Otto-von-Guericke-Universität Magdeburg
Insitut für Analysis und Numerik

Diplomarbeit

Wärme- und Stoffübergang in der flüssigkeitsbedüsten Wirbelschicht

Modellierung und numerische Simulation

Eingereicht am 20. Juni 2002
von Jan Blumschein
geboren am 1. Oktober 1970
in Magdeburg

Danksagung

Für die Betreuuung dieser Arbeit möchte ich
Herrn Professor Dr. Gerald Warnecke
meinen herzlichen Dank aussprechen.

Die Entstehung dieser Arbeit wäre nicht möglich gewesen
ohne die wertvollen Anregungen, die aus den Diskussionen mit
Stefan Heinrich, Mirko Peglow, Markus Henneberg,
Wolfram Heineken, Rüdiger Müller und Thilo Moshagen
hervorgegangen sind.

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit
selbständig und nur unter Verwendung der angegebenen Literatur
angefertigt habe.

Magdeburg, 20. Juni 2002

Jan Blumschein

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Gegenstand der Arbeit

Die vorliegende Arbeit dokumentiert die Modellbildung und numerische Simulation für einen verfahrenstechnischen Prozeß.

Flüssigkeitsbedüste Wirbelschichten werden zur Trocknung, Granulation und Agglomeration von in einer Suspension enthaltenen festen Stoffen angewandt. Am Institut für Apparate- und Umwelttechnik der Otto-von Guericke-Universität werden auf Versuchsanlagen unterschiedlicher Größe Modellversuche durchgeführt. Der Vergleich der dabei gewonnenen Meßwerte mit den Ergebnissen der numerischen Simulation erlaubt eine Einschätzung der Qualität des mathematischen Prozeßmodells (bzw. eine Korrektur der geschätzten Modellparameter).

Das der Simulation zugrundeliegende Prozeßmodell beruht auf den Ausführungen in [4, 15]. Gegenüber dem in diesen Arbeiten vorgestellten Modell wurden einige Vereinfachungen sowie kleinere Korrekturen vorgenommen.

Für das resultierende nichtlineare Randwertproblem kann im Rahmen dieser Arbeit kein Beweis der Existenz und Eindeutigkeit einer Lösung gegeben werden. Bei der numerischen Simulation wird unterstellt, daß eine (eindeutige) Lösung existiert.

Die Diskretisierung des Systems von Konvektions-Diffusionsgleichungen mit nichtlinearem Reaktionsterm in drei Raumdimensionen erfolgt mittels eines Finite-Volumen-Verfahrens.

Die numerische Simulation (Lösung des diskretisierten Randwertproblems) erfolgt mit Hilfe der vom Institut für Computeranwendungen der Universität Stuttgart und dem Institut für Wissenschaftliches Rechnen der Universität Heidelberg entwickelten Software-Toolbox `ug` („unstructured grids“). Die Simulation kombiniert eine Reihe aufwendiger Techniken:

- Iterative Lösung des Randwertproblems zu einem nichtlinearen System von fünf partiellen Differentialgleichungen
- Diskretisierung in drei Raumdimensionen auf verschiedenen Gebieten (Würfel, Zylinder)
- Lokale (adaptive) Gitterverfeinerung auf unstrukturierten Gittern
- Mehrgitter-Verfahren zur Lösung des linearisierten Gleichungssystems

Die Simulation eines Anfangswertproblems wurde aus Gründen des Aufwandes nicht ausgeführt. Der praktische Wert einer instationären Modellrechnung ist insofern gering, als zeitlich *und* räumlich aufgelöste Meßwerte zum Vergleich ohnehin kaum zur Verfügung stehen. Auf die von `ug` unterstützte Möglichkeit der Parallelisierung wurde ebenfalls aus Gründen des Aufwandes (der Einarbeitung und Programmierung) verzichtet.

Die Betrachtung der numerischen Software nimmt in dieser Arbeit einen ungewöhnlich breiten Raum ein. Sie versucht übersichtliche Darstellungen des Zusammenwirkens der komplexen Algorithmen zu geben und reflektiert die bei der praktischen Umsetzung auftretenden nicht unbeträchtlichen Schwierigkeiten.

1.2 Überblick

1.2.1 Modellierung

Das in dieser Arbeit verwendete Prozeßmodell der flüssigkeitsbedüsten Wirbelschicht baut auf dem in [4, 15] ausführlich vorgestellten Modell auf. Die Wirbelschicht wird in diesem Modell bezüglich der Porosität und der strömungsmechanischen Parameter als homogen angesehen. Der Stoffübergang von der flüssigen zur Gasphase durch Verdunstung und der damit verbundene Wärme- und Stofftransport werden eingehend betrachtet.

Gegenüber dem in der Studienarbeit [4] beschriebenen Modell wurden einige Veränderungen vorgenommen:

Die temperaturabhängigen Stoffwerte in den Gleichungen werden zum großen Teil als Konstanten angenommen. Der Wärmeaustausch mit der Wand wird vernachlässigt. Wärmeleitung und Diffusion in der Gasphase werden vernachlässigt.

Die thermodynamischen Größen werden als Enthalpien bilanziert, die erhaltenen Gleichungen werden anschließend in Temperatur-Koordinaten transformiert, wodurch die vorgenommenen Vereinfachungen klarer erkennbar werden.

Der aus der Flüssigkeits-Eindüsung stammende Quellterm wurde geringfügig modifiziert, wodurch die Singularität im Düsenzentrum beseitigt ist.

Mit Rücksicht auf die vorgenommenen Veränderungen wird in dieser Arbeit nochmals die vollständige Herleitung der Modellgleichungen aus Bilanzen wiedergegeben. Die Modellbildung führt auf ein System von fünf partiellen Differentialgleichungen, die über nichtlineare Reaktionsterme gekoppelt sind.

In dieser Arbeit wird nur das stationäre Problem (Spezialfall des Modells für verschwindende zeitliche Ableitungen) betrachtet. Ein Beweis der Existenz und Eindeutigkeit einer Lösung des resultierenden Randwertproblems kann im Rahmen dieser Arbeit nicht gegeben werden. Für ein weiter reduziertes Modellproblem werden die Eigenschaften möglicher Lösungen diskutiert.

1.2.2 Diskretisierung, Iterative Lösung

Diskretisierung. Der Abschnitt 3 erklärt die Diskretisierung des nichtlinearen Randwertproblemes in $\mathsf{N} = 3$ Raumdimensionen. Da Finite-Volumen-Verfahren häufig nur für den Fall $\mathsf{N} = 2$ erklärt werden, wird die hier verwendete Diskretisierung ausführlich beschrieben.

Die verwendete Finite-Volumen-Diskretisierung läßt sich unabhängig von der zugrundeliegenden konkreten Element-Geometrie darstellen und ist daher auf unstrukturierten Gittern, in denen verschiedene Elementtypen in unspezifizierter Anordnung vorliegen, besonders übersichtlich umzusetzen.

Es ist bekannt, daß die für Finite-Volumen-Verfahren „natürliche“ Upwind-Diskretisierung eine stabile Diskretisierung der Konvektionsterme liefert. Der Approximationsfehler der Finite-Volumen-Diskretisierung ist nur von erster Ordnung in der Gitterweite. Da aber aus Gründen des begrenzten Speicherplatzes eine hohe räumliche Auflösung nur in Teilen des Gitters (durch lokale Gitterverfeinerung) erreicht werden kann, ist die für ein gleichförmig verfeinertes Gitter definierte *asymptotische* Fehler*ordnung* nur von sekundärem Interesse.

3D-Geometrie. Das Konzept der Beschreibung der geometrischen Objekte (Gebiet und Rand, Elemente, Gitterverfeinerung...) in drei Raumdimensionen in ug stellt eine unmittelbare Erweiterung der zweidimensionalen Strukturen dar. Die dreidimensionale Erweiterung bringt einige Besonderheiten mit sich, die in der ug-Dokumentation keine oder nur unzureichende Erwähnung finden. An dieser Stelle wäre, soweit es sich nicht um spezifische Probleme der Software-Umsetzung handelt, ein Verweis auf diesbezügliche Literatur angebracht.

Einige Punkte, die bei der dreidimensionalen Diskretisierung zu beachten sind (verfügbare Referenzelemente und -transformationen, Formfunktionen, Numerierung der Knoten; Parametrisierung des Randes, Kompatibilitätsbedingungen zur Gebietszerlegung und Gitterverfeinerung) werden in dieser Arbeit ausführlich betrachtet. Diese (bei weitem nicht vollständigen) Erklärungen können als Ergänzung in die ug-Dokumentation aufgenommen werden.

Iterative Lösung. Im Abschnitt 4 werden die zur numerischen Lösung des diskreten nichtlinearen Gleichungssystems verwendeten Algorithmen (linearer Mehrgitterzyklus, Newton-Iteration) behandelt. Da es sich um bekannte Algorithmen handelt, die in Standardwerken zur Numerik ausführlich beschrieben werden (hier werden die Standardwerkzeuge von ug verwendet), ist dieser Abschnitt relativ knapp gehalten. Die Algorithmen werden lediglich formal skizziert, um erkennbar zu machen, welche von ggf. mehreren Varianten verwendet wurde. Die Benennung der (einstellbaren) Parameter stellt einen Bezug zur softwareorientierten Beschreibung im folgenden Abschnitt her.

1.2.3 Software

Numerische Prozeduren. Während sich z.B. im Standardwerk von Hackbusch [14] noch Quelltext-Darstellungen von Mehrgitteralgorithmen im Umfang einer Textseite finden, ist eine solche Darstellung für die komplexen Algorithmen bei der Nutzung adaptiv verfeinerter unstrukturierter Gitter nicht mehr möglich. Die Tatsache, daß die zur numerischen Lösung (Approximation) eines komplexen mathematischen Problems verwendete umfangreiche Software von einer einzelnen Person nicht mehr vollständig überblickt werden kann, steht im Widerspruch zu der in der Mathematik geforderten methodischen Strenge.

Es stellt sich die Frage nach Möglichkeiten, für das Zusammenwirken und die Parametrisierung der komplexen numerischen Algorithmen eine übersichtliche und aussagekräftige Darstellung zu geben. Der Abschnitt 5 versucht, auf diese Frage eine Antwort zu finden.

Beim Hinzufügen eigener Bausteine zur ug-Software (dies war erforderlich, da für die Finite-Volumen-Diskretisierung eines nichtlinearen Systems keine geeignete Assemble-Routine zur Verfügung steht) müssen vom Anwender die Interface-Konventionen zu den bereits bestehenden Komponenten beachtet werden. Leider werden diese Konventionen in der Dokumentation nicht erklärt.[1]

Bei der Erforschung der ug-internen Mechanismen stößt man in verschiedenen Bereichen auf *objektorientierte* Strukturen (in der Programmiersprache C, nicht C++!) Als „natürliche“ Beschreibungsform für solche Strukturen ist seit etwa einem halben Jahrzehnt die *Unified Modeling Language (UML)* zu einer quasi standardisierten Sprache geworden.

[1] Die Dokumentation beschreibt die zugrundeliegenden Datenstrukturen (wobei einige Details einer Aktualisierung bedürfen), sie weist jedoch nicht darauf hin, auf welche Weise *nicht* auf diese zugegriffen werden soll, und erklärt nicht, warum, wie und in welcher Reihenfolge bestimmte Programmteile aufzurufen sind.

Aufbauend auf die Beschreibung einiger grundlegender Mechanismen sowie auf die Einführung von Grundbegriffen der Objektorientierung wird im Abschnitt 5 die ug-spezifische Realisierung der *numerischen Prozeduren* als objektorientierte Struktur interpretiert, in deren Benutzung die oben erwähnte, undokumentierte Interface-Konvention besteht. Der statische Zusammenhang und die Parametrisierung der *numproc-Klassen* werden anhand von UML-Klassendiagrammen veranschaulicht und für einige Beispiele im Detail beschrieben.

Bug report. Als Anhang D wurde die Dokumentation einer Anzahl von Fehlern in der ug-Software aufgenommen, die im Laufe der Zeit gesammelt wurden. Die Beschreibung der Fehler schließt Umstände und Ursachen des Auftretens ebenso wie Möglichkeiten der Beseitigung ein und kann daher unmittelbar zur Verbesserung der Software dienen. Zugleich gibt die detaillierte Dokumentation einen Eindruck vom Zeitaufwand, der für die Bewältigung dieser Probleme bei der Nutzung des Programmpaketes ug erforderlich ist.

Diese Zusammenstellung von Fehlern stellt nicht die Qualität der in ug implementierten numerischen Algorithmen in Frage: Der bei weitem überwiegende Teil der „ernsten“ Probleme (Programmabstürze) tritt *nicht* in den *numerischen Prozeduren*, sondern in den diesen „umgebenden“ Programmteilen auf. Als besonders verbesserungswürdig erwiesen sich

- die Verwaltung der *drei*dimensionalen geometrischen Strukturen (Gebietsbeschreibung),
- die Grafikausgabe,
- der Kommandointerpreter bzw. die Funktionalität der verfügbaren Kommandos.

Bei den numerischen Prozeduren macht sich lediglich ein Mangel an aussagekräftiger Dokumentation unangenehm bemerkbar.

Es bleibt anzumerken, daß sich sämtliche Ausführungen in dieser Arbeit auf die „aktuelle“ (d.h. letzte veröffentlichte) Version ug-3.8 von 1998 beziehen. Für 2002 ist vom IWR Heidelberg eine überarbeitete Version der Software angekündigt.

1.2.4 Modellrechnungen und Ergebnisse

Für das reduzierte Modellproblem von $\mathsf{M} = 2$ Gleichungen wurden mehrere Simulationen mit unterschiedlichen Quell-Funktionen ausgeführt, die die im Abschnitt 2.6.3 untersuchten nichtlinearen Eigenschaften des Modells belegen. Darüber hinaus wird die Diskretisierung auf verschiedenen dreidimensionalen Gebieten (Würfel, Zylinder) demonstriert.

Die Simulation des vollständigen (stationären) Modells wurde mit den Betriebsdaten von zwei Versuchen ausgeführt, bei denen auf Wirbelschicht-Anlagen verschiedener Größe Meßdaten der dreidimensionalen Temperaturverteilung aufgezeichnet wurden.

Der Vergleich der Simulationsergebnisse mit den Meßwerten bestätigt die grundlegende Übereinstimmung zwischen Modell und Prozeßverhalten. Die nicht zu vernachlässigenden Abweichungen zwischen Meß- und Rechenwerten weisen darauf hin, daß die Approximationsformeln für einige der Koeffizienten des Modells, insbesondere für die Dispersion (Feststoffvermischung), einer Korrektur bedürfen.

2 Modellierung

2.1 Ingenieurphysikalische Grundlagen

2.1.1 Wirbelschicht

Wirbelschichtreaktor. Der *Wirbelschichtreaktor* ist ein Behälter, in dem sich als *Bettmaterial* Feststoffpartikel einer geeigneten Körnung befinden. Über einen *Anströmboden* wird von unten Luft zugeführt.

Liegt die Strömungsgeschwindigkeit im Reaktor zwischen den für das Bettmaterial charakteristischen Werten von *Minimalfluidisierungsgeschwindigkeit* und *Partikelaustragsgeschwindigkeit*, so bildet sich im Reaktor eine *Wirbelschicht* mit Fluideigenschaften aus.

Anwendung zur Agglomeration/Granulation/Trocknung. Mittels einer *Düse* wird eine *Suspension* in die Wirbelschicht eingebracht. Die Tropfen haften zunächst auf den Feststoffpartikeln; der Wasseranteil verdunstet, während der Feststoffanteil zur Vergrößerung der Partikel beiträgt.

2.1.2 Phasengemisch

Der Raum der Wirbelschicht wird von einem Gemisch mehrerer Phasen ausgefüllt:

Der **Feststoff** liegt in Form relativ grobkörniger Partikel vor. Die in die Wirbelschicht eingedüste **Flüssigkeit** haftet als Tropfen auf der Oberfläche der Partikel. Der Raum zwischen den benetzten Partikeln wird von der **Gasphase**, im Falle der Wirbelschichttrocknung einem Gemisch aus Luft und Wasserdampf eingenommen.

Kontinuumsmodell. Für das mathematische Modell wird vereinfachend eine ideale Vermischung der Phasen angenommen: in einem beliebig kleinen Volumen sind Anteile aller drei Phasen anzutreffen. Die Grobkörnigkeit der Feststoffpartikel wird damit ignoriert.

Relatives Lückenvolumen. Der Anteil der Gasphase am Gesamtvolumen wird durch das *relative Lückenvolumen* ε bezeichnet. Es hängt von strömungsmechanischen Größen ab. Das Komplement von ε ist der *Feststoffanteil* $\overline{\varepsilon} = 1 - \varepsilon$. (Das Volumen des Flüssigkeitsfilms wird hier vernachlässigt.)

Partikeloberfläche. Für Verdunstung und Wärmeübergang ist die Größe der zur Verfügung stehenden Phasengrenzfläche ein entscheidender Parameter. Zu ihrer Beschreibung wird die *volumenbezogene Partikeloberfläche*

$$A^* = \frac{dA}{dV}$$

(Gutsoberfläche bezogen auf Gutsvolumen) eingeführt. Für monodisperses Bettmaterial (d. i. einheitliche Körnung) mit dem Partikeldurchmesser d_P gilt

$$A^* = \frac{dA}{dV} = \frac{\pi d_P^2}{\frac{1}{6}\pi d_P^3} = \frac{6}{d_P}.$$

Die pro Volumenelement der Wirbelschicht zur Verfügung stehende Partikeloberfläche A_{eff} ergibt sich durch Multiplikation mit dem Feststoffanteil $\overline{\varepsilon}$:

$$A_{\mathit{eff}} = \overline{\varepsilon} A^*$$

Dichte und Konzentration. Die *Dichte* ρ_i gibt an, welche Masse der Komponente i ein gegebenes Volumen unter den herrschenden Bedingungen (Druck, Temperatur) *allein* ausfüllen würde. Die *Konzentration* κ_i bezeichnet die im Stoffgemisch tatsächlich vorliegende Masse der Komponente i pro Volumeneinheit:

$$\kappa_i = \frac{dm_i}{dV}$$

So ist z.B. für das Bettmaterial die Konzentration κ_P um den Faktor $\overline{\varepsilon}$ kleiner als die Dichte: es gilt $\kappa_P = \overline{\varepsilon}\rho_P$.

Benetzte Partikel. Die eingedüste Flüssigkeit haftet in Form von Tropfen auf den Feststoffpartikeln der Wirbelschicht. Aufgrund der Oberflächenspannung breitet sich der Tropfen auf der Partikeloberfläche aus, wenn eine bestimmte Dicke erreicht ist. Daher kann vereinfachend eine konstante Filmdicke d_F (z.B. $1\mu m$) der Flüssigkeit angenommen werden.

Bei vollständiger Benetzung der Partikel mit einem Flüssigkeitsfilm der angenommenen Dicke d_F ergibt sich die *maximale Flüssigkeitskonzentration* $\kappa_{F,max} = A_{\mathit{eff}} d_F \rho_F$.
Der *Benetzungsgrad*

$$\varphi = \frac{A_{benetzt}}{A_{benetzt} + A_{unbenetzt}} = \frac{\kappa_F}{\kappa_{F,max}} = \frac{\kappa_F}{A_{\mathit{eff}} d_F \rho_F} \tag{1}$$

bezeichnet (unter obiger Annahme) das Verhältnis von benetzter Partikeloberfläche zur Gesamtoberfläche, das Komplement $\overline{\varphi} = 1 - \varphi$ entsprechend den Anteil der unbenetzten Fläche.

2.1.3 Thermodynamische Zustandsgrößen

Konstanter Druck. Die Druckdifferenzen in der Wirbelschicht sind derart gering, daß für die Thermodynamik der Druck überall gleich dem Atmosphärendruck gesetzt werden kann. (Für die strömungsmechanische Betrachtung der Wirbelschicht ist der Druckabfall über der Wirbelschicht von entscheidender Bedeutung.)

Alle Zustandsänderungen von Luft und Dampf finden somit bei konstantem Druck p statt; die Prozesse werden durch die Bilanzierung von *Enthalpien* beschrieben.

Ideale Gase. Im Betriebszustand des Wirbelschichtreaktors ist die Beschreibung der Luft als ideales Gas hinreichend genau: die temperatur- und druckabhängige *Dichte* ρ_L wird durch

$$\rho_L = \frac{p}{R_L T}$$

beschrieben, die *Gaskonstante* $R_L = R/M_L$ ist stoffabhängige Konstante.

Eine genauere Berücksichtigung der Stoffeigenschaften ist für das verdunstende Wasser (Abhängigkeit des Sättigungsdampfdruckes von der Temperatur) erforderlich.

Feuchte Luft. Die *feuchte Luft* ist ein Gemisch aus Luft und Wasserdampf. Die *Luftfeuchte* (oder der *Feuchtigkeitsgehalt*) Y_L ist definiert als

$$Y_L = \frac{\kappa_D}{\kappa_L}.$$

Die Aufteilung der Gasphase unter den einzelnen Komponenten wird durch Partialdrücke p_i beschrieben. Der (konstante) Systemdruck p ergibt sich nach dem DALTONschen Gesetz als Summe der einzelnen Partialdrücke. In einer Mischung idealer Gase gilt für den Partialdruck der Komponente i

$$p_i = \kappa_i R_i T, \qquad \frac{p_i}{p} = \frac{\kappa_i}{\rho_i}.$$

Da der Gasphase in der Wirbelschicht ein um den Faktor ε kleineres Volumen zur Verfügung steht, ist κ_i/ε anstelle von κ_i einzusetzen:

$$p_D = \frac{\kappa_D}{\varepsilon} R_D T, \qquad \frac{p_D}{p} = \frac{\kappa_D}{\varepsilon \rho_D}.$$

(p_L analog.) Für den Systemdruck folgt aus $p = p_L + p_D$

$$\varepsilon p = (\kappa_L R_L + \kappa_D R_D) T.$$

Enthalpien. Die Wirbelschicht wird als isobares System modelliert. Wir bilanzieren die *volumenbezogenen Enthalpien* h_i der einzelnen Phasen i, die insbesondere von den *Temperaturen* ϑ_i abhängen.

Die *spezifischen Wärmekapazitäten bei konstantem Druck* c_{pi} werden der Einfachheit halber als konstant angenommen, der Nullpunkt der Enthalpieskalen wird *(hier)* bei Atmosphärendruck und 0°C für trockene Luft bzw. flüssiges Wasser gesetzt.

Die volumenbezogenen Enthalpien der festen bzw. flüssigen Phase sind damit

$$h_P = \kappa_P c_{pP} \vartheta_P = \overline{\varepsilon} \rho_P c_{pP} \vartheta_P; \qquad h_F = \kappa_F c_{pF} \vartheta_F.$$

Die Enthalpie der feuchten Luft setzt sich aus der Enthalpie beider Komponenten zusammen, für den Dampf ist die *spezifische Verdampfungsenthalpie* Δh_{V0} des Wassers bei 0°C zu berücksichtigen. Die volumenbezogene Enthalpie h_L der feuchten Luft ergibt sich damit zu

$$h_L = \kappa_L c_{pL} \vartheta_L + \kappa_D c_{pD} \vartheta_L + \kappa_D \Delta h_{V0}.$$

Die Konzentration der (trockenen) Luft κ_L ist durch

$$p_L(\kappa_L, \vartheta_L) + p_D(\kappa_D, \vartheta_L) = p$$

implizit als Funktion von κ_D und ϑ_L bestimmt. κ_L erscheint im Modell nur in der Form $\kappa_L c_{pL} + \kappa_D c_{pD}$, wofür die Abkürzung $\kappa_Y c_{pY}$ eingeführt wird: Mit $\kappa_Y = \kappa_L + \kappa_D$ wird c_{pY} als spezifische Wärmekapazität der feuchten Luft definiert. Der Term

$$\kappa_Y c_{pY} = \kappa_L c_{pL} + \kappa_D c_{pD} = \frac{\varepsilon}{T_L} \left(p_L \frac{c_{pL}}{R_L} + p_D \frac{c_{pD}}{R_D} \right)$$

wird durch die Näherung

$$\frac{c_{pD}}{R_D} \approx \frac{c_{pL}}{R_L}$$

vereinfacht zu

$$\kappa_Y c_{pY} \approx \frac{\varepsilon}{T_L} p \frac{c_{pL}}{R_L} = \varepsilon \rho_L c_{pL},$$

wodurch sich die Ausrechnung von κ_L erübrigt.

2.1.4 Wärme- und Stofftransport

Strömung. Durch die von unten zugeführte Trägerluft bildet sich im Reaktor ein Strömungsfeld aus. Für die Modellierung wird angenommen, daß die Luft den Reaktor mit konstanter Geschwindigkeit w_L in vertikaler Richtung durchströmt. Dichteänderungen der Luft, Turbulenzen und der Massenzuwachs durch Verdunstung bleiben dabei unberücksichtigt.

Dispersion. Durch den Transport in der turbulenten Luftströmung erfolgt eine Durchmischung (Dispersion) des Bettmaterials. Dabei dominiert der Partikeltransport in vertikaler Richtung.

Die richtungsabhängige Intensität der Feststoffdispersion wird durch die *Dispersionsmatrix*

$$\underline{\underline{D}} = \begin{pmatrix} D_x & 0 & 0 \\ 0 & D_y & 0 \\ 0 & 0 & D_z \end{pmatrix}$$

in Diagonalgestalt beschrieben. Nach [15] wird lediglich der Dispersionskoeffizient in vertikaler Richtung D_z berechnet, der Dispersionskoeffizient in horizontaler Richtung $D_x = D_y$ wird zu $D_x = D_y = 0,1 D_z$ geschätzt.

2.1.5 Wärme- und Stoffübergang zwischen den Phasen

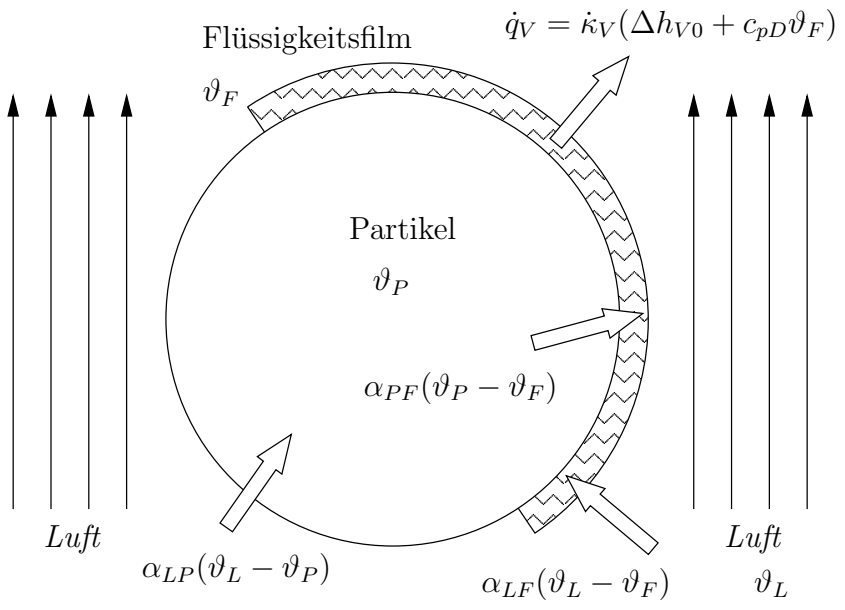


Abbildung 1: Wärmeübergang am Flüssigkeitsfilm

Wärmeübergang an den Grenzflächen. Über die Grenzflächen der einzelnen Phasen PL (Partikel–Luft), PF (Partikel–Flüssigkeitsfilm), FL (Flüssigkeitsfilm–Luft) wird durch das Zusammenwirken verschiedener Mechanismen (Wärmeleitung, Konvektion, Strahlung) Wärme transportiert. Der *(flächenbezogene) Wärmestrom*

$$\dot{q}_{ij}^{\square} = \frac{d\dot{Q}_{ij}}{dA}$$

über die Phasengrenzfläche A_{ij} (von i nach j) wird durch den *Wärmeübergangskoeffizienten* α_{ij} beschrieben:

$$\dot{q}_{ij}^{\square} = \alpha_{ij}(\vartheta_i - \vartheta_j)$$

Der *volumenbezogene Wärmestrom* $\dot{q}_{ij}$ ergibt sich durch Multiplikation mit der pro Volumenelement zur Verfügung stehenden Phasengrenzfläche:

$$\begin{aligned}
\dot{q}_{PF} &= A_{\mathit{eff}}\varphi\dot{q}_{PF}^{\square} = A_{\mathit{eff}}\varphi\alpha_{PF}(\vartheta_P - \vartheta_F) \\
\dot{q}_{FL} &= A_{\mathit{eff}}\varphi\dot{q}_{FL}^{\square} = A_{\mathit{eff}}\varphi\alpha_{FL}(\vartheta_F - \vartheta_L) \\
\dot{q}_{PL} &= A_{\mathit{eff}}\overline{\varphi}\dot{q}_{PL}^{\square} = A_{\mathit{eff}}\overline{\varphi}\alpha_{PL}(\vartheta_P - \vartheta_L)
\end{aligned}$$

Die im allgemeinen temperaturabhängigen Werte von α werden in dieser Arbeit als konstant angenommen.

Verdunstungsstrom. In der Wirbelschicht findet eine adiabate Sättigung von Luft mit Wasserdampf statt. Der Massenstrom $\dot{m}_V$ des Wassers, der an der Phasengrenzfläche A zwischen feuchter Luft und Wasser verdunstet (bzw. kondensiert für $\dot{m}_V < 0$), hängt nach [8] vom aktuellen Dampfdruck p_D und dem Sättigungsdampfdruck $p_{D,\infty}$ ab:

$$\dot{m}_V = \beta A \rho_D \ln \frac{p - p_D}{p - p_{D,\infty}}$$

Der Systemdruck p wird als konstant angenommen (Atmosphärendruck). Der *Dampfdruck* der feuchten Luft p_D hängt mit der Dampfkonzentration κ_D über $p_D = \frac{\kappa_D}{\varepsilon} R_D T$ zusammen.

Der *Sättigungsdampfdruck* $p_{D,\infty}$ des Wassers ist eine Funktion der Temperatur, die sich im relevanten Temperaturbereich ausreichend genau durch einen Ansatz der Form

$$p_{D,\infty}(\vartheta) = C_1 \exp \frac{C_2}{\vartheta + C_3}$$

(„Antoine-Gleichung“) approximieren läßt.

Die *Stoffübergangszahl* β beschreibt den Einfluß der Partikelumströmung auf den Transport des Wasserdampfes. Sie hängt von Lufttemperatur und Strömungsgeschwindigkeit ab (siehe [4, 15]), diese Abhängigkeiten werden hier vernachlässigt.

Die auf das Volumenelement bezogene benetzte Oberfläche ist $A_{\mathit{eff}}\varphi$, zusammen mit (1) ergibt sich

$$\dot{\kappa}_V = \beta A_{\mathit{eff}}\varphi\rho_D \ln \frac{p - p_D}{p - p_{D,\infty}} = \frac{\beta\rho_D}{d_F\rho_F}\kappa_F \ln \frac{p - p_D}{p - p_{D,\infty}}$$

für den Verdunstungsstrom (verdunstenden Massenstrom) je Volumeneinheit.

Enthalpiestrom des verdunstenden Wassers. Es wird angenommen, daß die zur Verdunstung notwendige Wärme $\dot{\kappa}_V(\Delta h_{V0} + (c_{pD} - c_{pF})\vartheta_F)$ dem Flüssigkeitsfilm entnommen wird, die zur Erwärmung von ϑ_F auf ϑ_L erforderliche Wärme hingegen der Luft entstammt. Unter dieser Voraussetzung entzieht der verdunstende Wasserstrom $\dot{\kappa}_V$ der verbleibenden flüssigen Phase den (volumenbezogenen) Wärmestrom

$$\dot{q}_V = \dot{\kappa}_V(\Delta h_{V0} + c_{pD}\vartheta_F).$$

2.1.6 Flüssigkeitseintrag

Die Flüssigkeit, die auf der Oberfläche der Bettpartikel verdunstet, wird kontinuierlich über eine *Düse* in die Wirbelschicht eingebracht. Im folgenden wird von der Verwendung einer im Inneren der Wirbelschicht befindlichen *Einstoffdüse* ausgegangen.

Die zugeführte Flüssigkeit verläßt die Düse in Form von Tropfen. Vereinfachend wird angenommen, daß jeder einzelne Tropfen sich so lange auf einer geradlinigen Bahn fortbewegt, bis er auf einem Korn der Wirbelschicht haften bleibt. Die gegenseitige Beeinflussung von Luft- und Tropfenstrom wird vernachlässigt.

Die Wahrscheinlichkeit der Flüssigkeitsabscheidung beim Zusammentreffen mit einem Feststoffpartikel wird durch den Abscheidegrad η_A modelliert. Mit der Porosität ε ergibt sich daraus eine mittlere Tropfenweglänge s_{Tr}. Die Kenntnis der mittleren Tropfenweglänge ermöglicht eine Berechnung der räumlichen Verteilung des eingedüsten Flüssigkeits-Massenstroms $\dot{m}_Q$ und damit des (volumenbezogenen) *Flüssigkeits-Quellstroms* $\dot{\kappa}_Q(\underline{x})$.

2.1.7 Vereinfachende Annahmen

Im folgenden werden die für die Modellierung getroffenen vereinfachenden Annahmen zusammengefaßt und um einige Punkte ergänzt.

Homogenes Bettmaterial. Die Feststoffpartikel in der Wirbelschicht werden als homogene Kugeln mit einem konstanten Durchmesser *(Korngröße)* d_P modelliert.

Pfropfenströmung. Die Fluidisierungsluft strömt mit einer räumlich und zeitlich konstanten Geschwindigkeit in vertikaler Richtung durch die Wirbelschicht. Eine Diffusion der Luft durch Verwirbelung findet nicht statt.

Der Druckabfall über der Wirbelschicht wird vernachlässigt.

Homogene Wirbelschicht. Das von der Wirbelschicht ausgefüllte Volumen ist unveränderlich. Die Eigenschaften der Wirbelschicht (Porosität, richtungsabhängiger Dispersionskoeffizient) sind räumlich und zeitlich konstant.

Vernachlässigung des Flüssigkeitsvolumens. Das Volumen der flüssigen Phase (Düsenstrahl und Flüssigkeitsfilm) wird vernachlässigt.

Düsenstrahl. In Ermangelung geeigneter Modelle wird die Wechselwirkung zwischen dem Strom der Trägerluft und dem aus der Düse austretenden Flüssigkeitsstrahl vernachlässigt: Luftstrom und Düsenstrahl lenken einander nicht ab. Zwischen Luftstrom und Düsenstrahl erfolgt kein Wärmeaustausch. Die im Düsenstrahl transportierten Tröpfchen verdunsten nicht.

2.2 Ableitung der Modellgleichungen aus Bilanzen

Im folgenden werden durch Bilanzierung ausgewählter Größen *Bilanzgleichungen* aufgestellt, die die Grundlage des mathematischen Modells der Wirbelschichttrocknung bilden.

Die Bilanzierung im Inneren des Gebietes Ω (der Wirbelschichtzone) liefert ein System partieller Differentialgleichungen, die Bilanzierung an der Randfläche $\partial\Omega$ liefert die zugehörigen Randbedingungen.

2.2.1 Integrale Bilanz

Als *Bilanzgröße* bezeichnen wir eine (lokal) integrierbare Funktion $u : \Omega \times [0,T] \to \mathbb{R}$. Physikalisch gesehen sei u eine *volumenbezogene extensive Größe* (eine *Dichte*), für die das einem Volumen G zugeordnete Quantum $U(G) := \int_G u dx$ sinnvoll zu definieren ist.

Der *Transport* der Größe u wird durch die vektorwertige *Stromfunktion* $\underline{s}(u)$ beschrieben. Der skalare *Quellterm* $\dot{u}_Q$ bezeichnet die Erzeugung oder Vernichtung von u.

Eine *(integrale) Bilanz* auf dem Teilgebiet $G \subseteq \Omega$ hat unter Berücksichtigung dieser Terme die allgemeine Form

$$\underbrace{\frac{d}{dt}\int_G u dx}_{\substack{\text{zeitliche}\\ \text{Änderung der}\\ \text{Menge von } u\\ \text{in } G}} = \underbrace{\int_G \dot{u}_Q dx}_{\substack{\text{in } G\\ \text{erzeugte}\\ \text{Menge}\\ \text{von } u}} - \underbrace{\int_{\partial G} \underline{s}(u)\cdot\underline{n} dx}_{\substack{\text{über den}\\ \text{Rand von } G\\ \text{tretender}\\ \text{Strom}}}. \tag{2}$$

Der vektorwertige *Strom* $\underline{s}(u)$ kann durch verschiedene Mechanismen angetrieben werden:

Advektion mit dem Geschwindigkeits-Vektorfeld $\underline{w}$:

$$\underline{s}_a(u) = u\underline{w}$$

Wärmeleitung mit dem Wärmeleitkoeffizienten λ (FOURIERsches Gesetz):

$$\underline{s}_l(u) = -\lambda\nabla u$$

Dispersion (Verwirbelung) im allg. richtungsabhängig, mit der Dispersionsmatrix $\underline{\underline{D}}$:

$$\underline{s}_d(u) = -\underline{\underline{D}}\nabla u$$

2.2.2 Bilanz in Divergenzform

Für die folgenden Betrachtungen seien u und $\underline{s}(u)$ stetig differenzierbar. Auf der linken Seite von (2) können Ableitung und Integral vertauscht werden, da das Integrationsgebiet nicht von t abhängt, die rechte Seite kann unter Anwendung des Gaußschen Satzes umgeformt werden: (2) ist äquivalent zu

$$\int_G \frac{\partial u}{\partial t} dx = \int_G \dot{u}_Q dx - \int_G \nabla\cdot\underline{s}(u) dx.$$

Soll diese Gleichung auf jedem $G \subseteq \Omega$ gelten, so muß die (differentielle) *Bilanz in Divergenzform*

$$\underbrace{\frac{\partial u}{\partial t}}_{\substack{\text{zeitliche} \\ \text{Änderung der} \\ \text{Konzentration} \\ \text{von } u}} = \underbrace{\dot{u}_Q}_{\substack{\text{pro Volumen} \\ \text{erzeugte} \\ \text{Menge von u}}} - \underbrace{\nabla \cdot \underline{s}(u)}_{\substack{\text{Divergenz} \\ \text{des Stromes} \\ \text{von u}}} \tag{3}$$

punktweise fast überall auf Ω erfüllt sein.

2.2.3 Bilanzierung in Lagrange-Koordinaten

Für ein Fluid, dessen Bewegung durch ein differenzierbares Vektorfeld beschrieben werden kann, bietet sich eine alternative Möglichkeit der Bilanzierung. Durch die Wahl eines mit dem Fluid bewegten Koordinatensystems wird der Advektionsterm zum Verschwinden gebracht.

Euler- und Lagrange-Koordinaten. In ortsfesten *Euler-Koordinaten* $(\underline{x}, t)$ wird die Bilanzgröße durch

$$u : (\underline{x}, t) \mapsto u(\underline{x}, t)$$

dargstellt; die Bilanz (3) bezieht sich auf diese Koordinatenwahl.

Der Transport der Größe u werde durch einen Advektionsterm $\underline{s}(u) = u\underline{w}$ beschrieben. Sei

$$\begin{array}{rccc} \underline{X}: & \Omega \times [0,T] & \to & \mathbb{R}^{\mathsf{N}} \\ & (\underline{\xi}, t) & \mapsto & \underline{X}(\underline{\xi}, t) \end{array}$$

ein stetig differenzierbares Vektorfeld mit $\underline{X}(\underline{\xi}, t_0) = \underline{\xi}$ für ein festes t_0 derart, daß die zeitliche Ableitung

$$\left.\frac{\partial \underline{X}}{\partial t}\right|_{(\underline{\xi},t)} = \underline{w}(\underline{X}(\underline{\xi}, t), t)$$

gerade die Strömungsgeschwindigkeit $\underline{w}(\underline{x}, t)$ des Fluids im Punkt $\underline{x} = \underline{X}(\underline{\xi}, t)$ in Euler-Koordinaten ist. Die Darstellung der Bilanzgröße u in *Lagrange-Koordinaten* $(\underline{\xi}, t)$ ist durch

$$u^{\underline{X}} : (\underline{\xi}, t) \mapsto u(\underline{X}(\underline{\xi}, t), t) \tag{4}$$

gegeben. Das Fluid ruht bezüglich der *Lagrange-Koordinaten* $(\underline{\xi}, t)$.

Bewegtes Kontrollvolumen. Ein bezüglich der Lagrange-Koordinaten festes Kontrollvolumen $\{(\underline{\xi}, t) : \underline{\xi} \in G_0\}$ wird bezüglich der Euler-Koordinaten als

$$G(t) = \underline{X}(G(t_0), t)$$

dargestellt: das Kontrollvolumen bewegt sich in der Strömung mit.

Der Transport des Fluids werde durch einen Advektionsterm $\underline{s}(u) = u\underline{w}$ beschrieben, damit wird die Bilanz (2) über dem ortsfesten Kontrollvolumen G zu

$$\frac{d}{dt}\int_G u dx = \int_G \dot{u}_Q dx - \int_{\partial G} (u\underline{w}) \cdot \underline{n} dx \tag{5}$$

Die hierzu analoge Bilanz über dem bewegten Kontrollvolumen $G(t)$ hat die Form

$$\frac{d}{dt}\int_{G(t)} u dx = \int_{G(t)} \dot{u}_Q dx. \tag{6}$$

Die Bilanzen (5) und (6) können als äquivalente Axiome[2] aufgefaßt werden, die durch den **Reynoldsschen Transportsatz** für stetig differenzierbare $u, \underline{w}$

$$\frac{d}{dt}\int_{G(t)} u dx = \int_{G(t)} \left[\frac{\partial u}{\partial t} + \nabla \cdot (u\underline{w})\right] dx \tag{7}$$

ineinander übergehen (zum Beweis des Transportsatzes siehe z.B. [6, 19]).

Anschaulich ist das Verschwinden des Transporttermes damit zu begründen, daß der Gebietsrand $\partial G(t)$ relativ zum Strömungsfeld unbeweglich ist.

> Zum Beweis der Äquivalenz von (5) und (6): Die Bilanz (5) ist nach der Argumentation in 2.2.2 für stetig differenzierbare u, $\underline{w}$ und ein festes (von t unabhängiges) Integrationsgebiet $G(t_0)$ äquivalent zu
>
> $$\int_{G(t_0)} \frac{\partial u}{\partial t} dx + \int_{G(t_0)} \nabla \cdot (u\underline{w}) dx = \int_{G(t_0)} \dot{u}_Q dx,$$
>
> dies ist nach dem Reynoldsschen Transportsatz (7) äquivalent zu
>
> $$\left[\frac{d}{dt}\int_{G(t)} u dx\right]_{t=t_0} = \int_{G(t_0)} \dot{u}_Q dx.$$

Bilanz in Differentialform. Die Integrale in (6) lassen sich unter Ausnutzung der stetig differenzierbaren Abbildung $\underline{X}$ in Integrale über das feste Volumen $G(t_0)$ transformieren: Wegen $G(t) = \underline{X}(G(t_0))$ gilt für jedes auf $G(t)$ integrierbare $f(\cdot, t)$ die Beziehung

$$\int_{G(t)} f(\underline{x}, t) d\underline{x} = \int_{G(t_0)} f(\underline{X}(\underline{\xi}, t), t) \det\left(\frac{\partial \underline{X}}{\partial \underline{\xi}}\right) d\underline{\xi}.$$

Mit der Abkürzung $J_X := \det \underline{\underline{J}}_X$ für die Determinante der von $\underline{\xi}$ und t abhängigen Jacobi-Matrix

$$\underline{\underline{J}}_X := \frac{\partial \underline{X}}{\partial \underline{\xi}} = \left(\frac{\partial X_i}{\partial \xi_j}\right)_{i,j=1\ldots\mathsf{N}}$$

wird die Bilanz (6) demnach zu

$$\frac{d}{dt}\int_{G(t_0)} u(\underline{X}(\underline{\xi}, t), t) J_X(\underline{\xi}, t) d\underline{\xi} = \int_{G(t_0)} \dot{u}_Q(\underline{X}(\underline{\xi}, t), t) J_X(\underline{\xi}, t) d\underline{\xi}.$$

Da nunmehr über ein von t unabhängiges Gebiet integriert wird, können Ableitung und Integral vertauscht werden, und die obige Gleichung ist äquivalent zu

$$\int_{G(t_0)} \frac{d}{dt}\Big(J_X(\underline{\xi}, t) u(\underline{X}(\underline{\xi}, t), t)\Big) d\underline{\xi} = \int_{G(t_0)} J_X(\underline{\xi}, t) \dot{u}_Q(\underline{X}(\underline{\xi}, t), t) d\underline{\xi}.$$

[2] Ungeachtet der Möglichkeit, diese Bilanzen aus allgemeineren Prinzipien herzuleiten, handelt es sich um Axiome, da die Aussage, daß diese Bilanzen *für eine bestimmte physikalische Größe* (z.B. für die Enthalpie) gelten, im Rahmen der mathematischen Theorie nicht bewiesen wird.

Soll diese Gleichung für jedes $G(t_0) \subseteq \Omega$ gelten, so muß die differentielle Bilanz

$$\frac{d}{dt}\Big(J_X(\underline{\xi},t)u(\underline{X}(\underline{\xi},t),t)\Big) = J_X(\underline{\xi},t)\dot{u}_Q(\underline{X}(\underline{\xi},t),t)$$

punktweise fast überall auf Ω erfüllt sein.

Mit der Notation $u^{\underline{X}} := u \circ \underline{X}$ nach (4) läßt sich die differentielle Bilanz schreiben als

$$\frac{d}{dt}(J_X u^{\underline{X}}) = J_X \dot{u}_Q^{\underline{X}}. \tag{8}$$

Zu beachten ist, daß die *substantielle* Ableitung

$$\frac{d}{dt}(J_X u^{\underline{X}}) := \frac{d}{dt}(J_X(\underline{\xi},t)u(\underline{X}(\underline{\xi},t),t))$$

die Ableitung der in Lagrange-Koordinaten dargestellten Größe $u^{\underline{X}}$ bezeichnet.

Die durch (8) gegebene Möglichkeit der Bilanzierung wird im Abschnitt 2.3.1 für die Bilanz der Luft-Enthalpie angewendet. Die Kennzeichnung der Größen in Lagrange-Koordinaten als solche wird dort im Interesse der Übersichtlichkeit weggelassen.

Physikalische Deutung. Die Jacobi-Determinante $J_X(t)$ ist der Faktor, der das Verhältnis des transportierten Volumens $|G(t)|$ zum Ausgangsvolumen $|G(t_0)|$ angibt. Durch Multiplikation mit diesem *Volumenänderungsfaktor* werden die Dichten nicht auf das Volumen zum Zeitpunkt t, sondern auf das korrespondierende Volumen zum Zeitpunkt t_0 bezogen.

2.2.4 Bilanzierung am Rand

Der auf Ω definierte Strom $\underline{s}(u)$ soll ein auf $\overline{\Omega}$ stetig differenzierbares Vektorfeld sein, d.h. $\underline{s}(u)$ ist auf dem Rand $\partial\Omega$ definiert und dort gleich der stetigen Fortsetzung des Stromes im Innern.

Der in Normalenrichtung zum Rand $\partial\Omega$ ($\underline{n}$ bezeichnet den äußeren Normaleneinheitsvektor) fließende Anteil des Stromes $\underline{s}(u)$ wird durch am Rand wirkende Mechanismen bestimmt:

Undurchlässige Wände Der Strom über den Gebietsrand ist Null:

$$\underline{n} \cdot \underline{s}(u)|_{\partial\Omega} = 0$$

Zufluß Das über den Rand strömende Quantum von u wird von außen vorgegeben:

$$\underline{n} \cdot \underline{s}(u)|_{\partial\Omega} = \underline{n} \cdot \underline{w}u|_{\partial\Omega}$$

Wärmeübergang Der Temperaturverlauf an der Wand wird durch einen Sprung von ϑ_i (im Innern) auf ϑ_a (auf der Wand) angenähert. Der Wärmestrom wird durch den Wärmeübergangskoeffizienten α bestimmt:

$$\underline{n} \cdot \underline{s}(h)|_{\partial\Omega} = \alpha(\vartheta_i - \vartheta_a)|_{\partial\Omega}$$

Die Randbedingungen zu den partiellen Differentialgleichungen auf Ω ergeben sich aus jeweils einer der obigen Gleichungen.

2.3 Bilanzmodell

Die folgenden Größen werden auf dem Reaktorinneren (genauer: der Wirbelschichtzone) Ω als Funktionen des Ortes und der Zeit aufgefaßt und entsprechend bilanziert:

- Konzentration der benetzenden Flüssigkeit κ_F
- Konzentration des Wasserdampfes κ_D
- volumenbezogene Enthalpie der Luft h_L
- volumenbezogene Enthalpie des Flüssigkeitsfilms h_F
- volumenbezogene Enthalpie der Partikel h_P

Als Transportterme werden berücksichtigt

- die Dispersion durch Verwirbelung der Feststoffpartikel,
- die Advektion der Gasphase als Pfropfenströmung.

Wärmeleitung, Diffusion und Verwirbelung in der Gasphase werden ebenso wie die temperaturabhängige Ausdehnung der Gase vernachlässigt.

Der Wärmeübergang zwischen den einzelnen Phasen sowie der verdunstende Massenstrom und die von diesem mitgeführte Enthalpie werden als Reaktionsterme berücksichtigt.

Als Quellterme treten der Massenstrom der eingedüsten Flüssigkeit $\dot{\kappa}_Q$ und der von dieser mitgeführte Wärmestrom in Erscheinung.

Nichtverschwindende Randbedingungen ergeben sich durch die Eigenschaften der am Anströmboden in die Wirbelschicht eintretenden Trägerluft.

Die Bilanzen werden hier unmittelbar in ihrer Differentialform aufgestellt, auf eine Herleitung über die Bilanzierung am Volumenelement wird verzichtet. Es sei vorausgesetzt, daß alle vorkommenden Ableitungen existieren.

2.3.1 Bilanzen im Innern der Wirbelschicht

Die Bilanzierung im Inneren der Wirbelschichtzone Ω liefert eine partielle Differentialgleichung für jede Bilanzgröße. Anstelle der Enthalpie-Gleichungen werden Differentialgleichungen für die korrespondierenden Temperaturen aufgestellt.

Flüssigkeitskonzentration. Für die Bilanz der Flüssigkeitskonzentration κ_F sind zu berücksichtigen:

Quellen: Konzentrationsstrom der Tropfenabscheidung $\dot{\kappa}_Q$ (von außen vorgegeben)

Senken: Verdunstungsstrom $\dot{\kappa}_V$ (abhängig von κ_F und den Temperaturen, stark nichtlinear)

Transportterm: bestimmt durch die Dispersion der Feststoffpartikel, an deren Oberfläche sich die Flüssigkeit befindet: $\underline{s}(\kappa_F) = -\underline{\underline{D}}\nabla\kappa_F$

Bilanz: Die Bilanz der Flüssigkeitskonzentration ergibt sich zu

$$\frac{\partial \kappa_F}{\partial t} = \nabla \cdot (\underline{\underline{D}} \nabla \kappa_F) + \dot{\kappa}_Q - \dot{\kappa}_V. \tag{9}$$

Dampfkonzentration. Für die Dampfkonzentration κ_D sind zu berücksichtigen:

Quellen: Verdunstungsstrom $\dot{\kappa}_V$

Transport: Advektion $\underline{s}_a(\kappa_D) = \kappa_D w_L \underline{e}_z$

Bilanz:

$$\frac{\partial \kappa_D}{\partial t} = -w_L \frac{\partial \kappa_D}{\partial z} + \dot{\kappa}_V \tag{10}$$

Enthalpie der Partikel. Bilanz von $h_P = \kappa_P c_{pP} \vartheta_P$ mit $\kappa_P = \overline{\varepsilon} \rho_P$:

Transport: Wärmetransport durch Feststoffdispersion $\underline{s}(h_P) = -\underline{\underline{D}} \nabla h_P$

Quellen/Senken: Wärmeübergang $-\dot{q}_{PL}$, $-\dot{q}_{PF}$

Bilanz:

$$\frac{\partial h_P}{\partial t} = \nabla \cdot (\underline{\underline{D}} \nabla h_P) - \dot{q}_{PL} - \dot{q}_{PF} \tag{11}$$

Da die Koeffizienten κ_P und c_{pP} konstant sind, kann diese Gleichung ohne weiteres in der Form

$$\frac{\partial \vartheta_P}{\partial t} = \nabla \cdot (\underline{\underline{D}} \vartheta_P) - \frac{1}{\kappa_P c_{pP}} (\dot{q}_{PL} + \dot{q}_{PF}) \tag{12}$$

geschrieben werden.

Enthalpie des Flüssigkeitsfilms. Die volumenbezogene Enthalpie des Flüssigkeitsfilms h_F ist definiert als

$$h_F = \kappa_F c_{pF} \vartheta_F.$$

In die Bilanz von h_F gehen die folgenden Terme ein:

Transport: Wärmetransport durch Feststoffdispersion $\underline{s}(h_F) = -\underline{\underline{D}} \nabla h_F$

Quellen: Quellterm $\dot{q}_Q = \dot{\kappa}_Q c_{pF} \vartheta_Q$
Verdunstung $-\dot{q}_V = -\dot{\kappa}_V (\Delta h_{V0} + c_{pD} \vartheta_F)$
Wärmeübergang $-\dot{q}_{FL}$, $\dot{q}_{PF}$

Die Bilanzierung dieser Größen ergibt

$$\frac{\partial h_F}{\partial t} = \nabla \cdot (\underline{\underline{D}} \nabla h_F) + \dot{q}_Q - \dot{q}_V + \dot{q}_{PF} - \dot{q}_{FL}. \tag{13}$$

Mit $h_F = \kappa_F c_{pF} \vartheta_F$ ergibt sich unter Berücksichtigung von (9)

$$\begin{aligned} \frac{\partial \vartheta_F}{\partial t} &= \nabla \cdot (\underline{\underline{D}} \nabla \vartheta_F) + \frac{1}{\kappa_F} \Big((\nabla \kappa_F) \underline{\underline{D}} (\nabla \vartheta_F) + (\nabla \vartheta_F) \underline{\underline{D}} (\nabla \kappa_F) \Big) \\ &\quad + \frac{\dot{\kappa}_Q}{\kappa_F} (\vartheta_Q - \vartheta_F) - \frac{\dot{\kappa}_V}{\kappa_F} \left(\frac{\Delta h_{V0} + (c_{pD} - c_{pF}) \vartheta_F}{c_{pF}} \right) + \frac{1}{\kappa_F c_{pF}} (\dot{q}_{PF} - \dot{q}_{FL}). \end{aligned} \tag{14}$$

Der Term $(\nabla \kappa_F) \underline{\underline{D}} (\nabla \vartheta_F) + (\nabla \vartheta_F) \underline{\underline{D}} (\nabla \kappa_F)$ wird im folgenden vernachlässigt.

Enthalpie der Luft. Die volumenbezogene Enthalpie der feuchten Luft h_L ist definiert durch

$$h_L = (\kappa_L c_{pL} + \kappa_D c_{pD})\vartheta_L + \kappa_D \Delta h_{V0}. \tag{15}$$

In die Bilanz der Luft-Enthalpie gehen als *Quellterme* der Wärmestrom aus der Verdunstung $\dot{q}_V = \dot{\kappa}_V(\Delta h_{V0} + c_{pD}\vartheta_F)$ und die Wärmeübergangs-Terme $\dot{q}_{PL}$, $\dot{q}_{FL}$ ein.

Der *Transport* der Enthalpie erfolgt durch Advektion $\underline{s}_a(h_L) = \underline{w}_L h_L$ im durch den Geschwindigkeitsvektor $\underline{w}_L$ beschriebenen Strömungsfeld, das von der variablen Enthalpie abhängig ist: Die Enthalpieerhöhung durch *isobare* Wärmezufuhr bewirkt neben einer Temperaturerhöhung zugleich eine Ausdehnung (Erhöhung des spezifischen Volumens) der Luft. Das Strömungsfeld wird außerdem durch den verdunstenden Dampfstrom beeinflußt.

Auch ohne Kenntnis der konkreten Gestalt des Strömungsfeldes der Luft $\underline{X}(\underline{\xi}, t)$ kann die Bilanz für die volumenbezogene Enthalpie der feuchten Luft

$$\frac{d}{dt}\left(J_X h_L\right) = J_X(\dot{q}_V + \dot{q}_{PL} + \dot{q}_{FL})$$

in *Lagrange-Koordinaten* aufgestellt werden.

Der Term $J_X \kappa_L$ ist bezüglich der Lagrange-Koordinaten konstant, da für die (trockene) Luft keine Quellen bzw. Senken existieren. Aus der Definition der Enthalpie (15) ergibt sich somit

$$J_X(\kappa_L c_{pL} + \kappa_D c_{pD})\frac{d\vartheta_L}{dt} + (\Delta h_{V0} + c_{pD}\vartheta_L)\frac{d}{dt}(J_X \kappa_D) = J_X(\dot{q}_V + \dot{q}_{PL} + \dot{q}_{FL}).$$

Da der Wasserdampf vom Luftstrom mitgeführt wird, wird (bei vernachlässigter Dampf-Diffusion) der Transport des Dampfes ebenfalls durch das Strömungsfeld $\underline{X}$ der Luft beschrieben, und die zu (10) äquivalente Bilanz in Lagrange-Koordinaten lautet

$$\frac{d}{dt}(J_X \kappa_D) = J_X \dot{\kappa}_V.$$

Durch Einsetzen dieser Beziehung und unter Berücksichtigung von

$$\dot{q}_V = \dot{\kappa}_V(\Delta h_{V0} + c_{pD}\vartheta_F)$$

kann die Abhängigkeit von der Jacobi-Determinanten J_X des Strömungsfeldes eliminiert werden, und es ergibt sich für die Zeit-Ableitung der Temperatur in Lagrange-Koordinaten

$$\frac{d\vartheta_L}{dt} = \frac{\dot{q}_{PL} + \dot{q}_{FL} + \dot{\kappa}_V c_{pD}(\vartheta_F - \vartheta_L)}{\kappa_L c_{pL} + \kappa_D c_{pD}}. \tag{16}$$

Für die Transformation dieser Gleichung in Euler-Koordinaten über die Beziehung

$$\frac{d\vartheta_L}{dt} = \frac{\partial \vartheta_L}{\partial t} + \nabla \cdot (\underline{w}_L \vartheta_L)$$

wird nun vereinfachend angenommen, daß die räumlichen Schwankungen des Strömungsfeldes wesentlich kleiner sind als der mittlere Betrag der Strömungsgeschwindigkeit $\underline{w}_L$ und daher vernachlässigt werden können: Es wird mit einer konstanten Strömungsgeschwindigkeit in vertikaler Richtung $\underline{w}_L = w_L \underline{e}_z$ gerechnet, damit ist

$$\nabla \cdot (\underline{w}_L \vartheta_L) = w_L \frac{\partial \vartheta_L}{\partial z}.$$

Mit der Abkürzung $\kappa_Y c_{pY} := \kappa_L c_{pL} + \kappa_D c_{pD}$ ergibt sich aus (16) die Gleichung

$$\frac{\partial \vartheta_L}{\partial t} = -w_L \frac{\partial \vartheta_L}{\partial z} + \frac{1}{\kappa_Y c_{pY}}(\dot{q}_{PL} + \dot{q}_{FL}) + \frac{\dot{\kappa}_V c_{pD}}{\kappa_Y c_{pY}}(\vartheta_F - \vartheta_L). \tag{17}$$

2.3.2 Randbedingungen

Die Bilanzierung am Rand $\partial\Omega$ liefert die zum System partieller Differentialgleichungen gehörenden Randbedingungen.

Feste und flüssige Komponente. Bettpartikel und der auf ihnen befindliche Flüssigkeitsfilm können nicht über die Reaktorgrenzen hinweg transportiert werden. Der Wärmeaustausch mit der Wand wird vernachlässigt. Daher müssen die Normalkomponenten der den Größen κ_F, h_F und h_P zugeordneten Transportterme am Rand verschwinden:

$$\begin{aligned} \underline{n} \cdot \underline{s}(\kappa_F)|_{\partial\Omega} &= 0 \\ \underline{n} \cdot \underline{s}(h_F)|_{\partial\Omega} &= 0 \\ \underline{n} \cdot \underline{s}(h_P)|_{\partial\Omega} &= 0 \end{aligned}$$

Daraus ergeben sich die *natürlichen* Randbedingungen

$$\underline{n} \cdot \underline{\underline{D}} \nabla \kappa_F|_{\partial\Omega} = 0 \tag{18}$$
$$\underline{n} \cdot \underline{\underline{D}} \nabla h_F|_{\partial\Omega} = 0 \tag{19}$$
$$\underline{n} \cdot \underline{\underline{D}} \nabla h_P|_{\partial\Omega} = 0 \tag{20}$$

bzw. für die korrespondierenden Temperaturen

$$\underline{n} \cdot \underline{\underline{D}} \nabla \vartheta_F|_{\partial\Omega} = 0 \tag{21}$$
$$\underline{n} \cdot \underline{\underline{D}} \nabla \vartheta_P|_{\partial\Omega} = 0. \tag{22}$$

Gasphase. Der Stoff- und Wärmetransport in der Gasphase wird durch Advektion in einem vorgegebenen Strömungsfeld beschrieben. Für die Größen κ_D und ϑ_L, denen Advektionsgleichungen zugeordnet sind, werden Randbedingungen nur auf dem Einströmrand

$$\partial\Omega_{ein} := \{\underline{x} \in \partial\Omega, \underline{w}_L(\underline{x}) \cdot \underline{n}(\underline{x}) < 0\}$$

(d.h. am Anströmboden) benötigt, dort ist der Zustand der Gasphase durch den Zustand der von außen zugeführten Trägerluft gegeben:

$$\kappa_D|_{\partial\Omega_{ein}} = \kappa_{D,ein} \tag{23}$$
$$\vartheta_L|_{\partial\Omega_{ein}} = \vartheta_{L,ein} \tag{24}$$

Der Wärmeaustausch mit der Reaktorwand wird vernachlässigt.

2.3.3 Quellterme

Der Massenstrom $\dot{\kappa}_Q$ und der von diesem mitgeführte Wärmestrom $\dot{q}_Q = \dot{\kappa}_Q c_{pF} \vartheta_Q$ treten als Quellterme in Erscheinung.

Der Massenstrom $\dot{\kappa}_Q(\underline{x})$ wird als integrierbare Funktion des Ortes von außen eingeprägt (siehe 2.5), die Temperatur ϑ_Q der eingedüsten Flüssigkeit wird als konstant angenommen.

2.3.4 Reaktionsterme

Wärmeübergang. Die volumenbezogenen Wärmeströme $\dot{q}_{ij}$ sind für konstant angenommene α_{ij} jeweils einer Temperaturdifferenz proportional:

$$\begin{aligned} \dot{q}_{PL} &= A_{\mathit{eff}}\overline{\varphi}\alpha_{PL}(\vartheta_P - \vartheta_L) \\ \dot{q}_{PF} &= A_{\mathit{eff}}\varphi\alpha_{PF}(\vartheta_P - \vartheta_F) \\ \dot{q}_{FL} &= A_{\mathit{eff}}\varphi\alpha_{FL}(\vartheta_F - \vartheta_L) \end{aligned}$$

Verdunstung. Der verdunstende Massenstrom

$$\dot{\kappa}_V = \frac{\beta\rho_D}{d_F\rho_F}\kappa_F \ln\frac{p - p_D}{p - p_{D,\infty}}$$

entzieht der flüssigen Phase den Wärmestrom

$$\dot{q}_V = \dot{\kappa}_V(\Delta h_{V0} + c_{pD}\vartheta_F).$$

2.3.5 Koeffizienten

Die nachfolgenden Koeffizienten werden vereinfachend als konstant angenommen:

$\underline{w}_L$	Strömungsgeschwindigkeit	$\underline{w}_L = w_L\underline{e}_z,\ w_L > 0$
ε	Porosität	$\varepsilon \in (0,1)$
$\underline{\underline{D}}$	Dispersionsmatrix	$\underline{\underline{D}} = \mathrm{diag}(D_x, D_y, D_z),\ D_x, D_y, D_z > 0$
c_{pi}	Spezifische Wärmekapazitäten	$c_{pi} > 0$
α_{ij}	Wärmeübergangskoeffizienten	$\alpha_{ij} > 0$
β	Stoffübergangszahl	$\beta > 0$

2.3.6 Arbeitsbereich des Systems

Das physikalische Modell ist nur innerhalb gewisser Schranken für die Zustandsgrößen gültig.

Die *Temperaturen* im System müssen zwischen dem Erstarrungs- und dem Siedepunkt der eingebrachten Flüssigkeit beim Systemdruck liegen: Für reines Wasser muß gelten

$$0°\mathrm{C} \le \vartheta_i \le 100°\mathrm{C}\ (i = P, F, L).$$

Für die Gültigkeit des Benetzungsmodells muß die Bedingung $0 \le \varphi \le 1$ bzw.

$$0 \le \kappa_F \le \kappa_{F,max}$$

für die *Flüssigkeitskonzentration* erfüllt sein. Die *Dampfkonzentration* muß zwischen Null und dem beim Systemdruck maximal möglichen Wert liegen:

$$0 \le \kappa_D \le \varepsilon \cdot \rho_D(p, \vartheta_L).$$

2.4 Stationäre Modellgleichungen

2.4.1 Randwertproblem

Gebiet. Das System partieller Differentialgleichungen wird auf dem Reaktorinneren (der Wirbelschicht) Ω betrachtet. $\Omega \subset \mathbb{R}^3$ ist ein einfach zusammenhängendes konvexes Gebiet mit stückweise glattem Rand.

Stationäre Gleichungen. Die zeitlichen Ableitungen in den Gleichungen (9), (10), (12), (14) und (17) werden zu Null gesetzt, dadurch entsteht das *stationäre* Gleichungssystem

$$
\begin{aligned}
-\nabla \cdot (\underline{\underline{D}} \nabla \kappa_F) &= \dot{\kappa}_Q - \dot{\kappa}_V && (25)\\
w_L \frac{\partial \kappa_D}{\partial z} &= \dot{\kappa}_V && (26)\\
-\nabla \cdot (\underline{\underline{D}} \nabla \vartheta_P) &= \frac{1}{\kappa_P c_{pP}} (-\dot{q}_{PL} - \dot{q}_{PF}) && (27)\\
-\nabla \cdot (\underline{\underline{D}} \nabla \vartheta_F) &= \frac{\dot{\kappa}_Q}{\kappa_F}(\vartheta_Q - \vartheta_F) - \frac{\dot{\kappa}_V}{\kappa_F} \frac{\Delta h_{V0} + (c_{pD} - c_{pF})\vartheta_F}{c_{pF}} + \frac{1}{\kappa_F c_{pF}} (\dot{q}_{PF} - \dot{q}_{FL}) && (28)\\
w_L \frac{\partial \vartheta_L}{\partial z} &= \frac{\dot{\kappa}_V c_{pD}}{\kappa_Y c_{pY}} (\vartheta_F - \vartheta_L) + \frac{1}{\kappa_Y c_{pY}} (\dot{q}_{PL} + \dot{q}_{FL}) && (29)
\end{aligned}
$$

Das System (25)–(29) besteht aus drei elliptischen Gleichungen zweiter Ordnung und zwei Gleichungen erster Ordnung, die über ihre rechten Seiten gekoppelt sind.

Die Funktionen κ_i und ϑ_i sollen so gewählt sein, daß alle vorkommenden Ableitungen zumindest im L^2-Sinne existieren:

$$
\begin{gathered}
\kappa_F \in H^2(\Omega), \quad \vartheta_P \in H^2(\Omega), \quad \vartheta_F \in H^2(\Omega),\\
\kappa_D \in L^2(\Omega), \quad \tfrac{\partial \kappa_D}{\partial z} \in L^2(\Omega),\\
\vartheta_L \in L^2(\Omega), \quad \tfrac{\partial \vartheta_L}{\partial z} \in L^2(\Omega).
\end{gathered}
$$

Randbedingungen. Den einzelnen Gleichungen sind Randbedingungen unterschiedlichen Typs zugeordnet; die Randbedingungen (18), (23), (22), (21) und (24) werden aus der instationären Modellierung übernommen:

$$
\begin{aligned}
\underline{n} \cdot \underline{\underline{D}} \nabla \kappa_F|_{\partial\Omega} &= 0\\
\kappa_D|_{\partial\Omega_{ein}} &= \kappa_{D,ein}\\
\underline{n} \cdot \underline{\underline{D}} \nabla \vartheta_P|_{\partial\Omega} &= 0\\
\underline{n} \cdot \underline{\underline{D}} \nabla \vartheta_F|_{\partial\Omega} &= 0\\
\vartheta_L|_{\partial\Omega_{ein}} &= \vartheta_{L,ein}
\end{aligned}
$$

2.4.2 Koeffizientenfunktionen

Im folgenden sollen die Eigenschaften der Koeffizientenfunktionen (Reaktionsterme, Quellterme etc.) in den Gleichungen (25)–(29) im Gültigkeitsbereich des Modells (siehe 2.3.6) näher untersucht werden.

Die betrachteten Terme sind allesamt im Arbeitsbereich des Reaktors stetig differenzierbare Funktionen ihrer Argumente (Ausnahme: $\dot{T}_{FQ}$ nur für von 0 weg beschränktes κ_F).

In der numerischen Rechnung ist nicht auszuschließen, daß der Lösungsalgorithmus in einem Zwischenschritt physikalisch „unsinnige Werte“ (d.h. Werte außerhalb des Arbeitsbereiches) liefert. Daher werden

- φ auf das Intervall $[0, 1]$, und
- ϑ_i auf das Intervall $[1°\mathrm{C}, 99°\mathrm{C}]$

beschränkt, um einer *unerwünschten* Vorzeichenumkehr in den Koeffizienten vorzubeugen. (Der Verdunstungskoeffizient $C_V(\kappa_D, \vartheta_F)$ darf das Vorzeichen wechseln.)

Quellstrom. Der Flüssigkeits-Quellstrom $\dot{\kappa}_Q$ wird als Funktion des Ortes eingeprägt.

Die im Abschnitt 2.5 berechnete Funktion $\dot{\kappa}_Q$ (35) ist beschränkt und L^2-integrierbar, aber unstetig: sie springt am Rande des Strahlbereiches.

Wärmetransport durch Quellstrom. Der Term

$$\frac{\dot{\kappa}_Q}{\kappa_F}(\vartheta_Q - \vartheta_F) =: \dot{T}_{FQ}$$

in (28) wächst unbeschränkt für konstantes ϑ_F und $\kappa_F \to 0$. In einer stabilen Lösung mit $\kappa_F \to 0$ muß sich daher ϑ_F derart an ϑ_Q annähern, daß $\left|\frac{\vartheta_Q - \vartheta_F}{\kappa_F}\right| < M$ für eine feste Schranke $M < \infty$ erfüllt ist.

Verdunstungskoeffizient. Der Verdunstungsstrom

$$\dot{\kappa}_V = \dot{\kappa}_V(\kappa_F, \kappa_D, \vartheta_F) = \kappa_F \cdot C_V(\kappa_D, \vartheta_F)$$

ist linear in κ_F; der Verdunstungskoeffizient

$$C_V(\kappa_D, \vartheta_F) := \frac{\beta}{d_F \rho_F} \rho_D(\vartheta_F) \cdot \ln \frac{p - p_D(\kappa_D, \vartheta_F)}{p - p_{D,\infty}(\vartheta_F)}$$

ist für $p_{D,\infty}(\vartheta_F) < p$ (bzw. $\vartheta_F < 100°\mathrm{C}$) eine stetig differenzierbare Funktion von (κ_D, ϑ_F). C_V kann abhängig von (κ_D, ϑ_F) das Vorzeichen wechseln: $C_V > 0$ („Verdunstung“) im Falle $p_D < p_{D,\infty}$; $C_V < 0$ („Kondensation“) für $p_D > p_{D,\infty}$.

Wärmetransport durch Verdunstung. Der Term

$$\frac{\dot{\kappa}_V}{\kappa_F} \frac{\Delta h_{V0} + (c_{pD} - c_{pF})\vartheta_F}{c_{pF}} = C_V \frac{\Delta h_{V0} + (c_{pD} - c_{pF})\vartheta_F}{c_{pF}} =: \dot{T}_{FV}$$

in (28) ist für beschränkte (ϑ_F, C_V) stetig differenzierbar und beschränkt (κ_F im Nenner kürzt sich heraus).

Der Term

$$\frac{\dot{\kappa}_V c_{pD}}{\kappa_Y c_{pY}}(\vartheta_F - \vartheta_L) =: \dot{T}_{LV}$$

in (29) ist der Temperaturdifferenz $\vartheta_F - \vartheta_L$ proportional, der Koeffizient $\frac{\dot{\kappa}_V c_{pD}}{\kappa_Y c_{pY}}$ kann das Vorzeichen wechseln und hängt stetig differenzierbar von $(\kappa_F, \kappa_D, \vartheta_F, \vartheta_L)$ ab.

Wärmeübergang. Die Terme

$$\frac{A_{eff}}{\kappa_P c_{pP}}\varphi\alpha_{PF}(\vartheta_F-\vartheta_P)=:\dot{T}_{PF},\qquad \frac{A_{eff}}{\kappa_P c_{pP}}\overline{\varphi}\alpha_{PL}(\vartheta_L-\vartheta_P)=:\dot{T}_{PL}$$

in (27) sind den Temperaturdifferenzen proportional, der nichtnegative und beschränkte Koeffizient hängt über φ bzw. $\overline{\varphi}$ stetig differenzierbar von κ_F ab.

Die Terme

$$\begin{aligned}\frac{A_{eff}}{\kappa_F c_{pF}}\varphi\alpha_{PF}(\vartheta_P-\vartheta_F)&=\frac{\alpha_{PF}}{d_F\rho_F c_{pF}}(\vartheta_P-\vartheta_F), &=:\ \dot{T}_{FP},\\ \frac{A_{eff}}{\kappa_F c_{pF}}\varphi\alpha_{PL}(\vartheta_L-\vartheta_F)&=\frac{\alpha_{PL}}{d_F\rho_F c_{pF}}(\vartheta_L-\vartheta_F) &=:\ \dot{T}_{FL}\end{aligned}$$

in (28) sind den Temperaturdifferenzen proportional, die Koeffizienten sind positive Konstanten (κ_F im Nenner kürzt sich gegen φ!)

Die Terme

$$\frac{A_{eff}}{\kappa_Y c_{pY}}\overline{\varphi}\alpha_{PL}(\vartheta_P-\vartheta_L)=:\dot{T}_{LP},\qquad \frac{A_{eff}}{\kappa_Y c_{pY}}\varphi\alpha_{FL}(\vartheta_F-\vartheta_L)=:\dot{T}_{LF}$$

in (29) sind den Temperaturdifferenzen proportional, die nichtnegativen beschränkten Koeffizienten hängen stetig differenzierbar von $\varphi(\kappa_F)$ und $\kappa_Y c_{pY}(\vartheta_L)$ ab.

Randterme. Die am unteren Rand $\partial\Omega_{ein}$ eingeprägten Randfunktionen

$$\begin{aligned}\kappa_{D,ein}:&\ \partial\Omega_{ein}\to\mathbb{R}\\ \vartheta_{L,ein}:&\ \partial\Omega_{ein}\to\mathbb{R}\end{aligned}$$

werden als Konstanten angenommen.

2.5 Modellierung des Flüssigkeitseintrags

Durch eine Düse werden Flüssigkeitstropfen in die Wirbelschicht eingebracht, die an den Bettpartikeln haften bleiben und dort verdunsten. Durch die mathematische Modellierung des Geschwindigkeitsfeldes und der Massenverteilung dieser Tropfen wird die räumliche Verteilung des Flüssigkeits-Quellenfeldes $\dot{\kappa}_Q$ berechnet. Das resultierende Gleichungssystem ist unter den gewählten Voraussetzungen geschlossen lösbar.

2.5.1 Modellannahmen

Der Düsenstrahl wird vereinfachend als homogener kegelförmiger Tropfenstrom betrachtet.

Im folgenden wird ein auf das Düsenzentrum bezogenes Kugelkoordinatensystem mit den Koordinaten (r,ϕ,θ) verwendet; diese seien so gewählt, daß die Umrechnung in lokale kartesische Koordinaten (ξ,η,ζ) durch

$$\xi=r\cos\phi\sin\theta,\quad \eta=r\sin\phi\sin\theta,\quad \zeta=r\cos\theta$$

gegeben ist (d.h. der Winkel θ wird von der Düsenachse aus gezählt, siehe Abb. 2).

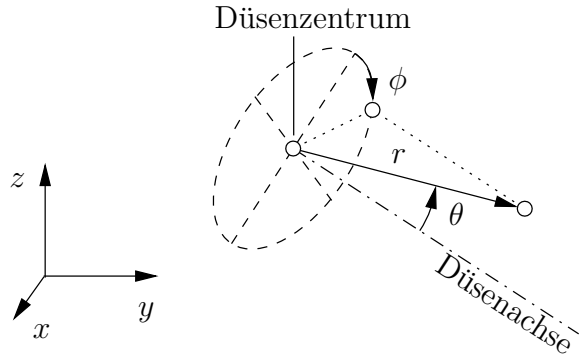


Abbildung 2: Lokales Koordinatensystem der Düse

Geradlinige Tropfenbahn. Es wird angenommen, daß sich die in die Wirbelschicht eingebrachten Flüssigkeitstropfen vom Düsenzentrum ausgehend mit konstanter Geschwindigkeit auf einer geradlinigen Bahn bewegen, bis sie auf einem Korn des Bettmaterials haften bleiben. (Die Ablenkung der Tropfen durch den Gasstrom wird vernachlässigt.)

Das Geschwindigkeitsfeld der Tropfen $\underline{u}_{Tr}$ ist somit radialsymmetrisch bezüglich des Düsenzentrums:

$$\underline{u}_{Tr} = u_{Tr}\underline{e}_r$$

($\underline{e}_r$ Einheitsvektor in radialer Richtung).

Radialsymmetrische Tropfenverteilung. Für die Berechnung der räumlichen Verteilung der Tropfenkonzentration $\kappa_{Tr}(\underline{x})$ wird von folgenden vereinfachenden Voraussetzungen ausgegangen:

- Die eingebrachte Flüssigeit wird innerhalb eines Kegels mit einem Öffnungswinkel $2\theta_D$ um die Düsenachse gleichmäßig verteilt, außerhalb dieses Kegels ist die Tropfenkonzentration Null.
- Die auf die Tropfenabscheidung wirkenden Einflußgrößen sind ortsunabhängig.

Aus Symmetriegründen hängt so die Tropfenkonzentration κ_{Tr} innerhalb des Strahlkegels nur von der Entfernung von der Düse ab:

$$\kappa_{Tr}(\underline{x}) = \begin{cases} \kappa_{Tr}(r) & (\theta \leq \theta_D) \\ 0 & (\theta > \theta_D) \end{cases} .$$

2.5.2 Tropfenbilanz

Tropfenabscheidung. In der Bilanz der Tropfenkonzentration im Düsenstrahl tritt die pro Volumenelement an den Bettpartikeln abgeschiedene Flüssigkeitsmenge $\dot{\kappa}_Q$ als Senke (Quellterm mit negativem Vorzeichen) in Erscheinung:

$$\frac{\partial \kappa_{Tr}}{\partial t} = -\nabla \cdot \underline{s}(\kappa_{Tr}) - \dot{\kappa}_Q.$$

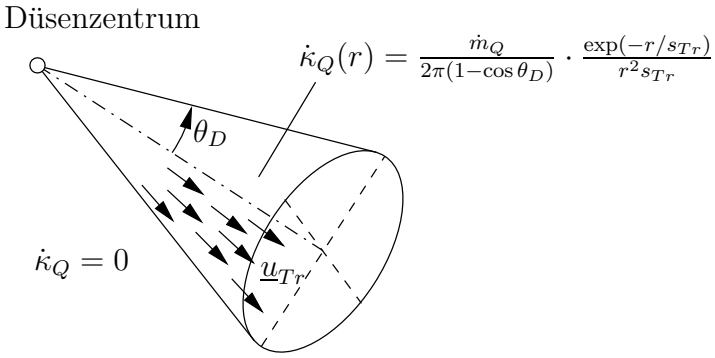


Abbildung 3: Düsenstrahl

Im stationären Fall ($\frac{\partial \kappa_{Tr}}{\partial t} = 0$) ergibt sich $\dot{\kappa}_Q$ als negative Divergenz des Tropfenstromes:

$$\dot{\kappa}_Q = -\nabla \cdot \underline{s}(\kappa_{Tr}). \tag{30}$$

Der im Düsenstrahl transportierte Flüssigkeitsstrom $\underline{s}(\kappa_{Tr})$ ist gleich dem Produkt aus der örtlichen Tropfenkonzentration und der Bahngeschwindigkeit der Tropfen:

$$\underline{s}(\kappa_{Tr}) = \kappa_{Tr}\underline{u}_{Tr}.$$

Aus der Radialsymmetrie des Geschwindigkeitsfeldes $\underline{u}_{Tr}(\underline{x}) = u_{Tr}\underline{e}_r(\underline{x})$ folgt

$$\nabla \cdot \underline{s}(\kappa_{Tr}) = \nabla \cdot (\kappa_{Tr} u_{Tr} \underline{e}_r) = u_{Tr}(\underline{e}_r \cdot \nabla \kappa_{Tr} + \kappa_{Tr} \nabla \cdot \underline{e}_r),$$

und mit $\nabla \cdot \underline{e}_r = 2/r$ ergibt sich

$$-\dot{\kappa}_Q = \nabla \cdot \underline{s}(\kappa_{Tr}) = u_{Tr}\left(\frac{\partial \kappa_{Tr}}{\partial r} + \frac{2\kappa_{Tr}}{r}\right). \tag{31}$$

Tropfenweglänge. Setzt man voraus, daß

- die Flüssigkeitstropfen und die Bettpartikel gleichförmige Kugeln sind und
- die Wahrscheinlichkeit der Absorption für den Fall, daß die Bahn eines Tropfens ein Bettpartikel berührt, für alle Flüssigkeitstropfen gleich einer ortsunabhängigen Konstanten η_A ist,

so erhält man aus geometrischen Betrachtungen eine mittlere Tropfenweglänge

$$S_{Tr} = \frac{2}{3}\frac{\varepsilon}{1-\varepsilon}\frac{d_P}{\eta_A} \tag{32}$$

in dem Sinne, daß gilt[3]

$$\dot{\kappa}_Q = \frac{u_{Tr}}{S_{Tr}}\kappa_{Tr}.$$

[3]Zur Berechnung der Abscheidung in einer Schüttschicht wird bei Löffler [17] die Beziehung

$$-\frac{d\kappa}{ds} = \frac{3}{2}\frac{1-\varepsilon}{\varepsilon}\frac{\eta_A}{d_P}\kappa$$

verwendet; die Lösung wird dort aber für ein paralleles Geschwindigkeitsfeld angegeben.

Zusammen mit (31) folgt für die Tropfenkonzentration κ_{Tr} die gewöhnliche Differentialgleichung

$$\frac{d\kappa_{Tr}}{dr} + \left(\frac{1}{S_{Tr}} + \frac{2}{r}\right)\kappa_{Tr} = 0,$$

mit der durch $C \in \mathbb{R}$ parametrisierten Schar von Lösungen

$$\kappa_{Tr}(r) = C\frac{\exp(-r/S_{Tr})}{r^2}. \tag{33}$$

Massenbilanz der Eindüsung. Die Konstante C wird nun aus der pro Zeiteinheit eingedüsten Flüssigkeitsmasse $\dot{m}_Q$ bestimmt:

Durch eine Sphäre $B(r)$ um das Düsenzentrum tritt der Flüssigkeitsstrom

$$\begin{aligned}\dot{m}_{Tr}(r) &= \int_{B(r)} \underline{s}_{Tr}(\underline{x}) \cdot d\underline{n} \\ &= \int_0^{\theta_D}\int_0^{2\pi} u_{Tr}\kappa_{Tr}(r) r^2 \sin\theta d\phi d\theta \\ &= 2\pi r^2 u_{Tr}\kappa_{Tr}(r)(1-\cos\theta_D)\end{aligned}$$

Im Grenzwert $r \to 0$ muß dieser Flüssigkeitsstrom gleich der eingedüsten Flüssigkeitsmenge $\dot{m}_Q$ sein:

$$\dot{m}_Q = \lim_{r\to 0}\dot{m}_{Tr}(r) = Cu_{Tr}2\pi(1-\cos\theta_D)\lim_{r\to 0}\exp\left(-\frac{r}{S_{Tr}}\right) = Cu_{Tr}2\pi(1-\cos\theta_D),$$

so daß folgt

$$C = \frac{\dot{m}_Q}{2\pi(1-\cos\theta_D)u_{Tr}}.$$

Der Tropfenstrom im Düsenstrahl $\underline{s}(\kappa_{Tr})$ ergibt sich mit (33) zu

$$\underline{s}(\kappa_{Tr}) = \kappa_{Tr}\underline{u}_{Tr} = \frac{\dot{m}_Q}{2\pi(1-\cos\theta_D)} \cdot \frac{\exp(-r/S_{Tr})}{r^2}\,\underline{e}_r,$$

der auf den Bettpartikeln abgeschiedene Flüssigkeitsstrom zu

$$\dot{\kappa}_Q(r) = -\nabla\cdot\underline{s}(\kappa_{Tr}) = \frac{\dot{m}_Q}{2\pi(1-\cos\theta_D)} \cdot \frac{\exp(-r/S_{Tr})}{r^2 S_{Tr}}. \tag{34}$$

2.5.3 Diskussion

Modifikation in Düsennähe. Die Funktion $\dot{\kappa}_Q$ nach (34) ist integrierbar: aus $\dot{\kappa}_Q \geq 0$ folgt $|\dot{\kappa}_Q| = \dot{\kappa}_Q$, und nach der Konstruktion ist

$$\|\dot{\kappa}_Q\|_{L^1(\mathbb{R}^3)} = \int_{\mathbb{R}^3}\dot{\kappa}_Q dx = \dot{m}_Q < \infty.$$

$\dot{\kappa}_Q$ ist jedoch in der Umgebung des Düsenzentrums nicht beschränkt, was der physikalischen Deutung widerspricht.

(Die Funktion ist nicht einmal L^2-integrierbar, was vom Standpunkt der Theorie der Differentialgleichungen wünschenswert wäre: $\dot{\kappa}_Q$ nach (34) liegt nicht in $L^p(\mathbb{R}^3), p \geq 3/2$.)

Daher wird die Funktion $\dot{\kappa}_Q$ auf einer Kugel $B_R(\underline{x}_D)$ um das Düsenzentrum modifiziert.

Ausgehend von der Annahme, daß der abgeschiedene Flüssigkeitsstrom einen gewissen Maximalwert $\dot{\kappa}_{Q,max}$ nicht überschreiten kann, ist es naheliegend, $\dot{\kappa}_Q$ auf diesen Wert zu begrenzen. Für den Wert von $\dot{\kappa}_{Q,max}$ sind zunächst keine Zahlenwerte bekannt.

Eine andere Modellvorstellung geht davon aus, daß in der Umgebung des Düsenzentrums überhaupt keine Tropfenabscheidung stattfindet, da in diesen Bereich keine Feststoffpartikel eindringen können. Demnach wird $\dot{\kappa}_Q$ auf einer Kugel $B_R(\underline{x}_D)$ zu Null gesetzt.

Beide Modellvorstellungen unterscheiden sich erheblich, aber nur auf einem mikroskopischen Bereich. Da die Kontinuumsmodellierung ohnehin nur für Bereiche anwendbar ist, die wesentlich größer als die einzelnen Partikel sind, ist dies nicht problematisch, solange das Integral der Differenz über den modifizierten Bereich hinreichend klein ist.

Der Radius R wird in der Größenordnung des Partikeldurchmessers d_P gewählt. Wegen

$$\int_{B_R(\underline{x}_D)} \dot{\kappa}_Q dx = \dot{m}_Q \left[1 - \exp\left(-\frac{R}{S_{Tr}}\right)\right] < \dot{m}_Q \frac{R}{S_{Tr}} \quad (\text{für } R > 0)$$

ist die Modifikation für $R \ll S_{Tr}$ unproblematisch.

Die zweite Variante „$\dot{\kappa}_Q = 0$ auf $B_R(\underline{x}_D)$“ ergibt sich unmittelbar aus der in 2.5.2 aufgestellten Bilanz, wenn η_A auf $B_R(\underline{x}_D)$ zu Null gesetzt wird. Überdies liefert sie eine erheblich einfachere Berechnungsvorschrift als die Variante der Begrenzung von $\dot{\kappa}_Q$ und wird deshalb bevorzugt. Der so berechnete Quellstrom

$$\dot{\kappa}_Q(r) = \frac{\dot{m}_Q}{2\pi(1-\cos\theta_D)} \cdot \begin{cases} \frac{\exp((R-r)/S_{Tr})}{r^2 S_{Tr}} & (r > R, \theta \leq \theta_D) \\ 0 & (\text{sonst}) \end{cases} . \tag{35}$$

unterscheidet sich im Abscheidungsbereich ($r > R$) nur um den Faktor $\exp(R/S_{Tr})$ von der Funktion nach (34). Für diese modifizierte Funktion gilt $\dot{\kappa}_Q \in L^2(\mathbb{R}^3)$ und $\dot{\kappa}_Q \in L^\infty(\mathbb{R}^3)$.

Abscheidung an der Apparatewand. Die obige Bilanz der Tropfenabscheidung erstreckt sich über ganz $\mathbb{R}^3$. Der die Wirbelschichtzone durchdringende Anteil des Tropfenstromes $\int_{\mathbb{R}^3 \setminus \Omega} \dot{\kappa}_Q dx$ schlägt sich aber nicht wie oben vorausgesetzt auf den Partikeln, sondern auf der Wandung nieder. Dieser Anteil ist für $\text{dist}(\underline{x}_D, \partial\Omega) \gg S_{Tr}$ vernachlässigbar klein.

2.6 Vereinfachtes Modellproblem

Zum besseren Verständnis der beim Lösen des Ausgangsproblems zu bewältigenden Schwierigkeiten soll ein auf zwei Gleichungen reduziertes Modellproblem betrachtet werden.

2.6.1 Physikalische Motivation

Wir vernachlässigen den Wärmeaustausch bzw. die Temperaturabhängigkeiten der Parameter und betrachten nur die zwei Bilanzgrößen κ_F (Flüssigkeitskonzentration) und κ_D (Dampfkonzentration).

Damit erhalten wir:

$$\frac{\partial \kappa_F}{\partial t} = \nabla \cdot (\underline{\underline{D}} \nabla \kappa_F) - \dot{\kappa}_V(\kappa_F, \kappa_D) + \dot{\kappa}_Q \tag{36}$$

$$\frac{\partial \kappa_D}{\partial t} = \nabla \cdot (\underline{w}_L \kappa_D) + \dot{\kappa}_V(\kappa_F, \kappa_D) \tag{37}$$

Bei vernachlässigter Lufteintrittsfeuchte erhalten wir als Randbedingungen

$$\underline{n} \cdot \underline{\underline{D}} \nabla \kappa_F|_{\partial\Omega} = 0, \tag{38}$$

$$\kappa_D|_{\partial\Omega_{ein}} = 0. \tag{39}$$

Der Verdunstungsstrom $\dot{\kappa}_V$ ist gegeben durch

$$\dot{\kappa}_V(\kappa_F, \kappa_D) = \frac{\beta}{d_F} \frac{\rho_D}{\rho_F} \kappa_F \ln \frac{\varepsilon \rho_D - \kappa_D}{\varepsilon \rho_D - \kappa_{D,\infty}}$$

wobei die (eigentlich stark temperaturabhängigen) Parameter $\frac{\beta}{d_F}\frac{\rho_D}{\rho_F} =: B$, $\varepsilon\rho_D =: C$ und $\kappa_{D,\infty}$ als Konstanten angenommen werden.

2.6.2 Stationäres Modellproblem

Indem wir weiterhin vereinfachend setzen

$$u := \kappa_F, \quad v := \kappa_D, \quad v^\infty := \kappa_{D,\infty}, \quad \underline{w} := \underline{w}_L \quad \text{und} \quad f := \dot{\kappa}_Q,$$

erhalten wir mit verschwindenden zeitlichen Ableitungen das Problem

$$-\nabla \cdot (\underline{\underline{D}} \nabla u) + uc(v) = f \qquad (\text{„}\textit{Diffusion}\text{“}) \tag{40}$$

$$\nabla \cdot (\underline{w} v) - uc(v) = 0 \qquad (\text{„}\textit{Advektion}\text{“}) \tag{41}$$

mit den Randbedingungen

$$\underline{n} \cdot \underline{\underline{D}} \nabla u|_{\partial\Omega} = 0 \qquad (\text{„}\textit{natürliche Randbedingung}\text{“}), \tag{42}$$

$$v|_{\partial\Omega_{ein}} = 0 \qquad (\text{„}\textit{Dirichlet-Randbedingung}\text{“}). \tag{43}$$

Beide Gleichungen sind über den Reaktionsterm $uc(v) := Bu \ln \frac{C-v}{C-v^\infty}$ gekoppelt.

Wir bemerken zum Reaktionsterm $z(u, v) := uc(v)$:

- $z(u, v)$ ist linear in u.
- $z(u, v)$ ist nichtlinear in v und nur für $v < C$ definiert.
- $z(u, v)$ ist (im Definitionsbereich $v < C$) stetig differenzierbar, somit läßt sich der Reaktionsterm lokal, d.h. für kleine Änderungen von (u, v) linearisieren.

Für den Koeffizienten $c(v) = \ln \frac{C-v}{C-v^\infty}$ gilt weiterhin:

- $c(v)$ ist monoton fallend mit $c(v) = 0$ bei $v = v^\infty > 0$, d.h. $c(v)$ kann abhängig von v das Vorzeichen wechseln.
- $c(v) \to -\infty$ für $v \to C$.
- Das Wachstum von $c(v)$ für fallende Argumente läßt sich durch eine lineare Schranke abschätzen.

Der Quellterm f kann aus physikalischer Sicht nicht negativ werden: $f \geq 0$ auf Ω.

2.6.3 Diskussion

Ohne Einschränkung der Allgemeinheit sei im folgenden $\underline{\underline{D}} = \underline{\underline{I}}$, was sich im Falle einer Diagonalmatrix $\underline{\underline{D}}$ ohne weiteres durch eine Transformation des Gebietes herbeiführen läßt. Damit wird das vereinfachte Problem zu

$$-\Delta u + uc(v) = f, \qquad \underline{n} \cdot \nabla u|_{\partial\Omega} = 0 \tag{44}$$

$$\nabla \cdot (\underline{w}v) - uc(v) = 0, \qquad v|_{\partial\Omega_{ein}} = 0 \tag{45}$$

Wir suchen $u \in H^1(\Omega)$, $v \in L^2(\Omega)$ mit $\nabla \cdot (\underline{w}v) \in L^2(\Omega)$.

Betrachtung der entkoppelten Gleichungen. Es werden zunächst die Eigenschaften der entkoppelt betrachteten Gleichungen untersucht.

Das *Diffusionsproblem*

$$-\Delta u + c(\underline{x})u = f; \qquad \underline{n} \cdot \nabla u|_{\partial\Omega} = 0 \tag{46}$$

ist eine lineare elliptische Gleichung zweiter Ordnung mit natürlichen Randbedingungen, der Reaktionskoeffizient c ist ortsabhängig.

Falls das zugehörige homogene Problem

$$-\Delta u + c(\underline{x})u = 0; \qquad \underline{n} \cdot \nabla u|_{\partial\Omega} = 0$$

nichttriviale Lösungen hat (z.B. für konstantes $c(\underline{x}) = -\lambda_i$ mit den vom Gebiet Ω abhängigen Eigenwerten $\lambda_i \geq 0$ von „$-\Delta$“), so ist dieses Problem nicht eindeutig lösbar.

Das nichtlineare *Advektionsproblem*

$$\nabla \cdot (\underline{w}v) - uc(v) = 0; \qquad v|_{\partial\Omega_{ein}} = 0 \tag{47}$$

hat für hinreichend reguläre u, $c(v)$, $\underline{w}$ (ggf. auf einem Teilgebiet von Ω) eine eindeutige Lösung, die sich mit Hilfe der Charakteristikentheorie als Lösung eines Anfangswertproblems konstruieren läßt.

Eigenschaften glatter Lösungen. Im folgenden sollen die Eigenschaften glatter Lösungen $(u, v) \in C^2(\Omega) \times C^1(\Omega)$ des Problems (44), (45) untersucht werden.

Dabei werden die im Abschnitt 2.6.2 diskutierten Eigenschaften von $c(v)$ sowie $f \geq 0$ ausgenutzt.

Die Raumkoordinaten $\underline{x} = (x_1, x_2, x_3)$ werden zur Hervorhebung von x_3 mit (x, y, z) bezeichnet: für die Ableitung in vertikaler Richtung kann $\partial_z \equiv \frac{\partial}{\partial z} \equiv \frac{\partial}{\partial x_3}$ geschrieben werden. Das Strömungsfeld werde durch einen konstanten Vektor in z-Richtung $\underline{w} = w\underline{e}_z$ mit $w > 0$ beschrieben. Das Gebiet Ω soll von prismatischer Gestalt (d.h. als Tensorprodukt eines ebenen Gebietes mit einem Intervall darstellbar) sein, wobei die Grundfläche des Prismas mit dem Einströmrand zusammenfällt: $\Omega = \partial\Omega_{ein} \times (0, H)$. (Die beiden letztgenannten geometrischen Vereinfachungen entsprechen der tatsächlich modellierten Situation.)

1. Die Randbedingung $\underline{n} \cdot \nabla u|_{\partial\Omega} = 0$ liefert die „Kompatibilitätsbedingung“

$$0 = \int_{\partial\Omega} \nabla u \cdot \underline{n} d\underline{x} = \int_{\Omega} \Delta u d\underline{x} = \int_{\Omega} (uc(v) - f) d\underline{x},$$

damit folgt für nichttriviale f

$$\int_\Omega uc(v)d\underline{x} = \int_\Omega f d\underline{x} := m > 0.$$

2. Die Advektionsgleichung führt wegen $\underline{w} = w\underline{e}_z$ für jedes feste $(x, y) \in \partial\Omega_{ein}$ auf das Anfangswertproblem zur gewöhnlichen Differentialgleichung

$$\partial_z v = \frac{1}{w}u(z)c(v), \quad v(0) = 0. \tag{48}$$

Für beschränktes $u(z)$ und $-\infty < v < C - \epsilon$ ist $uc(v)$ z-gleichmäßig Lipschitz-stetig bezüglich v. Nach dem Satz von Picard-Lindelöf hat das Anfangswertproblem (48) eine eindeutige Lösung $\tilde{v} \in C^1([0, h])$ für ein $h > 0$.

3. Sei für eine differenzierbare Lösung $\tilde{v}$ des Problems (48) $\tilde{v}(z_0) = v^\infty$ für irgendein $z_0 \in (0, H]$ und damit $c(\tilde{v}(z_0)) = 0$. Ohne Einschränkung können wir unter allen z_0 mit dieser Eigenschaft das kleinste auswählen, da die Menge $\{z \in [0, H] : \tilde{v}(z) = v^\infty\}$ für stetiges $\tilde{v}$ abgeschlossen ist. Wegen $\tilde{v}(0) = 0 \neq v^\infty$ ist $z_0 > 0$. Das Problem

$$\partial_z v = \frac{1}{w}u(z)c(v), \quad v(z_0) = v^\infty$$

ist zumindest lokal (vorwärts und rückwärts!) eindeutig lösbar, und $\bar{v} \equiv v^\infty$ ist eine Lösung dieses Problems. Damit kann aber $\tilde{v}$ nicht zugleich Lösung sein, d.h. eine Lösung $\tilde{v}$ von (48) kann nirgends den Wert $\tilde{v} = v^\infty$ annehmen. Aus $\tilde{v}(0) = 0 < v^\infty$ folgt wegen der Stetigkeit der Lösungen $\tilde{v} < v^\infty$ und damit $c(\tilde{v}) > 0$ für jede Lösung $\tilde{v}$ von (48).

4. Aus $\int_\Omega uc(v)d\underline{x} > 0$ folgt: Es gibt eine Menge $G \subset \Omega$ vom Maße größer Null, auf der $uc(v) > 0$ ist. Mit $c(v) > 0$ auf ganz Ω muß $u(\underline{x}) > 0$ auf einer Menge positiven Maßes sein.

5. Sei $u(\underline{x}) < 0$ irgendwo auf Ω. Das negative Minimum der stetigen Funktion u wird in einem Punkt $\underline{x}_0 \in \overline{\Omega}$ angenommen. Dann gilt $\Delta u(\underline{x}_0) \geq 0$ für $\underline{x}_0 \in \Omega$; wegen der Randbedingung $\underline{n} \cdot \nabla u|_{\partial\Omega} = 0$ gilt dies auch im Falle $\underline{x}_0 \in \partial\Omega$.

Wegen $c(v) > 0$ auf ganz Ω ist nun $u(\underline{x}_0)c(v(\underline{x}_0)) < 0$, mit $-\Delta u(\underline{x}_0) \leq 0$ und $f \geq 0$ kann die Gleichung $-\Delta u + c(\underline{x})u = f$ daher in $\underline{x}_0$ und wegen der Stetigkeit von $u, c(v)$ auch in einer Umgebung von $\underline{x}_0$ nicht erfüllt sein.

Also ist für eine glatte Lösung des Randwertproblems (44), (45) $u(\underline{x}) \geq 0$ auf Ω.

6. Das Anfangswertproblem (48) hat wegen $\frac{1}{w}uc(v) \geq 0$ bezüglich z monoton nichtfallende Lösungen mit $0 \leq v < v^\infty$, $c(0) \geq c(v) > 0$.

7. Mit $\partial_z v = \frac{1}{w}u(z)c(v), v(0) = 0, v(H) < v^\infty$ folgt (für $\underline{w} \perp \partial\Omega_{ein}$)

$$\int_\Omega uc(v) = \int_{\partial\Omega_{ein}} \int_0^H w\partial_z v(x, y, z)dzd(x, y) = w \int_{\partial\Omega_{ein}} v(x, y, H)d(x, y) < w|\partial\Omega_{ein}|v^\infty,$$

d.h. (glatte) Lösungen des Randwertproblems (44), (45) können nur für

$$m = \int_\Omega f d\underline{x} < w|\partial\Omega_{ein}|v^\infty$$

existieren.

3 Diskretisierung

3.1 Gitter

Für die Diskretisierung des betrachteten Problems wird das Gebiet $\Omega \subset \mathbb{R}^{\mathsf{N}}$ in eine endliche Anzahl disjunkter Teilgebiete zerlegt. Der einfachste Fall, die Zerlegung eines polygonalen Gebietes in Dreiecke für $\mathsf{N} = 2$, wird als *Triangulierung* angesprochen. In dieser Arbeit wird stattdessen der allgemeinere Begriff *Gitter* verwendet; dem Zusammenhang entsprechend ist entweder die Menge der Teilgebiete oder auch die von dieser induzierte Menge der Knoten, Kanten und Seiten gemeint. Das korrespondierende Objekt in `ug` heißt `GRID`.

Das Programmpaket `ug` unterstützt *unstrukturierte Gitter*. Hierunter werden Zerlegungen verstanden, in denen die *Elemente* (Teilgebiete) keiner regelmäßigen Anordnung genügen müssen und Elemente verschiedener geometrischer Typen nebeneinander vorliegen können.

Bei der *Gitterverfeinerung* werden einzelne Elemente des Gitters unter Beachtung bestimmter Regeln weiter zerlegt. Die Verfeinerung kann ggf. nur für ausgewählte Elemente durchgeführt werden *(lokale Verfeinerung)* und abhängig von einem *Fehlerschätzer* gesteuert werden *(adaptive Verfeinerung)*. Aus einem *Grobgitter* entsteht durch wiederholte Verfeinerung eine Hierarchie von Gittern. Die Diskretisierung des betrachteten Problems auf mehreren einander verfeinernden Gittern bildet die Grundlage des *Mehrgitteralgorithmus* zur Lösung der diskreten Gleichungssysteme.

Die Gitterverfeinerung wird durch die `ug`-Software selbständig vorgenommen, auf die zugrundeliegenden Algorithmen wird in dieser Arbeit nicht eingegangen. Zur Erstellung des Grobgitters muß der Software zunächst eine Beschreibung des zu zerlegenden Gebietes mitgeteilt werden, die Erstellung des Gitters kann manuell oder mit Hilfe eines *Gittergenerators* erfolgen.

3.1.1 Gebiet

Das Gebiet Ω ist zunächst nichts anderes als eine zusammenhängende, beschränkte, offene Teilmenge des $\mathbb{R}^{\mathsf{N}}$. Für die betrachtete Problemstellung soll Ω einfach zusammenhängend und konvex sein, der Rand von Ω soll aus endlich vielen glatten Stücken bestehen.

Für die numerische Rechnung wird eine „maschinenlesbare“ Beschreibung des Gebietes benötigt. Zweckmäßig ist die Verwendung einer Parameterdarstellung des Randes $\partial\Omega$, da der Rand ohnehin zur Auswertung der Randbedingungen benötigt wird.

In `ug` wird Ω durch mehrere parametrisierte Randsegmente beschrieben (die Beschreibung durch eine einzige in sich geschlossene parametrisierte Kurve bzw. Fläche ist nicht möglich).

> Die `ug`-Struktur `DOMAIN` enthält nur allgemeine Eigenschaften des Gebietes (Konvexität, Lage und Größe einer einhüllenden Kugel, Anzahl der Randsegmente). Die eigentliche geometrische Beschreibung von Ω steckt in einer Anzahl von Strukturen des Typs `BOUNDARY_SEGMENT`.

Die Wahl der Rand-Parametrisierung steht im Zusammenhang mit der topologischen Struktur des Grobgitters und beeinflußt die Qualität der verfeinerten Gitter, siehe 3.1.5.

3.1.2 FE-Gebietszerlegung

Das Gebiet $\Omega \subset \mathbb{R}^{\mathsf{N}}$ wird in eine endliche Anzahl n von *Elementen*[4] $\mathsf{L}_i \subset \mathbb{R}^{\mathsf{N}}$ zerlegt. Die Elemente L_i sind häufig (aber nicht notwendig) konvexe Polygone bzw. Polyeder. Im allgemeinen ist Ω selbst nicht polygonal und wird dann durch ein polygonales Gebiet $\mathsf{P}(\Omega)$ ersetzt. Die Gesamtheit aller Elemente L_i (bzw. ihrer Knoten C, Kanten E und Seiten S) wird als *Gitter* $\mathsf{G} = \{\mathsf{L}_i\}_{i=1\ldots\mathsf{n}}$ angesprochen.

Durch die Zerlegung sollen die folgenden Bedingungen erfüllt sein:

- Jedes Element L stammt aus einer Menge zulässiger Elemente, d.h. es läßt sich durch eine *reguläre* (stetige und invers stetige) Transformation T auf ein Referenzelement $\mathsf{R} \subset \mathbb{R}^{\mathsf{N}}$ aus der Menge der zugelassenen Referenzelemente $\mathcal{R}$ (siehe 3.2.2) abbilden:

  $$\forall\, \mathsf{L} \in \mathsf{G}\ \exists\, \mathsf{R} \in \mathcal{R},\ \exists\, \mathsf{T} \in \mathcal{T}(\mathsf{R})\ :\ \mathsf{L} = \mathsf{T}\mathsf{R}.$$

  Die Menge $\mathcal{T}$ der zulässigen Referenztransformationen ist für die einzelnen Referenzelemente unterschiedlich, sie kann affin lineare oder auch bi- bzw. trilineare Abbildungen umfassen (zur ug-Implementierung siehe 3.2.3).

- Die Elemente (hier *offen* angenommen) sind paarweise disjunkt:

  $$\mathsf{L}_i \cap \mathsf{L}_j = \emptyset \quad (i \neq j)$$

- Der Abschluß aller Elemente füllt $\overline{\mathsf{P}(\Omega)}$ aus:

  $$\bigcup_{i=1}^{\mathsf{n}} \overline{\mathsf{L}_i} = \overline{\mathsf{P}(\Omega)}$$

- Jeder Punkt aus $\overline{\mathsf{P}(\Omega)}$ ist Eckpunkt entweder aller oder keines der angrenzenden Elemente: Für $x \in \overline{\mathsf{L}_i} \cap \overline{\mathsf{L}_j}$ gilt

  $$x \in \mathsf{C}(\mathsf{L}_i) \ \Leftrightarrow\ x \in \mathsf{C}(\mathsf{L}_j).$$

Die FE-Gebietszerlegung in polygonale (polyedrische) Elemente wird unmittelbar durch den geometrischen Teil der ug-Datenstruktur repräsentiert.

> Den geometrischen Objekten Elemente L, Knoten C und Kanten E entsprechen die ug-Strukturen `ELEMENT`, `VERTEX` und `EDGE`; die Seiten S (für $\mathsf{N} = 3$) werden durch die `ELEMENT`-Strukturen impliziert, für sie existiert keine eigenständige Darstellung. Durch eine Struktur vom Typ `VERTEX` werden die gemeinsamen Eigenschaften aller Knoten dargestellt, die auf verschiedenen Gitter-Leveln an derselben Position liegen. Die Struktur `NODE` speichert die komplementäre level-abhängige Information.

Die Konstruktion und Verfeinerung der Gitter für nicht-polygonale Gebiete Ω wird von ug in einer solchen Weise vorgenommen, daß auf dem Rand der polygonalen Aproximierenden $\partial\mathsf{P}(\Omega)$ liegende Knoten C stets auch auf dem Rand $\partial\Omega$ liegen.

[4] Diese geometrischen Elemente $\mathsf{L} \subset \mathbb{R}^{\mathsf{N}}$ der Gebietszerlegung sind nicht zu verwechseln mit den Elementen einer Finite-Elemente-Diskretisierung, die jeweils ein geometrisches Element mit einem Satz von Basisfunktionen und Funktionalen (Freiheitsgraden) vereinen.

3.1.3 FV-Gebietszerlegung

Definition. Die Finite-Volumen-Diskretisierung basiert auf einer zur FE-Gebietszerlegung (FE-Gitter G) dualen Zerlegung (FV-Gitter H) von $\mathsf{P}(\Omega)$:

- Jedem Knoten (Eckpunkt) $\mathsf{C}_i \in \mathsf{G}$ entspricht genau ein diesen Knoten enthaltendes Kontrollvolumen *(control volume)* $\mathsf{V}_i \in \mathsf{H}$. (Die V_i bilden wiederum eine vollständige disjunkte Zerlegung in polygonale Teilgebiete, sind aber nicht notwendig konvex.)
- Jedem Element $\mathsf{L}_k \in \mathsf{G}$ entspricht genau ein Knoten $\mathsf{M}_k \in \mathsf{H}$ im Element-Inneren.
- Jeder Kante $\mathsf{E}_{ij} \in \mathsf{G}$ entspricht genau eine Grenzfläche *(control volume face)* $\mathsf{F}_{ij} \in \mathsf{H}$. (Diese sind aus mehreren *sub control volume faces*, siehe unten, zusammengesetzt.)
- ($\mathsf{N} = 3$) Jeder Seite $\mathsf{S}_m \in \mathsf{G}$ entspricht genau eine Kante $\mathsf{K}_m \in \mathsf{H}$. (Diese Kanten bestehen jeweils aus zwei linearen Abschnitten beiderseits des Schnittpunktes mit der Seite S_m; sie werden für die Diskretisierung nicht benötigt.)

Für das duale Gitter H werden die folgenden geometrischen Festlegungen getroffen:

- Die dualen Knoten $\mathsf{M}_k \in \mathsf{H}$ werden als Element-Mittelpunkte (Schwerpunkte) der $\mathsf{L}_k \in \mathsf{G}$ gewählt.
- Die Flächen $\mathsf{F}_{ij} \in \mathsf{H}$ schneiden die Kanten $\mathsf{E}_{ij} \in \mathsf{G}$ in den Mittelpunkten der E_{ij}.
- ($\mathsf{N} = 3$) Die Kanten $\mathsf{K}_m \in \mathsf{H}$ schneiden die Seiten $\mathsf{S}_m \in \mathsf{G}$ in deren Schwerpunkten.

Die Beschreibung der FV-Diskretisierung benötigt weiterhin

- die Teilvolumina *(sub control volumes)* $\mathsf{V}_i^k := \mathsf{V}_i \cap \mathsf{L}_k$,
- die Teilflächen *(sub control volume faces)* $\mathsf{F}_{ij}^k := \mathsf{F}_{ij} \cap \mathsf{L}_k$.

Ausrechnung. Für die Ausrechnung der zu einem Gitter gehörenden FV-Geometrie (d.h. die Berechnung der für die Diskretisierung benötigten Punkt-Koordinaten, Normalvektoren, Flächen und Volumina) wird die vorhandene Quelldatei `$(UGROOT)/np/algebra/fvgeom.c` benutzt.

> Implementierung in `fvgeom.c`: Anstelle der (Flächen-, Volumen-)Schwerpunkte der Elemente bzw. Seiten werden die arithmetischen Mittelwerte der Ecken eingesetzt. Beide Punkte stimmen genau dann überein, wenn die Referenztransformation des Elementes (bzw. deren Einschränkung auf die betreffende Seite) affin linear ist.

Alternative Definitionen. Es gibt andere als die hier dargestellte Verfahrensweise zur Konstruktion eines dualen Gitters: So werden in [16] das *Voronoi-Diagramm* und das *Dalton-Diagramm* für Dreiecksgitter eingeführt. Die in dieser Arbeit beschriebene Gebietszerlegung entspricht einer Verallgemeinerung des letzteren. Das Voronoi-Diagramm, in einfacher Weise definiert durch die Zuordnung jedes Punktes aus Ω zu dem ihm am nächsten liegenden Knoten, hat wünschenswerte theoretische Eigenschaften, ist aber im Falle dreidimensionaler unstrukturierter Gitter schwierig zu handhaben.

Grenzflächen. Die nach der obigen Vorschrift konstruierten Grenzflächen *(sub control volume faces)* F_{ij}^k werden für $\mathsf{N} = 2$ durch zwei, für $\mathsf{N} = 3$ durch vier Eckpunkte begrenzt: eine Kantenmitte, zwei Seitenmitten und den Element-Mittelpunkt. Sie sind ebene Vierecke im Falle von Tetraederelementen oder affinen Bildern von Prisma bzw. Würfel, liegen aber im allgemeinen Fall nicht in einer Ebene (siehe Anhang C.2.1).

Die Grenzflächen F_{ij}^k können für den nicht-ebenen Fall zum Beispiel als vier geraden Kanten einbeschriebene Minimalfläche oder über eine Darstellung durch die Formfunktionen des Referenzvierecks definiert werden. Diese beiden Definitionen stimmen im allgemeinen Fall nicht überein (siehe Anhang C.2.2).

Die Anwendung von Quadraturformeln auf diesen gekrümmten Flächen bringt einen zusätzlichen Diskretisierungsfehler mit sich.

Gebietsrand. Die zu einem Kontrollvolumen V_i gehörenden Teile des approximierten Gebietsrandes $\partial\mathsf{P}(\Omega)$ werden als *boundary faces*

$$\mathsf{B}_i := \partial\mathsf{V}_i \cap \partial\mathsf{P}(\Omega)$$

angesprochen. (Diese werden für die Auswertung von Neumann-Randbedingungen benötigt.)

3.1.4 Gitter-Verfeinerung

Für die Gitterverfeinerung werden die Standardwerkzeuge von ug benutzt, die folgenden Ausführungen beschränken sich auf die Einführung einiger zum Verständnis erforderlicher Begriffe.

Mehrgitter. Für das Mehrgitterverfahren wird eine Menge von einander verfeinernden Gittern $\mathsf{G}_0 \subset \mathsf{G}_1 \subset \ldots \subset \mathsf{G}_L$, ein *Mehrgitter*, verwendet. Die Schreibweise „$\mathsf{G}_{\ell-1} \subset \mathsf{G}_\ell$“ bezeichnet, daß die Elemente von G_ℓ durch *Verfeinerung* aus den Elementen von $\mathsf{G}_{\ell-1}$ hervorgehen.

Verfeinerung von Elementen. Elemente werden durch Einfügen neuer Knoten, Kanten und Seiten unterteilt oder *verfeinert*. Einem Eltern-Element in $\mathsf{G}_{\ell-1}$ entsprechen ein oder mehrere Kind-Elemente in G_ℓ.

Im Falle der konformen Verfeinerung eines Elementes stimmt der Abschluß des Eltern-Elementes mit dem Abschluß der disjunkten Vereinigung der zugehörigen Kind-Elemente überein.

Im nichtkonformen Fall (nicht-polygonales Ω) soll für die Rand-Approximation $\mathsf{P}_\ell(\Omega) \to \Omega$ mit $\ell \to \infty$ gelten. Neue Knoten werden daher nicht auf einer Seite, sondern auf dem korrespondierenden Rand-Segment eingefügt. Die Vereinigung der Kind-Elemente stimmt in diesem Falle nicht mit dem Eltern-Element überein.

Die unveränderte Kopie eines Elementes L von $\mathsf{G}_{\ell-1}$ nach G_ℓ wird für die formale Definition ebenfalls als Verfeinerung aufgefaßt.

Algorithmische Aspekte. Die Verfeinerung von Elementen $\mathsf{L}_i \subset \mathsf{G}_\ell$ beim Übergang $\mathsf{G}_\ell \to \mathsf{G}_{\ell+1}$ kann

- für alle Elemente in G_ℓ (uniforme Verfeinerung),
- für ausgewählte Elemente aus G_ℓ (lokale Verfeinerung)

vorgenommen werden. Im letztgenannten Falle müssen zur Einhaltung der Gitter-Konsistenz (siehe 3.1.2) gegebenenfalls auch Nachbarn der ausgewählten Elemente verfeinert werden [1]. Dies wird durch ug selbsttätig realisiert.

Die Auswahl der zu verfeinernden Elemente kann von Hand oder durch einen Indikator-Algorithmus unter Zuhilfenahme eines Fehlerschätzers erfolgen.

Qualität. Die Entscheidung, wo bei der Unterteilung der Elemente neue Knoten, Kanten und Seiten eingefügt werden, beeinflußt die geometrische Qualität der resultierenden Kind-Elemente. Sie wird algorithmisch durch eine Menge von *refinement rules* vorgenommen.

Hier wird ohne Betrachtung von Details auf die Standard-Methoden von ug zurückgegriffen. Wir beobachten lediglich zur Kontrolle die Maxima und Minima der Innenwinkel für die einzelnen Gitter-Level.

Definition 3.1 (Qualität) **:** *Die Qualität γ eines Elementes L wird als das Minimum über alle Innenwinkel von L und deren 2π-Komplemente definiert.*

Die Qualität γ eines Gitters G wird als Minimum der Qualitäten der Elemente von G definiert:

$$\gamma(\mathsf{G}) := \min_{\mathsf{L}\in\mathsf{G}} \gamma(\mathsf{L})$$

Für $\mathsf{N} = 2$ sind die Innenwinkel Winkel zwischen benachbarten Kanten $\mathsf{E}_{ij}, \mathsf{E}_{jk}$:

$$\gamma(\mathsf{L}) := \min_{\mathsf{E}_{ij},\mathsf{E}_{jk}\in\mathsf{L}} \arccos \frac{|(\,\mathsf{x}_i - \mathsf{x}_j\,,\,\mathsf{x}_j - \mathsf{x}_k\,)|}{|\mathsf{x}_i - \mathsf{x}_j|\,|\mathsf{x}_j - \mathsf{x}_k|}.$$

Für $\mathsf{N} = 3$ werden Winkel zwischen Seiten gemessen:

$$\gamma(\mathsf{L}) := \min_{\mathsf{S}_{ijk},\mathsf{S}_{ijm}\in\mathsf{L}} \arccos |(\,n_{\mathsf{S}_{ijk}}\,,\,n_{\mathsf{S}_{ijm}}\,)|$$

mit den Normalvektoren der Seiten $\mathsf{S}_{ijk}, \mathsf{S}_{ijm}$

$$n_{\mathsf{S}_{ijk}} := \frac{(\mathsf{x}_i - \mathsf{x}_j) \wedge (\mathsf{x}_j - \mathsf{x}_k)}{|\mathsf{x}_i - \mathsf{x}_j| \cdot |\mathsf{x}_j - \mathsf{x}_k|}.$$

> Das ug-Kommando `quality $a` rechnet Minimum und Maximum der Elementwinkel getrennt aus. Die Optionen `quality $s` bzw. `quality $i` ermöglichen die Beurteilung der Qualität einer Menge ausgewählter Elemente.

Definition 3.2 (Quasiuniforme Folge von Gittern) **:** *Eine Folge $\{\mathsf{G}_\ell\}_{\ell\in\mathbb{N}}$ von Gittern heißt quasiuniform, wenn $\{\gamma(\mathsf{G}_\ell)\}_{\ell\in\mathbb{N}}$ gleichmäßig nach unten beschränkt ist: es existiert ein $\gamma_0 > 0$, sodaß $\gamma(\mathsf{G}_\ell) \geq \gamma_0\ \forall \ell \in \mathbb{N}$.*

3.1.5 Parametrisierung des Randes

Die stückweise Parametrisierung des Randes von Ω ist mathematisch insofern trivial, als daß stets eine solche exisitiert. Bei der Konstruktion bzw. Auswahl der Parametrisierung sind jedoch einige *nicht offensichtliche* Abhängigkeiten zu beachten.

Auf diese Abhängigkeiten wird in der ug-Dokumentation [10, 11, 12, 13] nicht hingewiesen.

Zerlegung in Randsegmente. Der Gebietsrand $\partial\Omega$ wird in zur Gebietszerlegung analoger Weise disjunkt und vollständig in (abgeschlossene) parametrisierte Segmente zerlegt:

- Die Vereinigung der Randsegmente überdeckt $\partial\Omega$.
- Bis auf die Segment-Ränder ist jeder Punkt in $\partial\Omega$ in genau einem Segment enthalten.

Parameterraum. Die einzelnen Randsegmente werden jeweils als stetiges Bild eines $\mathsf{N}-1$-dimensionalen *Intervalls* parametrisiert. Ohne Einschränkung kann hierfür $[0,1]^{\mathsf{N}-1}$ verwendet werden, obwohl ug der Bequemlichkeit halber ein beliebiges Intervall $[\alpha,\beta]$ ($\alpha,\beta \in \mathbb{R}^{\mathsf{N}-1}$ mit $\alpha < \beta$) zuläßt.

Zusammenhang zur Gebietszerlegung. (ug-spezifisch!) Durch die Eckpunkte der Parameterintervalle sind Knoten (und für N=3 auch Kanten) auf dem Gebietsrand definiert. Diese definieren eine zur $\mathbb{R}^{\mathsf{N}-1}$-Gebietszerlegung analoge Topologie.

Die Topologie der Randsegmente ist mit der Topologie der später eingefügten Elemente in Übereinstimmung zu bringen:

- Segment-Knoten müssen den Knoten von Elementen entsprechen,
- Segment-Kanten müssen Kanten von Elementen entsprechen,
- weitere Knoten und Kanten können seitens der Gitter-Elemente nach Belieben eingefügt werden (Einfügen von Knoten in das Grobgitter durch das Shell-Kommando `bn`, die Kanten werden durch `ie` impliziert; weitere ergeben sich durch Verfeinerung).

Daher kann die Zerlegung desselben Gebietes durch verschiedene Elementtypen verschiedene Parametrisierungen des Randes erfordern.

Einfluß auf die Gitter-Verfeinerung. Beim Einfügen eines neuen Knotens auf einem Randsegment wird die Position dieses Knotens *im Parameterraum* des Segmentes (z.B. als Schwerpunkt bestehender Knoten) bestimmt. Die Parametrisierung des Randes muß also gewährleisten, daß die Regularitätseigenschaften des verfeinerten Gitters nicht gestört werden. Für $\mathsf{N} = 2$ kann die triviale Parametrisierung durch die Bogenlänge verwendet werden, im Falle $\mathsf{N} = 3$ muß *jedes* Randsegment *regulär* auf ein Quadrat abgebildet werden.

Weitere Details und Beispiele zur Rand-Parametrisierung sind im Anhang C.1 zusammengestellt.

3.2 Elemente und Basisfunktionen

Für die Diskretisierung stellt `ug`

- eine Reihe von Elementtypen (d.h. -geometrien),
- zugehörige Formfunktionen (knotenorientierte Basisfunktionen) sowie die Interpolation über diese und deren Ableitungen, und
- zugehörige Referenztransformationen

bereit.

Während die Element-Geometrien bereits durch die Namen der Elemente unmißverständlich erklärt sind, vermißt man in der Dokumentation einen Hinweis darauf, *welche* Formfunktionen und Referenztransformationen durch die Standard-Implementierung bereitgestellt werden.

3.2.1 Charakterisierung der Elemente

Elementtypen. Für die räumliche Diskretisierung stellt `ug-3.8` die folgenden Elementtypen zur Verfügung:

dim			nodes	edges	sides	`tag`
$\mathsf{N}=2$	Dreieck	triangle	3	3		3
	Viereck	quadrilateral	4	4		4
$\mathsf{N}=3$	Tetraeder	tetrahedron	4	6	4	4
	Pyramide	pyramid	5	8	5	5
	Prisma	prism	6	9	5	6
	Würfel	hexahedron	8	12	6	7

Geometrische Charakterisierung. Aufgrund der konvexen polygonalen Geometrie wird jedes Element L allein durch seine $\mathsf{m}(\mathsf{L})$ Eckpunkte $\mathsf{C}_i(\mathsf{L})$ ($i = 1, \ldots, \mathsf{m}$) eindeutig charakterisiert. Für $\mathsf{N} = 2$ ist das (abgeschlossene) Element gleich der konvexen Hülle seiner Eckpunkte, für $\mathsf{N} = 3$ gilt dies nur im Falle ebener Seitenflächen (siehe auch C.2.2).

Topologische Charakterisierung. Die Zahl der Ecken C, Kanten E und Seiten S eines Elementes L, die Zahl der Ecken und Kanten jeder Seite sowie die Zuordnung der Ecken zu gemeinsamen Kanten bzw. Seiten sind unter beliebigen regulären Referenztransformationen invariant. Diese Eigenschaften müssen daher nicht für jedes Element gesondert gespeichert werden, hierfür reicht der Bezug auf das jeweilige Referenzelement aus.[5]

Übersicht. Eine Übersicht über die Elementtypen, die auf ihnen definierten Formfunktionen und die Konventionen der Knotennumerierung ist in Abb. 4 dargestellt.

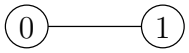
[5]Implementierung in `ug` (Typdefinition `ELEMENT` in `$(UGROOT)/gm/gm.h`): Die `ELEMENT`-Strukturen enthalten ein Datenfeld `tag`, das den Elementtyp (das Referenzelement) bezeichnet, sowie ein (geordnetes) Array von Zeigern auf die zugehörigen `NODE`s, hingegen keinen direkten Bezug zu `EDGE`s oder `SIDE`s.

In der folgenden Übersicht sind die in ug implementierten Referenzelemente (Koordinaten und ug-Numerierung der Ecken) und die zugehörigen Formfunktionen zusammengefaßt.

Bezeichnungen. Lokale Koordinaten $\underline{x} = (x_1, x_2, x_3)$; Komplement $\overline{x}_i := 1 - x_i$

Strecke
$\mathsf{N} = 1$, $\mathsf{m} = 2$
lineare Basis

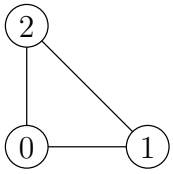
$\underline{\mathsf{p}}_0 = 0$
$\underline{\mathsf{p}}_1 = 1$



$\mathsf{v}_0(\underline{x}) = \overline{x}_1$
$\mathsf{v}_1(\underline{x}) = x_1$

Dreieck
$\mathsf{N} = 2$, $\mathsf{m} = 3$
lineare Basis

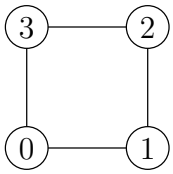
$\underline{\mathsf{p}}_0 = (0, 0)$
$\underline{\mathsf{p}}_1 = (1, 0)$
$\underline{\mathsf{p}}_2 = (0, 1)$



$\mathsf{v}_0(\underline{x}) = \overline{x_1 + x_2}$
$\mathsf{v}_1(\underline{x}) = x_1$
$\mathsf{v}_2(\underline{x}) = x_2$

Viereck
$\mathsf{N} = 2$, $\mathsf{m} = 4$
*bi*lineare Basis

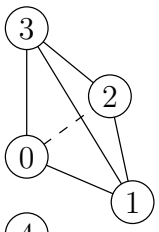
$\underline{\mathsf{p}}_0 = (0, 0)$
$\underline{\mathsf{p}}_1 = (1, 0)$
$\underline{\mathsf{p}}_2 = (1, 1)$
$\underline{\mathsf{p}}_3 = (0, 1)$



$\mathsf{v}_0(\underline{x}) = \overline{x}_1\overline{x}_2$
$\mathsf{v}_1(\underline{x}) = x_1\overline{x}_2$
$\mathsf{v}_2(\underline{x}) = x_1x_2$
$\mathsf{v}_3(\underline{x}) = \overline{x}_1x_2$

Tetraeder
$\mathsf{N} = 3$, $\mathsf{m} = 4$
lineare Basis

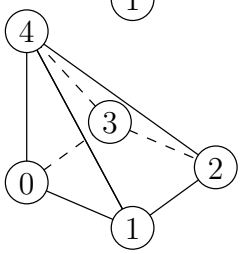
$\underline{\mathsf{p}}_0 = (0, 0, 0)$
$\underline{\mathsf{p}}_1 = (1, 0, 0)$
$\underline{\mathsf{p}}_2 = (0, 1, 0)$
$\underline{\mathsf{p}}_3 = (0, 0, 1)$



$\mathsf{v}_0(\underline{x}) = \overline{x_1 + x_2 + x_3}$
$\mathsf{v}_1(\underline{x}) = x_1$
$\mathsf{v}_2(\underline{x}) = x_2$
$\mathsf{v}_3(\underline{x}) = x_3$

Pyramide
$\mathsf{N} = 3$, $\mathsf{m} = 5$
stückweise *bi*lineare Basis mit
$x_m := \min\{x_1, x_2\}$

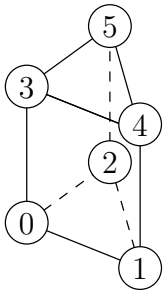
$\underline{\mathsf{p}}_0 = (0, 0, 0)$
$\underline{\mathsf{p}}_1 = (1, 0, 0)$
$\underline{\mathsf{p}}_2 = (1, 1, 0)$
$\underline{\mathsf{p}}_3 = (0, 1, 0)$
$\underline{\mathsf{p}}_4 = (0, 0, 1)$



$\mathsf{v}_0(\underline{x}) = \overline{x}_1\overline{x}_2 - \overline{x}_mx_3$
$\mathsf{v}_1(\underline{x}) = x_1\overline{x}_2 - x_mx_3$
$\mathsf{v}_2(\underline{x}) = x_1x_2 + x_mx_3$
$\mathsf{v}_3(\underline{x}) = \overline{x}_1x_2 - x_mx_3$
$\mathsf{v}_4(\underline{x}) = x_3$

Prisma
$\mathsf{N} = 3$, $\mathsf{m} = 6$
*bi*lineare Basis

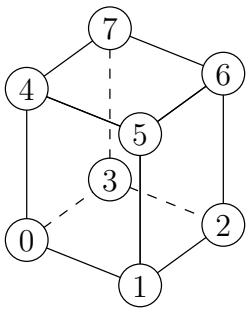
$\underline{\mathsf{p}}_0 = (0, 0, 0)$
$\underline{\mathsf{p}}_1 = (1, 0, 0)$
$\underline{\mathsf{p}}_2 = (0, 1, 0)$
$\underline{\mathsf{p}}_3 = (0, 0, 1)$
$\underline{\mathsf{p}}_4 = (1, 0, 1)$
$\underline{\mathsf{p}}_5 = (0, 1, 1)$



$\mathsf{v}_0(\underline{x}) = \overline{x_1 + x_2} \cdot \overline{x}_3$
$\mathsf{v}_1(\underline{x}) = x_1\overline{x}_3$
$\mathsf{v}_2(\underline{x}) = x_2\overline{x}_3$
$\mathsf{v}_3(\underline{x}) = \overline{x_1 + x_2} \cdot x_3$
$\mathsf{v}_4(\underline{x}) = x_1x_3$
$\mathsf{v}_5(\underline{x}) = x_2x_3$

Würfel
$\mathsf{N} = 3$, $\mathsf{m} = 8$
*tri*lineare Basis

$\underline{\mathsf{p}}_0 = (0, 0, 0)$
$\underline{\mathsf{p}}_1 = (1, 0, 0)$
$\underline{\mathsf{p}}_2 = (1, 1, 0)$
$\underline{\mathsf{p}}_3 = (0, 1, 0)$
$\underline{\mathsf{p}}_4 = (0, 0, 1)$
$\underline{\mathsf{p}}_5 = (1, 0, 1)$
$\underline{\mathsf{p}}_6 = (1, 1, 1)$
$\underline{\mathsf{p}}_7 = (0, 1, 1)$



$\mathsf{v}_0(\underline{x}) = \overline{x}_1\overline{x}_2\overline{x}_3$
$\mathsf{v}_1(\underline{x}) = x_1\overline{x}_2\overline{x}_3$
$\mathsf{v}_2(\underline{x}) = x_1x_2\overline{x}_3$
$\mathsf{v}_3(\underline{x}) = \overline{x}_1x_2\overline{x}_3$
$\mathsf{v}_4(\underline{x}) = \overline{x}_1\overline{x}_2x_3$
$\mathsf{v}_5(\underline{x}) = x_1\overline{x}_2x_3$
$\mathsf{v}_6(\underline{x}) = x_1x_2x_3$
$\mathsf{v}_7(\underline{x}) = \overline{x}_1x_2x_3$

Abbildung 4: Übersicht über Elemente, Knotennumerierung und Formfunktionen

Knotennumerierung. Für die korrekte Zuordnung der Ecken zu Kanten und Seiten ist sicherzustellen, daß die Ecken der Elemente stets in derselben Weise numeriert werden wie die des entsprechenden Referenzelementes. Bei der ug-internen Manipulation der Elemente (z.B. Verfeinerung) ist dies realisiert. In einer Reihe von Fällen müssen aber die Knoten eines Elementes bzw. einer Seite vom Anwender explizit bezeichnet werden:

- bei der Definition der Randsegmente (Erstellung einer `DOMAIN`), siehe 3.1.5,
- beim „manuellen“ Einfügen von Elementen in eine `DOMAIN` (zur Grobgitter-Definition),
- ggf. bei der Erstellung eigener *assemble routines*.

In den ersten beiden Fällen müssen die Knoten in der von ug implizit vorausgesetzten Reihenfolge angegeben werden, bei deren Nichteinhaltung treten Programmabstürze *(assertion failures)* auf, die keinen Rückschluß auf die Fehlerursache ermöglichen. In der Dokumentation [10, 11, 12, 13] wird weder auf die Notwendigkeit der Einhaltung der korrekten Knotenreihenfolge hingewiesen[6], noch findet sich an irgendeiner Stelle eine Beschreibung der zur Anwendung kommenden Konvention[7]. Somit ist die Erstellung eines Gitters mit prismatischen, pyramidalen oder quaderförmigen Elementen allein anhand der Dokumentation praktisch unmöglich.

Die undokumentierte, von ug vorausgesetzte Numerierung der Elemente ist in der Übersicht in Abb. 4 wiedergegeben.

Orientierung der Seiten. Der einem bestimmten Umlaufsinn einer Seite zugeordnete Normalenvektor ist entweder innere oder äußere Normale in bezug auf das Element.

Die Orientierung der Seiten ist nur gegenüber Referenztransformationen mit positiver Jacobi-Determinante invariant.

Die Orientierung der Seiten wird benötigt

- bei der Gitter-Erstellung: Elemente werden im Inneren eingefügt,
- für (orientierte) Normalvektoren in der Diskretisierung.

Randsegmente werden wie Seiten von Elementen behandelt (siehe auch 3.1.5 und C.1).

3.2.2 Referenzelemente und Formfunktionen

Die Referenzelemente R werden jeweils durch m Eckpunkte $\underline{\mathsf{p}}_i \in \{0,1\}^{\mathsf{N}}$ charakterisiert.

Auf den verschiedenen Referenzelementen R ist jeweils ein Satz von kanonischen Basisfunktionen $(\mathsf{v}_i)_{i=1}^{\mathsf{m}}$, den sogenannten *Formfunktionen (shape functions)*, mit den folgenden Eigenschaften definiert:

[6] Offensichtlich wurde die ug-Dokumentation bei der Einführung zusätzlicher Elementtypen nicht angepaßt. Die *manual page* `ELEMENT(3)` erwähnt nur die Elementtypen `triangle`, `quadrilateral` und `tetrahedron`, für die die Numerierung der Ecken unkritisch ist: für die simplizialen Elemente ist die Knotenreihenfolge aufgrund der Symmetrie unerheblich, das Viereck wird naheliegenderweise als Zyklus numeriert.

[7] Die Numerierung wird durch die Implementierung der Referenzelemente in `$(UGROOT)/gm/elements.c` festgelegt.

- Jedes v_i ist eine stetige Funktion $\mathsf{v}_i : \mathsf{R} \to \mathbb{R}$ mit $\mathsf{v}_i(\underline{\mathsf{p}}_j) = \delta_{ij}$. Jede Funktion u aus $\mathrm{span}\{(\mathsf{v}_i)_{i=1}^{\mathsf{m}}\}$ ist daher durch ihre Werte in den m Eckpunkten $\underline{\mathsf{p}}_i$ eindeutig bestimmt.
- Die Formfunktionen sind für simpliziale Elemente lineare, für die übrigen (ggf. stückweise) bi-/trilineare Polynome.
- Die Summe der Elemente (Funktionen) einer kanonischen Basis ist identisch eins:

$$\sum_{i=1}^{\mathsf{m}} \mathsf{v}_i(\underline{\lambda}) = 1 \quad \forall \underline{\lambda} \in \mathbb{R}^{\mathsf{N}}$$

- Die Restriktion jeder Formfunktion auf eine Seite (Kante) des Referenzelementes ist wieder eine Formfunktion auf dieser. (Daher ist in der Übersicht Abb. 4 die Strecke als Element mit aufgeführt.)

Die kanonische Basis wird in dreierlei Bedeutung gebraucht:

- als einfachster FE-Ansatzraum ($\mathbb{P}_1$- bzw. $\mathbb{Q}_1$-Elemente),
- als Ansatzraum für die FV-Diskretisierung,
- für die Darstellung der Referenztransformationen.

3.2.3 Referenztransformationen

Die Referenztransformationen T bilden jeweils ein Referenzelement R stetig auf ein „konkretes“ Element L des Gitters G ab.

In der Theorie (z.B. [5, 16]) werden üblicherweise die Ansatzfunktionen und die Referenztransformationen als zwei voneinander unabhängige Konzepte behandelt.

Durch ug sind gerade diejenigen Referenztransformationen T implementiert[8], die durch die kanonische Basis der Formfunktionen auf den Referenzelementen R dargestellt werden:

Für ein Element L mit den Eckpunkten $\underline{\mathsf{x}}_i = \underline{x}(\underline{\mathsf{p}}_i) \in \mathbb{R}^{\mathsf{N}}$, $i = 1, \ldots, \mathsf{m}$ ist die zugehörige Referenztransformation $\mathsf{T}(\mathsf{L})$ gegeben durch

$$\underline{x}(\underline{\lambda}) = \mathsf{T}(\mathsf{L})\underline{\lambda} = \sum_{i=1}^{\mathsf{m}} \mathsf{v}_i(\underline{\lambda}) \cdot \underline{\mathsf{x}}_i. \tag{49}$$

Finite Elemente mit dieser Eigenschaft „Ansatzraum = Raum der Referenztransformationen“ werden in der Literatur als *isoparametrische Elemente* bezeichnet, allerdings wird diese Bezeichnung nur für Elemente höheren Polynomgrades ($p \geq 2$) verwendet.

[8] Referenztransformationen (`LOCAL_TO_GLOBAL`[9] usw. in `$(UGROOT)/gm/shapes.h`) und Formfunktionen (`InterpolateFEFunction()` usw. in `$(UGROOT)/gm/shapes.c`) sind ohne Bezug aufeinander implementiert: Der Quelltext wird (mit anderen Bezeichnern) quasi wiederholt. Die Übereinstimmung beider wird weder in der Software noch in der Dokumentation reflektiert.

[9] Achtung: Das Makro `LOCAL_TO_GLOBAL` liefert die Referenztransformation, `TRANSFORMATION` berechnet nicht die Transformation, sondern deren Jacobi-Matrix! (Die irreführende Bezeichnung wurde offensichtlich bei der Erweiterung von affin-linearen zu beliebigen Referenztransformationen nicht angepaßt.)

Für den trivialen Fall simplizialer Elemente und affin-linearer Transformationen erübrigt sich eine solche Kennzeichnung; für die von ug unterstützten *nichttrivialen* Elemente (bei $\mathsf{N} = 3$) ist der Hinweis wertvoll, daß das oben genannte Konzept der Konstruktion der Referenztransformationen durchgängig für *alle* Elementtypen verwendet wird.

Eigenschaften der Referenztransformationen. Die Eigenschaften der durch die Darstellung (49) gegebenen Referenztransformationen hängen nur von der Geometrie der Elemente L ab. Insbesondere bewirkt die Einschränkung der Menge der zulässigen Elemente (z.B. auf Parallelogramme anstelle allgemeiner Vierecke) eine Einschränkung der Referenztransformationen auf eine bestimmte Klasse (z.B. affin lineare).

Die Geometrie der Elemente wird durch die Regeln zur Gitterverfeinerung und am Gebietsrand auch durch dessen Parametrisierung bestimmt, ist also nicht direkt beeinflußbar.

Bemerkung. Falls für je N Kanten eines Elementes L mit einer gemeinsamen Ecke C_i

$$(\underline{\mathsf{x}}_j - \underline{\mathsf{x}}_i) \wedge (\underline{\mathsf{x}}_k - \underline{\mathsf{x}}_i) \geq C \qquad \text{für } \mathsf{N} = 2, \text{ bzw.}$$
$$\left[\, \underline{\mathsf{x}}_j - \underline{\mathsf{x}}_i \,,\, \underline{\mathsf{x}}_k - \underline{\mathsf{x}}_i \,,\, \underline{\mathsf{x}}_m - \underline{\mathsf{x}}_i \,\right] \geq C \quad \text{für } \mathsf{N} = 3$$

bei geeigneter Kantenreihenfolge für ein festes $C > 0$ erfüllt ist (dies ist insbesondere dann der Fall, wenn die Kantenlängen nach unten und die Innenwinkel, für $\mathsf{N} = 3$ auch Winkel zwischen Seiten, von 0 und π weg beschränkt sind), so ist die durch (49) gegebene Referenztransformation T ein *Diffeomorphismus* von R auf L, dessen Jacobi-Determinante vom Betrage nicht kleiner als C ist.

Bemerkung. Die Jacobi-Determinante der Transformation ist nur dann positiv (und die Transformation läßt die Orientierung unverändert), wenn das Element mit der korrekten Orientierung der Seiten, d.h. insbesondere mit der korrekten Knotennumerierung, in das Gitter eingefügt wurde.

Bemerkung. Eine durch (49) gegebene Referenztransformation T ist affin-lineare Abbildung von $\mathbb{R}^{\mathsf{N}}$ nach $\mathbb{R}^{\mathsf{N}}$ genau dann, wenn die Bilder jedes Paares in R paralleler Kanten parallele Kanten in L sind.

Dies gilt *auch* für das Pyramidenelement mit nur stückweise bilinearen Formfunktionen: Der zu $x_m = \min(x_1, x_2)$ gehörende Anteil verschwindet für $\mathsf{v}_0 + \mathsf{v}_2 = \mathsf{v}_1 + \mathsf{v}_3$.

3.3 Finite-Volumen-Diskretisierung

In diesem Abschnitt soll die zur räumlichen Diskretisierung des zu lösenden Problems verwendete Finite-Volumen-Diskretisierung ausführlich beschrieben werden.

Das *stationäre* Problem ist damit bereits in ein endlichdimensionales Problem überführt. Das *instationäre* Problem wird mittels der vertikalen Linienmethode (MOL oder VMOL) diskretisiert: an die FV-Raumdiskretisierung schließt sich ein Zeitschrittverfahren an.

Die nachfolgende Darstellung bezieht sich auf die Diskretisierung auf *einem* Gitter G_ℓ (für festes $\ell \geq 0$), für das Mehrgitter-Verfahren wird diese Vorgehensweise auf jedem Gitter-Level $\ell = 0, \ldots, L$ wiederholt. Der Index ℓ wird im folgenden weggelassen.

Für die im Abschnitt 3.3.3 im Detail beschriebene Ausrechnung der Koeffizienten des diskreten nichtlinearen Systems wurde in Anlehnung an das ug-Paket „sc“ neuer C-Code erstellt.

3.3.1 Problemstellung

PDE. Das zu lösende Problem läßt sich als System von M (elliptischen bzw. parabolischen) partiellen Differentialgleichungen *in Divergenzform* über einem Gebiet $\Omega \subset \mathbb{R}^{\mathsf{N}}$ schreiben:

$$k u_t - \nabla \cdot (\underline{\underline{D}} \nabla u - \underline{w} u) + c(u) u = \nabla \cdot \underline{f} \tag{50}$$

Die Koeffizienten in (50) lassen sich folgendermaßen charakterisieren:

k	Zeitabhängigkeit	$k = 0$ stationäres Problem,
		$k = 1$ instationäres Problem
$\underline{\underline{D}}$	Diffusionsmatrix	konstante nichtnegative Diagonalmatrix $\underline{\underline{D}} \in (\mathbb{R}^{\mathsf{N}\times\mathsf{N}})^{\mathsf{M}}$
$\underline{w}$	Strömungsfeld	konstanter Vektor $\underline{w} \in (\mathbb{R}^{\mathsf{N}})^{\mathsf{M}}$
$\underline{f}(x)$	Quellenfeld	ortsabhängiger Vektor: $\underline{f} \in ([H^1(\Omega)]^{\mathsf{N}})^{\mathsf{M}}$
$c(u)$	Reaktionskoeffizient	stetige Funktion $\mathbb{R}^{\mathsf{M}} \supset I \to \mathbb{R}^{\mathsf{M}}$
		(hier Kopplung der Gleichungen bzw. Komponenten von u)

Für die Definition der Diskretisierung sei $u \in L^1(\Omega^t)^{\mathsf{M}}$ ($\Omega^t = \Omega$ oder $\Omega \times [0,T]$) derart, daß alle in (50) auftretenden Terme ebenfalls in $L^1(\Omega^t)^{\mathsf{M}}$ liegen. Die Divergenz-Terme sollen darüber hinaus auf gewissen $(\mathsf{N}-1)$-dimensionalen Flächen (den Kontrollvolumengrenzen) integrierbar sein. — Die Definition der Finite-Volumen-Diskretisierung setzt integrierbare Funktionen voraus; für die Theorie der Gleichungen wird $u \in L^2(\Omega^t)$ benötigt.

Randbedingungen. Für die einzelnen Komponenten von u sind Randbedingungen unterschiedlichen Typs wirksam:

Dirichletsche Randbedingung (RB 1. Art) am Einströmrand (Teil von $\partial\Omega$ mit $\underline{w} \cdot \underline{n} < 0$):

$$u|_{\partial\Omega\,:\,\underline{w}\cdot\underline{n}<0} = u_{in} \tag{51}$$

Neumannsche Randbedingung (RB 2. Art):

$$(\underline{\underline{D}} \nabla u) \cdot \underline{n}|_{\partial\Omega} = g \tag{52}$$

Natürliche Randbedingungen sind Neumannsche Randbedingungen mit $g = 0$. Bei der Implementierung werden Neumannsche und natürliche Randbedingungen unterschieden, da bei letzteren eine Anzahl von Rechenschritten (mit Argument 0) eingespart werden kann.

3.3.2 Formale Diskretisierung

Die hier beschriebene Finite-Volumen-Diskretisierung zeichnet sich durch Freiheitsgrade in den Knoten aus: Zur Diskretisierung auf dem Gitter G wird u in einem Raum stetiger Funktionen gesucht, dessen Elemente gerade durch ihre Werte in den $\mathsf{n}(\mathsf{G})$ Knoten C_i von G eindeutig bestimmt sind. Anstelle der „kontinuierlichen“ partiellen Differentialgleichung (50) wird nun das endlichdimensionale algebraische Gleichungssystem betrachtet, das durch die Integration von (50) über die den Knoten C_i zugeordneten Kontrollvolumina V_i entsteht.

Ansatzraum. Die Funktion u wird in einem Funktionenraum gesucht, dessen Elemente gerade durch die Knotenwerte

$$\mathsf{u}_i := u(\mathsf{C}_i) \in \mathbb{R}^{\mathsf{M}}$$

eindeutig bestimmt sind. Hierfür bietet sich die lineare Hülle der Formfunktionen an: Auf jedem Element $\mathsf{L}_k \in \mathsf{G}$ sei $u(x)$ gegeben durch

$$u(x)\Big|_{\mathsf{L}_k} := \sum_{\mathsf{C}_i \in \mathsf{L}_k} \mathsf{v}_i^k(\mathsf{T}_k^{-1}x)\mathsf{u}_i, \tag{53}$$

wobei $\mathsf{T}_k := \mathsf{T}(\mathsf{L}_k)$ die Referenztransformation zum Element L_k und v_i $(i = 1, \ldots, \mathsf{m}(\mathsf{L}_k))$ die auf dem zu L_k gehörenden Referenzelement definierten Formfunktionen bezeichnen.

Aufgrund der in 3.2.2 angeführten Eigenschaften der Formfunktionen sind die durch (53) gegebenen Ansatzfunktionen $u(x)$ für jedes $\mathsf{u} \in \mathbb{R}^{\mathsf{M}\times\mathsf{N}}$ wohldefinierte und global stetige Funktionen.

Abhängig von der Geometrie der zur Gebietszerlegung verwendeten Elemente enthält der Ansatzraum stückweise lineare, bi-/trilineare oder auch rationale Funktionen.

Integration über Kontrollvolumina. Die integrierbaren „kontinuierlichen" Terme $q \in L^1(\Omega)$ werden in diskrete, der Knotenmenge $\{\mathsf{C}_i\}_{i=1}^{\mathsf{n}(\mathsf{H})}$ des Gitters G zugeordnete Vektoren $\mathsf{I}_{\mathsf{H}}[q] \in \mathbb{R}^{\mathsf{n}(\mathsf{H})}$ überführt, indem die Integrale

$$\mathsf{I}_{\mathsf{H}}[q](\mathsf{V}_i) := \int_{\mathsf{V}_i} q dx$$

über die einzelnen Kontrollvolumina V_i $(i = 1, \ldots, \mathsf{n})$ betrachtet werden.

Alle Terme in Divergenzgestalt können wegen des Gaußschen Satzes als Integral über den Rand des Kontrollvolumens diskretisiert werden:

$$\mathsf{I}_{\mathsf{H}}[\nabla \cdot \underline{q}](\mathsf{V}_i) = \int_{\mathsf{V}_i} \nabla \cdot \underline{q} dx = \int_{\partial\mathsf{V}_i} \underline{q} \cdot \underline{n}_i dx$$

$\underline{n}_i$ bezeichnet die *äußere* Normale zum Kontrollvolumen V_i.

Für das Problem (50) erhalten wir die Diskretisierung

$$\alpha\frac{\partial}{\partial t}\int_{\mathsf{V}_i} u(x)dx - \int_{\partial\mathsf{V}_i} [\underline{\underline{D}}\nabla u(x) - \underline{w}u(x)] \cdot \underline{n}_i(x)dx + \int_{\mathsf{V}_i} c(u(x))u(x)dx = \int_{\partial\mathsf{V}_i} \underline{f}(x) \cdot \underline{n}_i(x)dx \tag{54}$$

für $i = 1, \ldots, \mathsf{n}(\mathsf{H})$.

Randbedingungen. Entsprechend der unterschiedlichen Typen von Randbedingungen wird die im folgenden beschriebene Diskretisierung jeweils nur für bestimmte Komponenten von u angewendet.

Dirichletsche Randbedingungen: Die Werte von u in den Knoten C_i, die auf einem Abschnitt von $\partial\Omega$ mit Dirichlet-Randbedingung liegen, werden gleich dem Randwert $u_{in}(\mathsf{C}_i)$ gesetzt. Die diesen Knoten zugeordneten Unbekannten können aus dem diskreten Problem eliminiert werden, indem die zugehörige Matrix-Zeile durch den i-tem Einheitsvektor ersetzt wird.

Neumannsche Randbedingungen: Die Diskretisierung des Terms $\nabla \cdot \underline{\underline{D}}\nabla u$ führt auf

$$\mathsf{I}_\mathsf{H}[\nabla \cdot \underline{\underline{D}}\nabla u](\mathsf{V}_i) = \int_{\mathsf{V}_i} \nabla \cdot \underline{\underline{D}}\nabla u dx = \int_{\partial \mathsf{V}_i} (\underline{\underline{D}}\nabla u) \cdot \underline{n}_i dx$$

Für ein Kontrollvolumen V_i am Rand von Ω ist auf $\mathsf{B}_i := \partial\mathsf{V}_i \cap \partial\Omega$ der Randwert $(\underline{\underline{D}}\nabla u)\cdot\underline{n} = g$ vorgegeben, d.h. wir erhalten

$$\mathsf{I}_\mathsf{H}[\nabla \cdot \underline{\underline{D}}\nabla u](\mathsf{V}_i) = \int_{\partial\mathsf{V}_i \setminus \partial\Omega} (\underline{\underline{D}}\nabla u) \cdot \underline{n}_i dx + \int_{\partial\mathsf{V}_i \cap \partial\Omega} g dx.$$

Der erste Summand trägt zur Steifigkeitsmatrix bei (siehe unten), der zweite Summand $\mathsf{g}_i := \int_{\mathsf{B}_i} g dx$ dagegen zur diskreten rechten Seite.

3.3.3 Numerische Integration

Da wir die einzelnen Integrale im allgemeinen nicht exakt ausrechnen können, werden Integrationsformeln (Quadraturformeln) verwendet. Der durch die Quadratur hervorgerufene Fehler kann toleriert werden, wenn er die durch den Ansatzraum vorgegebene Fehler-Ordnung des Verfahrens nicht stört.

Durch den stückweise (bi-/tri-)linearen Ansatzraum $\mathsf{U} := \mathrm{span}\{(\mathsf{v}_i)_{i=1}^{\mathsf{n}(\mathsf{H})}\}$ ist für die Approximation von $u \in C^1$ die Ordnung 1 vorgegeben. Daher sind für die numerische Integration Quadraturformeln erster Ordnung ausreichend. Die Einpunkt-Quadraturformel

$$\int_{\mathsf{V}_i} q(x)dx \approx |\mathsf{V}_i| q(\tilde{x}) =: \mathsf{I}[q](\mathsf{V}_i) \tag{55}$$

zum Integrationspunkt $\tilde{x}$ integriert auf V_i lineare q exakt und ist daher für glattes q von erster Ordnung, falls $\tilde{x}$ als Schwerpunkt $\mathsf{s}_i := \frac{1}{|\mathsf{V}_i|}\int_{\mathsf{V}_i} x dx$ von V_i gewählt wird:

Für ein (festes) Kontrollvolumen V_i sei ohne Einschränkung $\mathsf{s}_i = 0$. Für $q \in C^2(\mathsf{V}_i)$ gilt auf V_i die Taylor-Entwicklung in $x = 0 = \mathsf{s}_i$

$$q(x) = q_0 + q_1 \cdot x + O(\mathsf{h}_i^2)$$

mit $\mathsf{h}_i := \max_{x\in\mathsf{V}_i} |x|$. Die Anwendung der Quadraturformel (55) liefert

$$\begin{aligned}\int_{\mathsf{V}_i} q(x)dx - \mathsf{I}[q](\mathsf{V}_i) &= \int_{\mathsf{V}_i} (q_0 + q_1 \cdot x + O(\mathsf{h}_i^2))dx - (q_0 + q_1 \cdot \tilde{x} + O(\mathsf{h}_i^2))|\mathsf{V}_i| \\ &= \underbrace{q_0|\mathsf{V}_i| - q_0|\mathsf{V}_i|}_{=0} + \underbrace{q_1 \textstyle\int_{\mathsf{V}_i} x dx}_{=0} - \underbrace{q_1 \cdot \tilde{x}|\mathsf{V}_i|}_{=:R_1} + O(\mathsf{h}_i^2)|\mathsf{V}_i|.\end{aligned}$$

Der Term R_1 verschwindet, falls $\tilde{x} = 0$ (bzw. gleich dem Schwerpunkt) ist; wenn $\tilde{x}$ in V_i liegt, so gilt zumindest $R_1 = O(\mathsf{h}_i)|\mathsf{V}_i|$. Die Summation über das gesamte Gitter H_ℓ ergibt

$$\int_{\mathsf{P}_\ell(\Omega)} q(x)dx - \sum_{\mathsf{V}_i \in \mathsf{H}_\ell} \mathsf{I}_{\mathsf{H}_\ell}[q](\mathsf{V}_i) = O(\mathsf{h}_\ell^m)|\mathsf{P}_\ell(\Omega)|$$

mit der Gitter-Weite $\mathsf{h}_\ell := \max_{\mathsf{L}_i \in \mathsf{H}_\ell} \mathsf{h}_i$. Falls die Schwerpunkte s_i der Kontrollvolumina als Integrationspunkte gewählt werden, ist $m = 2$, anderenfalls gilt nur $m = 1$.

Implementierung. Die hier beschriebene Diskretisierung folgt der Realisierung für skalare Gleichungen im `ug`-Paket „`sc`“ *(scalar convection/diffusion)*, siehe auch [2]. Die Vorlage in `$(UGROOT)/../sc/pclib/stdfv.c` wurde auf Systeme von Gleichungen erweitert.

Terme in Divergenzgestalt. Für die Quadratur der Terme der Form

$$\mathsf{I}_\mathsf{H}[\nabla\cdot\underline{q}](\mathsf{V}_i)=\int\limits_{\partial\mathsf{V}_i}\underline{q}\cdot\underline{n}_i dx$$

wird $\partial\mathsf{V}_i$ in die Teilflächen *(sub control volume faces)* $\mathsf{F}_{ij}^k:=\partial\mathsf{V}_i\cap\partial\mathsf{V}_j\cap\mathsf{L}_k$ zerlegt:

$$\mathsf{I}_\mathsf{H}[\nabla\cdot\underline{q}](\mathsf{V}_i)=\sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\int_{\mathsf{F}_{ij}^k}\underline{q}\cdot\underline{n}_{ij}^k dx$$

Die F_{ij}^k sind für $\mathsf{N}=2$ Strecken, für $\mathsf{N}=3$ Vierecke. Mit $\underline{n}_{ij}^k$ wird die zu V_i äußere Normale auf F_{ij}^k bezeichnet, d.h. $\underline{n}_{ij}^k$ zeigt nach V_j, und $\underline{n}_{ji}^k=-\underline{n}_{ij}^k$. Für ebene F_{ij}^k ist $\underline{n}_{ij}^k$ auf F_{ij}^k konstant, und obige Gleichung kann geschrieben werden als

$$\mathsf{I}_\mathsf{H}[\nabla\cdot\underline{q}](\mathsf{V}_i)=\sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\underline{n}_{ij}^k\cdot\int_{\mathsf{F}_{ij}^k}\underline{q}dx.$$

Das Integral über $\underline{q}$ (bzw. $\underline{n}_{ij}^k\cdot\underline{q}$) wird nun durch eine Quadraturformel ausgewertet:

$$\widehat{\mathsf{I}}_\mathsf{H}[\nabla\cdot\underline{q}](\mathsf{V}_i)=\sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}|\mathsf{F}_{ij}^k|\underline{n}_{ij}^k\cdot\underline{q}(\tilde{x})$$

Die konkrete Gestalt der Diskretisierung hängt nun noch von der Wahl des Integrationspunktes $\tilde{x}\in\mathsf{F}_{ij}^k$ ab.

Zustandsvariable. $u(t)$ — Das Integral über das Kontrollvolumen

$$\mathsf{I}_\mathsf{H}[u(t)](\mathsf{V}_i)=\int\limits_{\mathsf{V}_i}u(x,t)dx$$

wird unmittelbar durch den Knotenwert $u(\mathsf{C}_i)$ approximiert:

$$\widehat{\mathsf{I}}_\mathsf{H}[u(t)](\mathsf{V}_i)=|\mathsf{V}_i|u(\mathsf{C}_i,t)=|\mathsf{V}_i|\mathsf{u}_i(t)$$

(Dieser Term wird nur bei der Diskretisierung des instationären Problems benötigt.)

Diffusionsterm. $\nabla\cdot(\underline{\underline{D}}\nabla u)$ — Die Zerlegung von $\partial\mathsf{V}_i$ in die Teilflächen F_{ij}^k ergibt:

$$\mathsf{I}_\mathsf{H}[\nabla\cdot(\underline{\underline{D}}\nabla u)](\mathsf{V}_i)=\sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\int_{\mathsf{F}_{ij}^k}[\underline{\underline{D}}\nabla u(x)]\cdot\underline{n}_{ij}^k dx$$

Mit der Darstellung von $u(x)$ nach (53) ergibt sich für konstantes $\underline{\underline{D}}$

$$\mathsf{I}_\mathsf{H}[\nabla\cdot(\underline{\underline{D}}\nabla u)](\mathsf{V}_i)=\sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\sum_{\mathsf{C}_\ell\in\mathsf{L}_k}\mathsf{u}_\ell\,\underline{n}_{ij}^k\cdot\underline{\underline{D}}\int_{\mathsf{F}_{ij}^k}\nabla(\mathsf{v}_\ell^k\circ\mathsf{T}_k^{-1})(x)dx.$$

Zur Quadratur wird der Gradient der Formfunktionen im Schwerpunkt x_{ij}^k der F_{ij}^k ausgewertet:

$$\widehat{\mathsf{I}}_{\mathsf{H}}[\nabla\cdot(\underline{\underline{D}}\nabla u)](\mathsf{V}_i) = \sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\sum_{\mathsf{C}_\ell\in\mathsf{L}_k} \mathsf{u}_\ell\,|\mathsf{F}_{ij}^k|\underline{n}_{ij}^k\cdot\underline{\underline{D}}\nabla(\mathsf{v}_\ell^k\circ\mathsf{T}_k^{-1})(x_{ij}^k)$$

Der Beitrag zur diskreten Steifigkeitsmatrix ist gegeben durch

$$\mathsf{D}_{ij}^k := -\sum_{\mathsf{E}_{i\ell}\in\mathsf{L}_k}|\mathsf{F}_{i\ell}^k|\underline{n}_{i\ell}^k\cdot\underline{\underline{D}}\nabla(\mathsf{v}_j^k\circ\mathsf{T}_k^{-1})(x_{i\ell}^k)\;,\quad \mathsf{D}_{ij}=\sum_{\mathsf{L}_k\ni\mathsf{C}_i}\mathsf{D}_{ij}^k.$$

Konvektionsterm. $\nabla\cdot(\underline{w}u)$ — Die Zerlegung von $\partial\mathsf{V}_i$ in die Teilflächen F_{ij}^k ergibt für konstantes $\underline{w}$:

$$\mathsf{I}_{\mathsf{H}}[\nabla\cdot(\underline{w}u)](\mathsf{V}_i) = \sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\int_{\mathsf{F}_{ij}^k}[\underline{w}u(x)]\cdot\underline{n}_{ij}^k dx = \sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\underline{w}\cdot\underline{n}_{ij}^k\int_{\mathsf{F}_{ij}^k}u(x)dx$$

Interpolierende Auswertung: $u(x)$ wird nach (53) durch die Formfunktionen interpoliert dargestellt. Für die Quadratur wird wie oben der Schwerpunkt von F_{ij}^k eingesetzt, es ergibt sich

$$\widehat{\mathsf{I}}_{\mathsf{H}}^{cen}[\nabla\cdot\underline{w}u](\mathsf{V}_i) = \sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\sum_{\mathsf{C}_\ell\in\mathsf{L}_k}|\mathsf{F}_{ij}^k|\underline{n}_{ij}^k\cdot\underline{w}\;\mathsf{u}_\ell\;\mathsf{v}_\ell^k(\mathsf{T}_k^{-1}(x_{ij}^k))\;,$$

der Beitrag zur diskreten Steifigkeitsmatrix ist

$$\mathsf{W}_{ij}^{cen} := \sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\sum_{\mathsf{C}_\ell\in\mathsf{L}_k}|\mathsf{F}_{ij}^k|\underline{n}_{ij}^k\cdot\underline{w}\;(\mathsf{v}_\ell^k\circ\mathsf{T}_k^{-1})(x_{ij}^k)\;.$$

Dies entspricht einer Verallgemeinerung der im Kontext der Differenzenverfahren als *zentrale Differenz* bekannten Diskretisierung. Diese hat schlechte Stabilitätseigenschaften [16], daher wird an ihrer Stelle die Upwind-Diskretisierung gebraucht.

Upwind-Auswertung: Es wird der Wert von u des Knotens eingesetzt, aus dessen Kontrollvolumen kommend das Strömungsfeld w die Grenzfläche überschreitet, d.h.

$$u(x)\Big|_{\mathsf{F}_{ij}^k} := \begin{cases}\mathsf{u}_i & (\underline{n}_{ij}^k\cdot\underline{w}\geq 0)\\ \mathsf{u}_j & (\underline{n}_{ij}^k\cdot\underline{w}<0)\end{cases}\;.$$

Damit ergibt sich

$$\widehat{\mathsf{I}}_{\mathsf{H}}^{up}[\nabla\cdot(\underline{w}u)](\mathsf{V}_i) = \sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}|\mathsf{F}_{ij}^k|\underline{n}_{ij}^k\cdot\underline{w}\;\mathsf{u}_\ell\qquad\begin{cases}\ell=i & (\underline{n}_{ij}^k\cdot\underline{w}\geq 0)\\ \ell=j & (\underline{n}_{ij}^k\cdot\underline{w}<0)\end{cases}$$

Der Beitrag zur diskreten Steifigkeitsmatrix ergibt sich als

$$\begin{aligned}\mathsf{W}_{ij}^{up} &= \sum_{\substack{\mathsf{L}_k\ni\mathsf{C}_i\\ \underline{w}\cdot\underline{n}_{ij}^k<0}}|\mathsf{F}_{ij}^k|\underline{n}_{ij}^k\cdot\underline{w}\;,\\ \mathsf{W}_{ii}^{up} &= \sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\substack{\mathsf{E}_{ij}\in\mathsf{L}\\ \underline{w}\cdot\underline{n}_{ij}^k>0}}|\mathsf{F}_{ij}^k|\underline{n}_{ij}^k\cdot\underline{w}\;.\end{aligned}$$

Reaktionsterm. $c(u)u$ — Dieser Term hat keine Divergenzform. Das Integral über das Kontrollvolumen

$$\mathsf{I}_\mathsf{H}[c(u)u](\mathsf{V}_i) = \int_{\mathsf{V}_i} c(u(x))u(x)dx$$

wird durch den Knotenwert $c(u(\mathsf{C}_i))u(\mathsf{C}_i)$ approximiert:

$$\widehat{\mathsf{I}}_\mathsf{H}[c(u)u](\mathsf{V}_i) = |\mathsf{V}_i|c(u(\mathsf{C}_i))\ u(\mathsf{C}_i)$$

Der Term

$$\mathsf{Z}_i(\mathsf{u}_i) := |\mathsf{V}_i|c(\mathsf{u}_i) \in \mathbb{R}^{\mathsf{M}\times\mathsf{M}}$$

trägt zur diskreten Steifigkeitsmatrix bei.

Quellterm. $\nabla \cdot \underline{f}$ — Die Zerlegung von $\partial\mathsf{V}_i$ in die Teilflächen F_{ij}^k liefert

$$\mathsf{I}_\mathsf{H}[\nabla\cdot\underline{f}](\mathsf{V}_i) = \sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}\int_{\mathsf{F}_{ij}^k}\underline{f}(x)\cdot\underline{n}_{ij}^k dx,$$

zur Quadratur wird der Wert von $\underline{f}$ im Schwerpunkt von F_{ij}^k eingesetzt:

$$\widehat{\mathsf{I}}_\mathsf{H}[\nabla\cdot\underline{f}](\mathsf{V}_i) = \sum_{\mathsf{L}_k\ni\mathsf{C}_i}\sum_{\mathsf{E}_{ij}\in\mathsf{L}_k}|\mathsf{F}_{ij}^k|\underline{n}_{ij}^k\cdot\underline{f}(x_{ij}^k).$$

Dieser Term trägt als

$$\mathsf{f}_i := \widehat{\mathsf{I}}_\mathsf{H}[\nabla\cdot\underline{f}](\mathsf{V}_i) \in \mathbb{R}^\mathsf{M}$$

zur diskreten rechten Seite bei.

3.3.4 Assemblierung

Die Approximation des diskreten Problems (54) durch numerische Integration läßt sich mit den im vorigen Abschnitt eingeführten Größen D_{ij}, W_{ij}, Z_{ij}, f_i, g_i schreiben als

$$\mathsf{Du} + \mathsf{Wu} + \mathsf{Z(u)u} = \mathsf{f} + \mathsf{g}.$$

Die Zusammenfassung dieser Größen zu $\mathsf{A(u)} := \mathsf{D} + \mathsf{W} + \mathsf{Z(u)}$, $\mathsf{b} := \mathsf{f} + \mathsf{g}$ führt auf das nichtlineare diskrete System

$$\mathsf{A(u)u} = \mathsf{b},$$

dessen Koeffizienten einem geeigneten Löser übergeben werden.

Da die Geometrie des FV-Gitters H_ℓ elementweise ausgerechnet und nicht vollständig gespeichert wird, läuft zur Berechnung der Koeffizienten der Diskretisierung eine Schleife über alle Elemente. Daher ist auch die separate (numerische) Integration über Sub-Kontrollvolumina $\mathsf{V}_i^k := \mathsf{V}_i \cap \mathsf{L}_k$ erforderlich.

Zur elementweisen „lokalen Assemblierung“ werden die lokalen Beiträge

$$\mathsf{A}_{loc}(\mathsf{L}_k) = \mathsf{D}^k + \mathsf{W}^k + \mathsf{Z}^k, \qquad \mathsf{b}_{loc}(\mathsf{L}_k) = \mathsf{f}^k + \mathsf{g}^k$$

für jedes Element L_k berechnet, anschließend werden diese über alle Elemente des Gitters summiert.

4 Iterative Gleichungslösung

4.1 Mehrgitterverfahren

4.1.1 Verfahren

Einordnung. Der Mehrgitteralgorithmus ist ein effizientes Verfahren zur numerischen Lösung der linearen Gleichungssysteme großer Dimension, die aus der Diskretisierung partieller Differentialgleichungen entstehen. Der Algorithmus koppelt Diskretisierungen auf Gittern verschiedener Weite zu einem iterativen Lösungsverfahren, das gegenüber der iterativen Lösung auf einem einzigen Gitter erhebliche Geschwindigkeitsvorteile bietet.

Das Mehrgitterverfahren ist in seiner Standardform seit etwa zweieinhalb Jahrzehnten bekannt, Eigenschaften und Varianten des Algorithmus wurden z.B. von Hackbusch [14] ausführlich beschrieben. Für die in dieser Arbeit dokumentierten Rechnungen wurden die Standardwerkzeuge der `ug`-Software benutzt. Daher soll das Mehrgitter-Verfahren hier nur kurz skizziert werden, um eine Einordnung der für die Modellrechnung verwendeten Algorithmen unter einer Vielzahl bekannter Varianten zu ermöglichen.

> In `ug` stehen *zwei* Mehrgitter-Algorithmen zur Verfügung: der *additive* und der *multiplikative* Algorithmus. Es ist nicht auf den ersten Blick zu erkennen, welcher der beiden „der“ klassische Mehrgitter-Algorithmus ist.
>
> Eine Beschreibung beider Algorithmen findet sich in der Arbeit von Bastian [2], aus der hervorgeht, daß der multiplikative Algorithmus mit dem klassischen übereinstimmt. Der additive Algorithmus wurde eingeführt, weil dieser durch die Einsparung von Kommunikationsaufwand erhebliche Vorteile bei der Parallelisierung bietet.

Idee. Klassische lineare Iterationsverfahren zur Lösung eines linearen Gleichungssystems $Ax = b$ reduzieren praktisch nur die zu großen Eigenwerten von A gehörenden „kurzwelligen“ Anteile des Defektes $d := Ax - b$.

Die „langwelligen“ Fehleranteile können separat eliminiert werden, indem das Problem der Defektreduktion in einen Raum transformiert wird, der einer Diskretisierung auf einem gröberen Gitter entspricht. In diesem Raum liegt für das Iterationsverfahren eine wesentlich bessere Konvergenzrate vor. Die Defektreduktion auf einem gröberen Gitter („Grobgitterkorrektur“) kann rekursiv, d.h. für mehr als zwei Gitter, ausgeführt werden.

4.1.2 Definition des linearen Mehrgitter-Zyklus

Gitter-Level. Gegeben seien Gitter $\mathsf{G}_0 \subset \mathsf{G}_1 \subset \cdots \subset \mathsf{G}_L$ zusammen mit Diskretisierungen des zu lösenden Problems auf jedem Gitter. (Für die Finite-Volumen-Diskretisierung werden entsprechend die dualen Gitter $\mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_L$ betrachtet.)

Die Diskretisierung des linearen Problems auf G_ℓ (bzw. H_ℓ) für ein festes $\ell \in \{0, 1, \ldots, L\}$

$$A_\ell x_\ell = b_\ell$$

wird als Gitter-Level bezeichnet. Die Lösung soll auf dem *maximalen Level* L ausgerechnet werden, die übrigen Level haben nur für Zwischenschritte des Algorithmus eine Bedeutung.

Zweigitter-Zyklus. Der Mehrgitter-Algorithmus $\mathcal{M}_{\ell-1}^{\ell}$ für den Spezialfall *zweier* Gitter $\mathsf{G}_{\ell-1}, G_\ell$ läßt sich durch die Aufeinanderfolge der folgenden Schritte beschreiben:

Berechne $x_\ell \mapsto \mathcal{M}_{\ell-1}^{\ell} x_\ell$ durch

1. Vorglättung
$$x_\ell^{pre} := \mathcal{S}_\ell x_\ell$$
2. Defekt[10]
$$d_\ell := A_\ell x_\ell^{pre} - b_\ell$$
3. Restriktion
$$d_{\ell-1} := \mathcal{R}_\ell d_\ell$$
4. Korrekturterm
$$c_{\ell-1} := \mathcal{L}_{\ell-1} d_{\ell-1}$$
5. Prolongation
$$c_\ell := \mathcal{P}_\ell c_{\ell-1}$$
6. Defektkorrektur
$$x_\ell^{cor} := x_\ell^{pre} - c_\ell$$
7. Nachglättung
$$\mathcal{M}_{\ell-1}^{\ell} x_\ell := x_\ell^{post} := \mathcal{S}_\ell x_\ell^{cor}$$

Komponenten. Als Komponenten des Zweigitter-Zyklus (und somit auf jedem *level* des Mehrgitter-Zyklus) treten neben den Koeffizienten A_ℓ, b_ℓ des auf G_ℓ diskretisierten Problems vier weitere Operatoren auf:

$\mathcal{S}_\ell$	Vor- und Nachglätter auf G_ℓ
$\mathcal{R}_\ell$	Restriktion von G_ℓ nach $\mathsf{G}_{\ell-1}$
$\mathcal{P}_\ell$	Prolongation von $\mathsf{G}_{\ell-1}$ nach G_ℓ
$\mathcal{L}_{\ell-1}$	Exakter oder näherungsweiser Löser auf $\mathsf{G}_{\ell-1}$

Mehrgitter-Zyklus. Der Mehrgitter-Zyklus $\mathcal{M}_0^L$ entsteht rekursiv aus den Mehrgitter-Zyklen $\mathcal{M}_0^{L-1}$ und $\mathcal{M}_{L-1}^L$, indem der Korrekturterm c_{L-1} in $\mathcal{M}_{L-1}^L$ nicht durch einen exakten Löser, sondern durch γ Iterationen des $\mathcal{M}_0^{L-1}$ berechnet wird: Formal setzen wir

$$\begin{aligned} \mathcal{L}_0 &:= A_0^{-1} \quad \text{und} \\ \mathcal{L}_\ell &:= (\mathcal{M}_0^\ell)^\gamma \quad \text{für } \ell = 1, \ldots, L-1. \end{aligned}$$

Die Rekursion bricht durch den Aufruf des sogenannten *Basislösers* $\mathcal{L}_{l_0} = A_{l_0}^{-1}$ anstelle des Mehrgitterzyklus $\mathcal{M}_0^\ell$ auf einem Gitter-Level $\ell = l_0$ ab, der in der formalen Definition ohne Einschränkung der Allgemeinheit zu $l_0 = 0$ gesetzt wurde. In der praktischen Realisierung können Gitter-Level unterhalb von l_0 existieren, die für die Gitter-Erzeugung benötigt werden, für den Mehrgitter-Algorithmus aber nicht relevant sind.

[10]Die Definition des Defektvektors $d := Ax - b$ entspricht der Darstellung u.a. bei Hackbusch [14] und Braess [5]. In der ug-Software werden ebenso wie bei Bastian [2] Defekt- und Korrekturvektor mit dem entgegengesetzten Vorzeichen versehen. Diese für die Theorie unerhebliche Vorzeichenkonvention muß beim Umgang mit der Software genau beachtet werden.

Mehrgitter-Iterationsverfahren. Durch wiederholte Anwendung des linearen Mehrgitterzyklus

$$x_L^{k+1} := \mathcal{M}_0^L x_L^k$$

(z.B. so lange bis $\|d_L^k\| < \delta$ für eine vorgegebenes Schranke δ erreicht ist) entsteht das lineare Mehrgitterverfahren.

> Die Unterscheidung von Iterationsschritt und Iterationsverfahren (Löser) erscheint auf dem Papier als überflüssiger Formalismus, die Trennung in `iter` und `linear_solver` erweist sich jedoch in der Software-Umsetzung als wesentlich.

Das Iterationsverfahren wird durch die folgenden Zahlenwerte parametrisiert:

γ	Anzahl der Iterationen des $\mathcal{M}_{l_0}^{\ell}$ pro Gitter-Level $\ell = l_0 + 1, \ldots, L-1$
ν_1, ν_2	Anzahl von Vor- und Nachglättungsschritten
l_0	Basis-Level: Gitter-Level, auf dem der Basislöser $\mathcal{L}_{l_0}$ aufgerufen wird
δ	Abbruchgenauigkeit[11]
m	Maximalzahl von Iterationsschritten des Gesamt-Zyklus $\mathcal{M}_{l_0}^L$

Die Parameter δ und m steuern den Mehrgitter-Löser, die übrigen den Mehrgitter-Zyklus.

4.1.3 Komponenten

Die in der formalen Definition als Operatoren $\mathcal{S}$, $\mathcal{R}$, $\mathcal{L}_0$, $\mathcal{P}$ eingeführten Prozeduren, die die Komponenten des Mehrgitter-Lösers bilden, sollen im folgenden näher charakterisiert werden.

Basis-Löser. Der Basislöser $\mathcal{L}_0$, formal definiert durch

$$\mathcal{L}_0 := A_0^{-1},$$

soll das lineare Problem $A_0 v_0 = d_0$ auf dem Basis-Level (0, allg. l_0) lösen. Hier können sowohl „exakte“ (direkte) als auch iterative Löser zum Einsatz kommen.

Glättungsoperator. Zur Vor- und Nachglättung werden iterative Gleichungslöser eingesetzt, die mit einem oder wenigen Iterationsschritten eine Reduktion der kurzwelligen (zu großen Eigenwerten von A gehörenden) Fehleranteile bewirken.

Als Vor- und Nachglätter können verschiedene Operatoren verwendet werden, für eine einfache Darstellung wurde oben derselbe Operator $\mathcal{S}$ benutzt.

Die folgenden Beispiele lassen sich basierend auf der Zerlegung der Matrix $A = D - L - U$ in eine Diagonalmatrix D, eine untere Dreiecksmatrix L und eine obere Dreiecksmatrix U darstellen:

Gedämpfte Jacobi-Iteration für $\omega \in (0, 1)$

$$x^{n+1} = \mathcal{S}_{JAC}(\omega) x^n := x^n - \omega D^{-1}(Ax^n - b)$$

[11] In `ug` wird anstelle einer absoluten Fehlerschranke δ die *relative* Reduktion `$red` des Residuums bezogen auf das Anfangs-Residuum vorgegeben.

Gauß-Seidel-Iteration *(abhängig von der Numerierung der Unbekannten!)*

$$x^{n+1} = \mathcal{S}_{GS}\, x^n := x^n - \omega(D-L)^{-1}(Ax^n - b)$$

Gitter-Transfer. Der Übergang zwischen den Diskretisierungen auf den einzelnen Gittern wird durch den *Restriktionsoperator* $\mathcal{R}$ und den *Prolongationsoperator* $\mathcal{P}$ realisiert. Eine in der Praxis übliche Wahl sind die folgenden kanonischen Operatoren.

Die *kanonische Prolongation* ergibt sich unmittelbar aus der Wahl der Basisfunktionen v_ℓ^i der Ansatzräume $\mathsf{U}_\ell = \operatorname{span}\{(\mathsf{v}_\ell^i)_{i=1}^{\mathsf{n}(\ell)}\}$ auf den einzelnen Gittern G_ℓ (bzw. H_ℓ). Unter der Voraussetzung $\mathsf{U}_{\ell-1} \subset \mathsf{U}_\ell$ ist jede Funktion $\mathsf{v} \in \mathsf{U}_{\ell-1}$ mit der Darstellung

$$\mathsf{v} = \sum_{i=1}^{\mathsf{n}(\ell-1)} \mathsf{u}_{\ell-1}^i \mathsf{v}_{\ell-1}^i$$

auf *eindeutige* Weise durch die Basisfunktionen des U_ℓ als

$$\mathsf{v} = \sum_{i=1}^{\mathsf{n}(\ell)} \mathsf{u}_\ell^i \mathsf{v}_\ell^i$$

darstellbar. Die somit implizit definierte Abbildung

$$\begin{aligned} \mathcal{P}_\ell : \mathbb{R}^{\mathsf{n}(\ell-1)} &\to \mathbb{R}^{\mathsf{n}(\ell)} \\ \mathsf{u}_{\ell-1}(\mathsf{v}) &\mapsto \mathsf{u}_\ell(\mathsf{v}) \end{aligned}$$

zwischen den Koeffizientenvektoren $\mathsf{u}_\ell := (\mathsf{u}_\ell^i)_{i=1}^{\mathsf{n}(\ell)}$ auf den einzelnen Gittern G_ℓ (bzw. H_ℓ) heißt *kanonische Prolongation*.

Die *kanonische Restriktion* ergibt sich als Adjungierte der kanonischen Prolongation:

$$\mathcal{R}_\ell := \mathcal{P}_\ell^*.$$

4.2 Newton-Löser

4.2.1 Verfahren

Definition. Das Newton-Verfahren ist ein iteratives Lösungsverfahren für nichtlineare Gleichungssysteme der Form

$$\underline{F}(\underline{u}) = \underline{g} \tag{56}$$

mit stetig differenzierbarer Abbildung $\underline{F} : \mathbb{R}^{\mathsf{M}} \to \mathbb{R}^{\mathsf{M}}$ unter Verwendung der Inversen der Jacobi-Matrix $\underline{\underline{J}}_F$ von $\underline{F}$:

$$\underline{u}^{k+1} := \underline{u}^k - \underline{\underline{J}}_F^{-1}(\underline{u}^k)(\underline{F}(\underline{u}^k) - \underline{g}) \tag{57}$$

Das Newton-Verfahren (57) ist lokal konvergent, d.h. es konvergiert für Startwerte aus einer Umgebung der (bzw. einer lokal eindeutigen) Lösung $\underline{u}$ von (56).

Implementierung. Da die Ausrechnung der Jacobi-Inversen für große M zu aufwendig ist, tritt an ihre Stelle ein (ggf. näherungsweiser) linearer Gleichungslöser $\mathcal{L} = \mathcal{L}(\underline{\underline{J}})$.

Die gesuchte Lösung wird durch eine Folge von *Iterierten* $\underline{u}^k$ approximiert, die nach der folgenden Vorschrift berechnet werden:

1. Berechne den *Defektvektor*[12] $\underline{d}$:

$$\underline{d}^k := \underline{F}(\underline{u}^k) - \underline{g}$$

2. Berechne den *Korrekturvektor* $\underline{v}$ als (Näherungs-)Lösung des linearen Systems $\underline{d} = \underline{\underline{J}}\,\underline{v}$:

$$\underline{v}^k := \mathcal{L}(\underline{\underline{J}}(\underline{u}^k))\underline{d}^k$$

3. Korrigiere den Lösungsvektor $\underline{u}$:

$$\underline{u}^{k+1} := \underline{u}^k - \lambda \underline{v}^k$$

Durch die Wahl eines geeigneten Dämpfungsfaktors λ (im Standardfall gleich 1) wird die Konvergenz des Verfahrens gesichert bzw. die Konvergenzgeschwindigkeit beeinflußt.

4.2.2 Algorithmische Details

Für die folgenden Ausführungen definieren wir den (skalaren) *Defekt der k-ten Iterierten*

$$s^k := \|\underline{d}^k\|$$

und die *Reduktion* des Defektes

$$\rho^k := \frac{s^k}{s^{k-1}}.$$

Abbruch. Die Newton-Iteration wird erfolgreich abgebrochen, wenn der Defekt s^k einen gewünschten Wert unterschreitet.

Für einen erfolglosen Abbruch des Algorithmus kommen verschiedene Gründe in Betracht:

- der lineare Löser $\mathcal{L}$ konvergiert nicht (praktisch: der lineare Defekt erreicht nach einer vorgegebenen Anzahl von Schritten nicht die gewünschte Schranke),
- die Liniensuche (falls angewendet) wird nicht akzeptiert,
- eine Maximalzahl von Schritten wurde (durch Newton selbst) erreicht.

Reassemblierung. Da die Ausrechnung der lösungsabhängigen Jacobi-Matrix $\underline{\underline{J}}$ aufwendig sein kann, wird diese nur bei Bedarf durchgeführt, d.h. wenn die „alte“ Jacobi-Matrix diejenige zum modifizierten Lösungsvektor zu schlecht approximiert. In der `ug`-Implementierung wird als Kriterium hierfür verwendet, daß die Reduktion ρ^k schlechter als eine vorgebbare Schranke ausfällt. Die Reassemblierung der Jacobi-Matrix wird in jedem Schritt vorgenommen, wenn eine quadratische Konvergenzrate der Iteration angenommen wird.

Die Ausrechnung des Defektvektors $\underline{d}$ in jedem Newton-Iterationsschritt (und sogar in jedem Schritt der Liniensuche) ist unumgänglich.

[12] Um Verwechslungen mit dem im linearen Mehrgitterzyklus (oder anderswo) auftretenden Defekt zu vermeiden, wird *dieser* Defektvektor im Zweifelsfalle als *nichtlinearer Defekt* bezeichnet.

Steuerung der linearen Fehlerreduktion. Der quadratischen Konvergenz des Newton-Algorithmus steht die nur lineare Konvergenz des Mehrgitterverfahrens gegenüber [14]. Wenn in jedem Newton-Schritt genau ein Mehrgitterzyklus ausgeführt wird, dominiert mit abnehmendem Fehler der Defekt der linearen Iteration. Zur Kompensation sind mehrere Mehrgitter-Schritte pro Newton-Schritt erforderlich, die nötige Anzahl von Schritten ist schwer a priori abzuschätzen.

In der praktischen Situation ist der lineare Gleichungslöser $\mathcal{L}$ insofern parametrisierbar, als eine Schranke für den linearen Defekt (d.h. für die Norm des Defektvektors $\underline{e} := \underline{\underline{J}}\,\underline{v} - \underline{d}$) vorgegeben werden kann. Diese Schranke muß um so kleiner gewählt werden, je kleiner der nichtlineare Defekt wird, damit stets $\|\underline{e}\| < \epsilon\|\underline{d}\|$ für ein hinreichend kleines ϵ erfüllt ist. Dies wird durch den Newton-Löser in ug selbsttätig realisiert.

Dämpfung. Das Konvergenzgebiet des Newton-Verfahrens kann durch Unterrelaxation, d.h. durch die Verwendung eines Dämpfungsparameters $\lambda < 1$, vergrößert werden. Das Problem der Wahl eines geeigneten λ kann durch einen numerischen Algorithmus gelöst werden: Bei der *Liniensuche* wird auf der Menge $\{\underline{u} - \lambda\underline{v} : \lambda \in (0, 1]\}$ ein Punkt $\underline{\tilde{u}}$ gesucht, der die Norm des Defektes $\underline{d}(\underline{\tilde{u}}) = \underline{F}(\underline{\tilde{u}}) - \underline{g}$ minimiert.

In der ug-Implementierung kann der Dämpfungsparameter λ fest vorgegeben werden, wobei Werte außerhalb des Intervalls $(-2, +2)$, für die keine Konvergenz vorliegt, von ug zurückgewiesen werden. Eine automatische Steuerung der Dämpfung wird durch die Liniensuche realisiert.

Liniensuche. Die Liniensuche kann mit mehr oder weniger großem Aufwand mehr oder weniger genau durchgeführt werden. Ein vergleichsweise hoher numerischer Aufwand für diese Suche lohnt sich dann, wenn der Aufwand für die (näherungsweise) Inversion der Jacobi-Matrix wesentlich größer als der Aufwand der zur Liniensuche benötigten Vektor-Additionen, Matrix-Multiplikationen und wiederholten Ausrechnungen (der Norm) des Defektvektors ist.

In der ug-Implementierung wird für eine Folge von Dämpfungswerten λ_j^k, $j = 1, \ldots, m$ der zugehörige Fehler

$$s_j^k := \|\underline{F}(\underline{u}^k - \lambda_j^k \underline{v}^k) - \underline{g}\|$$

errechnet und anschließend überprüft, ob die Stabilitätsbedingung

$$\frac{s_j^k}{s^k} \leq 1 - \frac{1}{4}|\lambda_j^k| \tag{58}$$

erfüllt ist. Sobald dies für ein $j \leq m$ der Fall ist, wird die Liniensuche abgebrochen und der Wert $\lambda = \lambda_j^k$ akzeptiert, d.h. es wird

$$\underline{u}^{k+1} := \underline{u}^k - \lambda_j^k \underline{v}^k$$

gesetzt (vgl. [5]).

Der Newton-Löser in ug stellt verschiedene Varianten der Liniensuche bereit:

Variante 0: Es wird keine Liniensuche durchgeführt, sondern mit einem festen Wert $\lambda = \lambda_0$ gerechnet.

Variante 1: Ausgehend von einem Startwert $\lambda_1^k := \lambda^{k-1}$ (bzw. λ_0) wird der Dämpfungsparameter fortgesetzt halbiert: $\lambda_{j+1}^k := \frac{1}{2}\lambda_j^k$.

Falls die Bedingung (58) nach m Schritten durch kein λ_j^k, $j = 1, \ldots, m$ erfüllt war, wird die Liniensuche (und damit der Newton-Algorithmus) erfolglos abgebrochen.

Variante 2: Der Dämpfungsparameter λ wird wie bei Variante 1 fortgesetzt halbiert; anstelle eines erfolglosen Abbruches wird unter allen λ_j^k dasjenige akzeptiert, das den kleinsten Fehler s_j^k liefert.

Variante 3: Der Parameter λ wird mittels eines (undokumentierten) Iterationsverfahrens angepaßt. Die Suche wird abgebrochen, sobald die Bedingung (58) erfüllt ist, im erfolglosen Falle wird der zuletzt berechnete Parameter λ_m^k akzeptiert.

Skalierung des Defektvektors. Der Defekt der einzelnen Lösungskomponenten kann bei Bedarf skaliert werden: Der Defekt wird in der ug-Implementierung komponentenweise berechnet, der Gesamt-Defekt ergibt sich als euklidische Norm der ggf. skalierten Defekt-Komponenten.

4.2.3 Assemblierung

Die Assemblierung hat die Aufgabe, wiederholt die lösungsabhängigen Werte von $\underline{d}(\underline{u})$ und $\underline{\underline{J}}(\underline{u})$ aus der Diskretisierung bereitzustellen.

Das diskretisierte Modellproblem hat die Form

$$\mathsf{Du} + \mathsf{Wu} + \mathsf{Z(u)u} = \mathsf{f} + \mathsf{g}.$$

Dies wird zusammengefaßt zu

$$\mathsf{A(u)u} = \mathsf{b},$$

diese Gleichung wird mit $\underline{F}(\underline{u}) = \underline{g}$ nach (56) identifiziert und durch den Newton-Algorithmus iterativ gelöst.

Die Jacobi-Matrix von $\underline{F}(\underline{u}) \equiv \mathsf{A(u)u}$ ist

$$\mathsf{J(u)} = \mathsf{A(u)} + \mathsf{A'(u)u},$$

diese muß im Newton-Verfahren wiederholt ausgerechnet werden. Da D und W nicht von u abhängen, ergibt sich $\mathsf{J(u)}$ als

$$\mathsf{J(u)} = \mathsf{D} + \mathsf{W} + \mathsf{Z(u)} + \mathsf{Z'(u)u}.$$

Die aus der Diskretisierung der Ortsableitungen entstehenden Matrizen D und W stehen unmittelbar aus der Finite-Volumen-Diskretisierung zur Verfügung.

Die Ableitung des diskreten Reaktionstermes $\mathsf{Z'(u)}$ wird durch numerische Differentation ermittelt, wobei die Diagonalgestalt von $\mathsf{Z(u)}$ bezüglich der Geometrie ausgenutzt wird: $\mathsf{Z(u)}$ besteht aus Diagonalblöcken, die jeweils einem geometrischen Objekt zugeordnet sind.

Der im Newton-Löser eingesetzte lineare Mehrgitter-Löser benötigt die Koeffizienten der Diskretisierungen auf allen für den Mehrgitter-Algorithmus relevanten Gittern. Die oben verwendeten Symbole A, J usw. stehen daher nicht für *eine* Matrix, sondern für eine durch die Menge der Gitter-Level $\{l_0, \ldots, L\}$ indizierte Menge von Matrizen.

5 Numerische Prozeduren in ug

5.1 Das Programmpaket ug

Das Programmpaket ug ist als akademisches Projekt mit der Zielstellung entstanden, einen „Baukasten“ möglichst universell einsetzbarer Software-Bausteine zur Unterstützung numerischer Methoden auf *unstrukturierten Gittern* zu entwickeln. Für zahlreiche Problemklassen stehen problemspezifische Bibliotheken zur Verfügung, die vom Anwender erweiterbar sind. Eine wesentliche Eigenschaft von ug, die Unterstützung der Parallelisierung auf verschiedenen Rechnerarchitekturen, wird in der vorliegenden Arbeit nicht berücksichtigt.

Das Interdisziplinäre Zentrum für Wissenschaftliches Rechnen der Universität Heidelberg stellt ug als C-Quellcode für wissenschaftliche Zwecke unentgeltlich zur Verfügung.

Das Konzept des „unstrukturierten Gitters“ wird ohne Frage mit einem gewissen Aufwand erkauft. Stärker als die durch die Verwaltung der komplexen Datenstrukturen bedingte zusätzliche Rechenzeit tritt der Aufwand der erforderlichen Einarbeitung in die sehr komplexen Software-Strukturen in Erscheinung. ug verfügt nicht über eine Oberfläche, hinter der die für die numerische Rechnung irrelevanten Details der Software verschwinden.

Dem unerfahrenen Anwender, der sich nicht mit dem Nachvollziehen der mitgelieferten Programmierbeispiele zufrieden gibt, sondern das Ziel verfolgt, die Leistungsfähigkeit des Paketes auszunutzen, stellen sich zahlreiche Probleme in den Weg, die aus zweierlei Sicht verständliche Ursachen haben:

ug wurde (zum großen Teil) von Mathematikern für Mathematiker geschaffen.

> In der Dokumentation wird daher auf zahlreiche Details nicht hingewiesen, die zwar aus mathematischer Sicht trivial sind, deren Nichtbeachtung aber schwere (und auf den ersten Blick nicht erkennbare) Konsequenzen nach sich ziehen kann.

ug wird ständig (und von verschiedenen Personen gleichzeitig) weiterentwickelt.

> Aus der Sicht des an dieser Entwicklung unbeteiligten Anwenders ist nicht zu erkennen, welche Teile des sehr umfangreichen Programmpaketes zum gefestigten Kern gehören (und damit hinreichend erprobt, dokumentiert und vor unangekündigten Veränderungen sicher sind) und welche sich eventuell noch im experimentellen Stadium befinden. Beide Teile stehen unkommentiert nebeneinander.
>
> > So arbeiten Teile des Paketes nicht korrekt in drei Raumdimensionen, zumindest nicht mit allen Typen von (Referenz)Elementen, ohne daß dies vermerkt ist.
>
> Die konsistente Erweiterung der modularen Software-Strukturen erfordert die Einhaltung gewisser Konventionen, die in der Dokumentation leider nicht erklärt werden.
>
> > Konzepte der objektorientierten Programmierung sind angelegt (vermutlich aus historischen Gründen in C und nicht in C++), werden aber an einigen Stellen wieder durchbrochen.

Die vorliegende Arbeit will dieser Tatsache Rechnung tragen, indem sie (zum Teil „triviale“) Details erklärt (diese Erklärungen können ergänzend in die Dokumentation aufgenommen werden), und auf eine Reihe von Unvollkommenheiten im Programmtext und in der Dokumentation hinweist.

5.2 ug-spezifische Software-Strukturen

Eine Reihe von Programmierkonzepten ist ug-spezifisch oder zumindest nicht zwingend in numerischer Software vorhanden. Für den Umgang mit ug ist ein grobes Verständnis dieser Konzepte unumgänglich. Es handelt sich dabei um Konzepte, die dem Mathematiker (und um solche sollte es sich bei dem überwiegenden Teil der ug-Nutzer handeln) aus seiner Ausbildung nicht geläufig sind. Leider fehlt in der Dokumentation eine Einführung in diese Konzepte. Einige der grundlegenden Strukturen sollen im folgenden erklärt werden.

5.2.1 Environment-Baum

Als *Environment* wird üblicherweise eine Sammlung von Variablen bezeichnet, die von der aufrufenden Umgebung an ein Programm übergeben werden. Insofern ist das ug-Environment Umgebung bezüglich einzelner Prozeduren, nicht bezüglich ug.

Der *Environment-Baum* in ug ist eine (mit einem einfachen Dateisystem vergleichbare) Baumstruktur, in der prinzipiell beliebige Daten abgelegt und später anhand ihres Namens wieder aufgefunden werden können.

Der Vergleich mit einem Dateisystem hinkt insofern, als die für ein solches typischen Operationen nicht unmittelbar verfügbar sind: auf dem Kommandozeilen-Niveau ist es nicht möglich, beliebige Einträge zu erzeugen und zu löschen, umzubenennen oder in ein anderes Verzeichnis zu kopieren bzw. zu verschieben.

Per Konvention werden Daten bestimmten Typs in einem für diese bestimmten Verzeichnis abgelegt (und von ug auch nur in diesem erwartet). Diese Konvention kann allerdings vom Anwender ohne weiteres verletzt werden.

Die im Zusammenhang mit dem Environment-Baum ablaufenden Vorgänge sind für den Anwender zum großen Teil nicht sichtbar. Erstellen, Verschieben und Löschen von Einträgen sind als C-Funktionen verfügbar und werden „unbemerkt“ ausgeführt. Von der Kommandozeile aus sind die Namen der einzelnen Einträge sichtbar, jedoch keinerlei Informationen über deren Größe oder Inhalt.

Der dem Environment zugeteilte Speicher ist üblicherweise relativ klein. Daher stehen im Environment nicht die numerischen Daten selbst, sondern nur sogenannte *Deskriptoren* (siehe unten).

Ein weiterer Unterschied zwischen einem gewöhnlichen Dateisystem und dem ug-Environment betrifft die Verzeichnisse. In ersterem ist ein Eintrag *entweder* Verzeichnis (d.h. enthält Referenzen auf andere Einträge) *oder* enthält Nutzdaten; die Verzeichnisse im ug-Environment enthalten beides. Damit ist der Datentyp `ENVVAR` eigentlich überflüssig, er wird vermutlich deshalb beibehalten, weil er um die Größe eines Zeigers kleiner ist.

> Beim Versuch, die Unterverzeichisse von `best full refrule` zu lesen, stürzt die ug-Anwendung mit einer Segmentverletzung ab. Die besagten Verzeichnisse sind leer! Der Lesezugriff wird daher nur aus Neugier erfolgen, oder (wie beim Autor dieser Arbeit) bei dem Versuch, das gesamte Environment systematisch zu erforschen.
>
> Offensichtlich wurden durch den Programmierer des *refinement rule manager* die Datentypen `ENVDIR` und `ENVVAR` verwechselt.

5.2.2 Mehrgitter und Datendeskriptoren

Mehrgitter. Eine Mehrgitter-Struktur (`MULTIGRID`) vereinigt sämtliche Daten einer konkreten Diskretisierung eines gegebenen Problems:

- Datenformatbeschreibung (`FORMAT`): Art und Anzahl der Vektorkomponenten, ...
- Randwertproblem (`BVP`): Beschreibung des Gebietes, der Randbedingungen etc.
- die einzelnen Gitter-Level (`GRID`)

Die Mehrgitter-Strukturen sind unter ihrem Namen (der vom Anwender durch das Kommando `new <mg-name>` vergeben wird) im Environment als `/Multigrids/<mg-name>` abgelegt. Vektor- und Matrixdeskriptoren sowie die sog. *Objekte*[13] der `NUMPROC`-Klassen sind als Untereinträge des Mehrgitters im Environment zugänglich.

Es ist ohne weiteres möglich, zugleich mehrere Mehrgitter-Objekte im Speicher zu halten. In diesem Falle ist mit besonderer Sorgfalt vorzugehen, da das Mehrgitter bei Kommandozeilen-Operationen implizit bezeichnet wird. Eine unbeabsichtigt auf dem falschen Mehrgitter ausgeführte Operation kann fatale Folgen nach sich ziehen.

Datendeskriptoren. Die einzelnen Vektoren und Matrizen werden in `ug` nicht als kompakte Strukturen abgelegt. Zusammenhängende Speicherblöcke sind nicht einem Vektorsymbol, sondern einem geometrischen Objekt zugeordnet, wodurch die Parallelisierung der Software erleichtert wird.

Der Zugriff auf die Daten erfolgt über sogenannte Deskriptoren. Diese kennen die Anordnung der zu einem Vektor (einer Matrix) gehörenden Daten. Sie werden in den Environment-Verzeichnissen `/Multigrids/<mg-name>/Vectors` bzw. `Matrices` abgelegt.

Vektoren (bzw. Matrizen) können bei Bedarf mittels der Kommandos `createvector` (bzw. `creatematrix`) angefordert werden. Bei der Bereitstellung wird lediglich ein freier Deskriptor (falls vorhanden) unter dem vorgegebenen oder generierten Namen im Environment eingetragen.

Die Anzahl verfügbarer Deskriptoren wird durch das sog. Format festgelegt. Der zu den Vektoren (Matrizen) gehörende Speicher wird entsprechend der Format-Spezifikation beim Erzeugen eines neuen Gitter-Levels durch den **User Data Manager** reserviert.

Format und Data Templates. Jeder Vektor- bzw. Matrixdeskriptor in `ug` wird anhand einer Vorlage *(template)* erstellt. Sämtliche für ein bestimmtes Problem benötigten *templates* werden in einem sog. *Format* (erstellt durch `newformat`, im Environment unter `/Formats/`) zusammengefaßt, dieses bestimmt zugleich die Anzahl der je Vorlage verfügbaren Datendeskriptoren. Ein Mehrgitter-Objekt ist stets auf ein bestimmtes Format bezogen.

Ein *vector template* definiert, ggf. separat für jedes Teilgebiet, welchen Typen von geometrischen Objekten (Knoten, Kanten, ...) wieviele Freiheitsgrade zugeordnet sind. Damit ist die Größe der zum Vektor gehörenden Datenbereiche festgelegt. Für die Matrizen gibt es entsprechende *matrix templates*. Beide finden sich unter `/Formats/<fmt-name>/` wieder.

[13] Mit dem Namen `Objects` werden in `ug` ausschließlich die parametrisierten Instanzen der `NUMPROC`-Klassen angesprochen, obwohl Objekte ebenso von anderen Klassen existieren.

Die einzelnen Vektorkomponenten werden jeweils durch einen einzelnen Buchstaben bezeichnet. Sie können bei Bedarf durch die Definition von *sub vector templates* unter einem eigenen Namen angesprochen werden. Die anhand von *sub vector templates* erzeugten Vektor-Deskriptoren greifen auf einen Teil des dem übergeordneten Vektor zugeordneten Datenbereiches zu, d.h. für Sub-Deskriptoren wird kein eigener Speicher reserviert.

5.2.3 Numproc-Klassen

Die in `ug` verfügbaren numerischen Prozeduren können zur Laufzeit parametrisiert und (natürlich unter Einhaltung bestimmter Regeln) beliebig miteinander kombiniert werden.

Klassenhierarchie. Die numerischen Prozeduren werden in `ug` als *Klassen* zur Verfügung gestellt.[14] Diese ordnen sich in eine dreistufige Hierarchie ein:

- Abstrakte Interfaces
- Implementierte Klassen
- Parametrisierte Instanzen

Diese Begriffe werden im Abschnitt 5.3.1 näher erläutert. Das UML-Klassendiagramm Abb. 5 (zur Syntax siehe 5.3.2) zeigt ein Beispiel für diese Einordnung.

Namen. Alle numerischen Prozeduren werden zur Laufzeit durch einen Namen aus drei durch Punkte getrennten Bestandteilen identifiziert.

Der erste Namensbestandteil bezeichnet das (abstrakte) Interface. Er entscheidet darüber, ob eine numerische Prozedur in einer bestimmten Rolle eingesetzt werden kann: linearer Iterationsschritt, Fehlerschätzer... Es wird unterstellt, daß eine Klasse mit einem bestimmten Interface-Namen bestimmte Funktionen zur Verfügung stellt (exportiert).

Der zweite Namensbestandteil bezeichnet eine (implementierte) Klasse, hinter der sich eine konkrete Funktionalität verbirgt (Gauß-Seidel-Iterationsschritt, ...).

Die Konstruktoren aller installierten Numproc-Klassen sind im Environment-Baum unter `/NumProcClasses/<`*`interface`*`>.<`*`class`*`>` (d.h. mit einem zweiteiligen Namen) abgelegt.

Durch den Aufruf von `npcreate <`*`instance`*`> $c <`*`class`*`>` wird eine Instanz der Klasse `<`*`class`*`>` erzeugt und im Verzeichnis `/Multigrids/<`*`mg`*`>/Objects/` des *aktuellen Mehrgitters* `<`*`mg`*`>` unter dem (dreiteiligen) Namen `<`*`interface`*`>.<`*`class`*`>.<`*`instance`*`>` abgelegt. Mittels `npinit <`*`instance`*`> <`*`Parameter...`*`>` werden dieser Parameter zugewiesen, es entsteht eine parametrisierte Instanz.

Die Identifizierung der Klassen bzw. Instanzen erfolgt jeweils nur über *einen* Namensbestandteil, d.h. alle Namenskomponenten müssen für sich eindeutig sein.

[14] Die in der `ug`-Dokumentation verwendeten Begriffe „Klasse“, „Objekt“, „abstrakte Basisklasse“ entstammen dem Kontext der *objektorientierten Programmierung*, sie werden im Abschnitt 5.3.1 erklärt. — Das Wort „Prozedur“ ist *nicht* im Sinne von `PASCAL` „`procedure`“ zu verstehen.

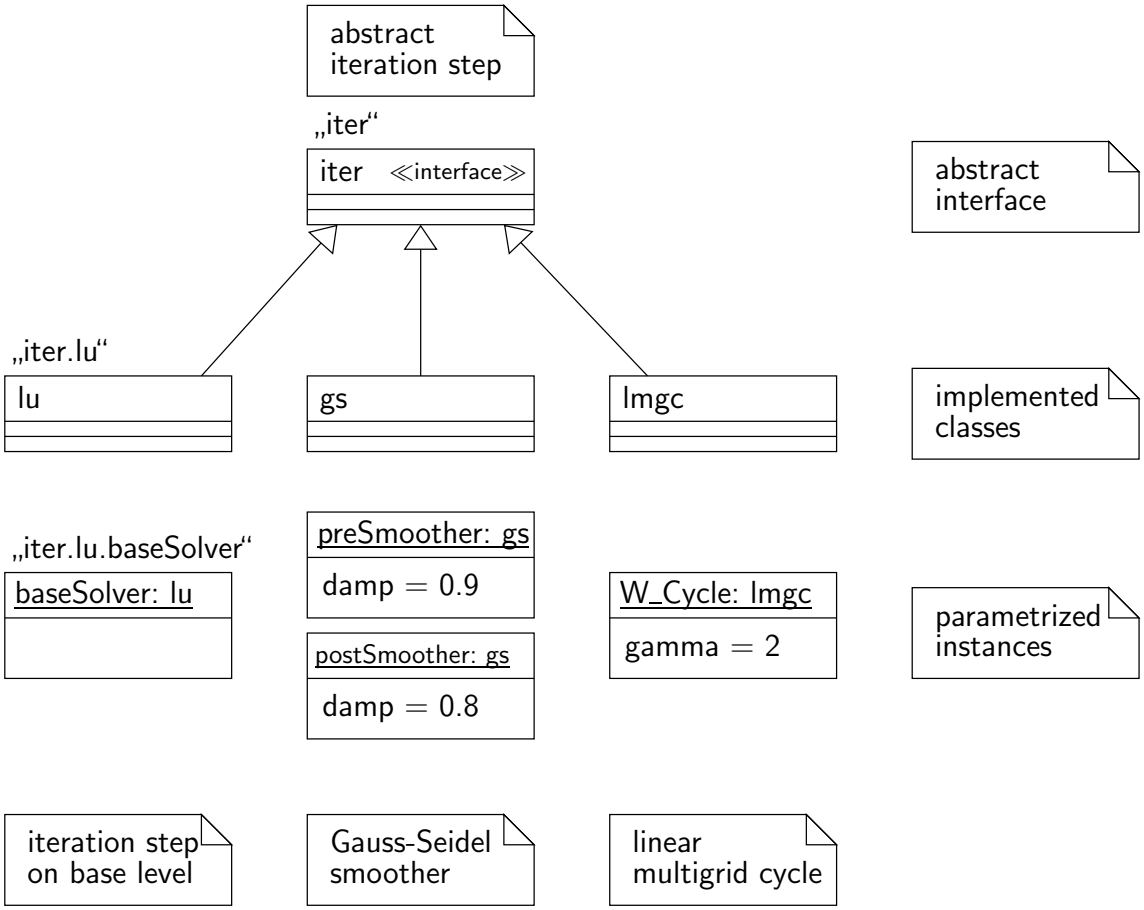


Abbildung 5: Hierarchie der Numproc-Klassen

5.3 Numerische Prozeduren und Objektorientierung

Objektorientierung in ug. Die Umsetzung der numerischen Algorithmen in der ug-Software folgt Konzepten der *objektorientierten Programmierung*. Dadurch wird eine flexible Austauschbarkeit und einfache Parametrisierbarkeit der einzelnen Komponenten erreicht.

Die objektorientierte Programmierung wird durch die Programmiersprache C nicht unmittelbar unterstützt, daher werden die benötigten Strukturen durch Elemente aus C nachgebildet. Die zusammengesetzten Objektstrukturen können jederzeit (gewollt oder ungewollt) mit unabsehbaren Folgen unterlaufen werden, um dies zu verhindern, macht sich die Einhaltung gewisser Konventionen bei der Erstellung, Nutzung und Modifikation der Quelltexte erforderlich.

Leider werden diese Konventionen in der ug-Dokumentation nicht erwähnt. Daher soll im folgenden ein Versuch unternommen werden, die ursprüngliche Intention nachzuzeichnen.

> Objektorientierung findet sich auch in anderen Teilen von ug: `OUTPUTDEVICE`, `PLOTOBJTYPE` usw. Nicht in jedem Falle kann aus Bezeichnern wie `someObject` hierauf geschlossen werden: z.B. ist ein `DRAWINGOBJ` lediglich ein Strom von `DOUBLE`-Zahlen, die als Beschreibung einer Grafik interpretiert werden.

Unified Modeling Language. Die ug-Dokumentation besteht aus einzelnen *manual pages*, die jeweils eine C-Funktion oder Datenstruktur im Detail beschreiben. Eine übersichtliche Darstellung des Zusammenwirkens der einzelnen Komponenten fehlt dabei; eine solche Beschreibung kann auch verbal kaum übersichtlich gegeben werden.[15]

Die funktionellen Zusammenhänge der einzelnen Komponenten lassen sich wesentlich besser grafisch veranschaulichen, wobei eine geeignete, allgemein verständliche Notation benötigt wird. Eine solche steht mit der *Unified Modeling Language (UML)* zur Verfügung. Die UML definiert verschiedene Typen von Diagrammen, die jeweils einen bestimmten Aspekt objektorientierter Software darstellen. Die komplexe Hierarchie der `numproc`-Klassen aus `ug` läßt sich relativ übersichtlich als UML-Klassendiagramm darstellen.

Als eine kompakte Zusammenfassung der UML, in der es trotz der Kürze gelungen ist, die Sichtweisen verschiedener Autoren kritisch gegenüberzustellen, sei [7] empfohlen.

UML-Diagramme stellen nicht notwendig alle Details dar. Sie eignen sich daher zur übersichtlichen Darstellung von Zusammenhängen ebenso wie zur Hervorhebung ausgewählter Aspekte. In diesem Sinne ersetzen sie nicht die detaillierte Darstellung einer *manual page*.

5.3.1 Begriffe

Um im Kontext der numerischen Mathematik über objektorientierte Programmierung zu sprechen, macht sich die Einführung einiger Grundbegriffe erforderlich.

Klasse ein strukturierter Datentyp, der Datenfelder (Variablen, UML: *attributes*) und Funktionen (Operationen, UML: *methods*) vereint. Die einer Klasse eigenen Operationen können auf die Datenfelder der jeweiligen Instanz zugreifen, ohne daß diese als Parameter des Funktionsaufrufes spezifiziert werden müssen.

Objekt *oder* Instanz Variable vom Typ einer Klasse, d.h. eine konkrete Realisierung, die Speicher belegt.

Sichtbarkeit Der Zugriff auf einzelne Elemente eines Klassen-Objektes kann nur dessen eigenen Funktionen gestattet sein: die entsprechenden Elemente heißen geschützt *(protected)* bzw. privat *(private)*. Praktisch bedeutet dies: die Daten können von außen nur durch bestimmte (öffentliche) Funktionen in definierter (konsistenter) Weise geändert werden. Auf diese Weise kann eine Klasse die Konsistenz ihrer Daten sicherstellen.

Die ug-Realisierung der Numproc-Klassen in C läßt keine Zugriffsbeschränkungen (private, protected) für einzelne Klassenelemente zu!

Vererbung (inheritance) Eine Kind-Klasse *(abgeleitete Klasse)* übernimmt die Eigenschaften (Elemente) einer Eltern-Klasse *(Basisklasse)* und fügt eigene hinzu.

Mehrere Kind-Klassen können von einer Eltern-Klasse erben, eine Kind-Klasse von mehreren Eltern-Klassen. Letzterer Fall wird als Mehrfachvererbung bezeichnet. Im Gegensatz zur biologischen (zufällige Aufteilung und Kombination des Erbgutes) bzw. juristischen Vererbung (Auseinandersetzung zwischen Erben) erbt „jeder alles“.

[15] Die wenigen „Wolken-Diagramme“ in [1], die die *Abhängigkeiten* zwischen Teil*paketen* der Software veranschaulichen, sind mit UML-Paketdiagrammen vergleichbar.

Interface Klasse, die bestimmte öffentliche Daten und/oder Funktionen spezifiziert, die von anderen ohne Kenntnis der zugrundeliegenden Details genutzt werden können. Die Implementierung der Funktionen kann den von der Interface-Klasse abgeleiteten Klassen überlassen werden *(abstrakte Klasse)*.

Implementierte Klasse Klasse mit einer vollständigen Implementierung. Nur von dieser können sinnvolle Instanzen gebildet werden.

Konstruktor Funktion, die zum Zwecke bzw. zum Zeitpunkt der Erzeugung eines Klassen-Objektes ausgeführt wird: Reserviere Speicher, belege Variablen mit Anfangswerten, setze Zeiger auf Funktionen...

Destruktor Funktion, die bei der Zerstörung eines Klassen-Objektes aufgerufen wird: Gib reservierten Speicher frei, ...

5.3.2 UML-Klassendiagramme

Die Unified Modeling Language führt eine Reihe von Begriffen ein, die die *statischen* Beziehungen von Klassen oder Instanzen untereinander beschreiben. Diese Beziehungen werden im *UML-Klassendiagramm* durch verschiedenartige Pfeile und Linien dargestellt. Die Skizze Abb. 6 demonstriert die Syntax wesentlicher Elemente der UML-Klassendiagramme.

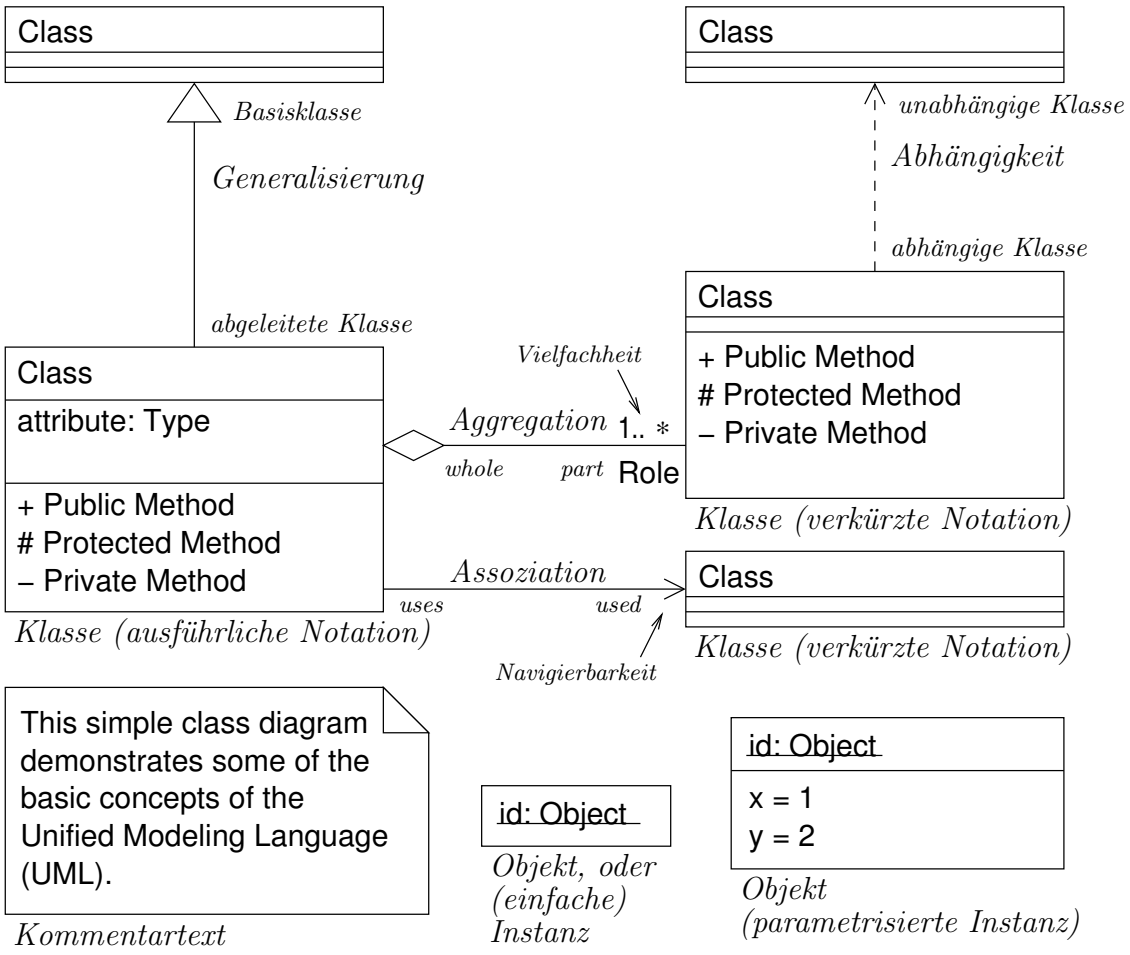


Abbildung 6: Syntax der UML-Klassendiagramme (Auswahl)

Generalisierung Mehreren Klassen gemeinsame Eigenschaften werden in eine allgemeinere Klasse gesetzt. Generalisierung wird i.d.R. durch Vererbung implementiert.

Abhängigkeit Veränderungen in (der Spezifikation) einer Klasse ziehen Veränderungen in einer anderen Klasse nach sich.

Assoziation Eine Klasse benutzt eine andere zur Erfüllung bestimmter Aufgaben.

Aggregation Eine Klasse ist funktioneller Bestandteil einer anderen.

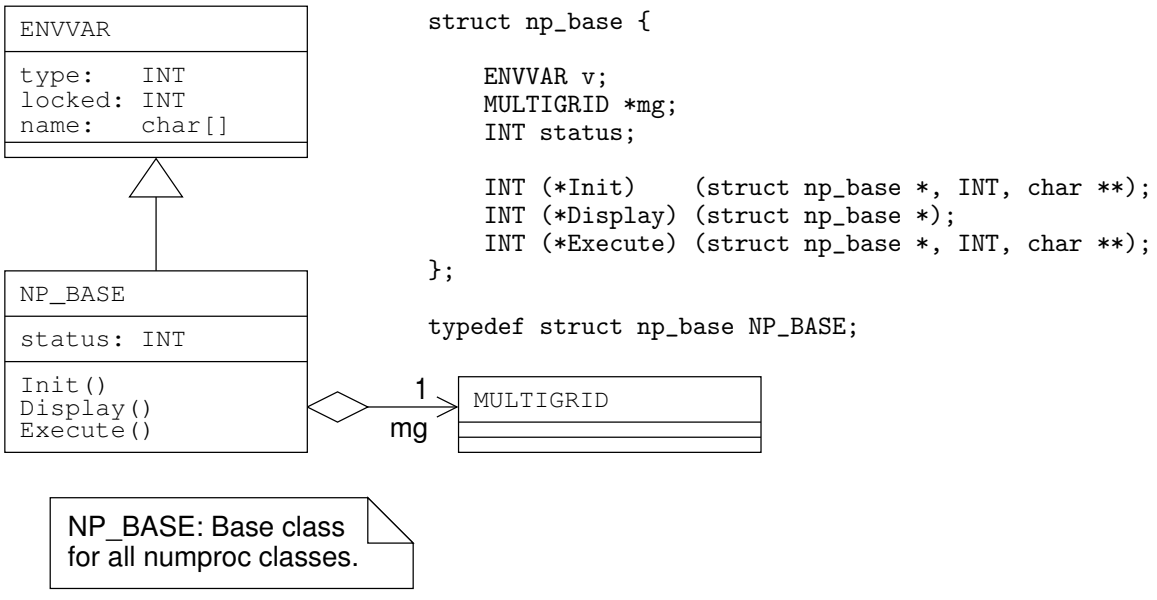
Komposition Eine Klasse ist integraler Bestandteil einer anderen.

Die verschiedenen Typen von Relationen, insbesondere Aggregation und Komposition, sind nicht mit mathematischer Strenge voneinander abzugrenzen: Verschiedene Autoren benutzen geringfügig voneinander abweichende Definitionen. Die genaue Semantik der Relationen hängt von der Sichtweise (konzeptionell, spezifizierend, implementierend) ab; diese Sichtweisen sind aber nicht Teil der UML-Spezifikation (siehe [7]). Oberflächlich übereinstimmende Konzepte werden in verschiedenen Programmiersprachen unterschiedlich umgesetzt.

5.3.3 Realisierung in `ug`

Die im C-Quelltext der numerischen Prozeduren in `ug` weitgehend eingehaltenen Formalismen lassen sich nun als Umsetzung der oben eingeführten Konzepte deuten. Da die Bezeichnungen *Klasse* und *Objekt* in `ug` nicht zufällig gebraucht werden, liegt es nahe, daß auch tatsächlich die obigen Konzepte gemeint sind (die `ug`-Dokumentation erwähnt diese nicht!)

Klassen werden als `struct`-Variablen implementiert. Die **Attribute** sind gewöhnliche Datenfelder in der Struktur, es gibt keine Zugriffsbeschränkungen. Die **Methoden** werden durch Funktionen realisiert, denen als erstes Argument ein Zeiger auf die betroffene Instanz zu übergeben ist. In der die Klasse repräsentierenden `struct`-Variablen wird je ein Zeiger auf jede zur Klasse gehörende Funktion abgelegt.




```
struct np_base {

    ENVVAR v;
    MULTIGRID *mg;
    INT status;

    INT (*Init)    (struct np_base *, INT, char **);
    INT (*Display) (struct np_base *);
    INT (*Execute) (struct np_base *, INT, char **);
};

typedef struct np_base NP_BASE;
```

Abbildung 7: Objektorientierung in `ug` am Beispiel von `NP_BASE`

Der (im Gegensatz etwa zu C++ in keinem Falle automatisch generierte) **Konstruktor** muß zumindest den für die Struktur benötigten Speicher reservieren und die der Klasse eigenen Methoden-Zeiger setzen. Für die ug-Numproc-Klassen wird der Speicherblock vom Umfang `sizeof(struct <theClass>)` vom Environment-Heap angefordert; damit sich die Instanz in die Environment-Struktur einordnet (und dort mit den üblichen Mechanismen aufgefunden werden kann), wird die Klasse von `ENVVAR` abgeleitet.

Die Konstruktoren sind (ug-spezifisch!) als Strukturen aus einer `ENVVAR` und dem Zeiger auf die Konstruktor-Funktion ebenfalls im Environment abgelegt. Diese Strukturen, die durch den Aufruf von `CreateClass()` erzeugt werden, können als Creator-Klassen interpretiert werden (sie exportieren nichts als eine Funktion, die einen Konstruktor aufruft).

Vererbung wird dadurch realisiert, daß die Basisklasse als erstes Feld in der Struktur der abgeleiteten Klasse enthalten ist. Damit kann ein Zeiger auf die abgeleitete Klasse als Zeiger auf die Basisklasse interpretiert werden (und umgekehrt, falls sicher ist, daß eine Instanz der abgeleiteten Klasse vorliegt). Mehrfachvererbung wird nicht unterstützt.

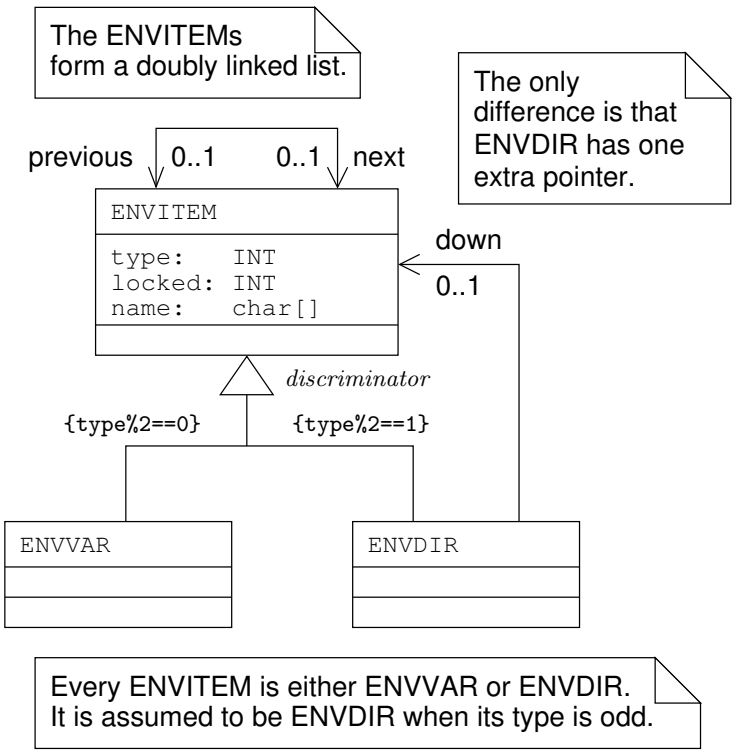
> Es gibt ein konkurrierendes, (in ug) vermutlich historisch älteres Konzept der Generalisierung: Mehrere `struct`-Typen werden zu einer `union` zusammengefaßt. So ist `ENVITEM` eine `union` aus `ENVVAR` und `ENVDIR` (siehe Abb. 8), die `union GEOM_OBJECT` ist entweder `NODE`, `EDGE` oder `ELEMENT`. Von der `union` können ohne Kenntnis des tatsächlich vorliegenden Datentyps genau die Felder benutzt werden, die in der Schnittmenge der einzelnen `struct`-Definitionen liegen. Auf diese Weise wird ein verallgemeinerter Datentyp (entsprechend einer Basisklasse) aus den spezielleren Typen definiert.
>
> Da die Definition der Basisklasse die Definitionen aller abgeleiteten Klassen einschließt, erzeugt diese Variante der Generalisierung stärkere Abhängigkeiten zwischen den Quelltexten als die oben beschriebene Vererbung.

Als **Aggregation** enthalten die Klassen-Strukturen jeweils einen Zeiger auf eine andere Klasse, der belegt (ungleich `NULL`) sein muß, damit die abhängige Klasse als initialisiert gilt. Für die `NUMPROC`-Klassen ist

- `status=ACTIVE`, falls die Zeiger auf alle untergeordneten Prozeduren gesetzt sind und der `status` dieser Prozeduren mindestens `ACTIVE` ist,
- `status=EXECUTABLE`, falls darüber hinaus auch alle Datendeskriptor-Zeiger gesetzt sind,
- `status=NOT_ACTIVE` sonst.

Beispiel: `NP_BASE`. Eine Gegenüberstellung von UML-Diagramm und Klassendefinition in C für die Klasse `NP_BASE` (von der alle anderen Numproc-Klassen abgeleitet sind) zeigt die Abb. 7. Im UML-Diagramm wurden im Interesse der Übersichtlichkeit die Aufrufparameter und Rückgabetypen der Operationen (Funktionen) weggelassen.

Eine wesentliche Eigenschaft von `NP_BASE` ist im Klassendiagramm nicht sichtbar (und kann als *dynamische* Eigenschaft in einem solchen auch nicht dargestellt werden): Die Operationen `Init()`, `Display()` und `Execute()` entsprechen den Shell-Kommandos `npinit`, `npdisplay` und `npexecute`, d.h. erstere werden durch letztere aufgerufen. Von der Shell aus sind auch *nur* diese unmittelbar erreichbar: der Kommandointerpreter „kennt“ nur `NP_BASE`.




```
typedef struct {
    INT type;
    INT locked;
    union envitem *next;
    union envitem *previous;
    char name[NAMESIZE];

} ENVVAR;

typedef struct {
    INT type;
    INT locked;
    union envitem *next;
    union envitem *previous;
    char name[NAMESIZE];
    union envitem *down;
} ENVDIR;

union envitem {
    ENVVAR v;
    ENVDIR d;
};

typedef union envitem ENVITEM;

#define IS_ENVDIR(p) (p->v.type%2==1)
```

Abbildung 8: Generalisierung am Beispiel der `union envitem`

Beispiel: Newton-Löser. Als umfangreicheres Beispiel für ein Klassendiagramm ist in Abb. 9 ein Ausschnitt aus der Klassenhierarchie des Newton-Lösers wiedergegeben.

Der Newton-Löser `NP_NEWTON` ist eine Implementierung des abstrakten nichtlinearen Lösers `NP_NL_SOLVER`. Er benötigt in seiner Eigenschaft als nichtlinearer Löser eine nichtlineare Assemble-Prozedur und darüber hinaus (Newton-spezifisch) einen linearen Löser.

Diese Prozeduren müssen jeweils ein bestimmtes abstraktes Interface besitzen; dem Newton-Löser können zur Laufzeit (d.h. vom Kommandointerpreter) beliebige Prozeduren mit dem passenden Interface zugewiesen werden. Die als `NP_NL_ASSEMBLE` und `NP_LINEAR_SOLVER` definierten Interfaces werden auf der Kommandozeile bzw. im Environment durch die Namen `nlass` bzw. `linear_solver` identifiziert.

Der Zugriff auf das Mehrgitter und die damit verknüpfte Problembeschreibung erfolgt über die aus `NP_BASE` ererbten Datenfelder. Die darin enthaltene Struktur `ENVVAR` ermöglicht auch das Auffinden der Numproc-Klasse anhand ihres Namens im Environment.

Das Klassendiagramm zeigt (nur) die von den abstrakten Interface-Klassen exportierten Funktionen. Deren konkrete Realisierung liegt jeweils in einer der abgeleiteten Klassen.

Daten-Deskriptoren. Die einer Klasse zugeordneten Vektor- und Matrixdeskriptoren sind im Diagramm Abb. 9 (und den folgenden Diagrammen) als Attribute oberhalb punktierter Linien dargestellt. Die Deskriptoren bestehen aber unabhängig von den Numproc-Klassen und werden ggf. von mehreren Prozeduren gemeinsam benutzt. Daher handelt es sich nicht um Attribute, sondern um eigenständige Objekte, mit denen eine Assoziation besteht. Zur Hervorhebung des Zusammenwirkens der verschiedenen numerischen Algorithmen und ihrer Parameter wurde diese etwas ungenaue, konzeptionelle Darstellung gewählt.

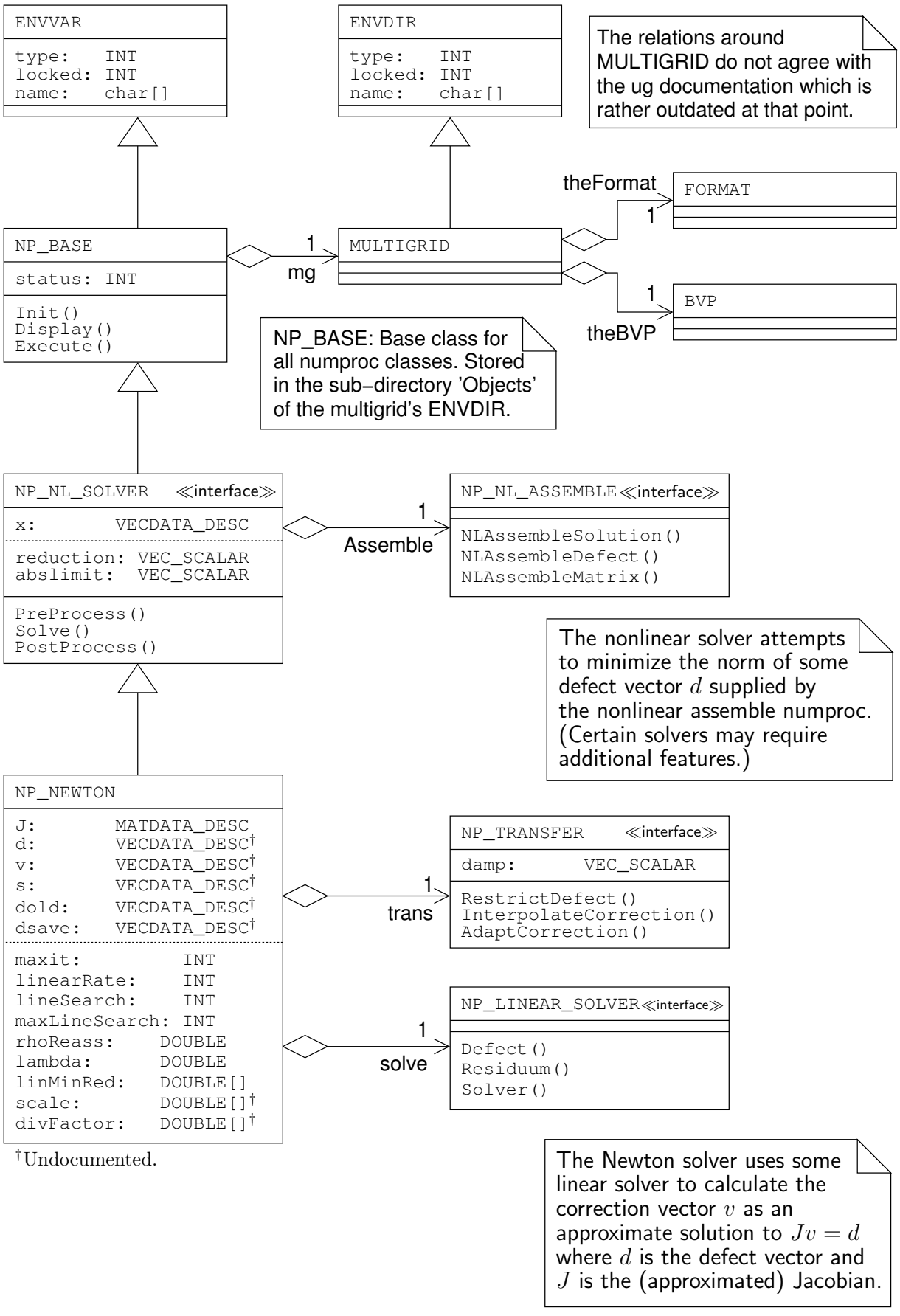


Abbildung 9: Klassenhierarchie des Newton-Lösers

Die Verwaltung (Übergabe) der Datendeskriptoren in `ug` ist kompliziert und nicht einheitlich. Die Datendeskriptoren können auf drei verschiedenen Wegen zugewiesen werden:

1. Die Vektor- bzw. Matrixsymbole werden bei der Initialisierung durch den Anwender (auf der Kommandozeile) mittels `npinit` benannt.
2. Die von einer numerischen Prozedur benötigten Deskriptoren werden automatisch beim *User Data Manager* angefordert, falls sie nicht explizit benannt wurden.
3. Die Deskriptoren werden beim *qualifizierten*[16] Funktionsaufruf von einer übergeordneten Klasse übergeben und mit dieser geteilt.

Ungeklärt bzw. in `ug` nicht einheitlich ist, welcher der drei Wege den Vorrang hat: Zum einen besitzen die Klassen eigene Deskriptoren (ggf. zugewiesen in `npinit`), zum anderen können in *jedem* Funktionsaufruf äquivalente Deskriptoren übergeben werden.

Ein ähnliches Problem taucht bei einigen Parametern auf: Eine Klasse besitzt einen (via `npinit` einstellbaren) Parameter. Dieser wird beim Aufruf über `npexecute` berücksichtigt, beim *qualifizierten* Aufruf durch andere Prozeduren wird an seiner Stelle jedoch ein von dieser übergebener Parameter benutzt, d.h. der eingestellte Parameter wird überschrieben.

Die Dokumentation läßt Hinweise auf dieses Verhalten vermissen.

5.4 Beschreibung ausgewählter numerischer Prozeduren

5.4.1 Linearer Mehrgitterzyklus

Die numerische Prozedur `iter.lmgc` (C-Typ `NP_LMGC`) implementiert einen allgemeinen linearen Mehrgitterzyklus (bei Bastian [2]: „multiplikativer Mehrgitterzyklus“). Die Zusammenhänge zu anderen Numproc-Klassen sind im Klassendiagramm Abb. 10 verdeutlicht.

Der Algorithmus baut auf den vier Teilprozeduren Basislöser, Vor- und Nachglätter und Gittertransfer auf, die den in 4.1 eingeführten Operatoren $\mathcal{L}$, $\mathcal{S}$ und $\mathcal{R}/\mathcal{P}$ entsprechen.

Während Vor- und Nachglätter, die in 4.1 der Einfachheit halber durch denselben Operator $\mathcal{S}$ dargestellt wurden, durch zwei verschiedene Algorithmen realisiert werden können und daher als zwei unabhängige Aggregationen (zwei Zeiger) implementiert sind, werden die beiden Operatoren Restriktion $\mathcal{R}$ und Prolongation $\mathcal{P}$ zu *einer* Klasse `Transfer` zusammengefaßt. Dies ist sinnvoll, weil dadurch sichergestellt werden kann, daß die beiden Operatoren stets zueinander adjungiert sind bzw. in geeignet zu definierender Weise zueinander passen.

Unklar ist, weshalb Vor- und Nachglätter vom Typ `NP_ITER` und nicht vom abgeleiteten Typ `NP_SMOOTHER` sind. In der vorliegenden Implementierung kann daher `NP_LMGC` sich selbst als Vor- bzw. Nachglätter benutzen, was sicher wenig sinnvoll ist.

> Es gibt als Glätter eingesetzte Prozeduren, die nicht von `NP_SMOOTHER` abgeleitet sind: `NP_TS` „transforming smoother“, `NP_SBGS` „equation block Gauss Seidel smoother“.

[16] Bei allen Aufrufen der Numproc-Klassen untereinander werden Datendeskriptor-Zeiger übergeben. Die Funktionen mit individuell verschiedenen Argumentlisten (daher „qualifiziert“) sind auf den von `NP_BASE` abgeleiteten Klassen definiert. Der Kommandointerpreter kennt nur `NP_BASE` und die darauf definierten Funktionen `Init()`, `Display()` und `Execute()` und setzt daher explizit zugewiesene Deskriptoren voraus.

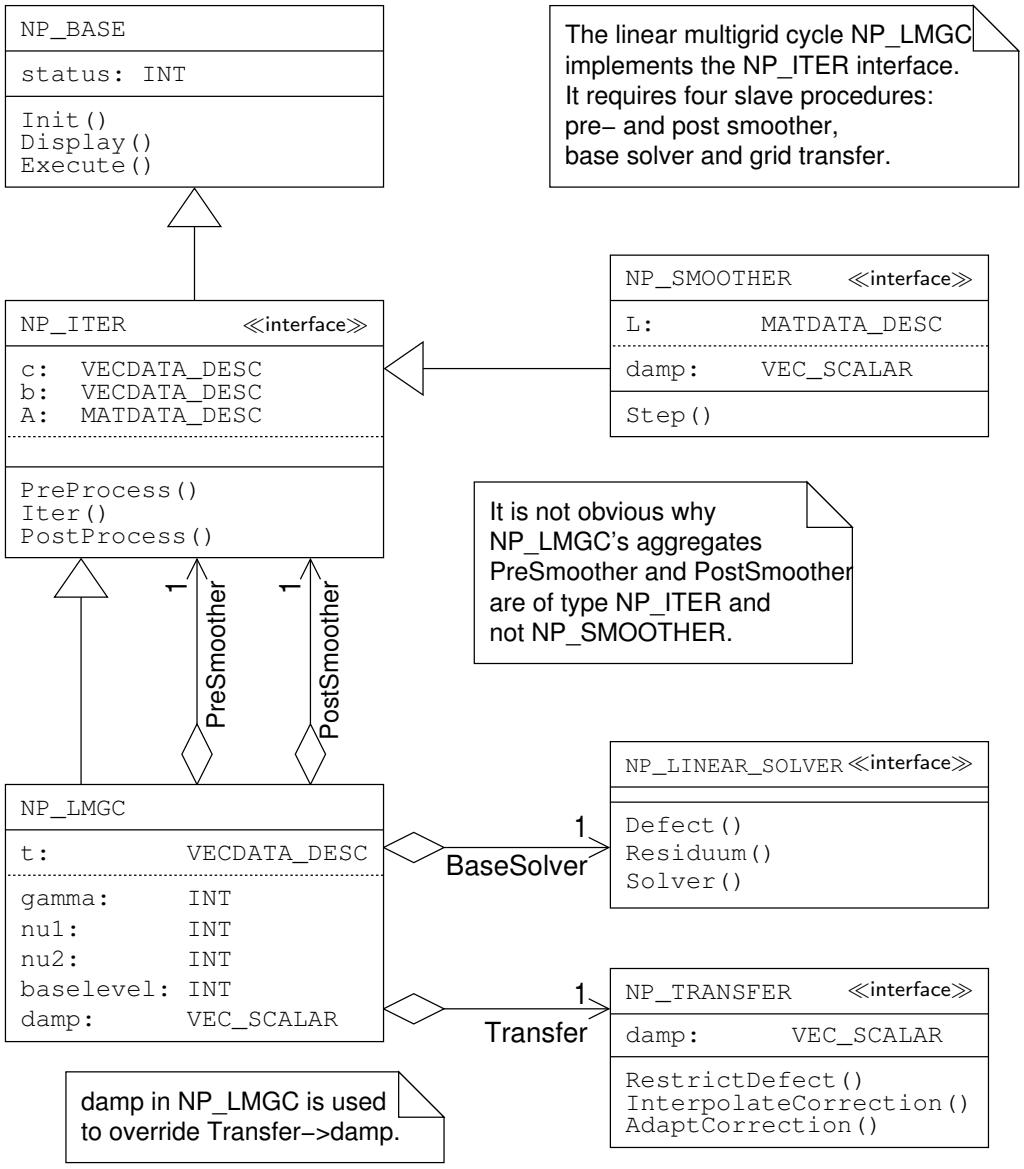


Abbildung 10: Klassenhierarchie des Mehrgitterzyklus

`NP_LMGC` wird durch die folgenden Werte parametrisiert:

Shell `npinit`	C/UML	Bedeutung
`$b`	`baselevel`	l_0: Gitterlevel, auf dem der Basislöser aufgerufen wird
`$g`	`gamma`	γ: Anzahl von Iterationen pro Gitterlevel `gamma=1` liefert „V-Zyklus“, `gamma=2` „W-Zyklus“
`$n1, $n2`	`nu1, nu2`	ν_1, ν_2: Anzahl von Vor- und Nachglättungsschritten
`$damp`†	`damp`	Dämpfungsfaktor, übergeben an `Transfer`

Der Parameter `$damp`† (†=*undokumentiert*) wird nicht in `NP_LMGC` selbst benutzt, sondern beim Aufruf von `Transfer->InterpolateCorrection()` an `NP_TRANSFER` übergeben.

> `NP_TRANSFER` besitzt einen eigenen Dämpfungsparameter `damp`. Dieser ist jedoch nur beim direkten Aufruf via `npexecute` wirksam, beim Aufruf durch andere Numproc-Klassen wird der Wert überschrieben.

5.4.2 Fehlerindikator

Die ug-Standardimplementierung installiert die Numproc-Klasse `error.indicator` als allgemein verwendbaren Fehlerschätzer (Fehlerindikator). Weitere Fehlerschätzer finden sich in den problemspezifischen Paketen.

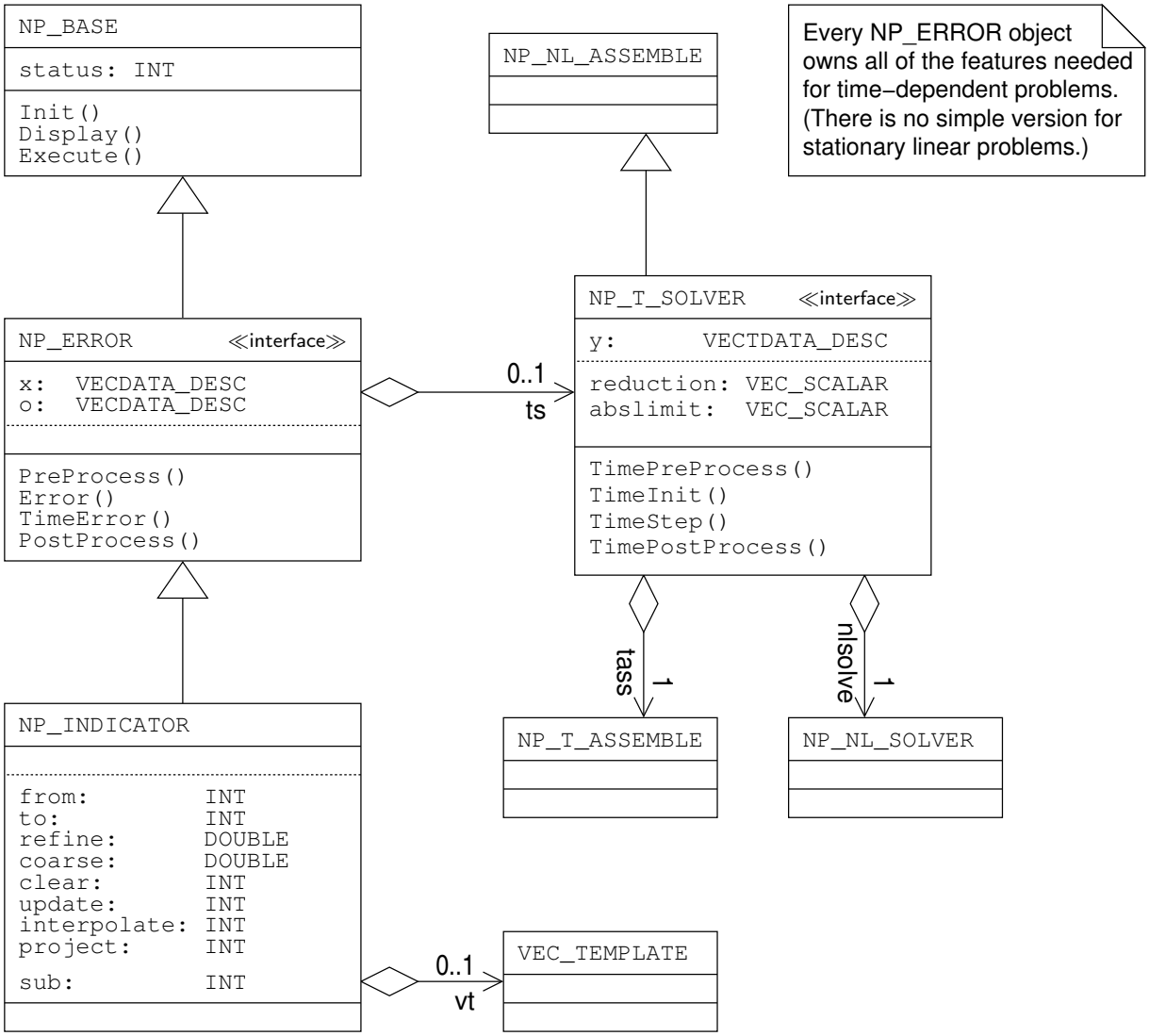


Abbildung 11: Klassenhierarchie des Fehlerindikators `error.indicator`

Der von `error.indicator` ausgeführte Algorithmus besteht aus drei Teilen:

1. Aufruf eines Fehlerschätzers für jedes einzelne Element auf dem Oberflächengitter
2. Erzeugung von Verfeinerungs-/Vergröberungsmarkierungen für die Elemente durch Vergleich der einzelnen vom Fehlerschätzer gelieferten Werte untereinander
3. ggf. Ausführung der Gitterverfeinerung bzw. -vergröberung für markierte Elemente durch Anwendung der dafür definierten Regeln (Aufruf von `AdaptMultigrid()`)

Leider ist der erste Teil, der eigentliche Fehlerschätzer, *nicht* durch die sonst in ug übliche Zuweisung einer numerischen Prozedur frei wählbar. Hier wurde die Möglichkeit einer Modularisierung der Algorithmen (Trennung von Fehlerschätzer und Indikator) verschenkt.

Möglicherweise ist mit der Trennung in NP_ERROR und NP_INDICATOR ursprünglich genau diese Modularisierung beabsichtigt gewesen. Sollte dies der Fall sein, so deutet dies darauf hin, daß die in ug angelegten objektorientierten Strukturen bei der Erstellung von NP_INDICATOR nicht hinreichend verstanden worden sind. Ein weiteres Indiz dafür sind die unten aufgezählten Unregelmäßigkeiten (Verstöße gegen nirgends niedergeschriebene Konventionen).

Interface. Die abstrakte Basisklasse NP_ERROR exportiert die Eigenschaften, die für beliebige Fehlerschätzer sinnvoll sind: sie beinhaltet ein Vektorsymbol für den Lösungsvektor sowie neben den PreProcess()/PostProcess()-Funktionen den Aufruf eines nicht näher spezifizierten Fehlerschätzers (-indikators).

Aufrufparameter für npinit:

Shell	C/UML	Bedeutung
$x <*vec*>	x	Vektorsymbol: Lösung
$o <*vec*>†	o	Vektorsymbol: Alte Lösung *(nur für* ***TimeError()*** *benötigt)*

Aufrufparameter für npexecute:

Shell	C/UML	Bedeutung
$i	PreProcess()	
$e	Error()	Aufruf des zeitunabhängigen Fehlerschätzers
$t <*time*>	TimeError()	Aufruf des zeitabhängigen Fehlerschätzers zum Zeitpunkt time
$s <*step*>	—	Zeitschrittweite (in Verbindung mit $t)
$p	PostProcess()	

Implementierung. Die Klasse NP_INDICATOR realisiert einen einfachen Fehlerindikator.

Standardmäßig wird ein *Gradienten-Fehlerschätzer* verwendet. Dieser berechnet die Gradienten der Lösung im Schwerpunkt für das gegebene Element und dessen Vater-Element und summiert über die euklidischen Normen der Gradienten-Differenzen in allen Vektorkomponenten.

Alternativ steht (undokumentiert) ein sogenannter *Minmax-Fehlerschätzer* zur Verfügung. Dieser gibt als Fehlerschätzwert für das Element die Differenz zwischen maximalem und minimalem Knotenwert in einer ausgewählten Vektorkomponente zurück.

Aufrufparameter für npinit:

Shell	C/UML	Bedeutung
$x <*vec*>, $o <*vec*>†‡	x, o	Vektorsymbole (aus NP_ERROR)
$from <*lev*>†	from	Minimaler Gitter-Level
$to <*lev*>†	to	Maximaler Gitter-Level
$refine <*val*>	refine	Fehlerschranke für Verfeinerung
$coarse <*val*>	coarse	Fehlerschranke für Vergröberung
$c†‡	clear	Lösche bestehende Markierungen
$p†‡	project	Projiziere auf niedere Gitter-Level
$r†‡	update	Verfeinere/vergröbere markierte Elemente
$i†‡	interpolate	Interpoliere Werte neuer Vektoren
$minmax <*vt*> <*svt*>†	vt, sub	Anwendung des Minmax-Indikators auf der durch svt bezeichneten Vektorkomponente

Aufrufparameter für `npexecute`:

Shell	C/UML	Bedeutung
`$c`†	`clear`	Lösche bestehende Markierungen
`$p`	`project`	Projiziere auf niedere Gitter-Level
`$r`	`update`	Verfeinere/vergröbere markierte Elemente
`$i`	`interpolate`	Interpoliere Werte neuer Vektoren

Die für die Basisklasse `NP_ERROR` verfügbaren `npexecute`-Parameter sind in `NP_INDICATOR` wirkungslos, da `NPErrorExecute()` nicht aufgerufen wird. (Hier wird der Mechanismus der Vererbung unterlaufen.)

`NP_INDICATOR` exportiert als `TimeError()` (Fehlerindikator für instationäre Probleme) eine Funktion `TimeIndicator()`, die lediglich die stationäre Version `Indicator()` aufruft. Dieser kann allerdings durch `npexecute` nicht aufgerufen werden, da das Argument `$t` nicht verfügbar ist.

Ausführungs-Optionen. Die über die eigentliche Fehlerschätzung hinausgehende Funktionalität von `error.indicator` wird durch die vier Kommandozeilen-Optionen (Schalter) `$p`, `$r`, `$i` und `$c` gesteuert.

> `$p`, `$r` und `$i` sind in der Dokumentation als `npexecute`-Argumente aufgeführt. Diese Argumente (und das undokumentierte `$c`) werden auch von `npinit` in derselben Bedeutung ausgewertet. Dort sind sie allerdings wirkungslos, da die korrespondierenden Variablen beim Aufruf von `npexecute` überschrieben werden.
>
> Die Belegung von `$i` und `$p` als `npexecute`-Argumente *in der obigen Bedeutung* ist verwirrend, da durch `npexecute $i` bzw. `$p` üblicherweise die Funktionen `PreProcess()` bzw. `PostProcess()` aufgerufen werden. Diese sind aber von `NP_INDICATOR` nicht implementiert.

`$c clear` (Undokumentiert) Vor dem Aufruf des Fehlerschätzers werden ggf. bestehende Verfeinerungs-/Vergröberungsmarkierungen entfernt (dies entspricht dem vorherigen Aufruf von `mark $c`).

`$p project` Vor dem Aufruf des Fehlerschätzers werden die Gitterfunktionen des Oberflächengitters mittels `StandardProject()` auf die niederen Gitterlevel projiziert.

`$r update` Anhand der durch den Fehlerschätzer erzeugten Markierungen wird das Mehrgitter durch den Aufruf von `AdaptMultiGrid()` lokal verfeinert bzw. vergröbert.

`$i interpolate` Nach ggf. erfolgter Gitterverfeinerung (-vergröberung) werden neu hinzugekommene Vektoren durch den Aufruf von `StandardInterpolateNewVectors()` belegt.

5.4.3 Newton-Löser

Der bereits in 5.3.3 kurz vorgestellte Newton-Löser (siehe auch Abb. 9) soll hier noch etwas eingehender behandelt werden. Dabei soll vor allem die Bedeutung der verwirrenden Vielzahl einstellbarer Parameter beleuchtet werden.

Beim Aufruf via `npexecute` sind lediglich die drei auf `NP_NL_SOLVER` definierten Schalter `$i` (PreProcess), `$s` (Solve) und `$p` (PostProcess) wirksam. Die umfangreichen Konfigurationsmöglichkeiten für `NP_NEWTON` werden von `npinit` aus erreicht.

Die Bedeutung der einzelnen Parameter wird im Abschnitt 4.2 genauer erläutert; dort werden weitgehend mit den `ug`-Bezeichnern übereinstimmende Symbole verwendet.

Die folgenden Vektor- und Matrixsymbole können `nl_solver.newton` von der Kommandozeile aus zugewiesen werden:

Shell	C/UML	Bedeutung
`$x` <*vec*>	`x`	Lösungsvektor (aus der Basisklasse `NP_NL_SOLVER`)
`$J` <*mat*>	`J`	Jacobi-Matrix
`$d` <*vec*>†	`d`	Defektvektor
`$v` <*vec*>†	`v`	Korrekturvektor
`$s` <*vec*>†	`s`	Gespeicherter Lösungsvektor
`$dold` <*vec*>†	`dold`	Alter Defektvektor
`$dsave` <*vec*>†	`dsave`	Gespeicherter Defektvektor

Die undokumentierten Vektorsymbole `dold` und `dsave` werden nur für die Variante 3 der Liniensuche benötigt.

Die drei Sub-Prozeduren `Assemble` (nichtlineare Assemble-Prozedur), `trans` (Gittertransfer) und `solve` (linearer Löser) werden als `npinit`-Argumente `$A`, `$T` und `$S` zugewiesen.

Die eigentliche Newton-Iteration wird durch die folgenden Werte parametrisiert:

Shell	C/UML	Bedeutung
`$maxit` <*n*>	`maxit`	Maximalanzahl Newton-Iterationsschritte
`$rhoreass` <*val*>	`rhoReass`	Schwelle für Neuberechnung der Jacobi-Matrix
`$divfac` <*list*>†	`divFactor`	Komponentenweiser Faktor für Abbruchkriterium (Divergenz des Newton-Algorithmus)

Die geforderte Genauigkeit des linearen Lösers `solve` wird abhängig von den folgenden Parametern gesteuert. Die Annahme zur Konvergenzrate dient der Steuerung der sukzessiven Reduktion der linearen Fehlerschranke abhängig von der nichtlinearen Reduktion ρ.

Shell	C/UML	Bedeutung
`$linminred` <*list*>	`linMinRed`	Linearer Reduktionsfaktor je Komponente
`$linrate` <*0*|*1*|*2*>	`linearRate`	0: quadratische, 1: lineare Konvergenzrate; 2: keine Annahme.

Die folgenden Parameter beeinflussen den Dämpfungsfaktor λ bzw. dessen Steuerung durch die Liniensuche:

Shell	C/UML	Bedeutung
`$lambda` <*val*>	`lambda`	Dämpfungsfaktor; Startwert in der Liniensuche
`$line` <*0*|*1*|*2*†|*3*†>	`lineSearch`	Modus der Liniensuche
`$lsteps` <*n*>	`maxLineSearch`	Maximale Zahl von Schritten in der Liniensuche

Der undokumentierte Parameter `$scale` <*list*> (C/UML: `scale`) dient der Skalierung der einzelnen Defekt-Komponenten.

6 Numerische Rechnungen

6.1 Ermittlung der Koeffizienten

In der Modellierung wurden eine Reihe von Parametern als Konstanten angenommen, die sich bei genauerer Betrachtung als Funktionen der Temperatur bzw. der strömungsmechanischen Parameter ergeben und somit ortsabhängig sind. Es wird zunächst davon ausgegangen, daß anstelle der ortsabhängigen Parameter mit deren räumlichen Mittelwerten gerechnet wird. Diese räumlichen Mittelwerte werden nun vor der eigentlichen numerischen Simulation anhand der gegebenen Betriebsparameter geschätzt.

Für die Details der physikalischen Grundlagen soll, soweit diese nicht im Abschnitt 2.1 erklärt worden sind, auf [4, 15] verwiesen werden.

6.1.1 Gegebene Größen

Betriebsparameter. Die folgenden Größen werden an der realen Anlage als Betriebsparameter eingestellt bzw. sind einer Messung unmittelbar zugänglich:

- die Abmessungen des Apparates,
- Gesamtmasse, Durchmesser, Dichte und spezif. Wärmekapazität des Bettmaterials,
- Temperatur, Feuchte und Massenstrom der Trägerluft,
- Temperatur und Massenstrom der eingedüsten Flüssigkeit,
- Position und Öffnungswinkel der Düse(n).

Bei der Nachrechnung von Versuchen sind nun die übrigen Parameter aus diesen gegebenen Daten zu berechnen.

Wirbelschichthöhe. Die Höhe der sich im Reaktor ausbildenden Wirbelschichtzone ist nicht ohne weiteres exakt meßbar, da die obere Grenze unscharf ist (die Annahme einer homogenen Wirbelschicht ist eine wesentlich vereinfachende Annahme). Da diese Höhe zur Beschreibung des (unveränderlichen) Diskretisierungs-Gebietes bekannt sein muß, wird sie vor der numerischen Rechnung aus der geschätzten Porosität ε ermittelt.

6.1.2 Mittelwerte

Für die ortsabhängigen Größen werden räumliche Mittelwerte geschätzt.

Die Eigenschaften der in die Wirbelschicht einströmenden Trägerluft sind vorgegeben, die entsprechenden Werte am Luftaustritt lassen sich aus einer stationären Gesamtbilanz berechnen.

Es ist bekannt, daß in der flüssigkeitsbedüsten Wirbelschicht ein starker Abfall der Lufttemperatur in der Nähe des Anströmbodens herrscht. Daher kann für die mittlere Lufttemperatur der Wert am Luftaustritt eingesetzt werden. Für die übrigen Kenngrößen, z.B. die

Luftfeuchte, sind abhängig von der Flüssigkeitsbelastung zwei Betriebspunkte zu unterscheiden:

ungesättigte Luft: Es kann ein über die Höhe linearer Verlauf der Luftfeuchte angenommen werden, die mittlere Luftfeuchte wird zu $\overline{Y}_L \approx \frac{1}{2}(Y_{L,ein} + Y_{L,aus})$ geschätzt.

nahezu gesättigte Luft: Die Verdunstung findet vor allem in der Nähe des Anströmbodens statt. Die mittlere Feuchte kann durch den Wert am Luftaustritt geschätzt werden: $\overline{Y}_L \approx Y_{L,aus}$.

6.1.3 Gesamtbilanzen

Luftfeuchte. Im stationären Betrieb setzt sich der Massenstrom der ausströmenden Luft aus der einströmenden Trägerluft und dem Dampfstrom aus der Eindüsung zusammen. Da der Massenstrom der trockenen Luft $\dot{m}_{L,0} = \frac{\dot{m}_{L,ein}}{1+Y_{L,ein}}$ beim Durchgang durch die Wirbelschicht unverändert bleibt, ergibt sich die Feuchte der ausströmenden Luft zu

$$Y_{L,aus} = Y_{L,ein} + \frac{\dot{m}_Q}{\dot{m}_{L,0}} \approx Y_{L,ein} + \frac{\dot{m}_Q}{\dot{m}_{L,ein}}.$$

Die mittlere Luftfeuchte wird entsprechend der Ausführungen im Abschnitt 6.1.2 geschätzt.

Lufttemperatur. Die Temperatur der ausströmenden Luft läßt sich durch eine Enthalpiebilanz bestimmen. Die Bilanz

$$\dot{m}_{L,aus} c_{pY,aus} \vartheta_{L,aus} = \dot{m}_{L,ein} c_{pY,ein} \vartheta_{L,ein} + \dot{m}_Q (c_{pF} \vartheta_F - \Delta h_{V0})$$

führt auf

$$\vartheta_{L,aus} = \frac{\dot{m}_{L,ein} c_{pY,ein} \vartheta_{L,ein} + \dot{m}_Q (c_{pF} \vartheta_Q + \Delta h_{V0})}{\dot{m}_{L,aus} c_{pY,aus}} \approx \vartheta_{L,ein} + \frac{\dot{m}_Q}{\dot{m}_{L,ein}} \frac{c_{pF}\vartheta_Q - \Delta h_{V0}}{c_{pL}}$$

Die mittlere Lufttemperatur wird gemäß 6.1.2 gleich $\vartheta_{L,aus}$ gesetzt.

Dichte der Luft. Die (mittlere) Dichte der feuchten Luft berechnet sich zu

$$\overline{\rho}_L = \frac{p}{R_Y(\overline{\vartheta}_L + T_0)},$$

für kleine Werte der Luftfeuchte kann $R_Y = Y R_D + (1 - Y) R_L$ durch R_L ersetzt werden.

6.1.4 Strömungsmechanische Größen

Strömungsgeschwindigkeit. Die Leerrohr-Strömungsgeschwindigkeit der Luft, die sich bei Abwesenheit der Bettpartikel (d.h. im Leerrohr) einstellen würde, ergibt sich mit dem Apparatequerschnitt A_{App} zu

$$w_0 = \frac{\overline{\dot{m}}_L}{\overline{\rho}_L A_{App}}.$$

In der Wirbelschicht steht der Luft ein um den Faktor ε kleineres Volumen zur Verfügung. Die tatsächliche *Strömungsgeschwindigkeit* ergibt sich zu

$$w_L = \frac{w_0}{\varepsilon}.$$

Dimensionslose Kennzahlen. Die *Reynolds-Zahl*

$$Re = \frac{w_0 d_P}{\nu_L(\overline{\vartheta}_L)}$$

wird auf die Leerrohr-Geschwindigkeit bezogen, in ihre Berechnung gehen der Durchmesser der Bettpartikel d_P und die (temperaturabhängige) kinematische Viskosität der Luft ν_L ein.

Die *Archimedes-Zahl*

$$Ar = \frac{g d_P^3 (\rho_P - \overline{\rho}_L)}{\nu_L^2(\overline{\vartheta}_L)\overline{\rho}_L}$$

hängt ebenfalls von den Eigenschaften der Luft und der Bettpartikel ab.

Porosität. Für die Berechnung der Porosität als Funktion der Reynolds- und Archimedes-Zahl werden von verschiedenen Autoren verschiedene empirische Formeln angegeben; die Ergebnisse weichen z.T. erheblich voneinander ab.

Nach [9, Todes, Goroško, Rozenbaum]:

$$\varepsilon = \left(\frac{18Re + 0,36Re^2}{Ar}\right)^{0,21},$$

die Reynolds-Zahlen im Wirbelpunkt bzw. Austragspunkt werden mit

$$Re_{WP} = \frac{Ar}{1400 + 5,22\sqrt{Ar}} \quad \text{bzw.} \quad Re_{AP} = \frac{Ar}{18 + 0,61\sqrt{Ar}}$$

angegeben.

Nach [3, Becher]

$$\varepsilon = \varepsilon_{mf}^{\frac{\ln(Re/Re_{AP})}{\ln(Re_{mf}/Re_{AP})}} \text{ mit } Re_{mf} = 42,9\,\overline{\varepsilon}_{mf}\left(\sqrt{1 + \frac{\varepsilon_{mf}^3}{\overline{\varepsilon}_{mf}^2}\frac{Ar}{3214}} - 1\right) \text{ und } Re_{AP} = \sqrt{\frac{4}{3}Ar}.$$

Tropfenweglänge. Die mittlere Tropfenweglänge S_{Tr}, die zur Berechnung des Quellenfeldes $\dot{\kappa}_Q$ benötigt wird, ergibt sich nach (32) unter Annahme eines Abscheidegrades $\eta_A > 0$ zu

$$S_{Tr} = \frac{2}{3}\frac{\varepsilon}{\overline{\varepsilon}}\frac{d_P}{\eta_A}.$$

6.1.5 Koeffizienten des Wärme- und Stoffaustausches

Dispersionskoeffizienten. Der vertikale Dispersionskoeffizient D_z wird als Funktion der Schlupfgeschwindigkeit $w_s := w_0 - w_{mf}$ berechnet:

$$D_z = \frac{w_s d_{App}}{Pe} \quad \text{mit} \quad Pe = 1.72\sqrt[3]{\frac{w_s^2}{g \cdot d_{App}}}.$$

Die Dispersionskoeffizienten in horizontaler Richtung werden zu $D_x = D_y = 0.1 D_z$ geschätzt.

Wärmeübergangskoeffizienten. Der Koeffizient α_{PL} für den Wärmeübergang Partikel-Luft wird nach [18, Schlünder/Tsotsas] aus den temperaturabhängigen Werten von Wärmeleitwert λ_L und Prandtl-Zahl Pr der Luft sowie der auf die Porosität bezogenen Reynolds-Zahl $Re_\varepsilon = Re/\varepsilon$ berechnet:

$$\alpha_{PL} = \frac{Nu\lambda_L(\vartheta_L)}{d_P} \quad \text{mit} \quad Nu = (1 + 1,5\overline{\varepsilon})\left(2 + \sqrt{Nu_{lam}^2 + Nu_{turb}^2}\right),$$

$$Nu_{lam} = 0,664Pr^{1/3}Re_\varepsilon^{1/2}, \quad Nu_{turb} = 0,037\frac{Re_\varepsilon^{0.8}Pr}{1 + 2,443Re_\varepsilon^{-0.1}(Pr^{2/3} - 1)}.$$

Die übrigen Wärmeübergangskoeffizienten werden geschätzt: $\alpha_{FL} = \alpha_{PL}$, $\alpha_{PF} = f_\alpha\alpha_{PL}$.

Stoffübergangskoeffizient. Der Stoffübergangskoeffizient β hängt von den temperaturabhängigen Werten der kinematischen Viskosität der Luft ν_L und des Diffusionskoeffizienten des Dampfes in Luft D_{DL} ab:

$$\beta = \frac{ShD_{DL}}{d_P} \quad \text{mit} \quad Sh = 2 + 0.72\sqrt{Re_0}\sqrt[3]{\nu_L/D_{DL}}.$$

6.2 Numerische Algorithmen

6.2.1 Zusammenwirken der numerischen Prozeduren

Alle verwendeten numerischen Prozeduren bis auf die problemspezifische nichtlineare Assemble-Prozedur `nlfv_ass` werden der `ug`-Standardinstallation entnommen.

Die Zusammenstellung der für die numerische Rechnung verwendeten Prozeduren und die eingestellten Parameter sind in Abb. 13 wiedergegeben. Zum Vergleich zeigt Abb. 12 einen Ausschnitt aus dem Skript, welches die in Abb. 13 dargestellte Konfiguration realisiert.

Die Grafik Abb. 13 zeigt die numerischen Prozeduren als *Instanzen* in einem UML-Klassendiagramm, wobei die eingestellten Parameter unmittelbar abzulesen sind. Die dargestellten Bezeichner sind die auf dem `ug`-Kommandozeilenniveau verwendeten Namen (in den Diagrammen im Abschnitt 5 wurden die Bezeichner aus dem C-Quelltext verwendet).

Der Newton-Löser ist mit allen seinen Komponenten (Mehrgitter-Löser, Glätter, Gitter-Transfer usw.) durch Aggregationen (Zeiger) verbunden. Dieser zusammenhängende Komplex wird daher durch einen einzigen Aufruf von `npexecute` ausgeführt.

Die übrigen Prozeduren (Fehlerschätzer, Gitterverfeinerung, *Grafikausgabe*) müssen separat vom Kommandointerpreter aus (praktisch: durch ein Skript) aufgerufen werden. Diese Aufrufe werden nacheinander in einer Schleife ausgeführt. Mit jedem Durchlauf wird die Lösung auf einem um eine Stufe verfeinerten Gitter berechnet, wobei die Lösung auf dem jeweils gröberen Gitter als Startwert dient.

Abb. 13 zeigt ebenfalls die einzelnen Prozeduren explizit zugewiesenen Datendeskriptoren und deren Unterteilung in Sub-Deskriptoren (Komponenten).

Die explizite Benennung von Datendeskriptoren (Symbolen) ist notwendig, wenn eine numerische Prozedur unmittelbar vom Kommandointerpreter aufgerufen wird. Sie ist auch dann erforderlich, wenn zwischen Prozeduren, die nicht durch eine Aggregation verbunden sind (und daher separat aufgerufen werden) Daten ausgetauscht werden sollen. Dies ist insbesondere bei der Grafikausgabe der Fall.

```
# Jacobi-Glätter
npcreate jac $c jac;
npinit jac $damp 0.5;
# Iterationsschritt für Basislöser
npcreate base $c lu;
npinit base;
# Basislöser
npcreate basesolver $c ls;
npinit basesolver $red 1e-4 $m 50 $I base $display no;
# Gitter-Transfer
npcreate transfer $c transfer;
npinit transfer $x sol $d red;
# Linearer Mehrgitterzyklus
npcreate lmgc $c lmgc;
npinit lmgc
$S jac jac basesolver # pre- post- base-
$T transfer # transfer routine
$n1 5 $n2 5 # number of smoothing steps
$g 1        # iterations per level
$b @BASELEVEL;
```

Abbildung 12: Skript zur Konfiguration der numerischen Prozeduren (Ausschnitt)

6.2.2 Eingestellte Parameter

Einige der (in Abb. 13 vollständig dargestellten) Parameter sollen kurz erläutert werden.

Mehrgitter-Zyklus. Die Anzahl der Glättungsschritte wurde mit $\nu_1 = \nu_2 = 5$ ungewöhnlich hoch gesetzt, die Anzahl der Iterationen pro Level dagegen bei $\gamma = 1$ belassen. Dies wird damit begründet, daß die Kompatibilität der Matrizen zwischen den Gitter-Leveln durch die ortsabhängigen Koeffizienten verletzt ist.

Mehrgitter-Löser. Hier werden bis zu `$m = 200` Iterationsschritte zugelassen. Diese Anzahl wird praktisch niemals erreicht, auf diese Weise wird einem vorzeitigen (erfolglosen!) Abbruch des Newton-Lösers vorgebeugt. Die Schranke für die lineare Reduktion wird vom Newton-Löser (variabel) vorgegeben.

Newton-Löser. Die Fehlerschranke für das diskrete nichtlineare Residuum wurde im Newton-Löser auf 10^{-10} eingestellt. (Eine relativ hohe Genauigkeit ist wegen der für kleine Fehler quadratischen Konvergenz des Algorithmus unproblematisch zu erreichen.)

Für die Liniensuche wurde die undokumentierte Option 3 eingestellt: diese zeichnet sich praktisch durch die besten Stabilitätseigenschaften aus.

Assemble-Prozedur. Mit dem Parameter `$upwind=1` wird die Verwendung der Upwind-Diskretisierung für die Konvektionsterme eingestellt.

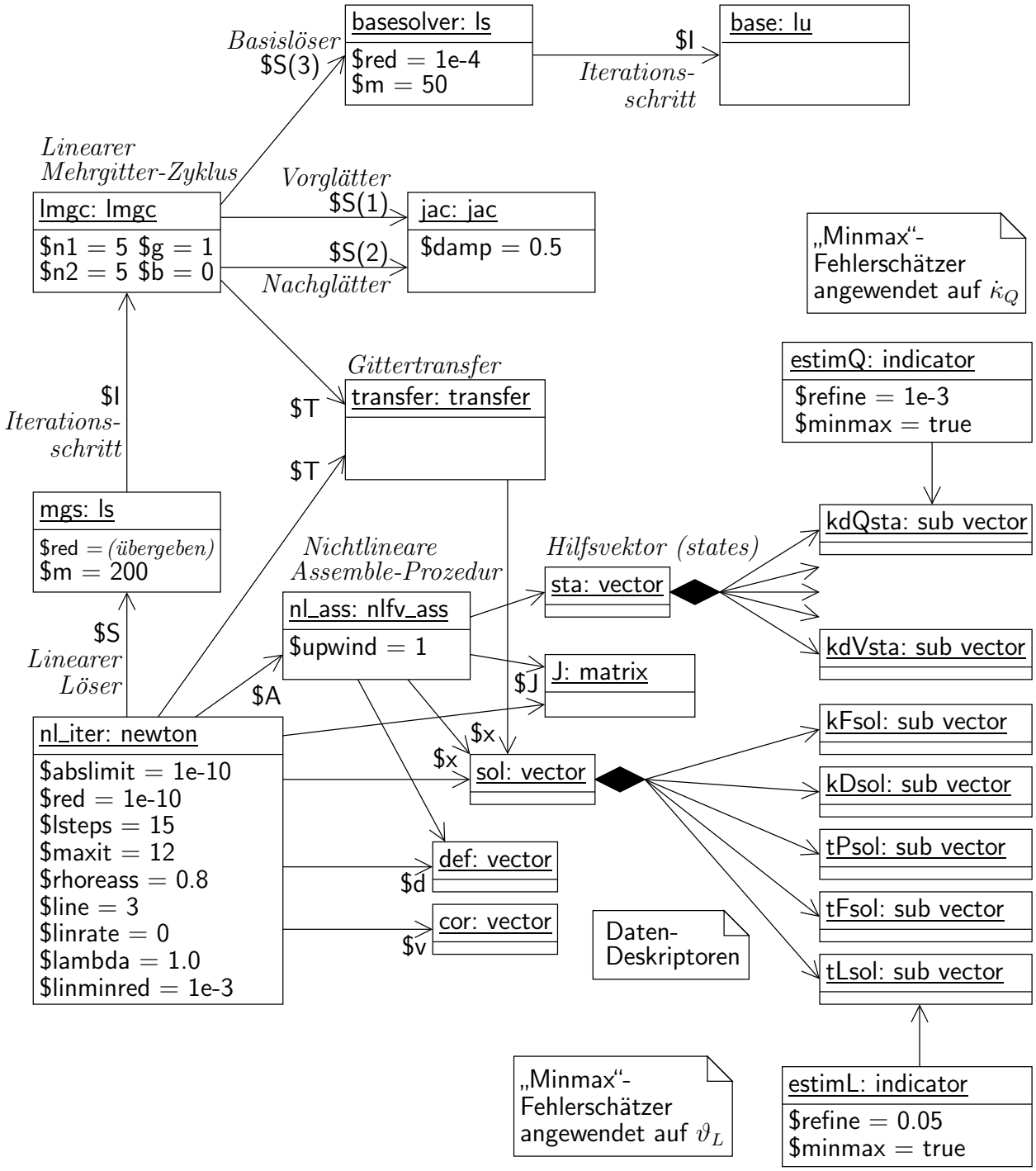


Abbildung 13: Konfiguration der numerischen Algorithmen

6.3 Simulation des vereinfachten Modellproblems

Das im Abschnitt 2.6 beschriebene vereinfachte Modellproblem mit zwei Komponenten wird zur Demonstration auf dem Einheitswürfel bzw. einem Zylinder mit Durchmesser und Höhe Eins numerisch gelöst.

6.3.1 Parameter

Koeffizienten. Für die Koeffizienten des vereinfachten Modells werden Zahlenwerte gewählt, die zumindest in der Größenordnung realer Werte liegen.

Als Betriebsparameter werden gewählt:

Strömungsgeschwindigkeit	w_L=3 $\frac{\mathrm{m}}{\mathrm{s}}$
Eindüsung	$\dot{m}_Q$=10^{-2} $\frac{\mathrm{kg}}{\mathrm{s}}$
Durchmesser der Bettpartikel	d_P=1 mm
Porosität	ε=0, 6

Die temperaturabhängigen Stoffwerte wurden für $\vartheta = 50°\mathrm{C}$ ermittelt. Bei einer angenommenen Flüssigkeitsfilmdicke von $d_F = 100\,\mu\mathrm{m}$ ergeben sich die Parameter der Verdunstung zu $B = 1.857\,\mathrm{s}^{-1}$, $C = 0.4073\,\mathrm{kg/m}^3$, $v^\infty = 49.676 \cdot 10^{-3}\,\mathrm{kg/m}^3$.

Die Dispersionskoeffizienten werden auf $D_z = 1\,\mathrm{m}^2/\mathrm{s}\,, D_x = D_y = 0,1\,\mathrm{m}^2/\mathrm{s}$ gesetzt.

Quellstrom. In den Modellrechnungen 1 und 2 wurde der Quellstrom $f \equiv \dot{\kappa}_Q$ räumlich konstant gesetzt, Varianten 3 und 4 verwenden die Approximation der Eindüsung aus Abschnitt 6.4.

Für die Rechnungen 1, 3 und 4 wurde $f$auf den „realistischen“ Wert von $0.01\,\frac{\mathrm{kg}}{\mathrm{s}}$ gesetzt. In Rechnung 2 wurde ein Quellstrom knapp unter der Sättigungsgrenze $Aw_L v^\infty = 0.149\,\frac{\mathrm{kg}}{\mathrm{s}}$ gewählt, wodurch die im Abschnitt 2.6.3 diskutierten nichtliearen Eigenschaften des Problems deutlich hervortreten.

6.3.2 Numerische Algorithmen

Für die Rechnungen wurde die im Abschnitt 6.2 beschriebene Konfiguration der numerischen Algorithmen (Newton-Löser, Linearer Mehrgitterlöser usw.) benutzt.

Das Startgitter aus acht Elementen wurde jeweils viermal adaptiv verfeinert. Die Gitterverfeinerung wird wie in Abb. 13 durch zwei Minmax-Fehlerindikatoren gesteuert. Durch den mit dem Quellstrom $f \equiv \dot{\kappa}_Q$ verknüpften Indikator wird im Falle der Eindüsung eine hinreichende räumliche Auflösung in der Nähe des Düsenzentrums gesichert. Der in Abb. 13 dargestellte Fehlerschätzer für ϑ_L wird durch einen Fehlerschätzer in der Komponente $v \equiv \kappa_D$ ersetzt.

6.3.3 Rechnungen und Ergebnisse

Rechnung 1: Ungesättigte Luft. Ein Massenstrom von $\dot{m}_Q = 10^{-2}\,\frac{\mathrm{kg}}{\mathrm{s}}$ wird gleichmäßig über den Einheitswürfel ($1\,\mathrm{m}^3$) verteilt. Die Abb. 14 im Anhang B.1 zeigt die errechnete räumliche Verteilung der Größen κ_F und κ_D.

Die Dampfkonzentration am Austritt von ca. $3.2 \cdot 10^{-3}\,\frac{\mathrm{kg}}{\mathrm{m}^3}$ liegt weit unter der Sättigungskonzentration ($49.7 \cdot 10^{-3}\,\frac{\mathrm{kg}}{\mathrm{m}^3}$). Die Verdunstung findet daher im gesamten Gebiet der Wirbelschicht mit annähernd gleichmäßiger Intensität statt.

Rechnung 2: Nahezu gesättigte Luft. Ein Massenstrom von $\dot{m}_Q = 0.14\,\frac{\mathrm{kg}}{\mathrm{s}}$ wird gleichmäßig über den Einheitswürfel verteilt. Die errechnete räumliche Verteilung der Größen κ_F und κ_D ist in Abb. 15 (Anhang B.1) dargestellt.

Die austretende Luft ist annähernd mit Wasserdampf gesättigt, die Verdunstung findet überwiegend in der Nähe des Anströmbodens statt. Daher wird die im oberen Teil eingebrachte Flüssigkeit nach unten transportiert, es bildet sich ein deutlicher Gradient in der Flüssigkeitskonzentration aus.

Da die Verdunstung effektiv nur in einem Teil der Wirbelschichtzone stattfinden kann, steigt die Flüssigkeitskonzentration gegenüber der Variante 1 überproportional an (ca. Faktor 33 bei nur 14-facher Eindüsung), wodurch eine größere innere Oberfläche zur Verfügung steht.

Rechnungen 3, 4: Eindüsung auf Würfel bzw. Zylinder. Der Flüssigkeits-Quellstrom wird wie in 2.5 durch eine Düse mit den folgenden Parametern modelliert:

Position des Düsenzentrums	$\underline{x}_D$=(0.3, 0.5, 0.9)
Strahlrichtung	$\underline{e}_D$=(0, 0, −1) (nach unten)
Halber Öffnungswinkel	θ_D=30°
Eingedüster Massenstrom	$\dot{m}_Q$=$10^{-2}\,\frac{\mathrm{kg}}{\mathrm{s}}$
Tropfenweglänge	S_{Tr}=10 mm

Die Abb. 16 zeigt die auf dem Einheitswürfel errechnete räumliche Verteilung der Größen κ_F und κ_D, in Abb. 17 sind die entsprechenden Ergebnisse für das Zylindergebiet dargestellt.

Da die Luftfeuchte weit von der Sättigungsfeuchte entfernt ist, verdunstet die eingedüste Flüssigkeit schon zum großen Teil, bevor sie mit den Partikeln in die düsenfernen Teile der Wirbelschicht transportiert wird. Dementsprechend bildet sich in der Flüssigkeitskonzentration ein deutliches Profil aus, das Maximum fällt mit dem Düsenzentrum zusammen.

6.4 Nachrechnung von Versuchen

6.4.1 Versuch 15

Der Versuch 15 wurde auf der Versuchsanlage DN 1500 mit Wassereindüsung von oben und Kunststoffgranulat als Bettmaterial durchgeführt. Im stationären Betrieb wurde die räumliche Temperaturverteilung in der Wirbelschicht gemessen.

Betriebsparameter.

Bettmaterial: Kunststoffgranulat	
Gesamtmasse	$m_P = 370\,\mathrm{kg}$
Partikeldurchmesser	$d_P = 3,3\,\mathrm{mm}$
Dichte	$\rho_P = 1377\,\frac{\mathrm{kg}}{\mathrm{m}^3}$
spezifische Wärmekapazität	$c_{pP} = 980\,\frac{\mathrm{J}}{\mathrm{kgK}}$

Geometrische Abmessungen des Apparates	
Durchmesser	$d_{App} = 1,5\,\mathrm{m}$
Querschnittsfläche	$A_{App} = 1,767\,\mathrm{m}^2$
Trägerluft	
Luftmassenstrom	$\dot{m}_{L,ein} = 0,3\,\frac{\mathrm{kg}}{\mathrm{s}}$
Temperatur	$\vartheta_{L,ein} = 80°\mathrm{C}$
Anfangsfeuchte	$Y_{L,ein} = 0,004\,\frac{\mathrm{kg}}{\mathrm{kg}}$
Düse	
Position	$\underline{x}_D = (\frac{1}{4}d_{App}, \frac{1}{2}d_{App}, h_{WS})^T$
Strahlrichtung	$\underline{n}_D = (0,0,-1)^T$ (nach unten)
Halber Öffnungswinkel des Strahles	$\theta_D = 30°$
Eingedüste Flüssigkeit	
Quellstrom (reines Wasser)	$\dot{m}_Q = 4,17 \cdot 10^{-3}\,\frac{\mathrm{kg}}{\mathrm{s}}$
Temperatur	$\vartheta_Q = 20°\mathrm{C}$
Geschätzte Parameter	
Flüssigkeitsfilmdicke	$d_F = 100\mu m$
Abscheidegrad	$\eta_A = 0,1$

Koeffizienten. Die Gesamtbilanz des Luftstromes liefert die folgenden Größen:

	Einströmende Luft	Ausströmende Luft
Massenstrom	$\dot{m}_{L,ein}$=$7,0\,\frac{\mathrm{kg}}{\mathrm{s}}$	$\dot{m}_{L,aus}$=$7,025\,\frac{\mathrm{kg}}{\mathrm{s}}$
Luftfeuchte	$Y_{L,ein}$ =$4,0 \cdot 10^{-3}\,\frac{\mathrm{kg}}{\mathrm{kg}}$	$Y_{L,aus}$ =$7,586 \cdot 10^{-3}\,\frac{\mathrm{kg}}{\mathrm{kg}}$
Temperatur	$\vartheta_{L,ein}$ =$65,0°\mathrm{C}$	$\vartheta_{L,aus}$ =$56,06°\mathrm{C}$

Als mittlere Lufttemperatur wird der Wert $\overline{\vartheta}_L = 57°\mathrm{C}$ angenommen.

Die Austrittsfeuchte der Luft liegt weit unter der Sättigungsfeuchte von ca. $314 \cdot 10^{-3}\,\frac{\mathrm{kg}}{\mathrm{kg}}$ bei 56°C. Daher wird ein über der Höhe annähernd linearer Feuchteverlauf in der Wirbelschicht angenommen, die mittlere Luftfeuchte wird auf $\overline{Y}_L = 5,8 \cdot 10^{-3}\,\frac{\mathrm{kg}}{\mathrm{kg}}$ geschätzt.

Mit diesen Annahmen ergeben sich die folgenden mittleren Werte:

Dichte der Luft	ρ_L =$1,065\,\frac{\mathrm{kg}}{\mathrm{m}^3}$
Kinematische Viskosität	ν_L =$1,858 \cdot 10^{-5}\,\frac{\mathrm{m}^2}{\mathrm{s}}$
Leerrohr-Gasgeschwindigkeit	w_0 =$3,725\,\frac{\mathrm{m}}{\mathrm{s}}$
Leerrohr-Reynoldszahl	Re_0=$661,5$
Archimedes-Zahl	Ar =$1,318 \cdot 10^6$

Für die Porosität ergeben die von verschiedenen Autoren angegebenen empirischen Formeln deutlich voneinander abweichende Ergebnisse:

	nach TGR [9]	nach Becher [3]
Reynolds bei Minimalfluidisation	Re_{mf}=$178,28$	Re_{mf}=$182,82$
Reynolds im Austragspunkt	Re_{AP}=$1834,9$	Re_{AP}=$1325,67$
Porosität	ε =$0,650$	ε =$0,715$
Wirbelschichthöhe	H_{WS} =$0,434\,\mathrm{m}$	H_{WS} =$0,534\,\mathrm{m}$

Gewählt wird der größere Wert $\varepsilon = 0,715$, da die zu diesem Wert gehörende Wirbelschichthöhe besser mit dem beobachteten Wert von $0,8\,\mathrm{m}$ übereinstimmt.

Die weiteren Parameter ergeben sich wie folgt:

Reale Strömungsgeschwindigkeit	$w_L = 5,21\,\frac{\mathrm{m}}{\mathrm{s}}$
Mittlere Tropfenweglänge	$S_{Tr} = 55,2\,\mathrm{mm}$
Dispersionskoeffizient	$D_z = 2,974\,\frac{\mathrm{m}^2}{\mathrm{s}}$
Wärmeübergangskoeffizient	$\alpha_{PL} = 269,2\,\frac{\mathrm{W}}{\mathrm{m}^2\mathrm{K}}$
Stoffübergangskoeffizient	$\beta = 0,1862\,\frac{\mathrm{m}}{\mathrm{s}}$

Die Dispersionskoeffizienten in horizontaler Richtung werden zu $D_x = D_y = 0.1 D_z$ geschätzt.

Die übrigen Wärmeübergangskoeffizienten werden zu $\alpha_{FL} = \alpha_{PL}$, $\alpha_{PF} = 0,1\alpha_{PL}$ angenommen.

Ergebnisse. Abb. 18–Abb. 23 im Anhang B.2 zeigen die errechnete räumliche Verteilung der Größen $\dot{\kappa}_V$, κ_F, κ_D, ϑ_L, ϑ_F und ϑ_P.

6.4.2 Versuch 97

Der Versuch 97 wurde auf der Versuchsanlage DN 400 mit Wassereindüsung aus der Wirbelschicht nach oben durchgeführt. Als Bettmaterial wurden Glaskugeln verwendet. Im stationären Betrieb wurde die räumliche Temperaturverteilung in der Wirbelschicht gemessen.

Betriebsparameter.

Geometrische Abmessungen des Apparates	
Durchmesser	$d_{App} = 0,4\,\mathrm{m}$
Querschnittsfläche	$A_{App} = 0,1257\,\mathrm{m}^2$
Bettmaterial: Glaskugeln	
Gesamtmasse	$m_P = 18\,\mathrm{kg}$
Partikeldurchmesser	$d_P = 1,16\,\mathrm{mm}$
Dichte	$\rho_P = 2471\,\frac{\mathrm{kg}}{\mathrm{m}^3}$
spezifische Wärmekapazität	$c_{pP} = 750\,\frac{\mathrm{J}}{\mathrm{kgK}}$
Trägerluft	
Luftmassenstrom	$\dot{m}_{L,ein} = 0,3\,\frac{\mathrm{kg}}{\mathrm{s}}$
Temperatur	$\vartheta_{L,ein} = 80°\mathrm{C}$
Anfangsfeuchte	$Y_{L,ein} = 0,008\,\frac{\mathrm{kg}}{\mathrm{kg}}$
Düse	
Position	$\underline{x}_D = (\frac{1}{2}d_{App}, \frac{1}{2}d_{App}, \frac{1}{5}h_{WS})^T$
Strahlrichtung	$\underline{n}_D = (0,0,1)^T$ (nach oben)
Halber Öffnungswinkel des Strahles	$\theta_D = 30°$
Eingedüste Flüssigkeit	
Quellstrom (reines Wasser)	$\dot{m}_Q = 4,17 \cdot 10^{-3}\,\frac{\mathrm{kg}}{\mathrm{s}}$
Temperatur	$\vartheta_Q = 20°\mathrm{C}$
Geschätzte Parameter	
Flüssigkeitsfilmdicke	$d_F = 100\mu m$
Abscheidegrad	$\eta_A = 0,1$

Koeffizienten. Die Gesamtbilanz des Luftstromes liefert die folgenden Größen:

	Einströmende Luft	Ausströmende Luft
Massenstrom	$\dot{m}_{L,ein}$=$0,3\,\frac{\mathrm{kg}}{\mathrm{s}}$	$\dot{m}_{L,aus}$=$0,304\,\frac{\mathrm{kg}}{\mathrm{s}}$
Luftfeuchte	$Y_{L,ein}$ =$8,0\cdot10^{-3}\,\frac{\mathrm{kg}}{\mathrm{kg}}$	$Y_{L,aus}$ =$22,0\cdot10^{-3}\,\frac{\mathrm{kg}}{\mathrm{kg}}$
Temperatur	$\vartheta_{L,ein}$ =80,0°C	$\vartheta_{L,aus}$ =45,7°C

Mit den Annahmen $\overline{\vartheta}_L = 50°\mathrm{C}$, $\overline{Y}_L = 15\cdot10^{-3}\,\frac{\mathrm{kg}}{\mathrm{kg}}$ ergeben sich die folgenden Werte:

Dichte der Luft	ρ_L =$1,082\,\frac{\mathrm{kg}}{\mathrm{m}^3}$
Kinematische Viskosität	ν_L =$1,790\cdot10^{-5}\,\frac{\mathrm{m}^2}{\mathrm{s}}$
Leerrohr-Gasgeschwindigkeit	w_0 =$2,221\,\frac{\mathrm{m}}{\mathrm{s}}$
Leerrohr-Reynoldszahl	Re_0=143,9
Archimedes-Zahl	Ar =$1,09\cdot10^5$

Die von verschiedenen Autoren angegebenen empirischen Formeln liefern auch hier deutlich voneinander abweichende Werte der Porosität:

	nach TGR [9]	nach Becher [3]
Reynolds bei Minimalfluidisation	Re_{mf}=34,89	Re_{mf}=38,87
Reynolds im Austragspunkt	Re_{AP}=496,83	Re_{AP}=381,23
Porosität	ε =0,606	ε =0,665
Wirbelschichthöhe	H_{WS} =0,147 m	H_{WS} =0,173 m

Gewählt wird der größere Wert $\varepsilon = 0,665$, da die zu diesem Wert gehörende Wirbelschichthöhe besser mit dem beobachteten Wert von 0,3 m übereinstimmt.

Die weiteren Parameter ergeben sich wie folgt:

Reale Strömungsgeschwindigkeit	w_L =$3,32\,\frac{\mathrm{m}}{\mathrm{s}}$
Mittlere Tropfenweglänge	S_{Tr} =15,7 mm
Dispersionskoeffizient	D_z =$0,431\,\frac{\mathrm{m}^2}{\mathrm{s}}$
Wärmeübergangskoeffizient	α_{PL}=$401,0\,\frac{\mathrm{W}}{\mathrm{m}^2\mathrm{K}}$
Stoffübergangskoeffizient	β =$0,2702\,\frac{\mathrm{m}}{\mathrm{s}}$

Die Dispersionskoeffizienten in horizontaler Richtung werden zu $D_x = D_y = 0.1D_z$ geschätzt. Für die Wärmeübergangskoeffizienten wird $\alpha_{FL} = \alpha_{PL}$, $\alpha_{PF} = 0,1\alpha_{PL}$ angenommen.

Ergebnisse. Abb. 24–Abb. 29 im Anhang B.2 zeigen die errechnete räumliche Verteilung der Größen $\dot{\kappa}_V$, κ_F, κ_D, ϑ_L, ϑ_F und ϑ_P.

6.4.3 Bewertung

Der Vergleich mit den Meßwerten zeigt, daß der im Reaktor gemessene Temperaturverlauf durch das Modell prinzipiell richtig wiedergegeben wird. Insbesondere sind der bei allen Messungen aufgezeichnete starke Temperaturabfall über dem Anströmboden und das durch die Eindüsung hervorgerufene Temperaturprofil deutlich zu erkennen.

Der quantitative Temperaturverlauf im Bedüsungsgebiet stimmt nicht mit den Meßwerten überein: gemessenen Temperaturschwankungen von einigen 10 K stehen berechnete Werte gegenüber, die ca. um den Faktor 10 kleiner sind. Vermutlich wird diese Abweichung dadurch hervorgerufen, daß die Feststoffvermischung in der Realität deutlich kleiner ist als im Modell angenommen.

A Nomenklatur

Allgemein verwendete Symbole

$\mathbb{N}$	Natürliche Zahlen
$\mathbb{R}$	Reelle Zahlen
$\mathbb{E}$	Euklidischer Raum
M	Dimension des Gleichungssystems
N	Zahl der Raumdimensionen
$\overline{x}$	für $x \in \mathbb{R}$: Komplement zur Eins $\overline{x} := 1 - x$ für $x \subset \mathbb{R}^{\mathsf{N}}$: Abschluß (bezüglich der euklidischen Norm-Topologie)
$\underline{x}$	Vektor
$\underline{\underline{x}}$	Matrix
$\overline{\Omega}$	Gebiet
$\partial\Omega$	Rand von Ω

`ug`	Das Programmpaket „ug“
`ug-3.8`	`ug`-Version 3.8 vom Herbst[17] 1998

Zur physikalischen Modellierung

Koordinaten.

Kartesische Raumkoordinaten x, y, z [m]

Kugelkoordinaten r [m], ϕ, θ [rad]
Umrechnung in lokale kartesische Koordinaten (ξ, η, ζ):

$$\begin{aligned} \xi &= r\cos\phi\sin\theta \\ \eta &= r\sin\phi\sin\theta \\ \zeta &= r\cos\theta \end{aligned}$$

Zeit t [s]

Flächen- und Volumenbezug. Für eine Größe $Q : (\mathbb{R}^3 \times \mathbb{R}) \supset (\Omega \times [0, T]) \to \mathbb{R}$ werden die folgenden abgeleiteten Größen eingeführt:

q	Volumenbezogene Größe	$q := \frac{\partial Q}{\partial V}$
$q^{\square}$	Flächenbezogene Größe	$q^{\square} := \frac{\partial Q}{\partial A}$

[17] Offizielle Versionskennzeichnung: `ug 3.8 from 1998/08/07 14:47:16.` —
Einige Quelldateien der Distribution enthalten RCS-IDs vom September/Oktober 1998.

Euler- und Lagrange-Koordinaten.

$\underline{X}$	Lagrange-Koordinatentransformation	$\underline{X} : \Omega \times [0,T] \to \mathbb{R}^{\mathsf{N}}$, $\frac{\partial \underline{X}}{\partial t} = \underline{w}$
$u^{\underline{X}}$	Größe in Lagrange-Koordinaten	$u^{\underline{X}} := u \circ \underline{X}$
$\underline{\underline{J}}_X$	Jacobi-Matrix der Transformation $\underline{\xi} \mapsto \underline{X}$:	$\underline{\underline{J}}_X := \frac{\partial \underline{X}}{\partial \underline{\xi}} = \left(\frac{\partial X_i}{\partial \xi_j}\right)_{i,j=1\ldots\mathsf{N}}$
J_X	Jacobi-Determinante zu $\underline{\underline{J}}_X$	$J_X := \det(\underline{\underline{J}}_X)$

Indizes.

App	Apparat	*F*	Flüssigkeit	*Tr*	Tropfen
D	Dampf; Düse	*L*	(trockene) Luft	*V*	Verdunstung
ein	Zustrom (Trägerluft)	*P*	Partikel	*Y*	Luftfeuchte
eff	effektiv	*Q*	Quellstrom	∞	Sättigung

Größen und Einheiten.

A	Fläche	$[\,\mathrm{m}^2\,]$
A^*	spezifische Oberfläche	$[\,\mathrm{m}^{-1}\,] = [\,\frac{\mathrm{m}^2}{\mathrm{m}^3}\,]$
α	Wärmeübergangskoeffizient	$[\,\frac{\mathrm{W}}{\mathrm{m}^2\,\mathrm{K}}\,]$
β	Stoffübergangszahl	$[\,\frac{\mathrm{m}}{\mathrm{s}}\,]$
c_p	spezifische Wärmekapazität bei konstantem Druck	$[\,\frac{\mathrm{J}}{\mathrm{kg}\,\mathrm{K}}\,]$
d	Durchmesser	$[\,\mathrm{m}\,]$
d_F	Filmdicke	$[\,\mathrm{m}\,]$
D, $\underline{\underline{D}}$	Dispersionskoeffizient, Dispersionsmatrix	$[\,\frac{\mathrm{m}^2}{\mathrm{s}}\,]$
ε	Porosität (relatives Lückenvolumen)	$[\,1\,] = [\,\frac{\mathrm{m}^3}{\mathrm{m}^3}\,]$
$\overline{\varepsilon}$	Feststoffanteil: $\overline{\varepsilon} = 1 - \varepsilon$	$[\,1\,] = [\,\frac{\mathrm{m}^3}{\mathrm{m}^3}\,]$
h	volumenbezogene Enthalpie	$[\,\frac{\mathrm{J}}{\mathrm{m}^3}\,]$
κ	Konzentration	$[\,\frac{\mathrm{kg}}{\mathrm{m}^3}\,]$
$\dot{\kappa}$	Konzentrationsstrom	$[\,\frac{\mathrm{kg}}{\mathrm{m}^3\mathrm{s}}\,]$
S_{Tr}	Mittlere Tropfenweglänge	$[\,\mathrm{m}\,]$
m	Masse	$[\,\mathrm{kg}\,]$
$\dot{m}$	Massenstrom	$[\,\frac{\mathrm{kg}}{\mathrm{s}}\,]$
M	molare Masse	$[\,\frac{\mathrm{kg}}{\mathrm{mol}}\,]$
p	Druck	$[\,\mathrm{Pa}\,] = [\,\frac{\mathrm{N}}{\mathrm{m}^2}\,]$
φ	Benetzungsgrad; $\overline{\varphi} = 1 - \varphi$	$[\,1\,] = [\,\frac{\mathrm{m}^2}{\mathrm{m}^2}\,]$
$\dot{q}^{\square}$	flächenbezogener Wärmestrom	$[\,\frac{\mathrm{W}}{\mathrm{m}^2\mathrm{s}}\,]$
$\dot{q}$	volumenbezogener Wärmestrom	$[\,\frac{\mathrm{W}}{\mathrm{m}^3\mathrm{s}}\,]$
R	allgemeine Gaskonstante	$[\,\frac{\mathrm{J}}{\mathrm{mol}\cdot\mathrm{K}}\,]$
R_i	spezielle Gaskonstante	$[\,\frac{\mathrm{J}}{\mathrm{kg}\cdot\mathrm{K}}\,]$
ρ	Dichte	$[\,\frac{\mathrm{kg}}{\mathrm{m}^3}\,]$
T	(absolute) Temperatur	$[\,\mathrm{K}\,]$
ϑ	(Celsius-)Temperatur	$[\,^\circ\mathrm{C}\,]$
V	Volumen	$[\,\mathrm{m}^3\,]$
w	Geschwindigkeit	$[\,\frac{\mathrm{m}}{\mathrm{s}}\,]$
Y	Feuchte	$[\,1\,] = [\,\frac{\mathrm{kg}}{\mathrm{kg}}\,]$

Spezielle Größen.

Δh_V	spezifische Verdampfungswärme	$[\,\frac{\mathrm{J}}{\mathrm{kg}}\,]$
$p_{D,\infty}$	Sättigungsdampfdruck	$[\,\mathrm{Pa}\,]$

Zur Diskretisierung

Diskretisierung.

A	Matrix des diskretisierten Systems
B	Randsegment (boundary segment)
C	Ecke (corner)
D	Diskrete Diffusionsmatrix
E	Kante (edge)
F	Seite (face)
G	FE-Gitter (primales Gitter)
H	FV-Gitter (duales Gitter)
I	Diskretisierungsoperator
J	Jacobi-Matrix
K	Kante im dualen Gitter
L	Element (element, „Land“)
M	Element-Mittelpunkt; Anzahl der Gleichungen
N	Zahl der Raumdimensionen
P	Polygonales Gebiet
R	Referenzelement
$\mathcal{R}$	Menge zugelassener Referenzelemente
S	Seite (side)
T	Referenztransformation
$\mathcal{T}$	Menge zugelassener Referenztransformationen
U	Ansatzraum
V	Kontrollvolumen
W	Diskrete Konvektionsmatrix
Z	Diskreter Reaktionsterm
b	Diskrete rechte Seite
c	Korrekturvektor (correction)
d	Defektvektor
f	Diskreter Quellterm
g	Diskretisierte Neumann-Randbedingung
h	Gitter-Weite, Element-Größe
m	Zahl der Ecken eines Elementes
n	Zahl der Elemente im Gitter
p	Lokale Koordinaten
s	Schwerpunkt
u	Vektor der Unbekannten
v	Formfunktion (shape function)
x	Eckpunkt(-koordinaten)

Mehrgitterverfahren.

$\mathcal{M}_i^k$	Mehrgitterzyklus (Gitter-Level i bis k)
$\mathcal{S}$	Glättungsoperator
$\mathcal{L}$	Lösungsoperator
$\mathcal{R}$	Restriktionsoperator
$\mathcal{P}$	Prolongationsoperator
$\mathcal{S}_{JAC}$	Jacobi-Glätter
$\mathcal{S}_{GS}$	Gauß-Seidel-Glätter
ν_1, ν_2	Anzahl von Vor- und Nachglättungsschritten
γ	Anzahl von Iterationen pro Gitter-Level
l_0	Basis-Level

Software, Quellcode

Schreibweisen. In den Ausführungen zur Software werden die folgenden Abkürzungen und Schreibweisen verwendet:

`some text`	Quelltext
`<var>`	variabler Parameter
`$(UGROOT)/`	Stammverzeichnis der `ug`-Installation
(1234)	Zeilennummer im Quelltext
(567ff)	Zeilen 567 und folgende
†	Undokumentiert
‡	Nicht existent; nicht verfügbar

Dokumentation von Fehlern. Die Beschreibung der einzelnen Fehler im Anhang D ist im allgemeinen in fünf Abschnitte gegliedert:

A Erscheinungsbild
B Nähere Umstände des Auftretens
C Fehlerursachen
D Weiterführende Hinweise
E Beseitigung des Fehlers

Zu den Modellrechnungen

Dimensionslose Kennzahlen.

Archimedes	$Ar = \frac{gd^3(\rho_S - \rho_L)}{\nu_L^2 \rho_L}$	Prandtl	$Pr = \frac{c_p \nu \rho}{\lambda}$
Nusselt	$Nu = \frac{\alpha d}{\lambda}$	Schmidt	$Sc = \frac{\nu}{D}$
Reynolds	$Re = \frac{wd}{\nu}$	Sherwood	$Sh = \frac{\beta d}{D}$
Peclet	$Pe = \frac{wd}{D} = Re \cdot Sc$		

B Grafische Darstellungen der Simulationsergebnisse

B.1 Vereinfachtes Modellproblem

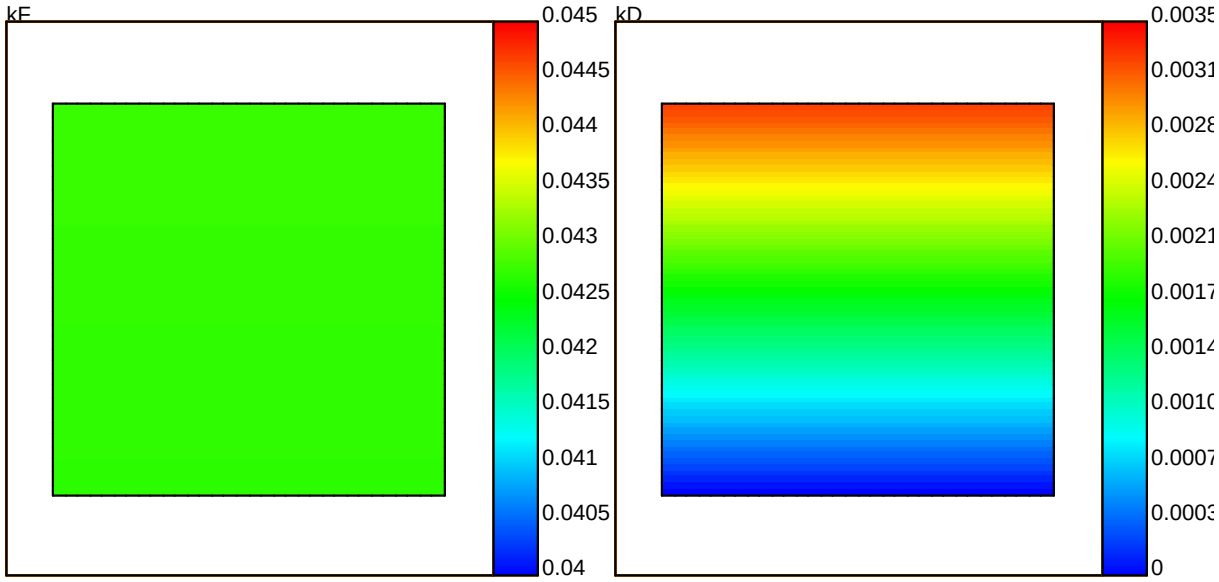


Abbildung 14: Simulationsergebnisse zum vereinfachten Modell, ungesättigte Luft $\dot{m}_Q = 0.01\,\frac{\mathrm{kg}}{\mathrm{s}}$ gleichmäßig über Würfel verteilt

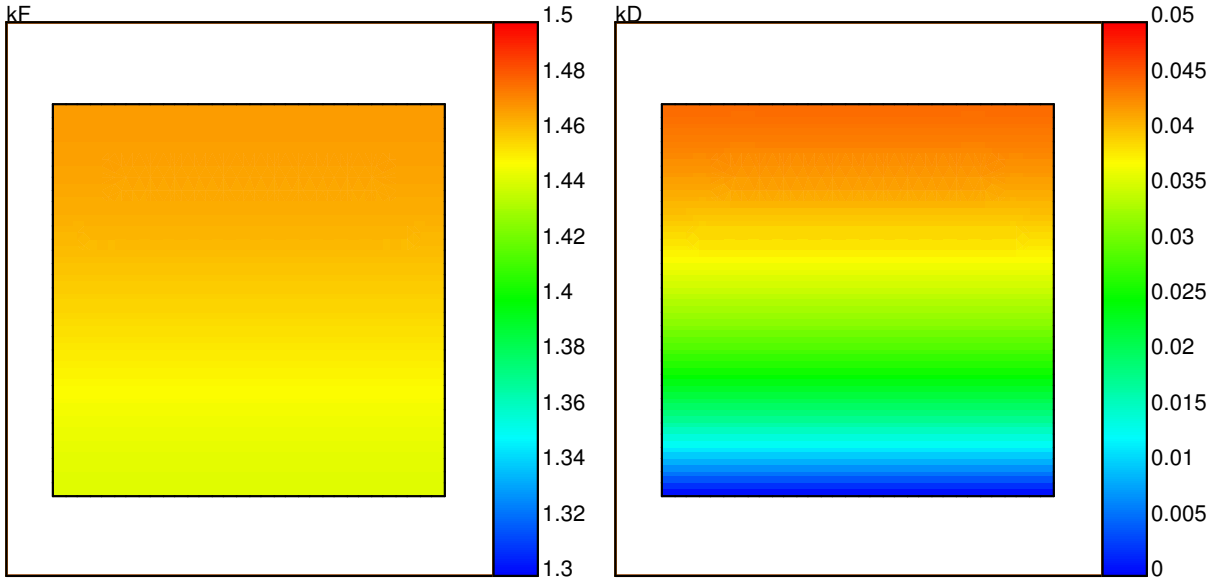


Abbildung 15: Simulationsergebnisse zum vereinfachten Modell, nahezu gesättigte Luft $\dot{m}_Q = 0.14\,\frac{\mathrm{kg}}{\mathrm{s}}$ gleichmäßig über Würfel verteilt

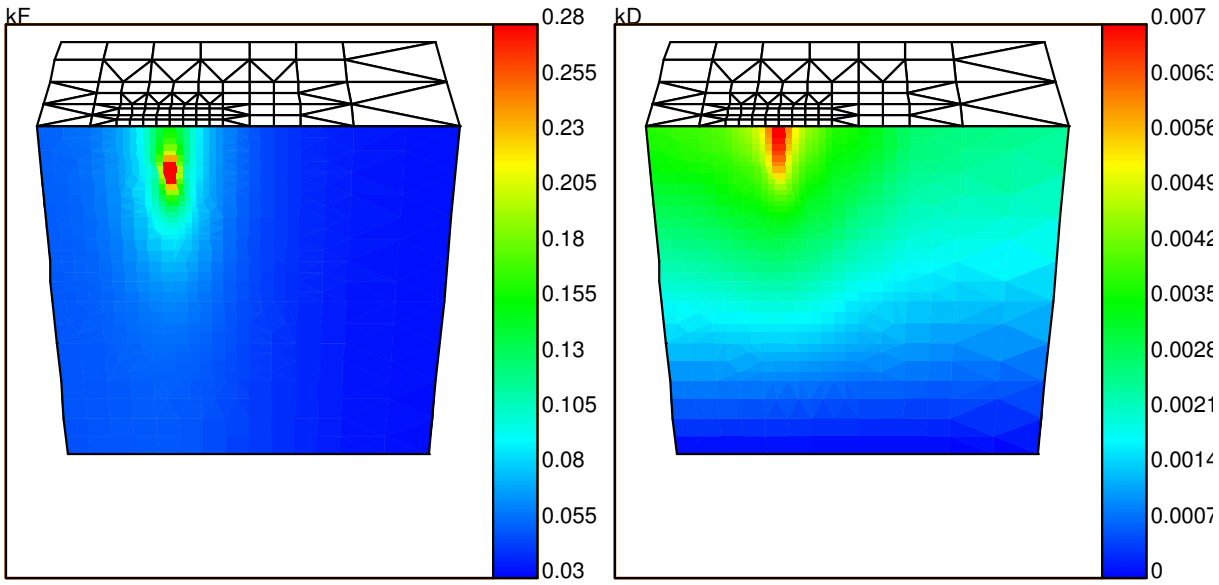


Abbildung 16: Simulationsergebnisse zum vereinfachten Modell, Eindüsung auf Würfel
Düse bei $(0.3, 0.5, 0.9)$ strahlt nach unten, $\dot{m}_Q = 0.01 \frac{\text{kg}}{\text{s}}$

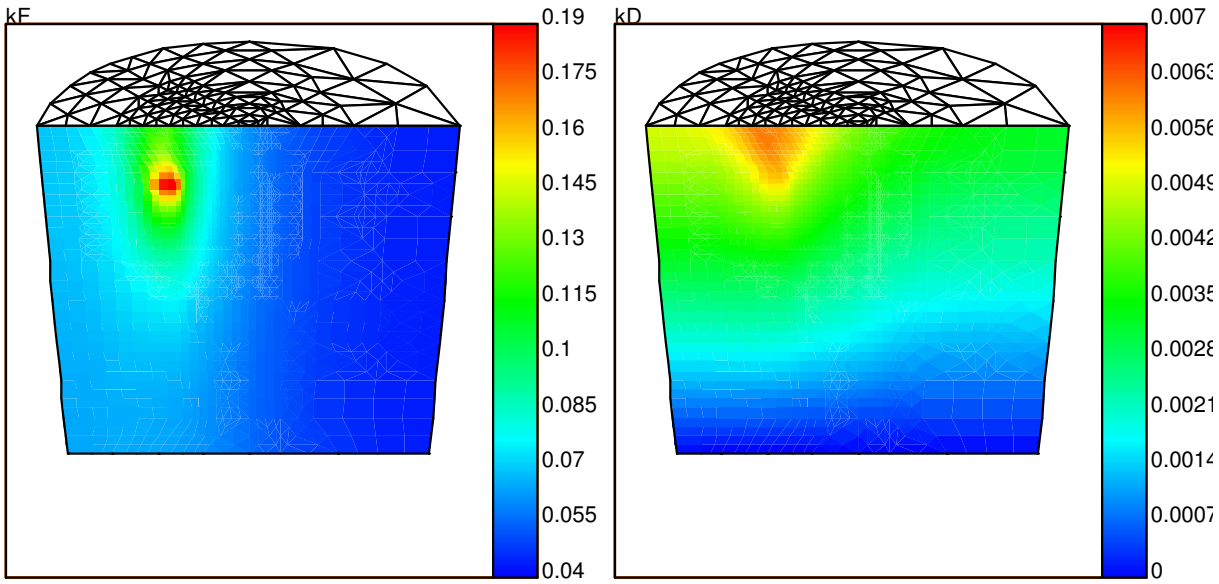


Abbildung 17: Simulationsergebnisse zum vereinfachten Modell. Eindüsung auf Zylinder
Düse bei $(0.3, 0.5, 0.9)$ strahlt nach unten, $\dot{m}_Q = 0.01 \frac{\text{kg}}{\text{s}}$

B.2 Versuch 15

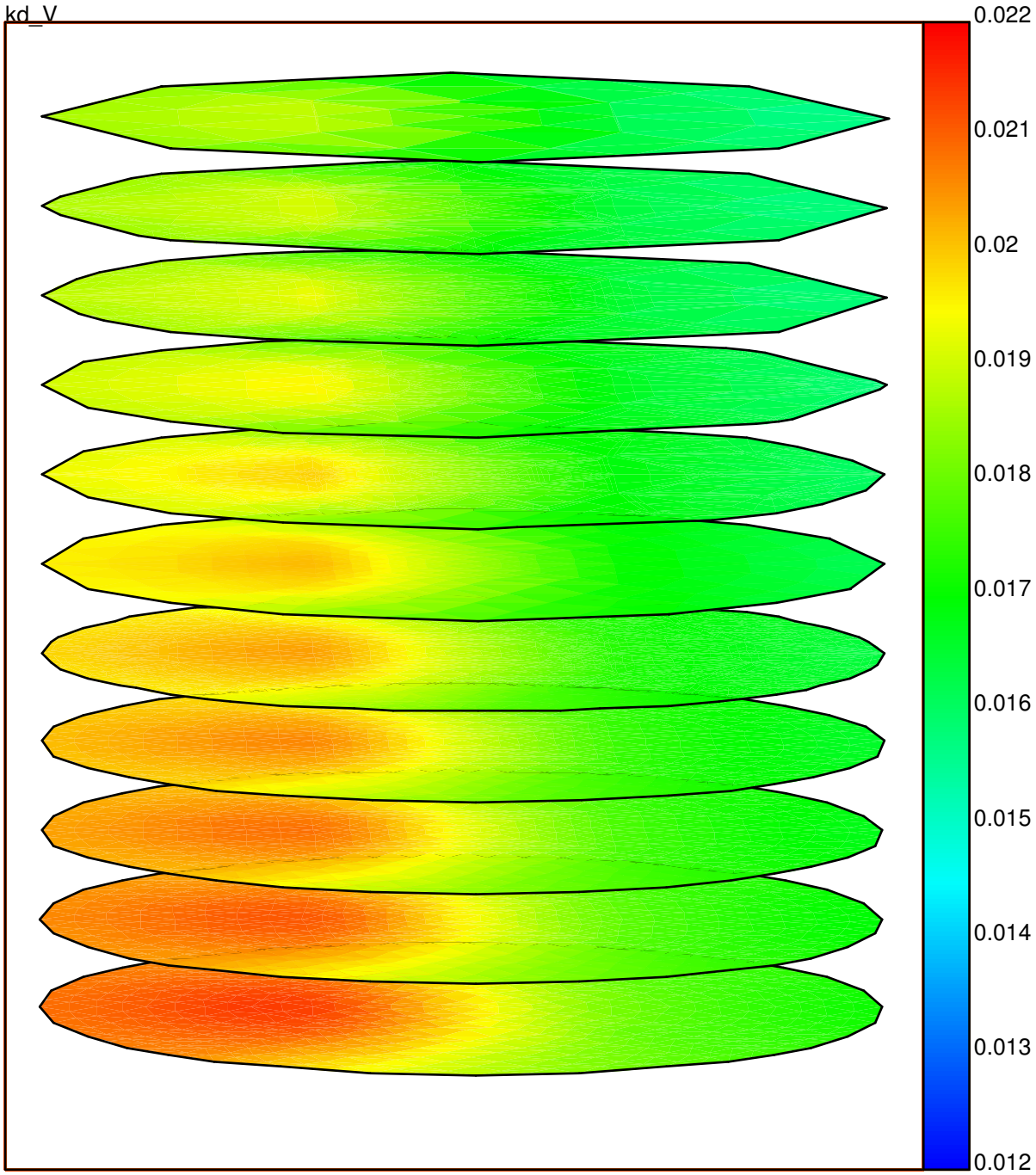


Abbildung 18: Simulationsergebnisse zu Versuch 15, $\dot{\kappa}_V$ [$\frac{\mathrm{kg}}{\mathrm{m^3 s}}$]

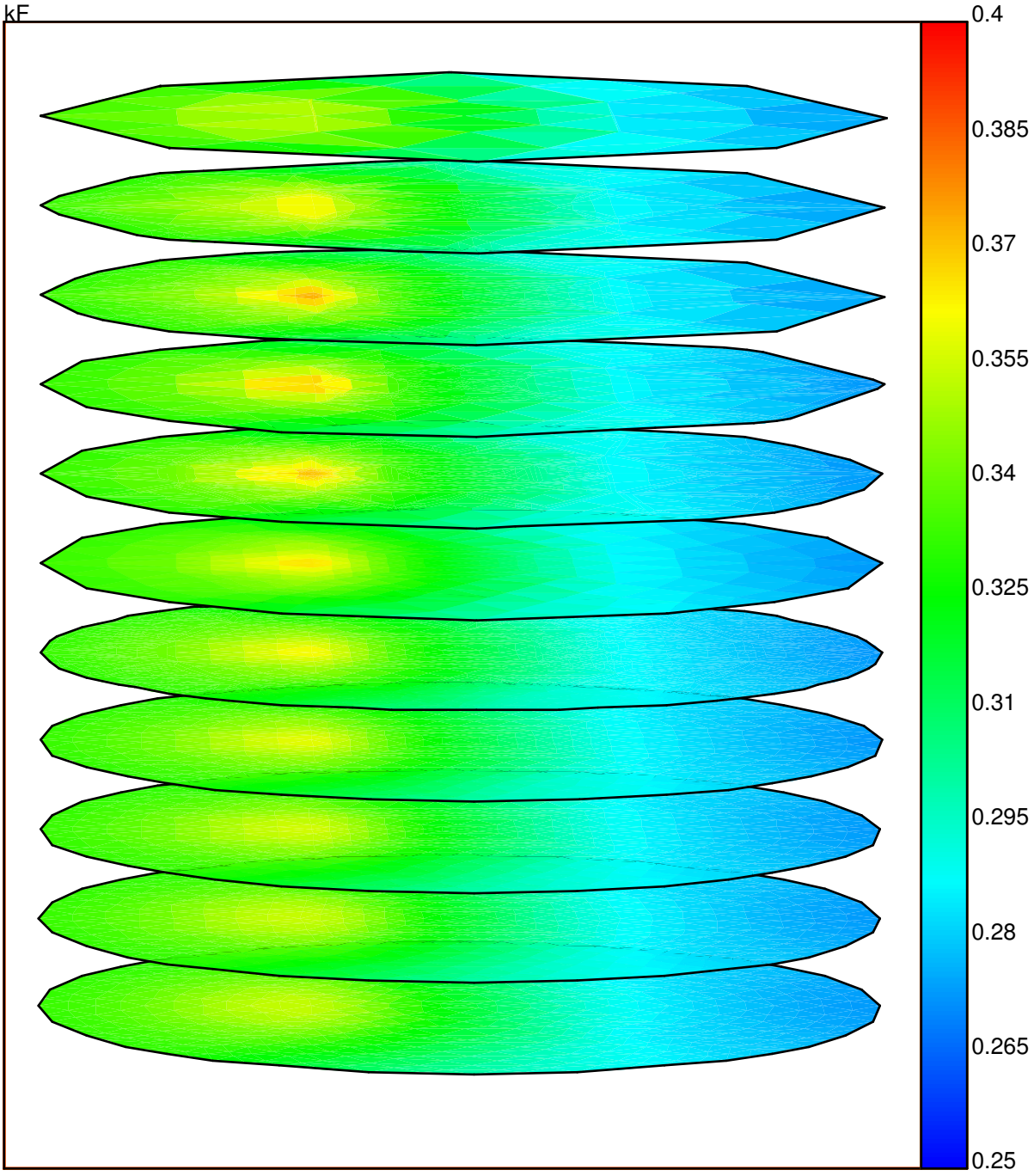


Abbildung 19: Simulationsergebnisse zu Versuch 15, κ_F $[\,\frac{\mathrm{kg}}{\mathrm{m}^3}\,]$

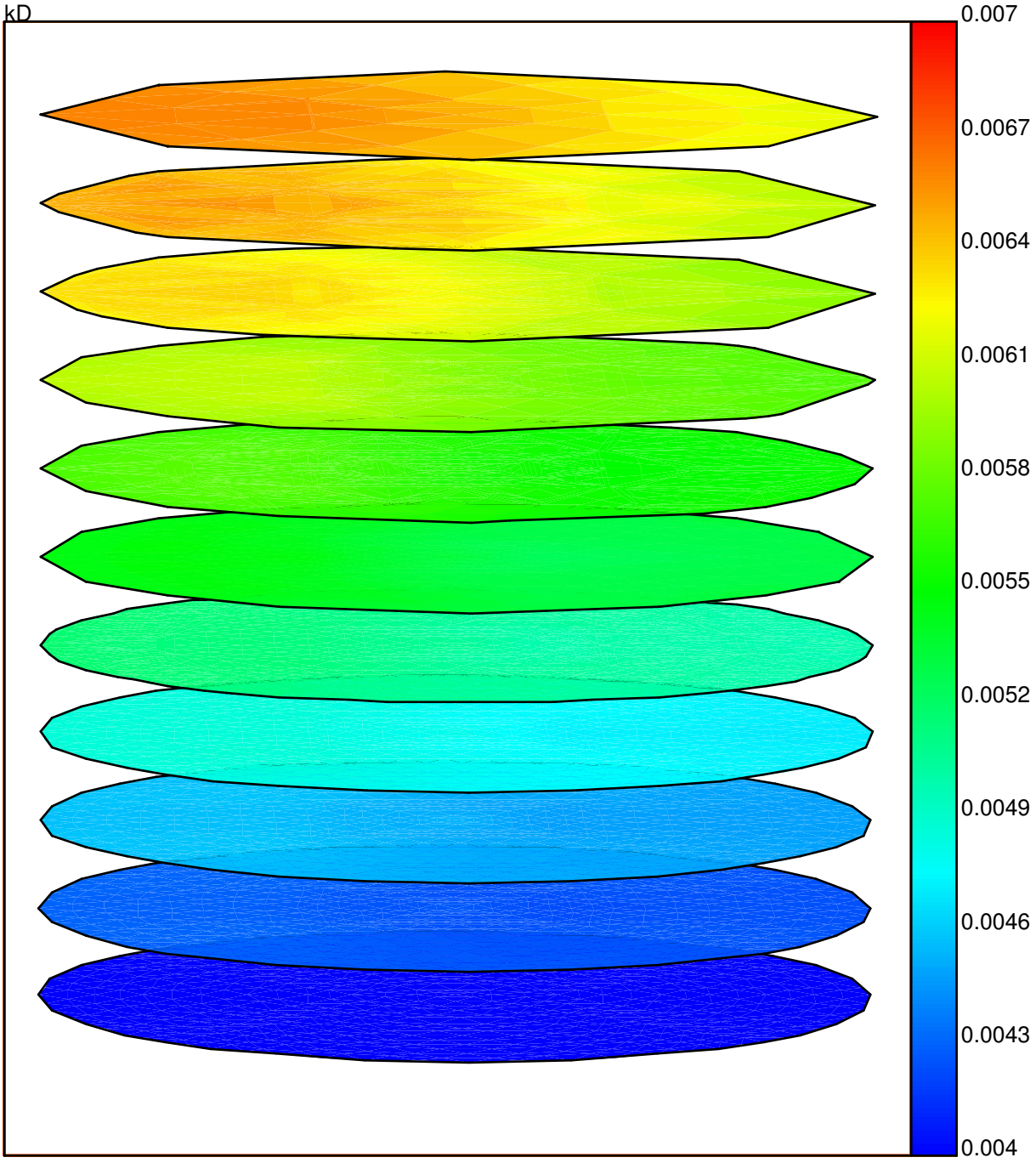


Abbildung 20: Simulationsergebnisse zu Versuch 15, κ_D [$\frac{\text{kg}}{\text{m}^3}$]

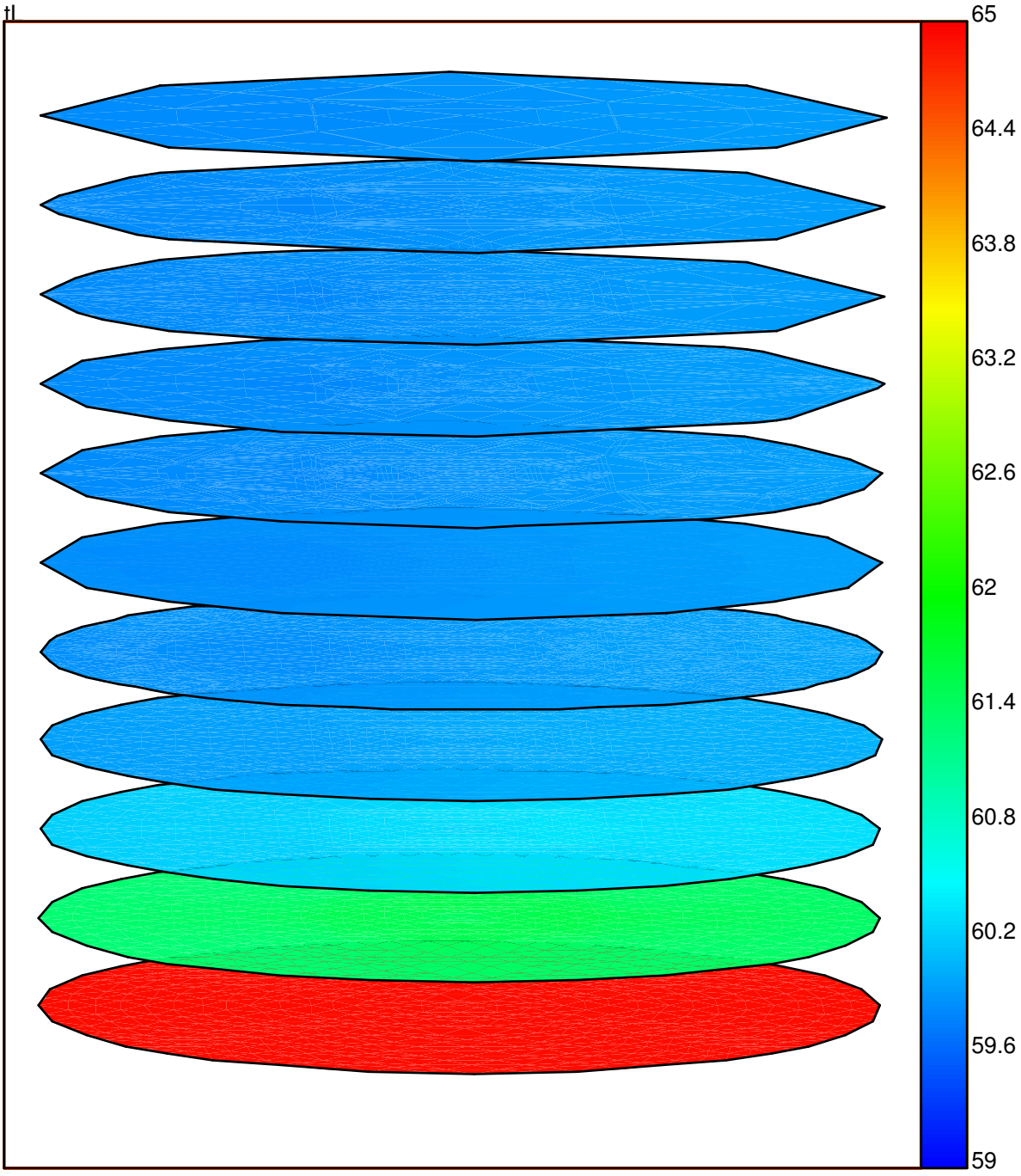


Abbildung 21: Simulationsergebnisse zu Versuch 15, ϑ_L [°C]

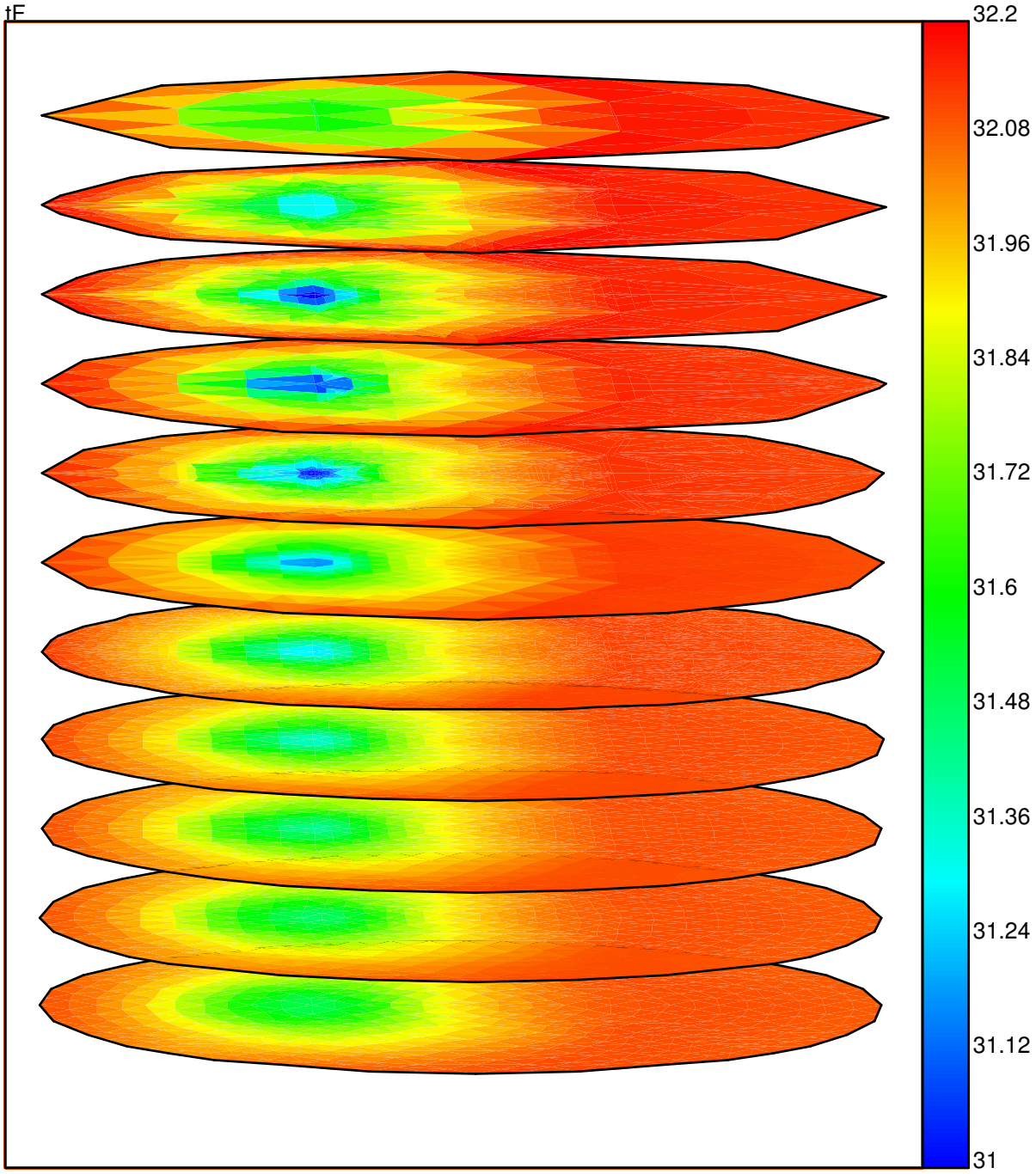


Abbildung 22: Simulationsergebnisse zu Versuch 15, ϑ_F [°C]

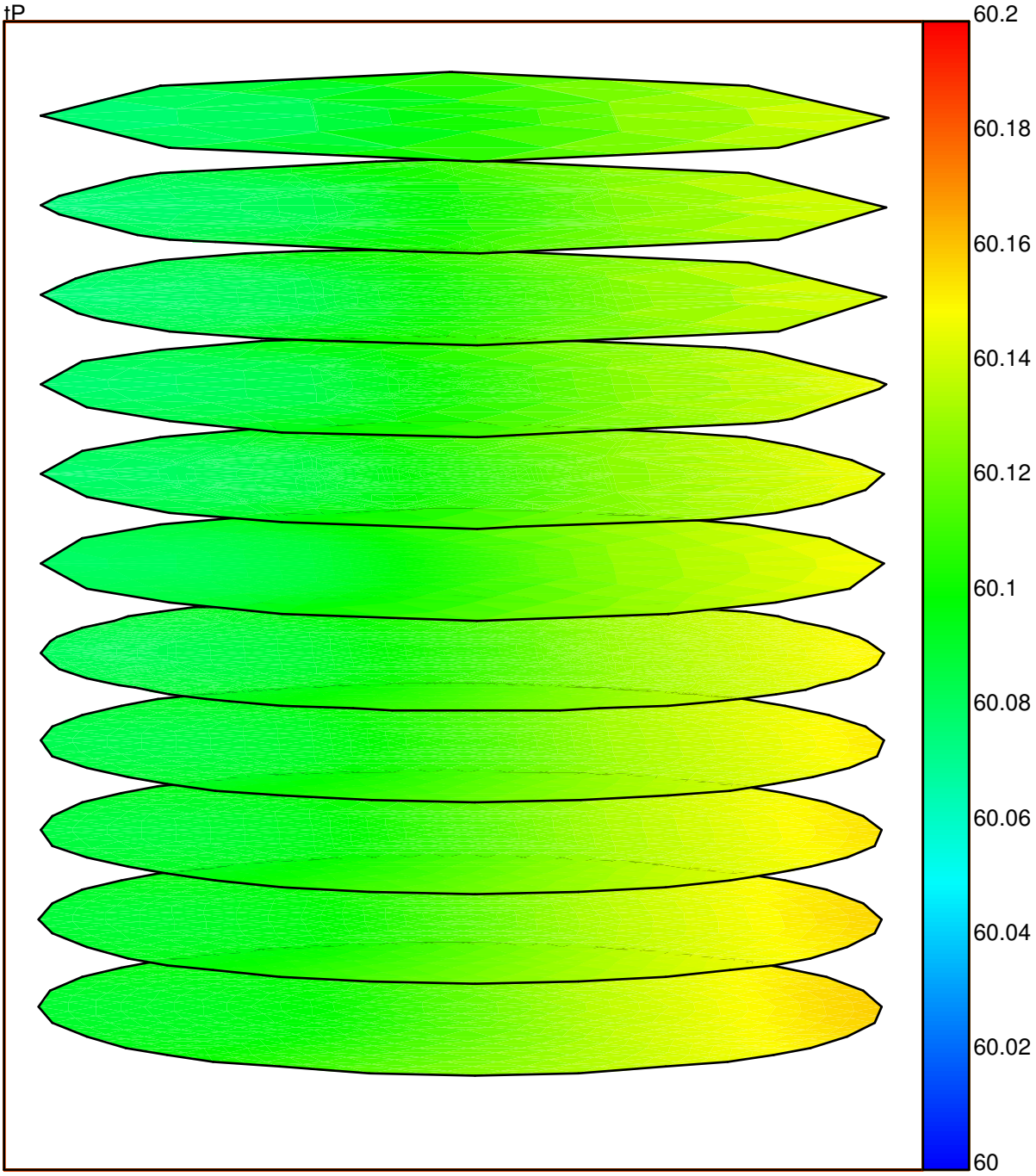


Abbildung 23: Simulationsergebnisse zu Versuch 15, ϑ_P [°C]

B.3 Versuch 97

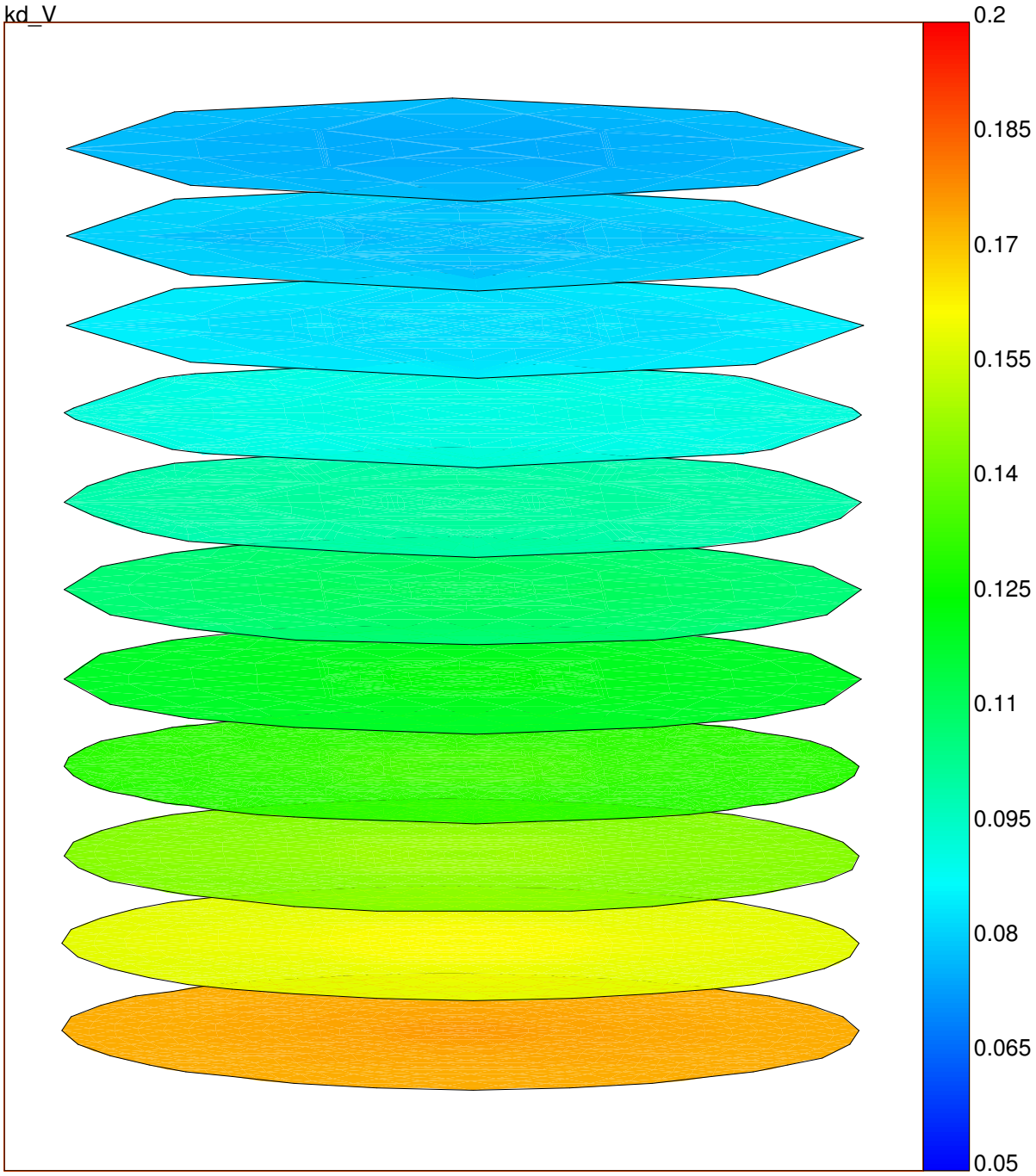


Abbildung 24: Simulationsergebnisse zu Versuch 97, $\dot{\kappa}_V$ [$\frac{\mathrm{kg}}{\mathrm{m^3 s}}$]

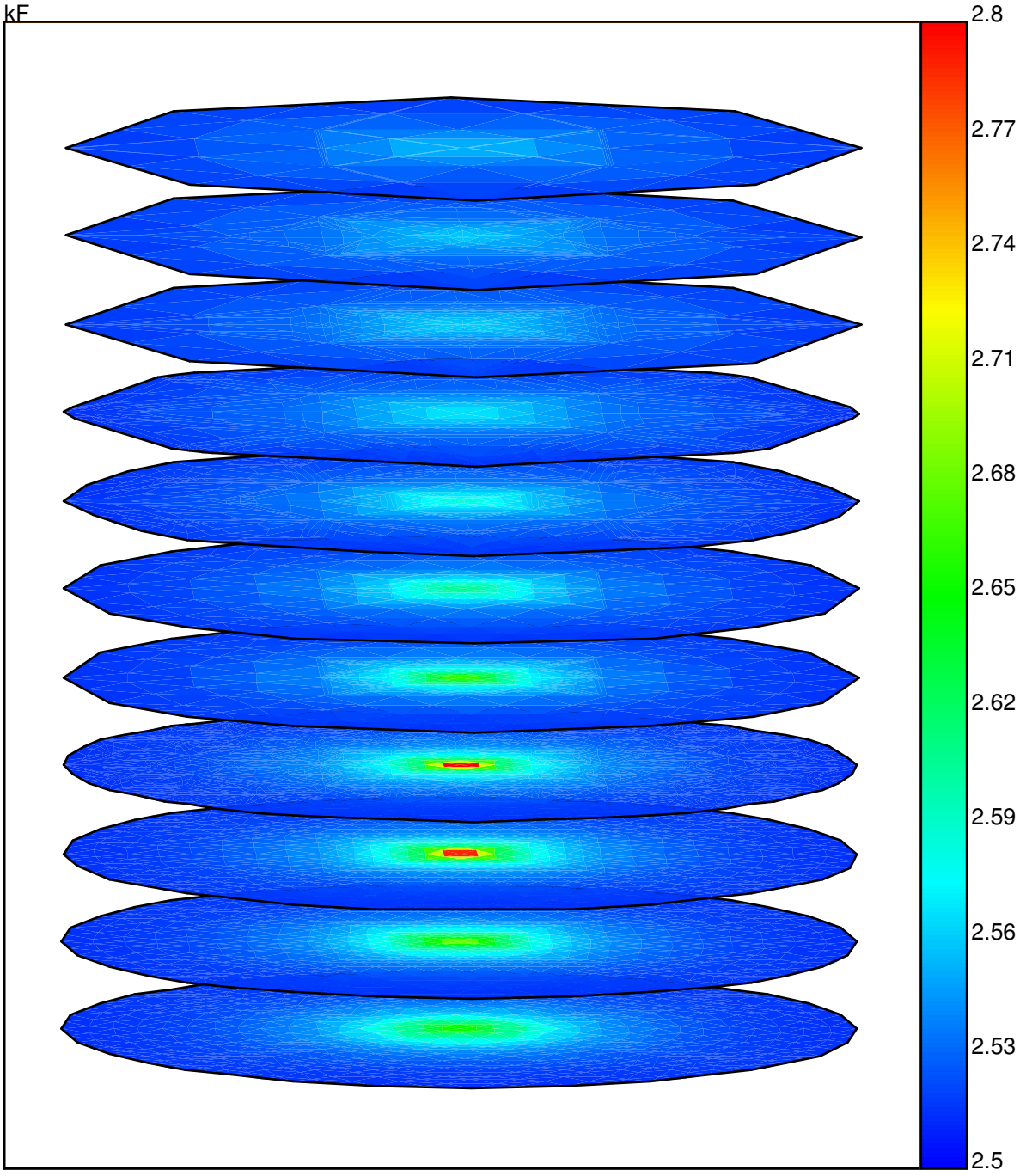


Abbildung 25: Simulationsergebnisse zu Versuch 97, κ_F $[\,\frac{\mathrm{kg}}{\mathrm{m}^3}\,]$

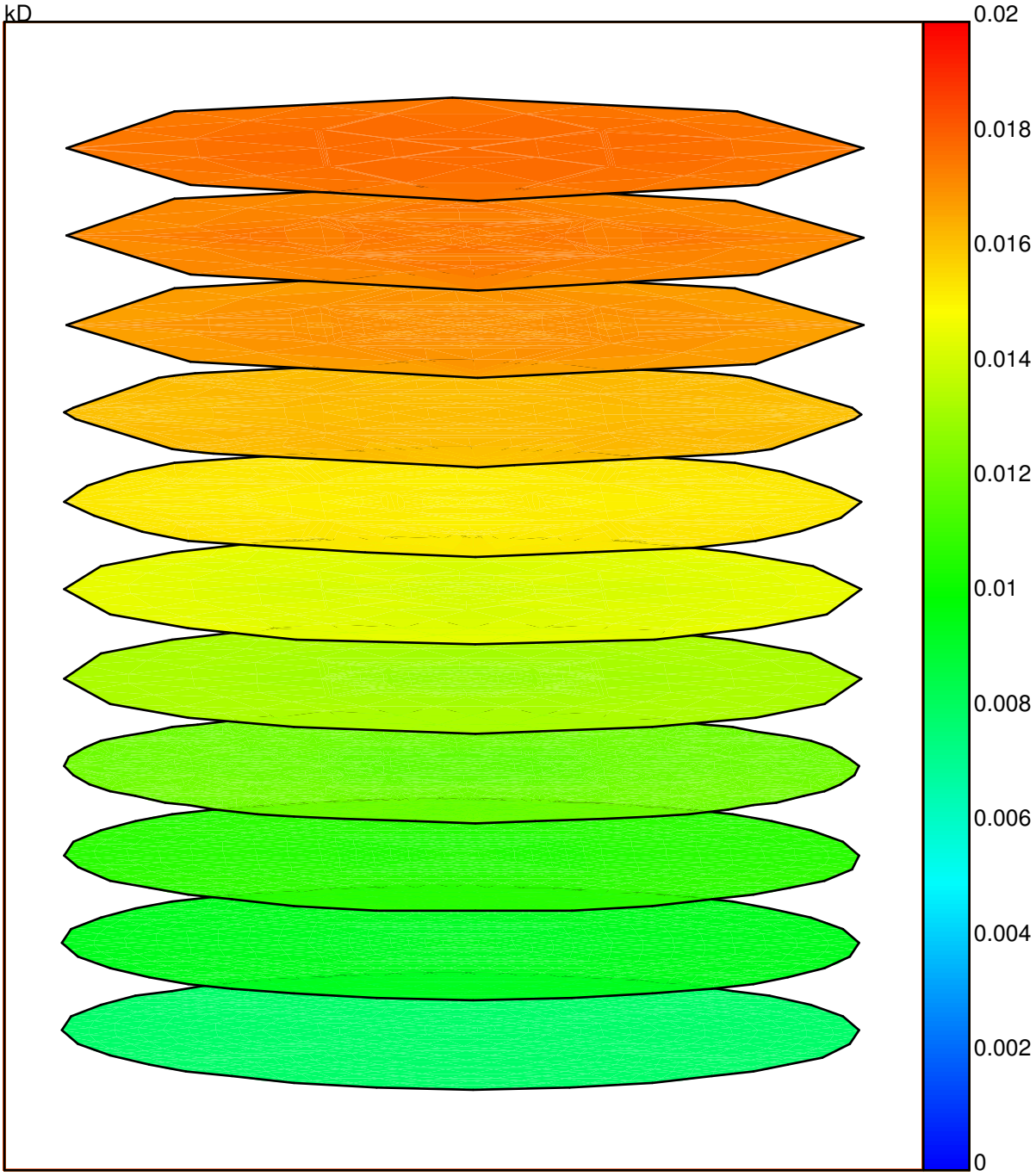


Abbildung 26: Simulationsergebnisse zu Versuch 97, κ_D [$\frac{\mathrm{kg}}{\mathrm{m}^3}$]

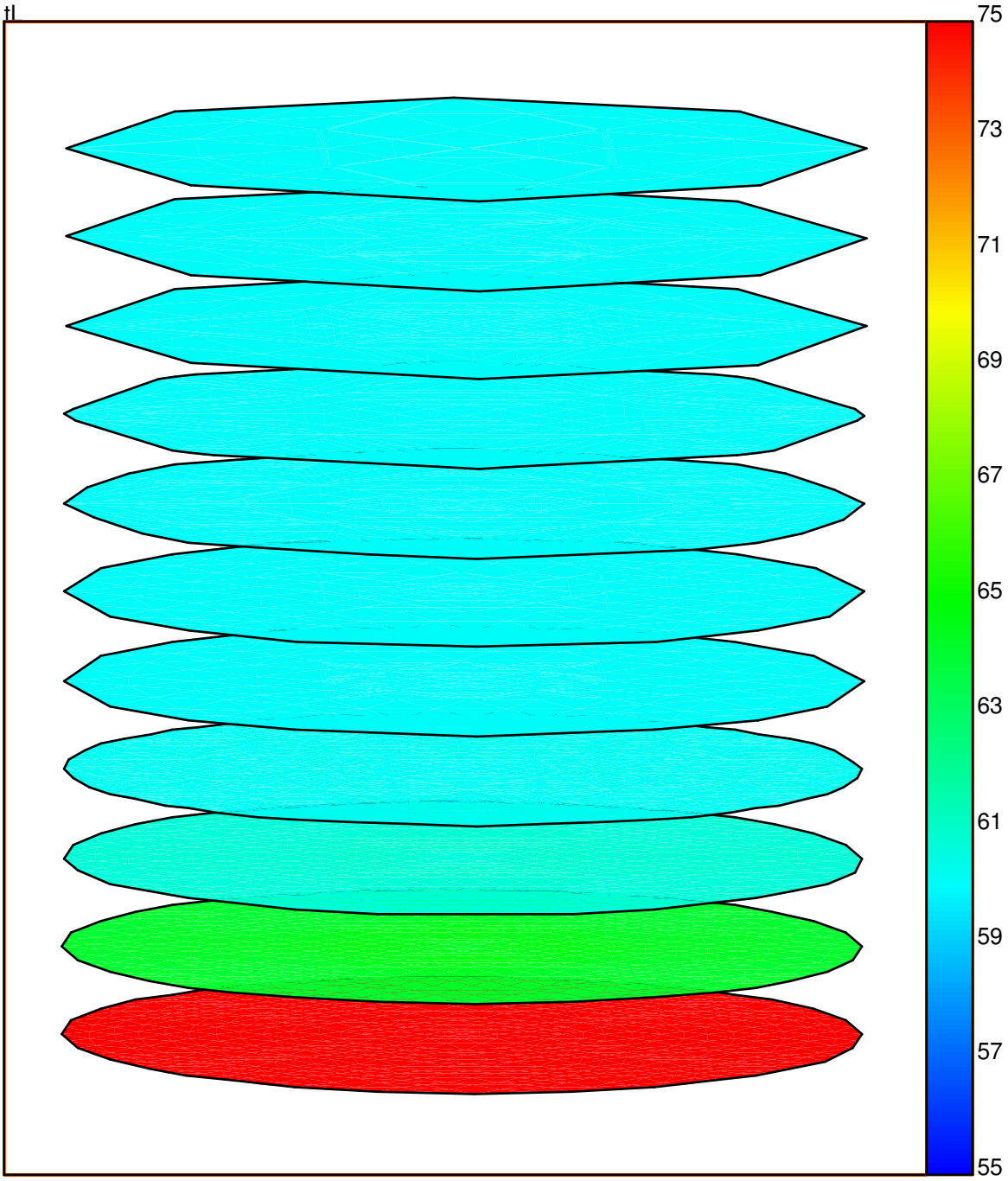


Abbildung 27: Simulationsergebnisse zu Versuch 97, ϑ_L [°C]

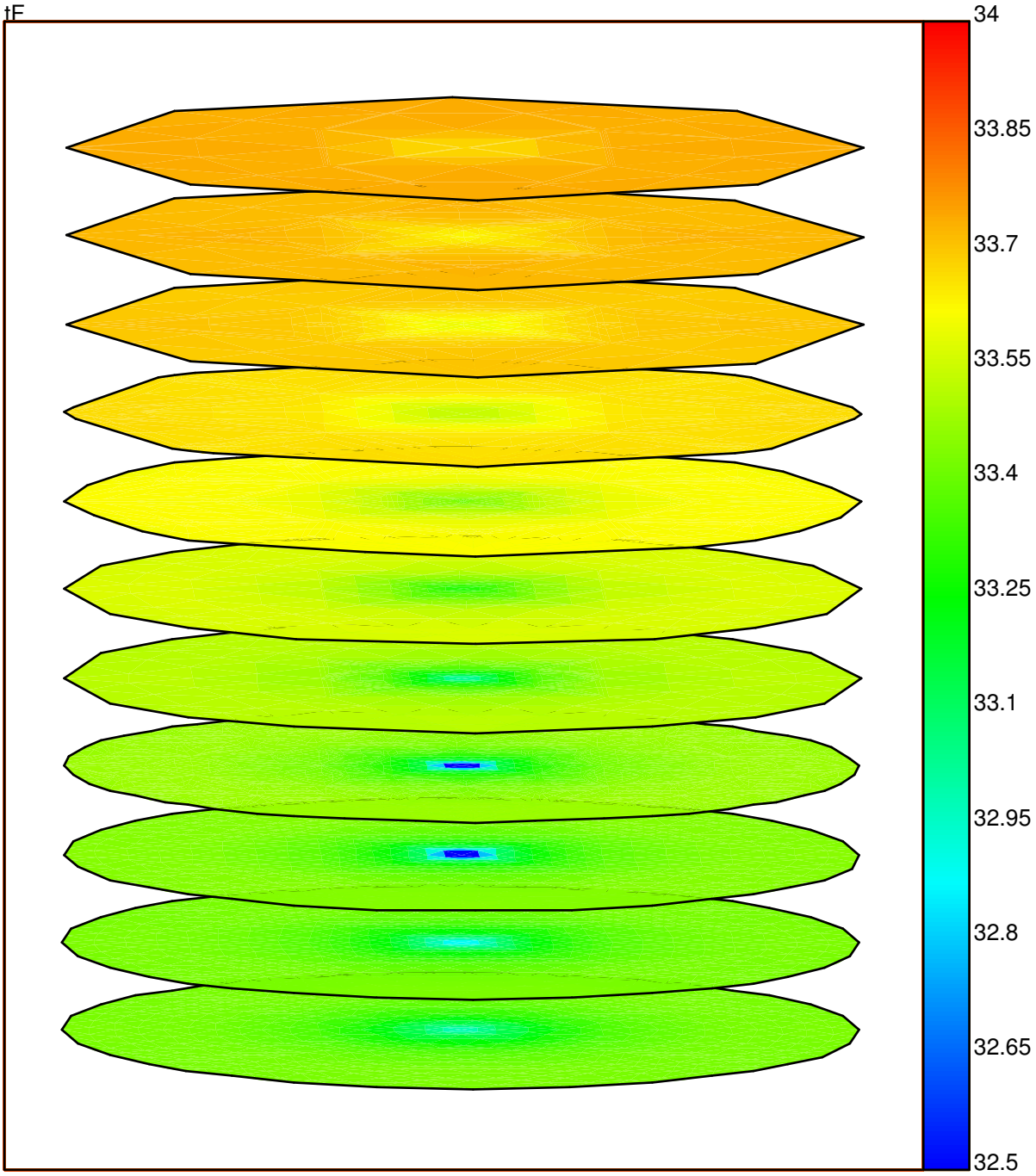


Abbildung 28: Simulationsergebnisse zu Versuch 97, ϑ_F [°C]

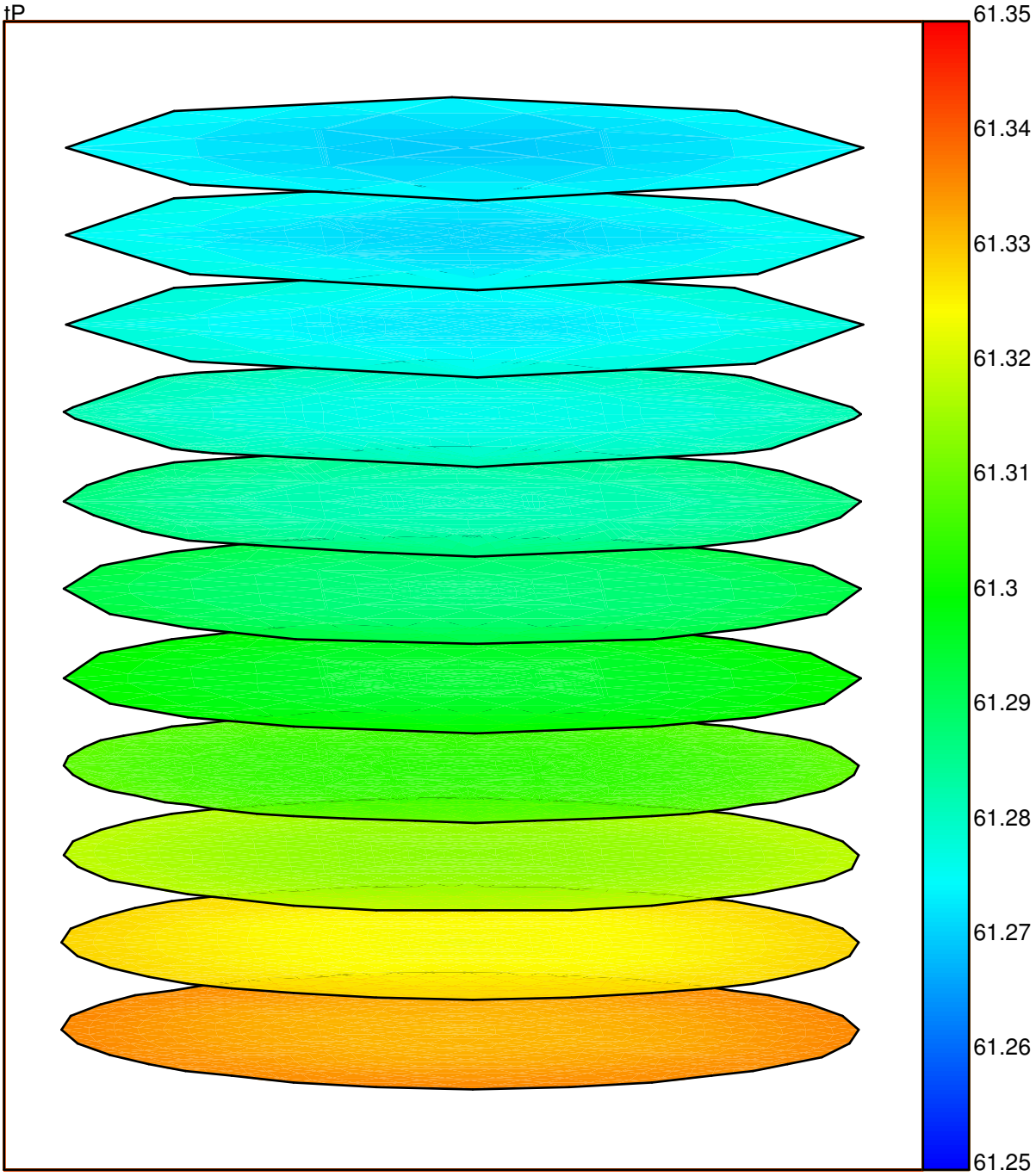


Abbildung 29: Simulationsergebnisse zu Versuch 97, ϑ_P [°C]

C Ergänzungen zur Geometrie

C.1 Parametrisierung des Randes

Auf den folgenden Seiten ist eine Reihe von Details zusammengestellt, die die Parametrisierung der Randsegmente zur Gebietsbeschreibung in ug betreffen. Diese sind im Zusammenhang der Ausführungen im Abschnitt 3.1.5 zu verstehen.

C.1.1 Begriffe

Regularität der Parametrisierung. Es ist nicht ohne weiteres klar, wie Randflächen, die nicht von der Gestalt eines Vierecks sind, parametrisiert werden können, ohne die Regularität der einbeschriebenen verfeinerten Gitter zu stören.

Offensichtlich sind die folgenden Bedingungen einzuhalten:

1. Eine Kante der Länge Null ist verboten, da die Jacobi-Matrix der zugehörigen Transformation verschwindet. Einbeschriebene Elemente, die eine solche Kante enthalten, haben offensichtlich einen Innenwinkel der Größe Null.

2. Ein Innenwinkel der Größe π („gestreckte“ Ecke) muß unterteilt werden:

   Sei (ohne Einschränkung) C_1 diese Ecke, C_0 und C_2 deren Nachbarn bezüglich des Randsegmentes. Das Dreieck $\mathsf{C}_0\mathsf{C}_1\mathsf{C}_2$ werde durch Halbierung der drei Kanten *im Parameterraum* und Einfügen drei neuer, die Mittelpunkte verbindender Kanten *im* $\mathbb{R}^3$ in vier Dreiecke zerlegt („reguläre Verfeinerung“).

   Diese Prozedur werde für das an C_1 grenzende Dreieck wiederholt. Für das nach dem n-ten derartigen Schritt erhaltene Dreieck geht der Innenwinkel bei C_1 mit $n \to \infty$ gegen π, sofern die Parametrisierung des Randsegmentes differenzierbar war.

   Das Problem tritt nicht auf, wenn der Innenwinkel π bei C_1 durch Einfügen einer Kante in das Grobgitter-Dreieck $\mathsf{C}_0\mathsf{C}_1\mathsf{C}_2$ halbiert wird.

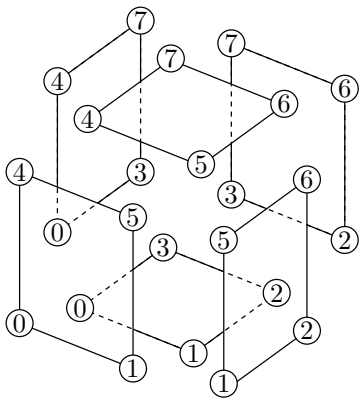
Orientierung der Randsegmente. Für die „maschinenlesbare“ Beschreibung der Randsegmente in ug ist die Orientierung der Segmente mit anzugeben. Bei der Beschreibung konkreter Parametrisierungen auf den folgenden Seiten wird die nachstehende Konvention verwendet:

Definition C.1 (Orientierung der Randsegmente) **:** *Ein durch die vier Ecken* $\mathsf{C}_0, \ldots, \mathsf{C}_3$ *gegebenes Randsegment zu* $\Omega \subset \mathbb{R}^3$ *heißt* rechts orientiert *in bezug auf das Gebiet genau dann, wenn die durch den Umlaufsinn* $\mathsf{C}_0 - \mathsf{C}_1 - \mathsf{C}_2 - \mathsf{C}_3$ *(bzw. im Parameterraum als* $\underline{x}_{\lambda_1} \times \underline{x}_{\lambda_2}$*) definierte „rechte“ Normale innere Normale an* Ω *ist. (Das Segment ist von außen gesehen rechts orientiert.)*

C.1.2 Würfel

Der Rand des Würfelgebietes $[-1, 1]^3$ wird in sechs ebene Randsegmente eingeteilt, die auf natürliche Weise parametrisiert werden (Parameterbereiche $[-1, 1]^2$). Die so definierte DOMAIN hat 6 Randsegmente und 8 Ecken.

Ost `east`	**Nord** `north`	**Deckel** `top`
C $(1, 2, 6, 5)$	C $(3, 2, 6, 7)$	C $(4, 5, 6, 7)$
links orientiert	rechts orientiert	links orientiert
$x = 1$	$x = \lambda_1$	$x = \lambda_1$
$y = \lambda_1$	$y = 1$	$y = \lambda_2$
$z = \lambda_2$	$z = \lambda_2$	$z = 1$
West `west`	**Süd** `south`	**Boden** `bot`
C $(0, 3, 7, 4)$	C $(0, 1, 5, 4)$	C $(0, 1, 2, 3)$
rechts orientiert	links orientiert	rechts orientiert
$x = -1$	$x = \lambda_1$	$x = \lambda_1$
$y = \lambda_1$	$y = -1$	$y = \lambda_2$
$z = \lambda_2$	$z = \lambda_2$	$z = -1$



Das Grobgitter und die erste Verfeinerungsstufe sind in Abb. 30 dargestellt.

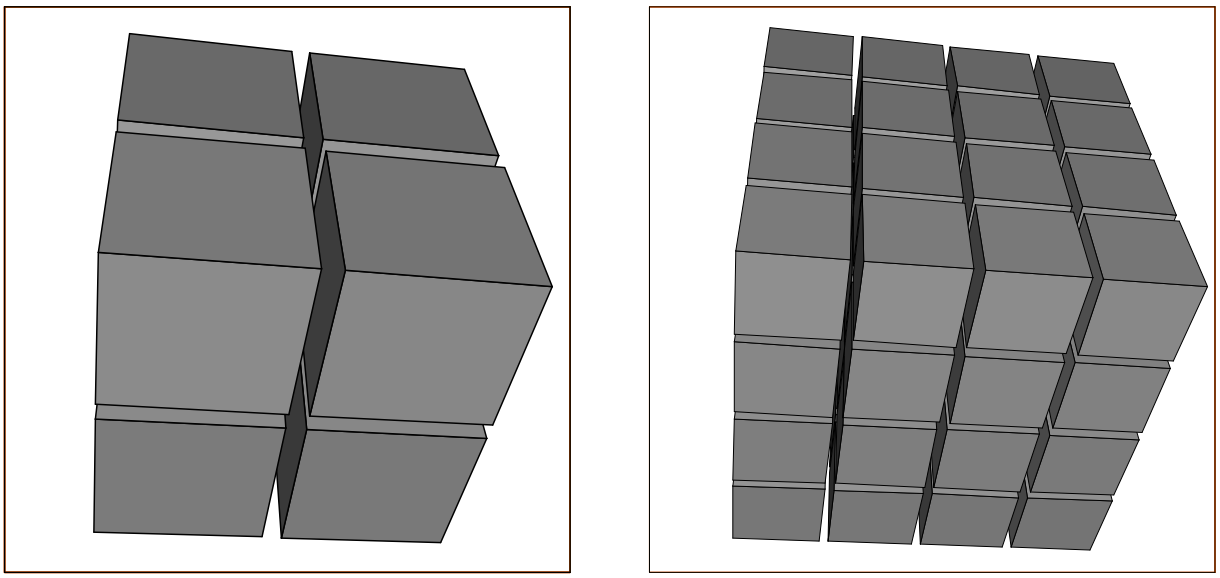


Abbildung 30: Grobgitter und erste Verfeinerung auf Würfel-Gebiet

Ein Beispiel für ein dem Würfelgebiet einbeschriebenes viermal lokal verfeinertes Gitter zeigt Abb. 31.

C.1.3 Zylinder

Die im folgenden beschriebene Rand-Parametrisierung erzeugt ein zylindrisches Gebiet mit 6 Randsegmenten und 8 Ecken. Das Gebiet ist topologisch äquivalent zu einem Würfel; die Parametrisierung des Würfels wird stückweise differenzierbar auf einen Zylinder abgebildet. Die Art der Parametrisierung wurde von `$(UGROOT)/../ns/appl3d/ns3d.c` übernommen; Achse des Zylinders ist – im Unterschied zur Vorlage in `ns3d.c` – die x_3-Achse.

Die Kreisfläche von Boden und Deckel wird parametrisiert durch

$$x_1 \;=\; \lambda_1 \frac{\|\underline{\lambda}\|_\infty}{\|\underline{\lambda}\|_2}$$

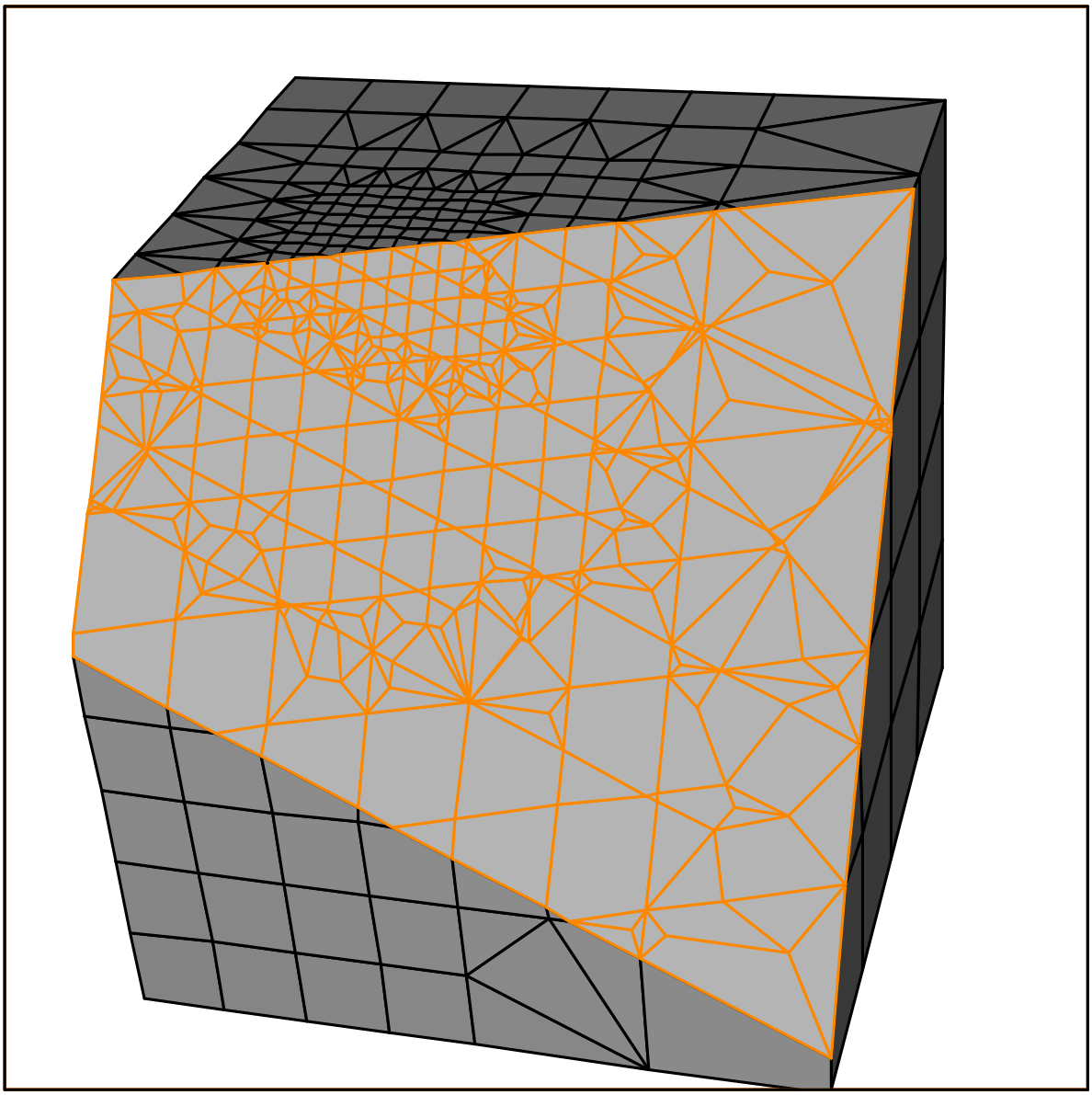


Abbildung 31: Gitter auf Würfel-Gebiet

$$x_2 \;=\; \lambda_2 \frac{\|\underline{\lambda}\|_\infty}{\|\underline{\lambda}\|_2}$$

mit stetiger Fortsetzung im Nullpunkt. Die Abb. 32 zeigt Linien konstanter Parameter λ_1 bzw. λ_2, die Linien für $\lambda_1 = 0.3$ und $\lambda_2 = 0.7$ wurden hervorgehoben.

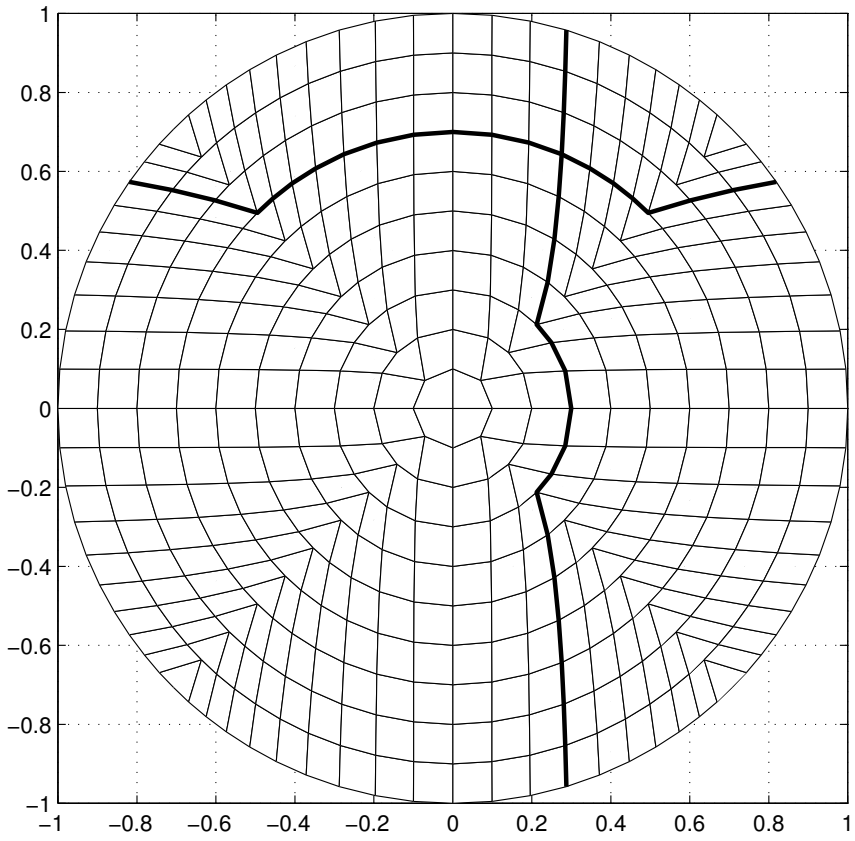


Abbildung 32: Parametrisierung des Kreisgebietes

Die sechs Rand-Segmente werden wie folgt dargestellt (Parameterbereiche $\lambda_i \in [-1, +1]$, Knoten-Indizes wie beim Referenz-Würfel):

Ost `east`
C $(1, 2, 6, 5)$
links orientiert
$x = \frac{1}{1+\lambda_1^2}$
$y = \frac{\lambda_1}{1+\lambda_1^2}$
$z = \lambda_2$

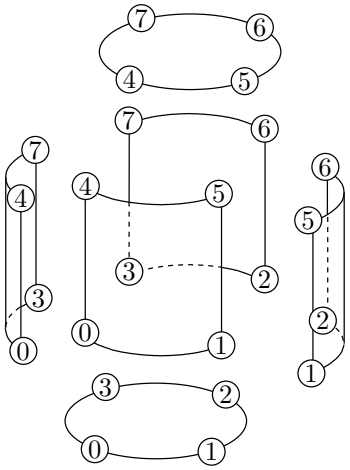
Nord `north`
C $(3, 2, 6, 7)$
rechts orientiert
$x = \frac{\lambda_1}{1+\lambda_1^2}$
$y = \frac{1}{1+\lambda_1^2}$
$z = \lambda_2$

Deckel `top`
C $(4, 5, 6, 7)$
links orientiert
$x = \lambda_1 \frac{\max(|\lambda_1|,|\lambda_2|)}{\sqrt{\lambda_1^2+\lambda_2^2}}$
$y = \lambda_2 \frac{\max(|\lambda_1|,|\lambda_2|)}{\sqrt{\lambda_1^2+\lambda_2^2}}$
$z = 1$

West `west`
C $(0, 3, 7, 4)$
rechts orientiert
$x = \frac{-1}{1+\lambda_1^2}$
$y = \frac{\lambda_1}{1+\lambda_1^2}$
$z = \lambda_2$

Süd `south`
C $(0, 1, 5, 4)$
links orientiert
$x = \frac{\lambda_1}{1+\lambda_1^2}$
$y = \frac{-1}{1+\lambda_1^2}$
$z = \lambda_2$

Boden `bot`
C $(0, 1, 2, 3)$
rechts orientiert
$x = \lambda_1 \frac{\max(|\lambda_1|,|\lambda_2|)}{\sqrt{\lambda_1^2+\lambda_2^2}}$
$y = \lambda_2 \frac{\max(|\lambda_1|,|\lambda_2|)}{\sqrt{\lambda_1^2+\lambda_2^2}}$
$z = -1$



Beim Einfügen des für das topologisch äquivalente Würfel-Gebiet „natürlichen“ Grobgitters (Unterteilung in acht kongruente Würfel-Elemente) werden die Elemente, die die Ursprungsknoten der `DOMAIN`-Definition mit Innenwinkel π enthalten, bei der Verfeinerung degeneriert. Daher wird stattdessen ein Grobgitter aus Prismen-Elementen eingefügt, das die Innenwinkel der Größe π unterteilt. Die Abb. 33 stellt beide Grobgitter und die nach der ersten Verfeinerung resultierenden Gitter gegenüber.

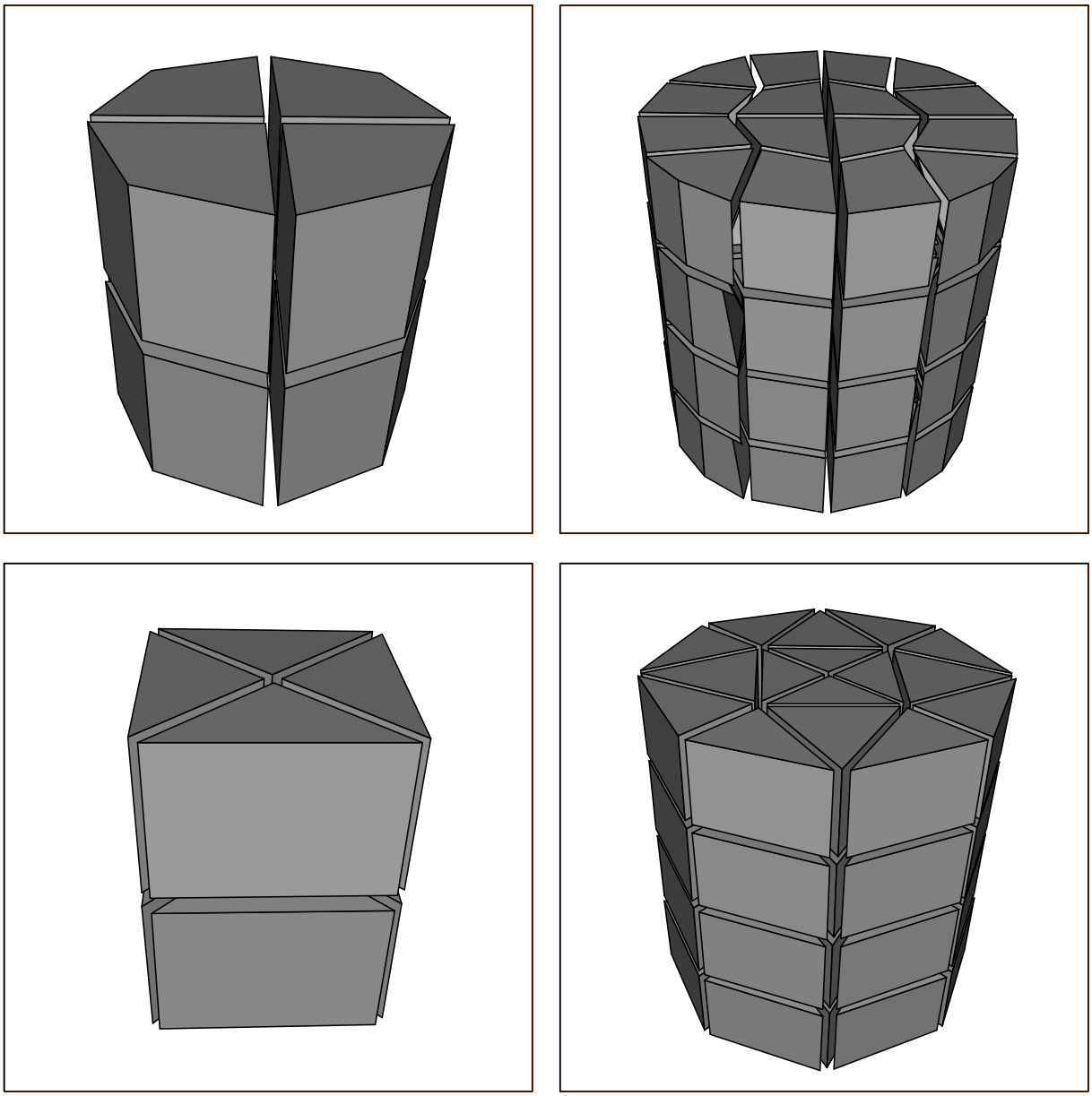


Abbildung 33: Verschiedene Grobgitter
und deren erste Verfeinerung auf Zylinder-Gebiet

Ein Beispiel für ein aus dem Prismen-Grobgitter durch viermalige lokale Verfeinerung hervorgegangenes Gitter ist in Abb. 34 dargestellt.

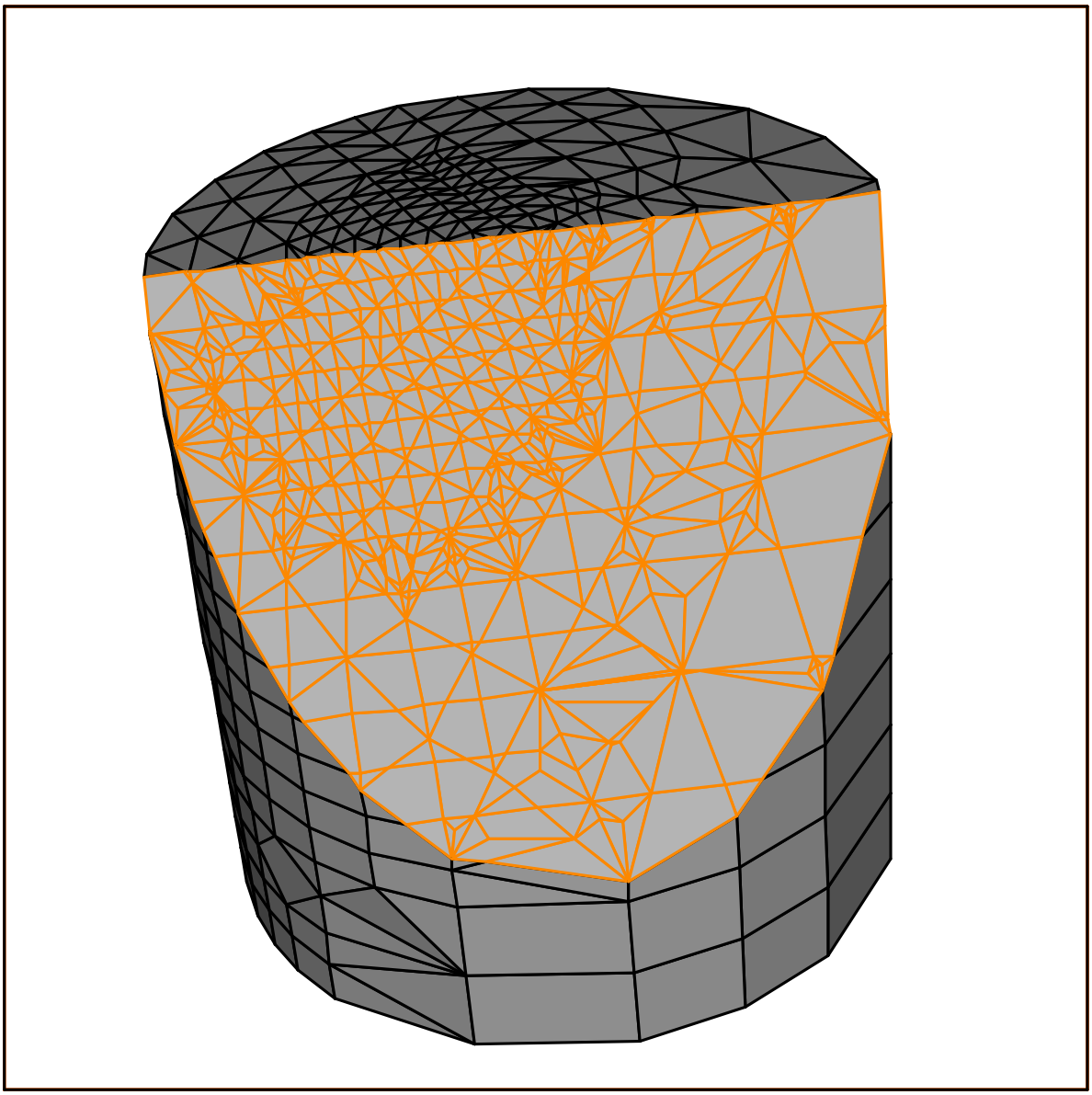


Abbildung 34: Gitter auf Zylinder-Gebiet

C.2 Geometrie der Kontrollvolumengrenzen

C.2.1 Gestalt der Kontrollvolumengrenzen in $\mathbb{R}^3$

Im Abschnitt 3.1.3 wird behauptet, daß die dort definierten Sub-Kontrollvolumengrenzen F_{ij} für $\mathsf{N} = 3$ unter bestimmten Voraussetzungen ebene Vierecke sind. Dies soll hier gezeigt werden.

Für die einzelnen Referenzelemente (siehe Abb. 4) sind die Koordinaten der vier Eckpunkte für ausgewählte Sub-Kontrollvolumengrenzen gegeben durch:

	Tetraeder	Pyramide		Prisma		Würfel
	F_{01}	F_{01}	F_{04}	F_{01}	F_{03}	F_{01}
Kantenmitte	$(\frac{1}{2}, 0, 0)$	$(\frac{1}{2}, 0, 0)$	$(0, 0, \frac{1}{2})$	$(\frac{1}{2}, 0, 0)$	$(0, 0, \frac{1}{2})$	$(\frac{1}{2}, 0, 0)$
Seitenmitte	$(\frac{1}{3}, \frac{1}{3}, 0)$	$(\frac{1}{2}, \frac{1}{2}, 0)$	$(\frac{1}{3}, 0, \frac{1}{3})$	$(\frac{1}{3}, \frac{1}{3}, 0)$	$(\frac{1}{2}, 0, \frac{1}{2})$	$(\frac{1}{2}, \frac{1}{2}, 0)$
Elementmitte	$(\frac{1}{4}, \frac{1}{4}, \frac{1}{4})$	$(\frac{3}{8}, \frac{3}{8}, \frac{1}{4})$	$(\frac{3}{8}, \frac{3}{8}, \frac{1}{4})$	$(\frac{1}{3}, \frac{1}{3}, \frac{1}{2})$	$(\frac{1}{3}, \frac{1}{3}, \frac{1}{2})$	$(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$
Seitenmitte	$(\frac{1}{3}, 0, \frac{1}{3})$	$(\frac{1}{3}, 0, \frac{1}{3})$	$(0, \frac{1}{3}, \frac{1}{3})$	$(\frac{1}{2}, 0, \frac{1}{2})$	$(0, \frac{1}{2}, \frac{1}{2})$	$(\frac{1}{2}, 0, \frac{1}{2})$
Ebene: $\underline{n}$ mit $\underline{n} \cdot \underline{x} = 1$	$(2, 1, 1)$	$(2, 0, 1)$	*n.ex.*	$(2, 1, 0)$	$(0, 0, 2)$	$(2, 0, 0)$

Die Referenzelemente lassen sich durch affin-lineare Transformationen so auf sich selbst abbilden, daß das Bild einer beliebigen Sub-Kontrollvolumengrenze F_{ij} eine der oben aufgeführten Grenzen ist.

Unter einer beliebigen (regulären) affin-linearen Transformation bleiben die vier Kanten der auf den Referenzelementen definierten Vierecke gerade Kanten, die Bilder der Ebenen, in denen die vier Eckpunkte liegen, sind eben. Somit sind im Falle durchweg affin-linearer Referenztransformationen alle Sub-Kontrollvolumengrenzen ebene Vierecke mit Ausnahme derjenigen, die den Kanten E_{i4} $(i = 0, \ldots, 3)$ eines Pyramidenelementes entsprechen.

Für den Tetraeder ist jede durch die Formfunktionen vermittelte Referenztransformation affin-linear, für die übrigen Elemente gilt dies nur in speziellen Fällen.

C.2.2 Geometrie allgemeiner Vierecke in $\mathbb{R}^3$

Bei der formalen Definition der Finite-Volumen-Gebietszerlegung tauchen Vierecke auf, deren vier Ecken ggf. nicht in einer Ebene liegen. Es stellt sich daher die Frage nach Definition und Eigenschaften solcher Vierecke.

Für die Definition eines vier beliebigen Punkten im Raum einbeschriebenen Vierecks kommen verschiedene Wege in Betracht:

- Bilineare Abbildung des Einheitsquadrates
- Minimalfläche, die einem Streckenzug aus vier geraden Kanten einbeschrieben ist
- Fläche, die gewisse Geradenscharen enthält...

Definition C.2 (Viereck in Parameterdarstellung) **:** *Gegeben seien vier Punkte* $\underline{x}_0, \ldots, \underline{x}_3 \in \mathbb{R}^3$. *Das Viereck* $F(\underline{x}_0, \underline{x}_1, \underline{x}_2, \underline{x}_3)$ *ist die Menge der Punkte, auf die das Einheitsquadrat* $[0, 1]^2$ *durch die bilineare Parameterdarstellung*

$$\begin{array}{rcl} \mathsf{T}(F) : \mathbb{R}^2 \supset [0,1]^2 & \rightarrow & \mathbb{R}^3 \\ \underline{\lambda} & \mapsto & \underline{x}(\underline{\lambda}) := \sum_{i=0}^{3} \mathsf{v}_i(\underline{\lambda}) \underline{x}_i \end{array} \tag{59}$$

mit

$$\begin{array}{ll} \mathsf{v}_0(\underline{\lambda}) = (1-\lambda_1)(1-\lambda_2) & \mathsf{v}_2(\underline{\lambda}) = \lambda_1\lambda_2 \\ \mathsf{v}_1(\underline{\lambda}) = \lambda_1(1-\lambda_2) & \mathsf{v}_3(\underline{\lambda}) = (1-\lambda_1)\lambda_2 \end{array} \tag{60}$$

abgebildet wird.

Bemerkung. Die Definition ist von der Reihenfolge der Punkte abhängig.

Bemerkung. Die v_i sind die in 3.2.2 eingeführten Formfunktionen des Einheitsquadrates.

Bemerkung. Das Viereck $F(\underline{x}_0, \underline{x}_1, \underline{x}_2, \underline{x}_3)$ enthält (mindestens) zwei Scharen von Geradenabschnitten, und zwar die Bilder der durch $\lambda_1 = const.$ bzw. $\lambda_2 = const.$ gegebenen Strecken. Insbesondere wird der Rand des Einheitsquadrates auf vier Geradenabschnitte abgebildet.

Definition C.3 (Viereck als Minimalfläche) **:** *Gegeben seien vier Punkte $\underline{x}_0, \ldots, \underline{x}_3 \in \mathbb{R}^3$. Das Viereck $F(\underline{x}_0, \underline{x}_1, \underline{x}_2, \underline{x}_3)$ ist die Fläche kleinsten Inhaltes, deren Rand durch die vier Strecken $\underline{x}_0\underline{x}_1$, $\underline{x}_1\underline{x}_2$, $\underline{x}_2\underline{x}_3$, $\underline{x}_3\underline{x}_0$ gegeben ist.*

Bemerkung. Für eine differenzierbare Parameterdarstellung $\underline{x}(\underline{\lambda})$ der Minimalfläche gilt

$$g_{11}l_{22} + g_{22}l_{11} - 2g_{12}l_{12} = 0$$

mit

$$\begin{aligned} g_{ij} &= \underline{x}_{\lambda_i} \cdot \underline{x}_{\lambda_j}, \\ l_{ij} &= \underline{n} \cdot \underline{x}_{\lambda_i\lambda_j}. \end{aligned}$$

Dies ist für die Parameterdarstellung (59) im allgemeinen nicht erfüllt, d.h. das durch (59) dargestellte Viereck stimmt nicht in jedem Fall mit der Minimalfläche überein.

Flächennormale. In jedem Punkt der Fläche ist durch die Ableitungen nach den Parametern

$$\begin{aligned} \underline{x}_{\lambda_1} &= (1-\lambda_2)(\underline{x}_1 - \underline{x}_0) + \lambda_2(\underline{x}_2 - \underline{x}_3) \\ \underline{x}_{\lambda_2} &= (1-\lambda_1)(\underline{x}_3 - \underline{x}_0) + \lambda_1(\underline{x}_2 - \underline{x}_1) \end{aligned}$$

der (orientierte) Normalenvektor

$$\underline{N}(\underline{\lambda}) := \underline{x}_{\lambda_1} \wedge \underline{x}_{\lambda_2}$$

zur Fläche F definiert. Die Einheitsnormale in $\underline{x}(\underline{\lambda})$ ist durch $\underline{n}(\lambda) := \frac{\underline{N}(\lambda)}{\|\underline{N}(\lambda)\|}$ gegeben.

Mit den Normalenvektoren in den vier Ecken $\underline{N}_i := \underline{N}(\underline{x}_i)$

$$\begin{array}{ll} \underline{N}_0 = (\underline{x}_3 - \underline{x}_0) \wedge (\underline{x}_0 - \underline{x}_1) & \underline{N}_2 = (\underline{x}_1 - \underline{x}_2) \wedge (\underline{x}_2 - \underline{x}_3) \\ \underline{N}_1 = (\underline{x}_0 - \underline{x}_1) \wedge (\underline{x}_1 - \underline{x}_2) & \underline{N}_3 = (\underline{x}_2 - \underline{x}_3) \wedge (\underline{x}_3 - \underline{x}_0) \end{array}$$

läßt sich der durch $\underline{\lambda}$ parametrisierte Normalenvektor der Fläche schreiben als

$$\underline{N}(\underline{\lambda}) = \sum_{i=0}^{3} \mathsf{v}_i(\underline{\lambda})\underline{n}_i. \tag{61}$$

Flächeninhalt. Die Fläche des Vierecks F berechnet sich allgemein zu

$$|F| = \int_F d\underline{x} = \int_{[0,1]^2} \left| \frac{\partial \underline{x}}{\partial \underline{\lambda}_1} \wedge \frac{\partial \underline{x}}{\partial \underline{\lambda}_2} \right| d\underline{\lambda} = \int_{[0,1]^2} |\underline{N}(\underline{\lambda})| d\underline{\lambda}.$$

In der ug-Implementierung (`AREA_OF_QUADRILATERAL`, `$(UGROOT)/gm/shapes.h` 165ff.) wird die Viereckfläche gemäß

$$A = \frac{|(\underline{x}_1 - \underline{x}_0) \wedge (\underline{x}_2 - \underline{x}_0)|}{2} + \frac{|(\underline{x}_3 - \underline{x}_0) \wedge (\underline{x}_2 - \underline{x}_0)|}{2}$$

berechnet. Diese Fläche entspricht der Summe der beiden Dreiecksflächen $\underline{x}_0\underline{x}_1\underline{x}_2$ und $\underline{x}_0\underline{x}_2\underline{x}_3$, die Formel ist korrekt genau für ebene konvexe Vierecke, im allgemeinen Fall liefert sie eine obere Schranke für den Flächeninhalt.

Integrationspunkt. Das Viereck F enthält den Schwerpunkt $\underline{x}_m$ seiner vier Eckpunkte als Bild von $\underline{\lambda}_m := (\frac{1}{2}, \frac{1}{2})$:

$$\underline{x}_m := \underline{x}(\underline{\lambda}_m) = \frac{1}{4} \sum_{i=0}^{3} \underline{x}_i$$

Dieser einfach ausrechenbare Punkt wird als Integrationspunkt verwendet. Für die Normale im Integrationspunkt $\underline{N}_m := \underline{N}(\underline{\lambda}_m)$ gilt

$$\underline{N}_m = \int_{[0,1]^2} \underline{N}(\underline{\lambda}) d\underline{\lambda}$$

Im Falle eines ebenen Vierecks ist die Normalenrichtung konstant, und es gilt

$$|F| = \int_{[0,1]^2} |\underline{N}(\underline{\lambda})| d\underline{\lambda} = \int_{[0,1]^2} \left| \sum_{i=0}^{3} \underline{N}_i \mathsf{v}_i(\underline{\lambda}) \right| d\underline{\lambda} = \left| \sum_{i=0}^{3} \underline{N}_i \int_{[0,1]^2} \mathsf{v}_i(\underline{\lambda}) d\underline{\lambda} \right| = \left| \frac{1}{4} \sum_{i=0}^{3} \underline{N}_i \right| = |\underline{N}_m|.$$

Regularität. Das Viereck heißt *regulär*, wenn keine drei Eckpunkte kollinear sind (d.h. insbesondere keine zwei Punkte zusammenfallen) und wenn keine der Ecken in der konvexen Hülle der anderen drei liegt.

Konvexität. Das Viereck liegt stets in der konvexen Hülle seiner Eckpunkte (die Darstellung durch Formfunktionen vermittelt eine Konvexkombination). Falls kein Punkt in der konvexen Hülle der anderen drei liegt (anderenfalls: ebenes nicht-konvexes Viereck), so liegen die vier Kanten des Vierecks auf dem Rand der konvexen Hülle der Eckpunkte und damit erst recht auf dem Rand der Viereckfläche.

Spezialfälle. *Ebenes konvexes Viereck:* Die vier Punkte liegen in einer Ebene. Die Einheitsnormale $\underline{n}(\underline{\lambda})$ ist konstant.

Ebenes Parallelogramm: Die gegenüberliegenden Seiten sind jeweils parallel und gleich lang. Die Normale $\underline{N}(\underline{\lambda})$ ist konstant. Die Referenztransformation T ist affin-linear. Die Verknüpfung der Formfunktionen mit der inversen Referenztransformation $\mathsf{v} \circ \mathsf{T}^{-1}$ ist bilinear bezüglich der durch die Kantenvektoren des Vierecks gegebenen Basis.

D Bugs in `ug-3.8`

Auf den folgenden Seiten wird eine Reihe von Unzulänglichkeiten im Quellcode der `ug`-Distribution 3.8 dokumentiert.

Die Dokumentation jedes einzelnen Fehlers ist im allg. in fünf Abschnitte gegliedert:

A Erscheinungsbild: Fehler-Symptome
B Nähere Umstände, unter denen der Fehler auftritt
C Details der Fehlerursachen
D Weiterführende Hinweise
E Änderungen im Quellcode zur Beseitigung des Fehlers

D.1 Übersicht

Programmabstürze. In gewissen Situationen stürzt die `ug`-Anwendung ab:

- es treten Schutzverletzungen *(segmentation violations)* auf, bzw.
- das Programm wird mit einer *assertion failure* beendet.

In beiden Fällen erhält der Anwender keine oder nur unzureichende Informationen über die Absturzursache (die in seiner Verantwortung liegen *kann*), nicht gespeicherte Daten sind in jedem Fall verloren.

Für einige der Abstürze konnten Ursachen im `ug`-Code ausfindig gemacht werden, diese sind in den Abschnitten D.2 und D.3 ausführlich dargestellt.

Kommandointerpreter. Eine Reihe von Fehlermeldungen des Kommandointerpreters ist irreführend: Die Meldung bezieht sich auf einen Folgefehler, der sich mittelbar aus einer von `ug` nicht korrekt behandelten Situation ergibt. Die Suche nach dem eigentlichen Fehler ist ohne Detailkenntnis der Software bzw. entsprechendes Gespür aussichtslos.

Für einige dieser Meldungen werden in D.4 Ursachen erklärt.

D.2 Segmentverletzungen

Segmentverletzungen treten bei dem Versuch des Zugriffs auf ein nicht zum Programm gehörendes Speichersegment auf. Die Auswirkungen sind unter einem modernen Betriebssystem auf den Prozeß beschränkt, in dem der Fehler auftritt. Die Anwendung wird vom Betriebssystem unmittelbar beendet.

Nicht jeder Zugriff auf eine fehlerhaft berechnete Adresse im Speicher führt zu einer vom Betriebssystem behandelten Segmentverletzung. Daher können Fehler dieser Art längere Zeit unbemerkt bleiben.

D.2.1 Absturz nach `openwindow` (X11)

A Die Beispielanwendung `simple` stürzt ab *(segmentation fault)*, wenn als erstes Kommando `openwindow ...` eingegeben wird.

B Das X11-Interface stürzt ab, wenn das Kommando `openwindow <Parameter>` manuell in die Shell eingegeben wird, und zu diesem Zeitpunkt noch kein Grafikfenster geöffnet war. Der Absturz tritt nicht auf, wenn `openwindow <Parameter>` aus einem Skript gelesen wird, in dem nach `openwindow ...` weitere Grafik-Befehle (z.B. `openpicture`) stehen.

C Die Segmentverletzung tritt in `$(UGROOT)/dev/xif/xgraph.c` (1538) auf: es wird auf `gw->backing_store` zugegriffen, während `gw==NULL` ist. Die Ursache liegt in einigen Inkonsistenzen in `$(UGROOT)/dev/xif/xgraph.c`.

In `xgraph.c` existieren drei gleichnamige Instanzen `GraphWindow * gw`:

1. (107) eine globale Instanz `GraphWindow *gw`
2. (1246) ein formaler Parameter in `int GraphOpen(GraphWindow *gw, ...)`
3. (1389) eine lokale Variable im Rumpf der Funktion `WINDOWID X11_OpenOutput(...)`

Nach den Sichtbarkeitsregeln der Sprache C wird die globale Instanz durch die lokalen Variablen verdeckt.

Die globale Instanz (107) `GraphWindow *gw` bezeichnet das aktuelle Grafikfenster, auf dem nachfolgende Grafikoperationen, insbesondere die `IFxxx()`-Funktionen (150-731), ausgeführt werden.

In der Funktion (1517ff) `X11_UpdateOutput()` wird mehrmals `gw->backing_store` (also die globale Instanz) referenziert, wobei vermutlich die in der Funktion lokal definierte Instanz `gwin` anstelle von `gw` gemeint war. Dies kann in Fällen, in denen `gw==NULL` ist (bzw. nicht initialisiert wurde), zu einer Segmentverletzung führen.

Es ist nicht klar ersichtlich, auf welchem Wege der globalen Variablen `gw` der jeweils gültige Wert zugewiesen werden soll. Die Variable wird nicht explizit initialisiert. An drei Stellen wird ihr ein Wert zugewiesen:

In (1157) `void DrawInfoBox(WINDOWID win, const char *info)` wird `gw` gesetzt, da die Variable von der später aufgerufenen Funktion `IFSetColor()` verwendet wird. Die Initialisierung der globalen Variablen scheint hier eine Nebenwirkung zu sein.

(1229) `SetCurrentGW(GraphWindow *g)` wird in `$(UGROOT)/dev/xif/xmain.c` (285) *(und nur dort!)* als Reaktion auf ein `EnterNotify`-Event (Maus tritt in `ug`-Fenster ein) aufgerufen.

(1492) `INT X11_ActivateOutput(WINDOWID win)` wird durch die Funktionen `PrepareGraph()`, `PrepareGraphWindow()` in `$(UGROOT)/graphics/uggraph/graph.c` aufgerufen. Diese Funktion wird aufgerufen durch `openpicture`, aber *nicht* durch `setcurrwindow`.

In (1394) `X11_OpenOutput()` wird *nicht* die globale Instanz, sondern die gleichnamige lokale Variable gesetzt!

In der Funktion `X11_OpenOutput()` wird zur Erzeugung eines neuen Grafikfensters eine lokale Variable (1389) `GraphWindow * gw` definiert und an die Funktion `int GraphOpen()` übergeben. Der formale Parameter heißt dort ebenfalls `GraphWindow * gw`. Beide Variablen überdecken in ihrem Gültigkeitsbereich die globale Variable (107) `GraphWindow *gw`.

In keiner der beiden Funktionen wird der globale Zeiger `gw` auf das neue Fenster gesetzt (was trotz der Namensüberdeckung mittels der Funktion `SetCurrentGW()` möglich wäre). Es ist nicht klar, ob in einer oder beiden Funktionen `gw` ursprünglich die Bedeutung der globalen Variablen hatte.

D Der Absturz wird durch den Zugriff auf das Feld `backing_store` in `GraphWindow` verursacht. Das Feld `backing_store` speichert eine Eigenschaft des aktuellen *X11-Screens* (nicht des Fensters) und wird für alle Variablen vom Typ `GraphWindow` auf denselben Wert `DoesBackingStore(DefaultScreenOfDisplay(display))` gesetzt. Insofern ist dieses Feld entbehrlich und könnte durch eine globale Variable (ähnlich `display_width`) ersetzt werden.

Die Verwendung globaler Variablen in unübersichtlichen Quelltexten ist eine potentielle Fehlerquelle, daher sollten die globalen Variablen `display`, `display_width`, etc. aus `$(UGROOT)/dev/xif/xmain.c` zusammengefaßt und auf andere Weise verwaltet werden.

E In `$(UGROOT)/dev/xif/xgraph.c` wurde folgendes geändert:

`gw->backing_store` wurde an vier Stellen durch `gwin->backing_store` ersetzt: (1176) in `DrawInfoBox(...)`; (1538, 1584, 1609) in `X11_UpdateOutput()`.

D.2.2 Absturz beim Aufruf elementarer Grafikroutinen

A Eine benutzerdefinierte Grafikroutine stürzt beim Aufruf der C-Funktion `void UgSetColor(long colorIndex)` (`$(UGROOT)/graphics/uggraph/graph.h` 103) mit einem *segmentation fault* ab.

B Die benutzerdefinierte Grafikroutine soll außerhalb der `ug`-Pictures zusätzliche „Dekoration“ anbringen und zeichnet daher unmittelbar in das Grafikfenster.

Die Segmentverletzung tritt in `$(UGROOT)/graphics/uggraph/graph.c` (1343) auf, wenn die benutzerdefinierte Grafikroutine nach `openwindow`, aber vor dem ersten `openpicture` aufgerufen wird.

C Nach dem Aufruf des Shell-Kommandos `openwindow` und vor dem ersten Aufruf von `openpicture` ist der in `graph.c` (75) global definierte Zeiger `CurrentOutputDevice` noch auf `NULL` gesetzt, obwohl ohne Zweifel ein `OUTPUTDEVICE` vorhanden ist, ohne das kein Fenster hätte geöffnet werden können. Der Aufruf irgendeiner Funktion, die diesen Zeiger implizit benutzt (z.B. `UgLine()`, `UgSetColor()`, usw.) führt zum Absturz der Anwendung.

Der Zeiger `CurrentOutputDevice` wird erst durch eine der Funktionen aus `graph.h` (80/81)

```
INT PrepareGraph (const PICTURE *thePicture)
INT PrepareGraphWindow (const UGWINDOW *theWindow)
```

gesetzt, die wiederum nur durch einige[18] der *high level drawing routines* aus `wop.h` (485-498) aufgerufen werden – aber *nicht* durch die Shell-Kommandos `setcurrwindow` bzw. `setcurrpicture`, wie es die Kommando-Namen erwarten ließen.

D Die implizite Initialisierung des Zeigers `CurrentOutputDevice` könnte als sinnvoll erachtet werden, wenn von einem unmittelbaren Aufruf der in `graph.h` enthaltenen *low level drawing functions* `UgLine()`, `UgSetColor()` etc. ausdrücklich abgeraten würde. Diese Funktionen erscheinen jedoch in der Dokumentation [11, 13], und der Anwender kann sich ermutigt fühlen, diese zu verwenden. Daher sollte in der Dokumentation auch erwähnt werden, daß vor dem Aufruf einer dieser Funktionen unbedingt eine Initialisierung mittels `PrepareGraph()` bzw. `PrepareGraphWindow()` auszuführen ist.

Die Initialisierungs-Funktionen `PrepareGraph()` und `PrepareGraphWindow()` werden in der gesamten `ug`-Dokumentation [10, 11, 12, 13] an *keiner* Stelle erwähnt!

Als „saubere“ Variante sollte die Initialisierung des Zeigers `CurrentOutputDevice` im Zuge von `openwindow` bzw. `setcurrwindow` erfolgen, wie man dies ohne Detailkenntnis erwarten würde.

[18] Es handelt sich um die in `$(UGROOT)/graphics/uggraph/wop.h` deklarierten Funktionen (485) `WorkOnPicture()`, (486) `DrawWindowText()`, (487) `DragPicture()`, (488) `ZoomPicture()`, (493) `DrawUgPicture()`, (494) `BulletDrawUgPicture()`, (497) `DrawPictureFrame()`, (498) `ErasePicture()` und nur diese.

E Vor `UgSetColor()` wird `PrepareGraph()` aufgerufen.

D.2.3 Absturz beim Lesen aus `best full refrule`

A Die `ug`-Anwendung stürzt ab *(segmentation fault)*, wenn eines der Kommandos

```
ls best full refrule/shortestie;    ls best full refrule/mra;
ls best full refrule/maxper;        ls best full refrule/maxarea;
```

in die Shell eingegeben wird.

B Das Lesen der *environment directories* schlägt aus nicht erkennbarer Ursache fehl.

C Die Deklaration `FULLREFRULE` in `$(UGROOT)/gm/rm.h` (286ff) imitiert die Struktur `ENVVAR` *(environment variable)* im Sinne einer *Vererbung*.

Instanzen von `FULLREFRULE` werden in `$(UGROOT)/gm/rm.c` (4247ff) erzeugt und in einer solchen Weise in den Environment-Baum eingefügt, daß das Environment davon ausgehen muß, daß das Speicherlayout der Strukturen mit dem Typ `ENVDIR` *(environment directory)* kompatibel ist, der aber um den Umfang eines Zeigers größer als `ENVVAR` ist.

Der mittels `GetNewEnvDirID()` bereitgestellte *ungerade* Identifizierer `theBFRRDirID` impliziert den Datentyp `ENVDIR`.

Das Environment *muß* abstürzen bei dem Versuch, den Inhalt dieser Verzeichnisse zu lesen, da der Zeiger auf die *refinement rule* sich an genau der Stelle im Speicher befindet, an der der Zeiger auf den Verzeichnisinhalt erwartet wird.

D Anstelle die `struct`-Felder von `ENVVAR` (bzw. `ENVDIR`) zu kopieren, sollte die abgeleitete Struktur `FULLREFRULE` als erstes Feld eine Variable vom Typ `ENVVAR` (`ENVDIR`) enthalten, um die „Vererbung“ robust gegenüber Änderungen der Definitionen in `$(UGROOT)/low/ugenv.h` zu machen.

E Die Zeilen (287-291) in `$(UGROOT)/gm/rm.h`

```
INT type;
INT locked;
union envitem *next;
union envitem *previous;
char name[NAMESIZE];
```

wurden durch

```
ENVDIR d;
```

ersetzt.

Da die besagten Verzeichnisse leer sind, wäre es sinnvoll, `ENVVAR` anstelle von `ENVDIR` zu verwenden. Dafür wären weitere Veränderungen in `$(UGROOT)/gm/rm.c` erforderlich.

D.2.4 Absturz nach `configure`

A Die `ug`-Anwendung stürzt ohne erkennbare Ursache ab *(segmentation fault)*.

B Es wurden mehrere Randwertprobleme installiert, oder mehr als ein Mehrgitter-Objekt mittels `new` angelegt. Das Shell-Komando `configure` wurde zuletzt für ein Randwertproblem ausgeführt, das nicht zum aktuellen Mehrgitter gehört.

C Die Datei `$(UGROOT)/dom/std/std_domain.h` enthält in (188-189) die folgende Makrodefinition:

```
#define IF_MARC(p) \
  if (PATCH_TYPE(currBVP->patches[BND_PATCH_ID(p)]) >= MARC_0_PATCH_TYPE)
```

Der in `$(UGROOT)/dom/std/std_domain.c` (116) global definierte Zeiger `STD_BVP *currBVP` bezeichnet das *aktuelle Randwertproblem*. Die Funktionen in `std_domain.c` benutzen diesen Zeiger in einer Bedeutung, die unterstellt, daß es sich um das zum momentan bearbeiteten Mehrgitter gehörende Randwertproblem handelt.

Der Zeiger `currBVP` wird unter anderem durch das Shell-Kommando `configure` unabhängig vom aktuellen Mehrgitter gesetzt. Wenn mehr als ein Randwertproblem installiert oder mehr als ein Mehrgitter-Objekt erzeugt wurde, kann somit `currBVP` auf ein Objekt zeigen, das nicht zum *aktuellen Mehrgitter* gehört.

Abhängig von der konkreten Situation (benötigte Zeiger sind `NULL`, Array-Größen stimmen ggf. nicht überein, Daten sind nicht initialisiert) können verschiedene Symptome auftreten.

D Da die verschiedenen Randwertprobleme in `ug` durch C-Code installiert werden, ist es sinnvoll, mehrere Varianten zugleich zur Verfügung zu stellen (ansonsten würde eine Modifikation des Gebietes oder der Randbedingungen eine Neuübersetzung des C-Codes erfordern). Das oben beschriebene Problem tritt nun auf, wenn *nach* der Mehrgitter-Erzeugung versehentlich das falsche Randwertproblem konfiguriert wird – an dieser Stelle fehlt eine Warnmeldung.

Der Fall, daß verschiedene Mehrgitter-Objekte zur gleichen Zeit vorhanden sind, wird, obwohl weniger naheliegend, von `ug` unterstützt und sollte ebenfalls korrekt behandelt werden.

Die Rolle der jeweils „aktuellen“ *(current)* Objekte ist klarzustellen, die Konsistenzbedingungen sind zu überprüfen.

E Das Domain-Modul ist sehr komplex, es wurde „sicherheitshalber“ nichts verändert. Der Fehler läßt sich durch sorgfältige Verwendung des Shell-Kommandos `configure` umgehen.

In den Dateien `$(UGROOT)/dom/std/std_domain.c` und `$(UGROOT)/dom/std/repair.c` gibt es 99 (neunundneunzig) weitere Referenzen auf `currBVP`!

D.2.5 Absturz bei Speichermangel

A Bei dem Versuch, ein Mehrgitter-Objekt mit einem großen Speicherbereich durch ein

Kommando wie z.B. `new myMultigrid ... $h 2000M` zu erzeugen, wird die `ug`-Anwendung mit der Meldung

```
CreateMultiGrid: cannot allocate 2097152000 bytes
Segmentation fault
```

(d.h. mit einem *segmentation fault*) beendet.

B Es wird unterstellt, daß der angeforderte Speicherplatz so groß ist, daß er vom Betriebssystem nicht bereitgestellt werden kann. In dieser Situation tritt ein *segmentation fault* auf, obwohl der Speichermangel von `ug` korrekt erkannt wird.

C Der zum Mehrgitter gehörende Heap-Speicherblock in der durch `new $h <heapSize>` angegebenen Größe wird in `$(UGROOT)/gm/ugm.c` (2966) mittels `malloc(heapSize)` vom Betriebssystem angefordert. Falls dieser Speicherblock nicht bereitgestellt werden kann, wird die gesamte im Aufbau befindliche `MULTIGRID`-Struktur `*theMG` durch den Aufruf von `DisposeMultiGrid(theMG)` (`ugm.c`, 2971) wieder zerstört.

Zu diesem Zeitpunkt ist `*theMG` noch nicht vollständig initialisiert, insbesondere sind die Zeiger `theMG->theHeap` und `theMG->grids[i]` noch `NULL`. Dies führt in mehreren der durch `DisposeBottomHeapTmpMemory(theMG)` (`ugm.c` 4000; `$(UGROOT)/gm/mgheapmgr.c` 98ff) aufgerufenen Funktionen zur Dereferenzierung eines `NULL`-Zeigers:

`PFIRSTVECTOR(theGrid)`	`$(UGROOT)/gm/algebra.c` (7873) in `DisposeIMatricesInGrid()`
`PFIRSTELEMENT(theGrid)`	`algebra.c` (1726) in `DisposeConnectionsFromMultiGrid()`
`theHeap->type`	`$(UGROOT)/low/heaps.c` (847) in `Release()`

Offensichtlich wurden bei der Einführung eines sog. *dynamic memory allocation model* notwendige Anpassungen nicht vorgenommen: `DisposeBottomHeapTmpMemory(theMG)` wird nur dann verwendet, wenn das Makro `#define DYNAMIC_MEMORY_ALLOCMODEL` gesetzt ist.

D Es ist klar, daß die Anwendung bei mangelndem Speicher nicht wie vorgesehen ausgeführt werden kann – sie sollte aber, um Datenverluste zu vermeiden und den Anwender nicht unnötig zu verunsichern, keinesfalls durch eine Segmentverletzung beendet werden. Die Voraussetzungen dafür sind in `ug` angelegt.

E Der betroffene Code wurde in dem Sinne verändert, daß der Versuch, Destruktor-Funktionen auf einem vermeintlichen Objekt an der Adresse `NULL` auszuführen, nicht als Fehler interpretiert wird (analog C++: `delete NULL` ist ungefährlich). Eingefügt wurden im einzelnen:

`algebra.c`	(1726)	`if (theGrid==NULL) continue;`
`algebra.c`	(7872)	`if (theGrid==NULL) return(0);`
`heaps.c`	(846)	`if (theHeap==NULL) return(0);`

Da der Abbruch der Mehrgittererstellung ein ernster Fehler ist, nach dem die Fortführung eines Skriptes nicht mehr sinnvoll ist, wurde außerdem in `ugm.c` (2969f) die Anweisung

```
PRINTDEBUG(gm,0,("CreateMultiGrid: cannot allocate...
```

geändert in

```
PrintErrorMessageF('E',"CreateMultiGrid", "cannot allocate...
```

D.3 Assertion failures

Eine *assertion* ist die Überprüfung einer *Invarianten* (einer Bedingung, die in einem korrekten Programm stets wahr ist) zur Laufzeit, die während der Programmentwicklung eingesetzt wird. In einem ausgereiften Programm sollte eine *assertion* niemals fehlschlagen, sie kann dann aus dem Quellcode entfernt werden (im allg. durch Entfernen von `#define DEBUG`).

Ein *assertion failure* ist mit der Ausgabe einer kurzen, fehlerabhängigen Meldung (insbesondere Quelldatei und Zeilennummer) verbunden und kann daher *dem Entwickler* einen Anhaltspunkt zur Fehlerursache liefern.

In `ug` gibt es eine Vielzahl von *assertions*, die unter gewissen Bedingungen fehlschlagen und zum Abbruch der Anwendung führen. Ein Teil dieser *assertion failures* ist durch Fehler des Anwenders bedingt: z.B. indem eine in sich inkonsistente Geometriebeschreibung gegeben wird. Andere *assertion failures* werden durch Fehler im `ug`-Programmcode ausgelöst.

Um dem Anwender die Möglichkeit zu geben, zwischen beiden Situationen zu unterscheiden, sollten erstere in reguläre `ug`-Fehlermeldungen (z.B. `PrintErrorMessage()`) umgewandelt werden. Die letzteren werden sicher im Zuge der Vervollkommnung der Software schrittweise beseitigt werden.

D.3.1 `id>0`, `ugm.c` 10558

A `Assertion failed: id>0, file ugm.c, line 10558`

B Der Fehler tritt während der Ausführung des Shell-Kommandos `fixcoarsegrid` auf.

C Eine Gebietsbeschreibung wurde angelegt, die in sich konsistent ist, bei der aber das geometrisch innere Gebiet mit dem Index `0` versehen wurde, während das äußere glücklich ist, die Nummer Eins zu sein. (Per Konvention trägt das Außengebiet immer den Index `0`.)

Oder: Ein Element wird in ein korrekt definiertes Gebiet eingefügt, wobei die Reihenfolge der Knoten derart vertauscht wurde, daß das Innere des Elementes nach außen gekehrt wird.

Gebiet und Element lassen sich ohne Beschwerden installieren, jedoch wird beim Aufruf von `fixcoarsegrid` die Anwendung mit dem oben genannten *assertion failure* beendet.

Die genannte assertion in `$(UGROOT)/gm/ugm.c` (10558) erfolgt im Zuge der Überprüfung der Konsistenz der Numerierung der Teilgebiete.

D Es gibt leider keine Möglichkeit, vor dem Aufruf von `fixcoarsegrid` die Numerierung der Teilgebiete zu überprüfen. Es wäre wünschenswert, eine gesonderte Routine für diese Überprüfung zur Verfügung zu stellen und diese dann von `FixCoarseGrid()` aufzurufen.

D.3.2 `!rc`, `fvgeom.c` 386

A `Assertion failed: !rc, file fvgeom.c, line 386`

B Der Fehler kann in Assemble-Routinen für Finite-Volumen-Verfahren auftreten.

C Die assertion wird durch das Makro `RETURN()` in `$(UGROOT)/np/algebra/fvgeom.c` (386) in der Funktion `EvaluateFVGeometry()` (`fvgeom.c` 216ff) ausgelöst. Diese Funktion stellt zur Geometrie eines gegebenen Elementes die für die Finite-Volumen-Diskretisierung benötigten Normalenvektoren, Integrationspunkte usw. bereit.

Nach der Berechnung einer Sub-Kontrollvolumengrenze und deren Normalenvektor wird überprüft, ob das Skalarprodukt des soeben berechneten Normalenvektors mit dem Vektor, der von einer Element-Ecke zur anderen gerichtet ist, negativ ist, was im Falle einer konsistenten Geometrie nicht auftreten sollte.

D Die genannte assertion wird ursprünglich eingeführt worden sein, um die korrekte Verwendung der Knoten-Indizes zu überprüfen.

Nicht geklärt ist, was hier unter einer „konsistenten Geometrie“ verstanden wird. Die Definition der Finite-Volumen-Diskretisierung setzt voraus, daß das obige Skalarprodukt stets positiv ist. Bei einer „aggressiv“ lokalen Gitterverfeinerung im dreidimensionalen Fall tritt bisweilen der oben genannte Fehler auf. Dies kann damit begründet werden, daß eine durch ihre vier Ecken gegebene Sub-Kontrollvolumengrenze (zu) stark von einer Ebene abweicht.

D.3.3 `!rc`, `wop.c` 4654

A `Assertion failed: !rc, file wop.c, line 4654`

B Der Fehler kann verschiedene Ursachen haben, er tritt u.a. in der in D.3.4 beschriebenen Situation auf.

C Der *assertion failure* wird in der Funktion `INT Draw3D(DRAWINGOBJ *q)` (Definition in `$(UGROOT)/graphics/wop.c`, 4424 ff) durch die Anweisung `RETURN(1)` ausgelöst, die zur Behandlung des Falles dient, daß ein *Instruktionsfeld* in dem Argument `DRAWINGOBJ *q` einen *ungültigen Wert* enthält.

Der Datentyp `DRAWINGOBJ *` ist nichts anderes als ein Zeiger auf ein `DOUBLE`-Array, das als eine Folge von (grafischen) Zeichenbefehlen interpretiert wird. Die erste `DOUBLE`-Zahl in jedem Befehl wird nach `char` umgewandelt und als *Instruktionsfeld* interpretiert, welches den Typ des jeweiligen Befehls bestimmt. Die Länge der einzelnen Befehle wird implizit bestimmt, das Ende der Befehlsfolge wird durch ein Instruktionsfeld mit dem Wert Null bezeichnet.

D Da keine Gesamtlänge des `DRAWINGOBJ`-Arrays übergeben wird, kann jede Verfälschung oder Fehlinterpretation der Befehlsfolge zu einer Verletzung von Speichergrenzen führen.

Das Konzept der durch `DRAWINGOBJ` repräsentierten Grafik-Sprache ist offensichtlich auf die Verwendung von Streams anstelle von Arrays zugeschnitten, bei denen die Erschöpfung der Daten erkannt wird, ohne daß die Gesamtlänge der übergebenen Daten von vornherein bekannt sein muß.

E Die *assertion* sollte durch eine reguläre `ug`-Fehlermeldung ersetzt werden, aus der hervorgeht, daß ein gewisses `DRAWINGOBJ` nicht interpretiert werden kann.

D.3.4 `plot` stürzt ab nach `setplotobject $x 1`

A `Assertion failed: !rc, file wop.c, line 4654`

B Die Option `$x 1` zum Shell-Kommando `setplotobject EScalar` bewirkt, daß die Kanten in derselben Farbe wie die Elemente dargestellt werden.

Bei Verwendung dieser Option im dreidimensionalen Fall, z.B. durch

```
setplotobject EScalar $s ... $x 1 ...;
setview ...;
plot;
```

führt die Anwendung des `plot`-Kommandos auf ein derart konfiguriertes `EScalar3D`-Plotobject zu dem oben genannten *assertion failure*.

C Der in D.3.3 untersuchte *assertion failure* weist auf ein verfälschtes `DRAWINGOBJ` hin. Eine der Ursachen findet sich in den Zeilen `$(UGROOT)/graphics/uggraph/wop.c` 12789ff:

```
if (EE3D_EdgeColor == 1)
{
  if (edgecolor != -1)
  {
    DO_2l(theDO) = edgecolor;
    DO_inc(theDO);
  }
}
else
{
  DO_2l(theDO) = EE3D_Color[COLOR_EDGE];
  DO_inc(theDO);
}
```

in der Funktion `EW_ElementEval3D_old()` (`$(UGROOT)/graphics/uggraph/wop.c` 12611ff). Diese Zeilen schreiben einen Farbwert in das gegebene `DRAWINGOBJ *theDO` und inkrementieren anschließend den Zeiger `theDO`. Im Falle `EE3D_EdgeColor==1` *und* `edgecolor==-1` (der nicht auftreten soll, aber dennoch auftritt) wird `theDO` *nicht* inkrementiert, und alle nachfolgenden Operationen schreiben auf eine um acht Bytes niedrigere Position in `*theDO`.

Das fehlerhaft erstellte `DRAWINGOBJ` *kann* nun den oben genannten *assertion failure* auslösen; wenn die an eine falsche Position geratenen Daten sich zufällig als syntaktisch korrektes `DRAWINGOBJ` interpretieren lassen, so wird die Grafik regellos deformiert sein (was in einigen Fällen auch beobachtet wurde).

Die globale Variable `EE3D_EdgeColor` wird durch die `setplotobject`-Option `$x` gesetzt, daher ist der Fehler an `setplotobject $x 1` gebunden.

Die lokale Variable `long edgecolor` wird mit dem Wert `-1` initialisiert und anschließend auf den Index der Farbe gesetzt, mit dem auch die zugehörige Fläche gefärbt wird. Diese Setzung wird für den Fall `EE3D_Property==0` *und* `EE3D_NoColor[...]!=0` (in diesem Falle wird der Grafikbefehl `DO_ERASE_SURRPOLYGON` eingefügt) *nicht* durchgeführt.

D Der oben wiedergegebene fehlerhafte Code wurde zunächst in den Zeilen (12789ff) in `wop.c` entdeckt; es gibt aber neun(!) Kopien dieser Zeilen in derselben Quelldatei:

- in der Funktion `EW_ElementEval2D()` (8481ff; hier: `EE2D_...` anstelle von `EE3D_...`)
- in `EW_ElementEval3D_old()` (Zeilen 12731ff, 12789ff, 12974ff, 13030ff)
- in `EW_ElementEval3D_new()` (Zeilen 13363ff, 13453ff, 13664ff, 13722ff)

Offensichtlich wurden diese Zeilen aus der alten in die neue Version kopiert, ohne daß der Fehler bemerkt wurde. Der Fehler wirkt sich in der Funktion `EW_ElementEval3D_new()` im Gegensatz zu `EW_ElementEval3D_old()` nicht aus, da dort die Variable `edgecolor` stets auf einen von `-1` verschiedenen Wert gesetzt wird.

Diese redundanten Zeilen (und zahlreiche weitere) sollten durch einen Funktionsaufruf oder ein Makro ersetzt werden, um den Code sicherer zu gestalten.

E Das Problem kann durch die Verwendung der Routine `EW_ElementEval3D_new()` anstelle von `EW_ElementEval3D_old()` umgangen werden, die immer dann benutzt wird, wenn für `EE3D_AmbientLight` (Shell: `setplotobject ... $a <`*value*`>`) ein Wert kleiner als Eins (also z.B. 0.9999) eingestellt wird. Der Fehler in den o.g. neunmal kopierten Quelltext-Zeilen (12789ff etc.) ist damit nicht beseitigt, dies soll den Autoren von `wop.c` überlassen bleiben.

D.3.5 `subdom>0`, `algebra.c` 583

A `Assertion failed: subdom>0, file algebra.c, line 583`

B Der *assertion failure* tritt beim Aufruf von `fixcoarsegrid` auf, wenn eine bestimmte Inkonsistenz des durch `DOMAIN` und eingefügte Elemente definierten Grobgitters vorliegt.

C Die *assertion* liegt in der Funktion `INT GetDomainPart()` (`$(UGROOT)/gm/algebra.c`, 567ff), die zu einem gegebenen geometrischen Objekt den *domain part index* zurückgibt. Die *assertion* prüft „nebenbei“ eine elementare Konsistenzbedingung: Sie wird aktiviert, wenn ein innerer Knoten (i.d.R. definiert mittels `in`) im Äußeren, d.h. in der Subdomain mit Index Null *zu liegen scheint*.

Das Makro `NSUBDOM()` liest die Subdomain-Information aus den dafür zuständigen Bits des *static control word*. Der o.g. Fehler kann also auftreten, wenn entweder der Knoten tatsächlich im falschen Teilgebiet liegt oder aber die Subdomain-Information aufgrund von Inkonsistenzen der Grobgitter-Beschreibung falsch oder überhaupt noch nicht gesetzt wurde.

Der *domain part index* wird in `SetEdgeAndNodeSubdomainFromElements(GRID *theGrid)` (`$(UGROOT)/gm/ugm.c` 10037ff) von den Elementen auf ihre Kanten und Knoten übertragen, diese Funktion wird im Verlaufe von `FixCoarseGrid()` aufgerufen. Der o.g. Fehler kann also auch dann auftreten, wenn der fragliche Knoten zwar an der richtigen Position liegt, aber zu überhaupt keinem Element gehört: Dem Knoten wurde aufgrund einer Verwechslung von Indizes kein Element zugeordnet, oder das Einfügen der Elemente schlug fehl, ohne daß dies zu einem Abbruch führte.

Eng verwandt ist der unter D.3.6 aufgeführte Fehler: ein Randknoten (i.d.R. definiert mittels `bn`) scheint auf einer Subdomain mit von Null verschiedenem Index zu liegen.

D.3.6 `(((((unsigned int *)(nd))[0]) & (((1<<(6)) ...`, `algebra.c` 591

A `Assertion failed:`
`(((((unsigned int *)(nd))[0]) & (((1<<(6)) - 1) << 3)) >> 3) == 0,`
`file algebra.c, line 591`

B Die Umstände des Auftretens der *assertion failure* entsprechen denen des zuvor (unter D.3.5) beschriebenen Fehlers.

C Die kryptische Meldung entsteht durch die ungewollte Expansion des Makros `NSUBDOM` in `ASSERT(NSUBDOM(nd) == 0)`. – Zu weiteren Einzelheiten siehe D.3.5.

E Die Meldung wird verständlich, wenn
```
ASSERT(NSUBDOM(nd) == 0);
```
durch
```
subdom = NSUBDOM(nd); ASSERT(subdom == 0);
```
ersetzt wird.

D.3.7 `!rc`, `wop.c` 10724

A `Assertion failed: !rc, file wop.c, line 10724`

B Zu einem dreidimensionalen *plot object* wurde eine Schnittebene *(cut)* mittels `setview $P <point> $N <normal-vector>` derart definiert, daß die Schnittebene eine Element*seite* berührt, das Gebiet aber durch die Schnittebene vollständig verdeckt wird.

D.4 Unverständliche Meldungen des Kommandointerpreters

D.4.1 `execute` versagt bei langen Dateinamen

A Die Eingabe des Shell-Kommandos `execute <scriptname>` führt zur Fehlermeldung: `ERROR in GetFullPathName: token too long`

B Das Shell-Kommando `execute` schlägt fehl, wenn `<scriptname>` länger als 63 Zeichen ist. Dies wird zum Problem, wenn Skripte mit absoluter Pfadangabe ausgeführt werden sollen.

C Die Funktion `GetFullPathname()` (`$(UGROOT)/ui/cmdint.c` 1652ff) kann keine Pfadnamen verarbeiten, die länger als `MAXTOKENLENGTH-1` sind.

(Genaugenommen darf *kein Token* in der Eingabezeile länger als dieser Wert sein.)

Das Makro `MAXTOKENLENGTH` (Definition in `cmdint.c` 83) ist auf den Wert 64 gesetzt.

D Das Makro `MAXTOKENLENGTH` sollte auf einen Wert gesetzt werden, der zumindest nicht kleiner als ein ggf. auf dem jeweiligen System bestehendes Limit für die Länge von Pfadnamen (z.B. `MAX_PATH`) ist. Leider gibt es kein auf allen Systemen verfügbares Makro mit dieser Bedeutung.

Die `ug`-Shell bietet keine Möglichkeit, das aktuelle Verzeichnis (im Dateisystem) zu wechseln: Das Kommando `cd` wirkt auf den `ug`-internen *„environment tree“*. Durch den Verzeichniswechsel ließen sich absolute Pfadangaben in vielen Fällen umgehen.

E Das Makro `MAXTOKENLENGTH` (`cmdint.c` 83) wird auf 128 gesetzt (bei Bedarf ist der Wert weiter zu vergrößern).

D.4.2 `setcurrwindow` versagt bei der Postscript-Ausgabe

A Mit dem Kommando `openwindow <`*`pos`*`> $d ps $n bild.eps` läßt sich ein virtuelles Grafikfenster öffnen, dessen Inhalt auf eine EPS-Datei umgeleitet wird. Der Aufruf von `setcurrwindow bild.eps` führt zu der Meldung: `ERROR in setcurrwindow: no window with this name open`.

B Der Fehler tritt auf, wenn der Name des Fensters (der zugleich Dateiname ist) einen Punkt enthält. Dies ist unangenehm, da unter den meisten Betriebssystemen eine durch einen Punkt abgetrennte Endung zur Kennzeichnung des Dateityps verwendet wird. (Die Dateinamen werden durch das PostScript-Outputdevice *nicht* mit einer Default-Endung versehen.)

C Der Punkt (ebenso wie jedes andere Zeichen außer `a-z`, `A-Z`, `0-9` und `_`) wird in der Funktion `SetCurrentWindowCommand()` (`$(UGROOT)/ui/commands.c` 8526ff) durch

```
if (sscanf(argv[0],expandfmt(CONCAT3(" setcurrwindow %",
    NAMELENSTR,"[a-zA-Z0-9_]")),winname)!=1)
```

(`commands.c` 8538) als nicht mehr zum Namen in `argv[0]` gehörende Begrenzung interpretiert.

D Aus Konsistenzgründen sollte ein Name, der von `setcurrwindow` nicht verarbeitet wird, schon von `openwindow` abgewiesen werden.

Die Unterstützung relativer oder absoluter Pfadangaben im Dateinamen wäre wünschenswert, abhängig vom Betriebssystem müssen weitere Zeichen im Namen zugelassen werden. Unter UNIX ist dies nicht ohne weiteres möglich: Ein Name, der `'/'` enthält, kann nicht im Environment-Baum eingetragen werden, da `'/'` als Verzeichnis-Separator bezüglich des Environment-Baumes interpretiert wird. Daher wäre es angebracht, den Dateinamen ggf. unabhängig vom Window-Namen setzen zu können.

E In `commands.c` (8538) wird die Formatbeschreibung `[a-zA-Z0-9_]` durch `[a-zA-Z0-9_.]` ersetzt.

D.4.3 `close` versagt für fehlerhafte Grobgitter

A Der Versuch, ein „mißratenes“ Mehrgitter durch das Kommando `close` zu löschen, führt zu der Meldung

```
> close
ERROR in close: closing the mg failed
ERROR in command execution
Error position: close
                     ^
```

die keinerlei Hinweis auf die Fehlerursache gibt.

B Das Kommando `fixcoarsegrid` wurde nicht ausgeführt, da dem Anwender bei der Gitterdefinition ein (möglicherweise nicht korrigierbarer) Fehler unterlaufen ist.

C Das Mehrgitter kann nicht *geschlossen* (d.h. aus dem Speicher entfernt) werden, bevor die Grobgitter-Definition durch das Kommando `fixcoarsegrid` *erfolgreich* abgeschlossen wurde.

D Falls es möglich ist, daß `FixCoarseGrid()` unter bestimmten Umständen nicht erfolgreich ausgeführt werden kann, und der Fehler in der Grobgitterdefinition nicht anderweitig korrigierbar ist, besteht der einzige Ausweg darin, die Anwendung zu schließen und neu zu starten. Damit wird der Sinn des Kommandos `close` untergraben.

E Da der Anwender möglicherweise nicht versteht, warum `close` fehlschlägt, sollte in der oben beschriebenen Situation eine passende Fehlermeldung erscheinen. Alternativ dazu könnte das Kommando `close` selbst `FixCoarseGrid()` aufrufen.

D.4.4 Randwertproblem wird nicht gefunden

A Die Meldung

```
ERROR in CreateMultiGrid: BVP not found
```

erscheint ohne erkennbare Ursache.

B Ein inkonsistentes Randwertproblem (z.B. mit fehlerhafter Numerierung der Randsegment-Knoten) kann ohne Probleme installiert werden, die Installation wird durch

```
BVP myBVP installed
```

bestätigt. Bei der Mehrgitter-Erzeugung mittels `new` erscheint die obige Fehlermeldung, obwohl das fragliche Randwertproblem im Environment sichtbar ist.

C Die in `$(UGROOT)/gm/ugm.c` (2985) ausgegebene Fehlermeldung ist irreführend: Das zuvor ausgeführte `BVP_Init()` gibt auch dann `NULL` zurück, wenn `theBVP` zwar gefunden wurde, aber nicht fehlerfrei initialisiert werden konnte.

E Literatur

[1] P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuss, H. Rentz-Reichert und C. Wieners. UG – a flexible software toolbox for solving partial differential equations. Available via anonymous FTP.

[2] Peter Bastian. *Parallele adaptive Mehrgitterverfahren.* Teubner-Skripten zur Numerik. Teubner-Verlag, Stuttgart, 1996.

[3] Rolf-Dieter Becher. *Untersuchung der Agglomeration von Partikeln bei der Wirbelschicht-Sprühgranulation.* Dissertation, Universität Karlsruhe, 1997. VDI Fortschrittsberichte Reihe 3 Nr. 500.

[4] Jan Blumschein. Numerische Simulation flüssigkeitsbedüster Gas-Feststoff-Wirbelschichten. Studienarbeit, Otto-von-Guericke-Universität Magdeburg, Institut für Analysis und Numerik, 2000.

[5] Dietrich Braess. *Finite Elemente – Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie.* Springer Verlag, Berlin–Heidelberg–New York, 2. Auflage, 1997.

[6] Alexandre J. Chorin und Jerrold E. Marsden. *A mathematical introduction to Fluid Mechanics*, Band 4 von *Texts in Applied Mathematics.* Springer Verlag, New York–Berlin–Heidelberg, 3. Auflage, 1992.

[7] Martin Fowler und Kendall Scott. *UML konzentriert.* Addison Wesley Longman, Bonn, 1. Auflage, 1998.

[8] Volker Gnielinski, Alfons Mersmann und Franz Thurner. *Verdampfung, Kristallisation, Trocknung.* Vieweg-Verlag, Braunschweig, 1993.

[9] W.D. Goroško, R.B. Rozenbaum und O.M. Todes. *Izvestija vyssich učebnych zavedenič. Neft' i Gaz*, 1, 125, 1958.

[10] The UG Group. *Application and Problem Class Documentation for UG*, October 27 1998. In der ug-Installation unter `$(UGROOT)/doc/applmanual.ps`.

[11] The UG Group. *UG Programming Manual*, October 27 1998. In der ug-Installation unter `$(UGROOT)/doc/progmanual.ps`.

[12] The UG Group. *UG Reference Manual, Part I*, October 27 1998. In der ug-Installation unter `$(UGROOT)/doc/refmanualI.ps`.

[13] The UG Group. *UG Reference Manual, Part II*, October 27 1998. In der ug-Installation unter `$(UGROOT)/doc/refmanualII.ps`.

[14] Wolfgang Hackbusch. *Multi-grid methods and applications.* Springer-Verlag, Berlin–Heidelberg–New York, 1. Auflage, 1985.

[15] Stefan Heinrich. *Modellierung des Wärme- und Stoffübergangs sowie der Partikelpopulationen bei der Wirbelschicht-Sprühgranulation.* Dissertation, Otto-von-Guericke-Universität Magdeburg, 2000.

[16] Peter Knabner und Lutz Angermann. *Numerik partieller Differentialgleichungen*. Springer Verlag, Berlin–Heidelberg–New York, 1. Auflage, 2000.

[17] Friedrich Löffler. *Staubabscheiden*. Georg Thieme Verlag, Stuttgart – New York, 1988.

[18] E. U. Schlünder und E. Tsotsas. *Wärmeübertragung in Festbetten, durchmischten Schüttgütern und Wirbelschichten*. Georg Thieme Verlag, Stuttgart – New York, 1988.

[19] Gerald Warnecke. *Analytische Methoden in der Theorie der Erhaltungsgleichungen*. Teubner Verlag, Stuttgart, 1. Auflage, 1999.