer Projekte nativ unterstützt, wesentlich sinnvoller. Durch die Objektorientierung forciert sie von Grund auf die Datenkapselung und die Wiederverwendbarkeit von Softwarekomponenten und erleichtert so das Erstellen klar strukturierter Programme.

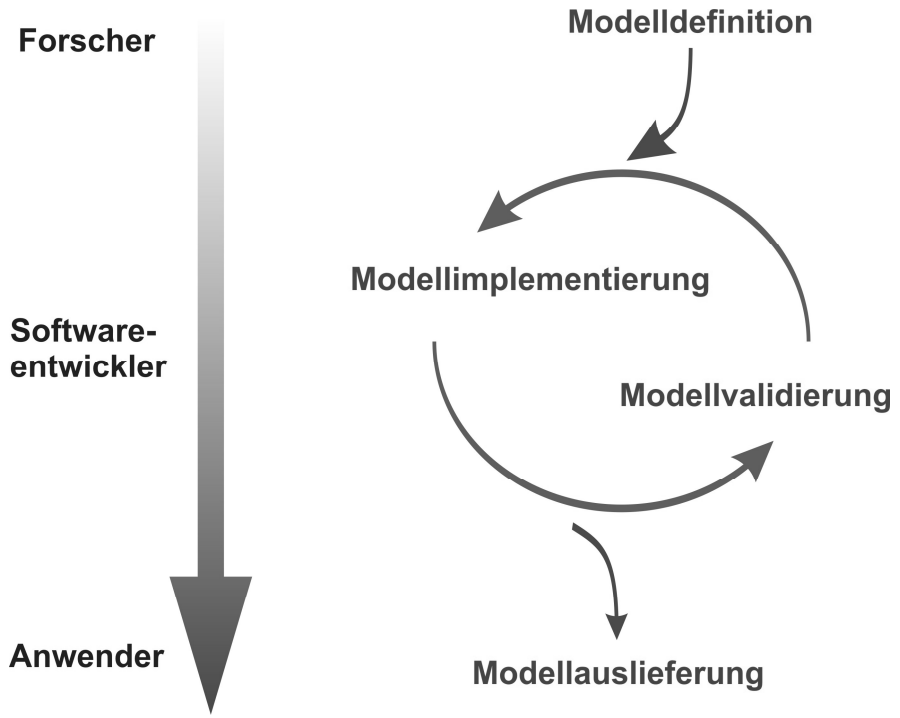


Bild 2-2: Der Modellentwicklungsprozess bezogen auf die genannten Anforderungsgruppen

Aus Sicht des Softwareprojekts sollte die Implementierung der FVM vor allem effizient, sicher sowie leicht wartbar sein. Für die Nachhaltigkeit der Programmierung ist wichtig, die einzelnen Softwarekomponenten außerdem möglichst flexibel und wieder verwendbar zu gestalten. Die Effizienz und Sicherheit wird durch die Verwendung moderner Programmiertechniken erreicht. C++ ermöglicht mit der TMP beispielsweise die Erstellung von ET für die Tensorarithmetik, siehe Kapitel 3.3. Die Wartbarkeit des Quelltextes resultiert natürlich direkt aus der Qualität des Softwareentwurfs. Die klare Strukturierung des Programms in geeignete funktionale Komponenten ist hier entscheidend. Klassen sollten generell über schlanke und möglichst allgemeine Schnittstellen verfügen. Die Daten der Klassen sollten außerdem immer möglichst stark gekapselt sein und dürfen nur über ausgewiesene Zugriffsfunktionen vom Anwender erreichbar sein. Das stellt die Konsistenz der enthaltenen Daten sicher und schafft zudem Übersichtlichkeit beim

Programmieren. Wo nicht erforderlich sollte auch nicht unnötig abgeleitet werden. Zwei Klassen, die sich in ihrer Funktionalität kaum überschneiden, gehören in der Regel auch nicht zusammen. Die Flexibilität bzw. Wiederverwendbarkeit von Komponenten hängt vor allem von ihrer Schnittstellendefinition ab. Diese sollte möglichst von vornherein auch zukünftige Anforderungen berücksichtigen. Da oft schwer absehbar ist, wie sich die Anforderungen an eine Komponente mit der Zeit verändern, sollten ihre Schnittstellen so gestaltet werden, dass diese leicht ergänz- bzw. erweiterbar sind. Eine bestehende Schnittstellendefinition darf sich aber niemals verändern. Das stellt sicher, dass nicht plötzlich ganze Programmmodule durch eine modifizierte Schnittstelle unbrauchbar werden.

Eines der wirkungsvollsten geistigen Werkzeuge, um Komplexität zu beherrschen, ist das hierarchische Ordnen, d. h. das Anordnen verwandter Konzepte in einer Baumstruktur. Ein Programm kann häufig als eine Menge von Bäumen oder als gerichteter azyklischer Graph von Klassen organisiert werden, siehe STROUSTROUP [90] zum Programmieren in C++. Eine grobe interne Aufteilung zum Ausbilden solcher Strukturen für numerische Anwendungen könnte sich beispielsweise in die Geometriedaten, die Gleichungsdaten, die Lösungsalgorithmen für das lineare Gleichungssystem, die Steueralgorithmen zum Initialisieren und Iterieren, die Parallelisierung, die Einlese- und Ausgabealgorithmen sowie in die Speicherverwaltung aufgliedern.

2.5 Funktionale Komponenten numerischer Anwendungen

Die Funktionalität numerischer Software kann nach ACOSTA [2] durch mehrere horizontal übereinander liegende Schichten ausdrückt werden. Er vergleicht die einzelnen Lagen in seinem Modell dabei mit einem Eisberg, bei dem der größte Teil unter der Wasseroberfläche liegt und nur die oberste Spitze für die Nutzer sichtbar ist. Das Modell von ACOSTA ist in Bild 2-3 dargestellt. Die unterste Lage bildet die Kommunikations-Schicht, die die funktionalen Komponenten zur Parallelisierung der Simulationssoftware und die grundlegenden Datenstrukturen enthält. Darüber befindet sich die Algebra-Schicht mit den Algorithmen zur Diskretisierung der Gleichungen. Die Details der Kommunikationsschicht bleiben vor der Algebra-Schicht dabei komplett verborgen. Sie greift auf die Funktionalität der Kommunikationsschicht ausschließlich über die vorhandenen Schnittstellen zu. Über der Algebra-Schicht liegt wiederum die Solver-Schicht mit den Algorithmen zum Lösen der LGS. Wie zuvor sind die Details der darunterliegenden Algebra-Schicht für die Solver-Schicht nicht einsehbar. Der Zugriff auf die Funktionalität der Algebra-Schicht erfolgt wieder über die spezifizierten Schnittstellen. Als letzte und oberste Lage setzt ACOSTA die User-Schicht, die über die notwendigen Schnittstellen zur Steuerung der Software durch den Anwender verfügt. Die User-Schicht ist als einzige Lage nach außen sichtbar und stellt die numerische Software als Ganzes dar.

Das Konzept von ACOSTA bildet eine vertikale Hierarchie ab, deren Ebenen von unten nach oben immer komplexere Komponenten beinhalten. Die Interaktion zwischen den Ebenen beschränkt sich dabei ausschließlich auf die jeweils unmittelbar darüber und darunter liegende Nachbarebene. ACOSTAs Modell folgt einer klassischen Strategie zur Untergliederung komplexer Systeme in immer einfachere Bausteine. Die Grundbausteine sind dabei stark gekapselt und möglichst allgemein gehalten. Dadurch wird ihre Wiederverwendbarkeit erleichtert. Das Konzept ist mit objektorientierten Strukturen leicht

umsetzbar. Die grundlegenden Prinzipien sind auch im Softwareentwurf des CFD-Tools enthalten.

ACOSTAs Eisbergmodell sieht keine horizontalen Schnittstellen für die funktionalen Schichten vor, durch die eine Interaktion zwischen Nutzer und Software möglich wäre. Derartige Schnittstellen können aber den Nutzwert einer Simulationssoftware erheblich steigern. Deshalb ist das Grundkonzept von ACOSTA für den Softwareentwurf des CFD-Tools neu überdacht worden. Bild 2-4 zeigt das resultierende Stufenkonzept des CFD-Tools. Die einzelnen Schichten sind reduziert und neu zugewiesen worden. Die Unterteilung der Schichten lautet nun von oben nach unten *Solver++*, *Solver++Plugins* sowie *Solver++Library*. Die Schichten verkörpern die verschiedenen funktionalen Komponenten des CFD-Tools, also das ausführbare Programm, die einzelnen integrierbaren Module zur Gleichungsdiskretisierung und Geometriemodifikation sowie die Programmbibliothek mit den grundlegenden Bausteinen zur Speicherung und Verwaltung der Daten. Wie zuvor bei ACOSTA bauen die einzelnen Schichten aufeinander auf und sind deshalb vertikal miteinander verzahnt. Den einzelnen Schichten stehen zur Interaktion verschiedene Anwendergruppen gegenüber. Die Anwendergruppen werden durch den Standardnutzer, den CFD-Experten und den Programmierer gebildet. Die Interaktionsmöglichkeiten zwischen den Softwareschichten und den Anwendergruppen ist 1 zu 1 zugeordnet, das heißt der Standardnutzer kommuniziert nur mit der *Solver++*-Schicht, der CFD-Experte mit der *Solver++Plugins*-Schicht und der Programmierer mit der *Solver++-Library*-Schicht. Die Anwendergruppen können dabei aber untereinander, wie in Kapitel 2.3 beschrieben, wechseln.

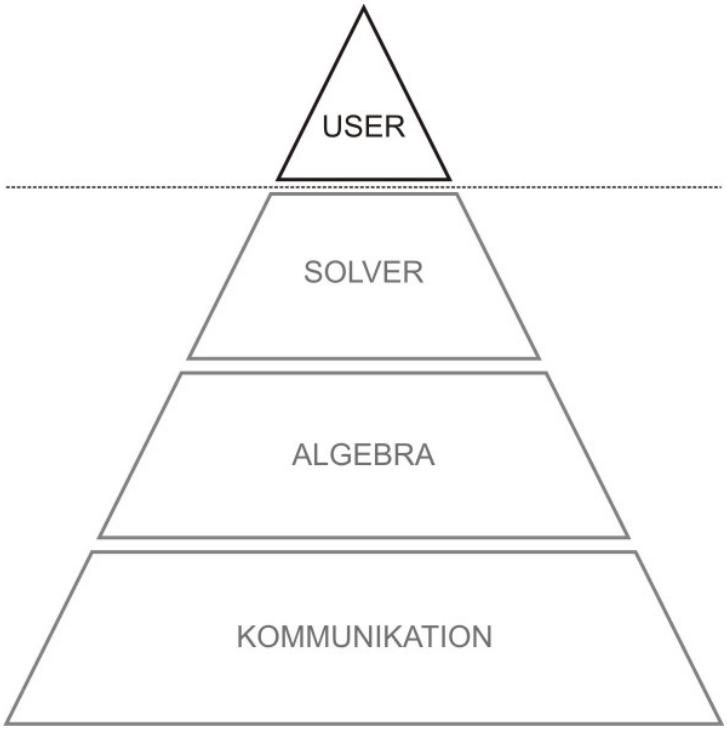


Bild 2-3: Das Schichtenmodell nach ACOSTA [2] angelehnt an einen Eisberg, nur die oberste Spitze als Benutzerschicht ist sichtbar

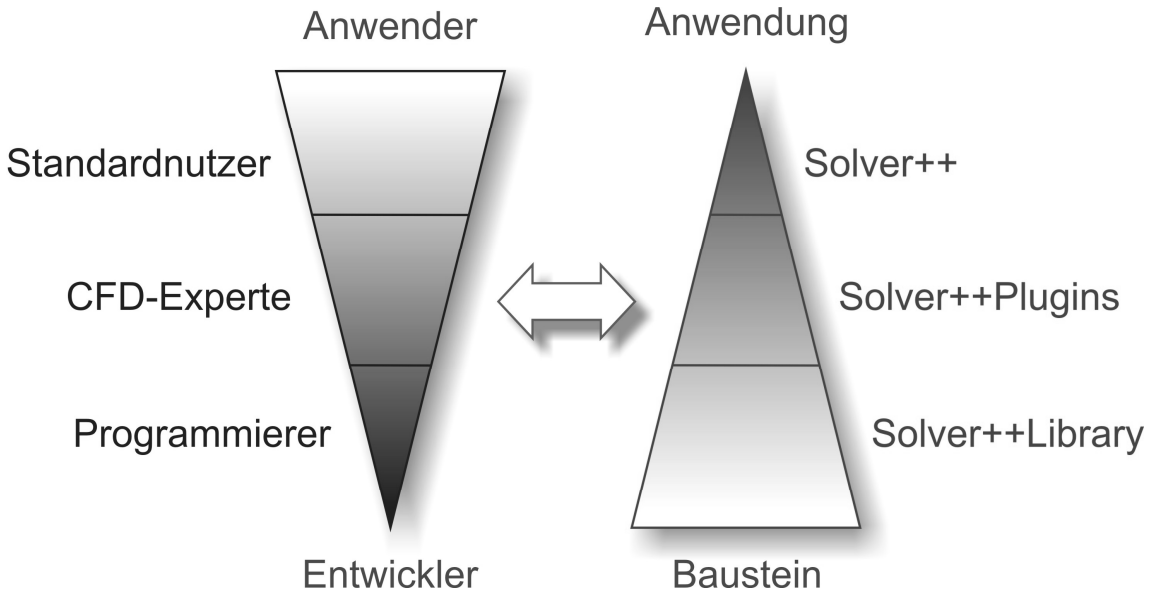


Bild 2-4: Das Stufenkonzept für das CFD-Tool; links: Klassifizierung der Benutzertypen von der breiten Masse der CFD-Anwender zur kleinen Gruppe der echten CFD-Programmierer; rechts: vom Solver++-Frontend für die Anwendung fertiger Modelle zur Solver++Library für die Bereitstellung der Modellgrundlagen

Drei klassische Interaktionsfälle sind für das Stufenkonzept vorgesehen. Im ersten Fall konfiguriert der Simulationsingenieur als Standardnutzer ganz einfach die Software, indem er das mathematische Modell für die Simulation auswählt. Dies geschieht über die Konfigurations- oder Parameterdateien. Der Fall bildet also die Interaktion mit den Anwenderschnittstellen der Simulationssoftware auf der oberen Ebene ab. Hierfür sind keine Programmierkenntnisse erforderlich. Die Interaktion erfolgt über textbasierte Dateien, beispielsweise im XML-Format. Der Standardnutzer sieht dabei lediglich die anwählbaren Module als funktionale Einheiten zur Gleichungsdiskretisierung oder Geometriemodifikation. Dieser Fall tritt bei jeder neuen Strömungssimulation auf.

Der zweite Fall stellt die Programmierung neuer Module oder Transportgleichungen durch den Wissenschaftler oder CFD-Experten zur Erweiterung der vorhandenen Modellierungsmöglichkeiten dar. Dies geschieht durch Interaktion mit den vorhandenen Programmierschnittstellen der Simulationssoftware auf der mittleren Ebene. Es sind hierfür grundlegende bis mittlere Programmierkenntnisse erforderlich. Durch die vorgegebenen strengen Schnittstellendefinitionen ist aber kein umfangreiches Softwaredesign nötig. Der CFD-Experte sieht beim Programmieren des neuen Moduls die funktionalen Komponenten der untersten Schicht aber nicht ihre Implementierungsdetails.

Das dritte und letzte Fallbeispiel bildet die Softwareentwicklung der funktionalen Komponenten des zugrunde liegenden Simulationsframeworks ab. Diese umfasst die Erweiterung und Optimierung des funktionalen Umfangs und die Definition neuer Anwenderschnittstellen. Der Programmierer benötigt deshalb umfangreiche Kenntnisse in der Softwareentwicklung bzw. dem Softwaredesign. Das Fallbeispiel stellt die Interaktion auf der untersten Ebene dar.

3 Grobstruktur des Softwaredesigns

Es gibt eine Vielzahl unterschiedlicher Faktoren, die auf den Entwurf der inneren Datenstruktur eines CFD-Codes mehr oder weniger direkt einwirken. Das vorliegende Kapitel beleuchtet die wichtigsten Faktoren näher und leitet daraus unter Berücksichtigung der Erkenntnisse aus Kapitel 2 die Grobstruktur für das Softwaredesign des CFD-Tools ab. Der erste Teilabschnitt ist dem Aufbau eines geeigneten Datenformates zur Implementierung der FVM gewidmet. Der folgende Teilabschnitt des Kapitels beschäftigt sich mit der Aufteilung der Software in funktionale Hauptgruppen. Die entwickelten Lösungsansätze für die native Verwendung der Tensorarithmetik und für die effiziente Parallelisierung der Software schließen das Kapitel ab.

3.1 Polyedrisches Datenformat

Das CFD-Tool benutzt die FVM zur Diskretisierung der PDG des aufgebauten mathematischen Modells. Voraussetzung dafür ist, dass für das betrachtete Lösungsgebiet ein numerisches Netz bzw. Gitter existiert, siehe Kapitel 2.1. Das Netz ist aus zweierlei Hinsicht erforderlich. Zum Einen definiert es die diskreten Orte, an denen die jeweiligen Variablen berechnet werden, zum Anderen ermöglicht es die Konstruktion der KV, die zur Integration der Differentialgleichungen herangezogen werden.

Die Art der verwendeten Rechennetze hat direkten Einfluss auf das interne Datenformat eines CFD-Codes, denn dieser muss während der Programmlaufzeit das Netz mit allen darauf definierten Variablen im Speicher abbilden. Der Entwurf des Datenformates reduziert sich also zunächst auf die Frage, mit welchen Netzen der CFD-Code arbeiten soll.

3.1.1 Netzarten

Grundsätzlich gibt es zwei Arten von numerischen Netzen, solche mit strukturiertem und solche mit unstrukturiertem Aufbau. Im 3-dimensionalen Fall, besitzen die feldinneren Knoten eines strukturierten Netzes genau 6 Nachbarknoten. Die Struktur des gesamten Netzes kann mithilfe des Indexraumes I und der Dimension i_{max}, j_{max} und k_{max} in den einzelnen Indexrichtungen i, j und k vollständig beschrieben werden. Jeder feldinnere Knoten besitzt also neben der eigentlichen Koordinate (x,y,z) im physikalischen Raum P eindeutige Koordinaten (i,j,k) im zugehörigen Indexraum I. Seine direkten Nachbarn werden durch Variation der Indices um ± 1 erreicht, vgl. FERZIGER und PERIC [23], also beispielsweise $(i-1,j,k)$ oder $(i+1,j,k)$. Erfolgt das Abschreiten der Knoten im strukturierten Netz immer auf dieselbe Art und Weise ist es nicht nötig, den Indexraum explizit zu speichern. Bei einem unstrukturierten Netz lassen sich die Nachbarschaftsverhältnisse zwischen den einzelnen Knoten nicht mehr so einfach ableiten. Es existiert kein durchgängiger Indexraum mehr und die Abhängigkeiten der Knoten im Netz müssen zusätzlich zu den reinen (x,y,z)-Koordinaten der Knoten bereitgestellt werden.

ZWART [108] schreibt, dass strukturierte Netze wegen ihres einfach zu beschreibenden inneren Aufbaus eher gebräuchlich waren als unstrukturierte Netze. Bei komplexen Geometrien zeigt sich allerdings schnell ein prinzipieller Nachteil der strukturierten Vernetzung. Die Notwendigkeit, einen durchgängigen Indexraum zu schaffen, schränkt deren Flexibilität erheblich ein. Die unstrukturierte Vernetzung unterliegt dieser Beschränkung nicht, wodurch unstrukturierte Netze in der Lage sind, auch hochkomplexe Geometrien ohne Probleme abzubilden.

Das Entwurfskonzept des CFD-Tools zielt auf ein möglichst breites Einsatzspektrum ab. Die Verwendung einer strukturierten Datenbasis würde dessen Einsatzmöglichkeiten von vornherein einschränken. Zum heutigen Zeitpunkt ist der Einsatz unstrukturierter Netze im CFD-Bereich weit verbreitet. So besitzen die meisten modernen CFD-Codes, wie beispielsweise CFX, FLUENT, STAR-CCM+ oder open∇FOAM [64], eine unstrukturierte Datenbasis. Diesem Trend wird auch im CFD-Tool entsprochen, zumal die Fähigkeit zur unkomplizierten Verarbeitung anspruchsvoller Geometrien neben Genauigkeit, Stabilität und Geschwindigkeit ein wichtiges Qualitätskriterium für jeden CFD-Code darstellt. Der durch das unstrukturierte Format verursachte zusätzliche Speicherbedarf kann in diesem Kontext übrigens außer Acht gelassen werden. Die auf dem Rechennetz definierten Lösungsvariablen verbrauchen im Normalfall wesentlich mehr Speicher.

Tabelle 3-1: Einsatzspektrum, Flexibilität und Genauigkeit eines CFD-Programms in Abhängigkeit von seiner internen Datenbasis; Die polyedrische Datenbasis geht als die flexibelste Lösung mit ausreichender Genauigkeit hervor

Datenbasis im CFD-Code		kartesisch	kont. strukt.	elem. unstrukt.	polyedrisch
Netztypen	kartesisch	●	●	●	●
	kont. strukt.		●	●	●
	elem. unstrukt.			●	●
	polyedrisch				●
Flexibilität[1)]		-	o	+	++
Genauigkeit		++	+	+	+

Legende:

●	*nutzbar*
-	*niedrig*
o	*neutral*
+	*hoch*
++	*sehr hoch*
kont. strukt.	*konturangepasst strukturiert*
elem. unstrukt.	*elementbezogen unstrukturiert*

[1)] Die Flexibilität kann mit einer Mehrblockstrategie generell erhöht werden

Tabelle 3-1 zeigt, welche Netztypen von den unterschiedlichen internen Datenformaten eines CFD-Codes verarbeitet werden können. Die Tabelle verfeinert die bisherigen Betrachtungen und enthält die strukturierten Formate kartesisch bzw. konturangepasst, sowie die unstrukturierten Formate element-bezogen bzw. polyedrisch. Bei den kartesisch strukturierten Netzen sind die Netzlinien immer entlang der Koordinatenachsen ausgerichtet. Die Verarbeitung komplexer Geometrien ist deshalb nicht möglich. Bei konturangepasst strukturierten Netzen folgen die Indexlinien des zugehörigen Indexraumes der Kontur der Geometrie. Die Flexibilität steigt gegenüber dem kartesischen Format erheblich. Elementbezogen unstrukturierte Netze besitzen gegenüber polyedrisch bzw. allgemein unstrukturierten Netzen eine eingeschränkte Anzahl vom Elementtypen. Meist sind das Tetraeder, Hexaeder, Prismen und Pyramiden. Die Konvention reduziert die Komplexität und den Speicherbedarf der Netze, da die Informationen über die Zellflächen bei vorgegebener Knotennummerierung eines jeden Elementtyps wegfallen. Das elementbezogene unstrukturierte Format offeriert zusammen mit dem allgemein polyedrischen Format die höchste Flexibilität bei der Vernetzung.

Ein genauere Betrachtung der in Tabelle 3-1 aufgeführten Netztypen lässt erkennen, dass ausgehend vom allgemeinen hin zum spezialisierten Format eine Abwärtskompatibilität vorliegt. So kann jedes strukturierte Netz beispielsweise als Spezialfall eines unstrukturierten Netzes aufgefasst werden, das sich auf den Elementtyp Hexaeder beschränkt. Übrigens werden unstrukturierte Netze, mit entsprechend ausgerichteten hexaedrischen Elementen bisweilen irreführender Weise als kartesische Netze bezeichnet. Solche Netze weisen gemeinhin aber keinen durchgängigen Indexraum auf, da sie die kartesische Struktur an den Rändern oft aufgeben. In Tabelle 3-1 fallen sie unter die Kategorie polyedrisch unstrukturiert.

Die Genauigkeit eines CFD-Codes wird stark vom jeweiligen Diskretisierungsschema der konvektiven Flüsse beeinflusst. Die Diskretisierung bis zu einer Genauigkeit zweiter Ordnung bereitet unabhängig von der Art der Datenhaltung keine Probleme. Der Einsatz höherwertiger Verfahren ist allerdings mit einer unstrukturierten Datenbasis nicht mehr so einfach möglich, denn dann ist für die Diskretisierung meist mehr als eine Nachbarzellschicht erforderlich. Die Beschränkung auf eine Genauigkeit zweiter Ordnung wird in Anbetracht der hohen Flexibilität einer unstrukturierten Datenhaltung für die meisten Fälle aber als völlig akzeptabel erachtet. Insgesamt bietet die polyedrische Datenbasis bei ausreichender Genauigkeit somit die höchste Flexibilität und das breiteste Spektrum nutzbarer Netzformate. Sie ist deshalb als Datenbasis im Softwareentwurf des CFD-Tools vorgesehen.

Es folgt ein erster Entwurf der Datenbasis.

Numerische Netze besitzen eine klare hierarchische Struktur, die den Aufbau der einzelnen Zellen regelt. Bild 3-1 zeigt einen Dodekaeder, der als Beispiel für eine allgemein unstrukturierte bzw. polyedrische Zelle dient. Der Dodekaeder besteht aus 12 Teilflächen. Jede der Teilflächen ist wiederum durch einen geschlossenen Polygonzug begrenzt. Jeder Polygonzug besitzt 5 Eckpunkte. Die Eckpunkte oder Knoten schließlich werden durch ihre Koordinaten definiert. Eine Zelle ist somit durch die Liste der Teilflächen, jede Teilfläche wiederum durch das begrenzende Polygon, und jedes Polygon durch die Liste der Eckpunkte definiert. Auf das gesamte Netz bezogen lautet die Reihenfolge der Hierarchieebenen von oben nach unten damit: Zellen, Flächen und Knoten. Es ist nicht nötig die Polygone als separate Hierarchieebene einzuführen. Ihr Aufbau kann bei der Flächen-

definition durch die Reihenfolge der einzelnen Knoten implizit berücksichtigt werden. Im Übrigen macht es für die spätere Diskretisierung Sinn, am Umlaufsinn der Knoten auch die Richtung der Flächennormalen festzumachen. Der Umlaufsinn der Knoten und die Flächennormale bilden deshalb in der Regel ein Rechtssystem.

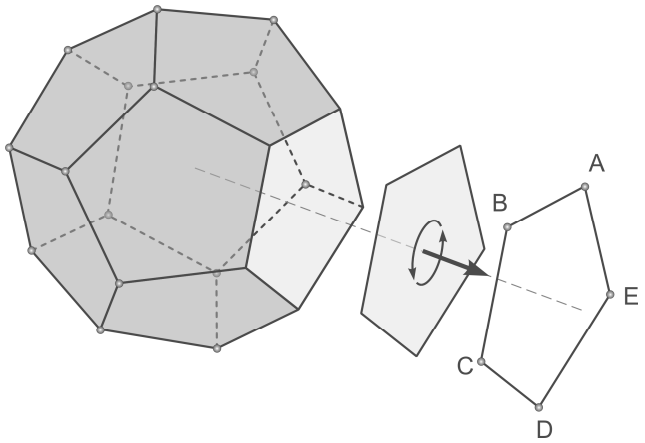


Bild 3-1: Dodekaeder als Beispiel für eine polyedrische Zelle; Diese besitzt 12 Teilflächen; Jede der Teilflächen wird durch ein Polygon mit 5 Eckpunkten begrenzt; Der Umlaufsinn der Eckpunkte oder Knoten um eine Teilfläche bildet mit der zugehörigen Flächennormale ein Rechtssystem

Die beschriebene Struktur objektorientiert nachzubilden, ist nicht gerade schwer. Die Herausforderung besteht allerdings darin, eine möglichst effiziente Implementierung zu schaffen. Zunächst liegt es nahe, die einzelnen Komponenten des Netzes als eigene Zell-, Flächen- und Knoten-Objekte vorzusehen und die gesamte Netzstruktur aus solchen Objektlisten aufzubauen. Nachteilig an diesem klassischen Ansatz ist, dass bei der späteren Diskretisierung viele indirekte Indexzugriffe entstehen und dadurch die Programmablaufgeschwindigkeit stark leidet. Im CFD-Tool werden die Komponenten deshalb nicht einzeln gespeichert. Stattdessen sieht der Entwurf vor, die Hierarchieebenen im Ganzen als Objekte abzubilden. Dadurch ist es möglich den gesamten Aufbau des Netzes in einige wenige eindimensionale Listen zusammenzufassen. Bild 3-2 zeigt die UML-Diagramme für die Klassen *PolyCellLayout* und *PolyFaceLayout*. Die Klassen sind identischen in Aufbau und Funktionalität. Die Doppelung erfolgt lediglich dafür, dass die Funktionsaufrufe der beiden Klassen bezogen auf die spezielle Bedeutung der Hierarchieebenen eindeutig unterscheidbar bleiben. Beide Objekte besitzen jeweils eine Liste von Indizes der nächst niedrigeren Hierarchieebene, sowie eine Liste von Hilfsindizes. Sollen beispielsweise alle Flächenindizes *iF* einer bestimmten Zelle *iC* aus dem *PolyCellLayout* ausgelesen werden, muss zunächst über die Zugriffsfunktionen *get_front_CF(iC)* und *get_back_CF(iC)*, aus der Liste von Hilfsindizes ein Anfangs- bzw. Endwert bestimmt werden. Mithilfe dieser beiden Werte können dann die zur Zelle gehörenden Flächenindizes über die Zugriffsfunktion *get_F(iCF)* aufgelesen werden. Das gleiche Prinzip gilt bei den zu einer Fläche gehörenden Knotenindices *iN* für die Klasse *PolyFaceLayout*. Der Aufbau der beiden Datenstrukturen kann zudem leicht anhand des in Bild 3-2 abgebildeten vollständig indizierten 2d-Netz nachvollzogen werden. Der Aufbau der beiden Datenstrukturen ist übrigens im 2d-Fall und im 3d-Fall völlig identisch. Das Vektorfeld mit den reinen Knotenkoordinaten vervollständigt schließlich die einzelnen

Komponenten der polyedrischen Datenbasis zur Speicherung des Netzes. Die Spezifikation der Vektorfelder findet sich weiter unten im Abschnitt über die verfügbaren Datentypen, der konkrete Entwurf der Vektorfelder folgt in Kapitel 3.3, das die Tensorarithmetik im CFD-Tool erläutert.

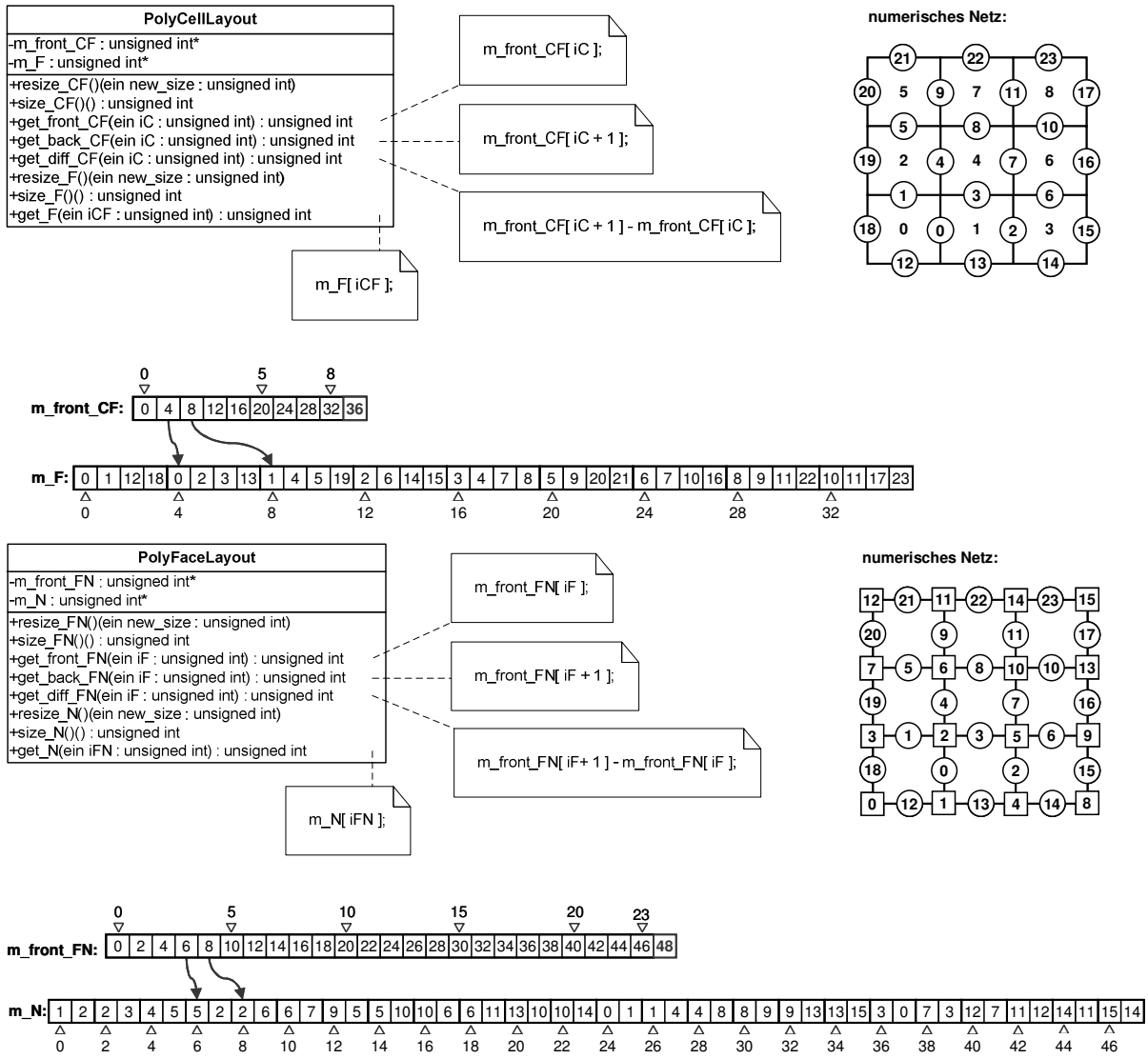


Bild 3-2: Datenstrukturen für die effiziente Speicherung polyedrischer Netze am Beispiel eines unstrukturiert indizierten 2d-Netzes mit 3x3 Zellen; Die Datenstrukturen sehen im 3d-Fall genauso aus; C = cell; CF = cell-face; F = face; FN = face-node; N = node

3.1.2 Mehrblockstrategie

Die Verwendung der Mehrblockstrategie steigert die Flexibilität der einzelnen Netzformate unabhängig voneinander, siehe Tabelle 3-1. Die Methodik führt einen zusätzlichen Abstraktionslevel ein, indem sie das zu vernetzende Lösungsgebiet zunächst in grobe Blöcke aufteilt. Die einzelnen Blöcke besitzen abhängig vom Format der Datenbasis eine hexaedrische oder allgemeine Form. Die Mehrblockstrategie erhöht die Flexibilität insbesondere bei CFD-Codes mit strukturierter Datenbasis beträchtlich. Sie

gestattet nämlich, den durchgängigen Indexraum an den Interfaces zwischen den Blöcken zu unterbrechen, siehe LILEĆ ET AL. [55]. Verschiedene Beispiele für die erfolgreiche Anwendung der Mehrblockstrategie bei der strukturierten Vernetzung komplexer Geometrien aus dem technischen Bereich finden sich beispielsweise in den Arbeiten von KRONSCHNABL [48], EINZINGER [22], STEINBRECHER [89], SCHUSTER [85] und BÖHM [10]. Über die klassische Anwendung hinaus, ist die Mehrblockstrategie auch noch in verschiedenen anderen Bereichen von Nutzen. Als Beispiele dienen hier die Berücksichtigung wechselnder Bezugssysteme und die geometriebasierte Parallelisierung. Angesichts der Vielfalt ihrer Anwendungsmöglichkeiten ist klar, dass auch die Mehrblockstrategie in den Entwurf des CFD-Tools aufgenommen wird.

Der Einsatz blockstrukturierter Rechennetze bei der Validierung neuer mathematischer Modelle ist äußerst sinnvoll. Generell führen strukturierte Netze, deren Zellausrichtung auf das Strömungsfeld abgestimmt ist, zu einer besseren Diskretisierung. Wie bereits gezeigt, steht die Verwendung strukturierter Netze aber nicht im Widerspruch zu einem unstrukturierten CFD-Code. Das Simulationsergebnis ist bei gleicher Diskretisierung und gleichem Netz unabhängig von der verwendeten Datenbasis. So zeigen die Anwendungsbeispiele in Kapitel 5 ausschließlich blockstrukturiert vernetzte Geometrien. Die Erzeugung von qualitativ hochwertigen strukturierten Rechennetzen für komplexe Geometrien verursacht allerdings einen hohen Zeitaufwand. Die Erfahrung zeigt, dass dieser Aufwand im industriellen Umfeld kaum gerechtfertigt ist. Im Gegensatz zur Wissenschaft ist dort meist ausreichend, ein qualitativ richtiges Ergebnis zu erzielen. Die Verwendung unstrukturierter Netze stellt in diesem Fall die flexiblere und schnellere Lösung dar. Das gewählte Design des CFD-Tools kann auf natürliche Weise beide Anwendungsfälle bedienen. Die blockweise gegliederte, unstrukturiert polyedrische Datenhaltung, im Weiteren als blockpolyedrisch bezeichnet, macht dies möglich.

3.1.3 Definition der Kontrollvolumen (KV)

Das numerische Netz bildet mit den Knoten und Zellen die Grundlage bei der Aufteilung des Lösungsgebiets in finite KV. Die KV müssen dabei in das Rechengebiet so eingepasst sein, dass sie sich nicht überlappen und das Rechengebiet vollständig auffüllen. Für die Konstruktion der KV gibt es prinzipiell zwei unterschiedliche Herangehensweisen, die zellzentrierte Methode und die knotenbasierte Methode, vgl. HIRSCH [37] oder FERZIGER und PERIC [23]. Bei der zellzentrierten Methode stimmen die Zellen mit den KV überein, und die Knoten liegen an den Ecken der KV. Beispiele für zellzentrierte Verfahren finden sich unter anderem bei ZWART [108], EINZINGER [22] und SKODA [86]. Bei der knotenbasierten Methode werden die KV dagegen um die Knotenpunkte des Netzes herumkonstruiert. Die KV setzen sich anteilsmäßig aus mehreren angrenzenden Zellen zusammen. Die Knoten bilden jetzt nicht mehr die Ecken sondern die Mittelpunkte der KV. Da auf diese Art und Weise sozusagen ein zweites numerisches Netz entsteht, wird die knotenbasierte Methode auch als duale Gitter Methode bezeichnet, siehe HUURDEMAN [39] oder REXROTH [71]. In der Literatur finden sich viele weitere Bezeichnungen für die knotenbasierte Methode. RAITHBY et al. [67] sowie RAW et al. [69] verwenden sie als elementbasierte FVM. Bei BALINGA und PATANKAR [7], wie auch bei SCHNEIDER und RAW [83] kommt sie als KV-basierte FEM zum Einsatz, bei BARTH [9] als Zell Knoten Methode. Das Grundprinzip bleibt aber immer dasselbe.

Die objektive Bewertung der beiden vorgestellten Methoden ist nicht leicht. ZWART [108] diskutiert ihre Vor- und Nachteile unter anderem anhand des Genauigkeit/Kosten-Verhältnisses, der speziellen Modellierung sowie persönlicher Präferenzen. Beleuchtet man Diskretisierung, Speicheraufwand und Geschwindigkeit stellt sich das Problem letztlich wirklich als eine Frage des persönlichen Geschmacks dar. Keines der beiden Verfahren bietet grundsätzlich eine höhere Ablaufgeschwindigkeit oder einen geringeren Speicherbedarf. Diese Eigenschaften hängen stark von der Güte der konkreten Implementierung ab. Liegt eine vergleichbare Diskretisierung vor, so gilt außerdem, dass deren Genauigkeit und Stabilität direkt von der Qualität des zugrundeliegenden numerischen Netzes abhängig. Dies ist für beide Verfahren in gleichem Maße zutreffend. Das Rechennetz sollte das Lösungsgebiet immer so abbilden, dass Genauigkeit und Stabilität nicht unnötig beeinträchtigt werden. Von einer hohen Netzqualität spricht man im Allgemeinen dann, wenn das Rechennetz diese Anforderungen erfüllt. Um die Netzqualität zu quantifizieren, können unterschiedliche Kenngrößen herangezogen werden. Die Verzerrung, die Streckung und das Expansionsverhältnis der Zellen sind gängige Größen für deren Beschreibung. Bei der zellzentrierten Methode stimmen die Zellen des Netzes direkt mit den finiten KV überein. Die Netzqualität hat deshalb eine unmittelbare Aussagekraft über die Auswirkung des Netzes auf das numerische Verfahren. Bei der knotenbasierten Methode, bei der sich die KV und die Zellen unterscheiden, stellt die Netzqualität nur ein mittelbares Kriterium dar. Dieser Sachverhalt ist die Grundlage dafür, dass im CFD-Tool die zellzentrierte Methode implementiert wird.

Das Dualitätsprinzip der knotenzentrierten Methode kann im Übrigen auch bei der Netzgenerierung genutzt werden. So kann ein mittels Delaunay-Triangulierung erzeugtes Netz leicht in ein polyedrisches Voronoi-Netz überführt werden.

3.1.4 Diskrete Speicherorte im Netz

Das numerische Netz definiert neben den KV auch die diskreten Speicherorte der Feldvariablen. Das CFD-Tool verwendet für den Aufbau der KV die zellzentrierte Methode. Dementsprechend wird der Großteil der Lösungsvariablen in den Mittelpunkten der Zellen, die bei der zellzentrierten Methode die KV repräsentieren, gespeichert. Die Volumenintegrale in den Transportgleichungen können so unmittelbar ausgewertet werden. Zusätzlich ist es hilfreich bestimmte Feldvariablen in den Mittelpunkten der Zellflächen zu speichern. Beispiele hierfür sind der Massenstrom $\dot{m}$ oder der Zellflächeninterpolationsfaktor λ. Beide werden bei der Diskretisierung der konvektiven Flüsse in den Transportgleichungen benötigt. Neben den KV und den Flächen stellen die Knoten des Netzes die dritte Möglichkeit zur Speicherung von Feldvariablen dar. Bei der zellzentrierten Methode wird davon allerdings meist kein Gebrauch gemacht. An den Knoten definierte Feldvariablen werden höchstens temporär zum Exportieren des Simulationsergebnisses in einem knotenbasierten Format benötigt. Um zu verdeutlichen, ob eine bestimmte Feldvariable den Zellen, Flächen oder Knoten des Netzes zugeordnet ist, unterscheidet das CFD-Tool logisch zwischen Zell-, Flächen- und Knotenfeldern. Bild 3-3 veranschaulicht die diskutierten Speichermöglichkeiten der Feldvariablen anhand eines einfachen polyedrischen Netzes. Von oben nach unten sind die Speicherorte für Knoten-, Flächen- und Zellfelder dargestellt.

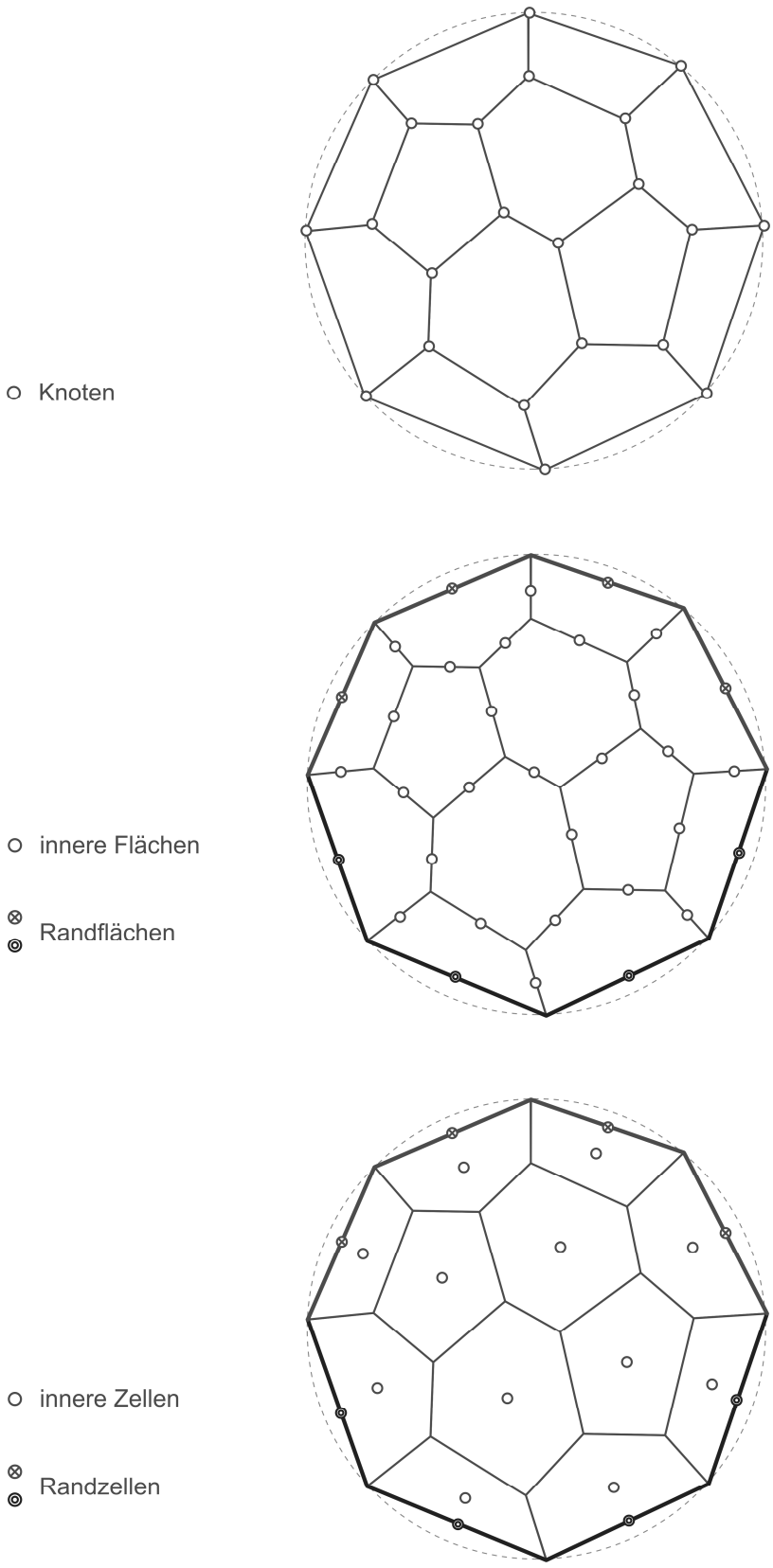


Bild 3-3: Knoten, Flächen und Zellen als diskrete Speichermöglichkeiten im unstrukturierten Polyedernetz; Bei Flächen- und Zellfeldern wird explizit zwischen dem inneren Bereich und dem oder den Randbereich(en) unterschieden

Die Speicherorte können zusätzlich danach gestaffelt werden, ob sie am Rand oder im Innern des Rechengebietes liegen. Dies ist insbesondere zur Angabe unterschiedlicher Randbedingungen für die spätere Diskretisierung wichtig. So unterscheidet das CFD-Tool bei den Zell- und Flächenfeldern zwischen Randbereichen, Interfaces und dem Gebietsinneren. Die Datenpositionen werden dabei immer so umsortiert, dass zuerst der netzinnere Bereich, danach die Interfaces und zuletzt die Randbereiche aneinandergereiht durchnummeriert sind. Dadurch ist der einfache bereichsweise Zugriff über den Start und Schlussindex des jeweiligen Bereichs bei der Diskretisierung möglich, siehe Bild 3-4.

Da das Netz selbst keine Randzellen besitzt, werden diese bei der zellzentrierten Methode als unendlich dünne Schicht erzeugt. Diese Zellschicht fällt mit den Randflächen des originären numerischen Netzes zusammen. Der Kunstgriff ist notwendig, um auch für die den Zellen zugeordneten Feldvariablen Randbedingungen spezifizieren zu können. Im Falle der Interfaces ist das nicht nötig. Allerdings muss jedem Interface die angrenzende Zellschicht des Nachbarnetzes mitgeteilt werden. Das polyedrische Netz in Bild 3-3 weist zwei unterschiedliche Randbereiche auf. Ein Vergleich der diskreten Speicherpositionen der Zell- und der Flächenfelder zeigt, dass diese an den Randbereichen jeweils übereinstimmen.

Das polyedrische Datenformat ermöglicht generell, die freie Umsortierung der Zell- und Flächenpositionen. Bei einem impliziten Lösungsverfahren beeinflusst die Konvektivität und Indizierung der feldinneren Zellen die Besetzungsstruktur der Koeffizientenmatrix. Durch die geschickte Sortierung der Zellen wird die Bandbreite der Matrix möglichst gering gehalten. Dies wirkt sich positiv auf die Zugriffszeiten und die Stabilität vieler iterativer Lösungsverfahren aus. Eine bewährte Strategie zur geschickten Indizierung der Zellen ist beispielsweise der Algorithmus nach CUTHILL und McKEE [17]. Die Konvention über die Speicherpositionen der Zellfelder im CFD-Tool verlangt lediglich, dass die feldinneren Zellen insgesamt nach vorne sortiert sind. Die relative Position der feldinneren Zellen zueinander ist davon nicht berührt und kann zur Reduktion der Bandbreite der Koeffizientenmatrix frei optimiert werden.

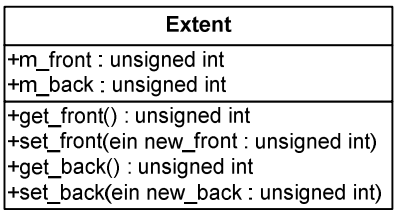


Bild 3-4: Miniklasse zur Speicherung des Anfangs- und Schlussindexes der unterschiedlichen Bereiche der Flächen- und Zellfelder im numerischen Netz

3.1.5 Hilfsstrukturen für die Diskretisierung

Bei der Diskretisierung der Differentialgleichungen werden neben den Datenstrukturen, die den Aufbau des numerischen Netzes beschreiben, zusätzliche Hilfsstrukturen benötigt, die den Indexzugriff auf die verschiedenen Feldvariablen erleichtern. Die KV sind bei der zellzentrierten Methode identisch mit den Zellen des numerischen Netzes. Die Auswertung eines Oberflächenintegrals erfordert demnach, die Werte aller betroffenen

zellbezogen gespeicherten Variablen von den Zellmittelpunkten an die dazwischen liegenden Zellflächen zu interpolieren. Iteriert ein Algorithmus hierfür einfach über alle Zellen des Netzes, wird der größte Teil dieser Interpolationen doppelt durchgeführt. Das liegt daran, dass jede innere Fläche Bestandteil von exakt zwei Zellen ist, vgl. Bild 3-5. Erfolgt die Auswertung der Oberflächenintegrale hingegen so, dass die Iterationsschleife über alle Flächen des Netzes läuft, erhält man einen wesentlich effizienteren Algorithmus. Die notwendigen Interpolationen werden dann für jede Fläche nur einmal durchgeführt. Das Ergebnis einer Interpolation kann dann jeweils für beide angrenzenden Zellen verwendet werden.

Wertet ein Oberflächenintegral Flüsse durch die Teilflächen eines KV aus, muss unterschieden werden, ob die Flüsse in das KV hinein- oder aus dem KV herausgehen. Hierfür wird die Richtung der Oberflächennormalen herangezogen. Ist ein Fluss positiv, so läuft er in Richtung der Oberflächennormalen, ist er stattdessen negativ, so läuft er genau entgegengesetzt. Bezeichnet der niedrigere Index zweier benachbarter Zellen die Master-Zelle und der höhere die Slave-Zelle und wird zudem die Oberflächennormale der dazwischen liegenden Fläche immer von der Master- zur Slave-Zelle hin ausgerichtet, so ist eindeutig definiert, welches der beiden angrenzenden KV der Fluss befüllt und welches er entleert. Diese Konvention ermöglicht für die Auswertung der Flüsse, über die Flächen des Netzes zu iterieren.

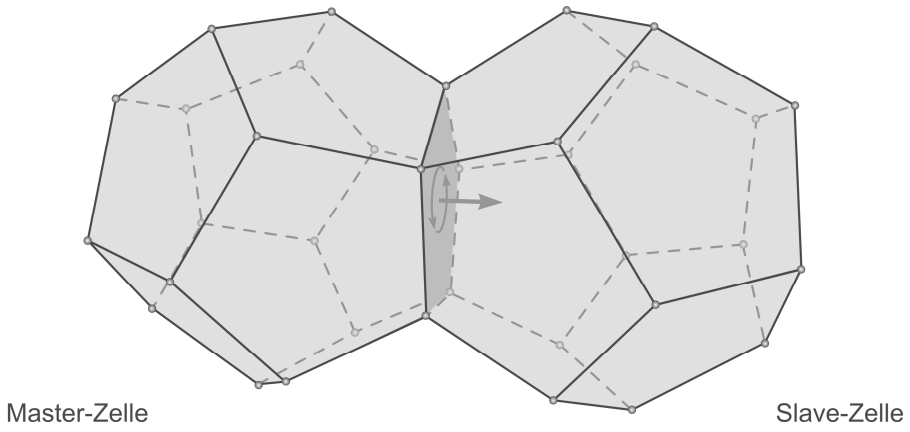


Bild 3-5: Die inneren Teilflächen besitzen eine Master- und eine Slave-Zelle; Ihr Umlaufsinn wird so gewählt, dass die jeweilige Flächennormale von der Master- zur Slave-Zelle zeigt

Die Entwürfe der Klassen *PolyCellLayout* und *PolyFaceLayout* zur Beschreibung des numerischen Netzes liefern keine Aussage darüber, welche Zellen zu einer bestimmten Fläche gehören. Deshalb müssen für die Diskretisierung der Oberflächenintegrale nach obigem Schema zwei Listen eingeführt werden, die die Verknüpfung zwischen einem Flächenindex *iF* und dem dazugehörigen Master-Zellindex bzw. Slave-Zellindex liefern. Bild 3-6 zeigt den Inhalt der beiden Listen *m_master_F* und *m_slave_F* für das angegebene unstrukturiert indizierte 2d-Netz. Das Vorgehen bleibt absolut identisch, wenn der 3d-Fall betrachtet wird. Die Randflächen besitzen eigentlich nur noch Master-Zellen. Durch die künstliche Schicht von Randzellen können aber dennoch Slave-Zellen für die Randflächen angegeben werden. Die Liste *m_slave_F* spiegelt dies ab dem Flächenindex *iF* = 12 wieder.

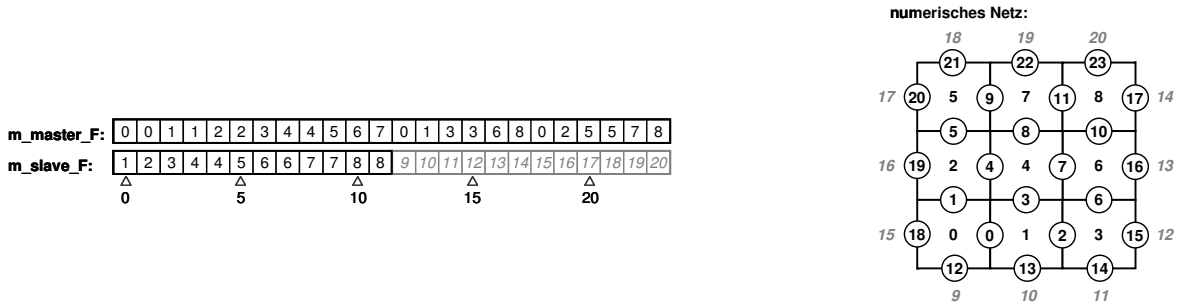


Bild 3-6: Verknüpfung zwischen den Teilflächen und ihren angrenzenden Master- und Slave-Zellen am Beispiel des unstrukturiert indizierten 2d-Netzes mit 3x3 Zellen; Die Listen *m_master_F* und *m_slave_F* ordnen jedem Flächenindex einen Master- und Slave-Zellindex zu und erleichtern so die Diskretisierung von Oberflächenintegralen; F = face

Für implizite Lösungsverfahren sind weitere Hilfsdaten notwendig, die die Besetzungsstruktur der Matrix des linearen Gleichungssystems wiedergeben. Das CFD-Tool verwendet ausschließlich Finite-Volumen-Diskretisierungen, die auf die erste Nachbarzellschicht einer Polzelle beschränkt bleiben. Die so entstehenden Rechenmolekühle gewährleisten eine geringe Bandbreite der Lösungsmatrix. Die sich ergebende Besetzungsstruktur der Matrix kann ähnlich wie das Layout des numerischen Netzes in eindimensionalen Listen gespeichert werden. Das CFD-Tool verwendet hierzu das komprimierte Zeilenspeicherformat CRS (Compressed-Rowwise-Storage), vgl. BARRET et al. [8], das lediglich die von Null verschiedenen Koeffizienteneinträge speichert. Die Klasse *PolyMatrixLayout* nimmt die Besetzungsstruktur der Matrix im beschriebenen Format auf. Ihr Entwurf ist in Bild 3-7 dargestellt. Um alle Spalteneinträge *iCol* für eine bestimmte Zeile *iRow* auszulesen, wird zunächst über die Funktionen *get_front_pos(iCol)* bzw. *get_back_pos(iCol)* die Start und Endposition im Hilfsarray der Spaltenindices bestimmt. Mit den zurückgegebenen Werten *iPos* können dann die Indizes aller Spalteneinträge mit der Funktion *get_column(iPos)* ausgelesen werden. Bild 3-7 zeigt das sich ergebende Datenbild im Falle des dargestellten unstrukturiert indizierten 2d-Netzes mit 3x3 Zellen. Die Datenstruktur sieht im 3d-Fall genau gleich aus.

Für die implizite Diskretisierung von Volumen- und Oberflächenintegralen werden zusätzliche Verknüpfungen abgeleitet, die den direkten Zugriff auf die Koeffizienten der Lösungsmatrix erlauben, siehe Bild 3-7. Bei der Auswertung der Volumenintegrale wird sowohl der Index der Zelle *iC* als auch der Index des Diagonalkoeffizienten *iPolePos* benötigt. Dafür wird die Liste *m_pole_C* eingeführt, die zu gegebenem Zellindex den entsprechenden Diagonalindex zurückliefert. Bei der Auswertung der Oberflächenintegrale müssen außerdem einem bestimmten Flächenindex *iF* die Indizes *iPolePos* und *iNborPos* des Diagonal- und des Nachbarkoeffizienten der Master-Zelle, bzw. des Diagonal- und Nachbarkoeffizienten der Slave-Zelle zugeordnet werden. Dafür sind die Listen *m_master_pole_F* und *m_master_nbor_F* bzw. *m_slave_pole_F* und *m_slave_nbor_F* vorgesehen, die zu gegebenem Flächenindex die entsprechenden Koeffizientenindizes zurückliefern. Zu beachten ist, dass die Listen *m_master_nbor_F*, *m_slave_pole_F* und *m_slave_nbor_F* nur über die inneren Flächen des Netzes gehen, denn zu den Randflächen existieren in der Matrix keine entsprechenden Einträge.

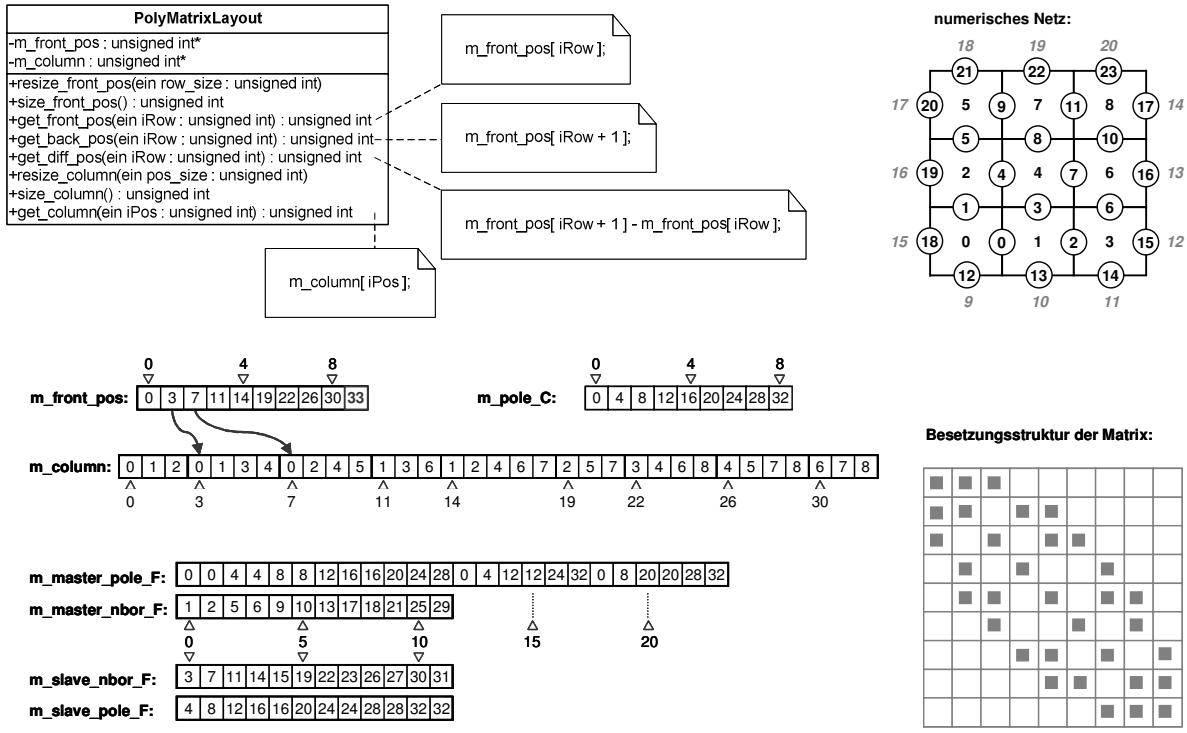


Bild 3-7: Entwurf zur Darstellung der Besetzungsstruktur der Matrix für das abgebildete unstrukturiert indizierte 2d-Netz; Das Schema bleibt im 3d-Fall gleich; Die Listen *m_pole_C bzw.* m*_master_pole_F*, *m_master_nbor_F*, *m_slave_nbor_F* und *m_slave_pole_F* ermöglichen den direkten Koeffizientenzugriff bei der Diskretisierung von Volumen- bzw. Oberflächenintegralen; C = cell; F = face; pole = Pol; nbor = Nachbar

3.1.6 Verfügbare Tensortypen

In partiellen Differentialgleichungen treten Variablen mit unterschiedlichen mathematischen Eigenschaften auf. Der Begriff Tensor ordnet die Variablen nach der Anzahl ihrer Komponenten ein. Jeder Tensor besitzt die Dimension d und den Rang r. Die Anzahl n_k der Komponenten eines Tensors ergibt sich zu:

$$n_k = d^r \,. \tag{3.1}$$

Der Softwareentwurf des CFD-Tools betrachtet ausschließlich den 3-dimensionalen Fall, sowie Tensoren vom Rang $r \leq 2$. Dies ist ausreichend für die in dieser Arbeit behandelten Differentialgleichungen. Die Datenbasis des CFD-Tools könnte aber ohne weitere Probleme auf Tensoren mit einem Rang $r > 2$ ergänzt werden.

Tensoren mit Rang 0 bezeichnet man auch als Skalare, Tensoren mit Rang 1 als Vektoren. Diese Benennungen sind leichter eingängig als die Unterscheidung der Tensoren nach ihrem Rang. Aus diesem Grund halten sie auch Einzug in die Datenbasis des CFD-Tools. Tensoren mit Rang 2 werden dann einfach als Tensoren bezeichnet. Beispiele für Skalare, Vektoren bzw. Tensoren aus der Fluidmechanik sind der statische Druck p, die Geschwindigkeit u_i bzw. der Geschwindigkeitsgradient $\partial u_i / \partial x_j$.

Tabelle 3-2 fasst die Klassifizierung der verfügbaren Tensortypen zusammen. Die angegebenen Datentypen können jeweils als Wertefeld oder Einzelwert deklariert werden. Der konkrete Klassenentwurf der Datentypen folgt in Kapitel 3.3, das die gesamte Tensorarithmetik für das CFD-Tool ausführlich behandelt. Skalare, Vektoren und Tensoren können als Zell-, Flächen- und Knotenfelder angelegt werden.

Tabelle 3-2: Zusammenfassung der verfügbaren Tensortypen im CFD-Tool; Dimension d=3; N entspricht der Feldgröße

Tensor Rang	Anzahl der Komponenten	Bezeichnung in der Mathematik	Variablenbezeichnung im CFD-Tool
0	1	Skalar	Sca
	N x 1	Skalar-Feld	ScaSet
1	3	3-Vektor	Vec
	N x 3	3-Vektor-Feld	VecSet
2	9	3x3-Matrix	Ten
	N x 9	3x3-Matrix-Feld	TenSet

3.2 Modularisierung der Hauptgruppen

Der vorliegende Abschnitt beschäftigt sich mit der Konkretisierung des in Kapitel 2.5 für das CFD-Tool entworfenen Schichtenmodells. Dafür wird der gesamte Entwurf zunächst in funktionale Hauptkomponenten aufgliedert und die notwendigen Verknüpfungen der einzelnen Komponenten zueinander geklärt.

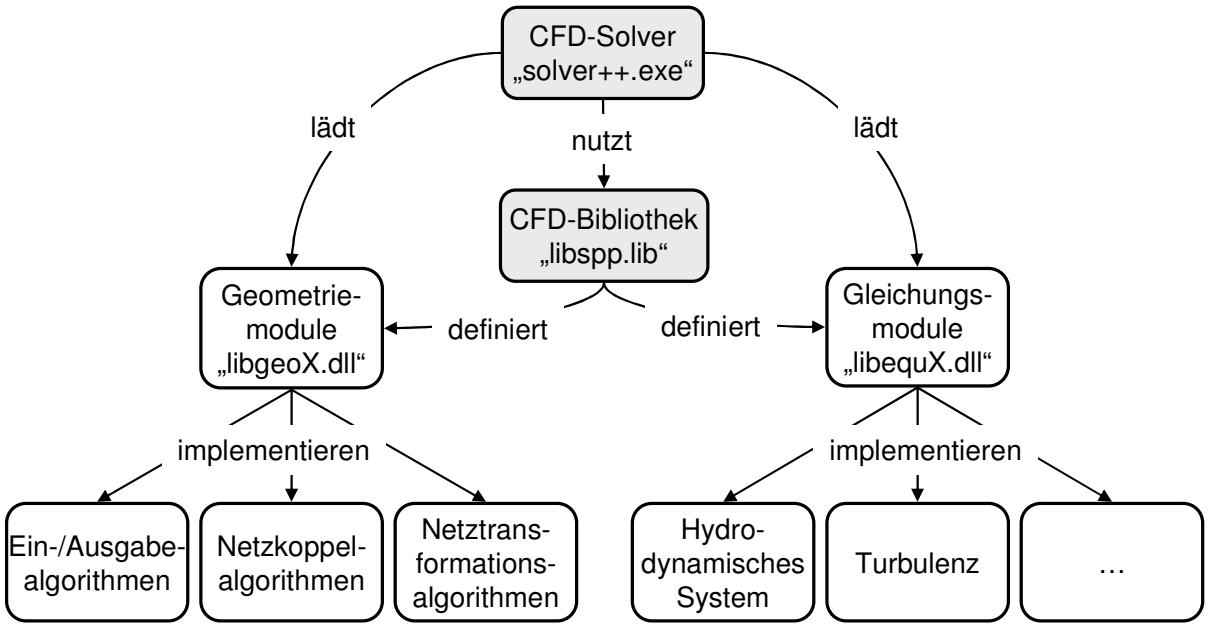


Bild 3-8: Konzept zur Aufgliederung der CFD-Software in Haupt- und Untergruppen und funktionaler Zusammenhang zwischen den einzelnen Modulen

Bild 3-8 veranschaulicht das Konzept der unterschiedlichen Haupt- und Untergruppen des CFD-Tools. Die Kernkomponente, die beinahe die gesamte Funktionalität der Software für die weiter außen liegenden Schichten beinhaltet ist die statische CFD-Bibliothek *libspp.lib*. Die beiden Module *libgeoX.dll* und *libequX.dll* stellen Platzhalter für alle dynamischen Laufzeitbibliotheken zur Geometriebearbeitung und Gleichungsmodellierung dar. Die einzelnen Module implementieren ihre Funktionalität durch Einbinden des zentralen Kernbausteins und der darin definierten Klassenschnittstellen. Die Spitze oder das Dach des gesamten Bauwerks bildet das ausführbare Programm *solver++.exe*, das die gesamte Funktionalität aller Module unter sich vereint. Es nutzt die Klasseninterfaces der Solverbibliothek *libspp.lib* und erschließt sich damit die Funktionalität der dynamischen Bibliotheken.

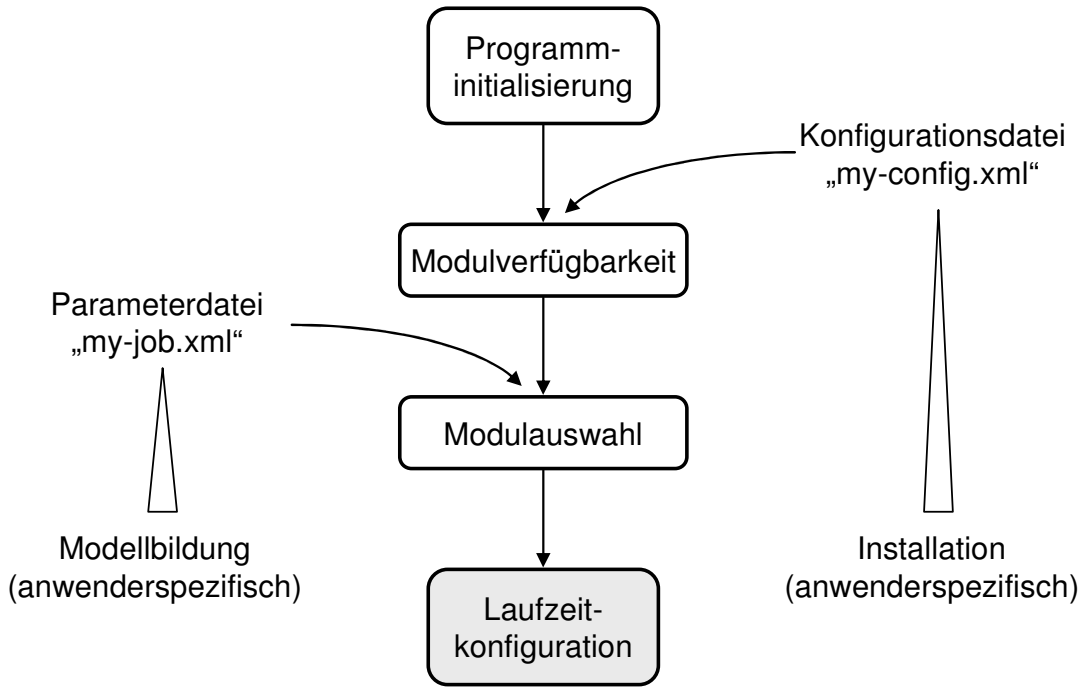


Bild 3-9: Informationsfluss beim Laden der Laufzeitkonfiguration als Teilmenge der verfügbaren Solvermodule

In Bild 3-9 ist der Informationsfluss dargestellt, der sich bei jeder CFD-Simulation mit dem CFD-Tool einstellt. Nach der Initialisierung des Programms wird zunächst eine zentrale Konfigurationsdatei ausgelesen, die alle verfügbaren Module spezifiziert. Die Konfigurationsdatei ist anwenderspezifisch, wodurch die Funktionalität des CFD-Tools ohne komplette Neuinstallation auch im Nachhinein noch leicht erweitert werden kann. Ist die Modulverfügbarkeit bekannt, können die für die konkrete CFD-Simulation benötigten Module dynamisch geladen werden. Die Auswahl der Module erfolgt dabei implizit durch die Modellbildung bzw. Parametrierung über das Jobfile. Insgesamt ergibt sich so die fertige Laufzeitkonfiguration für die jeweilige CFD-Simulation.

3.3 Tensorarithmetik

Das Lösen partieller Differentialgleichungen mithilfe der FVM erfordert die Integration der Gleichungen über jedes KV im Rechennetz. Während der anschließenden Diskretisierung der Integralterme treten relativ einfache arithmetische Ausdrücke auf. Sie beschreiben die lokale Änderung einer Lösungsvariable im KV in Abhängigkeit der Flüsse über die Ränder sowie der auftretenden Quellen und Senken. Trotz ihrer Einfachheit stellen diese Ausdrücke allerdings eine der häufigsten Fehlerquellen bei der programmiertechnischen Umsetzung der FVM dar. Dies liegt im Wesentlichen daran, dass Programmiersprachen wie beispielsweise C, C++ oder Fortran von sich aus keine Tensorrithmetik kennen. Die bei der Diskretisierung auftretenden Terme können deshalb nicht direkt in ihrer natürlichen und intuitiven Schreibweise in die Algorithmen übernommen werden. Vielmehr ist der Programmierer gezwungen, die Terme in ihre Komponentenschreibweise zu zerlegen, bevor er sie in den Quelltext einbinden kann. Zunächst kompakte Formeln werden so schnell unübersichtlich und lang, wodurch sich einschleichende Fehler, zum Beispiel durch Vertauschen einzelner Komponenten beim „Kopieren und Einfügen“, nur schwer erkennen lassen. Dieser Sachverhalt wird am besten an einem einfachen aber repräsentativen Beispiel erörtert. Als Modellgleichung dient die lineare Extrapolation der skalaren Variable ϕ. Nach Einstein'scher Summenkonvention lautet diese:

$$\phi = \phi_0 + \frac{\partial \phi_0}{\partial x_i} \cdot (r_i - r_{i,0}) \,. \tag{3.2}$$

In die Komponentenschreibweise gebracht, ergibt sich für selbige Gleichung dagegen:

$$\phi = \phi_0 + \frac{\partial \phi_0}{\partial x} \cdot (r_x - r_{x,0}) + \frac{\partial \phi_0}{\partial y} \cdot (r_y - r_{y,0}) + \frac{\partial \phi_0}{\partial z} \cdot (r_z - r_{z,0}) \,. \tag{3.3}$$

Beide Ausdrücke sind in mathematischem Sinne absolut äquivalent. Der Vergleich ihrer Schreibweisen macht aber dennoch deutlich, wie schnell grundsätzlich einfache Tensorarithmetik unter Verwendung der Komponentenschreibweise unübersichtlich und somit anfällig für Syntaxfehler wird.

3.3.1 Herkömmlich überladene Operatoren

Eine objektorientierte Programmiersprache wie C++ bietet auf den ersten Blick eine relativ einfache Lösung. Die Erstellung neuer Datentypen in Form von Objekten ist Bestandteil ihres Grundkonzepts. So ist es möglich, verschiedene Tensorklassen ohne größere Probleme zu implementieren. Die benötigten arithmetischen Verknüpfungsoperationen können den Klassen durch Überladen der Operatoren als Memberfunktionen hinzugefügt werden. Auf diese Art und Weise wird es möglich alle arithmetischen Ausdrücke in ihrer Kurzschreibweise zu implementieren. Der entstehende Quelltext wird leichter wartbar. Das vorgeschlagene Design hat aber einen entscheidenden Nachteil,

denn die Operanden binärer Ausdrücke werden in C++ immer paarweise evaluiert. Bestehen diese Ausdrücke nun aus kleinen Vektoren, beobachten VELDHUIZEN [99] sowie VELDHUIZEN und JERNIGAN [101] einen beträchtlichen Performance Verlust gegenüber einer expliziten Implementierung in Fortran oder C, die die Ausdrücke komponentenweise aber in einem Zug behandelt.

Bild 3-10 zeigt die einzelnen Teilschritte, die ein Programm für die Auswertung der Operatoren auf der rechten Seite von Modellgleichung (3.2) abarbeiten muss. Hierbei wird vereinfachend angenommen, dass der Gradient der Variable ϕ als eigene vektorielle Größe abgespeichert ist. Insgesamt entstehen während des Prozesses drei temporäre Objekte. Diese Objekte haben die Aufgabe, das Ergebnis der letzten Operation zwischenzuspeichern und für die jeweils nächste Operation zur Verfügung zu stellen. Gleichung (3.2) wird so Schritt für Schritt berechnet, bis schließlich das Ergebnis feststeht. Die temporären Objekte werden nach ihrem Gebrauch durch den Aufruf ihres Destruktors automatisch gelöscht. Die Zuweisung des Ergebnisses an die skalare Variable ϕ beendet den gesamten Prozess.

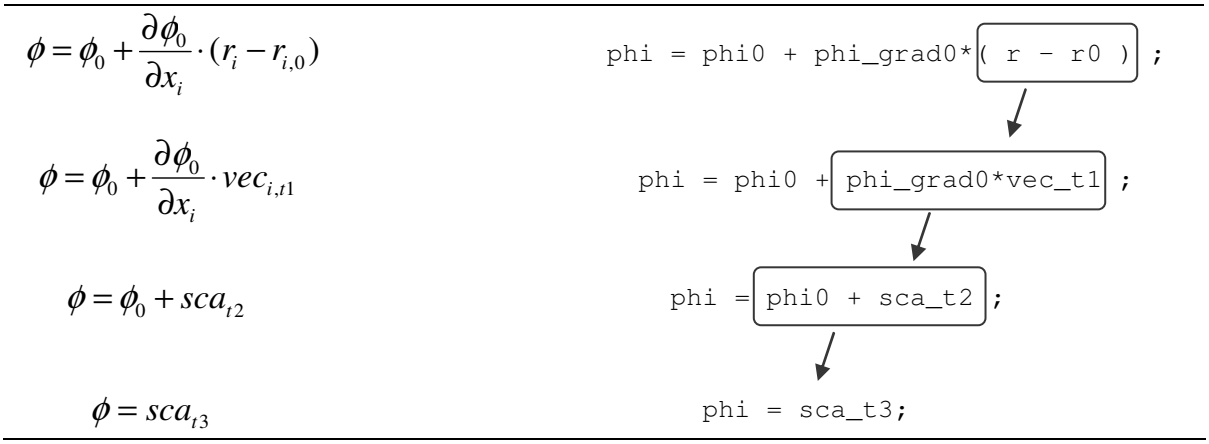


Bild 3-10: Entstehung temporärer Objekte bei der paarweisen Evaluation von Operatoren in C++ am Beispiel von Modellgleichung (3.2)

Das Anlegen und Löschen temporärer Objekte verursacht hohe Kosten. Es muss als Hauptursache für die schlechte Performance des herkömmlich objektorientierten Designs angeführt werden. Trotz der Optimierungstechniken moderner Compiler bleiben die temporären Objekte in den Ausdrücken bestehen, weil sie für die Weiterverarbeitung der einzelnen Zwischenergebnisse benötigt werden, siehe hierzu auch MEYERS [58] und HÄRDTLEIN [35]. Aus diesen Gründen ist das traditionelle Operatorüberladen in C++ für performance-relevante Bereiche wie dem wissenschaftlichen Hochleistungsrechnen kaum geeignet und kann mit handoptimierten Implementierungen, die die Tensorarithmetik explizit komponentenweise auflösen, nicht mithalten. Andererseits verlieren die Rechenergebnisse von schnellen aber fehlerhaften Programmen ihre Aussagekraft. Außerdem wird im wissenschaftlichen Bereich die qualitative Bewertung unterschiedlicher physikalischer Modelle unmöglich, wenn deren fehlerfreie Implementierung nicht gewährleistet werden kann.

Im Softwareentwurf des CFD-Tools werden die Designziele hohe Geschwindigkeit und leicht lesbare Tensorarithmetik gleich gewichtet. Die Umsetzung dieser Designvorgaben geschieht durch den Entwurf sogenannter ET, einer speziellen Technik auf dem Gebiet der TMP. Der Einsatz von ET ermöglicht es, die beschriebenen Performanceprobleme mit den herkömmlich überladenen Operatoren zu umgehen ohne die Vorteile einer kompakten Schreibweise zu verlieren, vgl. hierzu auch VELDHUIZEN [100].

Die folgenden Teilabschnitte beschreiben den Entwurf der ET für die Tensorarithmetik im CFD-Tool. Um die grundlegende Idee hinter ET zu verstehen, wird zunächst der Aufbau von Syntaxbäumen und deren traditionelle Umsetzung in einer möglichen Klassenhierarchie beschrieben. Aus den daraus ersichtlichen Gesetzmäßigkeiten leitet sich dann die Implementierung der ET ab.

3.3.2 Syntaxbäume zur Beschreibung der arithmetischen Ausdrücke

Bei der Auswertung eines arithmetischen Ausdrucks ist es im Allgemeinen vorteilhaft, dessen strukturellen Aufbau in der Gesamtheit zu erfassen. Das liegt daran, dass eine bestimmte Operation innerhalb des Ausdruckes jeweils auf alle Operanden angewandt werden muss, die Operanden aber selbst wieder Unterausdrücke bilden können. Diesem Sachverhalt wird übrigens automatisch entsprochen, wenn jemand einen Ausdruck per Hand von der kompakten in die komponentenweise Darstellung umformt.

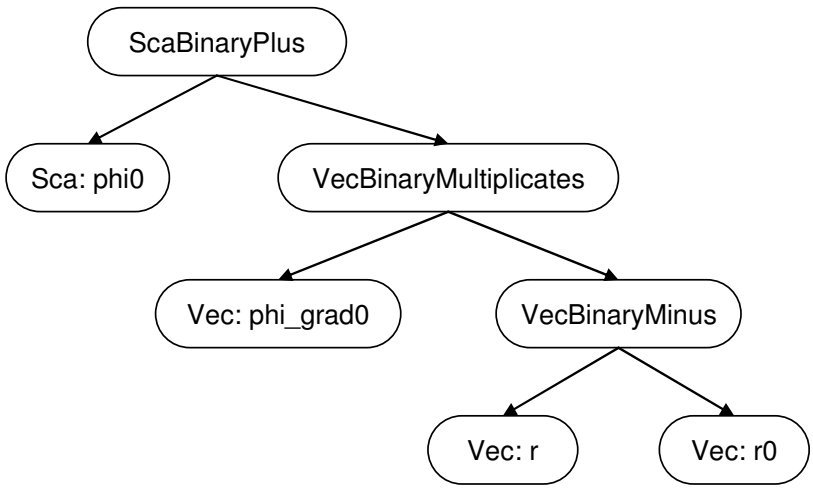


Bild 3-11: Graphische Darstellung des Syntaxbaumes nach dem Parsen von Modellgleichung (3.2)

Der strukturelle Aufbau eines Ausdruckes wird durch seine Syntax beschrieben. Die darin enthaltenen Informationen, das heißt die Operatoren, die Operanden und die Reihenfolge, mit der die Operatoren auf die Operanden angewendet werden müssen, kann durch einen Syntaxbaum ausgedrückt werden. Analysiert man beispielsweise Modellgleichung (3.2), so ergibt sich der Syntaxbaum, der in Bild 3-11 abgebildet ist. Der Baum ist dabei von Oben nach Unten zu lesen. Die Operatoren bilden jeweils die Verzweigungen des Baumes, an deren Ästen die Operanden eine Ebene tiefer sitzen. Ist ein Operand ein terminaler Ausdruck, sprich eine Variable, so bildet er ein Blatt, ist er dagegen wieder eine Operation, bildet er einen weiteren Verzweigungspunkt. In Modellgleichung (3.2) sind nur

binäre Operatoren, also Operatoren durch die genau zwei Operanden verknüpft werden, enthalten. Der abgebildete Syntaxbaum hat deshalb immer genau zwei Zweige an jedem Knoten. Natürlich können in einer Gleichung auch unäre Operatoren, also Operatoren durch die nur ein Operand verarbeiten wird, auftreten. Im zugehörigen Syntaxbaum entspringt dann am Operatorknoten nur ein Ast.

Die Verarbeitung eines Syntaxbaumes geschieht im Gegensatz zu seiner Erzeugung von unten nach oben. Dies erklärt sich dadurch, dass ein Operatorknoten immer erst dann ausgewertet werden kann, wenn die Werte der Operanden an seinen Ästen schon bekannt sind. Im Syntaxbaum aus Bild 3-11 muss demnach als Erstes die Subtraktion der Vektoren *r* und *r0* durchgeführt werden. Mit diesem Ergebnis kann als nächstes die Skalar-Multiplikation mit dem Gradienten *phi_grad0* und danach die Summation mit der Variablen *phi0* durchgeführt werden. Diese Verarbeitungsvorschrift entspricht genau dem Ablaufplan, der bereits bei der paarweisen Evaluation der Operatoren in Bild 3-10 dargestellt worden ist.

3.3.3 Implementierung im klassisch objektorientierten Design

Programmiertechnisch kann der Syntaxbaum für die Ausdrücke als Sonderfall eines Kompositums betrachtet werden. Das Kompositum stellt ein bekanntes Entwurfsmuster der objektorientierten Programmierung dar und ist beispielsweise im GoF-Buch [32] beschrieben. Die Teil-Ganzes Beziehung des Kompositums entspricht dabei den Verknüpfungen zwischen den Operatoren und ihren Operanden im Syntaxbaum. Bei der Implementierung des Musters werden die Verknüpfungsvorschriften genauso wie die Variablen durch konkrete Klassen dargestellt und von einer gemeinsamen abstrakten Basisklasse abgeleitet. Durch ein derartiges Design ist es möglich, sowohl terminale wie nicht terminale Ausdrücke einheitlich zu behandeln und eine hierarchische Objektstruktur aufzubauen. Letztere ist Grundvoraussetzung zur Nachbildung der in einem arithmetischen Ausdruck enthaltenen Syntax. Die Interpretation der aufgebauten Struktur wird dann durch Aufruf einer Zugriffsfunktion auf die Objektkomponenten, die in der Basisklasse des Entwurfs definiert ist, ausgelöst. In den abgeleiteten Kindklassen muss die Zugriffsfunktion dann jeweils ausimplementiert werden. Die Implementierung erfolgt dabei nach einem einfachen Schema: Verkörpert die Klasse eine Variable, so liefert der Aufruf der Zugriffsfunktion den Variablenwert zurück, repräsentiert die Klasse dagegen eine Verknüpfungsoperation, so bewirkt der Aufruf, dass die zugehörige Operation ausgeführt wird. Dies zieht dann wiederum den Aufruf der Zugriffsfunktionen der Operanden nach sich. Durch diese einfache Vorschrift wird die Objektstruktur sukzessive abgearbeitet. Der rekursive Prozess wird beendet, wenn alle Operanden einer Verknüpfungsoperation aus Variablen bestehen, und das Ergebnis der Operation nach oben weitergereicht werden kann. Das entspricht genau der beschriebenen Funktionsweise des Syntaxbaumes.

Da der Aufruf der Zugriffsfunktionen komponentenweise erfolgt, muss bei ihrer Implementierung der Rang der Ausdrücke berücksichtigt werden. Im CFD-Tool gibt es, bezogen auf die Definition in Kapitel 3.1, drei Typen von Variablen: Skalare, Vektoren und Tensoren. Für jeden Typ wird eine abstrakte Basisklasse vorgesehen, von der dann sowohl die Variable selbst als auch die Verknüpfungsoperationen der Variable abgeleitet werden. Insgesamt entstehen so drei Klassenhierarchien nach dem Entwurfsmuster des

Kompositums. Für die Zuordnung einer Operation zur richtigen Hierarchie ist nur der Typ ihres Ergebnisses ausschlaggebend. Betrachtet man Modellgleichung (3.2), bedeutet das beispielsweise, dass das Skalarprodukt zwischen dem Gradientenvektor *phi_grad0* und der Differenz aus den beiden Ortsvektoren *r* und *r0*, zu den skalaren Ausdrücken gehört. Die Operanden des Skalarproduktes werden dagegen zu den vektoriellen Ausdrücken gezählt. Dieses Vorgehen stellt eine typsichere Implementierung der gesamten Tensorarithmetik sicher. Fehlerhafte Konstrukte, die durch die Verknüpfung zweier nicht zueinanderpassender Typen entstehen, werden so automatisch vom Compiler erkannt und angezeigt. So macht das betrachtete Skalarprodukt nur dann Sinn, wenn beide Operanden vektorielle Ausdrücke sind.

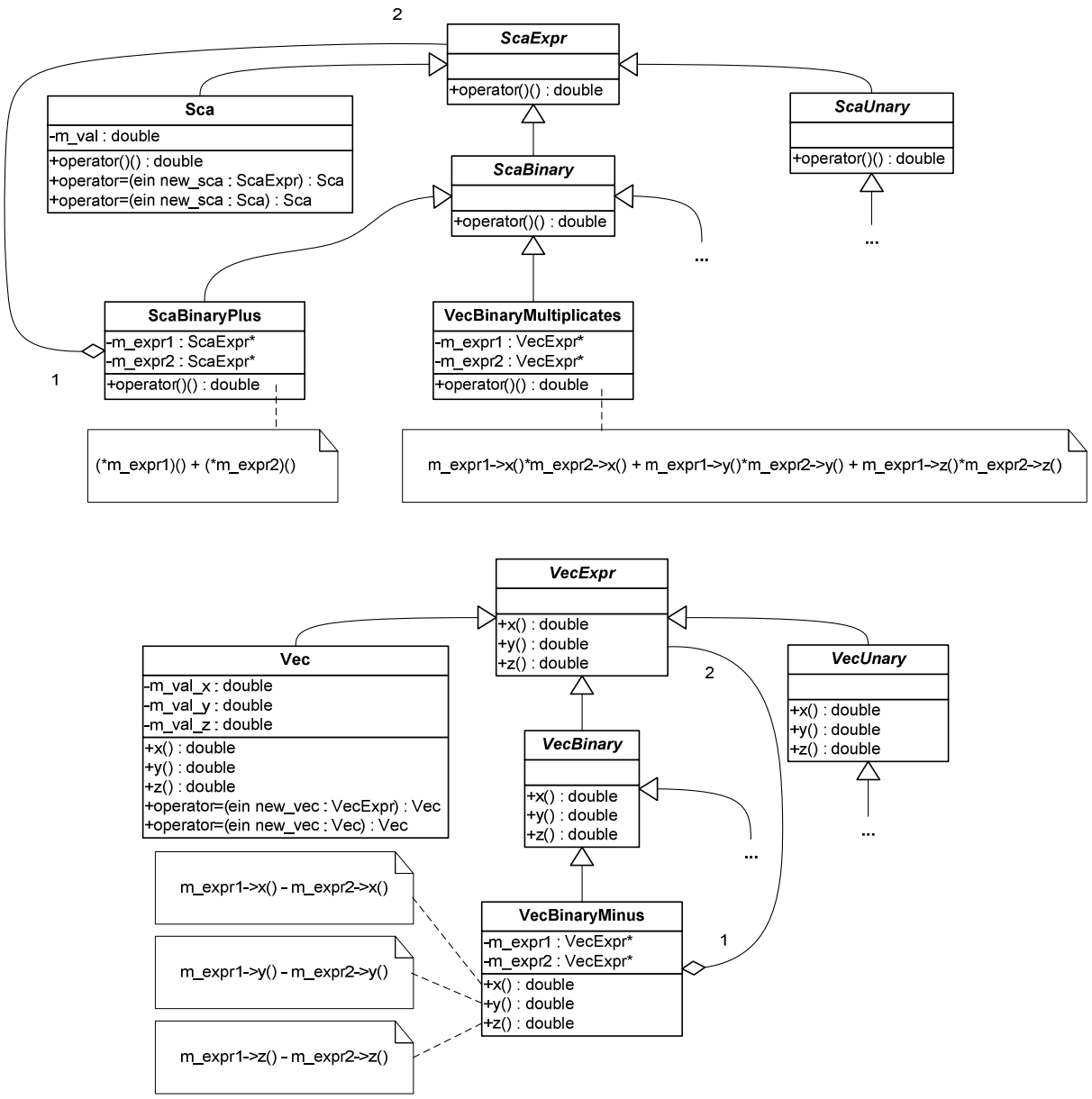


Bild 3-12: Klassendiagramm für die objektorientierte Implementierung zur Interpretation skalarer und vektorieller Ausdrücke mit Hilfe der Entwurfsvorlage des Kompositums [32]

Der resultierende Klassenentwurf zur Interpretation skalarer und vektorieller Ausdrücke ist in Bild 3-12 dargestellt. Zur besseren Übersicht sind dabei lediglich diejenigen Operatorklassen gelistet, die konkret bei der Auswertung von Modellgleichung (3.2) eingesetzt werden. Für das Verständnis des Entwurfs ist das völlig ausreichend. Die Stellen, an denen neue Operatorklassen in die Klassendiagramme eingefügt werden können, sind darüber hinaus jeweils durch drei Punkte gekennzeichnet. Die Zugriffsfunktionen für skalare Ausdrücke sind über die Klammeroperatoren definiert. Die Zugriffsfunktionen für vektorielle Ausdrücke sind dagegen über Member-Funktionen, die den jeweiligen Komponentennamen tragen, definiert. Ergänzend sind für die aufgeführten Verknüpfungsoperationen die Rechenvorschriften angegeben, die beim Aufruf ihrer Zugriffsfunktionen abgearbeitet werden müssen.

Nun bleibt noch die Frage zu klären, was den Aufruf der Zugriffsfunktionen bewirkt und gegebenenfalls die Evaluation des durch die Objektkomposition verkörperten Syntaxbaums auslöst. Die Antwort findet sich in den Zuweisungsoperatoren der Variablen, die automatisch die Zugriffsfunktionen der übergebenen Ausdrücke aufrufen. In den Klassenentwürfen von Bild 3-12 sind jeweils zwei spezialisierte Versionen für die Zuweisungsoperatoren angegeben. Die eine ist für die explizite Zuweisung eines Variablenwertes bestimmt, die andere für die Zuweisung eines Ausdrucks und dem darin implizit enthaltenen Ergebnis.

In Verbindung mit dem Aufbau von Syntaxbäumen ermöglicht der vorgestellte objektorientierte Klassenentwurf ganz allgemein die verzögerte und komponentenweise Auswertung arithmetischer Ausdrücke. Im Vergleich zur paarweisen Evaluation mittels überladener Operatoren zeigt das Design aber trotzdem keinerlei Performancevorteile. Durch die Regeln der Vererbung müssen die in den abstrakten Basisklassen definierten Zugriffsfunktionen als virtuell deklariert werden. Nur so wird zur Laufzeit die richtige Version der jeweiligen Kindklasse aufgerufen. Der Compiler kann für diese Funktionen damit aber keine Inlining-Optimierungen durchführen. Eine späte Bindung während der Laufzeit ist die Folge. Im entstehenden Overhead beim Funktionsaufruf liegt dann die schlechte Performance des Designs begründet. Für den Syntaxbaum muss also eine effizientere Implementierungstechnik gefunden werden, durch die sämtliche Optimierungen bereits zur Übersetzungszeit ablaufen können.

3.3.4 Implementierung mit Expression Templates (ET)

Das klassische Design zur Implementierung der Syntaxbäume basiert auf der Verwendung dreier abstrakter Basisklassen als Grundlage für die verschiedenen Skalar-, Vektor- und Tensorausdrücke. Die einzelnen Basisklassen enthalten dabei keinerlei eigene Daten. Sie definieren lediglich den rein virtuellen Komponentenzugriff, der die gemeinsame Schnittstelle der konkret abgeleiteten Kindklassen bildet. Durch den Vererbungsmechanismus wird zur Laufzeit sichergestellt, dass immer die richtige Memberfunktion aufgerufen wird, auch dann, wenn ein Objekt wie in den unären und binären Operationsklassen als Zeiger vom Typ der Basisklasse abgespeichert ist. Dieses Verhalten bezeichnet man in der objektorientierten Welt als Laufzeit Polymorphismus. Es gehört zu den grundlegenden Vorzügen einer jeden objektorientierten Sprache und macht die Implementierung als Kompositum erst möglich. Der Laufzeit-Polymorphismus ist gleichzeitig aber gerade für die schlechte Performance des klassischen Entwurfs verantwortlich. C++

ermöglicht durch die TMP, diesen Nachteil zu neutralisieren und eine wesentlich schnellere Implementierungsvariante für die Tensorarithmetik im CFD-Tool zu entwickeln.

In der objektorientierten Programmierung werden nach LANGER und KREFT [49] gemeinsame Eigenschaften durch gemeinsame Basisklassen ausgedrückt, in der TMP dagegen mittels Namenskonventionen und Gemeinsamkeiten bei der Namensgebung. An die Stelle der virtuellen Funktionen des objektorientierten Ansatzes treten also einfache nicht-virtuelle Funktionen, die lediglich einen bestimmten Namen und eine kompatible Signatur besitzen. Konsequenterweise erscheinen abstrakte Basisklassen bei der TMP dann überflüssig zu sein, da sie ja lediglich das Interface für den virtuellen Komponentenzugriff im objektorientierten Ansatz bereitstellen. Das ist aber nur eingeschränkt richtig, denn die zwangsweise Zugehörigkeit eines Ausdruckes zu einer der drei Klassenhierarchien begründet die Typsicherheit des klassischen Designs. Ergo darf das Erben von den gemeinsamen Basisklassen nicht entfallen, denn auch die templatebasierte Variante der Syntaxbäume soll Typsicherheit gewährleisten. Per Definition muss ein skalarer Ausdruck auch weiterhin klar von einem vektoriellen oder tensoriellen Ausdruck unterscheidbar bleiben. Es gilt also einen Mechanismus zu hinterlegen, bei dem das polymorphe Verhalten der Zugriffsfunktionen nicht über virtuelles Ableiten realisiert wird.

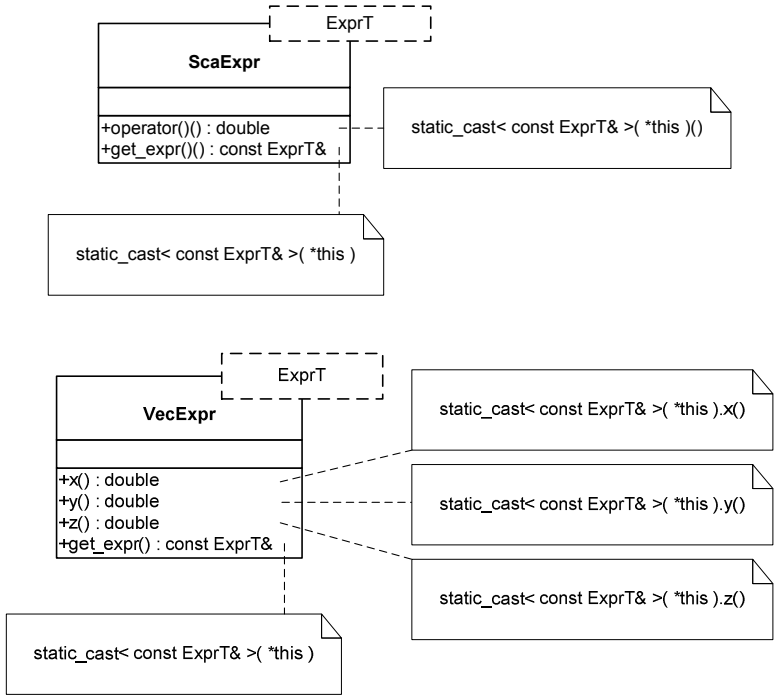


Bild 3-13: Skalare und vektorielle Grundtypen im templatebasierten Klassenentwurf der Tensorarithmetik; Die Member-Funktionen für den Komponentenzugriff setzen den Compilezeit-Polymorphismus des CRTP um [15]

Für den templatebasierten Neuentwurf der Klassen wird eine Technik der TMP genutzt, die als Curiously Recurring Template Pattern (CRTP) bekannt ist. Das von COPLIEN [15] geprägte Idiom ersetzt den langsamen und damit nachteiligen Laufzeit-Polymorphismus durch einen statischen Polymorphismus, der vom Compiler bereits bei der Programmerstellung aufgelöst werden kann. Das CRTP wird häufig in Kombination mit dem Barton-Nackman-Trick präsentiert, darf aber nicht mit diesem verwechselt werden, siehe

VANDEVOORDE und JOSUTTIS [98]. Es nutzt die Tatsache aus, dass eine Klasse ihre eigene Kindklasse rekursiv als Template-Parameter erhalten kann. Damit ist der Basisklasse der Typ ihrer Kindklasse bereits zur Compilezeit vollständig bekannt und jeder Funktionsaufruf kann korrekt aufgelöst werden.

Der Basisklassen-Entwurf zur typsicheren templatebasierten Implementierung der Syntaxbäume ist in Bild 3-13 dargestellt. Wie beim klassischen Design gibt es gemäß den drei verschiedenen Variablentypen, Skalar, Vektor und Tensor, auch beim templatebasierten Entwurf drei Klassenstämme, nämlich *ScaExpr*, *VecExpr* und *TenExpr*. Die Klasse *TenExpr* ist aus Platzgründen in Bild 3-13 allerdings nicht dargestellt. Die Klassenstämme sind nun nicht mehr abstrakt. Zur Ausbildung des Compilezeit-Polymorphismus besitzen sie jeweils den Template-Parameter *ExprT*, durch den ihnen der abgeleitete Klassentyp rekursiv mitgeteilt wird. Der Komponentenzugriff ist im Gegensatz zum klassischen Design nicht-virtuell. Stattdessen wird der Komponentenaufruf einfach durch ein statisches Cast an die im Template-Parameter angegebene Kindklasse weitergereicht und somit der Polymorphismus der herkömmlichen Vererbung imitiert. Da die Member-Funktionen nun nicht länger virtuell sein müssen, kann sie der Compiler wegoptimieren. Auf diese Weise wird verglichen mit dem klassischen Design wesentlich schnellerer Maschinencode erzeugt. Die Basisklassen besitzen zusätzlich die Methode *get_expr()*. Diese stellt eine reine Zweckdienlichkeit dar und verkürzt die Konvertierung der Basisklassen in den Klassentyp des Template-Parameters. Ein Anwendungsbeispiel für die Funktion *get_expr()* findet sich im nächsten Abschnitt, der sich mit den Creator-Funktionen beschäftigt.

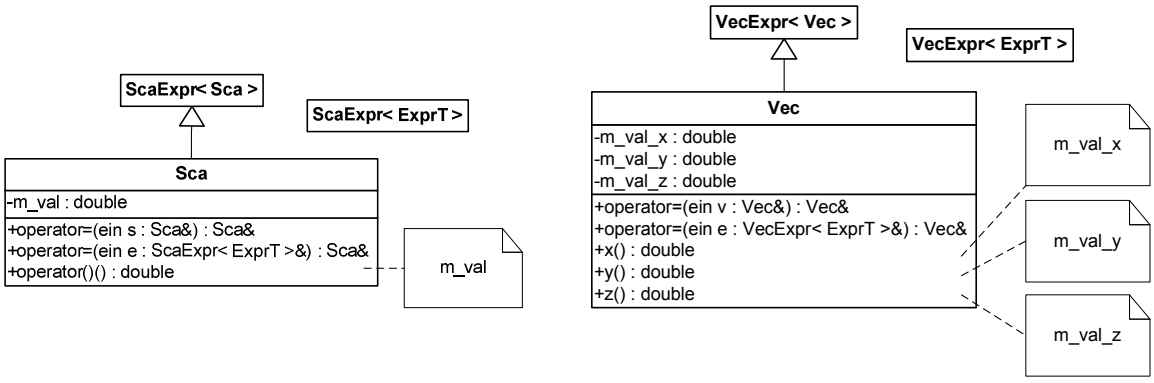


Bild 3-14: Skalare und vektorielle Datentypen im templatebasierten Klassenentwurf der Tensorarithmetik; Die Memberfunktionen für den Komponentenzugriff liefern die gespeicherten Werte zurück; Für Daten und Ausdrücke sind jeweils gesondert Zuweisungsoperatoren definiert

Die Klassen für die eigentlichen Variablentypen des CFD-Tools werden nun mithilfe des CRTP von den drei Basisklassen-Templates abgeleitet. Durch die Rekursion übergibt sich jeder Variablentyp im Template-Parameter an seine Basisklasse. Die Variablentypen selbst besitzen keine Template-Parameter mehr, sie sind ganz normale Klassen. Den zugehörigen Entwurf der Klassen *Sca* bzw. *Vec* für die Datentypen Skalar bzw. Vektor zeigt Bild 3-14. Die Klasse *Ten* für den Datentyp Tensor entfällt wieder aus Platzgründen. Sie wird bei der syntaktischen Analyse von Modellgleichung (3.2) nicht benötigt. Für den konkreten Entwurf einer Tensorklasse können die beiden abgebildeten Datentypen *Sca* und *Vec* als direkte Vorlage benutzt werden. Variablentypen stellen terminale Ausdrücke

dar. Somit müssen die komponentenweisen Zugriffsfunktionen ihrer Klassen wirklich abgespeicherte Werte zurückliefern. Neben diesen Zugriffsfunktionen besitzen die Klassen außerdem noch einen Zuweisungsoperator *operator=()*, der in zwei unterschiedlichen Versionen aus-geführt wird. Die Erste sieht allgemein die Zuweisung von Ausdruckstypen, die zweite speziell die von Variablentypen vor. Diese bereits vom klassischen Design her bekannte Diversifikation ermöglicht eine aggressive Optimierung bei der Zuweisung typgleicher Variablenklassen. Nebenbei bemerkt macht es durchaus Sinn, auch entsprechende Über-ladungen für die kombinierten Operatoren *operator+=()* und *operator-=()* in den Klassen vorzusehen. Der vorliegende Entwurf berücksichtigt das aber nicht.

Als nächstes werden die Klassen für unäre Ausdrücke im CFD-Tool von den drei Basisklassen-Templates *ScaExpr*, *VecExpr* und *TenExpr* abgeleitet. Jeder Ausdruck übergibt sich wieder im Template-Parameter an seine Basisklasse und implementiert den Compilezeit-Polymorphismus der CRTP-Technik. Die Ausdrücke sind selbst ebenfalls wieder Klassen-Templates. Sie werden mit den Typen des Operanden und der jeweiligen Rechenoperation parametriert. Hervorzuheben ist, dass die Operation eine eigenständige Hilfsklasse darstellt, die eine Schnittstelle für die komponentenweise Evaluierung besitzt. Man vergleiche diese Entwurfsstrategie mit dem klassischen Design. Dort ist eine Operation immer direkt in einen extra für sie spezialisierten Ausdruck integriert. Die Zuordnung eines Ausdrucks zu einem der drei Klassenstämme funktioniert wie im klassischen Design. Sie richtet sich jeweils nach dem Ergebnis der Operation. Bild 3-15 zeigt die beiden UML-Diagramme der Klassen *ScaUnary* bzw. *VecUnary* für unäre Skalar- bzw. Vektorausdrücke. Aus Platzgründen ist das entsprechende Klassen-Template *TenUnary* für unäre Tensorausdrücke nicht dargestellt. Es ist so wie die beiden abgebildeten Klassenentwürfe aufgebaut. Die Methoden für den Komponentenzugriff verwenden die Evaluierungsfunktionen der oben beschriebenen Operationsklassen. Somit wird wie im klassischen Entwurf durch den Komponentenzugriff implizit auch die Evaluierung der Ausdrücke angestoßen. Beispiele für die in Modellgleichung (3.2) benötigten Operationsklassen sind in Bild 3-16 zu finden. In der Gleichung sind zwar nur binäre Ausdrücke enthalten, die Operationsklassen für die hier besprochenen unären Ausdrücke besitzen jedoch denselben Aufbau, sieht man davon ab, dass ihre Evaluierungsfunktionen nur einen Operanden als Übergabeparameter akzeptieren.

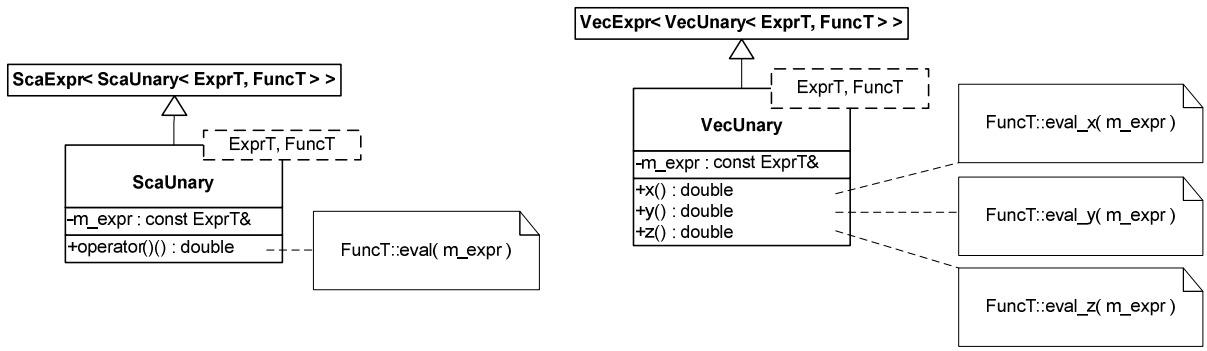


Bild 3-15: Unäre Skalar- und Vektorausdrücke im templatebasierten Klassenentwurf der Tensorarithmetik; Die Operationsklassen zur komponentenweisen Evaluierung werden als Template-Parameter übergeben

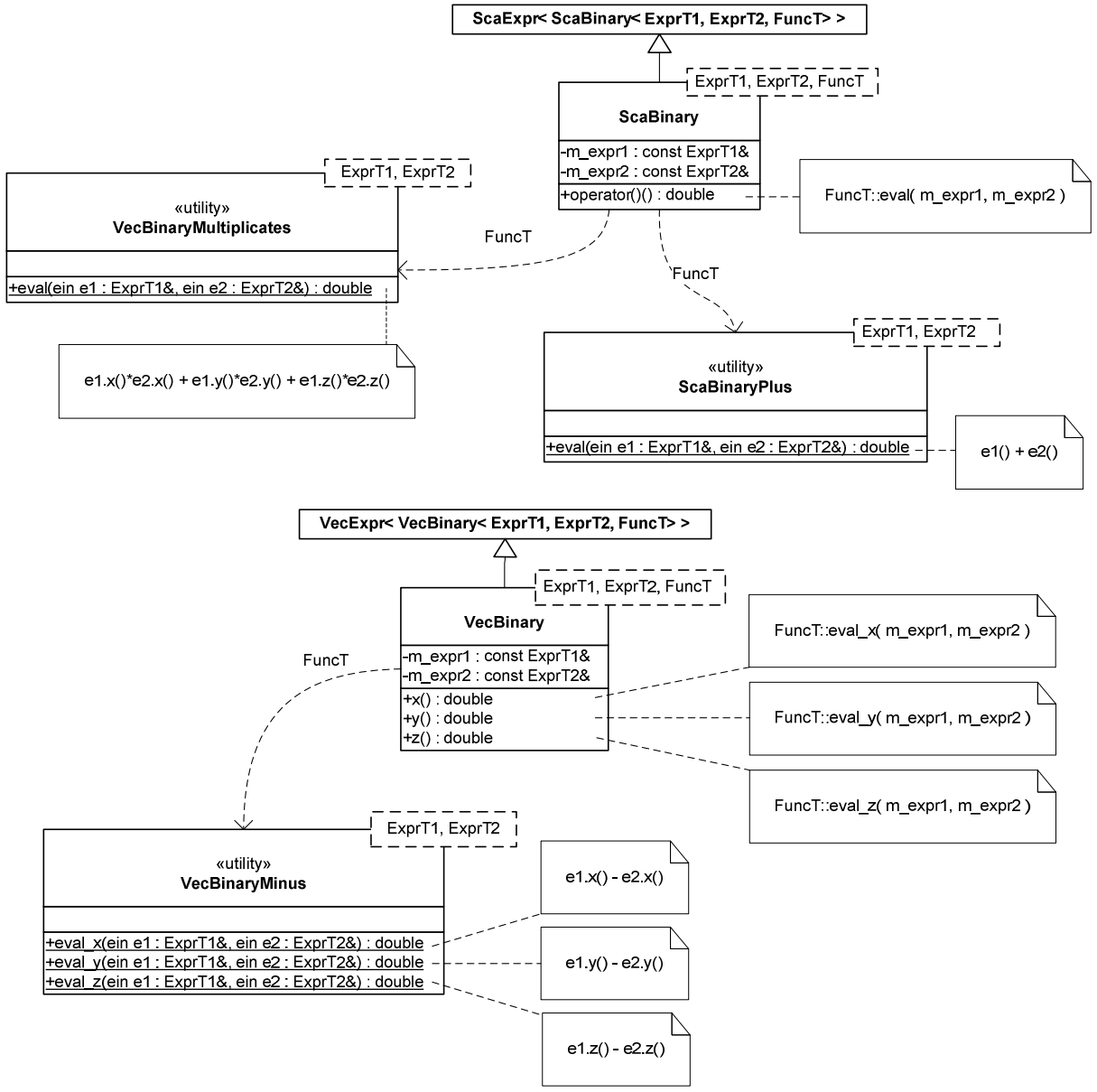


Bild 3-16: Binäre Scalar- und Vektorausdrücke im templatebasierten Klassenentwurf der Tensorarithmetik; Die Verknüpfungsoperationen sind in eigenständigen Operationsklassen enthalten; Es sind nur die zur Evaluierung von Modellgleichung (3.2) benötigten Operationsklassen abgebildet

Die Klassen für binäre Ausdrücke komplettieren die templatebasierte Implementierung der Syntaxbäume im CFD-Tool. Sie werden in gewohnter Weise rekursiv von den drei Basisklassen-Templates *ScaExpr*, *VecExpr* und *TenExpr* abgeleitet. Die erzeugten Ausdrücke bilden selbst wieder Klassen-Templates. Ihre Template-Parameter speichern die Typen der beiden Operanden, sowie der Verknüpfungsoperation. Das resultierende Design entspricht damit prinzipiell dem der unären Ausdrücke, siehe oben. In Bild 3-16 sind die beiden Klassen-Templates *ScaBinary* bzw. *VecBinary* für binäre Skalar- bzw. Vektorausdrücke zu sehen. Das Diagramm des dritten Klassen-Templates *TenBinary* für binäre Tensor-Ausdrücke entfällt wieder. Es gleicht denen der beiden angegebenen Entwürfen. Die Operationsklassen kapseln die eigentlichen Rechenoperationen für die Ausdrücke. Der Zugriff auf eine Operation erfolgt dabei jeweils mithilfe der Methoden zur

komponentenweisen Evaluierung, die alle Operationsklassen bereitstellen. Die Operationsklassen stellen reine Funktionsobjekte dar. Sie sind deshalb als Hilfsklassen konzipiert worden, welche ausschließlich statische Methoden besitzen, vgl. BOOCH [12]. Das bringt den Vorteil, dass eine Operationsklasse nicht extra instanziiert werden muss, bevor ihre Methoden zur komponentenweisen Evaluierung aufgerufen werden können. Die ohnehin notwendige Parametrierung eines Ausdrucks mit dem Typ der Operationsklasse ist völlig ausreichend. Die Kapselung der Operationen in eigenständigen Hilfsklassen macht außerdem die Implementierung neuer Operationen sehr bequem. So ist lediglich auf die Schnittstellenkonvention zu achten, die das Vorhandensein der Methoden zur komponentenweisen Evaluierung voraussetzt. Bild 3-16 zeigt beispielhaft die Entwürfe für die Operationsklassen, die zur Parametrierung der binären Ausdrücke in Modellgleichung (3.2) benötigt werden. Im Einzelnen sind das die Klassen *VecBinaryMultiplicates* zur Bildung des Skalarproduktes zweier Vektoren, *ScaBinaryPlus* zur Addition zweier Skalare sowie *VecBinaryMinus* zur Subtraktion zweier Vektoren. Bei der Instanziierung der Ausdrücke ist darauf zu achten, dass der Typ eines Ausdruckes mit dem Ergebnis der ihn parametrierenden Operation übereinstimmen muss. Die Operationstypen *VecBinaryMultiplicates* und *ScaBinaryPlus* sind beispielsweise nur für binäre Skalarausdrücke bestimmt, wohingegen der Typ *VecBinaryMinus* binäre Vektorausdrücke voraussetzt.

Durch die Nutzung der CRTP-Technik kann in der template-basierten Implementierung der Ausdrücke auf den Laufzeit-Polymorphismus ganz verzichtet werden. Dies führt zu einer höchst effizienten Implementierung. Denn ohne den traditionellen Laufzeitpolymorphismus bietet die CRTP-Technik eine wesentlich besser optimierbare Implementierung für die Tensorarithmetik im CFD-Tool, vgl. hierzu auch ISERNHAGEN und HELMKE [40]. Insgesamt führt die Implementierung mit ET zu einer Ansammlung lose gekoppelter Klassenschablonen für die arithmetischen Ausdrücke im CFD-Tool. Es sei noch einmal betont, dass die Einführung einer neuen Operation explizit zu keiner neuen Ausdrucksklasse führt. Vielmehr wird die bestehende Klassenschablone für die binären bzw. unären Ausdrücke einfach mit der neuen Operation parametriert. Damit unterscheidet sich der templatebasierte Entwurf grundsätzlich vom klassischen Design.

3.3.5 Creator-Funktionen und überladene Operatoren

Die Anwendung der ET-Technik schafft eine effiziente Implementierung für die Tensorarithmetik im CFD-Tool. Die entwickelten Klassen-Templates für die arithmetischen Ausdrücke bilden die mathematischen Formeln als Syntaxbäume ab. Dadurch ist es möglich, die komponentenweise Auswertung einer Gleichung vom Compiler erledigen zu lassen. Durch die Klassen-Templates alleine ergibt sich allerdings noch keine benutzerfreundliche Klassen-Bibliothek. Bislang muss der Programmierer die Syntaxbäume, die die Formeln repräsentieren, von Hand erstellen. Dabei entstehen im Allgemeinen komplexe und geschachtelte Template-Aufrufe. Die Parameter eines Klassen-Templates sind bei dessen Instantiierung immer vollständig anzugeben. Für eine anwenderfreundliche Implementierung zur Vermeidung von Programmierfehlern ist das völlig inakzeptabel. Die Bildung der Syntaxbäume sollte vor dem Programmierer vollständig verborgen bleiben. Eine elegante Lösung für das vorliegende Problem bietet die Einführung sogenannter Creator-Funktionen. Das sind Funktions-Templates, die dabei helfen, die Erstellung von Klassen-Templates wesentlich zu vereinfachen. Denn im

Gegensatz zu Klassen-Templates müssen die Parameter von Funktions-Templates nicht immer explizit angegeben werden. Der Compiler kann diese selbst herleiten, wenn sie durch die einzelnen Argumenttypen der Funktions-Templates eindeutig hervorgehen (automatic type deduction, siehe STROUSTRUP [90]).

Tabelle 3-3: Liste aller überladenen unären und binären Operatoren zur Erstellung arithmetischer Ausdrücke; s und t sind Skalare; a_i und b_i sind 3-Vektoren; a_{ij} und b_{ij} sind 3x3-Matrizen

Unäre Operatoren	Typ des Operanden	Typ des Ergebnisses	Schreibweise in der Mathematik	Schreibweise im CFD-Tool
Minus	Sca	Sca	$-s$	- s
	Vec	Vec	$-a_i$	- a
	Ten	Ten	$-a_{ij}$	- A
Plus	Sca	Sca	$+s$	+ s
	Vec	Vec	$+a_i$	+ a
	Ten	Ten	$+a_{ij}$	+ A
Binäre Operatoren	**Typen der Operanden**	**Typ des Ergebnisses**	**Schreibweise in der Mathematik**	**Schreibweise im CFD-Tool**
Subtraktion	Sca	Sca	$s-t$	s - t
	Vec	Vec	a_i-b_i	a - b
	Ten	Ten	$a_{ij}-b_{ij}$	A - B
Addition	Sca	Sca	$s+t$	s + t
	Vec	Vec	a_i+b_i	a + b
	Ten	Ten	$a_{ij}+b_{ij}$	A + B
Matrizenmultiplikation	Ten, Vec	Vec	$a_{ij}\cdot b_j$	A * b
	Ten, Ten	Ten	$a_{ij}\cdot b_{jk}$	A * B
Skalarmultiplikation	Sca	Sca	$s\cdot t$	s * t
	Sca, Vec	Vec	$s\cdot a_i$	s * a
	Vec, Sca	Vec	$a_i\cdot s$	a * s
	Sca, Ten	Ten	$s\cdot a_{ij}$	s * A
Skalardivision	Sca	Sca	t/s	t / s
	Vec, Sca	Vec	a_i/s	a / s
	Ten, Sca	Ten	a_{ij}/s	A / s
Skalarprodukt (inneres Produkt)	Vec	Sca	$a_i\cdot b_i$	a * b

Bei der Parametrierung der Klassen-Templates für die arithmetischen Ausdrücke im CFD-Tool muss der Typ der Hilfsklasse für die jeweilige Operation angegeben werden. Die konkrete Instanz eines arithmetischen Ausdrucks ist fix mit dieser Operation verbunden. Es liegt nahe, die Creator-Funktionen über die Funktionsnamen direkt mit

den jeweiligen Operationen in Bezug zu setzen und ihnen lediglich die Operanden als Argumente zu übergeben. Ihr Aufruf im Quelltext entspricht dann weitgehend dem Erscheinungsbild mathematischer Funktionen. Besonders elegant ist die Verwendung überladener Operatoren als Creator-Funktionen, siehe HÄRDTLEIN [35] und LEHN [51]. Auf diese Weise bleiben im Quelltext auch die Verknüpfungssymbole der originären mathematischen Formeln erhalten. Darüber hinaus berücksichtigt der Compiler bei der Auswertung überladener Operatoren die üblichen Rechenregeln wie „Punkt vor Strich" sowie die Klammersetzung.

```
template< class ExprT1, class ExprT2 >
inline ScaBinary< ExprT1, ExprT2, ScaBinaryPlus< ExprT1, ExprT2 > >
operator+( const ScaExpr< ExprT1 >& e1, const ScaExpr< ExprT2 >& e2 )
{
   typedef ScaBinary< ExprT1, ExprT2, ScaBinaryPlus< ExprT1, ExprT2 > > Expr;
   return Expr( e1.get_expr(), e2.get_expr() );
}
```

Bild 3-17: Überladener Operator+ als Creator-Funktion zur impliziten Erstellung des Klassen-Templates für den zugehörigen arithmetischen Ausdruck

Bild 3-17 beschreibt beispielhaft die Erzeugung eines Klassen-Templates zur Addition zweier skalarer Ausdrücke. Für die Creator-Funktion wird der binäre Operator *operator+()* überladen. Als Argumente werden nur die beiden Operanden übergeben. Die Typdefinition innerhalb des Funktionsrumpfes ist der besseren Übersichtlichkeit halber gewählt und für die grundsätzliche Implementierung unerheblich. Der Operator ruft lediglich den Konstruktor des Klassen-Templates auf. Er ist deshalb als Inline-Funktion deklariert und kann vom Compiler wegoptimiert werden. Die restlichen Creator-Funktionen im CFD-Tool sind genau wie der überladene Operator *operator+()* aufgebaut. Die Creator-Funktionen für die unären Ausdrücke weisen allerdings nur ein Argument auf. Ist eine bestimmte arithmetische Operation gleichzeitig für Skalar-, Vektor- und Tensorausdrücke definiert, so ist die zugehörige Creator-Funktion auch als Skalar-, Vektor- und Tensorversion vorhanden.

Tabelle 3-4: Liste der wichtigsten Funktionen zur Erstellung der zugehörigen arithmetischen Ausdrücke; s ist ein Skalar; $\vec{a}$ und $\vec{b}$ sind 3-Vektoren; A ist eine 3x3-Matrix

Funktionen	Typ des Operanden	Typ des Ergebnisses	Schreibweise in der Mathematik	Schreibweise im CFD-Tool
Betrag	Sca	Sca	$\lvert s \rvert$	abs(s)
	Vec	Sca	$\lvert \vec{a} \rvert$	abs(a)
Quadrat	Sca	Sca	s^2	sqr(s)
Quadratwurzel	Sca	Sca	$\sqrt{s}$	sqrt(s)
Determinante	Ten	Sca	$\det \boldsymbol{A}$	det(A)
Kreuzprodukt	Vec	Vec	$\vec{a} \times \vec{b}$	cross(a, b)
…	…	…	…	…

Alle unären und binären Operatoren, die im CFD-Tool als Creator-Funktionen genutzt werden, listet Tabelle 3-3 auf. Die Operatoren bewirken, dass der Quelltext für arithmetische Formeln weitgehend der originären Schreibweise in der Mathematik entspricht. Es existieren allerdings nicht für alle arithmetischen Operationen überladene Operatoren. Die Creator-Funktionen sind in solchen Fällen normale Funktions-Templates. Um die gute Lesbarkeit des Quelltextes zu bewahren, lehnt sich die Namensgebung dieser Funktionen nahe an die gebräuchlichen Namen der mathematischen Operationen an. Tabelle 3-4 enthält einige wichtige Creator-Funktionen, die im CFD-Tool als einfache Funktions-Templates definiert sind. Die Liste ist nicht vollständig. Sie liefert lediglich einige Beispiele zum Vergleich der verschiedenen Schreibweisen in der Mathematik und im CFD-Tool.

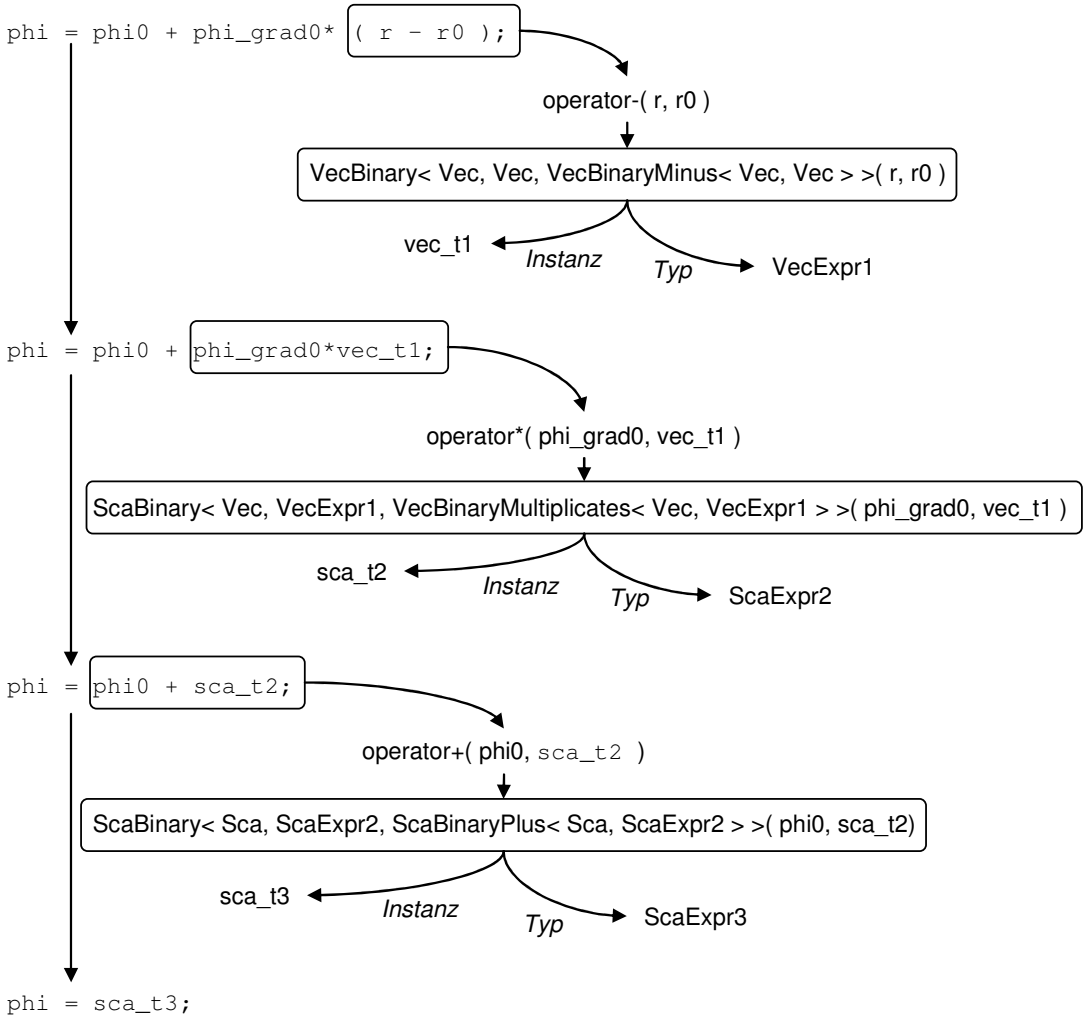


Bild 3-18: Ablauf bei der automatischen Erstellung des Syntaxbaums für Modellgleichung (3.2) durch Creator-Funktionen in Form überladener Operatoren

Die Creator-Funktionen machen die entwickelten ET des CFD-Tools zu einem hocheffizienten Werkzeug. Dies gilt sowohl für die überladenen Operatoren wie auch für die normalen Funktions-Templates. Die überladenen Operatoren funktionieren wie die entsprechenden mathematischen Symbole. Die einzelnen Operationen können einfach anein-

ander gehängt werden. Die Funktions-Templates werden wie gewöhnliche mathematische Funktionen benutzt. Sie akzeptieren als Argumente neben reinen Variablen genauso arithmetische Ausdrücke. Der Compiler leitet die notwendigen Instantiierungen aus den im Quelltext formulierten Gleichungen ab und erstellt die zugehörigen Syntaxbäume. Diese ermöglichen ihm anschießend die komponentenweise Evaluation der Gleichungen für die Wertzuweisung. Beide Prozessabläufe sind vollständig gekapselt und werden bei der Programmerstellung automatisch vom Compiler angestoßen.

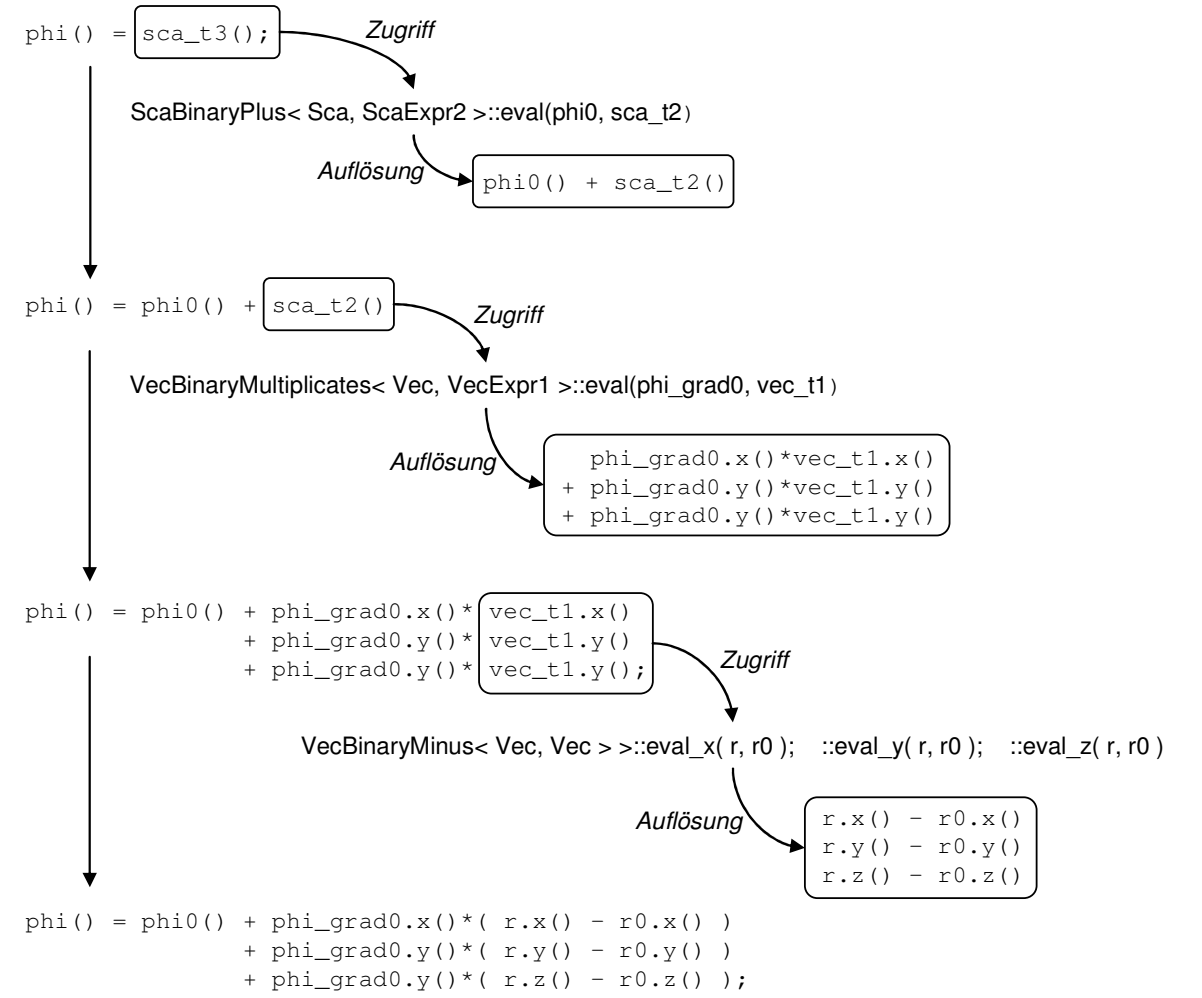


Bild 3-19: Ablauf bei der komponentenweisen Evaluation von Modellgleichung (3.2), ausgelöst durch den Aufruf des Zuweisungsoperators der Variablen *phi*

Die einzelnen Arbeitsschritte, die der Compiler nacheinander in den beiden Teilprozessen sinngemäß abarbeiten muss, sind in Bild 3-18 und Bild 3-19 zusammengefasst. Als Beispiel dient die bekannte Modellgleichung (3.2). Der Aufbau des Syntaxbaumes ist in Bild 3-18 dargestellt. Ausgehend von der ursprünglichen Gleichung löst der Compiler die einzelnen Operatoren sukzessive in der vorgeschriebenen Reihenfolge auf und bildet die Instanzen für die entsprechenden Ausdrücke. Nach und nach entsteht so der Syntaxbaum, der bei der komponentenweisen Evaluation der Gleichung im zweiten Teilprozess benötigt wird. Die definierten Typen und Instanzen in Bild 3-18 dienen nur der Über-

sichtlichkeit. Sie entstehen so natürlich nicht explizit während der Programmerstellung. Bild 3-19 beschreibt die folgende komponentenweise Evaluation. Diese wird durch den Zugriff auf die Hilfsvariable *sca_t3* bei der Wertzuweisung an die Variable *phi* ausgelöst. Da die Hilfsvariable in Wirklichkeit ein Ausdruck ist und den kompletten Syntaxbaum enthält, muss dieser schrittweise bis zu den terminalen Ausdrücken ausgewertet werden. So wird Modellgleichung (3.2) automatisch in ihre Komponentenschreibweise gebracht und im Maschinencode eingebettet. Die Teilschritte werden alle vom Compiler abgearbeitet. Die Gefahr von Fehlern bei der umständlichen komponentenweisen Formulierung der Gleichung durch den Programmierer wird vollständig umgangen.

Durch ET-Programmierung laufen die beiden Teilprozesse bereits während der Programmerstellung ab. Im Maschinencode sind die arithmetischen Gleichungen dann bereits vollständig in ihrer Komponentenschreibweise enthalten. Dadurch entstehen während der Programmausführung keine unnötigen Laufzeitverluste. Die Creator-Funktionen erlauben gleichzeitig, die Gleichungen im Quelltext in ihrer kompakten und übersichtlichen Tensor-Schreibweise zu formulieren.

3.3.6 Datenfelder und Adapterklassen

Für die Finite Volumen (FV) Diskretisierung müssen die meisten Variablen als Datenfelder angelegt werden. Sie beschreiben die diskrete Werteverteilung auf den Stützstellen des Rechennetzes. Die Klassen *ScaSet*, *VecSet* und *TenSet* sind dazu dar, die Datentypen Skalar, Vektor und Tensor als Feld zu speichern. Bild 3-20 zeigt die zugehörigen UML-Klassendiagramme für die Datenfeldklassem *ScaSet* und *VecSet*. Der Entwurf der Klasse *TenSet* für die Tensorfelder ist nicht dargestellt. Er entspricht denen der beiden abgebildeten Klassen *ScaSet* bzw. *VecSet*. Da die Datenfelder im CFD-Tool meist sehr viele Datenobjekte enthalten, gilt es, ein möglichst effizientes Speicherkonzept für die Datenfeldklassen zu entwickeln. So liegen deren Einträge nicht direkt objektbezogen als echte Skalare, Vektoren oder Tensoren, sondern komponentenbezogen als Rohdatensätze vor. Diese Strategie verspricht ein Höchstmaß an Speichereffizienz und sichert gleichzeitig einen konstant schnellen Indexzugriff auf die Komponenten. Die Member-Funktionen *resize()* bzw. *size()* ermöglichen die Änderung bzw. Abfrage der vorliegenden Feldgröße. Die Namensgebung lehnt sich an die entsprechenden Funktionen in den Containerklassen der C++ Standardbibliothek an.

Skalare, Vektoren und Tensoren sind terminale Ausdrücke. Das ändert sich natürlich nicht, wenn diese Variablentypen als Datenfeld angelegt sind. Die Funktionen der Datenfeldklassen, die den komponentenweisen Zugriff auf die Datenfelder ermöglichen, liefern deshalb auch hier echte Werte zurück. Sie benötigen allerdings noch zusätzlich die Position des Eintrags im Datenfeld als Argument. Für die Objektzuweisung besitzen die Datenfeldklassen nur eine Version des Zuweisungsoperators *operator=()*. Sie regelt die direkte Kopie typgleicher Datenfeldobjekte.

Die algebraischen Bilanzgleichungen, die bei der FV Diskretisierung entstehen, gelten im Allgemeinen nur in bestimmten Teilbereichen des Rechenfeldes. Das liegt daran, dass die Feldgrenzen wegen der Randbedingungen oft anders diskretisiert werden müssen als das Feldinnere. Im CFD-Tool werden die algebraischen Bilanzgleichungen generell für jedes Kontrollvolumen einzeln formuliert. Klassen-Templates für feldweise definierte algebra-

ische Ausdrücke gibt es nicht. Dementsprechend brauchen die Datenfeldklassen keinen Zuweisungsopertor für derartige Objekte bereitstellen.

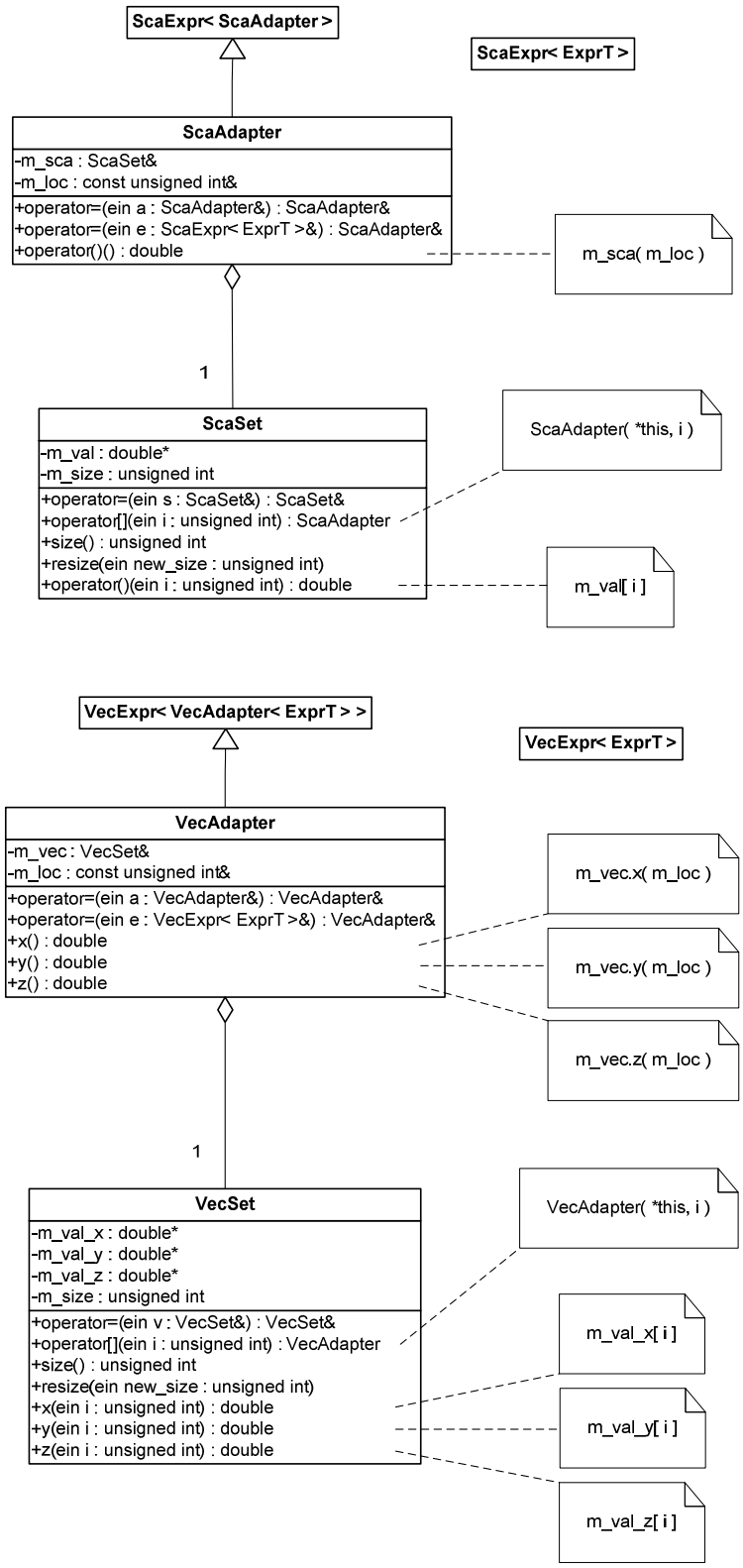


Bild 3-20: Feldweise skalare und vektorielle Daten- und Adaptertypen; Die eckigen Klammeroperatoren der Datenklassen liefern Adaptertypen zurück; Diese erlauben die Verwendung der Datenfelder zusammen mit den Klassen-Templates für die arithmetischen Ausdrücke

Das gewählte Speicherkonzept der Datenfeldklassen bringt ein Problem mit sich. Der Entwurf der Datenfeldklassen *ScaSet*, *VecSet* und *TenSet* sieht vor, die Datenfelder komponentenbezogen zu speichern. Daher ist der herkömmliche Indexzugriff auf die eigentlichen Datenobjekte nicht direkt möglich. Die Datenfeldklassen müssen die Datenobjekte vom Typ *Sca*, *Vec* bzw. *Ten* erst aus den komponentenbezogenen Rohdaten erstellen und diese dann per Instanz zurückliefern. Das macht den objektbezogenen Indexzugriff auf die Datenfelder extrem ineffizient. Trotzdem erscheint es sinnvoll zu sein, die Datenfeldklassen mit den entwickelten ET kombinieren zu wollen. Das gilt besonders vor dem Hintergrund, dass bei der FV Diskretisierung die meisten Variablen als Datenfelder vorliegen.

Mit den Adapterklassen *ScaAdapter*, *VecAdapter* bzw. *TenAdapter* ist es möglich, den vorliegenden Konflikt aufzulösen. Die drei Adapterklassen fungieren als Platzhalter für die Datenklassen *Sca*, *Vec* bzw. *Ten*. Ihre Schnittstellen entsprechen damit genau denen der zu vertretenden Datenklassen. Bild 3-20 zeigt die zugehörigen UML-Klassendiagramme für die Klassen *ScaAdapter* und *VecAdapter*. Das entsprechende Diagramm für die Klasse *TenAdapter* ist aus Platzgründen nicht dargestellt. Die Adapterklassen werden mithilfe des CRTP von den Basisklassen-Templates *ScaExpr*, *VecExpr* bzw. *TenExpr* abgeleitet. Dadurch können sie in Kombination mit den restlichen Klassen-Templates für die arithmetischen Ausdrücke verwendet werden. Im Gegensatz zu den Datenklassen besitzen die Adapterklassen keine eigenen Komponentendaten. Stattdessen enthalten sie eine Referenz auf ein Datenfeldobjekt und eine Indexvariable. Die Verknüpfung zwischen Datenfeld und Index ermöglicht den Adapterklassen wie normale Datenklassen aufzutreten. Die komponentenweisen Zugriffsfunkionen der Adapterklassen rufen beispielsweise einfach die entsprechenden Funktionen des Datenfeldobjekts auf und übergeben diesen zusätzlich die Indexvariable.

Mithilfe der entwickelten Adapterklassen kann der objektbezogene Indexzugriff in den Datenfeldklassen *ScaSet*, *VecSet* und *TenSet* nun leicht umgesetzt werden. Als Zugriffsfunktion nutzen die Datenfeldklassen den überladenen Klammeroperator *operator[]()*, vgl. Bild 3-20. Dieser gibt einfach ein passendes Adapterobjekt mit Referenzen auf die Datenfeldklasse und den Index zurück. Die Adapterobjekte enthalten nur Referenzen. Sie können vom Compiler bei der Auflösung der arithmetischen Ausdrücke vollständig wegoptimiert werden.

Anhand eines Beispiels wird abschließend demonstriert, wie einfach die entwickelten Datenfeldklassen und ET des CFD-Tools zur Formulierung algebraischer Gleichungen auf diskreten Rechengebieten eingesetzt werden können. Dazu wird Modellgleichung (3.2) zunächst so formuliert, dass sie die lineare Extrapolation des nächsten feldinneren Wertes ϕ_n an den Rechengebietsrand ϕ_{bnd} beschreibt:

$$\phi_{bnd} = \phi_n + \left.\frac{\partial \phi}{\partial x_i}\right|_n \cdot (r_{i,n} - r_{i,bnd}) \,, \tag{3.4}$$

wobei die Indices *n* bzw. *bnd* die jeweiligen Kontrollstellen im Rechengebiet markieren. Bild 3-21 zeigt, wie eine entsprechende Implementierung für Gleichung (3.4) mithilfe der

entwickelten ET im CFD-Tool aussieht. Die arithmetischen Ausdrücke werden bei der Programmerstellung automatisch komponentenweise aufgelöst. Alle Variablen stellen Datenfeldobjekte dar. Die Variable ϕ erhält den Namen *phi*. Der Gradient der Variablen ϕ wird wie zuvor als eigenständige Variable verarbeitet. Er erhält den Name *phi_grad*. Die Bezeichnung des Ortsvektors r im Quelltext bleibt erhalten.

```
phi[bnd] = phi[n] + phi_grad[n]*( r[n] - r[bnd] );
```

```
phi(bnd) = phi(n) + phi_grad.x(n)*( r.x(n) - r.x(bnd) )
                  + phi_grad.y(n)*( r.y(n) - r.y(bnd) )
                  + phi_grad.z(n)*( r.z(n) - r.z(bnd) );
```

Bild 3-21: Implementierung von Gleichung (3.4) mithilfe der entwickelten ET sowie komponentenweise Auflösung der arithmetischen Ausdrücke durch den Compiler

3.4 Parallelisierung

Die CFD-Simulation stellt hohe Anforderung sowohl an den Arbeitsspeicher als auch an den Prozessor eines Computers. Dabei schlägt die gewählte Modellkomplexität direkt auf die Rechenzeiten und den Speicherverbrauch durch. Die Verfeinerung des physikalischen Modells resultiert meist in einem komplexeren und umfangreicheren Gleichungssystem mit einer zwangsläufig höheren Anzahl benötigter Rechenoperationen. Die Verbesserung des geometrischen Modells durch Verfeinerung der Rechennetze oder Aufnahme weiterer geometrischer Komponenten bewirkt zunächst einen höheren Speicherbedarf, erfordert gleichzeitig aber auch eine größere Rechenleistung, da die Modellgleichungen für eine höhere Anzahl von Kontrollvolumen ausgewertet werden müssen. Die Gesamtperformance eines CFD-Codes ist somit sowohl vom verfügbaren Speicher als auch von der verfügbaren Rechnergeschwindigkeit abhängig. Durch Erweiterung des Rechenspeichers bzw. Erhöhung der Prozessorleistung kann diesem Sachverhalt bis zu einem bestimmten Maß entsprochen werden. Darüber hinaus ist die Steigerung der Gesamtperformance nur noch durch Parallelisierung der verwendeten Software möglich. Im Bereich des wissenschaftlichen Hoch- und Höchstleistungsrechnens ist die Verwendung hochparalleler Anwendungen Quasistandard.

Nach FLYNN [27] können parallele Computer in vier unterschiedliche Klassen eingeteilt werden. Die Klassifizierung erfolgt dabei nach der Anzahl der Steuerungseinheiten und der Verteilung des Speichers, siehe Bild 3-22. Der klassische von Neumann Rechner zum Beispiel bildet als Single-Instruction-Single-Data Maschine (SISD) das eine, die Multiple-Instruction-Multiple-Data Maschine (MIMD) mit mehreren unterschiedlichen Prozessoren und Datenströmen das andere Extrem dieser Taxonomie, siehe PACHECO [65] und RICHTER[73]. Hardwareseitig können MIMD Computersysteme zusätzlich nach ihrer Speicherarchitektur in shared memory, zu Deutsch gemeinsamer genutzter Speicher, und distributed memory, zu Deutsch verteilter Speicher, unterteilt werden. Shared memory steht allgemein für eine starke Hardwarekopplung, distributed memory für eine

eher lose Hardwarekopplung mit langsamerem Nachrichtenaustausch über das Netzwerk. Vergleiche hierzu finden sich z.B. bei DONGARR und DUNIGAN [19], sowie bei JIA und SUNDÉN [42]. Eine andere Möglichkeit MIMD Computersysteme genauer zu spezifizieren richtet sich nach dem verwendeten Programmiermodell. In der Literatur haben sich die beiden Bezeichnungen Single-Program-Multiple-Data (SPMD) und Multiple-Program-Multiple-Data (MPMD) etabliert, siehe DAREMA ET AL. [18] und KARP [44].

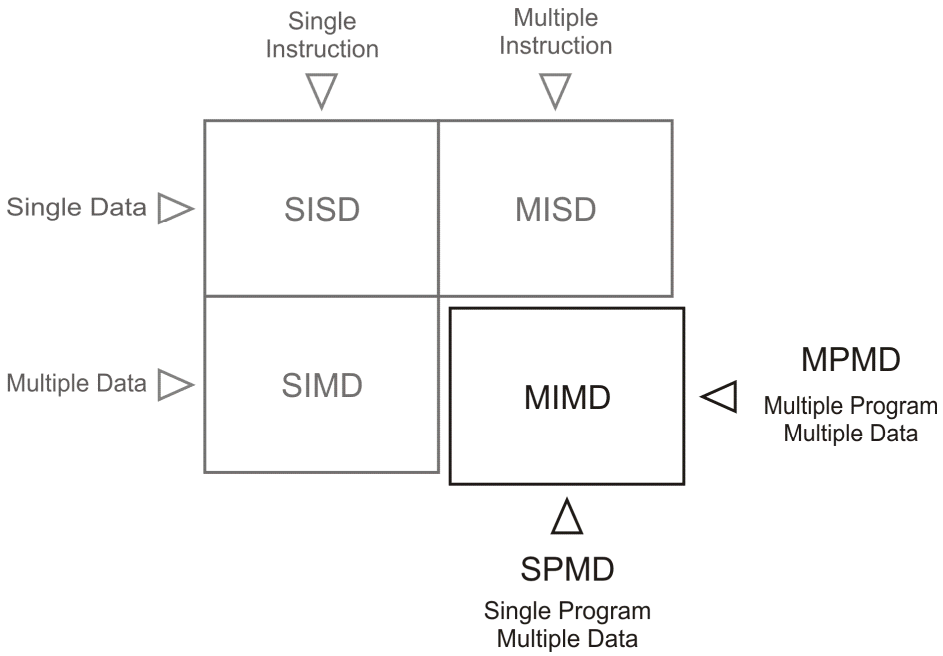


Bild 3-22: Flynns Taxonomie [27] und die Programmiermodelle Single-Program-Multiple-Data SPMD und Multiple-Program-Multiple-Data MPMD für parallele Computersysteme

Für die numerische Strömungssimulation ist SPMD von besonderem Interesse. Hier führen alle Prozesse dasselbe Programm auf jeweils verschiedenen sich nicht überlappenden Bereichen der Gesamtgeometrie aus. Die einzelnen Prozesse werden entweder lokal auf einem Computer oder verteilt auf mehreren vernetzten Computern ausgeführt. SPMD ist für shared memory wie auch für distributed memory oder einer Kombination aus beidem geeignet und damit hochflexibel. Über ein vorgegebenes Kommunikationsmodell tauschen die Prozesse an den gemeinsamen Schnittstellen zwischen den einzelnen Teilgeometrien ihre Daten aus. Falls der Kommunikationsaufwand beschränkt bleibt, skaliert diese Methode mit der Anzahl der Prozesse. Da die Prozesse zudem nur die von ihnen zu bearbeitenden Teilmenge der Gesamtdaten laden müssen, bleibt der Speicherbedarf pro Prozess beschränkt. Das US amerikanische nationale Institut für Standards und Technologie NIST beschreibt SPMD als den gegenwärtig am weitesten verbreiteten Stil der parallelen Programmierung [95].

Auch der vorgestellte Softwareentwurf dieser Arbeit verwendet SPMD als zugrunde liegendes paralleles Programmiermodell. Darüber hinaus wird jedem Prozess genau ein definierter Teilbereich der gesamten Rechengeometrie zugewiesen. Dies stellt keine Einschränkung der Programmfunktionalität dar, da bei stark unterschiedlichen Geometrieaufteilungen mehrere Prozesse auf einem Prozessorkern zusammengefasst werden

können. Die Kommunikation zwischen den einzelnen Prozessen bei SPMD wird als Message-Passing bezeichnet. Jeder Prozess besitzt einen von den restlichen Prozessen unabhängigen Speicher. Der direkte Speicherzugriff zwischen unterschiedlichen Prozessen ist nicht möglich, der Datentransfer erfolgt ausschließlich über den Nachrichtenaustausch, siehe THOLE [92]. Für Computer mit verteiltem Speicher stellt Message-Passing das optimale Programmiermodell dar.

Durch die Kapselung der Prozessdaten und die Reduktion auf den reinen Nachrichtenaustausch ist Message-Passing auch für MPMD Anwendungen geeignet. Prinzipiell können völlig verschiedene Programme miteinander gekoppelt werden, solange sie einer gemeinsamen Kommunikationsreihenfolge genügen. Hier wird aber ohne Einschränkung für die vorliegende Problemstellung ausschließlich der SPMD Fall betrachtet.

Die Parallelisierung eines Programms mit Message-Passing erfolgt explizit durch den Programmierer. Der Datenaustausch zwischen den Prozessen muss im Kontrollfluss der Software durch Kommunikations- und Steuerroutinen berücksichtigt werden. Üblicherweise geschieht dies unter Zuhilfenahme fertiger Programmbibliotheken, die die gesamte Organisation und den Aufbau des Nachrichtenaustausches sowie die Verwaltung der einzelnen Prozesse übernehmen.

Bedingt durch die Synchronisation des Programmflusses sowie den Nachrichtenaustausch fällt bei parallelen Programmen nach dem Message-Passing Paradigma ein gewisser zeitlicher Overhead an, der zunimmt je enger die einzelnen Prozesse miteinander verbunden sind, d. h. je mehr Daten zwischen den Prozessen ausgetauscht werden müssen und je stärker der Kontrollfluss eines Prozesses von dem der anderen abhängt. Der optimale Parallelisierungsgrad wird dann erreicht, wenn die Gesamtaufgabe in völlig unabhängige Teilaufgaben zerlegbar ist. Nach BADER und KRANNICH [6] kann die Güte einer Parallelisierung durch die parallele Beschleunigung S und die parallele Effizienz E bewertet werden. Beide Kenngrößen sind sowohl von der Anzahl der Prozesse P, als auch von der charakteristischen Problemgröße n abhängig.

Die parallele Beschleunigung berechnet sich als Quotient aus der Rechenzeit für einen Prozess und der Rechenzeit für P Prozesse:

$$S(n,P)=\frac{T(n,1)}{T(n,P)}. \tag{3.5}$$

Die parallele Effizienz ist als Verhältnis der parallelen Beschleunigung zur Anzahl der Prozesse definiert:

$$E(n,P)=\frac{T(n,1)}{P\cdot T(n,P)}=\frac{S(n,P)}{P}. \tag{3.6}$$

Aus den Gleichungen (3.3) und (3.4) folgt, dass bei idealem Parallelisierungsgrad $E(n, P) = 1$ der Speedup linear mit der Anzahl der Prozesse skaliert. Entsteht durch die Parallelisierung ein Overhead, gilt $E(n, P) < 1$ und der Speedup skaliert unterproportional mit der Anzahl der Prozesse. Die parallele Effizienz eines Programms wird aber nicht nur durch die Wahl des Programmflusses beeinflusst. Bei jedem Datenaustausch entsteht ein

zusätzlicher Overhead durch das Message-Passing System selbst. Somit hängt die Effizienz auch von der Geschwindigkeit der verwendeten Programmbibliothek ab. Die beiden bekanntesten Standards für Message-Passing sind die Parallel Virtual Machine (PVM) [33] und das Message-Passing Interface (MPI) [56][57]. Ein Vergleich zwischen beiden Spezifikationen würde hier zu weit führen, findet sich aber z. B. bei GROPP und LUSK [34].

Da MPI im wissenschaftlichen Hoch- und Höchstleistungsrechnen heute den de facto Standard für paralleles Programmieren mit Message-Passing definiert, wird es bei der Grobstrukturierung des vorgestellten Softwarekonzeptes ausgewählt. Die konkrete Implementierung erfolgt mit der open-source Bibliothek MPICH2 des Argonne National Laboratory, die beide Message-Passing Standards, MPI-1 und MPI-2, unterstützt. MPICH2 ist sowohl in der Programmiersprache FORTRAN als auch in den Programmiersprachen C/C++ verfügbar. Die Bibliothek ist in beiden Sprachen hoch performant. Für Windows und für verschiedene Unix Plattformen stehen zudem fertige Distributionen zum Download aus dem Internet bereit.

Ein Message-Passing-System bietet dem Programmierer die notwendige Funktionalität zur Parallelisierung seiner Software. Dabei ist es von Vorteil die Parallelisierung bereits in der frühen Phase des Softwareentwurfs zu berücksichtigen, da sie mitunter tief in die Programmstruktur eingreift. Auf den nächsten Seiten werden die grundlegenden Funktionen von MPI und die damit in Verbindung stehenden Designentscheidungen für die Parallelisierung des CFD-Tools vorgestellt.

3.4.1 Initialisieren der parallelen Umgebung und Bereitstellen des Kontextes

Steht bereits beim Start die Anzahl der Prozesse fest und ändert sich diese über die Programmlaufzeit nicht, so ist die Anwendung statisch. Im Gegensatz dazu wird eine Anwendung als dynamisch bezeichnet, wenn die Anzahl der Prozesse während der Laufzeit variieren kann. Das Aufspalten der Prozesse (forking) ist im MPI-2 Standard enthalten, wird aber noch nicht von vielen Implementierungen unterstützt. Die ausgewählte Bibliothek MPICH2 erfüllt zwar grundsätzlich diesen Standard, für das CFD-Tool entstehen aber keine Nachteile, wenn die Prozessanzahl bereits beim Start bekannt sein muss. Zur Sicherung der Portierbarkeit auf Computer die über keine MPICH2 Installation verfügen, wird deshalb ein statisches Modell verwendet.

Für die Initialisierung der Parallelen Umgebung stellt das Message-Passing-System die Initialize-Funktion zur Verfügung. Solang diese Funktion noch nicht aufgerufen wurde, kann keine andere Funktionalität des Message-Passing-Systems genutzt werden. Die Initialisierung muss genau einmal erfolgen. Der mehrfache Aufruf der Initialize-Funktion stellt einen Fehler dar. Im CFD-Tool rufen alle Prozesse die Initialize-Funktion als ersten Befehl in der Main-Routine der Hauptanwendung auf.

Alle Prozesse, die von der parallelen Umgebung gestartet worden sind, werden automatisch zu einer Gruppe, die als Welt (world) bezeichnet wird, zusammengefasst. Diese Gruppe repräsentiert dabei vereinfacht die Liste aller gestarteten Prozesse und ist lokal in jedem Prozess verfügbar. Bestimmte Problemstellungen erfordern, dass von dieser Gruppe weitere Untermengen gebildet werden. Prozesse können zu jeder Zeit ohne Referenz auf andere Prozesse neue Gruppen bilden oder bestehende Gruppen zerstören. Den kon-

kreten Anwendungsfall aus der CFD liefert die Multiphysik Simulation. Hier bilden die Gruppen Bereiche mit bestimmten physikalischen Eigenschaften. Da diese wiederum durch ein eigenes Rechenmodell abgebildet werden, ist hier innerhalb einer Gruppe zudem auch ein bestimmter Programmfluss vorgegeben.

Für die Kommunikation zwischen den Prozessen einer Gruppe werden sogenannte Intrakommunikatoren definiert. Ein Intrakommunikator ist ein Handle auf einen spezifischen Kontext, den die Prozesse innerhalb der Gruppe teilen. Grundsätzlich können Nachrichten, die innerhalb eines bestimmten Kontexts gesendet werden, nur wieder in demselben Kontext empfangen werden. Wird der Kontext als Radiofrequenz interpretiert, dann verwenden alle Kommunikatoren, die auf diesen Kontext verweisen, dieselbe Radiofrequenz. Die Motivation für den Kontext ist Modularität. Die Prozesse einer Gruppe können für ihre Kommunikation mehrere unterschiedliche Kontexte bereitstellen. Das ist insbesondere für die objektorientierte Programmierung wichtig. Die Datenkapselung stellt hier eine grundlegende Spracheigenschaft dar. Kann den Objekten einer Klasse exklusiv ein bestimmter Kontext für den Datenaustausch zugeordnet werden, dann ist die Datenkapselung bei der Kommunikation zwischen diesen Objekten automatisch gewährleistet, auch wenn in der gleichen Prozessgruppe noch andere Objekte Nachrichten senden. Die Umsetzung dieses Konzepts findet sich bei der Detaillierung der Komponentenbausteine wieder, siehe Kapitel 4.1.1. Auf makroskopischer Ebene bietet der Kontext die Sicherheit, dass unterschiedliche Programmmodule, die in dieselbe Umgebung integriert werden, nicht ungewollt untereinander interagieren, siehe Kapitel 3.2.

3.4.2 Kommunikationsformen zwischen den Prozessen

Bei der Punkt-zu-Punkt-Kommunikation sind immer genau zwei Prozesse beteiligt. Ein Prozess tritt als Sender, der andere als Empfänger auf. Aus Programmsicht heißt das, dass sowohl der Sender seinen Zielprozess als auch der Empfänger seinen Quellprozess explizit benennen muss. Grundsätzlich wird bei dieser Kommunikationsart unterschieden, ob eine Nachricht synchron oder asynchron versendet bzw. empfangen wird.

Der synchrone Nachrichtenaustausch ist vergleichbar mit einem Handschlag der beiden beteiligten Prozesse. Unter bestimmten Umständen kann das zur Blockade des Programmflusses führen. Im CFD-Tool arbeiten die einzelnen Prozesse jeweils auf nicht überlappenden Gebieten der gesamten Rechengeometrie. Über die Interfaces zwischen den einzelnen Gebieten müssen die Strömungsvariablen der angrenzenden Rechenzellen ausgetauscht werden. Bild 3-23 stellt diesen Vorgang zunächst als synchronen Datentransfer im Sequenzdiagramm dar. Würden beide Prozesse zuerst senden und danach empfangen, wäre der Programmfluss zwangsläufig blockiert. Abhilfe schafft die Berücksichtigung der Prozessnummer, die innerhalb der Prozessgruppe bekannt ist. Durch kreuzweises Vertauschen des Sende- bzw. Empfangsbefehls abhängig von der Prozessnummer wird die Blockade aufgehoben. Bilden die Seiten eines Interfaces eine Periodizität, bei der das Rechengebiet auf sich selbst verweist, kann die Blockade so allerdings nicht behoben werden. Eine Option wäre hier, wie in Bild 3-24 illustriert, zwischen internen und externen Interfaces zu unterscheiden und Nachrichten lediglich an den externen Interfaces über MPI auszutauschen. Diese Strategie wird beispielsweise im blockstrukturierten CFD-Löser NS3D verfolgt, siehe SKODA [86] und EINZINGER [22].

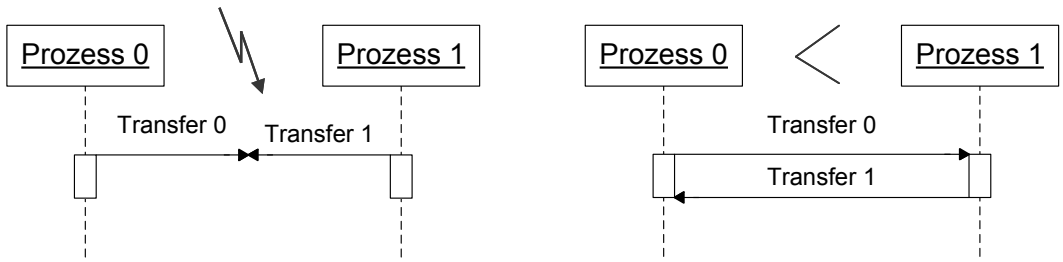


Bild 3-23: Datenfluss bei synchroner Punkt-zu-Punkt-Kommunikation; Links: Blockade der beteiligten Prozesse durch gleichzeitigen synchronen Datenversand; Rechts: Lösen der Blockade durch Platzieren des Sende- und Empfangsaufrufs in Abhängigkeit von der Prozessnummer

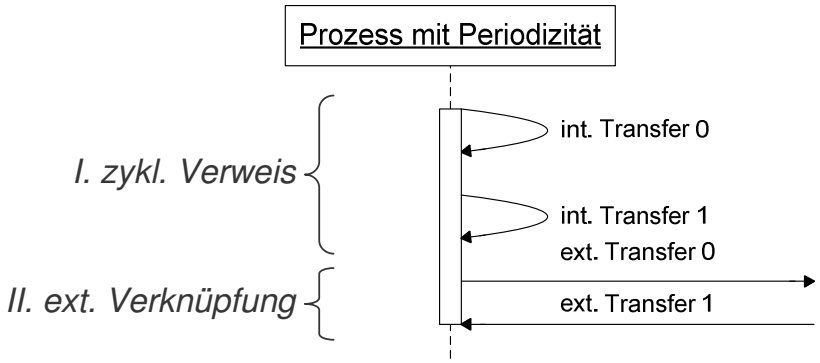


Bild 3-24: Datenfluss bei synchroner Punkt-zu-Punkt Kommunikation; Umgehen der Blockade bei Selbstverweisen durch Unterscheidung von internem und externem Datentransfer

Eine mit Hinblick auf die Objektorientierung konsistentere Lösung bietet der asynchrone Nachrichtenaustausch über MPI. Er ist deshalb im CFD-Tool generell für den Datentransfer an allen Rechengebietsgrenzen vorgesehen. Bei der asynchronen Kommunikation müssen die Prozesse ihren Programmfluss zunächst nicht aufeinander abstimmen. Der Sende- und Empfangsbefehl wird ohne Rücksicht auf den Partnerprozess abgesetzt. Jeder Prozess triggert einfach alle in ihm enthaltenen Interfaces ab. Die bei dieser Kommunikationsart notwendige Pufferung der Daten erfolgt durch das Message-Passing-System und ist nicht Aufgabe der parallelen Anwendung selbst. Allerdings muss der vollständige Abschluss des Nachrichtenaustausches explizit geprüft werden, bevor auf die ausgetauschten Daten zugegriffen wird. Geschieht das nicht, läuft der CFD-Code Gefahr mit korrumpierten Daten zu rechnen. Zur Überprüfung, ob ein asynchroner Kommunikationsvorgang abgeschlossen ist, stellt das Message-Passing-System sowohl eine nicht blockierende Test- als auch eine blockierende Warteroutine bereit. Für die Beendigung des Austauschvorganges an den Interfaces genügt die blockierende Wartefunktion, denn es ist nicht von Bedeutung, an welchem Interface der Vorgang als erstes abgeschlossen ist, sondern nur, dass er vollständig an allen Interfaces eines Prozesses abgeschlossen ist. Bild 3-25 veranschaulicht den Ablauf beim asynchronen Datentransfer an den Interfaces des CFD-Tools.

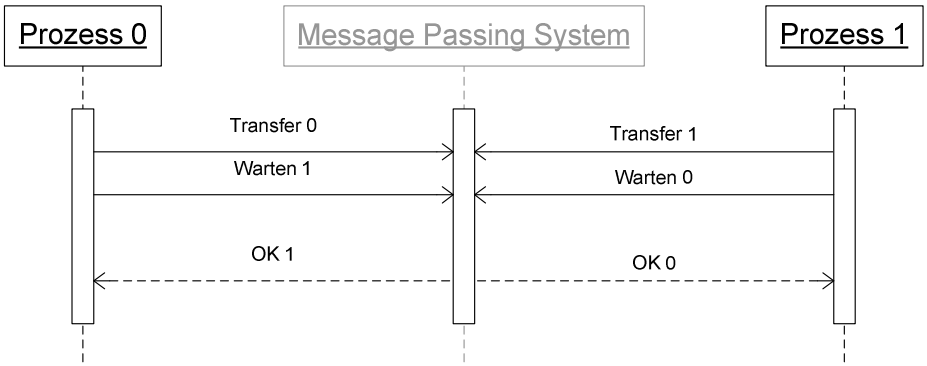


Bild 3-25: Datenfluss bei asynchroner Punkt-zu-Punkt-Kommunikation; Keine Blockade möglich, aber Test des erfolgreichen Datentransfers nötig

Im Gegensatz zur Punkt-zu-Punkt-Kommunikation, sind bei der kollektiven Kommunikation immer alle Prozesse einer bestimmten Gruppe beteiligt. Um eine kollektive Kommunikation mit einer Untermenge von Prozessen in einer Gruppe ausführen zu können, muss grundsätzlich eine neue Gruppe, die diese Untermenge beschreibt, gebildet werden, denn an der kollektiven Kommunikation müssen immer alle Prozesse einer Gruppe teilnehmen. Für einige kollektive Funktionen existieren aber Platzhalterargumente, die bei der Datenübergabe verwendet werden können, um den aufrufenden Prozess von der Operation auszuschließen. Die einfachste Form der kollektiven Kommunikation stellt die Barriere dar. Die Barrier-Funktion blockiert den aufrufenden Prozess solange, bis alle anderen Prozesse der Gruppe die Funktion aufgerufen haben. Um die Synchronisation der einzelnen Prozesse innerhalb der Module des CFD-Tools zusätzlich abzusichern, wird die barrier-Funktion nach allen größeren Datenbearbeitungsblöcken, wie zum Beispiel der Iteration eines Gleichungsobjektes, siehe Kapitel 4.2.1, aufgerufen. Zum reinen Um- bzw. Neuverteilen von Daten auf die Prozesse einer Gruppe dienen die Broadcast-, Scatter- und Gather-Funktionen. All diese Operationen haben gemeinsam, dass die betroffenen Daten bei deren Übermittlung nicht verändert werden. Die globalen Reduce-Funktionen hingegen dienen dazu, die übergebenen Daten bei deren Transfer mithilfe des angegebenen Operators zu verändern. Als Operatoren stehen unter Anderem die Summe, das Produkt und der Minimum- bzw. Maximum-Operator zu Verfügung. Das Residuum ist eine wichtige Kenngröße in jeder impliziten CFD-Simulation. Es wird als globaler Indikator für die Konvergenz der Rechnung herangezogen. Da das gesamte Gleichungssystem einer Prozessgruppe im CFD-Tool durch die Parallelisierung auf den Speicher mehrerer Prozesse aufgeteilt ist, muss das Residuum zunächst über alle Prozesse der Gruppe aufsummiert und das Ergebnis der Operation dann wieder auf die Prozesse verteilt werden. Dies geschieht während der Iteration der Gleichungsobjekte durch die Allreduce-Funktion in Verbindung mit dem Summen-Operator.

3.4.3 Ein- und Ausgabe von Daten

Die Ein- und Ausgabeströme einer Konsolenanwendung stellen die externen Schnittstellen dar, über die während der Laufzeit Informationen ausgetauscht werden können. Beim CFD-Tool sind dies erstens die Ausgabeströme *cout* und *cerr* – der Eingabestrom *cin* und der Ausgabestrom *clog* werden nicht verwendet – und zweitens die Ein- und Ausgabe-

ströme zum Lesen und Schreiben von Dateien. Berücksichtigt man, dass das CFD-Tool eine SPMD Anwendung ist, ergibt sich die Frage nach der Organisation der Dateneinbzw. -ausgabe bezogen auf die Parallelisierung.

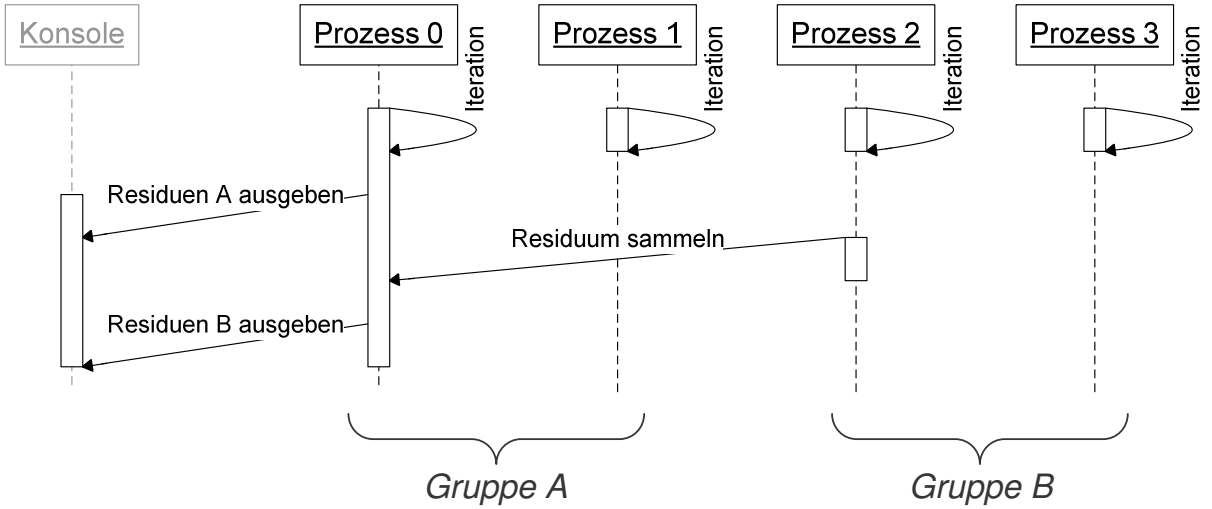


Bild 3-26: Ablauf der Residuenausgabe in der Konsole; Der Masterprozess der Weltgruppe, hier Prozess 0 aus Gruppe A, sammelt die Residuen von den Masterprozessen aller übrigen Gruppen ein, hier Prozess 2 aus Gruppe B, und gibt diese in der Konsole synchronisiert aus

Als Erstes wird diese Frage für die Konsolenausgabeströme geklärt. Das Message-Passing-System generiert für die parallele Anwendung automatisch eine einheitliche Umgebung. Das bedeutet, dass die Konsolenausgabeströme aller Prozesse mit der aufrufenden Konsole verknüpft werden. Wollen nun mehrere Prozesse ohne zusätzliche Synchronisation gleichzeitig Daten ausgeben, kommt es im Allgemeinen zu einem Durcheinander und die ausgegebenen Daten werden unlesbar. Im CFD-Tool beschränkt sich die Ausgabe von Statusinformationen mit dem Ausgabestrom *cout* im Wesentlichen auf die aktuellen Residuen der Gleichungsobjekte und die damit in Verbindung stehenden Informationen. Dabei schreiben alle Gleichungsobjekte eines Prozesses ihren aktuellen Iterationsstatus ungeachtet der Prozesszugehörigkeit in einen lokalen Ausgabepuffer. Dieser Ausgabepuffer wird nach jedem Iterationsdurchgang der Gleichungen von der Iterationsschleife des Masterprozesses aller lokalen Gruppen an den Master der Weltgruppe gesandt und von eben diesem, wie in Bild 3-26 dargestellt, synchronisiert ausgegeben. Bei den übrigen Prozessen läuft die Residuenausgabe ins Leere. Für die Ausgabe von Fehlerbenachrichtigungen durch den Ausgabestrom *cerr* muss anders verfahren werden. Da jeder Fehler unweigerlich zum Abbruch der parallelen Anwendung führen muss, ist beim Auftreten eines Fehlers keine Synchronisation zwischen den Prozessen mehr möglich. Dies wird weiter unten im Abschnitt über die Fehlerbehandlung begründet. Für die Fehlerausgabe ist das an dieser Stelle allerdings nebensächlich. Die Konsequenz heißt lediglich, dass auch die Fehlerausgabe nicht synchronisiert ablaufen kann. Die Information über das Auftreten des Fehlers wird unabhängig davon ausgegeben. Im ungünstigsten Fall steht sie allerdings im Konflikt mit einer anderen Ausgabe und ist nur schwer lesbar, siehe Bild 3-27.

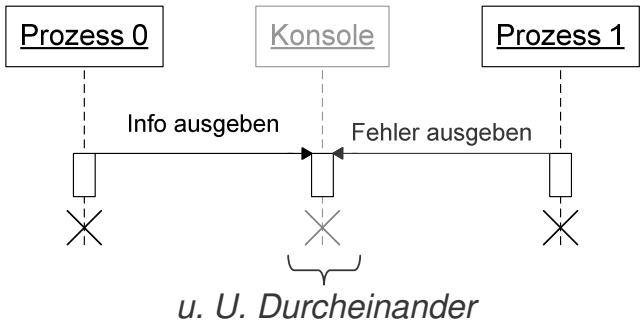


Bild 3-27: Ablauf der Fehlerausgabe in der Konsole; Eine Synchronisation ist hier nicht möglich; Im ungünstigsten Fall kollidiert die Fehlerausgabe mit einer anderen Ausgabe und ist nur schwer lesbar

Nun bleibt die Prozessorganisation beim Lesen und Schreiben von Dateien zu lösen. Der MPI-2 Standard [57] definiert Funktionen, die das parallele Einlesen bzw. Speichern von Daten aus bzw. ins Dateisystem regeln. Die Routinen setzen voraus, dass ein zusammenhängender Datensatz, in der Regel ein Array eines bestimmten Datentyps, einfach auf die betreffenden Prozesse aufgeteilt ist. Soll dieser Datensatz in einem File, gespeichert werden, wird jedem Prozess ein definierter Bereich innerhalb des Files zugewiesen, in den er sein Datenfragment schreibt. Das Gleiche gilt umgekehrt für den Fall, dass der Datensatz eingelesen wird. Die direkte Verwendung dieser MPI-Funktionen kann allerdings beim Lesen und Speichern der komplexen CFD-Daten sehr aufwändig sein. Darüber hinaus ist zu beachten, dass durch die Funktionen die modulare Nutzung bestehender Softwarebibliotheken zur High-End Datenverarbeitung erschwert wird. Als Beispiel seien hier nur die Formate XML [104], Tecplot, CGNS [3] und HDF5 [91] genannt, für die solche Bibliotheken frei verfügbar im Internet existieren.

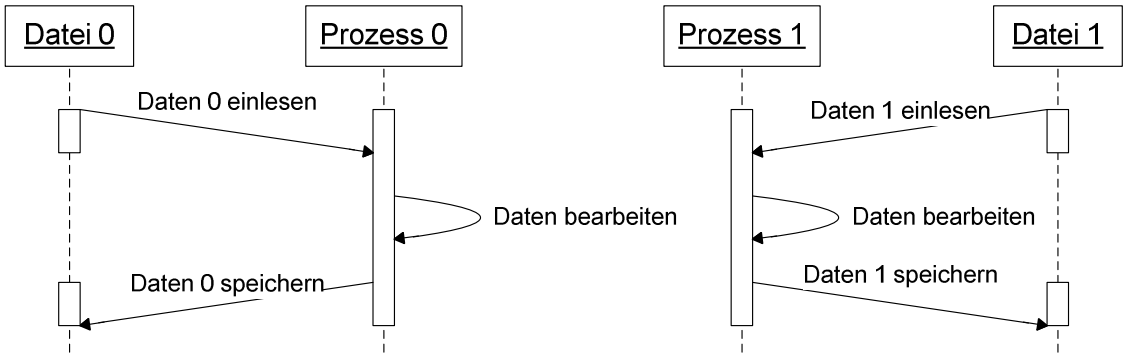


Bild 3-28: Modell für die verteilte, voll symmetrische Datenein- bzw. Datenausgabe

Für das CFD-Tool sind zwei Modelle entwickelt worden, die die parallelisierte Datenverarbeitung beim Lesen und Schreiben der Dateien abbilden, ohne die MPI-Ein- und Ausgabefunktionen zu verwenden. Das erste der beiden Modelle repräsentiert die sogenannte voll symmetrische Konfiguration. Hier liest bzw. schreibt jeder Prozess seine eigene Datei. Das zugehörige Sequenzdiagramm ist in Bild 3-28 visualisiert. Jeder Prozess agiert völlig unabhängig von den anderen. Während des Lesens bzw. Schreibens der Daten ist keinerlei Synchronisation notwendig. Deshalb ist die symmetrische Konfigu-

ration durch einen sehr hohen Parallelisierungsgrad geprägt. Eine Konsequenz und mitunter ein Nachteil des Modells ist allerdings, dass die Daten völlig verteilt in unabhängigen Dateien abgelegt werden. Soll das vermieden werden oder ist das unmöglich, so bietet das zweite Modell, siehe Bild 3-29, eine Lösung. Das gezeigte Sequenzdiagramm stellt die asymmetrische Konfiguration dar. Im Gegensatz zur vorherigen Lösung agieren die einzelnen Prozesse nun in gegenseitiger Abhängigkeit. Das Lesen und Schreiben aller Daten wird vom Masterprozess übernommen. Durch den sequentiellen Ablauf müssen aber alle Prozesse synchronisiert werden. Die asymmetrische Konfiguration ist deshalb im Vergleich zur symmetrischen durch einen geringeren Parallelisierungsgrad gekennzeichnet. Vorteilhaft ist hier aber die zentrale Datenhaltung.

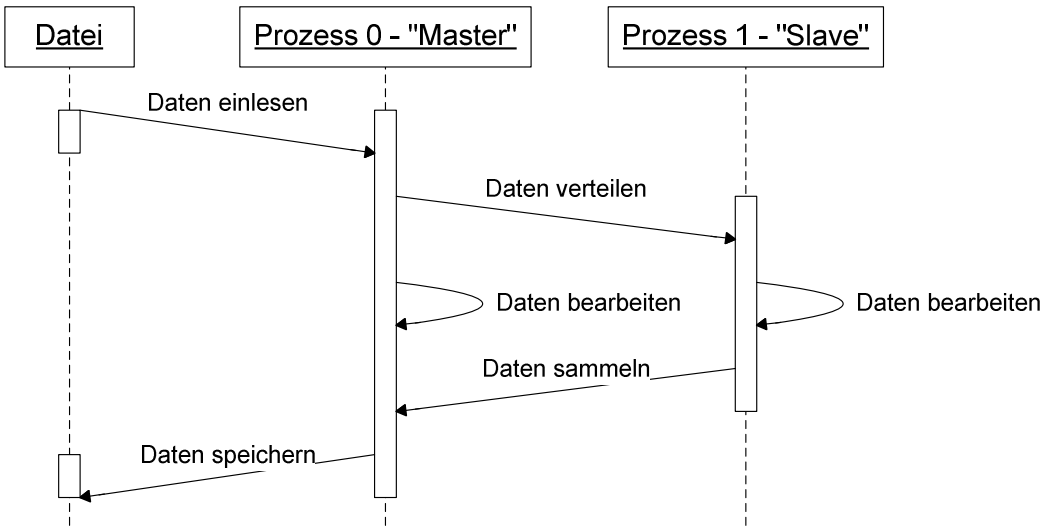


Bild 3-29: Modell für die zentralisierte, asymmetrische Datenein- bzw. Datenausgabe

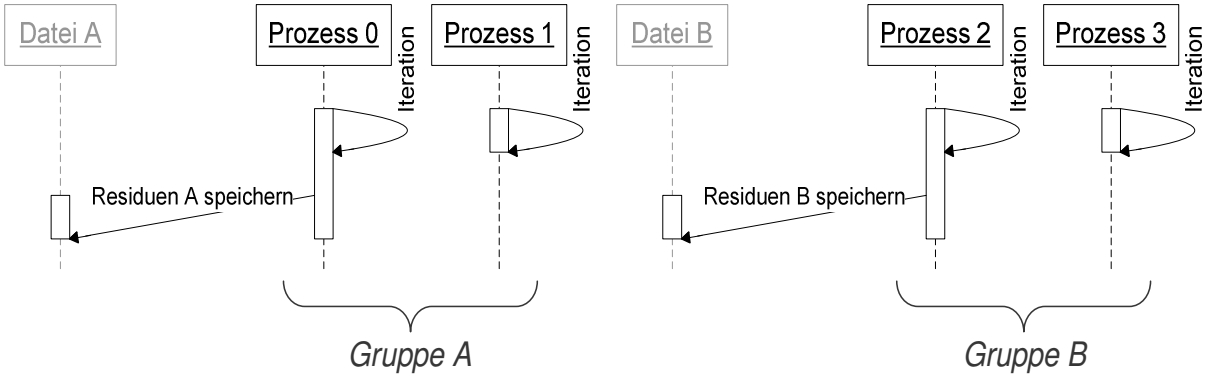


Bild 3-30: Residuenaufzeichnung; Die Masterprozesse einer jeden Gruppe, hier Prozess 0 aus Gruppe A und Prozess 2 aus Gruppe B, speichern die Residuen für ihre Gruppe jeweils in einer eigenen Datei

Durch ihre völlig unterschiedlichen Lösungsansätze können die beiden vorgestellten parallelen Ein- bzw. Ausgabemodelle jeweils alleine nicht jede Problemstellung abdecken. Vielmehr muss auf den Einzelfall bezogen der jeweils geeignetste Lösungsansatz ausgewählt werden. Dabei kann die spezielle Lösung natürlich etwas von den vorgestellten idealisierten Abläufen der beiden Modelle abweichen. Im CFD-Tool erfolgt beispielsweise

die Residuenaufzeichnung – nicht zu verwechseln mit der bereits diskutierten Residuenausgabe – im Grunde völlig symmetrisch. Diese Konfiguration ist hier am sinnvollsten, weil die Residuen nach jeder Iteration der Gleichungsobjekte fortlaufend abgespeichert werden sollen. Alle Residuen sind nun aber innerhalb ihrer jeweiligen Prozessgruppe gleich, sodass nur die Gruppenmaster an der Aufzeichnung beteiligt werden müssen. Das Modell zur Residuenaufzeichnung ist in Bild 3-30 dargestellt. Obwohl hier nicht alle Prozesse völlig gleichmäßig am Speichern der Daten beteiligt sind, ist das symmetrische Grundprinzip trotzdem klar zu erkennen.

3.4.4 Fehlerbehandlung

Der in der Programmiersprache C++ bekannte Mechanismus der Ausnahmebehandlung (exception handling) stellt ein mächtiges Werkzeug dar. Er wurde speziell entwickelt, um die Fehlerbehandlung in Programmen, die aus unabhängig entwickelten Komponenten bestehen, zu unterstützen, siehe STROUSTRUP [90]. Das Konzept trennt das Auftreten eines Fehlers von dessen Behandlung. Damit ist es möglich, einen Fehler, der in einem Programmmodul auftritt und dort nicht behandelt werden kann, ins Hauptprogramm weiterzureichen und erst dort zu behandeln. Trifft ein Programmmodul konkret auf ein Problem, das es nicht lösen kann, wirft es einfach eine Ausnahme (throw) und hofft darauf, dass die übergeordnete Ebene dieses Problem behandelt (catch). Bei der Parallelisierung mit MPI ist nun zu beachten, dass beim Werfen der Ausnahme die Abarbeitung des unmittelbar folgenden Befehlscodes unterbrochen und stattdessen an die Stelle zur Ausnahmebehandlung gesprungen wird. Da ein Fehler in einer SPMD Anwendung nicht zwangsläufig in allen Prozessen gleichzeitig auftritt, wird der koordinierte Programmablauf zwischen den Prozessen gestört. Dies kann zur Blockade der Anwendung führen, falls im Befehlscode nach der Stelle, an der der Fehler auftritt, ein Synchronisationsvorgang zwischen den Prozessen stattfindet.

Der MPI Standard besagt für die Fehlerbehandlung allgemein, dass in einer SPMD Anwendung jeder Fehler zum Abbruch aller Prozesse führen muss. Hierfür wird die Abort-Funktion bereitgestellt, die das Message-Passing-System veranlasst, die gesamte parallele Anwendung sofort abzubrechen, falls sie irgendwo innerhalb eines Prozesses aufgerufen wird. Bezogen auf den Ausnahmebehandlungsmechanismus bedeutet das, dass jede Ausnahme zum Aufruf der Abort-Funktion führt. Das muss bei der Entwicklung aller Module des CFD-Tools berücksichtigt werden.

3.4.5 Beenden der parallelen Umgebung

Zum normalen Beenden einer mit MPI parallelisierten Anwendung steht die Finalize-Funktion zur Verfügung. Die Funktion räumt alle bestehenden MPI Datenstrukturen auf und muss von jedem Prozess aufgerufen werden. Kommunikationsvorgängen, die noch aktiv sind, werden allerdings nicht abgebrochen, dies muss durch den Programmablauf selbst gewährleistet sein. Nach Beendigung der parallelen Umgebung kann kein MPI-Befehl mehr ausgeführt werden. Selbst eine erneute Initialisierung durch die Initialize-Funktion ist nicht möglich. Im CFD-Tool rufen alle Prozesse die Finalize-Funktion als letzten Befehl in der Main-Routine der Hauptanwendung auf.

4 Detaillierung der Komponentenbausteine

Kapitel 4 detailliert das zuvor entworfene Grobdesign der Softwarekomponenten. Es kombiniert die entworfenen Klassenbausteine und präsentiert die programmiertechnisch umgesetzte Repräsentation von Geometrie und Gleichungen. Zur Berechnung der notwendigen geometrischen Größen sowie zur Diskretisierung der Gleichungsterme werden sogenannte Schemataklassen eingeführt. Darüber hinaus beschäftigt sich das Kapitel mit der Lösung der bei der Diskretisierung entstehenden LGS. Den Abschluss des Kapitels bildet die Vorstellung der Algorithmenklassen, die die korrekte Ablaufsteuerung des CFD-Tools während der numerischen Simulation sicherstellen.

4.1 Repräsentation der numerischen Netze

4.1.1 PolyMesh- und PolySlice-Klassen

Die Klasse *PolyMesh* bildet den zentralen Speicherort für die Daten der Zell-, Flächen-, und Knotenfelder sowie für die Datenstrukturen zur Speicherung des Aufbaus des numerischen Netzes. Die Daten sind allesamt durch Objektkomposition in die Klasse *PolyMesh* integriert und können jeweils über geeignete Zugriffsfunktionen abgerufen bzw. gesetzt werden.

Überdies beinhaltet die Klasse *PolyMesh* eine Liste von *PolySlice*-Objekten. Diese Objekte repräsentieren die einzelnen Bereiche des numerischen Netzes, also das Innere, die Interfaces und die Ränder. *PolySlice*-Typen nutzen die in Kapitel 3.1 definierte Klasse *Extent* zur Speicherung der Zellen- bzw. Flächenausdehnung für die Diskretisierung. Die Klassen *InternalPolySlice*, *PolyInterfaceSlice* und *PolyBoundarySlice* sind alle von der Basisklasse *PolySlice* abgeleitet und können somit in die Liste von *PolySlice*-Objekten aufgenommen werden. Zur Spezifizierung neuer oder zum Löschen alter Bereiche besitzt das *PolyMesh* die Member-Funktionen *add_slice()*, *remove_slice()* und *remove_all_slices()*. Mit der Funktion *get_all_slices()* kann die gesamte Liste von Netzbereichen ausgegeben werden. *PolyMesh*- und *PolySlice*-Klassen sind Zeigerklassen, die Eigenschaften zur Referenzzählung erben sie von der gemeinsamen Basisklasse *Base*.

Neben den bereits aufgezählten Datenstrukturen beinhaltet die Klasse *PolyMesh* außerdem noch MPI-Kommunikatoren, die für den netzbezogenen Datenaustausch zwischen den verschiedenen Prozessen benötigt werden. Die Modelle zum Datenaustausch sind in Kapitel 3.4 beschrieben. Auf die Kommunikatoren kann über entsprechende get- bzw. set-Funktionen zugegriffen werden.

Für die Berechnung rein geometrischer Größen, wie dem Zellvolumen oder dem Zellflächeninhalt, stellt die Klasse *PolyMesh* die Funktion *apply()* zur Verfügung. Diese erwartet Hilfsobjekte vom Typ *GeoScheme*. Die Objekte werden innerhalb der *apply()*-Funktion an die *PolySlice*s durchgereicht. Diese wählen dann schließlich die für ihren Bereichstyp geeignete Berechnungsfunktionalität der *GeoScheme*-Objekte aus. Die *GeoScheme*-Objekte werden noch genauer unter Punkt 4.1.3 vorgestellt.

Zum Laden und Speichern der Daten besitzt die Klasse *PolyMesh* die Member-Funktionen *load()* und *save()*. Für die Aktualisierung der einzelnen Daten, beispielsweise nach einer geometrischen Transformation ist die Funktion *update()* vorgesehen.

Der statische Klassenentwurf für das *PolyMesh* und die verschiedenen *PolySlice*-Typen ist in Bild 4-1 zu sehen. Auf die explizite Darstellung der zu Anfang erwähnten Datenfeld- und Layoutobjekte wird der Übersichtlichkeit halber verzichtet. Die Entwürfe für diese Objekttypen finden sich in Kapitel 3.

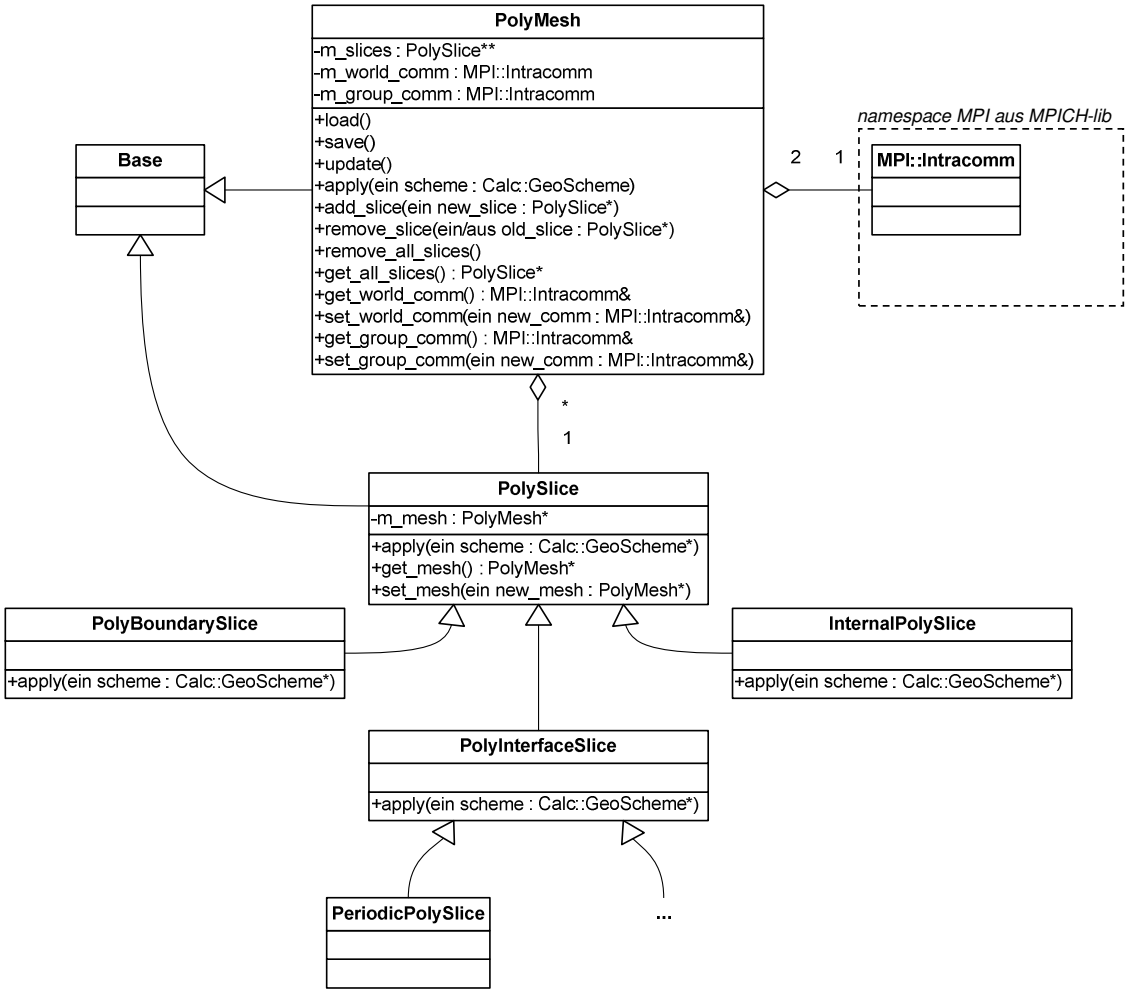


Bild 4-1: Die *PolyMesh*-Klasse als zentrale Datenstruktur, die die *PolySlice*-Klassen für den inneren Bereich, die Interfaces und die Ränder des numerischen Netzes sowie die MPI-Kommunikatoren für die Parallelisierung bereitstellt

4.1.2 Geometrievariablen

Bei der Diskretisierung der Erhaltungsgleichungen werden verschiedene Variablen benötigt, die die geometrischen Verhältnisse im numerischen Netz beschreiben. Die Variablen werden von der Klassen *PolyMesh* nach dem Laden des numerischen Netzes automatisch berechnet und können über die jeweiligen Zugriffsfunktionen der Klasse bequem direkt abgerufen werden. Tabelle 4-1 listet die wichtigsten Geometrievariablen sortiert als Knoten-, Flächen-, und Zellfelder, sowie die entsprechenden Zugriffs-

funktionen der Klasse *PolyMesh* auf. Die einzelnen Variablen werden mithilfe der Klassen *ScaSet*, *VecSet* und *TenSet*, die in Kapitel 3.3 entworfen wurden, abgebildet. Neben den gängigen Variablen zur Beschreibung der geometrischen Verhältnisse beinhaltet die Liste auch den Zellflächeninterpolationsfaktor λ zur Interpolation zweier Zellwerte an die dazwischen liegende Fläche und den Tensor G_{ij} zur Gradientenberechnung mithilfe des Least-Squares-Ansatz.

Tabelle 4-1: Liste der wichtigsten Variablen zur Beschreibung der geometrischen Eigenschaften des numerischen Netzes im CFD-Tool

Variable	Beschreibung	Speicherort im Netz	Variablentyp in *PolyMesh*	Zugriffsfunktion im *PolyMesh*
$x_{i,n}$	Knotenkoordinate	Knotenfeld	VecSet	get_center_N()
$x_{i,f}$	Zellflächenmittelpunkt	Flächenfeld	VecSet	get_center_F()
λ	Zellflächeninterpolationsfaktor		ScaSet	get_lambda_F()
n_i	Zellflächennormale		VecSet	get_normal_F()
S	Zellflächeninhalt		ScaSet	get_area_F()
$x_{i,c}$	Zellmittelpunkt	Zellenfeld	VecSet	get_center_C()
Ω	Zellvolumen		ScaSet	get_volume_C()
G_{ij}	Tensor für die Least-Squares-Gradientenberechnung		TenSet	get_gij_C()

Die relative Lage der Knotenkoordinaten $x_{i,n}$, der Flächenmittelpunkte $x_{i,f}$ und der Zellenmittelpunkte $x_{i,c}$ bezogen auf ein KV ist in Bild 4-2 zu sehen. Die Knotenvariablen stellen einen integralen Bestandteil des numerischen Netzes dar. Sie werden zusammen mit dem Zellen- und dem Flächen-Layout beim Einlesen des Netzes initialisiert.

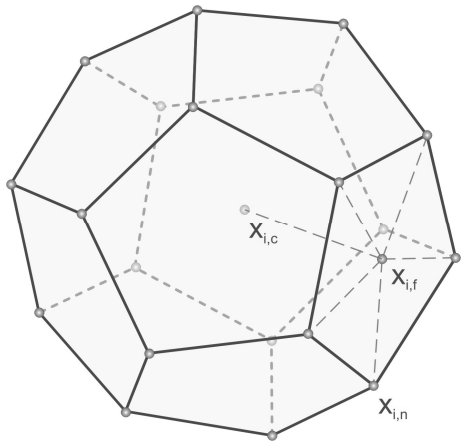


Bild 4-2: Knotenkoordinaten $x_{i,n}$, Flächenmittelpunkte $x_{i,f}$ und Zellenmittelpunkte $x_{i,c}$ eines KV

4.1.3 Berechnungsschemata

Dieser Abschnitt stellt den Entwurf der Klassen für die Berechnungsschemata der Netzvariablen des CFD-Tools vor. Die Schemata übernehmen die korrekte Initialisierung der Variablen aus Tabelle 4-1 und werden automatisch nach dem Einlesen des Netzes ausgeführt. Für die geometrischen Berechnungen müssen die drei unterschiedlichen Bereiche Gebietsinneres, Interface und Gebietsrand des Netzes berücksichtigt werden. Diesem Sachverhalt trägt die Basisklassen *GeoScheme* Rechnung. Sie definiert über die drei Member-Funktionen *on_internal()*, *on_interface()* und *on_boundary()* die benötigten Schnittstellen. Der zugehörige statische Klassenentwurf ist in Bild 4-3 zu sehen. Die Klassen für die konkreten Berechnungsschemata werden von dieser Basisklasse abgeleitet. Es ist möglich, für eine bestimmte Geometrievariable unterschiedliche Berechnungsphilosophien anzuwenden. Dafür müssen die Berechnungsschemata flexibel erstellt und gespeichert werden können. Die Basisklasse *GeoScheme* ist deshalb als Zeigerklasse konzipiert. Die nötigen Eigenschaften für die Referenzzählung leitet sie von der Klasse *Base* ab. Um die Berechnungsschemata von den Diskretisierungsschemata, siehe Kapitel 4.2.2, abzugrenzen sind sie im Namensraum *Calc::* angesiedelt.

Die Geometrievariablen müssen in einer bestimmten Reihenfolge berechnet werden. Dies ist nötig, da die Variablen teils voneinander abhängen. Die nachfolgenden Berechnungsvorschriften sind in der Reihenfolge aufgeführt, die auch die Klasse PolyMesh bei der Variablenberechnung verwendet. Die zugehörigen Klassenentwürfe der Berechnungsschemata für die einzelnen Geometrievariablen sind aus Platzgründen nicht explizit dargestellt. Sie folgen direkt aus dem Entwurf der Basisklasse *GeoScheme*.

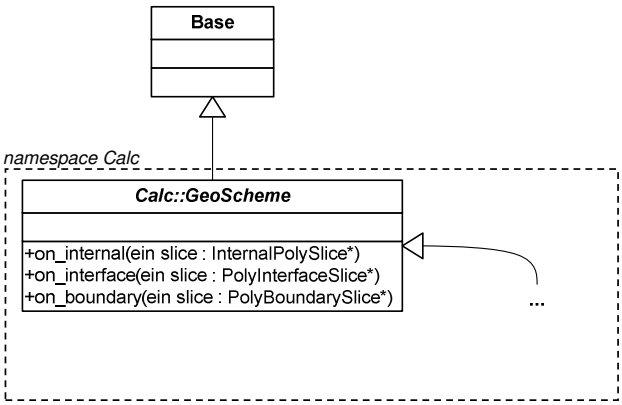


Bild 4-3: Basisklasse zur Implementierung der Berechnunsschemata für die Geometrievariablen des numerischen Netzes; Die Berechnung der Variablen gliedert sich in die unterschiedlichen Bereiche des Netzes auf

Flächenbezogene Größen

Als erstes müssen die Zellflächenmittelpunkte $x_{i,f}$ berechnet werden. Dafür sind die Knotenkoordinaten $x_{i,n}$ notwendig. Die Werte werden einfach über alle KV-Flächen des Gebietsinneren, der Interfaces und der Gebietsränder bestimmt. Im CFD-Tool sind zwei unterschiedliche Vorschriften implementiert. Bei der kantengewichteten Berechnungs-

philosophie nach WUNDERER [106] werden die Zellflächenmittelpunkte über Gewichtungsfaktoren l_n bestimmt:

$$l_n = \left| x_{i,n_{next}} - x_{i,n} \right| + \left| x_{i,n} - x_{i,n_{last}} \right|, \tag{4.1}$$

$$x_{i,f} = \frac{\sum_{n_f} l_n \cdot x_{i,n}}{\sum_{n_f} l_n}. \tag{4.2}$$

Die geometrischen Zellflächenmittelpunkte dagegen werden einfach als Mittelwerte der Kontenkoordinaten der einzelnen Zellflächen gebildet:

$$x_{i,f} = \frac{\sum_{n_f} x_{i,n}}{\sum_{n_f} 1}. \tag{4.3}$$

Gleichung (4.3) stellt das standardmäßige Berechnungsschema im CFD-Tool dar.

Als nächstes ist die Berechnung der Zellflächeninhalte S_f an der Reihe. Sie erfordert wieder die Knotenkoordinaten $x_{i,n}$ sowie die gerade bestimmten Flächenmittelpunkte $x_{i,f}$. Die Werte werden direkt über alle KV-Flächen des Gebietsinneren, der Interfaces und der Gebietsränder bestimmt. Der Zellflächeninhalt setzt sich dabei aus den Teilflächeninhalten der durch den Flächenmittelpunkt $x_{i,f}$ und jeweils zwei Knotenkoordinaten aufgespannten Dreiecke zusammen:

$$S_f = \frac{1}{2} \left| \sum_{n_f} \left(x_{i,n} - x_{i,f} \right) \times \left(x_{i,n_{next}} - x_{i,f} \right) \right|. \tag{4.4}$$

Die Bestimmung der Zellflächennormalenvektoren $n_{i,f}$ erfolgt unter Zuhilfenahme der Knotenkoordinaten $x_{i,n}$ sowie der Flächenmittelpunkte $x_{i,f}$. Zur Normierung der Zellflächennormalenvektoren werden außerdem die bereits berechneten Zellflächeninhalte S_f herangezogen. Dies ist nicht zwingend notwendig, spart aber Rechenzeit. Die Werte werden unabhängig voneinander über alle KV-Flächen des Gebietsinneren, der Interfaces und der Gebietsränder bestimmt. Die Berechnungsvorschrift für die Zellflächennormalenvektoren lautet:

$$n_{i,f} = \frac{1}{2S_f} \sum_{n_f} \left(x_{i,n} - x_{i,f} \right) \times \left(x_{i,n_{next}} - x_{i,f} \right). \tag{4.5}$$

Zellbezogene Größen

Für die Initialisierung der Zellmittelpunkte $x_{i,c}$ sind die Flächenmittelpunkte $x_{i,f}$ notwendig. Die Werte werden zunächst für alle KV des Gebietsinneren berechnet. Die Zellmittelpunkte an den Interfaces werden danach ausgetauscht. Die Zellmittelpunkte an den entarteten KV der Gebietsränder korrespondieren mit den jeweiligen Flächenmittelpunkten $x_{i,f}$ und werden von diesen übernommen. Wie bei den Flächenmittelpunkten gibt es auch für die Zellmittelpunkte im CFD-Tool zwei Berechnungsschemata. Das flächengewichtete Schema lautet:

$$x_{i,c} = \frac{\sum_{f_c} S_f \cdot x_{i,f}}{\sum_{f_c} S_f}. \tag{4.6}$$

In Gleichung (4.6) treten die bereits berechneten Flächeninhalte S_f, siehe Gleichung (4.4), auf. Für die geometrische Bestimmung der Zellmittelpunkte wird einfach der Mittelwert der zugehörigen Flächenmittelpunkte $x_{i,f}$ gebildet:

$$x_{i,c} = \frac{\sum_{f_c} x_{i,f}}{\sum_{f_c} 1}. \tag{4.7}$$

Gleichung (4.7) stellt das standardmäßige Berechnungsschema im CFD-Tool dar.

Für die Berechnung der Zellvolumen Ω_c werden sowohl die Knotenkoordinaten $x_{i,n}$, die Flächenmittelpunkte $x_{i,f}$ als auch die Zellmittelpunkte $x_{i,c}$ benötigt. Die Werte werden zunächst für alle KV des Gebietsinneren gebildet. Die Werte an den Interfaces werden danach ausgetauscht, die Werte an den entarteten KV der Gebietsränder zu Null gesetzt. Jedes Zellvolumen setzt sich aus den Teilvolumina der durch den Flächenmittelpunkt, den Zellmittelpunkt und jeweils zwei Zellflächenknoten aufgespannten Tetraeder zusammen:

$$\Omega_c = \frac{1}{6} \sum_{f_c} \sum_{n_f} \left(x_{i,f} - x_{i,c} \right) \cdot \left[\left(x_{i,n} - x_{i,f} \right) \times \left(x_{i,n_{next}} - x_{i,f} \right) \right]. \tag{4.8}$$

Geometrische Hilfsgrößen

Für die Berechnung der Zellflächeninterpolationsfaktoren λ_f werden sowohl die Flächenmittelpunkte $x_{i,f}$ als auch die Mittelpunkte der Master-Zellen $x_{i,m}$ und der Slave-Zellen $x_{i,s}$ benötigt. Die Werte werden unabhängig voneinander über alle KV-Flächen des Gebietsinneren, der Interfaces und der Gebietsränder bestimmt. Die Zellflächeninterpo-

lationsfaktoren ermöglichen die Interpolation einer Variable, die als Zellenfeld gespeichert ist, an die KV-Flächen, siehe Kapitel 4.2.2. Die Vorschrift zur Bestimmung der Interpolationsfaktoren λ_f lautet:

$$\lambda_f = \frac{\left| x_{i,f} - x_{i,m} \right|}{\left| x_{i,s} - x_{i,m} \right|}. \tag{4.9}$$

Zur Gradientenberechnung der Zellvariablen kann neben dem Gaußschen Integralsatz auch der Least-Squares-Ansatz herangezogen werden, siehe beispielsweise open∇FOAM [64] und Gleichung (4.15). Für die Berechnung der Hilfstensoren werden dabei lediglich die Mittelpunkte der Master-Zellen $x_{i,m}$ und der Slave-Zellen $x_{i,s}$ benötigt:

$$s_{i,f} = \left(x_{i,s} - x_{i,m} \right), \tag{4.10}$$

$$G_{ij,c} = \left(\sum_{f_c} \frac{s_{i,f} \cdot s_{i,f}^{\,T}}{s_{i,f}^{\,2}} \right)^{-1}. \tag{4.11}$$

Die Werte werden als erstes für alle Zellen des Gebietsinneren gebildet. Danach werden die Tensoren an den Interfaces ausgetauscht und an den entarteten Zellen der Gebietsränder als Nulltensoren angenommen.

4.1.4 Geometriemodifikation

Konservative Diskretisierungen auf bewegten numerischen Netzen müssen einer geometrischen Zwangsbedingung folgen, die als Space Conservation Law (SCL) oder Raumerhaltungsgleichung bekannt ist, siehe THOMAS und LOMBARD [93]. Die Formel lässt sich direkt über die Leibnitz Regel aus Gleichung (1.5) herleiten, wenn $f = 1$ gesetzt wird:

$$\frac{d}{dt} \int_{\Omega} d\Omega = \int_{S} w_i n_i dS \,. \tag{4.12}$$

Gleichung (4.12) besagt, dass die Änderung des KV-Inhalts während eines Zeitintervals gleich der Änderung der durch die Bewegung generierten Flüsse über die KV-Grenzen sein muss. Werden bei der Diskretisierung der Erhaltungsgleichungen Zellflächengeschwindigkeiten verwendet, die das SCL nicht erfüllen, entstehen im Rechengebiet ungewollt Massequellen und dadurch unter Umständen auch Konvergenzschwierigkeiten. Verformen sich die Netze bei der Bewegung nicht, wie das beispielsweise in den rotierenden Schaufelkanälen von Turbomaschinen der Fall ist, können die Zellflächengeschwindigkeiten direkt aus der vorgegebenen Netzkinematik errechnet werden, siehe SKODA [86].

4.2 Repräsentation der Erhaltungsgleichungen

4.2.1 Equation- und Scope-Klassen

Die *Equation*-Klasse definiert die zentrale Schnittstelle für die Diskretisierung der unterschiedlichen Erhaltungsgleichungen. Alle Gleichungen müssen im CFD-Tool deshalb von der *Equation*-Klasse abgeleitet werden. Die Schnittstelle der *Equation*-Klasse beinhaltet die *init()*-Funktion und die *iterate()*-Funktion. Die init()-Funktion belegt das Strömungsgebiet zu Beginn einer Rechnung mit der vorgegebenen Startlösung und den Randwerten der Lösungsvariablen der *Equation*-Klasse. Die *iterate()*-Funktion diskretisiert und löst das Problem einmal. Falls die zu lösende Problemstellung mehrere Gleichungsobjekte beinhaltet, die gekoppelte Gleichungen repräsentieren, muss die *iterate()*-Funktion so lange für jedes Gleichungsobjekt aufgerufen werden, bis die Gesamtlösung konvergiert ist. Aus diesem Grund liefert die *iterate()*-Funktion nach dem Lösungsschritt das erzielte Endresiduum, siehe Kapitel 4.3, zurück. Die Kontrolle der momentanen Lösungsqualität im Verhältnis zum eingestellten Abbruchkriterium ist damit direkt möglich, siehe Kapitel 4.4.

Die Diskretisierung der Gleichungen erfolgt basierend auf dem numerischen Netz, das durch die *PolyMesh*-Klasse repräsentiert wird. Deshalb speichert die Equation-Klasse einen entsprechenden Verweis auf ein *PolyMesh*-Objekt. Der Verweis kann über die Funktionen *get_mesh()* und *set_mesh()* bearbeitet werden.

In der *PolyMesh*-Klasse werden die unterschiedlichen Bereiche des numerischen Netzes durch Slice-Objekte dargestellt, siehe Kapitel 4.1.1. Das Gegenstück hierzu bilden die Scope-Objekte in der *Equation*-Klasse. Die Scopes repräsentieren dabei die Randbedingungen der Gleichungen. Zu den einzelnen Slices gibt es korrespondierende Scopes. Dem *InternalPolySlice* ist das *InternalScope*, dem *PolyInterfaceSlice* das *InterfaceScope* und dem *PolyBoundarySlice* das *BoundaryScope* zugeordnet. Die *Equation*-Klasse und die Scopes bilden damit zusammen mit der *PolyMesh*-Klasse und den Slices eine duale Vererbungshierarchie. Für die Angabe neuer bzw. das Löschen alter Scopes besitzt die *Equation*-Klasse Zugriffsfunktionen, die denen für die Slices in der *PolyMesh*-Klasse gleichen. Wie die *PolyMesh*- und die Slice-Klassen aus Kapitel 4.1.1 sind auch die *Equation*- und die Scope-Klassen als Zeigerklassen konzipiert. Die Eigenschaften zur Referenzzählung erben sie wieder von der Basisklasse *Base*.

Da für die Diskretisierung der Randbedingungen der Start- und Stop-Index des zugehörigen Randbereiches des Netzes benötigt wird, speichert jedes Scope-Objekt einen direkten Verweis auf das assoziierte Slice-Objekt. Zur Implementierung der *get_slice()*-Funktionalität in den Scopes wird eine Strategie verwendet, die als „Covariant Return Types“ bezeichnet wird.

Das vorgestellte Design für die Gleichungsdiskretisierung ordnet die Randbedingungen den Gleichungen zu. Als Alternative könnten primitive Randbedingungen, also vorgegebene Werte und Gradienten, auch direkt den Lösungsvariablen zugeordnet werden. Dies erscheint im ersten Hinblick logisch, da die Randbedingungen ja direkt die Werte der Lösungsvariablen beeinflussen. Im CFD-Tool wird dennoch der andere Weg eingeschlagen. Dies liegt zum Einen daran, dass nicht nur primitive sondern auch gemischte Randbedingungen implementierbar sein sollen, die gleichzeitig mehrere Größen beeinflussen. Eine solche Randbedingung ist der vorgegebene Totaldruck am Rechengebietseintritt, der implizit sowohl den statischen Druck als auch die Geschwindigkeiten vorgibt.

Ein anderer wichtiger Grund die Klassenhierarchien der Randbedingungen direkt auf die Gleichungsklassen zu beziehen, besteht darin, Systeme aus mehreren Gleichungen in einem Objekt zusammengefasst betrachten zu können. Ein Beispiel hierfür ist die enge Koppelung zwischen Impulsgleichungen und Massenerhaltung, die in Abhängigkeit diskretisiert und gelöst werden können, siehe ZWART [108]. In diesem Fall ist die Formulierung der Randbedingungen bezogen auf das Gesamtsystem wesentlich bequemer, als für jede Lösungsvariable einzeln.

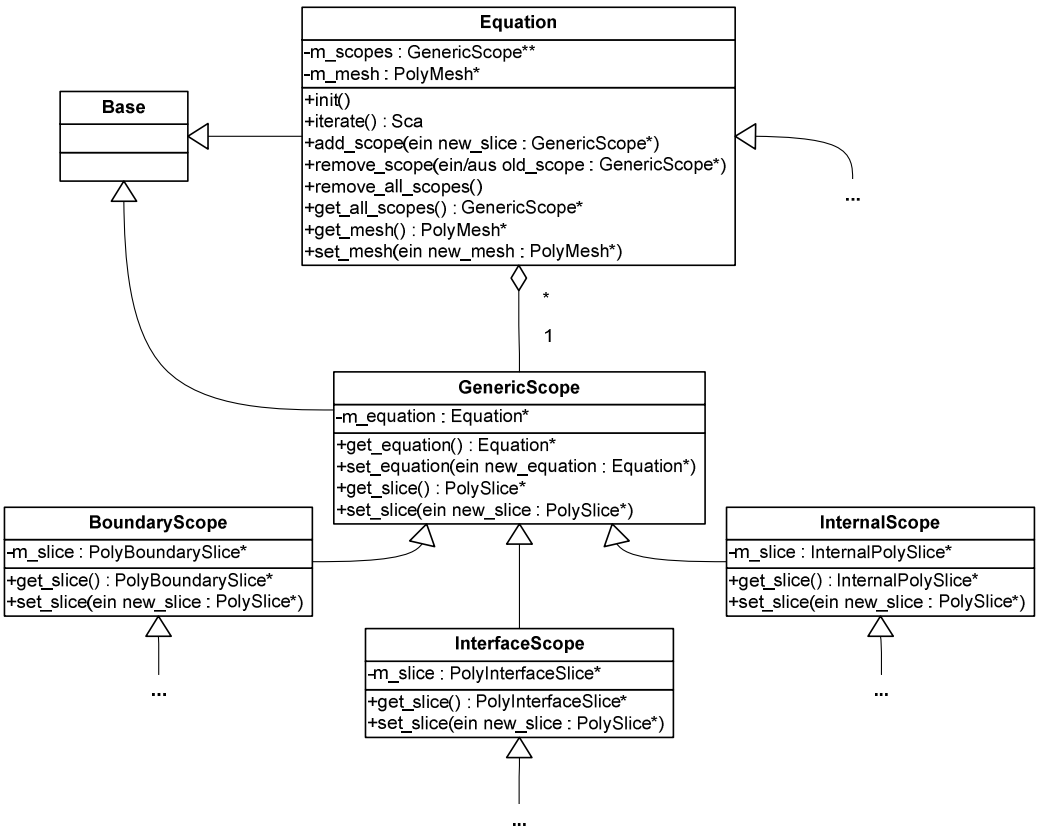


Bild 4-4: Die *Equation*-Klasse als Basis für die Gleichungsdiskretisierung; Die *Scope*-Klassen implementieren verschiedene Randbedingungen; Die *Equation*-Klasse und die *Scope*-Klassen bilden eine duale Vererbungshierarchie mit der *Polymesh*-Klasse und den *PolySlice*-Klassen

4.2.2 Berechnungsschemata

Dieser Abschnitt stellt den Entwurf der Diskretisierungsschemata des CFD-Tools vor. Die Klassen übernehmen die korrekte Diskretisierung der Terme in den Differentialgleichungen, dabei sind verschiedene Versionen für vektorielle und skalare Terme vorgesehen. Es gibt Schemata zur Gradientenberechnung, sowie Konvektions-, Diffusions-, Massenstrom- und Zeitschemata. Die unterschiedlichen Randbedingungen werden, wie zuvor bei der Berechung der Geometrievariablen, über entsprechende Member-Funktionen berücksichtigt. Die Gradientenschemata besitzen die Funktionen *on_internal()*, *on_interface()* und *on_boundary()*, die Diffusions-, Konvektions- und Massenstromschemata dagegen *on_internal()*, *on_interface()* sowie *on_fix_grad()* bzw. *on_fix_value()* für die ent-

sprechenden Randbedingungen. Für die Zeitschemata gibt es explizite und implizite Implementierungen. Sie besitzen darüber hinaus lediglich die Funktion *on_internal()* zur Diskretisierung. Bild 4-5 zeigt exemplarisch die statische Klassenhierarchie für die skalaren Konvektionsschemata, die zur flexiblen Auswahl wieder als Zeigerklassen konzipiert sind. Um die Diskretisierungsschemata von den reinen Berechnungsschemata aus Kapitel 4.1.3, abzugrenzen sind diese im Namensraum *Impl::* bzw. *Expl::* angesiedelt.

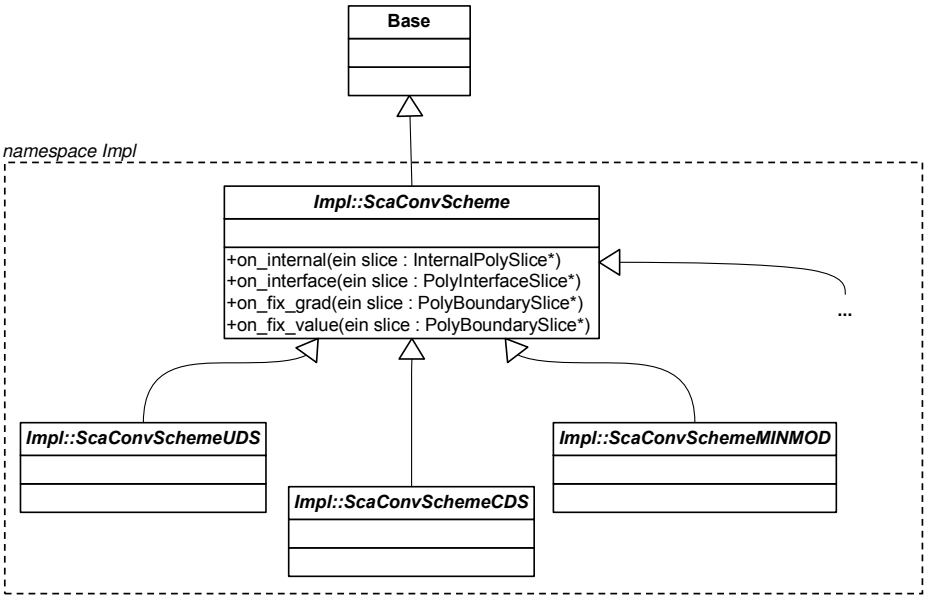


Bild 4-5: Die Klassenhierarchie der skalaren Konvektionsschemata als Beispiel für das Design der Diskretisierungsschemata; Die Konvektionsschemata unterscheiden nach Feldinnerem, Interfaces, von Neumann- und Dirichlet-Randbedingungen

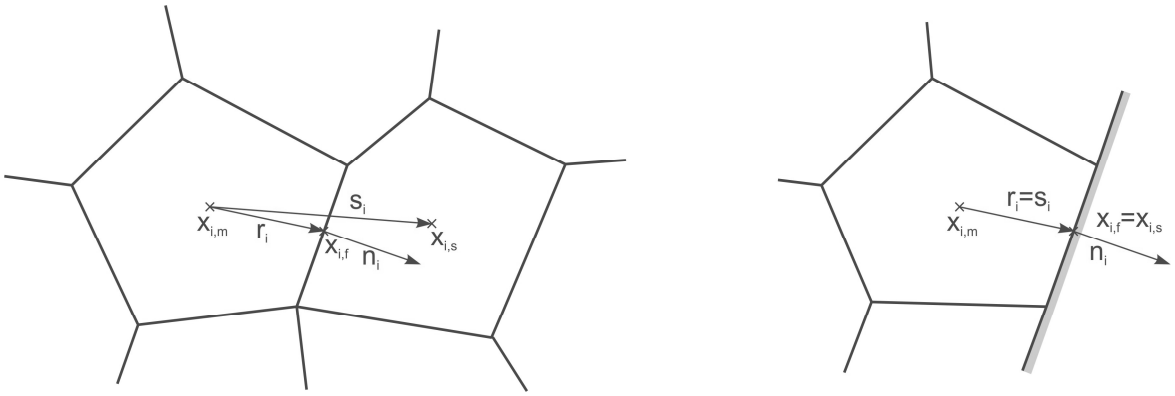


Bild 4-6: Definition einiger häufig für die Diskretisierung benötigter geometrischer Größen

Die folgenden Abschnitte beschäftigen sich mit der Berechnung der Gradienten, sowie der Diskretisierung der einzelnen Terme am Beispiel der skalaren Transportgleichung (1.9). Dafür werden zunächst einige wichtige geometrische Größen definiert, siehe Bild 4-6. $s_{i,f}$ bezeichnet den Zellverbindungsvektor zwischen der Master-Zelle *m* und der Slave-Zelle *s*. Ist dieser Vektor normiert, dann ist er als $\tilde{s}_{i,f}$ dargestellt. $n_{i,f}$ ist der Nor-

maleneinheitsvektor an der Zellfläche *f*, siehe auch Kapitel 4.1.3. Der Verbindungsvektor zwischen Zellmittelpunkt $x_{i,m}$ und Flächenmittelpunkt $x_{i,f}$ schließlich wird als $r_{i,f}$ bezeichnet.

Gradientenberechnung

Bei der Diskretisierung sind die Gradienten der in den Zellen gespeicherten Lösungsvariablen erforderlich. Diese können im CFD-Tool auf zwei unterschiedliche Arten berechnet werden. Mit dem Gaußschen Integralsatz und der Approximation der Oberflächen- und Volumenintegrale lässt sich eine Beziehung zwischen dem Gradient der skalaren Variable ϕ im Zellmittelpunkt und ihren Werten an den Flächenmittelpunkten eines KV herstellen:

$$\left(\frac{\partial \phi}{\partial x_i}\right)_c \approx \frac{\sum_{f_c} \phi_f n_{i,f} S_f}{\Omega_c}, \tag{4.13}$$

$$\phi_f = \lambda_f \phi_s + (1-\lambda_f)\phi_m . \tag{4.14}$$

Gleichung (4.14) stellt hierbei eine lineare Interpolation der Variablenwerte ϕ_m der Master- und ϕ_s der Slave-Zelle an die KV-Fläche *f* mithilfe des Interpolationsfaktors λ_f, siehe Gleichung (4.8), dar. In den Anwendungsbeispielen dieser Arbeit wurde ausschließlich die Gaußsche Methode zur Gradientenberechnung herangezogen.

Beim Least-Squares-Ansatz wird der Gradient als Minimierungsproblem aufgefasst und nach folgendem Prinzip gelöst:

$$\left(\frac{\partial \phi}{\partial x_i}\right)_c \approx \sum_{f_c} \frac{1}{s_{i,f}^{\,2}} G_{ij,m} \cdot s_{i,f} (\phi_s - \phi_m), \tag{4.15}$$

wobei der Zellverbindungsvektor $s_{i,f}$ nach Gleichung (4.10), und der Hilfstensor $G_{ij,m}$ nach Gleichung (4.11) berechnet werden.

Die beiden Berechnungsmethoden liegen im CFD-Tool sowohl in einer skalaren als auch in einer vektoriellen Form vor, die darüber hinaus zur Stabilisierung der Diskretisierung unterrelaxiert werden können. Die Relaxation erfolgt nach folgender Vorschrift, wobei mit *old* die letztbekannten Werte und mit *new* die neu berechneten Werte gekennzeichnet werden:

$$\left(\frac{\partial \phi}{\partial x_i}\right)_c = \omega^g \left(\frac{\partial \phi}{\partial x_i}\right)_c^{new} + (1-\omega^g)\left(\frac{\partial \phi}{\partial x_i}\right)_c^{old} . \tag{4.16}$$

Der Relaxationsfaktor ω^g liegt normalerweise bei etwa 0.8.

Konvektive Flüsse

Der konvektive Fluss F_f^c in Gleichung (1.9) über die KV-Fläche *f* lautet:

$$F_f^c = \left(\dot{m} - \dot{m}_w\right)_f \phi_f \,, \tag{4.17}$$

wobei die Differenz $\dot{m} - \dot{m}_w$ den relativen Massenstrom durch die KV-Oberfläche *f* darstellt. Dieser wird als bekannt angenommen und aus dem momentanen Geschwindigkeitsfeld berechnet. Zur Diskretisierung der konvektiven Flüsse muss nun die Zellvariable ϕ von der Master-Zelle *m* und der Slave-Zelle *s* an die KV-Flächen *f* interpoliert werden. Die Genauigkeit der gewählten Interpolationsvorschrift steht dabei im Gegensatz zur resultierenden Stabilität des numerischen Verfahrens, siehe PATANKAR [66].

Im Folgenden werden einige Interpolationsverfahren, die im CFD-Tool verfügbar sind, vorgestellt. Zur übersichtlichen und dimensionslosen Darstellung wird die Normalized-Variable-Formulation (NVF) nach LEONARD und MOKHTARI [52] herangezogen. Diese beruht auf einer lokal eindimensionalen Betrachtung des konvektiven Flusses. Mit dem Wert ϕ_U der stromauf liegenden Zelle *U* und dem Wert ϕ_D der stromab liegenden Zelle *D* kann die Variable ϕ zunächst in eine dimensionslose Darstellung gebracht werden:

$$\tilde{\phi} = \frac{\phi - \phi_U}{\phi_D - \phi_U} = 1 - \frac{\phi_D - \phi}{\phi_D - \phi_U} \,. \tag{4.18}$$

Im Gegensatz zu strukturierten Netzen ist es bei allgemein unstrukturierten Netzen nicht möglich den Wert ϕ_U der stromauf liegenden Zelle *U* direkt anzugeben. Um diesen Konflikt aufzulösen, wird eine Formulierung nach JASAK et. al [41] verwendet, bei der die Differenz $\phi_D - \phi_U$ durch den Gradienten von ϕ der zentralen Zelle C ausgedrückt wird:

$$\phi_D - \phi_U = 2\left(\frac{\partial \phi}{\partial x_i}\right)_C \left(x_{i,D} - x_{i,C}\right). \tag{4.19}$$

Der Verbindungsvektor $x_{i,D} - x_{i,C}$ der Zellmittelpunkte der stromab liegenden und der zentralen Zelle entspricht dabei der Definition von s_i aus Bild 4-6. Durch Einsetzen von Gleichung (4.19) in Gleichung (4.18) ergibt sich für die dimensionslose Variable $\tilde{\phi}$:

$$\tilde{\phi} = 1 - \frac{\phi_D - \phi}{2\left(\frac{\partial \phi}{\partial x_i}\right)_C \left(x_{i,D} - x_{i,C}\right)} \,. \tag{4.20}$$

Gleichung (4.20) benötigt den stromauf liegenden Wert ϕ_U nicht mehr und kann deshalb problemlos auch im unstrukturierten Fall verwendet werden. Welche der beiden an die jeweilige Zellfläche *f* grenzenden Zellen als zentrale bzw. stromab liegende Zelle gewertet wird, ist abhängig vom lokalen Massenstrom $\dot{m} - \dot{m}_w$. Insgesamt ergibt sich:

$$\begin{aligned} &\phi_C = \phi_m; \quad \phi_D = \phi_s; \quad x_{i,D} - x_{i,C} = s_i; \qquad \lambda = \lambda_\mathrm{f}; \qquad \forall\left(\dot{m} - \dot{m}_w\right)_f \geq 0 \\ &\phi_C = \phi_s; \quad \phi_D = \phi_m; \quad x_{i,D} - x_{i,C} = -s_i; \quad \lambda = \left(1 - \lambda_\mathrm{f}\right); \quad \forall\left(\dot{m} - \dot{m}_w\right)_f < 0 \end{aligned}, \tag{4.21}$$

wobei m wieder die Master-Zelle und s die Slave-Zelle bezeichnet. In Gleichung (4.21) ist außerdem der allgemeine Interpolationsfaktor λ mit angegeben, der aus dem Zellflächeninterpolationsfaktor λ_f, siehe Gleichung (4.8), ebenfalls in Abhängigkeit vom lokalen Massenstrom $\dot{m} - \dot{m}_w$ berechnet werden kann.

Die einfachste Interpolationsvorschrift zur Diskretisierung der konvektiven Flüsse ist die Stromauf-Interpolation bzw. das Upwind-Differencing-Scheme (UDS). Die Interpolationsvorschrift nimmt als Zellflächenwert $\tilde{\phi}_f$ einfach den Wert $\tilde{\phi}_C$ der stromauf liegenden Zelle an:

$$\tilde{\phi}_f = \tilde{\phi}_C . \tag{4.22}$$

Das UDS ist uneingeschränkt stabil, aber lediglich erster Ordnung genau. Der entstehende Diskretisierungsfehler wirkt bei Strömungssimulation wie eine scheinbar erhöhte physikalische Diffusion, siehe zum Beispiel JUSUF [43]. Trotz dieses Nachteils wird das UDS wegen seiner hohen Stabilität aber häufig zusammen mit hochauflösenden Interpolationsschemata eingesetzt. Dabei wird das UDS impliziten berücksichtigt, und das hochauflösende Schema über einen Korrekturterm explizit nachgezogen. Die Methode wurde erstmals von KHOSLA und RUBIN [45] vorgeschlagen. Für den Zellflächenwert ϕ_f ergibt sich:

$$\phi_f = \left(\phi_f^{UDS}\right)^{impl} + \beta\left(\phi_f^{HOS} - \phi_f^{UDS}\right)^{\exp l}, \tag{4.23}$$

wobei β als Blending-Faktor mit $0 \leq \beta \leq 1$ fungiert. ϕ_f^{HOS} dient als Platzhalter für das hochauflösende Diskretisierungsschema (High-Order-Scheme).

Das Central-Differencing-Scheme (CDS) ist eine Interpolationsvorschrift, die zweiter Ordnung genau ist. In der dimensionslosen Darstellung der NVF lautet sie:

$$\tilde{\phi}_f = \lambda + (1 - \lambda)\tilde{\phi}_C . \tag{4.24}$$

Nachteilig am CDS ist, dass es im Gegensatz zum UDS nicht uneingeschränkt stabil ist und oszillierende Lösungen erzeugen kann, siehe PATANKAR [66]. Im CFD-Tool ist das CDS über die nachgezogene Korrektur nach Gleichung (4.23) implementiert

Die stabile und gleichzeitig zweite Ordnung genaue Approximation der konvektiven Flüsse ist ein fundamentales Problem in der Numerik. Hochauflösende limitierende Verfahren zielen darauf ab, eine stabile Diskretisierung höherer Ordnung bereitzustellen. Sie stellen eine Kombination verschiedener Basisverfahren dar, siehe SKODA [86]. Im Rahmen dieser Arbeit werden das OSHER-Schema nach CHAKRA-VARTHY und OSHER [14], sowie das MINMOD-Schema (MINimum MODulus) nach HARTEN [36] verwendet.

Die Interpolationsvorschrift für das OSHER-Schema lautet in der NVF:

$$\tilde{\phi}_f = \begin{cases} (\lambda+1)\tilde{\phi}_C & 0 \le \tilde{\phi}_C \le \dfrac{1}{(\lambda+1)} \\ 1 & \dfrac{1}{(\lambda+1)} \le \tilde{\phi}_C \le 1 \\ \tilde{\phi}_C & sonst \end{cases} . \tag{4.25}$$

Mit dem MINMOD-Schema ergibt sich in der NVF für die Variable $\tilde{\phi}_f$:

$$\tilde{\phi}_f = \begin{cases} (\lambda+1)\tilde{\phi}_C & 0 \le \tilde{\phi}_C \le \dfrac{1}{2} \\ \lambda + (1-\lambda)\tilde{\phi}_C & \dfrac{1}{2} \le \tilde{\phi}_C \le 1 \\ \tilde{\phi}_C & sonst \end{cases} . \tag{4.26}$$

Bild 4-7 zeigt das Normalized-Variable-Diagramm (NVD) für das UDS, das CDS, das OSHER- sowie das MINMOD-Schema. Das ausgegraute Dreieck sowie die Diagonale des UDS spannen zusammen den stabilen Bereich auf. Sowohl das OSHER- als auch das MINMOD-Schema verlassen diesen Bereich nicht. Nur das CDS kann Instabilitäten erzeugen. Bild 4-7 macht zudem deutlich, wie stark der Verlauf der Kurven beim CDS, beim OSHER- und beim MINMOD-Schema von der Inhomogenität des Rechennetzes ($\lambda < 0{,}5$ oder $\lambda > 0{,}5$) abhängt. Unabhängig davon sind die drei letztgenannten Verfahren allesamt zweiter Ordnung genau.

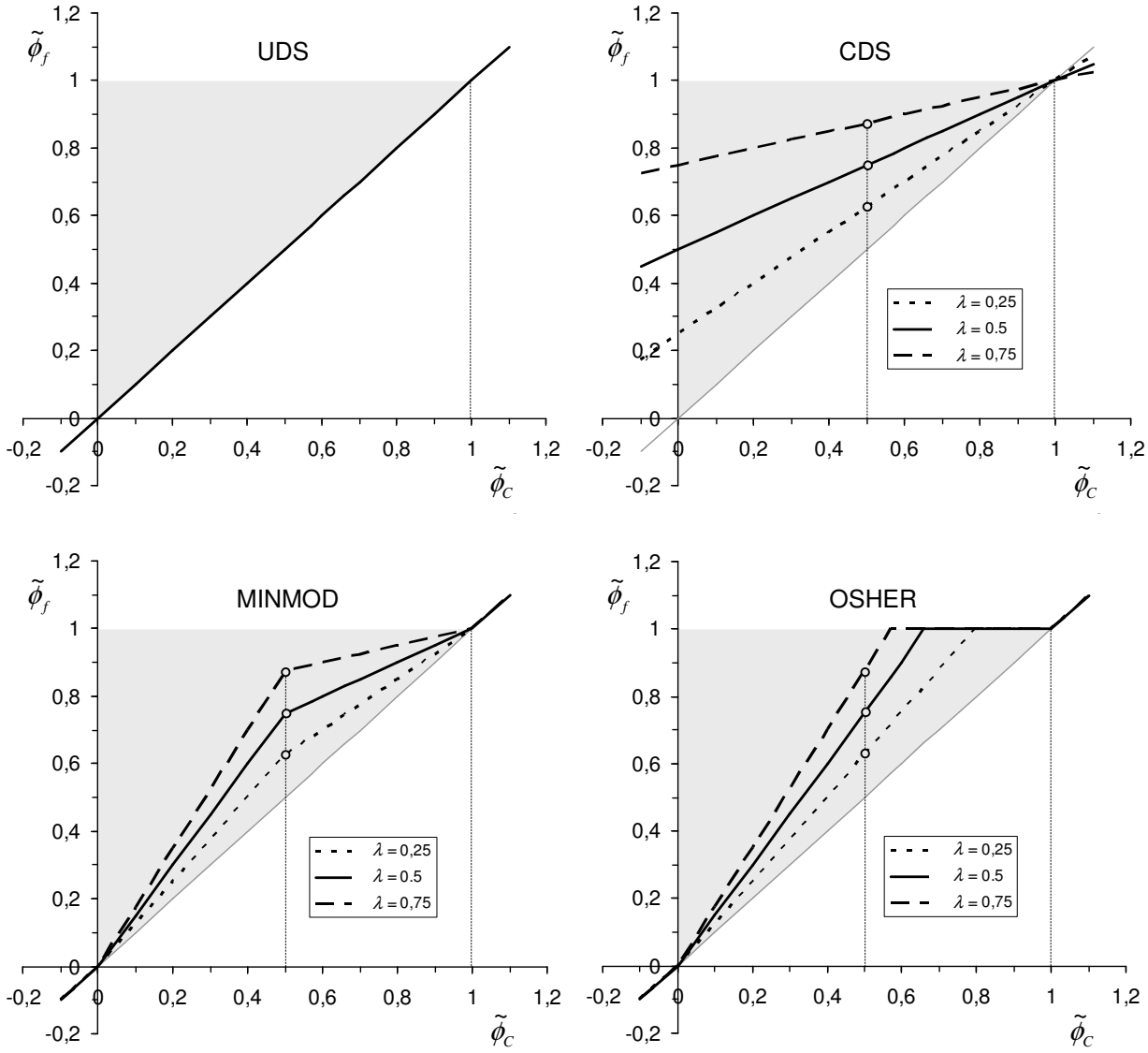


Bild 4-7: Diskredisierungsschemata der konvektiven Flüsse im NVD; Die Verläufe der Kurven ergeben sich durch variierende Werte des Interpolationsfaktors λ; Für homogene Rechennetze gilt $\lambda = 0,5$

Diffusive Flüsse

Auf die linke Seite gebracht lautet der diffusive Fluss F_f^d in Gleichung (1.9) über die KV-Fläche *f*:

$$F_f^d = -\left(\Gamma \frac{\partial \phi}{\partial x_i} n_i\right)_f S_f . \tag{4.27}$$

Der Gradient von ϕ in der Master-Zelle *m* bzw. der Slave-Zelle s lässt sich über den Gauss-Ansatz nach Gleichung (4.13) oder den Least-Squares-Ansatz nach Gleichung (4.15) berechnen und anschließend mit der linearen Interpolation nach Gleichung (4.14) an die Zellfläche *f* interpolieren. Die implizite Verarbeitung der entstehenden Terme

würde aber nicht nur die unmittelbaren, sondern auch die nächst weiter entfernten Nachbarzellen des KV betreffen. Dies würde zu großen Rechenmolekülen führen, die im CFD-Tool so nicht vorgesehen sind. MUZAFERIJA [61] schlägt vor, statt der Ortsableitung von ϕ in Normalenrichtung die Änderung in Zellverbindungsrichtung zu verwenden:

$$\frac{\partial \phi}{\partial x_i} \cdot n_i \approx \frac{\partial \phi}{\partial x_i} \cdot \tilde{s}_i = \frac{\phi_s - \phi_m}{|s_i|}, \tag{4.28}$$

wobei $\tilde{s}_i$ der normierte Zellverbindungsvektor s_i ist. Für wenig gescherte KV gilt $n_i \approx \tilde{s}_i$, vergleiche Bild 4-6. Der Term in Gleichung (4.28) kann leicht implizit diskretisiert werden. Da neben der Master-Zelle *m* lediglich die Slave-Zellen *s* als unmittelbarer Nachbar betroffen ist, bleiben die resultierenden Rechenmoleküle klein. Die Abweichung dieser Approximation kann zusätzlich über eine explizite Nachkorrektur kompensiert werden:

$$F_f^d = -\Gamma_f \left[\frac{\partial \phi}{\partial x_i} \cdot \tilde{s}_i + \overline{\frac{\partial \phi}{\partial x_i}} \cdot \left(n_i - \tilde{s}_i \right) \right]_f S_f . \tag{4.29}$$

Der überstrichene Term in Gleichung (4.29) bezeichnet den linear nach Gleichung (4.14) an die Zellfläche *f* interpolierten Gradienten von ϕ, der explizit in der Diskretisierung behandelt wird. Mit der Approximation aus Gleichung (4.28) ergibt sich dann für die diffusiven Flüsse:

$$F_f^d = -\Gamma_f \left\{ \left[\frac{\left(\phi_s - \phi_m \right)}{|s_i|} \right]^{impl} + \left[\overline{\frac{\partial \phi}{\partial x_i}} \cdot \left(n_i - \tilde{s}_i \right) \right]^{expl} \right\}_f S_f . \tag{4.30}$$

ZWART [108] schlägt eine leicht modifizierte Variante von Gleichung (4.30) vor:

$$F_f^d = -\Gamma_f \left\{ \left[\alpha \frac{\left(\phi_s - \phi_m \right)}{|s_i|} \right]^{impl} + \left[\overline{\frac{\partial \phi}{\partial x_i}} \cdot \left(n_i - \alpha \tilde{s}_i \right) \right]^{expl} \right\}_f S_f \tag{4.31}$$

mit

$$\alpha = \frac{1}{n_i \tilde{s}_i} . \tag{4.32}$$

Hierin ist α ein zusätzlicher Skalierungsfaktor. Fällt die Richtung von n_i mit der Richtung von $\tilde{s}_i$ zusammen so gilt $\alpha = 1$, andernfalls verstärkt der Faktor den implizit verar-

beiteten Term in Gleichung (4.31). Für stark verzerrte Netze führt ZWART [108] zusätzlich den Faktor ω^d ein. Die diffusiven Flüsse schreiben sich dann folgendermaßen:

$$F_f^d = -\Gamma_f \left\{ \left[\frac{(\phi_s - \phi_m)}{|s_i|} \cdot \left(\alpha + \frac{1-\alpha}{\omega^d} \right) \right]^{impl} + \left[\overline{\frac{\partial \phi}{\partial x_i}} \cdot (n_i - \alpha \tilde{s}_i) - \frac{(\phi_s - \phi_m)}{|s_i|} \cdot \frac{1-\alpha}{\omega^d} \right]^{\exp l} \right\}_f S_f \cdot \quad (4.33)$$

ω^d hat dabei typischerweise einen Wert von 0.8.

Beide Diskretisierungsmethoden für die konvektiven Flüsse sind als Schemataklassen im CFD-Tool hinterlegt. Die Ergebnisse im Anwendungsteil wurden dabei ausschließlich nach der Methodik aus Gleichung (4.30) erzielt.

Im Falle einer vektoriellen Größe ist der diffusive Fluss umfangreicher als für eine skalare Größe. Im Falle der Impulsgleichungen lautet der diffusive Fluss mit dem Newtonschen Stoffgesetz für den viskosen Spannungstensor beispielsweise:

$$F_{i,f}^{u,d} = -\mu_f \left(\frac{\partial u_i}{\partial x_j} n_j + \frac{\partial u_j}{\partial x_i} n_j \right)_f S_f - \mu_f \left(\frac{2}{3} \delta_{ij} \frac{\partial u_k}{\partial x_k} n_j \right)_f S_f \,. \quad (4.34)$$

Der erste Term in Gleichung (4.34) entspricht dem skalaren Term aus Gleichung (4.27). Deshalb wird er analog dazu diskretisiert. Die beiden restlichen Terme werden explizit mit den an die Zellflächen *f* interpolierten Gradienten der Geschwindigkeiten berechnet. Im CFD-Tool ist der diffusive Fluss für die Impulsgleichungen als eigenes Diskretisierungsschema hinterlegt.

Instationärer Term

Der Term für die Zeitableitung aus Gleichung (1.9) gibt die zeitliche Entwicklung der skalaren Größe ϕ wieder und wird als Volumenintegral behandelt. Für die Zeitableitung selbst stehen verschiedene implizite und explizite Schemataklassen zur Verfügung. Beispielhaft sollen an dieser Stelle die Euler-Diskretisierung:

$$\frac{d(\rho\phi\Omega)_c}{dt} = \frac{(\rho\phi\Omega)_c^n - (\rho\phi\Omega)_c^o}{\Delta t}, \quad (4.35)$$

sowie die Drei-Zeitebenen-Methode, siehe SKODA [86]:

$$\frac{d(\rho\phi\Omega)_c}{dt} = \frac{(3\rho\phi\Omega)_c^n - (4\rho\phi\Omega)_c^o + (\rho\phi\Omega)_c^{oo}}{2\Delta t} \quad (4.36)$$

genannt werden. In Gleichung (4.35) und (4.36) steht *n* für den aktuellen Zeitschritt, wohingegen *o* einen und *oo* zwei Zeitschritte davor bezeichnet. Die Euler-Diskretisierung ist lediglich erster Ordnung genau, wohingegen die Drei-Zeitebenen-Methode eine Genauigkeit zweiter Ordnung aufweist. Nachteilig dabei ist allerdings, dass die Methode Werte von ϕ für die letzten beiden Zeitschritte benötigt. Die Drei-Zeitebenen-Methode ist als rein implizites Zeitschrittverfahren implementiert. Die Euler-Methode steht als implizites sowie explizites Verfahren zur Verfügung. Im expliziten Fall müssen die restlichen Terme von Gleichung (1.9) zusätzlich mit Werten von ϕ^o berechnet werden. Das explizite Schema benutzt hierfür einfach das durch die Diskretisierung der übrigen Terme entstandene LGS und wertet es für ϕ^o aus.

Massenströme

Für die Auswertung der konvektiven Terme in Gleichung (1.9) wird der Massenstrom $\dot{m}_f$ an den Zellflächen benötigt. Im CFD-Tool ist der Massenstrom direkt als Flächenvariable vorgesehen. Er kann aus den Zellwerten der Master-Zelle *m* und der Slave-Zelle *s* folgendermaßen berechnet werden:

$$\dot{m}_f = \left[\overline{\rho} \cdot \left(\overline{u_i} + \overline{\frac{\partial u_i}{\partial x_j}} r_j \right) n_i \right]_f S_f \tag{4.37}$$

mit

$$r_i = x_{i,f} - \overline{x_{i,c}} \,. \tag{4.38}$$

Die überstrichenen Terme bezeichnen dabei eine lineare Interpolation entsprechend Gleichung (4.14). r_i stellt hier den Verbindungsvektor zwischen dem Flächenmittelpunkt $x_{i,f}$ und dem Integrationspunkt $\overline{x_{i,c}}$ der Linearkombinationen aus $x_{i,m}$ und $x_{i,s}$ dar. Sind die KV des numerischen Netzes wenig geschert, verschwindet $\overline{x_{i,c}}$.

Quellterme

Die Quellterme werden als Volumenintegrale behandelt und können durch Iteration über alle KV des numerischen Netzes sehr einfach bestimmt werden. Da die Quellterme der einzelnen Gleichungen sich in der Regel stark unterscheiden, siehe beispielsweise die Terme auf der rechten Seite der turbulenten Gleichungen in Kapitel 5.1.2, ist für sie kein eigener Klassenentwurf als Diskretisierungsschema vorgesehen.

4.3 Löser für lineare Gleichungssysteme

Bei der impliziten Diskretisierung entsteht für die einzelnen partiellen Differentialgleichungen jeweils ein LGS, das die algebraischen Bilanzgleichungen für jedes KV im Lösungsgebiet enthält. Die einzelnen Gleichungen haben die Form:

$$a_P \phi_P + \sum_{NB} a_{NB} \phi_{NB} = b_P \,. \tag{4.39}$$

Die Matrixeinträge a_P der Hauptdiagonalen und a_{NB} der Nachbarn beinhalten dabei die bei der Diskretisierung implizit behandelten Terme, wohingegen die rechte Seite b_P die explizit behandelten Terme, die nachgezogenen Korrekturen sowie die Beiträge der Randbedingungen enthält. Zur Stabilisierung von Gleichung (4.39) muss diese in mehreren Schritten gelöst werden. In jedem Schritt wird immer nur ein Bruchteil der jeweiligen Lösung verwendet. Diese Methodik stellt keinen zusätzlichen zeitlichen Nachteil dar, da das mathematische Modell fluidmechanischer Problemstellungen aus mehreren voneinander abhängigen Gleichungen besteht, die im Allgemeinen ohnehin in einer Iterationsschleife sequentiell hintereinander gelöst werden, siehe Kapitel 4.4. Mit dem Unterrelaxationsfaktor ω^ϕ stellt sich folgender funktionaler Zusammenhang für die Variable ϕ zum Iterationsschritt n ein:

$$\phi_P^n = \phi_P^{n-1} + \omega^\phi \left(\phi_P'^n - \phi_P^{n-1} \right), \tag{4.40}$$

wobei ϕ' die eigentliche Lösung des LGS bezeichnet. Die Beziehung aus Gleichung (4.40) kann in Gleichung (4.39) eingearbeitet werden. Insgesamt ergibt sich dann für das zu lösende LGS:

$$\underbrace{\frac{a_P}{\omega^\phi}}_{a_P^*} \phi_P^n + \sum_{NB} a_{NB} \phi_{NB}^n = \underbrace{b_P + \frac{1-\omega^\phi}{\omega^\phi} a_P \phi_P^{n-1}}_{b_P^*} \,. \tag{4.41}$$

Gleichung (4.41) weist gegenüber Gleichung (4.39) eine erhöhte Diagonaldominanz auf. Darin liegt die stabilisierende Wirkung dieser Methodik begründet.

Zur Lösung des LGS (4.41) greift das CFD-Tool auf eine Programmbibliothek zurück, die von FLURL [24] entwickelt worden ist. Sie enthält mit den beiden vorkonditionierten Krylov-Unterraum-Verfahren BiCGstab nach VAN DER VORST [96] und BiCGstab(l) nach SLEIJPEN et al. [87], sowie der ILU-D und einem algebraischen Mehrgitterverfahren nach RAW [68] verschiedene hocheffiziente iterative Lösungsalgorithmen.

Zur Bewertung der Güte der Lösung zum Iterationsschritt n wird das Residuum herangezogen. Diese Größe beschreibt das Ungleichgewicht zwischen rechter und linker Seite des LGS und ist wie folgt definiert:

$$r_P^n = b_P^* - a_P^* \phi_P^n + \sum_{NB} a_{NB} \phi_{NB}^n \ . \tag{4.42}$$

r_P ist eine Feldvariable. Zur einfacheren Beurteilung wird sie betragsweise ausgewertet. Die im CFD-Tool verwendete P2- oder Euklidische Norm lautet:

$$\left\| r_P^n \right\|_2 = \sqrt{\sum_P \left(r_P^n \right)^2} \ . \tag{4.43}$$

4.4 Programmablaufsteuerung

Die Programmablaufsteuerung eines CFD-Codes ist eng mit der gewählten Zeitdiskretisierung verknüpft. Es lassen sich drei grundlegende Strategien aufzählen. Im Einzelnen ist das der stationär-implizite, der instationär-implizite und der instationär-explizite Ansatz. Bei der stationär-impliziten Simulation werden die Gleichungen des mathematischen Modells in einer nichtlinearen Iterationsschleife so lange sequentiell gelöst, bis das gewünschte Konvergenzkriterium für alle Gleichungen erfüllt ist. Die nichtlinearen Iterationen werden auch als äußere Iterationen bezeichnet, um sie von den inneren Iterationen zur Lösung der implizit diskretisierten Gleichungssysteme zu unterscheiden. Für die instationär-implizite Simulation muss zusätzlich um die äußere Iterationsschleife noch eine Zeitschleife gepackt werden. Die Zeitschleife iteriert mit der eingestellten Zeitschrittweite so lange bis die gewünschte Anzahl an Zeitschritten erreicht ist. Instationär-explizite Verfahren besitzen ebenfalls eine Zeitschleife, benötigen aber innerhalb dieser keine zusätzliche Iterationsschleife mehr, da für die Berechnung des mathematischen Modells zum aktuellen Zeitpunkt nur ein einziger Durchgang über die explizit diskretisierten Gleichungen nötig ist.

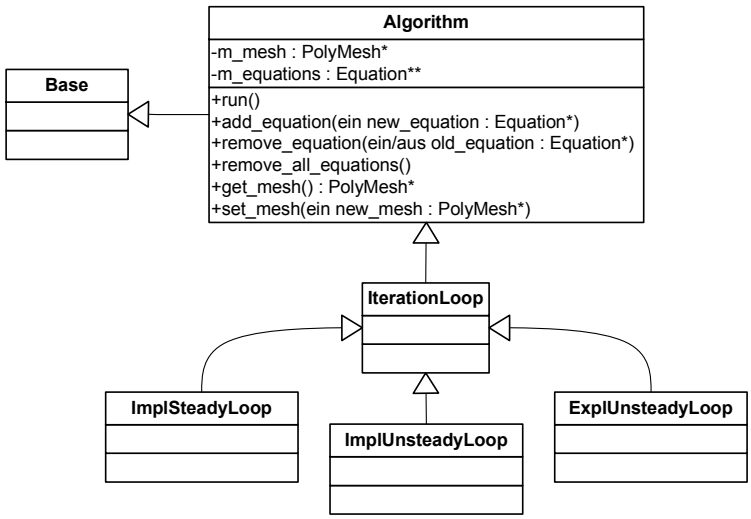
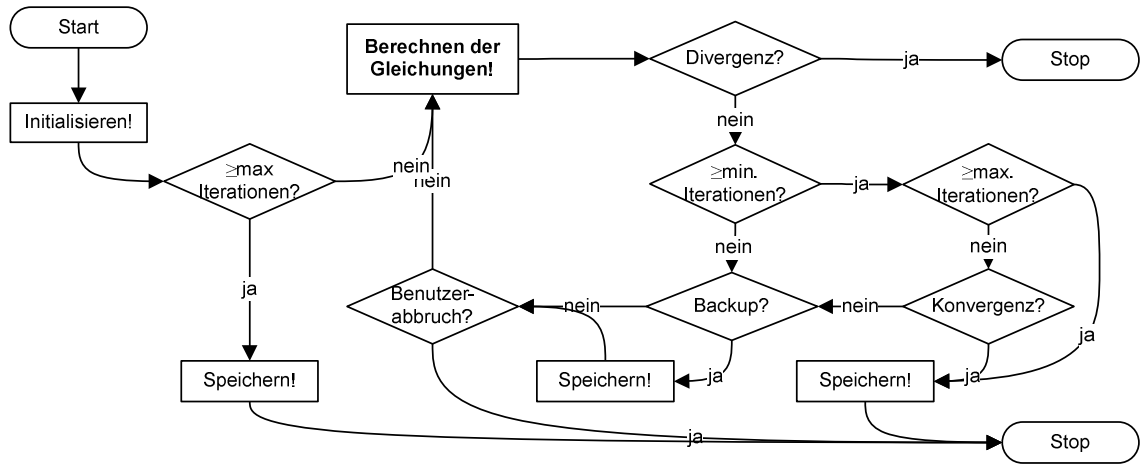
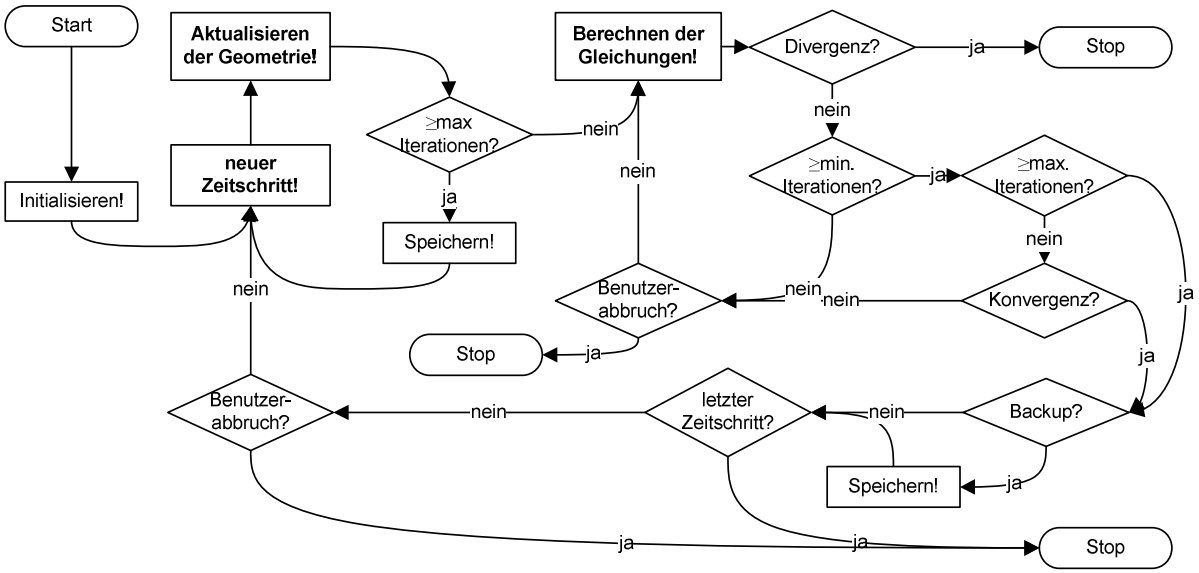


Bild 4-8: Klassenentwurf der Steueralgorithmen für die Iterationsschleifen im CFD-Tool; Zu unterscheiden ist die implizit-stationäre, die implizit-instationäre und die explizit-instationäre Schleife

Ablaufplan bei implizit-stationären Simulationen:



Ablaufplan bei implizit-instationären Simulationen:



Ablaufplan bei explizit-instationären Simulationen:

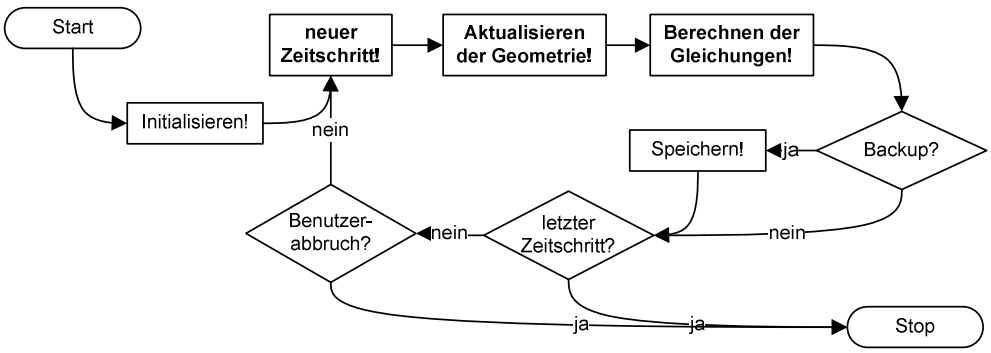


Bild 4-9: Flussdiagramme der unterschiedlichen Steueralgorithmen zur Implementierung von implizit stationären, implizit instationären und explizit instationären Verfahren

Obwohl diese drei Strategien auf den ersten Blick relativ einfach zu überblicken scheinen, ergeben sich dennoch komplexe Programmabläufe, wenn zusätzliche Steuerparameter berücksichtigt werden. Vor allem die impliziten Verfahren weisen hier schnell eine ganze Kaskade von Abfragen auf, da beispielsweise sinnvoll ist für die äußere Iterationsanzahl sowohl ein Minimum als auch ein Maximum vorzugeben oder ein Backup-Intervall einzuführen.

Die Berücksichtigung der Besonderheiten der einzelnen Verfahren in einem Gesamtalgorithmus ist zwar möglich, führt aber zu extrem unübersichtlichen und schwer wartbaren Code. Im CFD-Tool werden deshalb Algorithmenobjekte eingeführt, die die drei Strategien getrennt umsetzen und den Programmablauf jeweils passend steuern. Bild 4-8 zeigt den entsprechenden Klassenentwurf der Basisklasse *Algorithm*, von der die einzelnen Iterationsschleifen abgeleitet werden. Zum Lösen der Gleichungen müssen die Algorithmenobjekte natürlich die einzelnen Gleichungen sowie das numerische Netz kennen. Über entsprechende Zugriffsfunktionen können deshalb die Gleichungen und das Netz geladen werden. Über die Memberfunktion *run()* wird danach die Ablaufsteuerung gestartet.

Bild 4-9 zeigt die Ablaufdiagramme der drei im CFD-Tool implementierten Algorithmen für stationär-implizite, instationär-implizite und instationär-explizite Simulationen. Die Iterationen über die einzelnen Gleichungen sowie die inneren Iterationen der impliziten Verfahren sind nicht konkret ausmodelliert. Sie sind in der Aktion „*Berechnung der Gleichungen!*“ zusammengefasst.

5 Validierung am konkreten Anwendungsfall

Die in den vorhergehenden Kapiteln entwickelten Konzepte und umgesetzten Klassenentwürfe werden nun validiert. Dazu dient der klassische Modellentwicklungsprozeß, der in Kapitel 2.3 diskutiert wurde. Der Prozeß beschreibt in ausreichender Weise die Anforderungen, die die einzelnen Anwendergruppen an die CFD-Software stellen. Als konkretes und anspruchsvolles Beispiel wird die Simulation der kavitierenden Strömung in hydraulischen Maschinen ausgewählt. Zunächst erfolgt in Analogie zum beschriebenen Modellentwicklungsprozeß die Definition des physikalischen Modells. Danach schließt sich die Implementierung der Gleichungen mit den hierfür entwickelten Diskretisierungswerkzeugen an. Der letzte Teil des Kapitels zeigt schließlich an zwei ausgesuchten Testfällen die mit der verwendeten Modellierung erzielbaren Ergebnisse.

5.1 Modellbildung

5.1.1 Hydrodynamisches System

Die reibungsbehaftete Strömung eines inkompressiblen Newtonschen Fluids wird durch das Geschwindigkeitsfeld u_i sowie das statische Druckfeld p definiert. Für deren Berechnung stehen die Massenerhaltung nach Gleichung (5.1) und die Impulserhaltung nach Gleichung (5.2) zur Verfügung:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_j)}{\partial x_j} = 0, \tag{5.1}$$

$$\frac{\partial (\rho u_i)}{\partial t} + \frac{\partial (\rho u_i u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j}. \tag{5.2}$$

Der viskose Spannungstensor τ_{ij} kann mit Hilfe des Newtonschen Stoffgesetzes wie folgt modelliert werden:

$$\tau_{ij} = \mu \left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \delta_{ij} \frac{\partial u_k}{\partial x_k} \right]. \tag{5.3}$$

In Gleichung (5.3) bezeichnet δ_{ij} das Kronecker Symbol. Für inkompressible Strömungen stellen die Dichte ρ und die dynamische Viskosität μ Stoffkonstanten dar, wodurch sich in den Gleichungen (5.1)-(5.3) zunächst einige Vereinfachungen ergeben würden. Bei der Modellierung kavitierender Strömungen gilt dies allerdings nicht mehr. In der vorliegenden Arbeit wird die Implementierung eines Kavitationsmodells demonstriert, das auf der homogenen Mischungshypothese beruht. Die Strömung wird dabei als künstliches Mischfluid modelliert, dessen Einzelphasen zwar als inkompressibel betrachtet werden, dessen Stoffwerte sich aber aus der lokalen Zusammensetzung der Einzelphasen ergeben.

Die Gleichungen (5.1)-(5.3) können somit nicht weiter vereinfacht werden und stellen die Basis für die Modellierung des hydrodynamischen Systems dar. Da die inkompressible Strömung lediglich ein Spezialfall der allgemeinen Navier-Stokes Gleichungen ist, gelten obige Gleichungen natürlich auch für die Modellierung der kavitationsfreien inkompressiblen Strömung.

Mittelung der Grundgleichungen

Die direkte Lösung des hydrodynamischen Systems bedeutet im Falle der turbulenten Strömung in Turbomaschinen einen Rechenaufwand, der bis in die nahe Zukunft nicht realisierbar sein wird. Die statistische Auswirkung der Turbulenz auf die Strömung kann aber mit den Reynolds-gemittelten Navier-Stokes Gleichungen (RANS) modelliert werden. Dafür müssen die lokal an einem Ort vorherschenden Strömungsgrößen zunächst in den zeitlichen Mittelwert $\bar{\phi}$ und den Schwankungsanteil ϕ' aufgeteilt werden, siehe REYNOLDS [70]. Dieses Vorgehen ist sowohl für stationäre wie auch für instationäre Probleme zulässig, solange die globale zeitliche Änderung der Strömung wesentlich langsamer als die der turbulenten Schwankungen verläuft. Der Reynoldssche Separationsansatz laut:

$$\phi(x_i,t)=\bar{\phi}(x_i,t)+\phi'(x_i,t), \tag{5.4}$$

wobei die zeitlich gemittelte Größe $\bar{\phi}$ definiert ist als:

$$\bar{\phi}(x_i,t)=\frac{1}{\Delta t}\int_{t}^{t+\Delta t}\phi(x_i,t)dt\,. \tag{5.5}$$

Die Integrationszeit Δt in Gleichung (5.5) ist dabei so zu wählen, dass sie klein gegenüber der globalen zeitlichen Änderung, aber hinreichend groß gegenüber den turbulenten Fluktuationen ist. Aus Gleichung (5.4) ergibt sich außerdem, dass der Schwankungswert bei abermaliger Mittelung verschwindet, also $\overline{\phi'}=0$.

Der Separationsansatz wird nun auf die Erhaltungsgleichungen (5.1)-(5.3) angewendet. Da sich die Gemischdichte der kavitierenden Strömung aus den beiden inkompressiblen Einzelphasen zusammensetzt, Kapitel 5.1.3, wird vorausgesetzt, dass diese keinen turbulenten Schwankungen unterliegt und deshalb nicht in Schwankungs- und Mittelwert separiert werden muss. Die anschließende Mittelung der Erhaltungsgleichungen ergibt die Reynolds-gemittelten Gleichungen, die sowohl für das inkompressible Newtonsche Reinals auch das kavitierende Mischfluid gelten:

$$\frac{\partial\rho}{\partial t}+\frac{\partial\left(\rho\bar{u}_j\right)}{\partial x_j}=0, \tag{5.6}$$

$$\frac{\partial(\rho\overline{u}_i)}{\partial t}+\frac{\partial\left(\rho\overline{u}_i\overline{u}_j\right)}{\partial x_j}=-\frac{\partial\overline{p}}{\partial x_i}+\frac{\partial}{\partial x_j}\left\{\mu\left[\left(\frac{\partial\overline{u}_i}{\partial x_j}+\frac{\partial\overline{u}_j}{\partial x_i}\right)-\frac{2}{3}\delta_{ij}\frac{\partial\overline{u}_k}{\partial x_k}\right]-\rho\overline{u_i'u_j'}\right\}. \tag{5.7}$$

Die Gleichungen beschreiben im Gegensatz zu den originären Erhaltungsgleichungen den Transport der zeitlich gemittelten Größen $\overline{p}$ und $\overline{u}_i$. Sie enthalten zusätzlich den sogenannten Reynolds-Spannungstensor $-\rho\overline{u_i'u_j'}$, der aus der Mittelung der konvektiven Terme herrührt, siehe WILCOX [103]. Da die Komponenten des Reynolds-Spannungstensors a priori unbekannt sind, ist das Gleichungssystem nun unterbestimmt. Die Bestimmung von $-\rho\overline{u_i'u_j'}$ ist Aufgabe der Turbulenzmodellierung, siehe Kapitel 5.1.2.

5.1.2 Turbulenzmodellierung

Um das gemittelte Gleichungssystem wieder zu schliessen, müssen die einzelnen Komponenten des Reynolds-Spannungstensors bestimmt bzw. modelliert werden. Dabei ist darauf zu achten, dass die grundlegende Physik der turbulenten Strömung möglichst gut erhalten bleibt. Die hierfür verwendeten Turbulenzmodelle lassen sich nach der jeweiligen Modellierung des Reynolds-Spannungstensors in zwei Hauptgruppen aufgliedern, vergleiche SCHUSTER [85]. Bei den Reynolds-Spannungsmodellen (RSM) werden die Korrelationen des Reynolds-Spannungstensors entweder direkt durch Transportgleichungen (differentielle RSM) oder mittels algebraischer Ausdrücke (algebraische RSM) bestimmt. Bei den Wirbelviskositätsmodellen wird der Reynolds-Spannungstensor über die Wirbelviskositätsannahme nach BOUSSINESQ [13] durch die mittleren Strömungsgrößen ausgedrückt:

$$-\rho\overline{u_i'u_j'}=2\mu_T S_{ij}-\frac{2}{3}\delta_{ij}\left(\mu_T\frac{\partial\overline{u}_k}{\partial x_k}+\rho k\right)+\rho\overline{u_i'u_j'}^{HOT} \tag{5.8}$$

mit

$$k=\frac{1}{2}\overline{u_i'u_i'}\,. \tag{5.9}$$

k beschreibt die turbulente kinetische Energie und μ_T die turbulente Viskosität. Die in Gleichung (5.8) auftretenden Terme höherer Ordnung $\rho\overline{u_i'u_j'}^{HOT}$ beinhalten quadratische und kubische Approximationen des Reynolds-Spannungstensors in Abhängigkeit des Deformationstensors S_{ij} (5.10) und des Rotationstensors Ω_{ij} (5.11). Diese Terme sind im originären Boussinesq-Ansatz gleich Null. S_{ij} und Ω_{ij} definieren sich folgendermaßen:

$$S_{ij}=\frac{1}{2}\left(\frac{\partial\overline{u}_i}{\partial x_j}+\frac{\partial\overline{u}_j}{\partial x_i}\right), \tag{5.10}$$

$$\Omega_{ij} = \frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j} - \frac{\partial \overline{u}_j}{\partial x_i} \right). \tag{5.11}$$

Das Schließungsproblem verlagert sich nun auf die Bestimmung der turbulenten kinetischen Energie k und der turbulenten Viskosität μ_T. Im Rahmen dieser Arbeit wird die turbulente kavitierenden Strömung mit dem Standard-k-ε-Modell und dem nichtlinearen LCL-Modells simuliert.

Standard-k-ε-Modell

Eines der am häufigsten verwendeten Turbulenzmodelle ist das Standard-k-ε-Modell von LAUNDER und SPALDING [50]. Es beschreibt die turbulente Viskosität durch:

$$\mu_T = C_\mu f_\mu \rho \frac{k^2}{\varepsilon} \tag{5.12}$$

mit dem Proportionalitätsfaktor C_μ, der Dämpfungsfunktion f_μ, sowie der Dissipationsrate ε. Das Modell besitzt zwei Transportgleichungen für k und ε:

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial\left(\rho \overline{u}_j k\right)}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_T}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] + P_k - \rho\varepsilon\,, \tag{5.13}$$

$$\frac{\partial(\rho \varepsilon)}{\partial t} + \frac{\partial\left(\rho \overline{u}_j \varepsilon\right)}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_T}{\sigma_\varepsilon} \right) \frac{\partial \varepsilon}{\partial x_j} \right] + C_{\varepsilon 1} f_1 P_k \frac{\varepsilon}{k} - C_{\varepsilon 2} f_2 \rho \frac{\varepsilon^2}{k}, \tag{5.14}$$

wobei die Produktionsrate der turbulenten kinetischen Energie gegeben ist durch:

$$P_k = -\rho \overline{u_i' u_j'} \frac{\partial \overline{u}_i}{\partial x_j} = 2\mu_T S_{ij} S_{ij}\,. \tag{5.15}$$

Die Transportgleichung für ε ist laut FERZIGER und PERIĆ [23] als Modellgleichung anzusehen. Das Standard-k-ε-Modell ist nur für hohe Reynoldszahlen gültig. Die Konstanten des Modells sind im Anhang zu finden.

Nichtlineares Low-Reynolds k-ε-Modell (LCL)

Das nichtlinearen Low-Reynolds k-ε-Modell nach LIEN et al. [53] berücksichtigt im Gegensatz zum Standard-k-ε-Modell die Anisotropie der Turbulenz. Die in Gleichung (5.8) auftretenden Terme höherer Ordnung $\overline{\rho u_i' u_j'}^{HOT}$ werden beim LCL-Modell folgendermaßen entwickelt:

$$\begin{aligned}
\overline{\rho u_i' u_j'}^{HOT} &= 4C_1 \frac{\mu_T k}{\varepsilon}\left(S_{ik}S_{kj} - \frac{1}{3} S_{kl}S_{lk}\delta_{ij} \right) + 4C_2 \frac{\mu_T k}{\varepsilon}\left(\Omega_{ik}S_{kj} + \Omega_{jk}S_{ki}\right) \\
&+ 4C_3 \frac{\mu_T k}{\varepsilon}\left(\Omega_{ik}\Omega_{jk} - \frac{1}{3}\Omega_{kl}\Omega_{kl}\delta_{ij} \right) + 8C_4 \frac{\mu_T k^2}{\varepsilon^2}\left(S_{ki}\Omega_{lj} + S_{kj}\Omega_{li}\right)S_{kl} \\
&+ 8C_5 \frac{\mu_T k^2}{\varepsilon^2}\left(\Omega_{il}\Omega_{lm}S_{mj} + S_{il}\Omega_{lm}\Omega_{mj} - \frac{2}{3} S_{lm}\Omega_{mn}\Omega_{nl}\delta_{ij} \right) \\
&+ 8C_6 \frac{\mu_T k^2}{\varepsilon^2} S_{ij}S_{kl}S_{lk} + 8C_7 \frac{\mu_T k^2}{\varepsilon^2} S_{ij}\Omega_{kl}\Omega_{kl} .
\end{aligned} \tag{5.16}$$

Die entsprechenden Konstanten des LCL-Modells sind im Anhang zu finden.

Wandbehandlung

Die vollentwickelte turbulente Grenzschicht an einer Wand lässt sich untergliedern in eine viskose Unterschicht, den logarithmischen Bereich und eine turbulente Kernströmung. Bei Turbulenzmodellen, die bis in die viskose Unterschicht integriert werden, ist für die wandnächste Zelle ein dimensionsloser Wandabstand $y^+ = O(1)$ einzuhalten. Das dimensionslose Geschwindigkeitsprofil weist bis etwa $y^+ < 5$ einen linearen Verlauf auf, d.h. $u^+ = y^+$. Bei großen Reynolds-Zahlen ist die viskose Unterschicht allerdings sehr dünn und nur mit erheblichem Vernetzungsaufwand aufzulösen. Bei Verwendung eines Turbulenzmodells mit Wandfunktion müssen die Mittelpunkte der wandnächsten KV erst im logarithischen Bereich der Grenzschicht liegen, wo das dimensionslose Geschwindigkeitsprofil u^+ dem logarithmischen Wandgesetz folgt. Dieses lautet für hydraulisch glatte Wände:

$$u^+ = \frac{1}{0.41}\ln y^+ + 5.2 . \tag{5.17}$$

Die dimensionslosen Größen y^+ und u^+ sind wie folgt definiert:

$$y^+ = \frac{y\overline{u}_t}{\nu}, \qquad u^+ = \frac{\overline{u}_t}{u_\tau}, \qquad u_\tau = \sqrt{\frac{\tau_W}{\rho}}, \tag{5.18}$$

wobei $\overline{u}_t$ die wandtangentiale Geschwindigkeit, u_τ die Schubspannungsgeschwindigkeit und y den Wandabstand des wandnächsten Rechenpunktes bezeichnen. Die Strömung befindet sich in der logarithmischen Schicht nach DURBIN und PETTERSSON REIF [21] in einem lokalen Gleichgewicht zwischen turbulenter Produktionsrate P_k und Dissipation ε, wodurch gilt:

$$u_\tau = C_\mu^{1/4} \sqrt{k} \,. \tag{5.19}$$

Für die Dissipation in den wandnächsten Rechenpunkten ergibt sich schließlich folgender Zusammenhang:

$$\varepsilon = \frac{C_\mu^{3/4} k^{3/2}}{0.41\, y} \,. \tag{5.20}$$

Der logarithmische Bereich erstreckt sich etwa zwischen $30 < y^+ < 500$. Der dimensionslose Wandabstand ist Teil des Simulationsergebnisses und muss nach jeder Rechnung überprüft werden.

5.1.3 Kavitationsmodellierung

Wie bereits in Kapitel 5.1.1 angedeutet, wird die Zweiphasenströmung durch die homogene Mischungshypothese modelliert. Die Stoffeigenschaften des künstlichen Fluids werden dabei über die Dampfvolumenfraktion α, die das Verhältnis des Dampfvolumens Ω_v zum Gesamtvolumen Ω einer Zelle beschreibt, berechnet, siehe FROBENIUS [30]:

$$\rho = \alpha \cdot \rho_v + (1-\alpha) \cdot \rho_l \,, \tag{5.21}$$

$$\mu = \alpha \cdot \mu_v + (1-\alpha) \cdot \mu_l \tag{5.22}$$

mit

$$\alpha = \frac{\Omega_v}{\Omega} = \frac{\Omega_v}{\Omega_v + \Omega_l} \,. \tag{5.23}$$

Die tiefgestellten Zeichen *v* bzw. *l* bezeichnen jeweils die pure Dampf- bzw. Flüssigphase des Mischfluids. Die Dampfvolumenfraktion α aus Gleichung (5.22) kann Werte $0 \le \alpha \le 1$ annehmen, wobei $\alpha = 0$ die reine Flüssigkeit und $\alpha = 1$ den reinen Dampf beschreibt. Gleichung (5.21) folgt direkt aus der Massenerhaltung. Gleichung (5.22) stellt eine Annahme dar. Dabei wird einfach davon ausgegangen, dass die dynamische Viskosität des künstlichen Mischfluids linear proportional zu seiner Gemischdichte ist.

Die Modellierung des Strömungsproblems mit den grundlegenden Erhaltungsgleichungen aus den Abschnitten 5.1.1 und 5.1.2 kann durch die Formulierung als Mischfluid unverändert übernommen werden. Es muss lediglich jede Gleichung für die Mischung formu-

liert werden. Die Verteilung der Dampfvolumenfraktion α wird über eine zusätzliche Transportgleichung beschrieben:

$$\frac{\partial \alpha}{\partial t} + \frac{\partial (\alpha u_j)}{\partial x_j} = \frac{\rho_l}{\rho} \cdot \frac{d\alpha}{dt} = \frac{1}{\rho_v} \dot{m}_{trans} = q \,. \tag{5.24}$$

$\dot{m}_{trans}$ bezeichnet die Zwischenphasen-Massenstrom-Transferrate pro Einheitsvolumen und tritt als Quellterm für α auf. Mit der Annahme, dass $\rho_v / \rho_l << 1$ für die meisten Fluide gültig ist, lässt sich $\dot{m}_{trans}$ beschreiben durch:

$$\dot{m}_{trans} = \frac{\rho_v}{\alpha \cdot \frac{\rho_v}{\rho_l} + (1-\alpha)} \frac{d\alpha}{dt} \approx \frac{\rho_v}{(1-\alpha)} \frac{d\alpha}{dt} \,. \tag{5.25}$$

Mithilfe sphärischer Blasendynamik und mit der Annahme, dass der Dampf als homogen verteilte Dampfblasen im KV vorliegt, kann Gleichung (5.23) umgeschrieben werden:

$$\alpha = \frac{n_0 \frac{4}{3} \pi R^3}{1 + n_0 \frac{4}{3} \pi R^3} \,. \tag{5.26}$$

Hierin bezeichnet n_0 die Anzahl der Keime pro Einheitsvolumen Flüssigkeit bzw. die Keimkonzentration und R den Keimradius. Die Größe n_0 ist ein Modellparameter im Kavitationsmodell. Mit Gleichung (5.26) läßt sich die totale zeitliche Ableitung von α bestimmen:

$$\frac{d\alpha}{dt} = \frac{3\dot{R}}{R} \alpha (1-\alpha) \,, \tag{5.27}$$

wobei $\dot{R}$ je nach Vorzeichen die Wachstums- bzw. Kollapsgeschwindigkeit aller Einzelblasen verkörpert. Mit Gleichung (5.27) lässt sich $\dot{m}_{trans}$ aus Gleichung (5.25) neu formulieren:

$$\dot{m}_{trans} = \rho_v 3 \alpha \frac{\dot{R}}{R} \,. \tag{5.28}$$

Die Rayleigh-Plesset-Gleichung beschreibt den Wachstums- und Kollapsmechanismus einer Einzelblase. Vernachlässigt man Oberflächenspannung, Reibung und Trägheit ver-

einfacht sich die Rayleigh-Plesset-Gleichung zur Rayleigh-Gleichung. Diese wird zur Beschreibung von $\dot{R}$ herangezogen, siehe im Detail auch FROBENIUS [30]:

$$\dot{R}^2 = \frac{2}{3}\frac{p_b(T) - p}{\rho_l}\,. \tag{5.29}$$

$p_b(T)$ in Gleichung (5.29) ist der statische Druck innerhalb der Dampfblase, der in der vorliegenden Modellierung gleich dem Dampfdruck $p_v(T)$ der Flüssigkeit gesetzt wird. $p_v(T)$ ist wiederum abhängig von der statischen Temperatur des Mischfluids. Im vorliegenden Kavitationsmodell wird $p_v(T)$ nun durch den neu definierten effektiven Dampfdruck $p_{eff}(T, \mathit{quality})$ ersetzt, der zusätzlich vom Reinheitsgrad der Flüssigkeit abhängt. Im Falle einer entkeimten Flüssigkeit, die keine gelösten Gase mehr enthält, gilt $p_{eff} = p_v$. Für die Simulation der kavitierenden Strömung muss p_{eff} damit abhängig von der Temperatur und der vorherrschenden Fluidqualität bestimmt werden. Für die Kalibrierung des Kavitationsmodells sind die Messergebnisse zweier unterschiedlicher Temperaturen ausreichend. Sobald das Druckniveau des Kavitationsbeginns bestimmt worden ist, kann p_{eff} so angepasst werden, dass das numerisch bestimmte Niveau des Kavitationsbeginns mit der Messung übereinstimmt.

Zusammen mit den Gleichungen (5.28) und (5.29) erhält man schließlich den im vorgestellten Kavitationsmodell verwendeten Ausdruck für $\dot{m}_{trans}$:

$$\dot{m}_{trans} = \begin{cases} +c_{prod}\rho_v \dfrac{3\alpha}{R}\sqrt{\dfrac{2}{3}\dfrac{p_{eff} - p}{\rho_l}} & p_{eff} > p \\ -c_{dest}\rho_v \dfrac{3\alpha}{R}\sqrt{\dfrac{2}{3}\dfrac{p - p_{eff}}{\rho_l}} & p_{eff} < p\,. \end{cases} \tag{5.30}$$

Die Konstanten c_{prod} und c_{dest} sind Korrekturfaktoren, die bei blasendynamischen Kavitationsmodellen weit verbreitet sind. Für die Berechnung der kavitierenden Strömung in Turbomaschinen nehmen sie die Werte $c_{prod} = 50$ und $c_{dest} = 0.02$ an, siehe FROBENIUS [30].

Beim Aufstellen des hydrodynamischen Systems für die kavitierende Strömung muss berücksichtigt werden, dass diese auch bei inkompressibler Modellierung der Einzelphasen divergenzbehaftet ist. Betrachtet man die Kontinuitätsgleichung (5.1) in ihrer nicht konservativen Form, erhält man:

$$\frac{\partial u_i}{\partial x_i} = \left(\frac{1}{\rho_v} - \frac{1}{\rho_l}\right)\dot{m}_{trans} \approx \frac{1}{\rho_v}\dot{m}_{trans}\,, \tag{5.31}$$

wobei wieder die Annahme $\rho_v / \rho_l << 1$ unterstellt wurde. Auf der rechten Seite tritt im Gegensatz zur inkompressiblen Strömung ein zusätzlicher Term auf, der bei der Modellimplementierung des hydrodynamischen Systems berücksichtigt werden muss.

5.2 Modellimplementierung

5.2.1 Druckkorrektur mit dem SIMPLEC-Algorithmus

Für die Berechnung des Geschwindigkeitsfeldes $\overline{u}_i$ werden die Impulsgleichungen herangezogen. Zur Lösung des Druckfeldes $\overline{p}$ bleibt in druckbasierten Verfahren dann nur die Massenerhaltung über. Da diese den statischen Druck aber nicht enthält, muss durch Kombination von Impulsgleichungen und Massenerhaltung eine Bestimmungsgleichung für den statischen Druck hergeleitet werden. Einen weit verbreiteten Ansatz zur Berechnung des Druckfeldes bilden die so genannten Druckkorrekturverfahren. Einer der bekanntesten Vertreter dieser Verfahren ist der SIMPLE-Algorithmus (Semi Implizit Procedure for Pressure-Linked Equations), siehe PATANKAR [66]. In der vorliegenden Arbeit wird eine verbesserte und stabilere Variante dieses Verfahrens verwendet, der SIMPLEC-Algorithmus von VAN DOORMAL und RAITHBY [97]. Die Druckkorrekturgleichung des SIMPLEC-Algorithmus für inkompressible Strömungen lautet:

$$\frac{\partial}{\partial x_i}\left(\frac{\Omega_P}{a_P^{u_i} + \sum_{NB} a_{NB}^{u_i}} \frac{\partial p'}{\partial x_i}\right) = \frac{\partial\left(\overline{u}_i^{n*}\right)}{\partial x_i} . \tag{5.32}$$

Sie verknüpft das in den Reynolds-gemittelten Impulsgleichungen zur Iteration *n* vorläufig berechnete Geschindigkeitsfeld $\overline{u}_i^{n*}$ mit dem zu lösenden Druckkorrekturfeld p'. In Gleichung (5.32) stellt Ω_P das Volumen der Zelle *P* dar. Die Größen $a_P^{u_i}$ und $a_{NB}^{u_i}$ bilden die Matrixeinträge der diskretisierten Impulsgleichungen für die Zelle *P* sowie ihre unmittelbaren Nachbarzellen *NB*.

Da die kavitierende Strömung im Gegensatz zur inkompressiblen Strömung nicht divergenzfrei ist, muss die Druckkorrekturgleichung (5.32) modifiziert werden. Mit der Kontinuitätsgleichung in ihrer nicht konservativen Form (5.31) ergibt sich, siehe auch FROBENIUS [30]:

$$\frac{\partial}{\partial x_i}\left(\frac{\Omega_P}{a_P^{u_i} + \sum_{NB} a_{NB}^{u_i}} \frac{\partial p'}{\partial x_i}\right) = \frac{\partial\left(\overline{u}_i^{n*}\right)}{\partial x_i} - \frac{1}{\rho_v}\dot{m}_{trans} . \tag{5.33}$$

Das endgültige Geschwindigkeitsfeld $\overline{u}_i^{n}$ und das Druckfeld $\overline{p}^{n}$ erhält man durch folgende Korrekturvorschriften:

$$\bar{u}_i^n = \bar{u}_i^{n*} + u_i', \qquad \bar{p}^n = \bar{p}^{n-1} + p' \tag{5.34}$$

mit

$$u_i' = -\frac{\Omega}{a_P^{u_i} + \sum_{NB} a_{NB}^{u_i}} \frac{\partial p'}{\partial x_i}. \tag{5.35}$$

Die Berechnung des hydrodynamischen Systems beinhaltet damit die Reynolds-gemittelten Impulsgleichungen und die Druckkorrekturgleichung. Die Massenerhaltung wird nicht mehr explizit benötigt, sie ist implizit in der Druckkorrekturgleichung enthalten. Die Diskretisierung der Reynolds-gemittelten Impulsgleichungen und der Druckkorrekturgleichung des SIMPLEC-Algorithmus erfolgt analog zum Vorgehen aus Kapitel 2.1. Für die diskreten Impulsgleichungen ergibt sich dann:

$$\begin{aligned} &\frac{d\left(\rho \bar{u}_i^*\right)_c}{dt}\Omega_c + \sum_{f_c}\left(\dot{m} - \dot{m}_w\right)_f \bar{u}_{i,f}^* = \\ &= -\frac{\partial \bar{p}^{n-1}}{\partial x_i}\Omega_c + \sum_{f_c}\left(\mu\left[\left(\frac{\partial \bar{u}_i^*}{\partial x_j} + \frac{\partial \bar{u}_j^*}{\partial x_i}\right) - \frac{2}{3}\delta_{ij}\frac{\partial \bar{u}_k^*}{\partial x_k}\right] - \rho\overline{u_i' u_j'}\right)_f n_i S_f + f_i^{Kroll}\Omega_c \end{aligned} \tag{5.36}$$

mit

$$\vec{f}^{\,Kroll} = \rho\left(\vec{\omega} \times \vec{u}\right). \tag{5.37}$$

Die zusätzliche Kraft $\vec{f}_i^{\,Kroll}$ entsteht durch die Formulierung der Impulsgleichungen nach KROLL [47], bei der die absoluten Geschwindigkeiten $\bar{u}_i^*$ durch die relativen Massenströme $\dot{m} - \dot{m}_w$ konvektiert werden.

Die diskrete Druckkorrekturgleichung lautet ebenfalls unter Berücksichtigung der Relativgeschwindigkeiten:

$$\sum_f \overline{\left(\frac{\Omega_P}{a_P^{u_i} + \sum_{NB} a_{NB}^{u_i}}\right)_f \left(\frac{\partial p'}{\partial x_i} n_i\right)_f} S_f = \sum_f \left[\left(\bar{u}_i^* - w_i\right) n_i\right]_f S_f - \frac{1}{\rho_v}\dot{m}_{trans}\Omega_P. \tag{5.38}$$

Der überstrichene Term in Gleichung (5.38) stellt dabei eine lineare Interpolation auf die Zellfläche *f* dar. Bei der zellzentrierten Variablenanordnung, die im CFD-Tool verwendet wird, ist das Geschwindigkeits- und das Druckfeld durch die Diskretisierung entkoppelt, was zu unerwünschten Oszillationen im Durckfeld führen kann. Um dieser Problematik

zu entgehen, wird für die Interpolation der Geschwindigkeiten $\overline{u}_i^*$ an die Zellfläche *f* in Gleichung (5.38) ein Vorschlag von RHIE und CHOW [72] verwendet:

$$\overline{u}_{i,f}^* = \overline{\overline{u}_{i,f}^*} - \overline{\left(\frac{\Omega_P}{a_P^{u_i} + \sum_{NB} a_{NB}^{u_i}} \right)_f \left[\left(\frac{\partial p}{\partial n} \right)_f - \overline{\left(\frac{\partial p}{\partial n} \right)_f} \right]^{n-1}} \quad . \tag{5.39}$$

Die überstrichenen Terme in Gleichung (5.39) bilden dabei lineare Interpolationen an die Zellfläche *f*, entsprechend Gleichung (4.14).

Die einzelnen Terme des diskreten hydrodynamischen Systems (5.36)-(5.38) können nun leicht mit den entworfenen Schemataklassen des CFD-Tools, siehe Kapitel 4.2.2, implementiert werden. Die sequentielle Lösung der LGS erfolgt dann durch die iterativen Lösungsalgorithmen aus Kapitel 4.3.

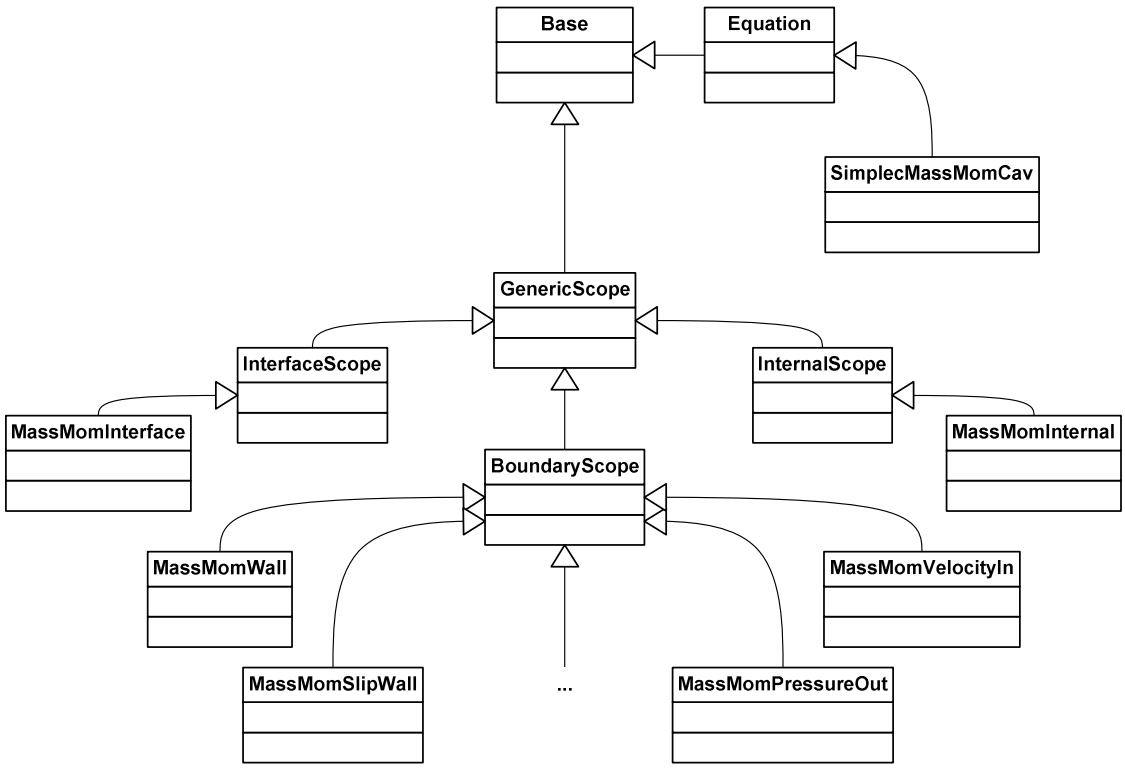


Bild 5-1: Klassenhierarchie zur Implementierung des hydrodynamischen Systems kavitierender Strömungen mit der Druckkorrektur nach dem SIMPLEC-Algorithmus

Als Kontainer für das diskrete hydrodynamische System wird die Klasse *SimplecMassMomCav* von der Klasse *Equation*, siehe Kapitel 4.2.1, abgeleitet. Die Klassen für die zugehörigen Randbedingungen werden von den jeweiligen Scope-Klassen, ebenfalls Kapitel 4.2.1, abgeleitet. Die *SimplecMassMomCav*-Klasse lädt die Randbedingungs- bzw. Scope-Klassen abhängig von der jeweiligen Problemstellung und verwendet sie zur Aus-

wahl der richtigen Diskretisierungsfunktionalität der Schemataklassen. Für die Berechnung des hydrodynamischen Systems ergibt sich insgesamt das in Bild 5-1 dargestellte Klassendiagramm. Die Scope-Klassen zeigen lediglich die zur Simulation der kavitierenden Strömung benötigten Randbedingungen. Die Formulierung der Randbedingungen findet sich in Anhang.

5.2.2 Skalare Transportgleichungen

Die Diskretisierung der einzelnen Transportgleichungen der Turbulenzmodelle sowie der Transportgleichung für die Dampfvolumenfraktion des Kavitationsmodells erfolgt in Analogie zur skalaren Transportgleichung aus Kapitel 2.1. Die einzelnen Terme der diskreten Gleichungen werden dann über die entworfenen Schemataklassen zur Diskretisierung der skalaren Terme berücksichtigt. Zur Lösung des resultierenden LGS werden die iterativen Lösungsalgorithmen aus Kapitel 4.3 verwendet.

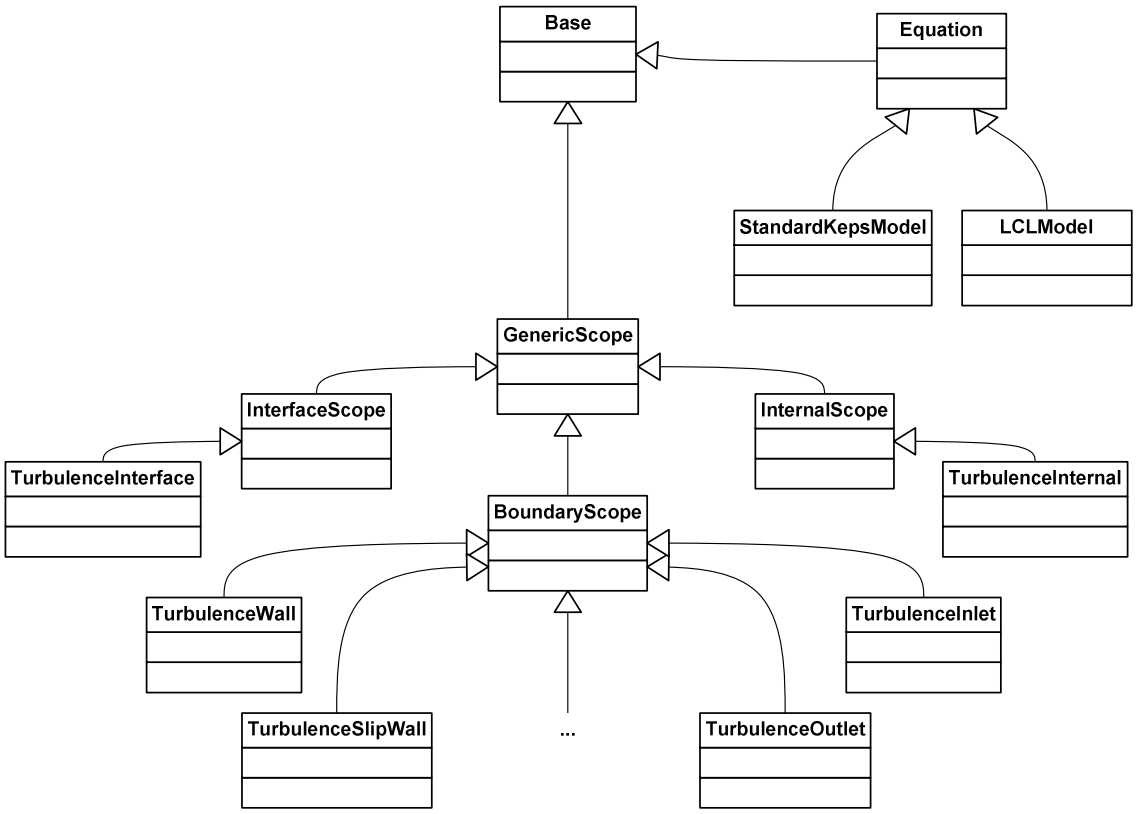


Bild 5-2: Klassenhierarchie zur Implementierung des Standard-k-ε-Modells und des LCL-Turbulenzmodells

Als Kontainer der diskreten Transportgleichungen für die turbulente kinetische Energie k und die Dissipation ε dienen die Klassen *StandardKepsModel* und *LCLModel*. Die beiden Klassen sind von der Basisklasse *Equation*, siehe Kapitel 4.2.1, abgeleitet. Die Berücksichtigung der einzelnen Randbedingungen der Turbulenzmodelle gelingt wie im hydrodynamischen System durch Programmierung der entsprechenden Scope-Klassen, die von den zugehörigen Basisklassen *InternalScope*, *InterfaceScope* und *BoundaryScope* abgeleitet wer-

den. Die Formulierung der Randbedingungen findet sich im Anhang. Insgesamt ergibt sich die in Bild 5-2 dargestellte Klassenhierarchie.

Als Kontainer für die Transportgleichnung der Dampfvolumenfraktion des Kavitationsmodells wird die Klasse *VolumeFractionEquation* erstellt. Für die Randbedingungen der Kavitation werden entsprechende Scopeklassen abgeleitet, mit denen die *VolumeFraction-Equation*-Klasse die notwendigen Randbedingungen für die Kavitation berücksichtigen kann, siehe Anhang. Insgesamt ergibt sich die in Bild 5-3 dargestellte Klassenhierarchie.

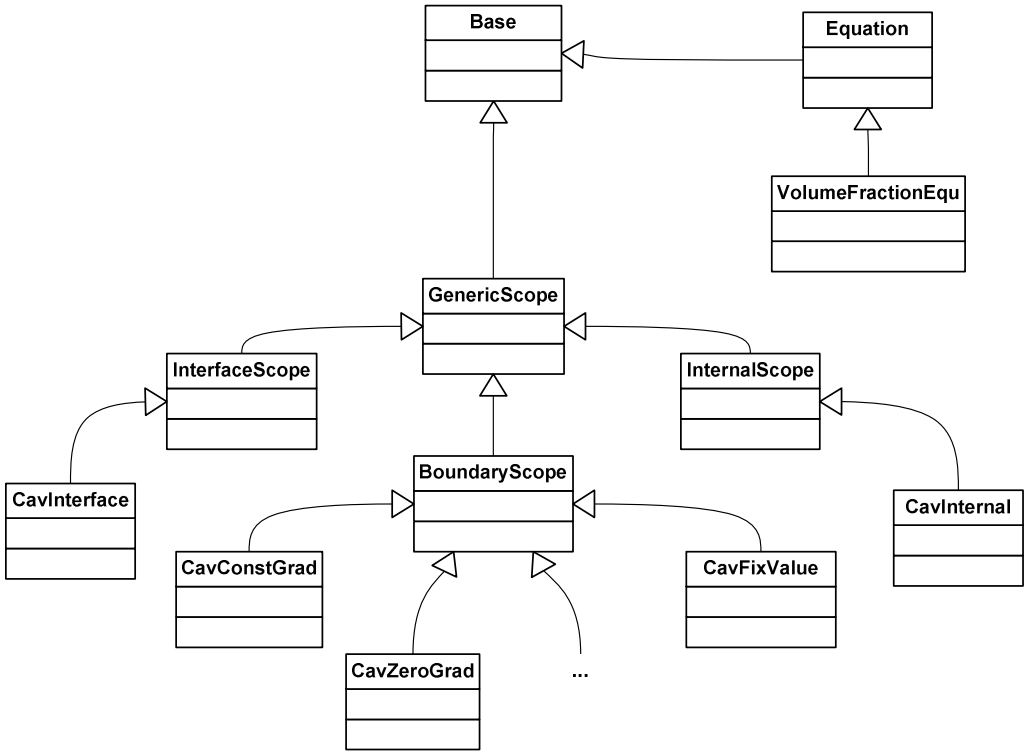


Bild 5-3: Klassenhierarchie zur Implementierung der Transportgleichung für den Dampfvolumenanteil des Kavitationsmodells

5.3 Simulation der kavitierenden Strömung in Turbomaschinen

Die Simulation kavitierender Strömungen ist eine herausfordernde Aufgabe für den Entwicklungsingenieur hydraulischer Strömungsmaschinen. Das Auftreten von Kavitation beeinflusst sowohl das akustische Verhalten als auch den Wirkungsgrad von Strömungsmaschinen negativ. Im Falle von Kavitationserosion wird darüber hinaus sogar die Lebensdauer der betroffenen Pumpen oder Turbinen beeinträchtigt. Mithilfe eines möglichst allgemeingültigen und gleichzeitig robusten Kavitationsmodells kann das Auftreten von Kavitation bereits während der Entwicklungsphase erkannt und analysiet werden. Der Entwicklungsingenieur hat infolgedessen die Möglichkeit, die negativen Auswirkungen der Kavitation durch geeignete Maßnahmen entweder zu reduzieren oder bestenfalls sogar ganz zu vermeiden. Durch die Kavitationssimulation kann der Betriebsbereich einer Strömungsmaschine von vornherein entsprechend eingeschränkt, oder die Laufradgeometrie auf ihr Kavitationsverhalten hin optimiert werden.

In den folgenden beiden Teilabschnitten wird die Funktionalität des in Kapitel 5.1.3 vorgestellten Kavitationsmodells demonstriert. Das Kaviationsmodell ist in mehreren Forschungsprojekten und im Rahmen dieser Arbeit am Lehrstuhl für Fluidmechanik (FLM) der Technische Universität München speziell für die stationäre Berechnung des Förderhöhenabfalls von hydraulischen Strömungsmaschinen entwickelt worden. Es kann zur Berechnung der Kavitation sowohl in anderen Fördermedien als Wasser als auch in unterschiedlichen Temperaturbereichen herangezogen werden.

5.3.1 Numerische Ergebnisse Heizungsumwälzpumpe – Wasser

Die Demonstration des Kavitationsmodells erfolgt an zwei industrierelevanten Testfällen. Als erstes dient hierfür der Prototyp einer Heizungsumwälzpumpe. Über die vom Hersteller angegebenen Kenndaten hinaus, ist die Pumpe am Lehrstuhl für Fluidsystemtechnik in Darmstadt (FST) insbesondere hinsichtlich ihres temperaturabhängigen Kavitationsverhaltens experimentell in unterschiedlichen Betriebspunkten untersucht worden. Der Versuchsstand ist dabei stets mit Wasser befüllt gewesen.

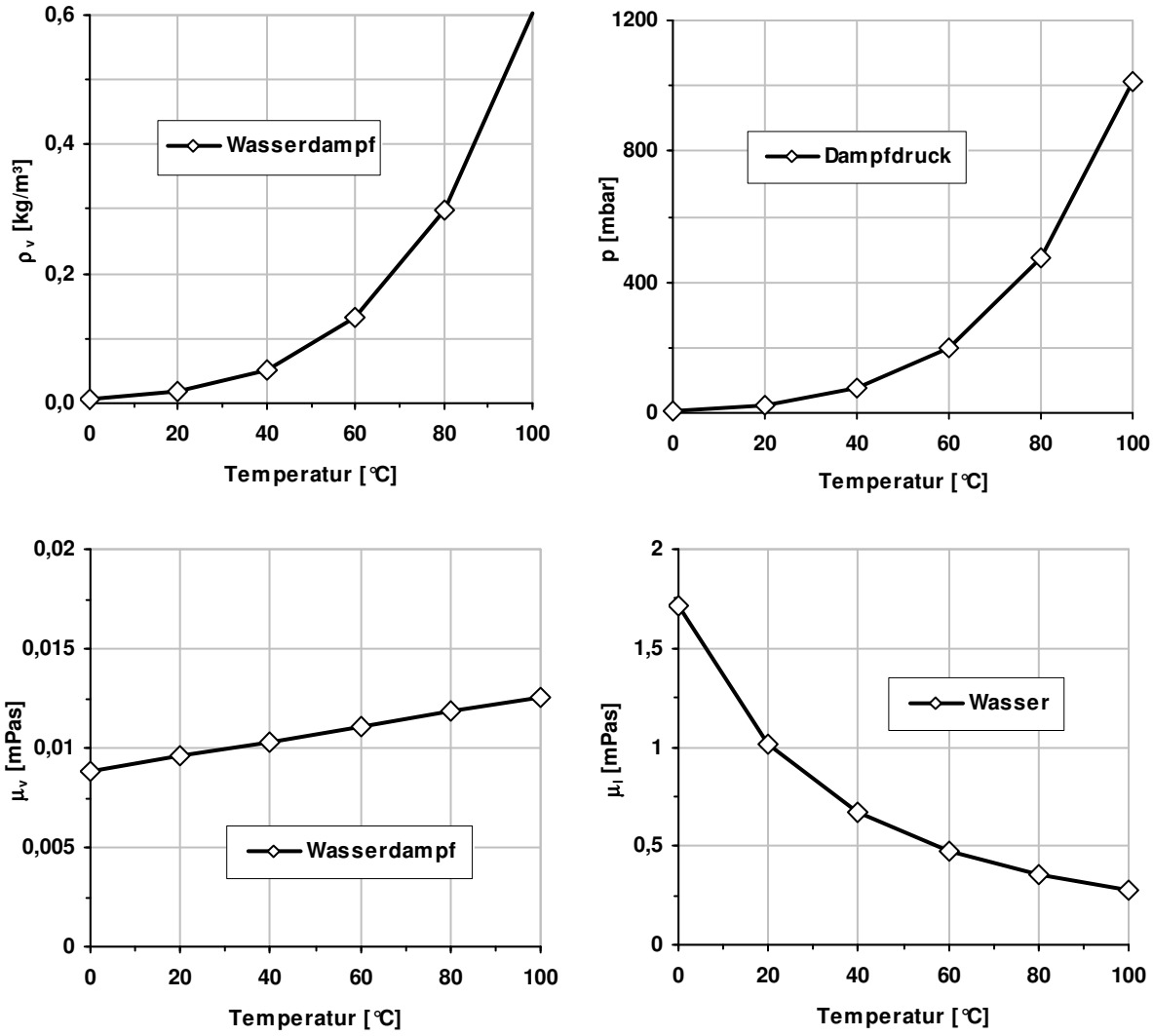


Bild 5-4: Stoffwerte von Wasser und Wasserdampf in Abhängigkeit von der Temperatur

Bild 5-4 zeigt die temperaturabhängigen Stoffdaten von Wasser sowie von gesättigtem Wasserdampf. Die Dichte ρ_v des Wasserdampfes steigt von $0.017 kg/m^3$ bei 25 °C auf $0.6 kg/m^3$ bei 100 °C stark an. Im Gegensatz dazu verhält sich die Dichte ρ_l des Wassers kaum temperaturabhängig. Sie beträgt $1000\ kg/m^3$ bei 25 °C und $950\ kg/m^3$ bei 100 °C. Das Dichteverhältnis der beiden Phasen liegt im betrachteten Bereich damit ungefähr bei 3-4 Zehnerpotenzen. Der Dampfdruck p_v steigt von $0.03 bar$ bei 25 °C auf $1 bar$ bei 100 °C exponentiell an. Die dynamische Viskosität μ_l des Wassers sinkt mit zunehmender Temperatur ab. Sie liegt für 25 °C bei $1 mPas$ und für 100 °C bei $0.28 mPas$. Die dynamische Viskosität μ_v des Wasserdampfes verhält sich umgekehrt. Sie steigt mit zunehmender Temperatur leicht an und liegt für 25 °C bei $0.01 mPas$ bzw. für 100 °C bei $0.0125 mPas$.

Der Meridianschnitt der Gehäuse- und Laufradgeometrie der untersuchten Heizungsumwälzpumpe ist in Bild 5-5 dargestellt. Das Laufrad besitzt 7 Laufschaufeln. Der Laufraddurchmesser beträgt $D = 66.5 mm$. Für eine Nenndrehzahl von $n = 2445\ 1/\min$ befindet sich der Auslegungspunkt laut Hersteller bei einem Volumenstrom von $Q_{opt} = 2.5\ m^3/h$.

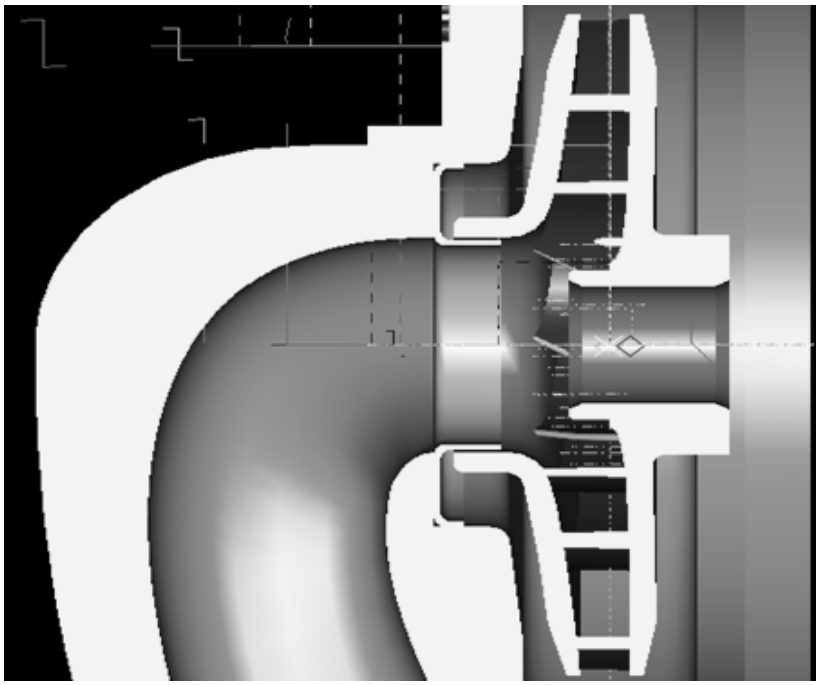


Bild 5-5: Meridianschnitt durch die untersuchte Heizungsumwälzpumpe

Bei der Vernetzung der Heizungspumpe wird sowohl der saug- als auch der druckseitige Radseitenraum berücksichtigt. Gerade der saugseitige Radseitenraum hat einen starken Einfluss auf das Kavitationsverhalten von Kreiselpumpen, siehe FROBENIUS [30], sowie SCHILLING und BOGNER [79]. Aufgrund der großen Unterschiede zwischen Gehäuse- und Laufradbreite wird außerdem die Spirale in die Vernetzung mit aufgenommen. Der Einlaufkrümmer der Pumpengeometrie wird dagegen vernachlässigt. Um den Einfluss der Einlassrandbedingung so gering wie möglich zu halten, wird allerdings

ein gerader Einlauf an die bestehende Geometrie der Pumpe anmodelliert. Die Vernetzung der Geometrie erfolgt strukturiert. Dieser zusätzliche Aufwand wird in Kauf genommen, da strukturierte Netze im Gegensatz zu unstrukturierten Netzen eine bessere Diskretisierung sicherstellen können, siehe Kapitel 3.1. Alle CFD-Simulationen werden mit dem vorgestellten Kavitationsmodell, das speziell für stationäre Simulationen ausgelegt ist, durchgeführt. Dies macht wegen der Berücksichtigung der Spirale eine Frozen-Rotor-Modellierung erforderlich, bei der die Strömung im stationären Relativsystem des eingefrorenen Laufrades berechnet wird. Der Einfluss der Spirale auf die Strömung innerhalb der einzelnen Kanäle des Laufrades muss hierbei allerdings durch verschiedene relative Laufradpositionen ausgemittelt werden.

Bereits im Vorfeld der Simulationen werden die verhältnismäßig geringen Abmessungen der Heizungsumwälzpumpe berücksichtigt. So führt die Vernetzung mit einer mittleren Knotenanzahl bereits zu relativ kleinen y^+-Werten. Trotzdem verlangt die Turbulenzmodellierung mit einem Low-Reynolds-Ansatz bei der Viskosität von Wasser aber eine unverhältnismäßig feine Netzauflösung, die einen erheblichen Rechenaufwand darstellt. Deshalb wird für die Simulation des Kavitationsverhaltens der Heizungsumwälzpumpe ein Turbulenzmodell mit Wandfunktion gewählt, das zu einer vergleichsweise groben Vernetzung führt. Gerade durch die Berücksichtigung der Spirale würde die Rechengeometrie für den anderen Fall der Low-Renolds-Modellierung und ausreichend feiner Vernetzung wesentlich unhandlicher werden. Ein Novum stellt die Herangehensweise dar, die Auswirkung der Kavitation auf die integralen Kenndaten der Pumpe mit der relativ groben Vernetzung korrekt abzubilden. Für die Turbulenzmodellierung wird das in Kapitel 5.1.2 vorgestellte Standard-k-ε-Modell verwendet. Zur Umgehung der Staupunkt Anomalie kommt dabei zusätzlich die Realizability-Bedingung nach Moore & Moore zum Einsatz, siehe auch SKODA [86]. Der Turbulenzgrad in der Eintrittsebene wird für die gesamte Untersuchung mit $Tu = 2\%$ angegeben. Damit können die Werte für k und ε aus der jeweiligen Zuströmgeschwindigkeit am Eintritt berechnet werden, siehe Anhang.

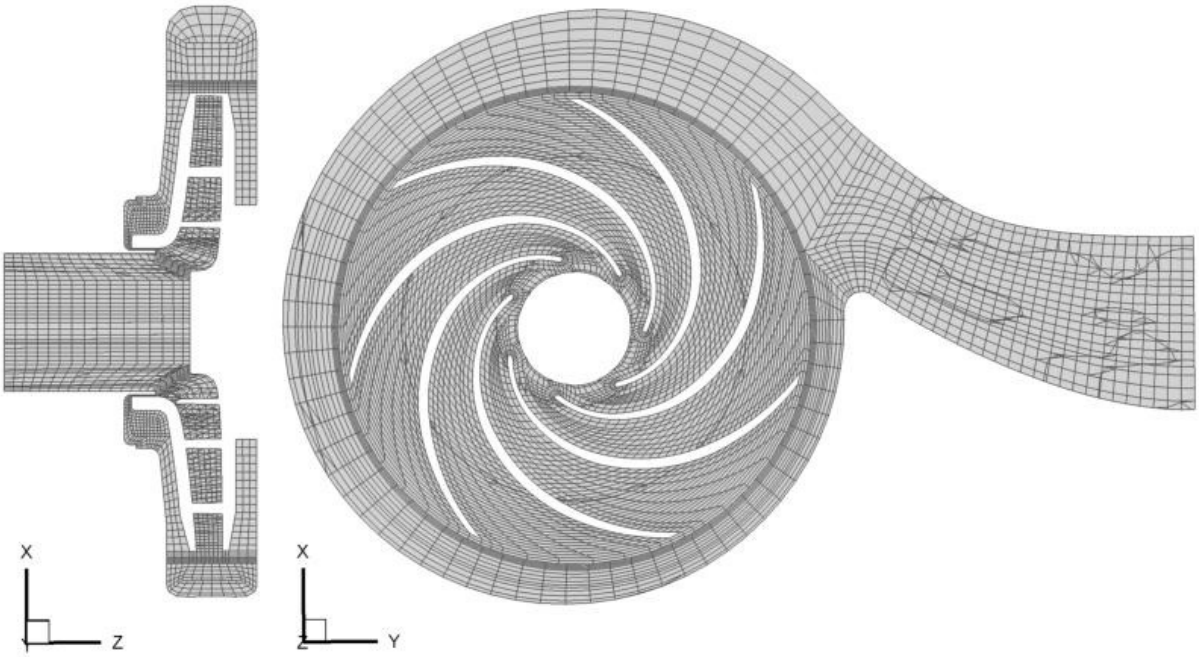


Bild 5-6: Schnitte durch das Rechennetz der Heizungspumpe; links: Seitenansicht; rechts: Draufsicht

Zwei Schnitte durch die Seitenansicht und die Draufsicht des für die Turbulenzmodellierung mit Wandfunktion erzeugten Rechennetzes sind in Bild 5-6 abgebildet. In beiden Ansichten ist die relativ grobe Netzauflösung gut zu erkennen. Vor allem die Seitenansicht zeigt deutlich die erforderliche Reduktion der Zellschichtdicke auf wenige Zellen in den Radseitenräumen. In Bild 5-7 ist die 3-dimensionale Ansicht des Laufradnetzes im Detail dargestellt. Die Vernetzung erfolgte jeweils durch einen C-Block um die Laufschaufeln, kanalweisen Koppelblöcken sowie einlass- und auslassseitigen Ringblöcken. Das gesamte Rechennetz für die numerischen Untersuchungen an der Heizungsumwälzpumpe umfasst lediglich etwa 126000 Knoten mit zirka 15000 Knoten pro Kanal.

Bild 5-7: C-Netz-Strategie zur Vernetzung der Schaufelkänalen im Laufrad der Heizungsumwälzpumpe

Bild 5-8 zeigt die y⁺-Verteilung einer Simulation mit dem Volumenstrom $Q = 2.5\, m^3/h$ und der Drehzahl $n = 2223\ 1/\min$. Etwa 75% aller Zellen liegen im optimalen Bereich von $30 < y^+ < 500$. Weitere 22% der Zellen liegen im Bereich $y^+ > 12$. Weniger als 3% der Zellen liegen in einem y⁺-Bereich $y^+ < 12$, diese Zellen befinden sich fast alle im vorderen Radseitenraum. Insgesamt sind über 97% aller Zellen im noch zulässigen y⁺-Bereich $12 < y^+ < 500$.

Aufgrund der Frozen-Rotor-Modellierung werden alle Rechnungen grundsätzlich jeweils für 4 Laufradstellungen durchgeführt und die Ergebnisse der einzelnen Rechnungen danach über die Laufradstellungen gemittelt. Bei 7 Schaufeln ergibt sich ein Teilungswinkel $t = 51.4°$. Bei 4 zu berechnenden Schaufelstellungen muss das Laufrad pro Rechnung somit relativ zur vorherigen Winkelposition jeweils um 12.86° weitergedreht werden. Die fünfte Drehung führt dann wieder zur Überdeckung der Schaufeln.

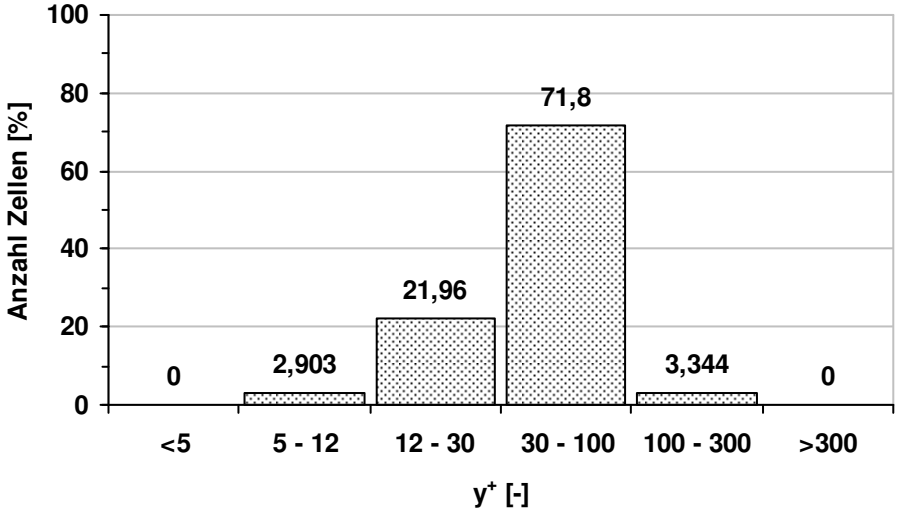


Bild 5-8: y^+-Verteilung für das verwendete Rechennetz; Q = 2.5 m^3/h; n = 2223 1/*min*

Um die Qualität der CFD-Simulationen mit dem verwendeten Rechennetz und bei der verwendeten Modellierung grundsätzlich zu überprüfen, wird das Betriebsverhalten der Heizungspumpe zunächst kavitationsfrei untersucht. Bild 5-9 zeigt die Einzelergebnisse der Frozen-Rotor-Modellierung und die Mittelwerte des berechneten Laufradwirkungsgrades der Heizungspumpe bei $n = 2445\ 1/\min$ und $Q = 2.5\ m^3/h$. Es ist ein erheblicher Einfluss der Diskretisierung der konvektiven Flüsse auf den errechneten Wirkungsgrad zu erkennen. Das liegt vor allem am relativ groben Rechennetz, wodurch im Falle der Upwind-Diskretisierung eine merkliche numerische Diffusion entsteht. Durch Beimischen eines höherwertigen Diskretisierungsschemas, wie dem MINMOD-Schema, siehe Kapitel 4.2.2, kann die numerische Diffusion stark reduziert werden. Bei der Berechnung der Drossel- und Förderhöhenabfallkurven stellte sich ein Blendingfaktor von $\beta = 0.5$ als gute Wahl heraus.

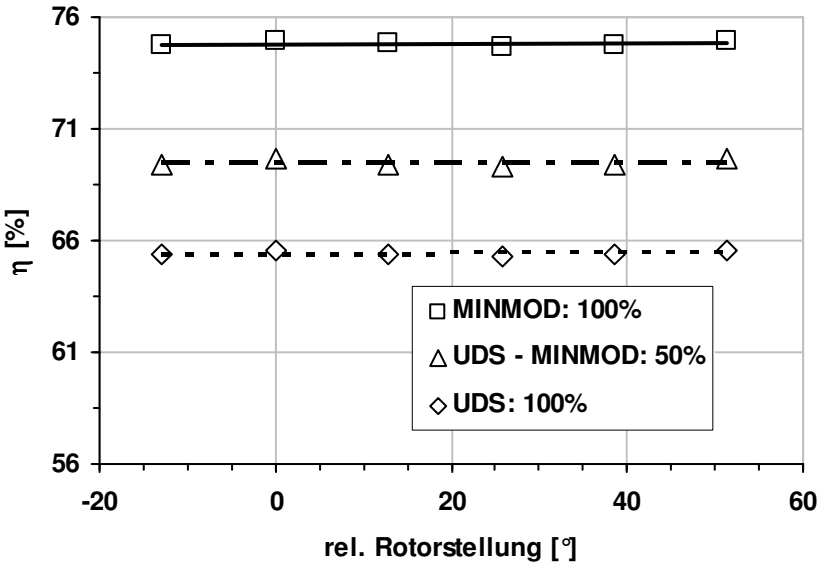


Bild 5-9: berechneter Laufradwirkungsgrad über der Rotorstellung bei der Frozen-Rotor-Modellierung und unterschiedlicher Diskretisierung der konvektiven Flüsse; Q = 2.5 m^3/h; n = 2445 1/*min*

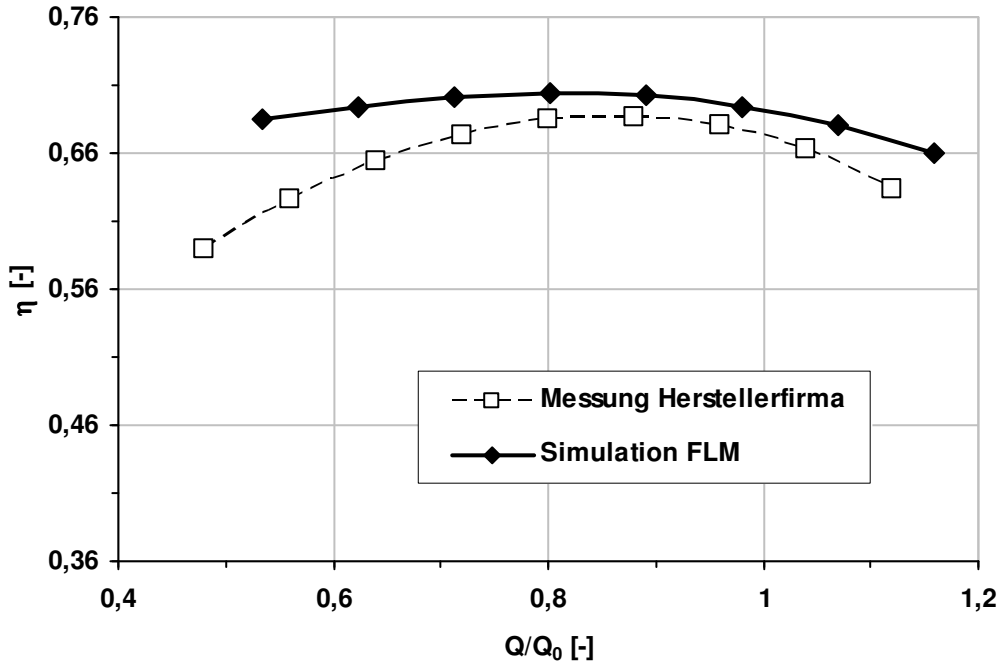


Bild 5-10: berechneter Laufradwirkungsgrad bei MINMOD-Diskretisierung der konvektiven Flüsse mit Blendingfaktor β = 0,5 und vom Hersteller angegebener hydraulischer Wirkungsgrad der Heizungspumpe; n = 2445 1/*min*

In Bild 5-10 ist die berechnete Kennlinie des Laufradwirkungsgrades der Heizungsumwälzpume für $n = 2445\ 1/\text{min}$ zusammen mit dem messtechnisch erfassten hydraulischen Wirkungsgrad nach Herstellerangaben abgebildet. Vor allem im Teil- und Überlastbereich stellt sich eine merkliche Differenz zwischen den beiden Kennlinien ein, die auf die Vernachlässigung der Spirale und der Radseitenräume bei der Berechnung des Laufradwirkungsgrades zurückzuführen ist. Der Verlauf der Kennlinie des Laufradwirkungsgrades liegt erwartungsgemäß über dem von der Herstellerfirma angegebenen hydraulischen Wirkungsgrad.

Bild 5-11 stellt zwei numerisch berechnete Drossellinien der Heizungsumwälzpumpe den unabhängig voneinander erzeugten experimentellen Daten der Herstellerfirma bzw. des FST Darmstadt gegenüber. Die berechneten Punkte zeigen eine hervorragende Übereinstimmung mit den korrellierten Messwerten. Die CFD-Simulationen wurden jeweils für eine Drehzahl von $n = 2445\ 1/\text{min}$ bzw. $n = 2223\ 1/\text{min}$ durchgeführt. Für die weiterführende detaillierte numerische Analyse des Kavitationsverhaltens der Heizungsumwälzpumpe, insbesondere für den Vergleich der einzelnen Simulationsergebnisse mit den zugehörigen kavitationsbezogenen Messdaten aus Darmstadt, wird ab jetzt nur noch eine Pumpendrehzahl von $n = 2223\ 1/\text{min}$ betrachtet.

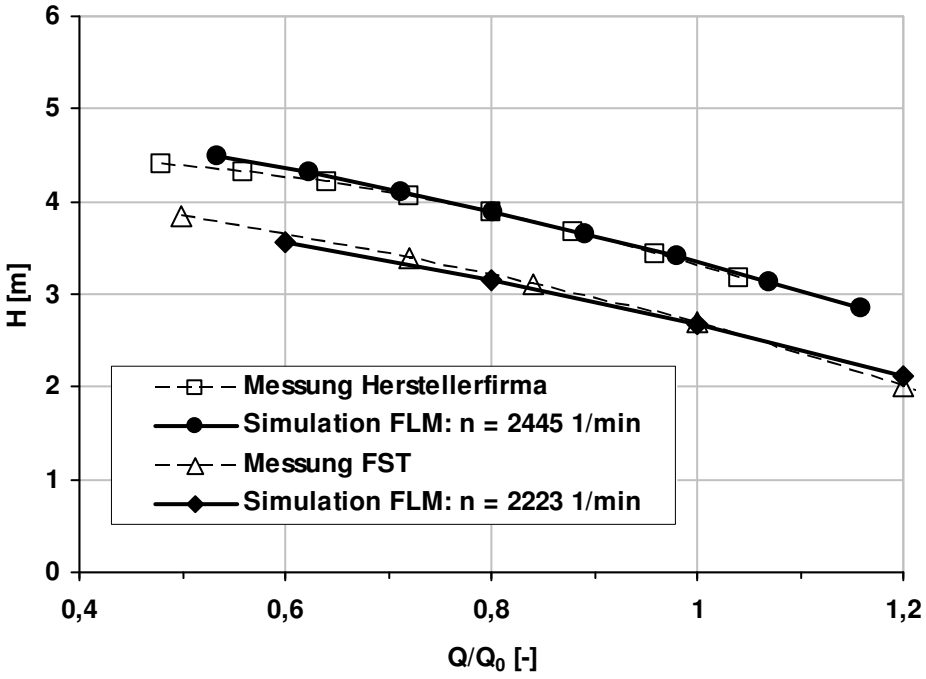


Bild 5-11: simulierte und gemessene Drossellinien der Heizungspumpe; *n* = 2445 1/*min* und *n* = 2223 1/*min*

Im AiF-Forschungsvorhaben „Kavitierende Strömungen in anderen Flüssigkeiten als Wasser“ [78], das vom FLM in Zusammenarbeit mit dem FST durchgeführt worden ist, ist das in der vorliegenden Arbeit verwendete blasendynamische Kavitationsmodell auf kalte kavitierende Pumpenströmungen kalibriert worden, siehe auch Kapitel 5.3.2. Auf den folgenden Seiten wird die Anwendbarkeit des Kavitationsmodells darüber hinaus für kavitierende Pumpenströmungen in unterschiedlichen Temperaturbereichen demonstriert. Die zur Kalibrierung des Modells notwendigen Messungen der $NPSH_{3\%}$-Kennlinien der Heizungspumpe wurden ausnahmslos vom FST Darmstadt durchgeführt. Die Messungen, bei denen als Fördermedium Wasser verwendet wurde, umfassen mit 25 °C Raumtemperatur, 85 °C und 100 °C drei unterschiedliche Temperaturniveaus.

Der $NPSH_{3\%}$-Wert einer Pumpe beschreibt den NPSH-Wert, an dem gerade 3% Förderhöhenverlust durch Kavitation eingetreten sind. Der NPSH-Wert bzw. die Förderhöhe sind wie folgt definiert:

$$NPSH = \frac{p_{t,in} - p_v}{\rho_l g}, \tag{5.40}$$

bzw.

$$H = \frac{p_{t,out} - p_{t,in}}{\rho_l g}, \tag{5.41}$$

wobei $p_{t,in}$ bzw. $p_{t,out}$ den Totaldruck am Kontrollvolumenein- bzw. –austritt darstellen.

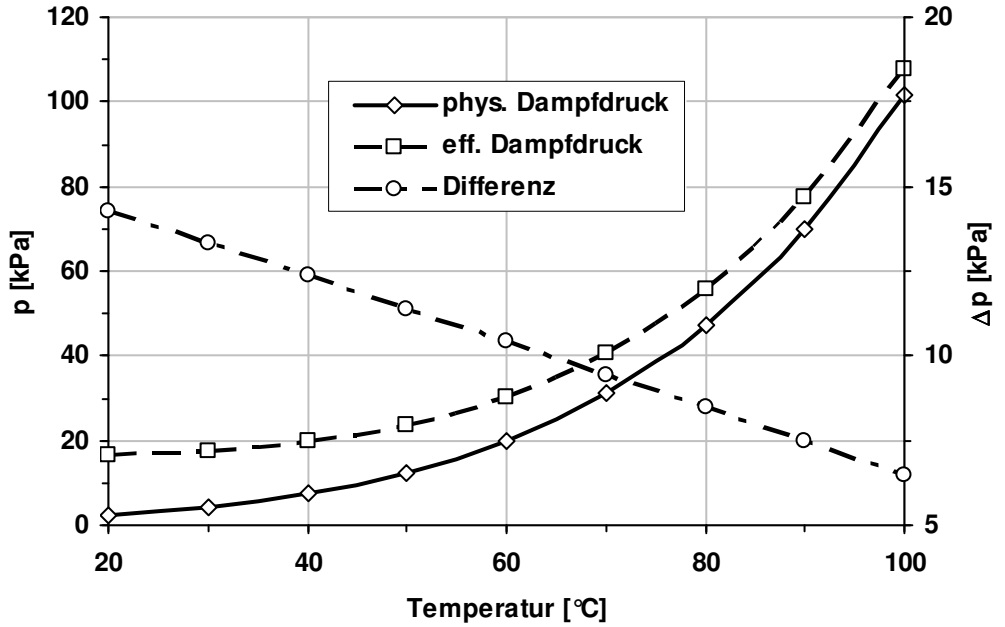


Bild 5-12: Vergleich zwischen physikalischem und bei der Kalibrierung des Kavitationsmodells bestimmten effektivem Dampfdruck, sowie die Differenz der beiden Drücke

Die Stoffwerte eines Fluids bleiben im Falle unterschiedlicher Temperaturniveaus im Allgemeinen nicht konstant. Aus Bild 5-4 kann der für die Simulation verwendete temperaturabhängige Verlauf der Stoffgrößen von Wasser entnommen werden. Die beiden Parameter Keimdichte und initiale Volumenfraktion des blasendynamischen Kavitationsmodells werden auf den standardmäßig für Wasser voreingestellten Werten belassen. Für die Keimdichte werden $n_0 = 10^8 \; Keime/m^3 \, Flüssigkeit$ und für die initiale Volumenfraktion $\alpha = 10^{-6} \; m^3 Dampf \,/ m^3 \, Gesamtvolumen$ angegeben.

Zur Bestimmung der $NPSH_{3\%}$-Werte werden die zugehörigen Förderhöhenabfallkurven für jedes Temperaturniveau berechnet. Ausgangspunkt ist dabei immer eine kavitationsfreie Rechnung, bei der das Druckminimum im Strömungsfeld über dem Dampfdruck des Fluids liegt. Danach wird das Druckniveau am Auslass schrittweise abgesenkt, bis der Steilabfall der Kurven erreicht ist. Für jede Rechnung müssen dabei der aktuelle NPSH-Wert und die aktuelle Förderhöhe ausgewertet und zu einer zusammenhängenden Förderhöhenabfallkurve vereinigt werden.

Bild 5-12 vergleicht den physikalischen Dampfdruck von Wasser mit dem bei der Kalibrierung des Kavitationsmodells mithilfe von Messdaten bestimmten effektiven Dampfdruck in Abhängigkeit von der Temperatur. Die Analyse der sich ergebenden Differenz zwischen den beiden Dampfdrücken liefert für den betrachteten Bereich zwischen 25 °C und 100 °C eine lineare Abhängigkeit von der Temperatur, sodass zur Kalibrierung des Kavitationsmodells lediglich zwei Messwerte ausreichen. In Bild 5-13 ist der mit dem so kalibrierten Kavitationsmodell berechnete Förderhöhenabfall für die drei untersuchten Temperaturniveaus bei der Drehzahl $n = 2223 \; 1/\text{min}$ und dem Volumenstrom $Q = 2.5 \, m^3/h$ dargestellt. Der Einfluss der Temperatur auf das Kavitationsverhalten der Pumpe ist deutlich an der Verschiebung des Bereiches des Steilabfalls der Förderhöhenabfallkurven erkennbar. Für die Darstellung des temperaturabhängigen

Kavitationsverhaltens der Heizungsumwälzpumpe wurden Frozen-Rotor-Rechnungen auf insgesamt 35 unterschiedlichen Druckniveaus durchgeführt.

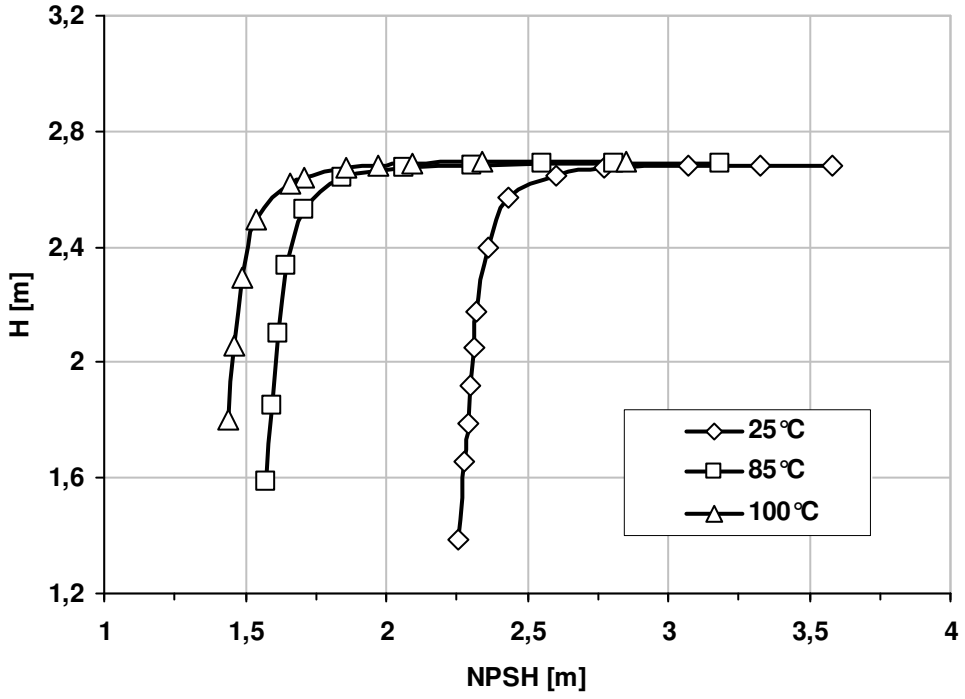


Bild 5-13: berechneter Förderhöhenabfall bei verschiedenen Temperaturen; Q = 2.5 m^3/h; n = 2223 1/*min*

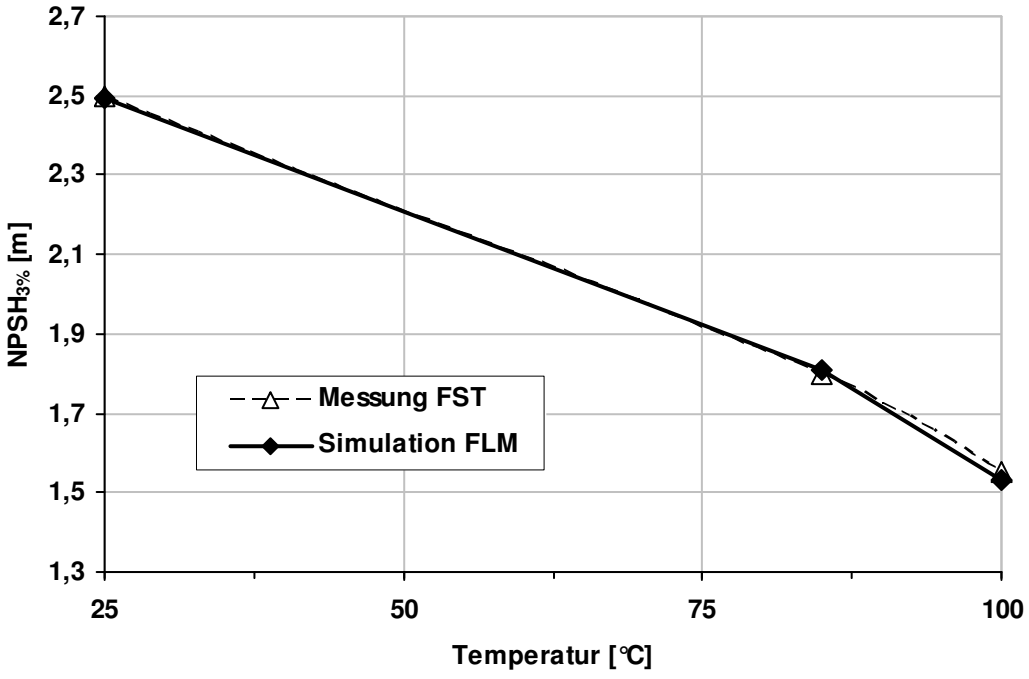


Bild 5-14: resultierende $NPSH_{3\%}$-Kennwerte in Abhängigkeit von der Temperatur; Q = 2.5 m^3/h; n = 2223 1/*min*

Bild 5-14 vergleicht die $NPSH_{3\%}$-Werte der drei Förderhöhenabfallkurven mit den am FST gemessenen Werten. Hier wird der Verlauf des effektiven Dampfdrucks über eine linear mit der Temperatur fallende Differenz zum physikalischen Dampfdruck eindeutig bestätigt. Die kalibrierten Rechenergebnisse zeigen eine hervorragend gute Übereinstimmung mit der Messung. Die $NPSH_{3\%}$-Punkte liegen sowohl bei der Simulation als auch bei der Messung für 25 °C bei $NPSH = 2.5\,m$, für 85 °C bei $NPSH = 1.8\,m$ und für 100 °C bei $NPSH = 1.55\,m$.

Die Analyse des fördermengenabhängigen Kavitationsverhaltens der Heizungsumwälzpumpe erfolgt bei einer Temperatur von 25 °C. Ausgehend vom gewählten Nennvolumenstrom $Q_0 = 2.5\,m^3/h$, wird der Förderhöhenabfall für drei weitere Volumenströme, $0.8 \cdot Q_0 = 2.0\,m^3/h$, $0.6 \cdot Q_0 = 1.5\,m^3/h$ sowie $1.2 \cdot Q_0 = 3.0\,m^3/h$, berechnet. Bild 5-15 zeigt die resultierenden Förderhöhenabfallkurven für die vier spezifizierten Volumenströme. Den Vergleich mit der Messung liefert jeweils die Auswertung des sich einstellenden $NPSH_{3\%}$-Wertes, der über der Fördermenge aufgetragen wird. Bild 5-16 stellt die am FST gemessene der berechneten $NPSH_{3\%}$-Kennlinie gegenüber. Insgesamt ist eine hervorragend gute Übereinstimmung zwischen Simulation und Messung zu erkennen. Für die Bestimmung der $NPSH_{3\%}$-Kennlinie aus den vier Förderhöhenabfallkurven wurden Frozen-Rotor-Simulationen auf insgesamt 63 unterschiedlichen Druckniveaus herangezogen.

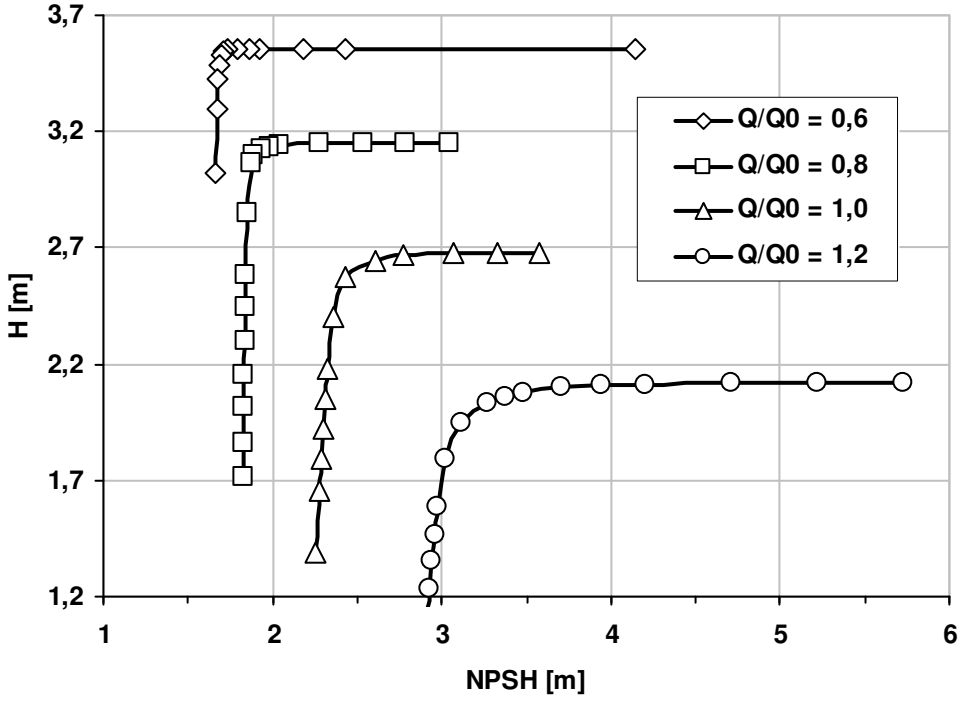


Bild 5-15: berechneter Förderhöhenabfall bei 25 °C Raumtemperatur für unterschiedliche Volumenströme; *n* = 2223 1/*min*

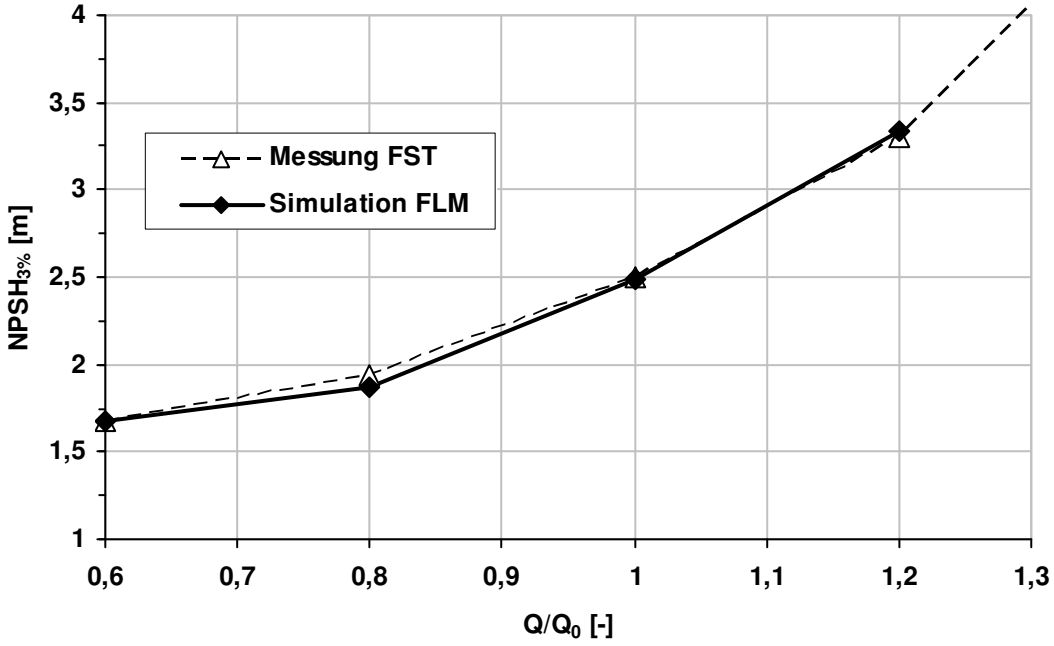


Bild 5-16: resultierende $NPSH_{3\%}$-Kennlinie bei 25 °C Raumtemperatur; n = 2223 1/*min*

Im Folgenden wird anhand der Simulationsergebnisse eine detaillierte Analyse des Förderhöhenabfalls und der Ausprägung der Kavitationsgebiete abhängig vom jeweiligen Druckniveau für den gewählten Nennvolumenstrom $Q_0 = 2.5\, m^3/h$ und die Drehzahl $n = 2223\ 1/\mathrm{min}$ durchgeführt. Bild 5-17 zeigt die mit der kavitationsfreien Lösung normierte relative Förderhöhenabfallkurve. Folgende Punkte sind dabei besonders markiert:

A)	Kavitationsbeginn bzw. $NPSH_{incipient}$
B)	etwa 1% Förderhöhenabfall
C)	etwa 3% Förderhöhenabfall bzw. $NPSH_{3\%}$ und beginnender Steilabfall
D)	etwa 10% Förderhöhenabfall
E)	etwa 20% Förderhöhenabfall
F)	etwa 30% Förderhöhenabfall

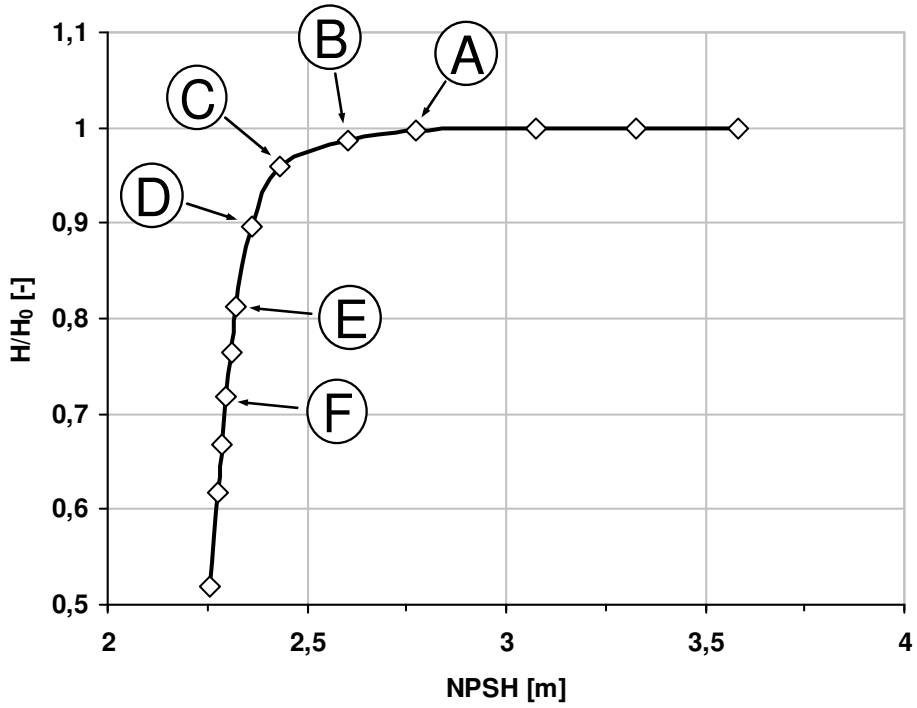


Bild 5-17: Analyse des Förderhöhenabfalls bei 25 °C Raumtemperatur; $Q = 2.5\ m^3/h$; $n = 2223\ 1/min$

Bild 5-18 zeigt die Ausdehnung der Kavitationsgebiete in der Heizungsumwälzpumpe charakterisiert durch die Isoflächen mit 10% Dampfvolumenfraktion in den ausgewählten Betriebspunkten und für eine relative Laufradstellung von 0°. Die Teilbilder sind dabei jeweils den markierten Punkten aus Bild 5-17 zugeordnet. Kavitation ist zum ersten Mal deutlich bei einem NPSH-Wert von $NPSH_{incipient} = 2.75\ m$ in den Schaufelkanälen nahe dem Druckstutzen der Spirale zu erkennen. Die dem Druckstutzen gegenüberliegenden Kanäle zeigen zu diesem Zeitpunkt noch keinerlei Kavitation. Bei einem NPSH-Wert von $NPSH = 2.6\ m$ hat die Pumpe etwa 1% ihrer Förderhöhe verloren. Die Kavitationsgebiete nahe dem Druckstutzen der Spirale sind angewachsen. Auch in den restlichen Schaufelkanälen ist nun Kavitation auszumachen. Bei einem NPSH-Wert von $NPSH = 2.5\ m$ hat die Pumpe etwa 3% ihrer Förderhöhe verloren. Die Kavitationsgebiete in allen Schaufelkanälen wachsen weiter an und engen den effektiven Querschnitt der Kanäle ein. An dieser Stelle beginnt der Steilabfall der Förderhöhenabfallkurve. Die Punkte D) – F) befinden sich im Steilabfall. Bei einem NPSH-Wert von $NPSH = 2.36\ m$ hat die Pumpe etwa 10% ihrer ursprünglichen Förderhöhe eingebüßt, bei einem NPSH-Wert von $NPSH = 2.32\ m$ etwa 20%, und bei einem NPSH-Wert von $NPSH = 2.3\ m$ bereits etwa 30%.

Wegen des Einflusses der Spirale verstopfen die kavitierenden Schaufelkanäle zu Anfangs unterschiedlich stark. Das führt nach und nach zu einer Umlenkung der Strömung hin zu noch freien Schaufelkanälen, was sich wiederum ausgleichend auf die Kavitationsgebiete auswirkt. In Punkt F) ist zwischen den Kavitationsgebieten der einzelnen Schaufelkanäle dann nur noch ein geringer Unterschied erkennbar.

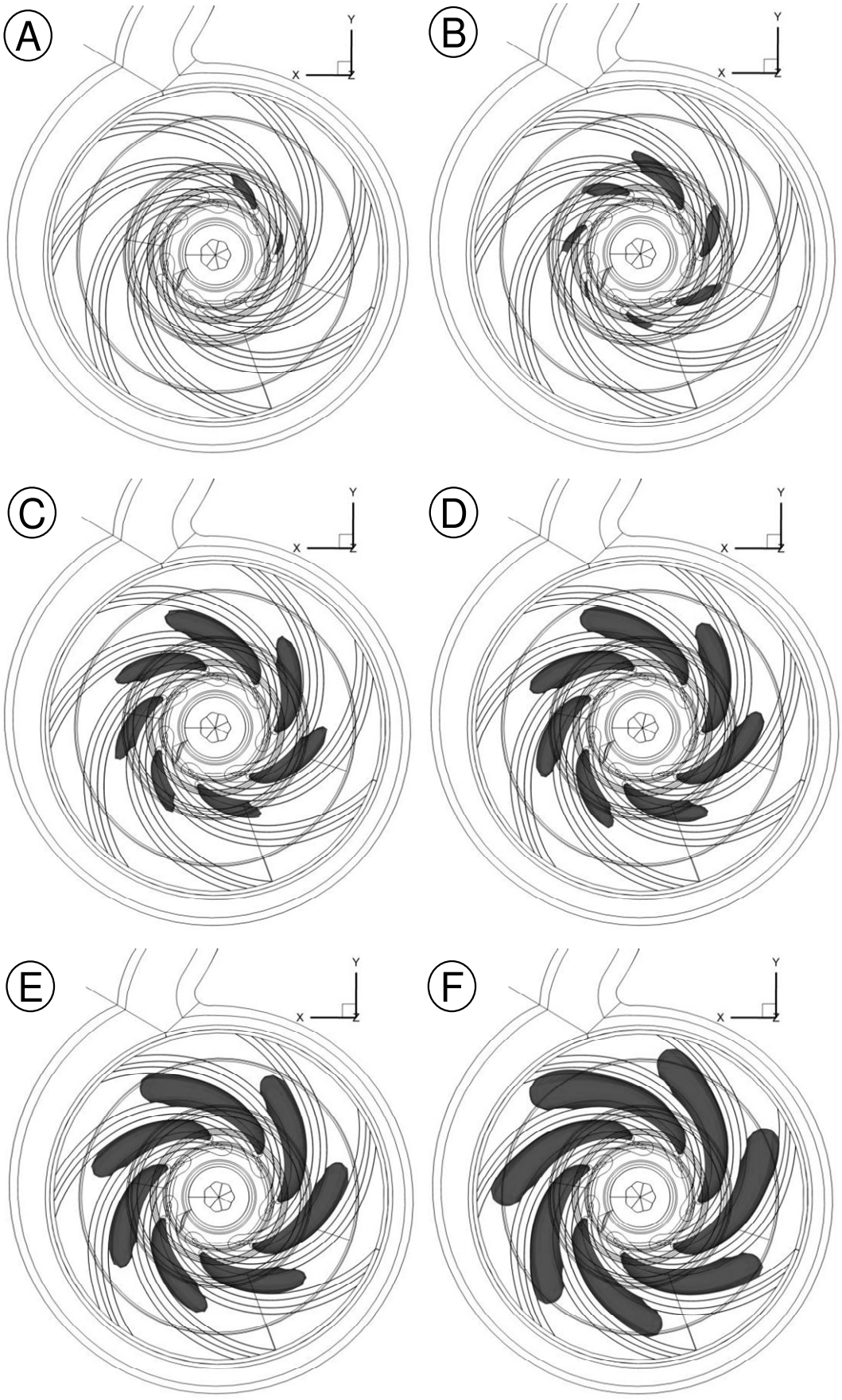


Bild 5-18: Isoflächen mit 10% Dampfvolumenfraktion für die ausgewählten Punkte der Förderhöhenabfallkurve; Q = 2.5 m^3/h; n = 2223 1/min

Zur genaueren Untersuchung des Einflusses der Spirale auf die Kavitationsgebiete in den einzelnen Schaufelkanälen werden die lokalen Strömungsverhältnisse bei beginnender Kavitation nun kanalweise analysiert. Dabei muss allerdings darauf hingewiesen werden, dass transiente Strömungsphänomene durch eine Frozen-Rotor-Modellierung nicht wiedergegeben werden können.

Bild 5-19 legt die Nummerierung der einzelnen Schaufelkanäle 1) – 7) sowie die Position der für die kanalweise Analyse benötigten Schnitte S1) – S3) fest. Die Position der Schnitte wurde dabei so gewählt, dass der erste Schnitt S1) stromauf vor den Schaufelvorderkanten, der zweite Schnitt S2) unmittelbar vor den Schaufelvorderkanten und der dritte Schnitt S3) stromab hinter den Schaufelvorderkanten, bzw. innerhalb der Schaufelkanäle liegt.

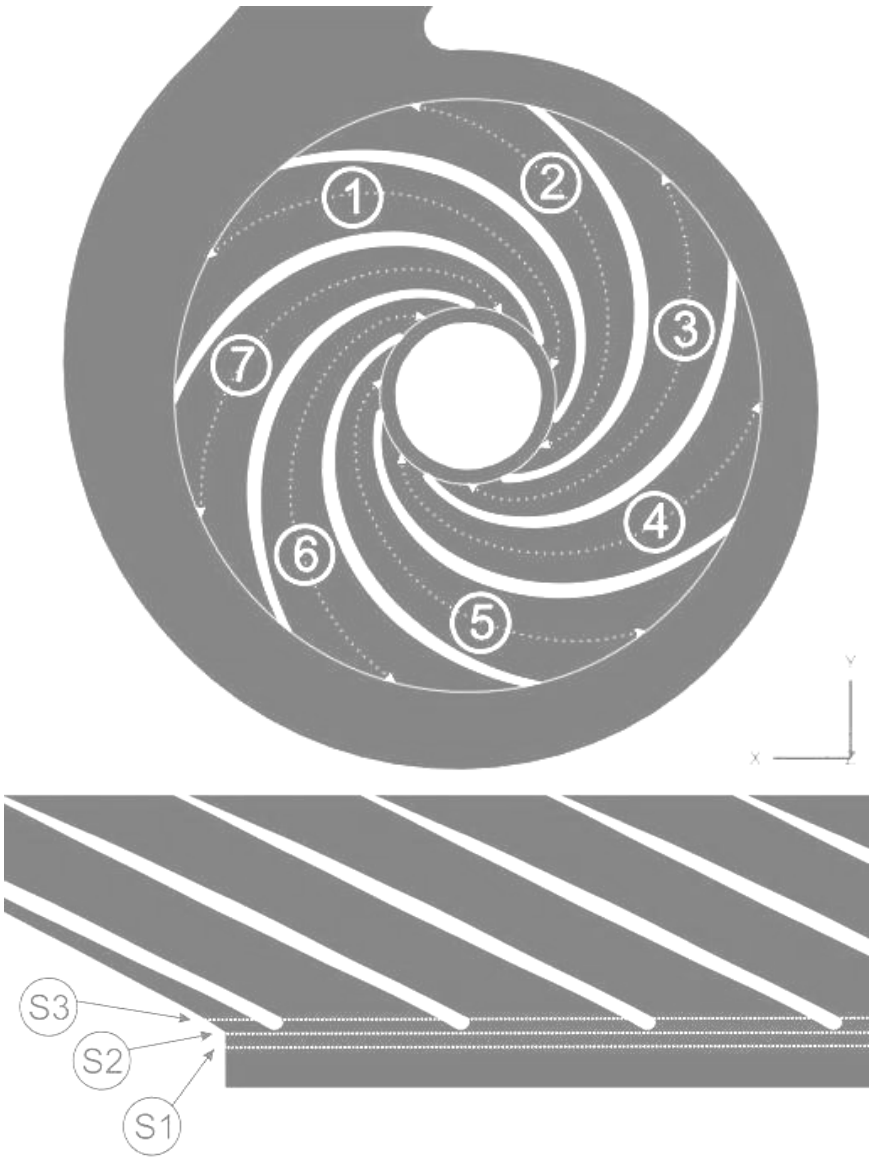


Bild 5-19: Nummerierung der Schaufelkanäle 1) – 7) sowie Position und Bezeichnung der Schnitte S1) – S3) durch die Schaufelkanäle der Heizungsumwälzpumpe

Bild 5-20 zeigt die Auswertung der Relativgeschwindigkeiten w entlang der einzelnen Schnitte S1) – S3) bei dem gewählten Nennvolumenstrom $Q_0 = 2.5\, m^3/h$, der Drehzahl $n = 2223\ 1/\text{min}$ und der relativen Rotorposition 0° für die kavitationsfreie Startlösung. Die Relativgeschwindigkeiten entlang der einzelnen Schnitte weisen eine deutliche Profilierung auf, die für den Schnitt S3) besonders ausgeprägt ausfällt. Die höchsten Relativgeschwindigkeiten treten dabei im Schaufelkanal 1) auf, die niedrigsten im Schaufelkanal 5). Entsprechend ist zu erwarten, dass die zu den Relativgeschwindigkeiten korrespondierenden statischen Drücke in Kanal 1) am niedrigsten und in Kanal 5) am höchsten sind.

Bild 5-21 wertet die kanalweise gemittelten statischen Drücke P normiert auf das mittlere statische Druckniveau P^* am Schnitt S3) aus. Die Drücke variieren um etwa 3% bezüglich des mittleren Druckniveaus. Der niedrigste berechnete Druckwert liegt in Kanal 1). Dort ist bei Kavitationsbeginn auch das erste Kavitationsgebiet auszumachen, siehe Bild 5-18. Der höchste berechnete Druck liegt in Kanal 5). Dieser Kanal bleibt bei sinkendem Druckniveau in der Pumpe am längsten kavitationsfrei, siehe Bild 5-18.

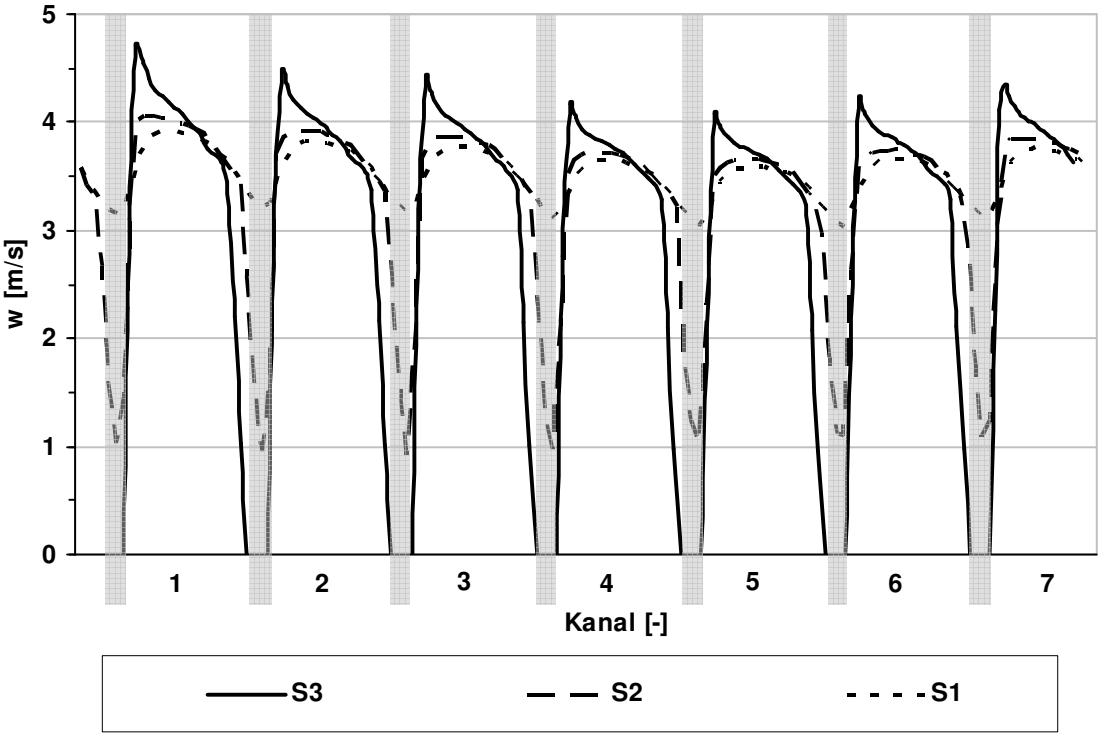


Bild 5-20: Verlauf der Relativgeschwindigkeit *w* entlang der Schnitte S1), S2) und S3) durch die Schaufelkanäle 1) – 7)

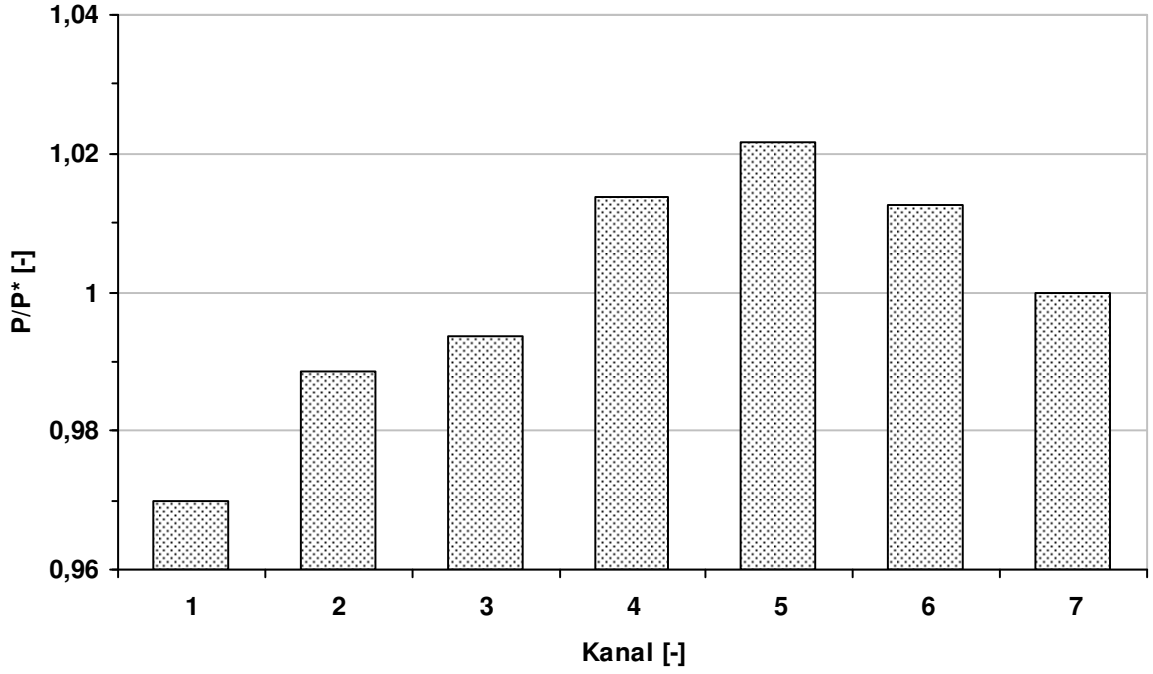


Bild 5-21: relative Verteilung der über den Schnitt S3) kanalweise gemittelten und entdimensionierten statischen Drücke über den Schaufelkanälen 1) – 7)

5.3.2 Numerische Ergebnisse Radialpumpe NQ26 – Glycol

Den zweiten Testfall bildet die Simulation der kavitierenden Strömung durch ein am FST vermessenes radiales Pumpenlaufrad mit spezifischer Drehzahl $n_q = 26\ \ 1/\min$. Die spezifische Drehzahl ist definiert als:

$$n_q = n \frac{Q^{\frac{1}{2}}}{H^{\frac{3}{4}}} \tag{5.42}$$

mit dem Volumenstrom Q in m^3/s und der Förderhöhe H in m. Bild 5-22 zeigt die Seitenansicht der Laufradgeometrie im Schnitt. Das Laufrad besitzt 6 Schaufeln. Der Laufraddurchmesser beträgt $D = 260mm$. Der Auslegungspunkt befindet sich für eine Laufraddrehzahl von $n = 1750\ \ 1/\min$ bei einem Volumenstrom von $Q_{opt} = 125\ \ m^3/h$.

Bei der Vernetzung der Geometrie wird wieder wie zuvor der Radseitenraum mit berücksichtigt, da er auf das Kavitationsverhalten der Kreiselpumpe erheblichen Einfluss hat, siehe FROBENIUS [30]. Zur Simulation der Förderhöhenabreißkurven wird das Rechennetz aber nur bis kurz hinter die Schaufelhinterkante gezogen, die Strömung im radialen Diffusor des Versuchsstandes am FST soll nicht berücksichtigt werden. Alle Rechnungen werden stationär mit nur einem Schaufelkanal durchgeführt. Der Turbulenzgrad am Rechengebietseintritt wird mit $Tu = 2\%$ angenommen. Die Werte für k und ε ergeben sich mit dem Laufraddurchmesser als charakteristisches Längenmaß aus der jeweiligen Zuströmgeschwindigkeit, siehe Anhang. Die konvektiven Flüsse werden mit dem OSHER-Schema, siehe Kapitel 4.2.2, diskretisiert.

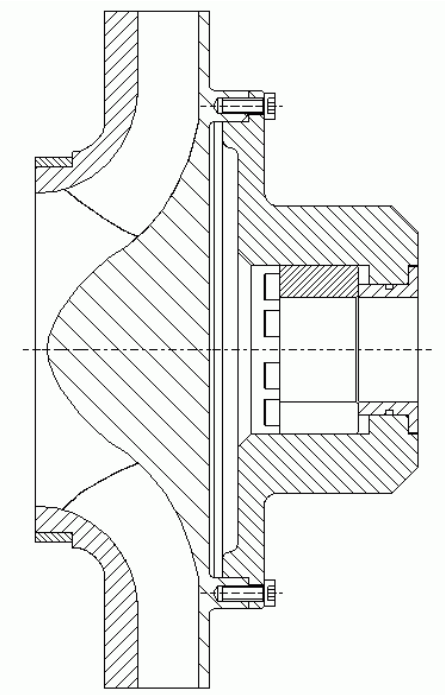


Bild 5-22: Schnitt durch die Geometrie des Pumpenlaufrades NQ26 in der Seitenansicht

Für die Berechnung der Förderhöhenabreißkurven wird Glycol bei 20 °C Raumtemperatur als Fördermedium verwendet. Bei dieser Temperatur beträgt die Dichte von Glycol $\rho_l = 1113\ kg/m^3$ und die dynamische Viskosität $\mu_l = 21.2\ mPas$. Der Sättigungsdampfdruck liegt bei $p_v = 0.07\ mbar$. Die entsprechende Dichte des Glycoldampfes ergibt sich zu $\rho_v = 0.17\ g/m^3$ und die dynamische Viskosität zu $\mu_v = 0.01\ mPas$. Als Keimdichte werden $n_0 = 10^8\ Keime/m^3\ Flüssigkeit$ angenommen. Außerdem wird eine initiale Volumenfraktion von $\alpha = 10^{-4}\ m^3 Dampf\ /m^3\ Gesamtvolumen$ gesetzt.

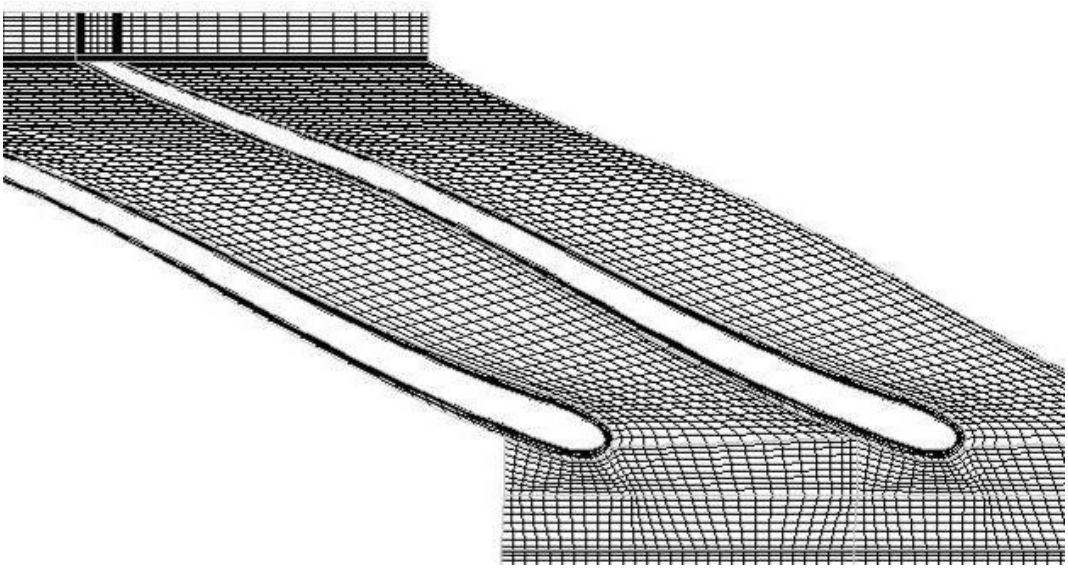


Bild 5-23: Rechennetz der Versuchspumpe; Blade-to-Blade-Ansicht in der konformen Abbildung

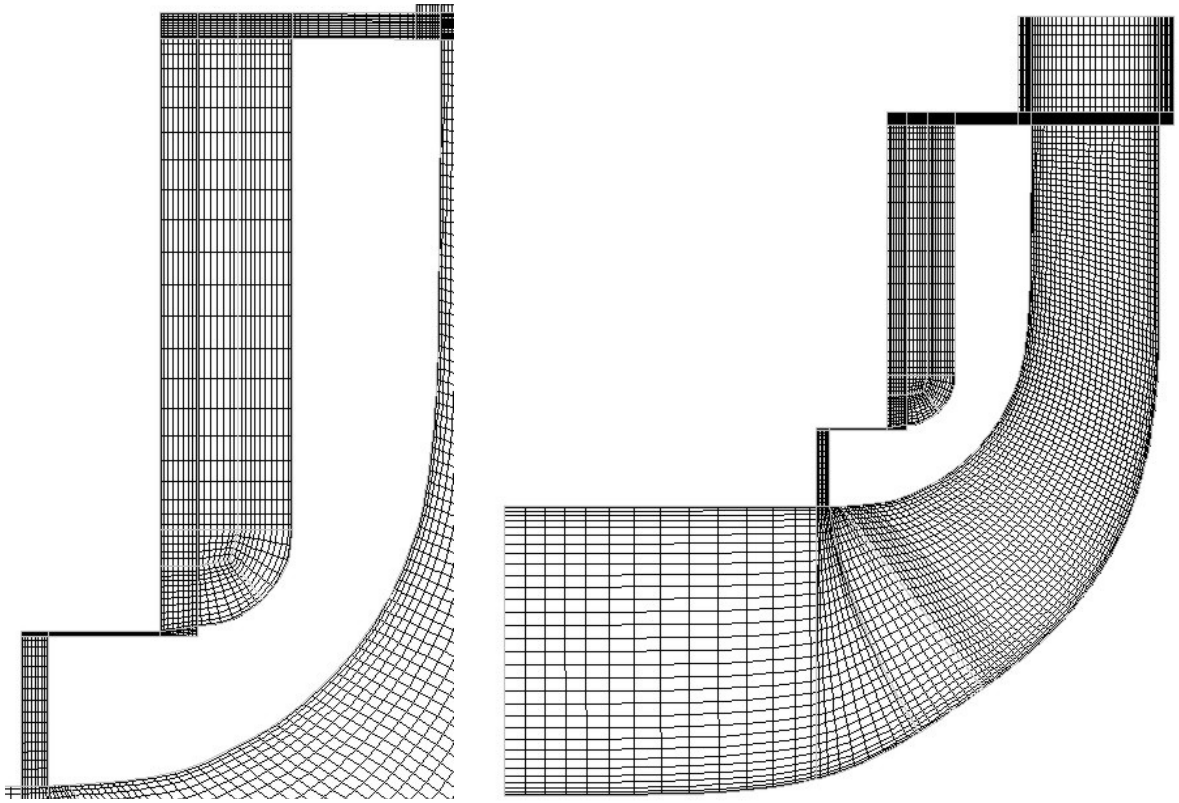


Bild 5-24: Rechennetz der Versuchspumpe im Meridianschnitt; links: Detailansicht des Radseitenraumes; rechts: gesamte Pumpe

Bei den Simulationen wird die 20mal höhere Viskosität von Glykol berücksichtigt, die auf demselben Rechennetz im Vergleich zu Wasser zu wesentlich geringeren y^+-Werten führt. Die Entscheidung bezüglich der turbulenten Wandmodellierung fällt deshalb auf einen Low-Reynolds-Ansatz. Als Turbulenzmodell wird das im Kapitel 5.1.2 vorgestellte nichtlineare LCL-Modell ausgewählt. Durch die für die Low-Reynolds-Modellierung günstigen Stoffeigenschaften von Glycol ist es möglich ein Low-Reynolds-Netz im gültigen y^+-Bereich $y^+ \approx 1$ mit einer Netzauflösung von lediglich etwa 260000 Zellen zu erzeugen. Bild 5-23 zeigt das verwendete Rechennetz in der Blade-to-Blade-Ansicht als konforme Abbildung einer kanalmittigen geometrischen Strombahn. Der Meridianschnitt durch das Rechennetz der Versuchspumpe sowie die Detailansicht des Radseitenraumes sind in Bild 5-24 zu sehen.

Bild 5-25 stellt eine repräsentative y^+-Verteilung im Rechengebiet für den Nennleistungspunkt bei $n = 1750\ \mathrm{1/min}$ dar. Es wird deutlich, dass beinahe alle Wandzellen im gültigen Bereich liegen. Weniger als 2% der Zellen liegen im Bereich $y^+ > 1$.

Für die Berechnung der volumenstromabhängigen Förderhöhenabfallkurven mit Glycol sind mehrere Berechnungsserien, wie sie in Kapitel 5.3.1 beschrieben wurden, notwendig. Ausgangspunkt ist zunächst wieder eine kavitationsfreie Rechnung. Durch die Auswertung dieser Rechnung ist das minimale Druckniveau im Strömungsfeld bekannt. Die nötige Verschiebung des gesamten Druckniveaus in der Maschine über die Druckrandbedingung am Auslass wird dann so bestimmt, dass das Druckminimum im Strömungsfeld über dem Dampfdruck des Fluids liegt. Es folgt dann eine Reihe von Strömungs-

simulationen, bei denen der Druck am Auslass und somit das Druckniveau der gesamten Maschine sukzessive immer weiter abgesenkt wird. Sinkt dabei das Druckminimum im Strömungsfeld unter den Dampfdruck von Glycol bildet sich an dieser Stelle ein Kavitationsgebiet aus, das anwächst, je weiter das Druckniveau der gesamten Maschine absinkt. Dabei muss der Betrag der Druckabsenkungen zwischen den einzelnen Simulationen ab beginnendem Steilabfall immer weiter reduziert werden. Der NPSH-Wert am Einlass des Rechengebietes und die Förderhöhe H der Maschine werden dann für jedes berechnete Druckniveau ausgewertet und zu einer zusammenhängenden Förderhöhenabfallkurve vereinigt.

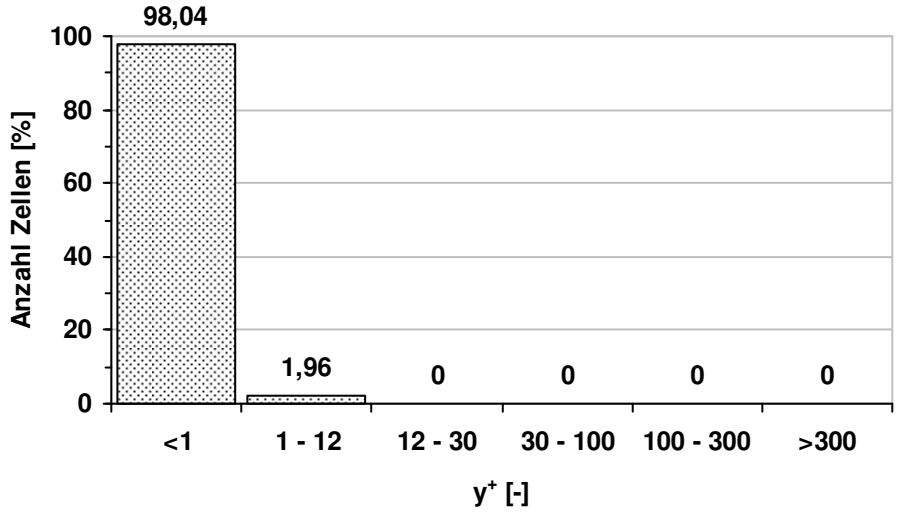


Bild 5-25: y^+-Verteilung für das Low Reynolds Netz der Versuchspumpe; n = 1750 1/*min*; Q = 125 m^3/h

Die Förderhöhe wird nach FLM-Konvention standardmäßig über die Totaldruckdifferenz zwischen den Bilanzflächen am Rechengebietsein- und -austritt berechnet, siehe Gleichung (5.41). Der Vorteil in der Verwendung der Totaldruckdifferenz liegt darin, dass sich Abweichungen der Bilanzierungsebenen zwischen Rechenmodell und Messaufbau lediglich durch relativ geringe Totaldruckverluste bemerkbar machen. Die Förderhöhe der Versuchspumpe ist am FST aber über die statische Druckdifferenz bestimmt worden. Zum Vergleich mit der Messung muss die numerisch berechnete Förderhöhe deshalb ebenfalls über die statische Druckdifferenz gebildet werden, was eine Extrapolation des statischen Drucks über den Rechengebietsaustritt hinaus mithilfe des Flächenverhältnisses zwischen Austrittsradius und Radius der Drucksensoren des Messaufbaus erfordert. Die über die statische Druckdifferenz zwischen der auslass- und einlassseitigen Bilanzierungsebene gebildete Förderhöhe ist dabei wie folgt definiert:

$$H = \frac{p_{out} - p_{in}}{\rho_l g}. \tag{5.43}$$

Der Förderhöhenabfall wird bei der Drehzahl $n = 1750\ 1/\mathrm{min}$ für drei unterschiedliche Volumenströme im Auslegungspunkt bei $Q_{opt} = 125\ m^3/h$ sowie im Teillastbereich bei

$0.8 \cdot Q_{opt} = 100\ m^3/h$ und im Überlastbereich bei $1.2 \cdot Q_{opt} = 150\ m^3/h$ berechnet. Den Vergleich mit der Messung liefert jeweils der sich einstellende $NPSH_{3\%}$-Wert, siehe Kapitel 5.3.1. Bild 5-26 zeigt die aus den zugrundeliegenden Förderhöhenabfallkurven gewonnenen $NPSH_{3\%}$-Werte. Die Förderhöhe ist dabei, wie gerade beschrieben, über die statische Druckdifferenz bestimmt worden. Insgesamt ist eine hervorragend gute Übereinstimmung zwischen Simulation und Messung zu erkennen. Lediglich im Teillastbereich stellt sich eine leichte Abweichung zwischen Simulationsergebnis und Messung ein.

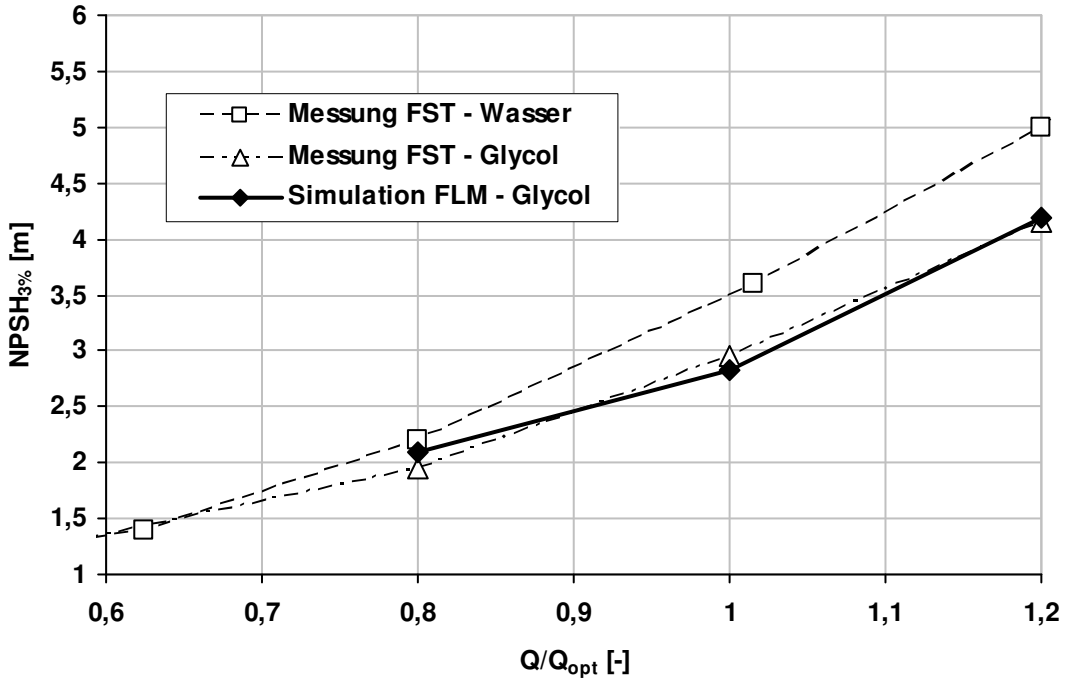


Bild 5-26: berechneter und gemessener $NPSH_{3\%}$-Verlauf für Glycol; *n* = 1750 1/*min*

Eine detaillierte Analyse des Förderhöhenabfalls sowie der Ausprägung der Kavitationsgebiete zum jeweiligen Druckniveau wird im Folgenden anhand der Ergebnisse im Optimalpunkt durchgeführt. Bild 5-27 zeigt die mit der jeweils kavitationsfreien Lösung normierten relativen Förderhöhenabfallkurven, die mit der Totaldruckdifferenz bzw. der statischen Druckdifferenz gewonnen wurden. Durch die Normierung fallen die Kurven zusammen. Folgende Punkte sind besonders markiert:

A)	Kavitationsbeginn bzw. $NPSH_{incipient}$
B)	etwa 0,5% Förderhöhenabfall
C)	etwa 1% Förderhöhenabfall und beginnender Steilabfall
D)	etwa 2% Förderhöhenabfall
E)	etwa 3% Förderhöhenabfall bzw. $NPSH_{3\%}$

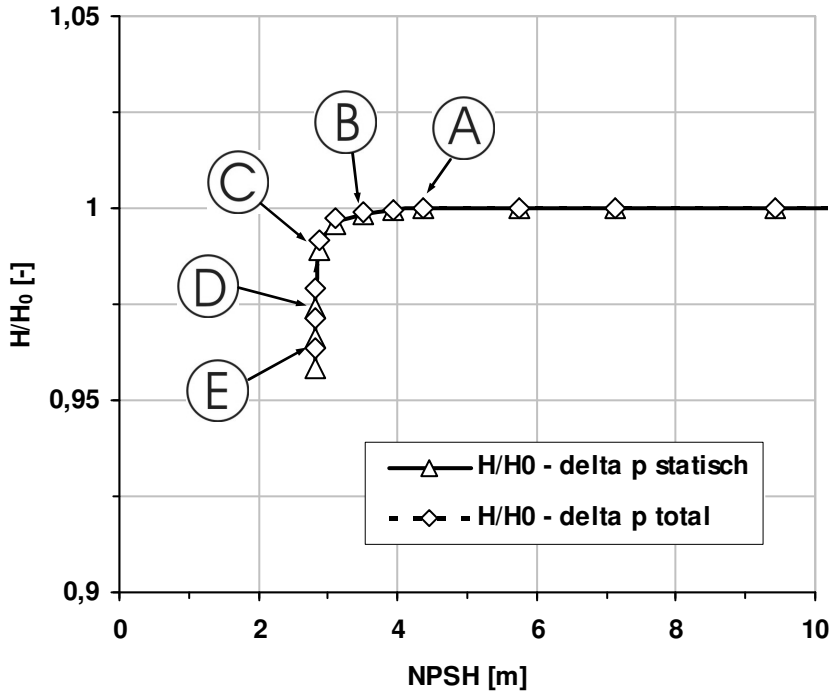


Bild 5-27: berechnete relative Förderhöhenabfallkurven; n = 1750 1/*min*; Q = 125 m^3/h

Alle weiteren Abbildungen sind diesen ausgewählten Punkten zugeordnet. Kavitation ist zum ersten Mal in Punkt A) bei einem NPSH-Wert von etwa $NPSH_{incipient} = 4.4\,m$ auf der Schaufeldruckseite zu erkennen. Dies ist auf den Einfluss des Radseitenraumes zurückzuführen, durch den sich auf der Schaufeldruckseite nahe der Deckscheibe eine ausgeprägte Saugspitze ausbildet. Bei einem NPSH-Wert von $NPSH = 3.5\,m$ hat die Pumpe etwa 0.5% ihrer Förderhöhe verloren. Das Kavitationsgebiet auf der Schaufeldruckseite ist deutlich angewachsen und beginnt die Strömung um die Schaufeln allmählich zu beeinflussen. Auf der Schaufelsaugseite ist in diesem Punkt noch keine Kavitation zu erkennen. Bei einem NPSH-Wert von $NPSH = 2.85\,m$ hat die Pumpe etwa 1% ihrer Förderhöhe verloren. Sowohl auf der Schaufeldruck- als auch auf der Schaufelsaugseite haben sich große Kavitationsgebiete ausgebildet. Das Kavitationsgebiet auf der Druckseite erstreckt sich nun soweit, dass der engste Querschnitt des Schaufelkanals eingeschnürt wird. Hier beginnt der Steilabfall der Förderhöhenabreißkurve. Bei einem NPSH-Wert von $NPSH = 2.83\,m$ hat die Pumpe dann schon etwa 2% ihrer Förderhöhe verloren. Das Kavitationsgebiet erstreckt sich nun über die ganze Schaufelbreite. Bei einem NPSH-Wert von $NPSH = 2.82\,m$ hat die Pumpe schließlich etwa 3% ihrer Förderhöhe eingebüßt. Die Kavitationsgebiete auf der Schaufeldruck- und der Schaufelsaugseite wachsen weiter stark an. Der engste Querschnitt ist nun deutlich eingeengt und die Strömung um die Schaufeln wird erheblich beeinflusst. Zusätzlich bildet sich hinter dem druckseitigen Kavitationsgebiet, verursacht von dessen Verdrängungswirkung, eine starke Querströmung aus.

Bild 5-28 und Bild 5-29 zeigen die jeweilige Ausprägung der Kavitationsgebiete in der konformen Abbildung in der Mitte des Schaufelkanals bzw. einem Meridianschnitt des Schaufelkanals nahe der Saugseite der Versuchspumpe für die ausgewählten Kavitationszustände. Im Punkt D), also bei etwa 2% Förderhöhenabfall, reicht das saugseitige

Kavitationsgebiet über die gesamte Schaufelbreite von der Deckscheibe bis zur Nabe. In der konformen Abbildung in der Mitte des Schaufelkanals ist die zunehmende Einschnürung des engsten Querschnittes während des Steilabfalls deutlich sichtbar.

Bild 5-30 und Bild 5-31 visualisieren die zugehörigen Verteilungen der meridionalen und normalen Relativgeschwindigkeiten w_m und w_n in der konformen Abbildung in der Mitte des Schaufelkanals. Blaue Farben im Meridiangeschwindigkeitsplot, violett bis weiße Farben im Normalgeschwindigkeitsplot weisen auf eine hohe negative Meridiangeschwindigkeits- bzw. hohe positive Normalgeschwindigkeitskomponente hinter dem druckseitigen Kavitationsgebiet hin. Das Kavitationsgebiet hat hier erheblichen Einfluss auf die Kanalströmung. Es kann, induziert durch die starken Querströmungen, zur instationären Wolkenablösung kommen. Die Geschwindigkeiten hinter dem Kavitationsgebiet auf der Saugseite werden dagegen kaum beeinflusst. An dieser Stelle ist deshalb eher mit Schichtkavitation zu rechnen.

Die für die ausgewählten Kavitationszustände über der Lauflänge s aufgetragenen normierten Schaufeldruckverteilungen an der Deckscheibe, in der Kanalmitte und an der Nabe sind in Bild 5-32 dargestellt. Die gestrichelte Linie markiert jeweils das Dampfdruckniveau. Die normierte Druckverteilung ist definiert als:

$$c_p = \frac{p}{\frac{1}{2}\rho u_{ref}^2}, \tag{5.44}$$

wobei die Umfangsgeschwindigkeit des Laufrades am Laufradaustritt als Referenzgeschwindigkeit u_{ref} herangezogen wird. Deutlich zu erkennen ist, dass der Druck bei auftretender Kavitation innerhalb des Kavitationsgebietes auf Dampfdruckniveau gehalten wird. Bei beginnender Kavitation in Punkt A) tritt eine ausgeprägte druckseitige Saugspitze in der Nähe der Deckscheibe auf. Entsprechend der Ausbreitung des Kavitationsgebietes in der Meridianansicht, erreichen die normierten Druckverläufe im Punkt D), bei etwa 2% Förderhöhenabfall, sowohl auf der Saug- als auch auf der Druckseite über die gesamte Schaufelbreite Dampfdruckniveau. Während hinter den Kavitationsgebieten auf der Druckseite ein relativ schneller Druckanstieg erfolgt, steigt der Druck hinter den Gebieten mit Schichtkavitation auf der Saugseitige dagegen verhältnismäßig langsam an.

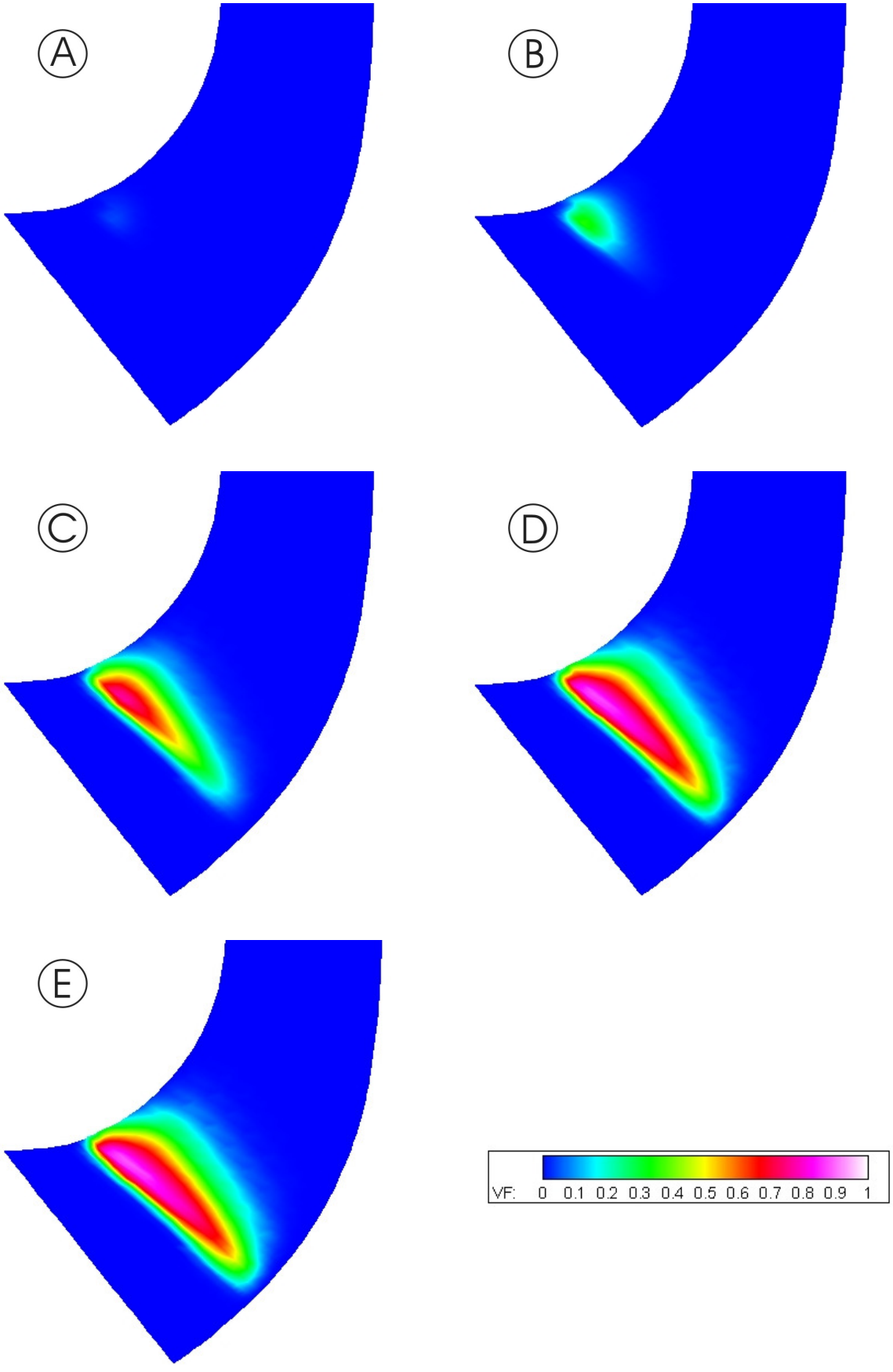


Bild 5-28: Dampfvolumenfraktion in der Meridianansicht nahe der Saugseite im Schaufelkanal für ausgewählte NPSH-Werte: Q = 125 m^3/h; n = 1750 1/min

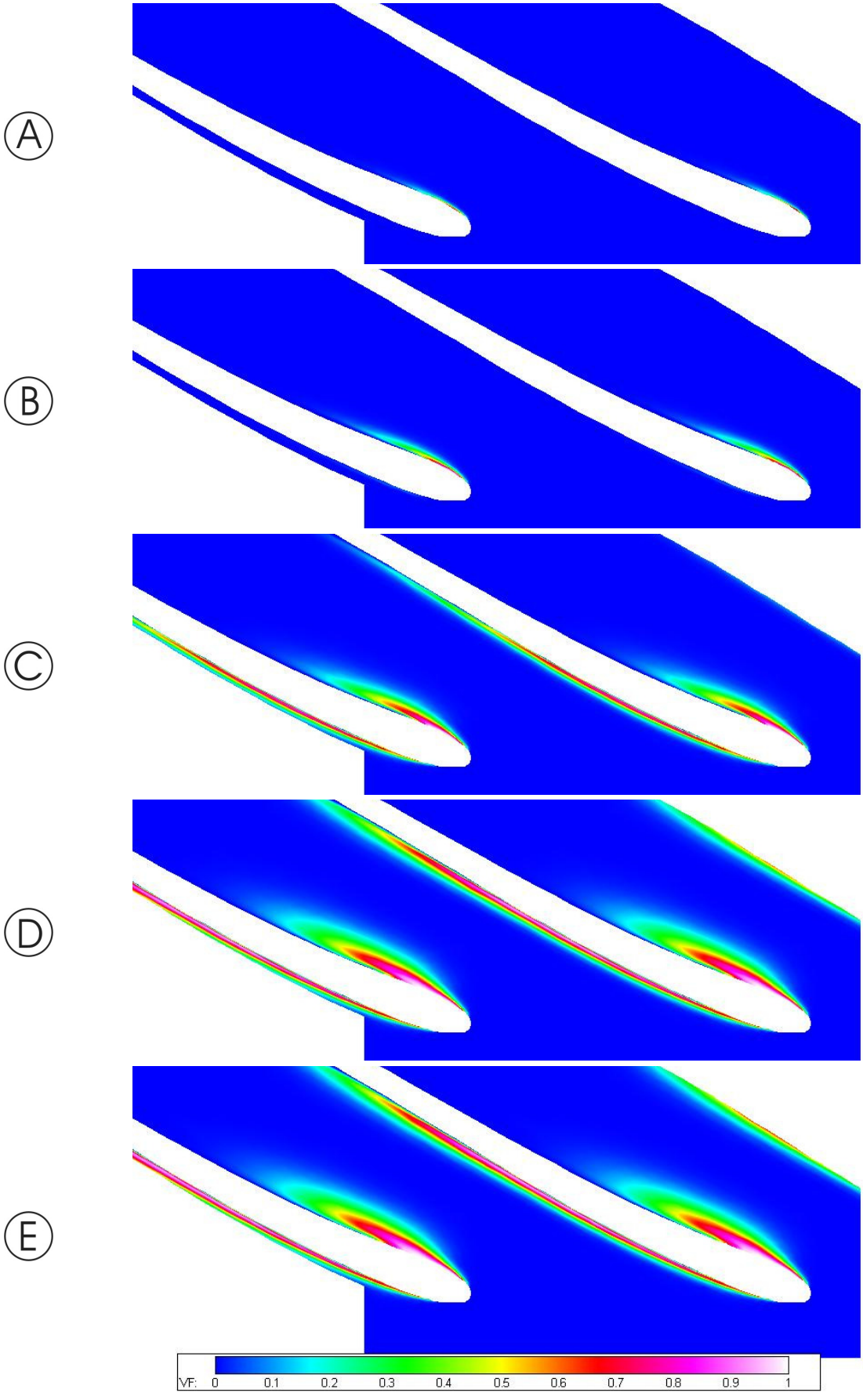


Bild 5-29: Dampfvolumenfraktion in der konformen Abbildung in der Mitte des Schaufelkanals für ausgewählte NPSH-Werte: Q = 125 m^3/h; n = 1750 1/min

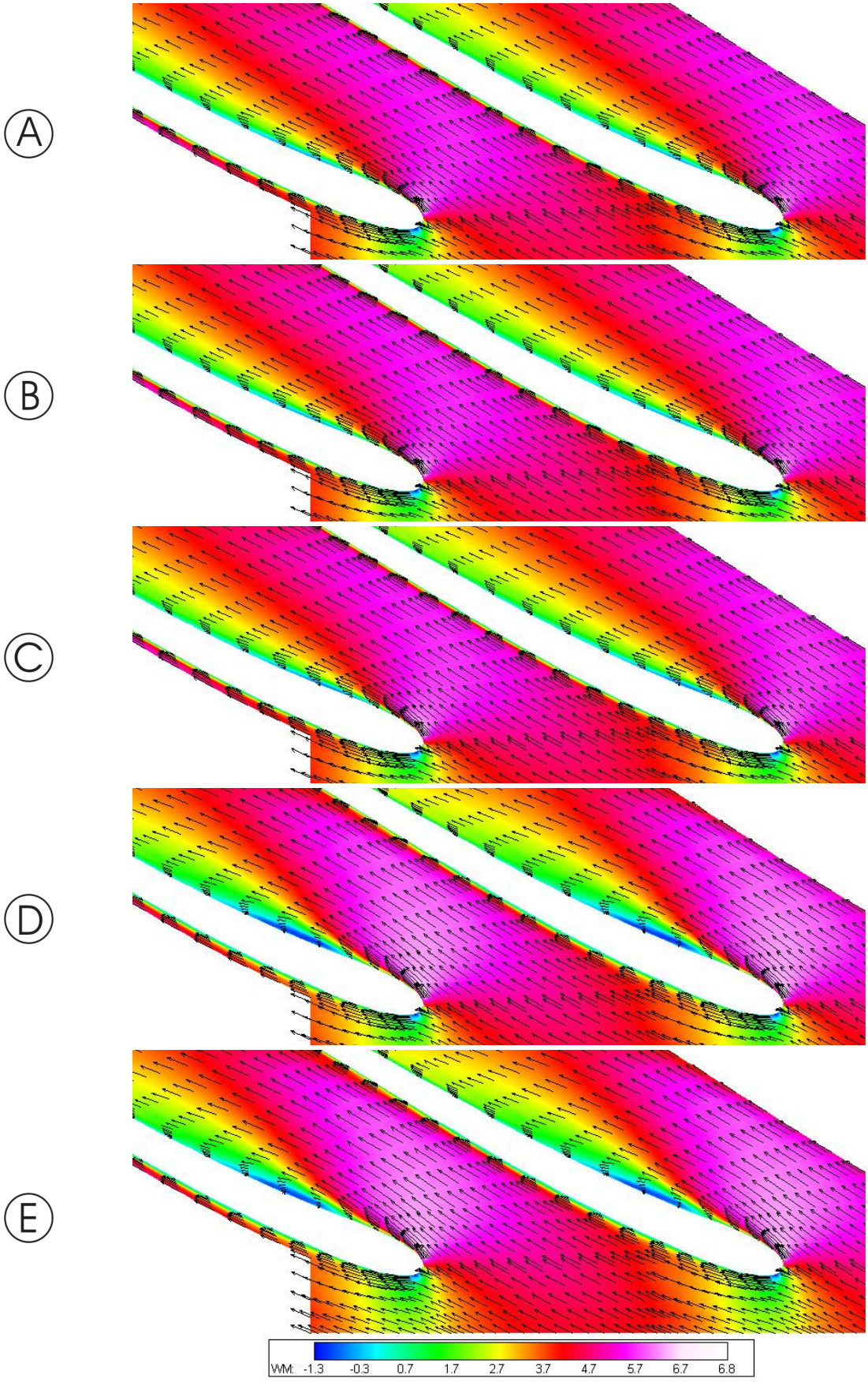


Bild 5-30: meridionale Relativgeschwindigkeiten w_m in der konformen Abbildung in der Mitte des Schaufelkanals für ausgewählte NPSH-Werte, sowie die zugehörigen Geschwindigkeitsvektoren tangential zur dargestellten Stromfläche: Q = 125 m^3/h; n = 1750 1/min

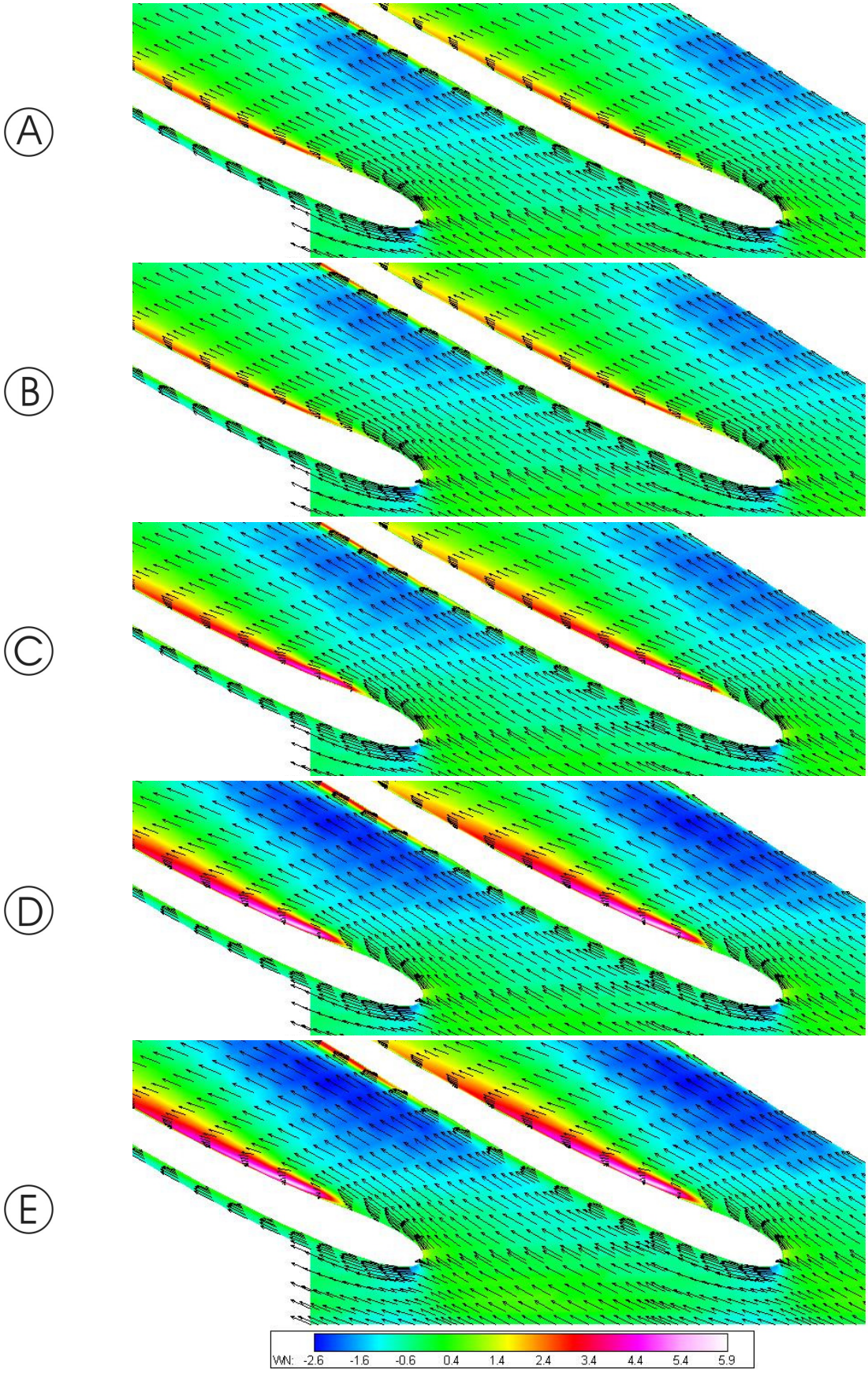


Bild 5-31: normale Relativgeschwindigkeiten w_n in der konformen Abbildung in der Mitte des Schaufelkanals für ausgewählte NPSH-Werte, sowie die zugehörigen Geschwindigkeitsvektoren tangential zur dargestellten Stromfläche: Q = 125 m^3/h; n = 1750 $1/min$

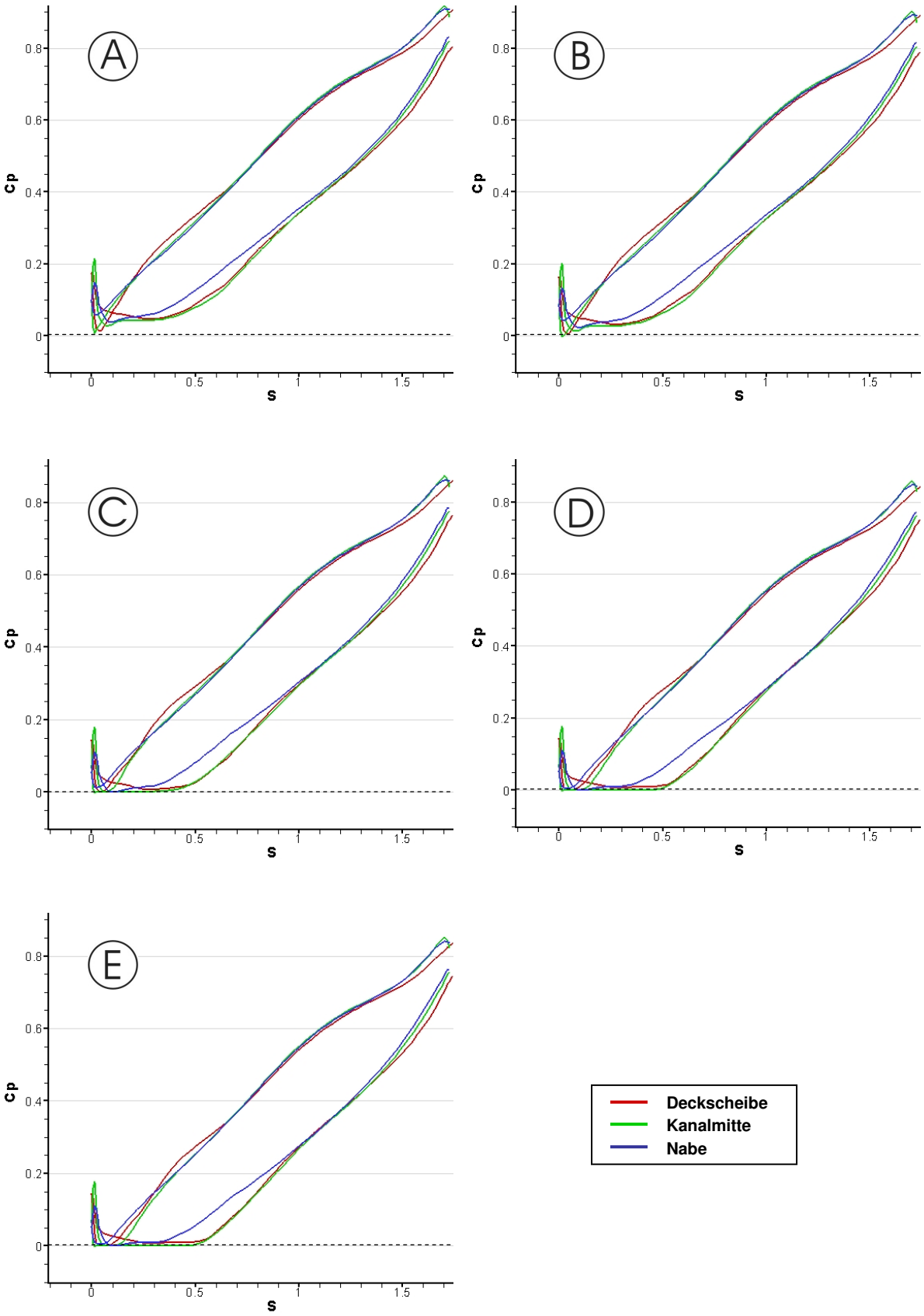


Bild 5-32: normierte Druckverteilungen entlang der Lauflänge für ausgewählte NPSH-Werte: Q = 125 m^3/h; n = 1750 1/min

6 Bewertung und Ausblick

In der vorliegenden Arbeit wird der Entwurf eines flexiblen Werkzeugs zur Simulation von Strömungsvorgängen mit der FVM vorgestellt. Besondere Aufmerksamkeit wird dabei vor allem dem modularen und objektorientierten Design der Software geschenkt, wodurch sowohl die Sicherheit als auch die Flexibilität bei der Implementierung neuer physikalischer Modelle wesentlich gesteigert werden kann.

Im Zuge der Konzepterstellung werden zunächst die Anforderungen der einzelnen Anwendergruppen, denen eine moderne Simulationssoftware gerecht werden muss, zusammengetragen und analysiert. Es ergeben sich drei Hauptanwendergruppen mit stark unterschiedlichen Anforderungsprofilen, der klassische Simulationsingenieur, der Forscher und der Programmierer. Als Kriterien bei der Anforderungsanalyse dienen Eigenschaften wie die Programmstabilität, die Rechenperformance, die Möglichkeit zur Implementierung neuer physikalischer Modelle, die Wartbarkeit des Quelltextes sowie die Parametrierbarkeit der Software. Das Ergebnis der Analyse wird in einem Schichtenmodell umgesetzt, das die Software grob strukturiert und die Interaktionsebenen der drei Anwendergruppen mit der Software beschreibt.

Aus den Erkenntnissen der vorangegangenen Konzeptfindungsphase geht der sich anschließende Entwurf der grundlegenden Datenstrukturen hervor. Insbesondere auf die bequeme Handhabung der bei der Diskretisierung der fluidmechanischen Gleichungen auftretenden Tensorarithmetik sowie die effiziente Programmparallelisierung wird geachtet. Die effiziente Implementierung der Tensorarithmetik ermöglicht eine bequeme Diskretisierung der Gleichungen. Bei der Implementierung werden zunächst die Syntaxbäume in einem klassisch objektorientierte Design beleuchtet und die hierbei entstehenden Probleme herausgearbeitet. Danach wird mithilfe der Template-Metaprogrammierung ein hocheffizienter Lösungsansatz entwickelt, der die Nachteile des klassischen Designs umgeht. Die Detaillierung der Klassen für die Repräsentation der numerischen Netze und der Erhaltungsgleichungen, für die Berechnung der geometrischen Größen, für die Diskretisierung der Gleichungsterme und für die Ablaufsteuerung der Programms schließen die programmiertechnische Umsetzung der erarbeiteten Konzepte ab.

Der gesamte Softwareentwurf wird dann anhand des klassischen Modellentwicklungsprozesses validiert. Der Prozess bildet die Anforderungen der verschiedenen Anwendergruppen gut ab. Als konkretes Beispiel dient die Entwicklung eines Kavitationsmodells, das die kavitierende Strömung in hydraulischen Strömungsmaschinen für den stationären Fall simulieren kann. Das Modell beruht auf sphärischer Blasendynamik und ist in der Lage kavitierende Strömungen auch anderer Flüssigkeiten als Wasser zu simulieren. Zudem beschränkt das Modell das Temperaturniveau der Strömung nicht auf die Raumtemperatur. Die Einführung des effektiven Dampfdrucks erweitert die Anwendbarkeit des Modells außerdem auf Flüssigkeiten, die noch gelöste Gase enthalten.

Das Kavitationsmodell wird bei der Simulation der Strömung durch eine Radialpumpe mit spezifischer Drehzahl $n_q = 26\ \mathrm{1/min}$ getestet. Als zu förderndes Fluid kommt Glykol zum Einsatz, dessen Stoffwerte sich bereits bei Raumtemperatur deutlich von denen des Wassers unterscheiden. Die Beurteilung der Simulationsergebnisse erfolgt durch Vergleich mit dem Experiment. Die errechneten $NPSH_{3\%}$-Werte stimmen sehr gut mit den zur Verfügung stehenden Messdaten des FST überein. Der Einfluss der auftretenden saug-

und druckseitigen Kavitationsgebiete auf das Strömungsfeld und den Druckverlauf in den Schaufelkanälen des Laufrades wird detailliert für einige ausgesuchte Punkte der Förderhöhenabfallkurve untersucht. Den zweiten Testfall für das Kavitationsmodell bildet die kavitierende Strömung durch eine Heizungsumwälzpumpe. Das Kavitationsverhalten der Pumpe wird hier insbesondere auch auf unterschiedlichen Temperaturniveaus betrachtet. Als Fördermedium kommt unbehandeltes Wasser zum Einsatz. Auch für diesen Fall zeigt sich eine hervorragende Übereinstimmung zwischen der Simulation und den experimentellen Daten des FST. Da das diskrete Modell der Heizungsumwälzpumpe auch die Spirale beinhaltet, kann zusätzlich deren Einfluss auf die Kavitationsgebiete innerhalb des Laufrades analysiert werden.

A Anhang

A.1 Turbulenzmodelle

Standard-k-ε-Modell

Konstanten:

$$C_\mu = 0.09\,, \qquad C_{\varepsilon 1} = 1.44\,, \qquad C_{e2} = 1.92\,, \qquad \sigma_k = 1.0\,, \qquad \sigma_\varepsilon = 1.3\,. \tag{A.1}$$

Dämpfungsfunktionen:

$$f_\mu = f_1 = f_2 = 1. \tag{A.2}$$

Low-Reynolds LCL-Modell

Konstanten:

$$\begin{aligned} &C_1 = \tfrac{3/4}{C_\mu\left(1000+\tilde{S}^3\right)}, \quad C_2 = \tfrac{15/4}{C_\mu\left(1000+\tilde{S}^3\right)}, \quad C_3 = \tfrac{19/4}{C_\mu\left(1000+\tilde{S}^3\right)}, \quad C_4 = -10C_\mu^2, \quad C_5 = 0\,, \\ &C_6 = -2C_\mu^2, \qquad C_7 = 2C_\mu^2, \qquad \sigma_k = 1.0\,, \qquad \sigma_\varepsilon = 1.3\,, \\ &C_\mu = \tfrac{2/3}{4+\tilde{S}+0.9\tilde{\Omega}}, \quad C_{\varepsilon 1} = 1.44\cdot\left(1+P_k'/P_k\right), \quad C_{e2} = 1.92 \end{aligned} \tag{A.3}$$

mit

$$\begin{aligned} P_k' &= 1.33\left(1-0.3e^{-\mathrm{Re}_T^2}\right)\left(P_k + 2\nu\,\tfrac{k}{y^2}\right)e^{-0.00375\mathrm{Re}_y^2}\,, \\ \tilde{S} &= \tfrac{k}{\varepsilon}\sqrt{2S_{ij}S_{ij}}\,, \qquad\qquad \tilde{\Omega} = \tfrac{k}{\varepsilon}\sqrt{2\Omega_{ij}\Omega_{ij}}\,. \end{aligned} \tag{A.4}$$

Die turbulenten Reynolds-Zahlen Re_T und Re_μ sind definiert als:

$$\mathrm{Re}_T = \frac{k^2}{\varepsilon\nu}, \qquad\qquad \mathrm{Re}_y = \frac{\sqrt{k}\,y}{\nu}\,. \tag{A.5}$$

Dämpfungsfunktionen:

$$f_1 = 1.0\,, \qquad f_2 = 1-0.3e^{-\mathrm{Re}_T^2}\,, \qquad f_\mu = \left(1-e^{-0.0198\cdot\mathrm{Re}_y}\right)\left(1+\tfrac{5.29}{\mathrm{Re}_y}\right). \tag{A.6}$$

A.2 Randbedingungen

Einlass

Die Geschwindigkeit am Einlass wird direkt als konstanter vektorieller Wert vorgeschrieben. Der statische Druck, bzw. die Druckkorrektur wird aus dem Gebietsinneren extrapoliert. Für die Dampfvolumenfraktion, sowie die Turbulenzgrößen werden feste Werte vorgegeben. Die turbulente kinetische Energie k_{in} wird dabei aus dem Turbulenzgrad Tu und der Geschwindigkeit berechnet:

$$k_{in} = \frac{3}{2} Tu^2 \, \overline{u}_{i,in}^2 \,. \tag{A.7}$$

Mit k_{in} ergibt sich für die Dissipationsrate ε_{in}:

$$\varepsilon_{in} = \frac{k_{in}^{\frac{3}{2}}}{L} \,, \tag{A.8}$$

wobei L ein charakteristisches Längenmaß darstellt, für das bei Turbomaschinen üblicherweise der Laufraddurchmesser D gewählt wird.

Auslass

Die Geschwindigkeiten, die Dampfvolumenfraktion und die Turbulenzgrößen am Auslass werden allesamt aus dem Gebietsinneren extrapoliert. Der statische Druck wird als konstanter Wert vorgeschrieben. Für die Druckkorrektur ergibt sich damit $p' = 0$.

Wand

An reibungsbehafteten Wänden ist die Geschwindigkeit gleich der Wandgeschwindigkeit. Der Druck, bzw. die Druckkorrektur, sowie die Dampfvolumenfraktion werden aus dem Gebietsinneren extrapoliert. Die turbulente kinetische Energie wird zu $k = 0$ gesetzt, außerdem verschwindet ihre Ableitung in Normalenrichtung $\partial k / \partial n = 0$. Der Wert der Dissipationsrate ε wird abhängig von der Wandmodellierung, siehe Kapitel 5.1.2, unterschiedlich bestimmt. Die entsprechende Implementierung ist beispielsweise bei SKODA [86] zu finden.

An reibungsfreien Wänden ist die Normalkomponente der Geschwindigkeit gleich der Wandgeschwindigkeit. Alle restlichen Größen werden mit einem Nullgradienten aus dem Gebietsinneren extrapoliert.

Literaturverzeichnis

[1] ABRAHAMS, A. ; GURTOVOY, A.: C++ Template Metaprogramming, Concepts, Tools, and Techniques from Boost and Beyond. Addison-Wesley, 2005

[2] ACOSTA, J. M.: *Numerial Algorithms for three Dimensional Computational Fluid Dynamic Problems*. Terrassa: Universidad Politécnica de Catalunya, tesis doctoral, 2001.

[3] American Institute of Aeronautics and Astronautics: The CFD General Notation System – Standard Interface Data Structures, AIAA Recommended Practice, 2007

[4] ATHAVALE, M. M. ; LI, H. Y. ; JIANG, Y. ; SINGHAL, A. K.: Application of the Full Cavitation Model to Pumps and Inducers. In: *Proceedings of the ISROMAC-8*, Honolulu, 2000

[5] BADER, R.: *Simulation kompressibler und inkompressibler Strömungen in Turbomaschinen*, Technische Universität München, Dissertation, 2000

[6] BADER, G. ; KRANNICH, K. D.: Einführung in das parallele Rechnen. Vorlesungsmanuskript des Lehrstuhls für numerische Mathematik und wissenschaftliches Rechnen der TU-Cottbus, 1997

[7] BALIGA, B. R.; PATANTAR S. V.: A control volume finite-element method for two-dimensional fluid flow and heat transfer. In: *Numerical Heat Transfer*, 6:245-261, 1983

[8] BARRET, R. ; BERRY, M. ; CHAN, T. F. ; DEMMEL, J. ; DONATO, J. M. ; DONGARRA J. ; EIJKHOUT, V. ; POZO, R. ; ROMINE, C. ; VAN DER VORST, H.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. 2. Auflage, SIAM books, 1993

[9] BARTH, T. J.: Aspects of unstructured grids and finite volume solvers for the Euler equations. Agard Reprot 787: Specil Course on Unstructured Grid Methods for Advection Dominated Flows, 1992

[10] BÖHM, C.: *Numerische Simulation des Fischdurchgangs durch Wasserturbinen*, Technische Universität München, Dissertation, 2004

[11] BOGNER, M. ; FLURL, B. ; SCHILLING, R.: Simulation of Cavitating Flows in Hydraulic Turbomachinery. In: CMFF'09: *The 14th International Conference on Fluid Flow Technologies*, Budapest, 2009

[12] BOOCH, G.: Object Oriented Analysis and Design with Applications. 3rd Edition, Addison-Wesley, 2007

[13] BOUSSINESQ, J.: Essai sur la Théorie des Eaux Courantes. In: *Mem. Présenté Acad. Sci.* Paris 23, 1877

[14] CHAKRAVARTHY, S.; OSHER, S.: High resolution applications of the OSHER upwind scheme for Euler equations. In: *AIAA paper 83-1943*, 1983

[15] COPLIEN, J. O.: Curiously Recurring Template Patterns. *C++ Report*: 24–27, February, 1995

[16] COUTIER-DELGOSHA, O. ; FORTES-PATELLA, R. ; REBOUD, J. L. ; HAKIMI, N.: Numerical simulation of cavitating flow in an inducer geometry. In: *4th European Conference on Turbomachinery*, Florenz, 2001

[17] CUTHILL, E. ; Mc KEE, J.: Reducing the bandwidth of sparse symmetric matrices. In: *Proceedings of the 1969 24th National Conference ACM*, S. 157–172, 1969

[18] DAREMA, F. ; GEORG, D. ; NORTON, A. ; PFISTER, G.: *A Single-Program-Multiple-Data Computational Model for EPEX-FORTRAN*, Research Report RC-11552, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1985.

[19] DONGARR, J. ; DUNIGAN, T.: *Message passing performance of various computers*, Technical Report, Oak Ride National Laboratory, University of Tennessee, 1996.

[20] DUPONT, P. ; OKAMURA, T.: Cavitating Flow Calculatins in Industry. In: *Proceedings of ISROMAC-9*, Honolulu, 2002

[21] DURBIN, P. ; PETTERSSON REIF, B.: *Statistical Theory and Modeling for Turbulent Flows*. Chichester : John Wiley & Sons, 2001

[22] EINZINGER, J.: *Numerische Simulation der Fluid-Struktur Interaktion in Turbomaschinen*, Technische Universität München, Dissertation, 2006

[23] FERZIGER, J. H. ; PERIĆ, M.: *Computational Methods for Fluid Dynamics*. 3rd Edition, Springer, Berlin, 2002

[24] FLURL, B.: Persönliche Mitteilungen. Lehrstuhl für Fluidmechanik, Technische Universität München, 2009

[25] FLURL, B. ; BOGNER M. ; SCHILLING R.: Finite Volume Method for the Simulation of the Fluid-Structure-Interaction in Arbitrary Domains. In: CMFF09: *The 14th International Conference on Fluid Flow Technologies*, Budapest, 2009

[26] FLURL, B. ; BOGNER M. ; SCHILLING R.: Finite Volume Method for the Simulation of the Fluid-Structure-Interaction in Real Life Applications. In: *ISROMAC13-2010: 13th International Symposium on Transport Phenomena and Dynamics of Rotating Machinery*, Honolulu, 2010

[27] FLYNN, M.: Some Computer Organizations and Their Effectiveness, In: *IEEE Trans. Comput.*, Vol. C-21, pp. 948, 1972.

[28] FORSYTHE, G. ; WASOW, W.: *Finite difference methods for partial differential equations*. John Willey & Sohns, Inc., New York, 1960

[29] FROBENIUS, M. ; SCHILLING, R. ; FRIEDRICHS, J. ; KOSYNA, G.: Numerical and Experimental Investigations of the Cavitating Flow in a Centrifugal Pump Impeller. In: *Proceedings of the ASME-Fluid Engineering Division Summer Meeting*, Montreal, 2002

[30] FROBENIUS, M.: *Numerische Simulation kavitierender Strömungen in hydraulischen Strömungsmaschinen*, Technische Universität München, Dissertation, 2004

[31] FROBENIUS, M. ; SCHILLING, R.: Cavitation Prediction in Hydraulic Machinery. In: *22nd IAHR Symposium on Hydraulic Machinery and Systems, Stockholm*, 2004

[32] GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, January 1995

[33] GEIST et al.: PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press Scientific and Engineering Computation, Cambridge, 1994

[34] GROPP, W. ; LUSK, E.: PVM and MPI are completely different. The Fourth European PVM - MPI Users' Group Meeting, Krakau, Polen, November 3-5, 1997

[35] HÄRDTLEIN, J.: Moderne Expression Templates Programmierung – Weiterentwickelte Techniken und deren Einsatz zur Lösung partieller Differentialgleichungen, Technische Fakultät Universität Erlangen-Nürnberg, Dissertation, 2007

[36] HARTEN, A.: High resolution schemes for hyperbolic conservation laws. In: *Journal of Computational Physics 49, Nr.3*. S. 357-393, 1983

[37] HIRSCH, C.: *Numerical Computation of Internal and External Flows*, volume 1: Fundamentals of Computational Fluid Dynamics. 2nd Edition, Wiley, 2007

[38] HOFMANN, M. ; STOFFEL, B. ; COUTIER-DELGOSHA, O. ; FORTES-PATELLA, R. ; REBOUD J. L.: Experimental and Numerical Studies on a Centrifugal Pump with 2D-Curved Blades in Cavitating Condition. In: *CAV 2001: 4th International Symposium on Cavitation*, California Institute of Technology, Pasadena, 2001

[39] HUURDEMAN, B.: *Numerische Simulation inkompressibler turbulenter Strömungen mit Mehrgitterverfahren auf unstrukturierten Gittern*, Universität Stuttgart, Dissertation, 1999

[40] ISERNHAGEN, R. ; HELMKE, H.: Softwaretechnik in C und C++, Modulare, objektorientierte und generische Programmierung. 4. Auflage, Carl Hanser Verlag, 2004

[41] JASAK, H. ; WELLER, H. ; GOSMAN, A.: High Resolution NVD Differencing Scheme for Arbitrary Unstructured Meshes, *International Journal for Numerical Methods in Fluids 31*. S.431-449, 1999

[42] JIA, R. ; SUNDÉN, B.: Parallelization of a multi-blocked CFD code via three strategies for fluid flow and heat transfer analysis, In: *Computers & Fluids* 33, S. 57-80, 2004

[43] JUSUF, J.: *Numerische Simulation der Strömung in Radseitenräumen von Turbomaschinen*, Technische Universität München, Dissertation, 2000

[44] KARP, A.: Programming for Parallelism, In: *Computer*, Vol. 20, No. 5, pp. 43-57, 1987.

[45] KHOSLA, P.; RUBIN, S.: A diagonally dominant second-order accurate implicit scheme. In: *Cumputers and Fluids 2*. S. 207-209, 1974

[46] KLIMANIS, N.: *Generic Programming and Algebraic Multigrid for Stabilized Finite Element Methods*, Georg-August-Universität zu Göttingen, Dissertation, 2006

[47] KROLL, N.: Berechnung von Strömungsfeldern um Propeller und Rotoren im Schwebeflug durch die Lösung der Euler-Gleichungen / Deutsche Forschungsanstalt für Luft- und Raumfahrt, Forschungsbereich Strömungsmechanik, Institut für Entwurfsaerodynamik, Braunschweig. 1989 – Forschungsbericht

[48] KRONSCHNABL, F.: *Numerische Strömungssimulation von Horizontalachsen-Windturbinen*, Technische Universität München, Dissertation, 2009

[49] LANGER, A. ; KREFT, K.: C++ Expression Templates, An Introduction to the Principles of Expression Templates. In: *C/C++ Users Journal*, March 2003

[50] LAUNDER, B. E. ; SPALDING, D. B.: The numerical computation of turbulent flows. In: *Computer Methods in Applied Mechanics and Engineering* 3, S.269-289, 1974

[51] LEHN, M. C.: FLENS – A Flexible Library for Efficient Numerical Solutions, Universität Ulm, Dissertation, 2008

[52] LEONARD, B.; MOKHTARI, S.: Beyond first-order upwinding: the ultra-sharp alternative for non-oscillatory steady-state simulation of convection. In: *International Journal for Numerical Methods in Engineering 30*. S.729-766, 1990

[53] LIEN, F. S. ; CHEN, W. L. ; LESCHZINER, M. A.: Low-Reynolds-number eddy-viscosity modelling based on non-linear stress-strain/ vorticity relations. In: RODI, W. et a. (Hrsg.): *Proceedings of the 3rd Int. Symposium on Engineering Turbulence Modelling and Experiments, Kreta*. Elsevier, Amsterdam u.a., S. 91-100, 1996

[54] LILEK, Ž.: *Ein Finite-Volumen-Verfahren zur Berechnung von inkompressiblen und kompressiblen Strömungen in komplexen Geometrien mit beweglichen Rändern und freien Oberflächen*, Universität Hamburg, Dissertation, 1995

[55] LILEK, Ž. ; MUZAFERIJA, S. ; PERIĆ, M. ; SEIDL, V.: An implicit finite-volume methode using nonmatching blocks of structured grid. In: *Numerical Heat Transfer*, Part B 32 (1997), S. 385-401

[56] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 1.3. University of Tennessee, Knoxville, Tennessee, May 30, 2008

[57] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 2.2. High Performance Computing Center Stuttgart (HLRS), Stuttgart, September 4, 2009

[58] MEYERS, S.: Effektiv C++ programmieren. Addison-Wesley, München, 2006

[59] MOIN, P. ; MAHESH, K.: Direct Numerical Simulation: A Tool in Turbulent Research. In: *Annu. Rev. Fluid Mech.* 30:539-78, 1998

[60] MÜLLER, A.: *Objektorientierte Strukturen für adaptive Multilevelverfahren zur Strömungssimulation,* Technische Universität München, Dissertation, 2000

[61] MUZAFERIJA, S.: *Adaptive finite volume method for flow predictions using unstructured meshes and multigrid approach*, University of London, Dissertation, 1994

[62] NOHMI, M. ; GOTO, A.: Cavitation CFD in a Centrifugal Pump. In: *Proceedings of the 5th International Symposium on Cavitation*, Osaka, 2003

[63] NOHMI, M. ; GOTO, A. ; IGA, Y. ; IKOHAGI, T.: Experimental and Numerical Study of Cavitation Breakdown in a Centrifugal Pump. In: *Proceedings of the 4th ASME-JSME Joint Fluid Engineering Conference*, Honolulu, 2003

[64] Open∇FOAM: The Open Source CFD Toolbox, OpenFOAM 1.4.1, Programmer's Guide, OpenCFD Limited, 2007

[65] PACHECO, Peter S.: Parallel Programming with MPI, University of San Francisco, Morgan Kaufmann Publishers, Inc., USA, 1997

[66] PATANKAR, S.: *Numerical heat transfer and fluid flow*. Hemisphere, New York, 1980

[67] RAITHBY, G. D.; Xu, W.-X.; Stubley G. D.: Prediction of incompressible free surface flows with an element-based finite volume method. In: *Computational Fluid Dynamics Journal*, 4:353-371, 1995

[68] RAW, M.: A Coupled Algebraic Multigrid Method for the 3D Navier-Stokes Equations. In: *AIAA-1996-297: 34th Aerospace Sciences Meeting and Exhibit*, Reno, 1996

[69] RAW, M. J. ; GALPIN P. F. ; HUTCHISON B. R. ; RAITHBY G. D. ; VAN DOORMAAL, J. P.: An element-based finite-volume method for computing viscous flows. Advanced Scientific Computing Ltd., 1994

[70] REYNOLDS, O.: On the dynamical theory of incompressible viscous fluids and the determination of the criterion. In: *Philosophical Transactions of the Royal Society of London, Series A 186*. S.123, 1895

[71] REXROTH, C.-H.: *Methoden zur effizienten Berechnung komplexer Strömungen auf unstrukturierten Gittern*, Universität Karlsruhe, Dissertation, 1996

[72] RHIE, C. ; CHOW, W.: A numerical study of the turbulent flow past an isolated airfoil with trailing edge separation. In: *AIAA Journal 21*, S.1525-1532, 1983

[73] RICHTER, R.: *3D Echtzeit-Entwurf von Beschaufelungen hydraulischer Strömungsmaschinen auf Multiprozessorsystemen*, Technische Universität München, Dissertation, 1999

[74] RIEDEL, N.: *Rotor-Stator Wechselwirkung in hydraulischen Maschinen*, Technische Universität München, Disseration, 1997

[75] SAUER, J. ; SCHNERR, G. H.: Unsteady Cavitating Flow – A New Cavitation Model Based on a Modified Front Capturing Method and Bubble Dynamics. In: *Proceedings of the ASME-Fluid Engineering Division Summer Meeting*, Boston, 2000

[76] SCHILLING, R.: Numerical calculation of the Q3D incompressible, inviscid flow in turbomachines. In: Proceedings of the 11th IAHR Symposium. Amsterdam, 1982

[77] SCHILLING, R.: CFD-aided design of hydraulic machinery bladings. In: VELENSEK, B. (Hrsg.): *CFD `91 Intensive Course on Computational Fluid Dynamics*. Ljubljana, 1991

[78] SCHILLING, R. ; BOGNER, M.: Entwicklung und experimentelle Validierung eines Codes zur numerischen Berechnung kavitierender Strömungen anderer Flüssigkeiten als Wasser in Kreiselpumpen. Abschlussbericht zum AiF Forschungsvorhaben Nr. 13759 N/1. 2006. – Forschungsbericht

[79] SCHILLING, R. ; BOGNER, M.: Kavitation in anderen Flüssigkeiten als Wasser IIa. Abschlussbericht zum Forschungsvorhaben FKM-Nr. 702282. 2007. – Forschungsbericht

[80] SCHILLING, R. ; FROBENIUS, M.: Numerical Simulation of the Two-Phase Flow in Centrifugal Pump Impellers. In: *Proceedings of the ASME-Fluid Engineering Division Summer Meeting*, Montreal, 2002

[81] SCHILLING, R. ; THUM, S. ; MÜLLER, N. ; KRÄMER, S. ; RIEDEL, N. ; MOSER, W.: Desing optimisation of hydraulic machinery blading by multi level CFD technique. In: *Proceedings of the 21st IAHR Symposium*. Lausanne, 2002

[82] SCHMIDT, D. P. ; RUTLAND, C. J. ; CORRADINI M. L.: A numerical study of cavitation flow through various nozzle shapes. In: *SAE Technical Paper Series 971597*, 1997

[83] SCHNEIDER, G. E.; RAW, M. J.: Control volume finite-element method for heat transfer and fluid flow using colocated variables – 1. Computational procedure. In: *Numerical Heat Transfer*, 11:363-390, 1987

[84] SCHNERR, G. H. ; SAUER, J.: Physical and Numerical Modeling of Unsteady Cavitation Dynamics. In: *International Conference on Multiphase Flow – ICMF 2001*, New Orleans, 2001

[85] SCHUSTER, M.: *Simulation gehäuseloser, hydraulischer Strömungsmaschinen*, Technische Universität München, Dissertation, 2000

[86] SKODA, R.: *Numerische Simulation abgelöster und transitionaler Strömungen in Turbomaschinen*, Technische Universität München, Dissertation, 2003

[87] SLEIJPEN, G. ; FOKKEMA, D.: BiCGSTAB(l) for Linear Systems involving Unsymmetric Matrices with Complex Spectrum. In: *Electronic Transactions on Numerical Analysis, 1*. S.11-32, 1993

[88] SMITH, G.: Numerical solution of partial differential equations: Finite Difference methods. 3rd Edition, Oxford: Clarendon Press, 1985

[89] STEINBRECHER, C.: *Numerische Simulation eines berührungsfrei gelagerten Rotors für eine Blutpumpe*, Technische Universität München, Dissertation, 2004

[90] STROUSTRUP, B.: Die C++ Programmiersprache. 4. aktualisierte Auflage, Addison-Wesley, München, 2000

[91] The HDF Group: HDF5 User's Guide, Release 1.8.4 November 2009, http://www.hdfgroup.org

[92] THOLE, C.-A.: Programmiersprachen für Höchstleistungsrechner: MPI und HPF, In: NAGEL, W. E. (Hrsg.): *Partielle Differentialgleichungen, Numerik und Anwendungen*. Forschungszentrum Jülich GmbH, S. 69-82, 1996

[93] THOMAS, P. ; LOMBARD C.: Geometric conservation law and ist application to flow computations on moving grids. In: *AIAA Journal 17*, S. 1030-1037, 1979

[94] THUM, S. ; SCHILLING, R.: Optimization of hydraulic machinery bladings by multilevel CFD techniques. In: *Proceedings of the 9th International Symposium on Transport Phenomena and Dynamics of Rotating Machinery*, Honolulu, 2002

[95] U.S. National Institute of Standards and Technology: Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., 17 December 2004

[96] VAN DER VORST, H.: BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. In: *SIAM Journal on Scientific and Statistical Computing 13*. S. 631-644, 1992

[97] VAN DOORMAL, J. P. ; RAITHBY, G. D.: Enhancements of the SIMPLE method for predicting incompressible fluid flows. In: *Journal of Numerical Heat Transfer*. 7:147-163, 1984

[98] VANDEVOORDE, D. ; JOSUTTIS, N.: C++ Templates – The Complete Guide. Addison-Wesley, 2003

[99] VELDHUIZEN, T. L.: Scientific Computing: C++ versus Fortran. In: *Dr. Dobb's Journal of Software Tools*, 22(11):34, 36–38, 91, 1997

[100] VELDHUIZEN, T. L.: Techniques for Scientific C++. Technical report, Indiana University Computer Science Technical Report #542, 2000

[101] VELDHUIZEN, T. L. ; JERNIGAN, M. E.: Will C++ be faster than Fortran? In: *Proceedings of the 1st International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'97)*, Lecture Notes in Computer Science. Springer-Verlag, 1997

[102] VISSER, F. C.: Some user experience demonstration the use of computational fluid dynamics for cavitation analysis and head prediction of centrifugal pumps. In: *Proceedings of the ASME-Fluid Engineering Conference*, New Orleans, 2001

[103] WILCOX, D.: *Turbulence Modeling for CFD*. DCW Industries, New York, 1998

[104] World Wide Web Consortium: Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation 26 November 2008, http://www.w3.org

[105] WU, C. H.: A general theory of the 3D flow in subsonic and supersonic turbomachines of axial, radial and mixed flow type. In: *NACA TN-2604*. 1952

[106] WUNDERER, R. Persönliche Mitteilungen. Lehrstuhl für Fluidmechanik, Technische Universität München, 2009

[107] WURSTHORN, S.: *Numerische Untersuchung kavitierender Strömungen in einer Modellkreiselpumpe*, Universität Karlsruhe, Dissertation, 2001

[108] ZWART, P. J.: *The Integrated Space-Time Finite Volume Method*, University of Waterloo, Dissertation, 1999