Tetrahedrisierung irregulärer Gitter und 3D-Helmholtz-Hodge-Zerlegung von Vektorfeldern

Alexander Wiebel

10. Oktober 2003

Inhaltsverzeichnis

1 Einleitung **1**

2 Theoretische Grundlagen **3**
2.1 Felder . . . 3
2.1.1 Vektorfelder . . . 3
2.1.2 Skalarfelder . . . 3
2.2 Punkt- und zellbasierte Felder . . . 4
2.3 Potentialfelder . . . 4
2.4 Differential-Operatoren . . . 4
2.4.1 Gradient . . . 4
2.4.2 Divergenz . . . 4
2.4.3 Rotation . . . 5
2.5 Helmholtz-Hodge-Zerlegung im kontinuierlichen Fall . . . 5

3 Numerik **6**
3.1 Methode der konjugierten Gradienten . . . 6
3.1.1 Vorbemerkungen . . . 6
3.1.2 Methode des steilsten Abstiegs . . . 7
3.1.3 Methode der konjugierten Richtungen . . . 7
3.1.4 Algorithmus . . . 8
3.1.5 Konvergenz . . . 9
3.2 Prekonditionierung . . . 9
3.3 Implementierung . . . 10
3.4 Gleichungssysteme mit nichtsymmetrischen Matrizen . . . 11

4 FAnToM **12**
4.1 DataSet . . . 12
4.2 Tensorfelder . . . 12
4.3 Irreguläre Gitter . . . 13
4.4 Unterscheidung der Algorithmen . . . 14
4.4.1 DataAlgos . . . 14

4.4.2 VisAlgos 14

5 Datensätze **15**
5.1 Bluntfin 15
5.2 Deltaflügel 15
5.3 Gasbrennkammer 16
5.4 ICE 17

6 Tetrahedrisierung irregulärer Gitter **20**
6.1 Zerlegung von beliebigen Zellen in Tetraeder 20
6.2 Implementierung 24
6.2.1 Zerlegung der Hexaeder 24
6.2.2 Zerlegung der Prismen 25
6.2.3 Zerlegung der Pyramide 25
6.2.4 Tetraedererzeugung 25

7 Diskrete Helmholtz-Hodge-Zerlegung **26**
7.1 Voraussetzungen 26
7.2 Berechnung der Potentialfelder 26
7.3 Berechnung der Komponentenfelder aus den Potentialfeldern 28
7.4 Implementierung 28
7.4.1 Behandlung der Randpunkte 28
7.4.2 Berechnung des Ausgangsfeldes 28
7.4.3 Berechnen der Potentiale 28
7.4.4 Berechnen der zellzentrierten Felder 29
7.4.5 Umrechnen der zellzentrierten Felder 29
7.4.6 Zusammengefasstes Vorgehen 30

8 Resultate **31**
8.1 Bilder 31
8.2 Lösen der Gleichungssysteme 31
8.3 Zu den Datensätzen 32

9 Ausblick **33**

A Bilder **34**

B Formel-Sammlung **40**
B.1 Formeln für die Konjugierte Gradienten Methode 40
B.2 Formeln zur Helmholtz-Hodge-Zerlegung 40

C Code-Fragmente **41**
C.1 Falltabelle 41
C.2 Auswahl der Zerlegung nach Zelltyp 43

Kapitel 1

Einleitung

Eine wichtige Teilaufgabe im Flugzeugbau ist die Strömungsanalyse. Ob beim Finden von Wirbeln, die, wenn sie auf Bauteilen auftreffen, zu Materialermüdungen oder gar Zerstörung der Bauteilen führen können, oder bei Untersuchungen zum Auftrieb bei bestimmten Flügelformen, sind strömungsmechanische Simulationen zu einem wesentlichen Teil der Entwicklungsarbeit geworden.

Um aus den Daten dieser Simulationen Schlussfolgerungen ziehen zu können, benötigen die Entwickler Hilfestellung. Diese Hilfestellung bekommen sie von Visualisierungsprogrammen an ihren Rechnern. Dort werden die Daten und insbesondere die wichtigen Merkmale graphisch aufbereitet. Es gibt viele Verfahren Strömungen im Zwei- oder Dreidimensionalen darzustellen. Die meisten arbeiten auf regulären Gittern. Dies ist jedoch ein entscheidender Nachteil, denn sehr viele Strömungssimulationen erzeugen Daten auf Gittern, die in wichtigen Regionen höher aufgelöst sind und zudem aus den unterschiedlichsten Arten von Zellen bestehen. Die Simulationen erzeugen also Daten auf irregulären Gittern. Hier beginnt nun der Bereich, in dem diese Arbeit ansetzt. Im Rahmen dieser Projektarbeit sollte ein Verfahren implementiert werden, das Strömungsdaten, die als Vektorfelder auf dreidimensionalen, irregulären Gittern vorliegen, in drei Komponenten zerlegt. Das in [8] vorgestellte Verfahren ist eine diskrete Nachahmung der Helmholtz-Hodge-Zerlegung. Die drei Komponenten stellen eine im Zweidimensionalen intuitiv verständliche Aufteilung der Vektorfelder dar. In der ersten Komponente erscheint das Vektorfeld ohne die in ihm enthaltene Drehbewegung, weshalb es auch rotationsfreier Anteil genannt wird. Dort sind nur noch Quellen und Senken vorhanden, welche so leichter erkannt werden können. Diese Quellen und Senken sind bei der zweite Komponente entfernt worden, sie heißt deshalb divergenzfreie Komponente. Ziel ist es, auch bei der zweiten Komponente die Merkmale wie Zykeln leichter zu erkennen. Die dritte Komponente ist, da sie rotations- und divergenzfrei ist, ein eher uninteressantes Feld.

Der in [8] vorgeschlagene Algorithmus funktioniert zwar auf irregulären Gittern, diese dürfen allerdings nur Tetraeder enthalten. Im Prinzip reduziert diese Beschränkung auf Tetraeder den Anwendungsbereich erheblich. Findet man allerdings eine Möglichkeit, Datensätze, deren Gitter nicht nur Tetraeder enthalten, in Datensätze mit Gittern, die nur

Tetraeder enthalten, umzuwandeln, so wäre die Beschränkung kein Hindernis mehr. Einen solchen Algorithmus zur Tetrahedrisierung irregulärer Gitter zu entwickeln und zu implementieren war der zweite Teil dieser Projektarbeit.

Beide Algorithmen sollten in dem Visualisierungssystem *FAnToM*[1] [3] realisiert werden. Betreut wurde die Arbeit von Dr. Gerik Scheuermann und seinem für *FAnToM* zuständigen Mitarbeiter Christoph Garth.

Die vorliegende Ausarbeitung beschreibt die Algorithmen und ihre Implementierung. Weiterhin werden einige theoretische Grundlagen, Informationen zum verwendeten Numerikpaket und Beschreibungen zu den zum Verständnis der Arbeit notwendigen Komponenten von *FAnToM* gegeben. Zusätzlich sind noch die zum Testen und Entwickeln der Implementierung benutzten Datensätze beschrieben.

[1] **F**ield **An**alysis using **To**pology **M**ethods

Kapitel 2

Theoretische Grundlagen

In diesem Kapitel sollen dem Leser die zum Verständnis der Arbeit benötigten mathematischen Grundlagen [2, 5] und Methoden ins Gedächtnis zurückgerufen werden. Alle Definitionen beziehen sich auf kartesische Koordinaten, da nur solche eine Rolle spielen.

2.1 Felder

Die in dieser Arbeit realisierten Verfahren arbeiten beide auf Vektorfeldern. Bei der Helmholtz-Hodge-Zerlegung ensteht in einem Zwischenschritt auch ein Skalarfeld. Deshalb sollen diese beiden Begriffe hier erklärt werden.

2.1.1 Vektorfelder

Man nennt $\vec{\mathbf{V}} = \vec{\mathbf{V}}(P)$ ein *Vektorfeld*, falls jedem Punkt $P \in R$ eines Raumteiles R ein Vektor $\vec{V} \in V$ zugeordnet wird. Die Zuordnung findet über eine Funktion $\varphi : R \to V$ statt.

Die Visualisierung beschäftigt sich nur mit endlichen diskreten Vektorfeldern. Deshalb wird bei den benutzten Vektorfeldern nicht jedem Punkt ein expliziter Wert zugewiesen, sondern nur einer endlichen Menge von Punkten. Die nicht vorgegebenen Werte können durch Interpolation erhalten werden.

2.1.2 Skalarfelder

Bei einem *Skalarfeld* $U = U(P)$ wird jedem Punkt P in einem Raumteil ein skalarer Wert U zugewiesen. Auch Skalarfelder werden in der Visualisierung nur für den diskreten Fall betrachtet. Wieder sind Werte nur an endlich vielen Stellen gegeben, nicht gegebene Werte werden interpoliert.

2.2 Punkt- und zellbasierte Felder

Skalar- und Vektorfelder wie die oben definierten nennt man *punktbasiert*. Die Helmholtz-Hodge-Zerlegung arbeitet aber nur auf *zellzentrierten* Feldern (hier auch zellbasiert genannt). Ein Feld ist zellbasiert, falls die Werte nicht an bestimmten Punkten, sondern auf bestimmten Bereichen (Zellen) gegeben sind, das heißt, alle Punkte in den Zellen den selben Wert haben. Dies gilt sowohl für Skalar- als auch für Vektorfelder.

2.3 Potentialfelder

Ein Feld heißt *Potentialfeld*, wenn der Wert eines Kurvenintegrals zwischen Punkten P_0 und P_1 nicht vom Integrationsweg zwischen P_0 und P_1 abhängt, sondern nur von der Lage der Punkte.

Ein stetiges Vektorfeld $\vec{\mathbf{V}}$ ist genau dann ein Potentialfeld, falls man $\vec{\mathbf{V}}$ als Gradient eines Potentials φ schreiben kann, das heißt als $\vec{\mathbf{V}} = \text{grad}\ \varphi$.

2.4 Differential-Operatoren

Die folgenden Definitionen von Differentialoperatoren nutzen den *Nablaoperator*

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right)$$

der die Schreibweise und das Rechnen mit den Operatoren vereinfacht.

2.4.1 Gradient

Der *Gradient* grad $U(P)$ an der Stelle $P = (x, y, z)$ eines 3D-Skalarfeldes U ist der Vektor der partiellen Ableitungen von U in P nach x, y und z. Anschaulich zeigt der Vektor in die Richtung der größten Änderung des Feldes an der Stelle P.

$$\begin{aligned} \text{grad}\ U(x,y,z) &= \nabla U(x,y,z) \\ &= \left(\frac{\partial U(x,y,z)}{\partial x}, \frac{\partial U(x,y,z)}{\partial y}, \frac{\partial U(x,y,z)}{\partial z} \right) \end{aligned}$$

2.4.2 Divergenz

Als *Divergenz* div $\vec{\mathbf{V}}$ eines Vektorfeldes $\vec{\mathbf{V}}$ bezeichnet man die Summe der partiellen Ableitungen der i-ten Komponenten des Feldes nach der i-ten Variable. In einem Strömungsfeld

stellt die Divergenz die Stoffmenge dar, die in einem Punkt pro Zeit und Volumen ensteht.

$$\begin{aligned} \text{div } \vec{\mathbf{V}} &= \nabla \cdot \vec{\mathbf{V}} \\ &= \frac{\partial \mathbf{V}_x}{\partial x} + \frac{\partial \mathbf{V}_y}{\partial y} + \frac{\partial \mathbf{V}_z}{\partial z} \end{aligned}$$

2.4.3 Rotation

Betrachtet man ein infinitesimal kleines Volumen in einem Vektorfeld, dann gibt die *Rotation* an, wie stark und um welche Achse sich das Volumen dreht. Die Rotation rot $\vec{\mathbf{V}}$ berechnet man, indem man ein Kreuzprodukt des Nablaoperators mit dem Vektorfeld bildet und diesen Ausdruck dann für die entsprechende Stelle auswertet.

$$\begin{aligned} \text{rot } \vec{\mathbf{V}} &= \nabla \times \vec{\mathbf{V}} \\ &= \left(\frac{\partial \mathbf{V}_z}{\partial y} - \frac{\partial \mathbf{V}_y}{\partial z}, \frac{\partial \mathbf{V}_x}{\partial z} - \frac{\partial \mathbf{V}_z}{\partial x}, \frac{\partial \mathbf{V}_y}{\partial x} - \frac{\partial \mathbf{V}_x}{\partial y} \right) \end{aligned}$$

2.5 Helmholtz-Hodge-Zerlegung im kontinuierlichen Fall

Für kontinuierliche Vektorfelder existiert schon seit längerer Zeit ein Verfahren, diese in „interessante“ Komponenten zu zerlegen. Die Helmholtz-Hodge-Zerlegung [1] eines Vektorfeldes ξ liefert drei Vektorfelder ∇u, $\nabla \times \mathbf{v}$ und $\mathbf{h}$.

$$\xi = \nabla u + \nabla \times \mathbf{v} + \mathbf{h}$$

Die Felder werden als *rotationsfreie*, *divergenzfreie* und *harmonische* Komponente von ξ bezeichnet, weil für das skalare Potentialfeld u, das Vektorpotentialfeld $\mathbf{v}$ und das Vektorfeld $\mathbf{h}$ folgendes gilt:

$$\begin{aligned} \nabla \times (\nabla u) &= 0 \\ \nabla \cdot (\nabla \times \mathbf{v}) &= 0 \\ \nabla \cdot \mathbf{h} &= 0 \\ \nabla \times \mathbf{h} &= 0 \end{aligned}$$

„Interessant“ ist diese Zerlegung, da zum Beispiel in 2D ∇u nur Quelle und Senken und $\nabla \times \mathbf{v}$ nur Wirbel enthält. Die harmonische Komponente ist meist ein betragsmäßig kleines oder fast konstantes Vektorfeld. Mit den Randbedingungen, dass ∇u senkrecht und $\nabla \times \mathbf{v}$ tangential zum Rand sind, ist die Zerlegung eindeutig.

Kapitel 3

Numerik

3.1 Methode der konjugierten Gradienten

Die Methode der konjugierten Gradienten [7] ist ein iteratives Verfahren zum Lösen von Gleichungssystemen der Form

$$\mathbf{A}x = b \tag{3.1}$$

bei denen $\mathbf{A}$ eine symmetrische, positiv definite Matrix ist. Solche Gleichungssysteme entstehen zum Beispiel bei der Helmholtz-Hodge-Zerlegung, die in Kapitel 7 besprochen wird, bei der Berechnung des Potentials der rotationsfreien Komponente eines Vektorfeldes und bei der Berechnung der Vektorfelder aus den Potentialen.

3.1.1 Vorbemerkungen

Eine Matrix $\mathbf{A}$ heißt *positiv definit*, falls für jeden Vektor $x \neq \vec{0}$ gilt:

$$x^T \mathbf{A} x > 0.$$

Eine *quadratische Form* ist eine skalarwertige Funktion $f : \mathbb{R}^n \to \mathbb{R}$ der Form

$$f(x) = \frac{1}{2} x^T \mathbf{A} x - b^T x + c$$

mit x und b Vektoren und einer skalaren Konstanten c. Es kann gezeigt werden, dass, falls $\mathbf{A}$ symmetrisch und positiv definit ist, die Funktion $f(x)$ durch die Lösung von Gleichung 3.1 minimiert wird.

Da $\mathbf{A}$ positiv definit ist, hat $f(x)$ ein Minimum an der Stelle x falls die Ableitung

$$\begin{aligned} f'(x) &= \nabla f \\ &= \frac{1}{2}\mathbf{A}^T x + \frac{1}{2}\mathbf{A}x - b \end{aligned} \tag{3.2}$$

dort den Wert Null annimmt. Gleichung 3.2 reduziert sich bei symmetrischem $\mathbf{A}$ auf $f'(x) = \mathbf{A}x - b$ und falls dies gleich Null gesetzt wird, ergibt sich wieder das zu lösende Gleichungssystem $\mathbf{A}x = b$. Damit ist gezeigt, dass das Minimum von $f(x)$ die Lösung von Gleichung 3.1 ist.

Diese Eigenschaft von quadratischen Formen leiten zu einer Hauptidee beim iterativen Lösen von Gleichungssystemen. Es wird versucht mit einem geeigneten Verfahren $f(x)$ zu minimieren. Ist ein minimierender Vektor x gefunden, so ist dieser die Lösung für Gleichung 3.1.

3.1.2 Methode des steilsten Abstiegs

Bei dieser Methode wird an einem beliebigen Punkt x_0 von f gestartet und versucht, in einer Reihe von Schritten $x_1, x_2, \ldots$ zum Minimum x zu gelangen. Die Schritte verlaufen immer in Richtung des steilsten Abstiegs, also entgegengesetzt der Richtung des Gradienten: $-f'(x_i) = b - \mathbf{A}x_i$.

Hier noch einige essentielle Definitionen: $e_i = x_i - x$ ist der Fehlervektor, der die Entfernung zur Lösung angibt,

$$r_i \quad = \quad b - \mathbf{A}x_i \tag{3.3}$$

ist das Residuum, das die Entfernung zum korrekten Wert von b angibt und es gilt $r_i = -\mathbf{A}e_i$.

Da $r_i = -f'(x_i)$ gilt, sind die Schritte gegeben durch

$$x_{i+1} = x_i + \alpha_i r_i \tag{3.4}$$

wobei α_i die Schrittlänge ist. Als Schrittlänge wird diejenige gewählt welche den kleinsten Wert von $f(x_{i+1})$ ergibt, das heißt, bei der $\frac{d}{d\alpha_i} f(x_{i+1}) = 0$ gilt. Wegen $\frac{d}{d\alpha_i} f(x_{i+1}) = f'(x_{i+1})^T r_i$ wird also α_i so gewählt, dass $f'(x_{i+1})$ senkrecht zu r_i ist. Einige leicht nachvollziehbare Umformungen führen von $r_{i+1}^T r_i = 0$ auf die Formel für die Schrittweite

$$\alpha_i \quad = \quad \frac{r_i^T r_i}{r_i^T \mathbf{A} r_i}. \tag{3.5}$$

Gleichung 3.3, Gleichung 3.4 und Gleichung 3.5 bilden zusammen die Methode des steilsten Abstiegs.

3.1.3 Methode der konjugierten Richtungen

Die oben erklärte Methode des steilsten Abstiegs zeigt sehr gut die Grundstrategie beim Finden von Lösungen für Gleichung 3.1, kann aber dazu führen, dass Schritte in eine schon einmal verfolgte Richtung gemacht werden müssen. Die Methode der konjugierten Richtungen umgeht dies, indem bei ihr nur Schritte gemacht werden, die orthogonal zu allen

bisherigen Schritten sind und bei denen α_i den optimalen Betrag hat. Die Lösung von Gleichung 3.1 wird damit in n (= Dimension des Gleichungssystems) Schritten erreicht. Diese Methode wird im Folgenden beschrieben, da sie nur sehr wenig verändert werden muss, um die konjugierte Gradienten-Methode darzustellen.

Die Idee, aufeinander senkrecht stehende Richtungen zu benutzen, scheitert daran, dass zur Berechnung des passenden α das Gleichungssystem schon gelöst sein müsste. Es werden statt dessen **A**-orthogonale Richtungen gewählt. Die Richtungen d_i, d_j sind **A**-*orthogonal* oder *konjugiert*, falls

$$d_i^T \mathbf{A} d_j \quad = \quad 0. \tag{3.6}$$

Werden konjugierte Richtungen gewählt, so berechnet sich die optimale Schrittweite durch:

$$\alpha_i \quad = \quad \frac{d_i^T r_i}{d_i^t \mathbf{A} d_i}. \tag{3.7}$$

Die **A**-orthogonalen Richtungen d_i erhält man durch die *Gram-Schmidt-Konjugation*. Mit einer beliebigen Menge von n linear unabhängigen Vektoren $u_0, u_1, \ldots, u_{n-1}$ liefert

$$d_i = u_i + \sum_{k=0}^{i-1} \beta_{ik} d_k \tag{3.8}$$

die Richtungen. Die Koeffizienten errechnen sich durch

$$\beta_{ij} = -\frac{u_i^T \mathbf{A} d_j}{d_j^T \mathbf{A} d_j}. \tag{3.9}$$

Diese Methode ist nicht effizient, denn offensichtlich müssen alle alten Richtungen im Speicher gehalten werden um neue zu berechnen, und das Verfahren hat eine Komplexität von $\mathcal{O}(n^3)$. Ein Algorithmus mit solcher Komplexität ist nicht praktikabel, da er keine Verbesserung gegenüber dem Gauß'schen Algorithmus darstellt.

3.1.4 Algorithmus

Die Methode der konjugierten Gradienten behebt diese Probleme indem die Richtungen hier durch Konjugation der Residuen berechnet werden. Das bedeutet, dass für die linear unabhängigen Vektoren, welche zur Gram-Schmidt-Konjugation benötigt werden, die Residuen eingesetzt werden: $u_i = r_i$. Die Residuen sind hierfür gut geeignet, da sie orthogonal zu allen vorherigen Richtungen sind und sich die Berechnung der Koeffizienten der Gram-Schmidt-Konjugation auf

$$\beta_{ij} = \begin{cases} \frac{1}{a_{i-1}} \frac{r_i^T r_i}{d_{i-1}^T \mathbf{A} d_{i-1}}, & i = j+1 \\ 0, & i > j+1 \end{cases} \tag{3.10}$$

vereinfacht und mit $\beta_i = \beta_{i,j-1}$ sogar nur noch

$$\beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i} \tag{3.11}$$

berechnet werden muss. Damit ist die Zahl der Koeffizienten sehr stark geschrumpft, die alten Richtungen werden zur Berechnung nicht gebraucht und die Komplexität der Konjugation hat sich sehr verkleinert. Zur Berechnung der Richtungen bleibt

$$d_{i+1} = r_{i+1} + \beta_{i+1} d_i. \tag{3.12}$$

Alle Gleichungen die für eine Durchführung der Methode benötigt werden sind im Anhang B.1 zusammengefasst.

3.1.5 Konvergenz

Den theoretischen Überlegungen folgend sollte die konjugierte Gradienten Methode nach n Iterationen die Lösung des Gleichungssystems gefunden haben. So war zumindest die Überlegung beim konjugierte Richtungen Verfahren. In der Praxis zeigt sich allerdings, dass durch numerische Rundungsfehler die Richtungen nicht mehr$\mathbf{A}$-orthogonal sind und auch die Residuen nicht mehr so genau sind. Dies leitet zu Überlegungen in welchen Fällen die Methode besser oder schlechter konvergiert. Ergebnis dieser Überlegungen ist, dass das Verfahren bei guter Wahl des Startpunktes oder guter Verteilung der Eigenwerte von $\mathbf{A}$ schneller konvergiert.

Die Konvergenzbetrachtungen beheben nicht nur die Nachteile der Rundungsfehler, sondern liefern sogar Möglichkeiten für Konvergenz in weit weniger als n Schritten. Eine Konvergenz in n Schritten wäre für die bei der diskreten Helmholtz-Hodge-Zerlegung entstehenden Gleichungssysteme auch viel zu langsam. Schon beim kleinsten getesteten Datensatz entstehen hier für n Werte von circa 34.000. Die Komplexität des Algorithmus verkleinert sich erheblich falls die Matrizen, wie es bei der diskreten Helmholtz-Hodge-Dekomposition glücklicherweise der Fall ist, nur dünn besetzt sind.

Bei den meisten Anwendungen wird nicht bis zur vollständigen Konvergenz gewartet, sondern eine obere Schranke für das Residuum festgelegt. Ist das Verfahren soweit fortgeschritten, dass das Residuum unter diese Grenze fällt, dann wird es abgebrochen und die aktuell angebotene Lösung akzeptiert. Auch die spart nochmal viele Iterationen ein.

3.2 Prekonditionierung

Wie im vorigen Abschnitt angedeutet hängt die Konvergenzgeschwindigkeit der konjugierten Gradienten Methode von der Matrix ab. Eine schlechte Konditionszahl und schlecht verteilte Eigenwerte verlangsamen sie stark. Um dies für ein gegebenes Gleichungssystem, welches

nun einmal eine bestimmte Matrix hat, zu verbessern, verwandelt man das Gleichungssystem in eines mit besser konditionierter Matrix, das aber das gleiche Problem löst. Bei der in Abschnitt 3.3 beschriebenen Implementierung wurde eine Cholesky-Prekonditionierung benutzt. Das Prinzip wird im Folgenden beschrieben.

Die Lösung des Gleichungssystem $\mathbf{A}x = b$ kann durch lösen von $\mathbf{M}^{-1}\mathbf{A}x = \mathbf{M}^{-1}b$ gefunden werden. Mit den durch Cholesky-Zerlegung $\mathbf{M} = \mathbf{E}\mathbf{E}^T$ gefundenen Matrizen $\mathbf{E}$ und $\mathbf{E}^T$ kann das Gleichungssystem in

$$\mathbf{E}^{-1}\mathbf{A}\mathbf{E}^{-T}x^* \quad = \quad \mathbf{E}^{-1}b$$

umgeformt werden und mit $x^* = \mathbf{E}^T x$ für x gelöst werden. Bei einer geeigneten Wahl einer symmetrischen, positiv definiten Matrix $\mathbf{M}$ ist das Problem nun in ein leichter und schneller zu lösendes[1] verwandelt worden.

3.3 Implementierung

Bei der Implementierung zur Helmholtz-Hodge-Zerlegung kam die Iterative Template Library[2] (ITL) zur Anwendung. Die ITL ist eine Sammlung von C++-Klassen, in denen iterative Methoden zum Lösen von linearen Gleichungssystemen realisiert sind.

Die ITL besitzt ein generisches Konzept, das heißt, sie kann mit verschiedenen Typen von Matrizen und verschiedenen Arten von Einträgen der Matrizen (double, float, ...) arbeiten. Für die Matrizen wurde die Matrix Template Library[3] (MTL) verwendet. Sie stellt verschiedenste Arten von Matrizen, Vektoren und Operationen auf den Matrizen zur Verfügung. Die Matrizen sind im allgemeinen so realisiert, dass sie effizient gespeichert werden und der Zugriff auf die Elemente sehr schnell erfolgen kann.

Die ITL bietet für die mit der MTL erzeugten Matrizen viele Möglichkeiten zur Prekonditionierung und nutzt einige Matrix-Operationen der MTL.

Zum Lösen der Systeme mit symmetrischen, positiv definiten Matrizen wurde die `cg`-Funktion benutzt. Als Prekonditionierer wurde ein Cholesky-Verfahren gewählt. Beides ist im Code-Beispiel Abb. 3.1 in den Zeilen `04` und `10` zu erkennen. Dem Prekonditionierer wird in Zeile `04` die Matrix übergeben und er wird in Zeile `05` aufgerufen. Der Iterator in Zeile `07` bekommt die rechte Seite `f` des Gleichungssystems, die gewünschte Höchstzahl an Iterationen `maxiter` und den maximalen Wert des Residuums `reltol` übergeben. Das Iterator Objekt ist für das Stoppen der Iteration bei Erreichen von `maxiter` oder `reltol` und die Rückmeldung an die GUI[4] zuständig. In Zeile `09` wird die Größe des Ergebnisvektors angepasst, um in Zeile `11` den Gleichungssystemlöser aufzurufen. Dieser erhält die Matrix und die rechte Seite des Gleichungssystems, den Vektor für die Lösung, den Prekonditionierer und den Iterator.

[1] Da $\mathbf{E}^{-1}\mathbf{A}\mathbf{E}^{-T}$ wieder symmetrisch und positiv definit ist, kann die Lösung wieder mit der konjugierten Gradienten Methode gefunden werden.

[2] `http://www.osl.iu.edu/research/itl/`

[3] http://www.osl.iu.edu/research/mtl/

[4] Graphical User Interface = Graphische Benutzeroberfläche

```
01  inline static void solve( Matrix& A, Vector& f, Vector& x,
02                       positive maxiter, double reltol )
03  {
04    itl::cholesky<Matrix> precond( A );
05    itl::cholesky<Matrix>::Precond p = precond();
06
07    my_iteration iter( f, maxiter, 0.0, reltol );
08
09    x.resize( f.size() );
10    itl::cg( A, x, f, p, iter );
11  }
```

Abbildung 3.1: Hilfsmethode zum Lösen von Gleichungen $\mathbf{A}x = f$

3.4 Gleichungssysteme mit nichtsymmetrischen Matrizen

Zur Lösung linearer Gleichungssysteme mit nichtsymmetrischen Matrizen wurde für die Implementierung die `bicg` Methode aus der ITL benutzt. Auf der ITL-Seite zu `bicg` wird auf [4] als Literatur verwiesen. Die Methode der bikonjugierten Gradienten soll hier nicht näher beschrieben und ebenfalls auf [4] verwiesen werden.

Kapitel 4

FAnToM

FAnToM ist ein Vektor- und Tensorfeldvisualisierungssystem, das an der Universität Kaiserlautern in der Arbeitsgruppe Visualisierung um Gerik Scheuermann entwickelt wird. Die für diese Arbeit wichtigen Aspekte von *FAnToM* werden in diesem Kapitel näher betrachtet. Diese sind im speziellen das DataSet mit seinen Komponenten. Als Quellen dienen hierzu die bei der Implementierung gesammelten Erfahrungen und [3].

4.1 DataSet

Um Datensätze verarbeiten zu können, das heißt neue Informationen aus ihnen zu gewinnen oder die in ihnen enthaltenen Informationen zu visualisieren, braucht man eine Schnittstelle, die einen strukturierten Blick und Zugriff auf die Daten erlaubt. In *FAnToM* liefert dies das DataSet. Es enthält eine Menge von Tensorfeldern. Dem DataSet können neue Felder hinzugefügt oder auch entfernt werden, und man kann von ihm aus auf die Felder zugreifen.

4.2 Tensorfelder

`FTensorField` ist die Klasse, welche alle Informationen zu einem Tensorfeld in Verbindung setzt. Sie enthält ein Gitter (`FGrid`) und eine Menge von Tensoren (`FTensorSet`), welche den Punkten auf dem Gitter zugeordnet sind. Die Tensoren in `FTensorSet` können Skalare (Ordnung 0), Vektoren (Ordnung 1), Matrizen (Ordnung 2) oder Tensoren höherer Ordnungen sein und eine beliebige Dimension besitzen.

Das Gitter besteht aus einer Menge von Punkten im Raum (`FPositionSet`) und deren Verbindung zu Zellen (`FCellDefinitions`). Es gibt sieben verschiedene Arten von Gittern. Sie können zweidimensional, dreidimensional oder zweidimensional im dreidimensionalen Raum sein. Die Punkte können in einem Rechtecksgitter, kurvilinear oder beliebig angeordnet sein. Die für diese Arbeit interessanten Gitter sind die beliebig angeordneten dreidimen-

sionalen Gitter (`FGrid3DArbitrary`), da die implementierten Algorithmen genau auf solchen arbeiten.

FPositionSet enthält die Punkte mit ihren Positionen. Die Zelldefinitionen beziehen sich nur auf diese Punkte, nicht auf ihre Lage im Raum. Zu jeder Zelle sind die Punkte gespeichert, die ihre Eckpunkte bilden. Die Reihenfolge, in der sie gespeichert sind, legt fest, welcher Punkt welche Ecke der Zelle repräsentiert. Der Aufbau von FGrid sollte anhand des Beispiel für irreguläre Gitter in Abb. 4.1 klar werden. Es ist aus [3] entnommen. Positionen

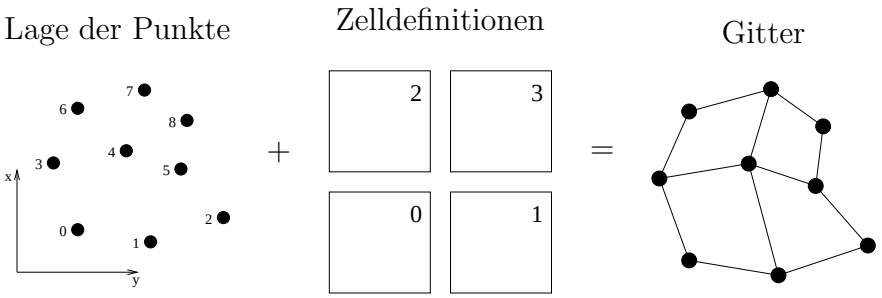


Abbildung 4.1: Verbindung von beliebig liegenden Punkte durch Zelldefinitionen zu einem irregulären Gitter

haben innerhalb eines PositionSets einen eindeutigen Index, über den sie identifiziert werden können. Zellen haben einen Index innerhalb einer Zelldefinition.

4.3 Irreguläre Gitter

Bei irregulären Gittern liegen die Punkte, wie schon erwähnt, beliebig im Raum verteilt. Es ist daher einleuchtend, dass nicht alle Punkte durch jede beliebige Art von Zellen verbunden werden können, zum Beispiel alle durch Würfel. Die Punkte werden daher oft sogar innerhalb eines Gitters durch verschiedene Zelltypen verbunden. Als Zelltypen stehen in *FAnToM* folgende zur Verfügung:

```
TRIANGLE_2D, AXIS_PARALLEL_TRI_2D, QUADRILATERAL_2D,
AXIS_PARALLEL_QUAD_2D, TRIANGLE_3D, QUADRILATERAL_3D,
TETRAHEDRON, AXIS_PARALLEL_TET, HEXAHEDRON, AXIS_PARALLEL_HEX,
PRISM, AXIS_PARALLEL_PRISM, PYRAM, BICUBICRECTANGLE_2D,
TIMETETRAHEDRON, TIMEPRISM, C1TIMEPRISM, UNDEFINED
```

Sie stellen Tetraeder, Dreiecke, Prismen[1] , Hexaeder, Vierecke und Pyramiden[2] dar, welche teils in zwei-, teils in dreidimensionaler Form existieren. Des weiteren gibt es einige Zelltypen für zeitlich veränderliche Vektorfelder.

[1] Dreiecksprismen, d.h sie haben eine dreieckige Grundseite und viereckige Seitenflächen

[2] Pyramiden mit viereckiger Grundseite

Irreguläre Gitter haben gegenüber regulären einen großen Handhabungsnachteil. Da bei einem regulären Gitter die Punkte regelmäßig angeordnet sind, kann man ihre Positionen leicht, mit einer Formel errechnen. Es ist daher sehr leicht für eine gegebene Position die Zelle zu bestimmen, in welcher sie liegt. Dies bereitet bei irregulären Gittern wesentlich größere Schwierigkeiten (siehe [6]). Weiterhin benötigen reguläre Gitter wesentlich weniger Speicherplatz, da die Positionen nicht alle explizit abgespeichert werden müssen, sondern aus einem Anfangspunkt berechenbar sind. Bei irregulären Gittern müssen alle Zellen einzeln gespeichert sein, da sie nicht unbedingt nur gleiche Zelltypen enthalten und im Voraus keine Aussage darüber getroffen werden kann, welcher Punkt zu welcher Zelle gehört.

Die in Kapitel 6 beschriebene Tetrahedrisierung arbeitet auf irregulären Gittern mit beliebigen Zellen und erzeugt daraus ein Gitter, das nur aus Tetraedern besteht. Da die Positionen bei der Tetrahedrisierung unberührt und somit beliebig verteilt bleiben, ist das entstehende Gitter obwohl es nur Tetraeder enthält, ein irreguläres Gitter.

4.4 Unterscheidung der Algorithmen

Die Algorithmen, welche *FAnToM* dem Benutzer zur Verfügung stellt, sind in zwei verschiedene Arten aufgeteilt: in *DataAlgos* und *VisAlgos*. Beide in dieser Arbeit beschriebenen Algorithmen sind DataAlgos, dies ist anhand der folgenden Definition des Begriffs und der Beschreibung der Algos leicht nachzuvollziehen. Die Unterscheidung nach den beiden Arten findet sich auch in der Menüstruktur von *FAnToM* wieder, wo beide als eigenes Menü zu finden sind.

4.4.1 DataAlgos

Algorithmen werden zu den DataAlgos gezählt, wenn sie den Datensatz, den sie bearbeiten, verändern oder besser gesagt aus dem bearbeiteten Datensatz neue Datensätze erzeugen. Der Zweck von DataAlgos ist also nicht, dem Benutzer Eigenschaften eines Datensatzes graphisch darzustellen (auch wenn einige doch eine Anzeige liefern), sondern aus der über den Datensatz vorhandenen Information neue Informationen abzuleiten. Diese neu gewonnenen Informationen könnten zum Beispiel die Rotation eines Vektorfeldes, dessen Ableitung oder das Feld selbst, allerdings geglättet, sein.

4.4.2 VisAlgos

Als VisAlgos werden alle Algorithmen bezeichnet, die den Datensatz bearbeiten, um daraus eine graphische Darstellung der Informationen zu erzeugen. Ein VisAlgo erzeugt, außer der den graphischen, keine neuen Informationen oder Daten. Als typische Beispiele für VisAlgos wären Marching Cubes, Colormapping, Stromlinien zeichnen und in *FAnToM* auch das Zeichnen der Gitterstruktur eines Datensatzes zu nennen.

Kapitel 5

Datensätze

Dieses Kapitel soll einen Überblick über die Datensätze geben, die neben einigen selbst erzeugten Datensätzen zum Test und zur Entwicklung der Implementierung der beiden Algorithmen genutzt wurden. Die Bilder und Informationen zum Deltaflügel und zur Gasbrenkammer und die Informationen zum ICE stammen aus [6]. Die Simulationen für diese Datensätze wurden bei der Deutschen Luft- und Raumfahrtgesellschaft (DLR) in Göttingen durchgeführt.

5.1 Bluntfin

Der Datensatz wurde bei der Simulation der Luftströmung um eine stumpfe Flosse (blunt fin), die aus einer Ebene heraus steht, erzeugt. Diese Flosse stellte ein Leitwerk eines Flugzeuges dar. Die ungehinderte Strömung würde parallel zur Seite des Leitwerkes verlaufen. Es wird angenommen, dass die Strömung über der Ebene symmetrisch zur Mitte des Blunt-fin verläuft. Deshalb ist im Datensatz nur die eine Hälfte der wirklichen Strömung enthalten. Das Leitwerk wurde sozusagen in der Mitte durchgeschnitten. In Abb. 5.1 ist das Gitter zu sehen. Es stellt die linke Hälfte der Strömung dar. Links in der Abbildung ist eine große Kurve zu sehen, sie zeichnet in größerem Radius den Verlauf der Vorderseite des Blunt-fin nach. Die Stelle im Vordergrund der Abbildung, die so aussieht als wäre am Rand ein Stück herausgeschnitten, ist der Teil, den das Leitwerk in der Simulation eingenommen hat. Das kurvilineare Gitter des inneren Teils des Datensatzes besteht aus 34.200 Punkten mit 31.117 Hexaederzellen.

5.2 Deltaflügel

Der Deltaflügel-Datensatz ist aus einer Simulation zur Strömung um den Flügel entstanden. Die Strömung und hier speziell die Wirbel sind für den Auftrieb interessant, den der Deltaflügel erzeugt. Für den Auftrieb ist sehr stark die Dreiecksform verantwortlich, da sie

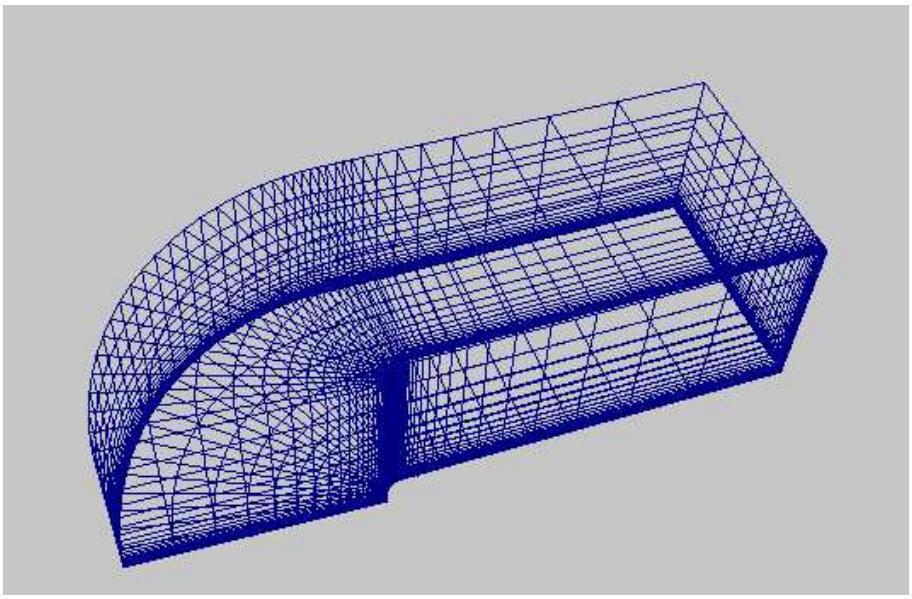


Abbildung 5.1: Gitter des inneren Teils des Bluntfin-Datensatzes

Wirbel erzeugt. Die Wirbel haben einen Anteil von 40%, das Profil trägt *nur* 60% bei. Der Flügel liegt in einem im Vergleich zu ihm sehr großen Zylindergitter. Die Strömung in diesem Gitter trifft mit etwa 25° auf die Oberfläche des Flügels. Interessant für die Tetrahedrisierung ist die Häufigkeit der vorkommenden Zelltypen. Das Gitter enthält bei 1,9 Millionen Punkten 6,3 Millionen Zellen, wovon 3,9 Millionen Tetraeder und 2,4 Millionen Prismen sind. Für die Tetrahedrisierung kommen nur die Prismen in Frage. Sie kommen nur direkt um den Flügel vor (in Abb. 5.2 leicht erkennbar).

5.3 Gasbrennkammer

Bei diesem Datensatz handelt es sich um die Simulation der Strömung in einer Verbrennungskammer für Gasheizungen. In Abb. 5.3 sind die neun Lufteinlässe oben und unten gut zu erkennen. Das Gas strömt von der linken Öffnung in die Kammer ein.

Dieser Datensatz ist für die Tetrahedrisierung nicht interessant, da er bei ca. 32.000 Punkte aus ca. 174.000 Zellen besteht, die alle bereits Tetraeder sind. Bilder zur Helmholtz-Hodge-Zerlegung des Datensatzes sind in Anhang A zu finden.

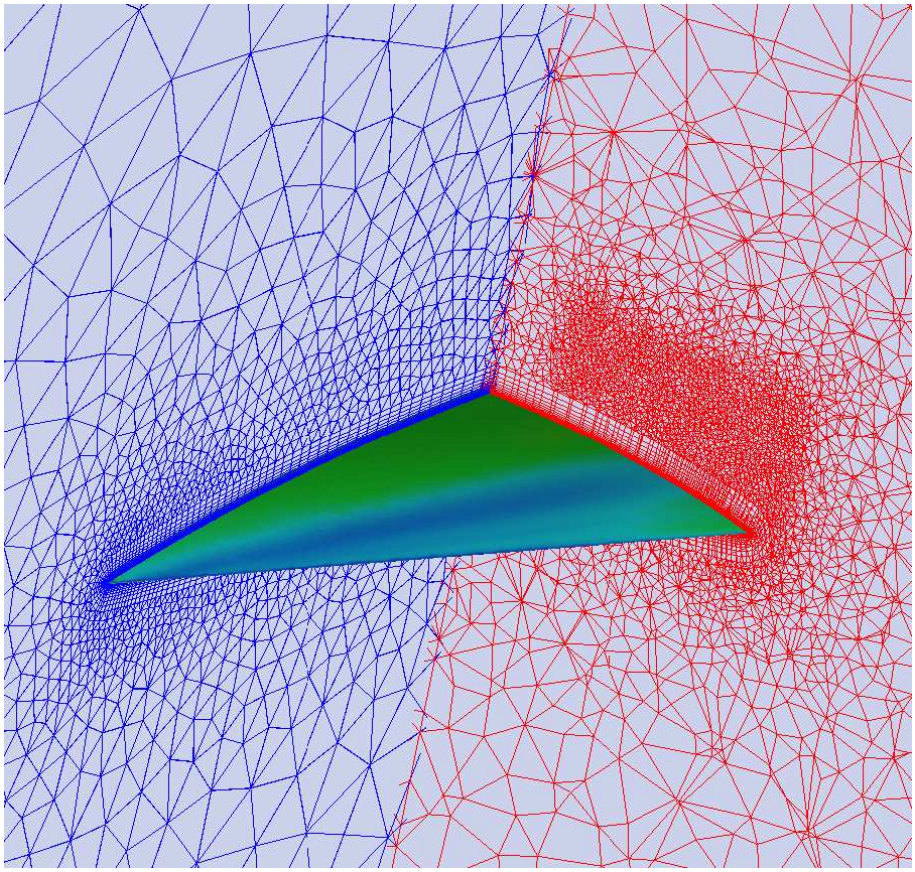


Abbildung 5.2: Deltaflügel

5.4 ICE

ICE ist wieder ein Datensatz, der durch Strömungssimulation entstanden ist. Die Strömung entsteht durch den im Verhältnis zur Geschwindigkeit des Zuges aus 15° angreifenden Wind. Der Zug ist die Lok und der erste Wagen eines deutschen ICE2. Das Gitter hat die Form eines großen Quaders, in dem sich die Wagen befinden. Es besteht aus einer Million Positionen, die 1,7 Millionen Prismen und 900.000 Tetraeder bilden. Abb. 5.4 zeigt einen Schnitt durch das Gitter und ein Colormapping des Drucks auf den Randflächen des Gitters. Man kann erkennen, dass hier, wie beim Deltaflügel, die Zellen in der Nähe des Objektes, hier also beim Zug, besser aufgelöst sind. Die Prismen kommen ebenfalls hauptsächlich in der Nähe des Zuges vor.

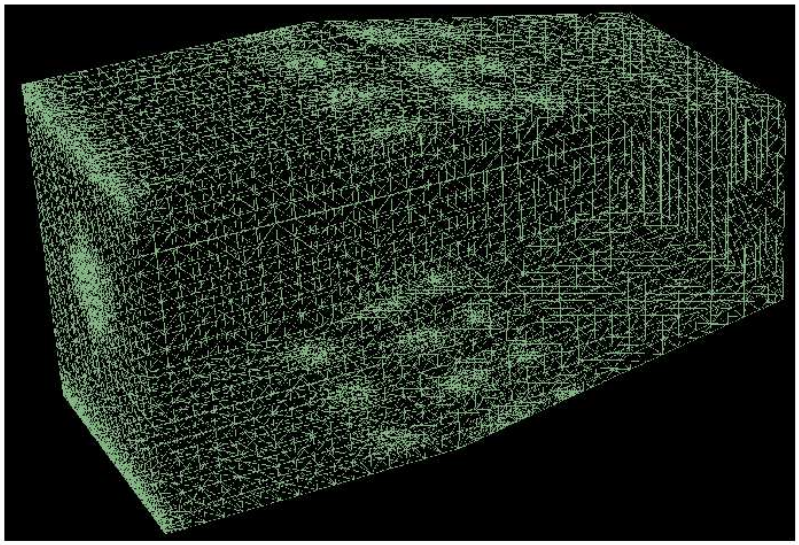


Abbildung 5.3: Gasbrennkammer mit einigen Stromlinien

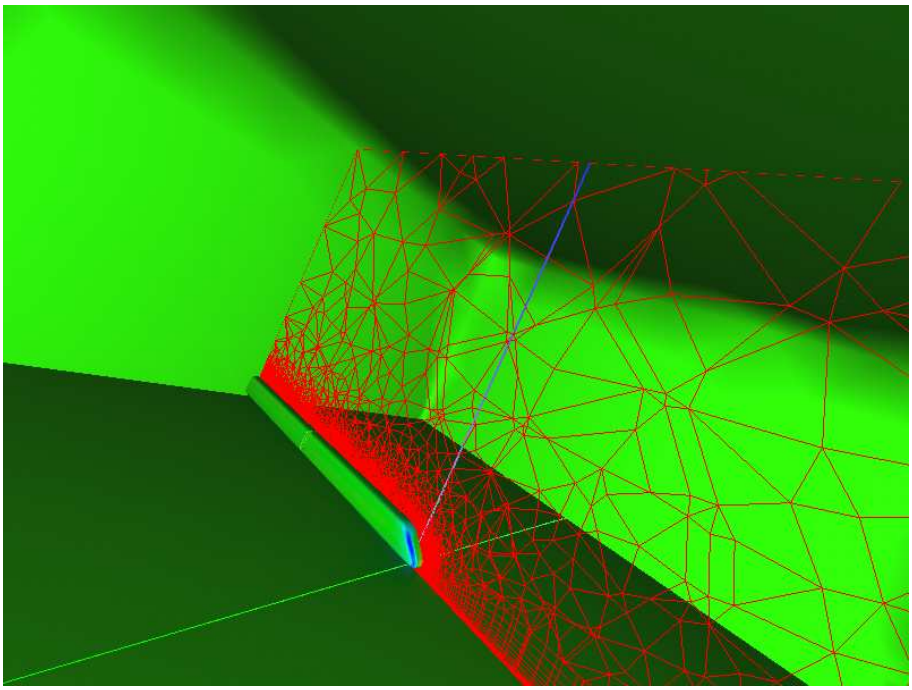


Abbildung 5.4: ICE-Zug

Kapitel 6

Tetrahedrisierung irregulärer Gitter

In diesem Kapitel wird die Strategie zur Tetrahedrisierung und die Implementierung dieser Strategie innerhalb von *FAnToM* beschrieben.

Irreguläre Gitter bestehen, wie in Kapitel 4 bereits beschrieben, aus beliebig verteilten Punkten, die Zellen verschiedener Typen bilden. Für die Tetrahedrisierung kommt von den möglichen Arten von Zellen eines irregulären Gitters nur ein Teil in Frage. Da eine Tetrahedrisierung nur in 3D sinnvoll ist, fallen alle 2D-Zellen weg. Auch die Zellen, welche für zeitabhängige Vektorfelder vorhanden sind, werden nicht betrachtet. Es bleiben also noch folgende zu behandelnde Zelltypen:

`TETRAHEDRON`	Tetraederzellen
`AXIS_PARALLEL_TET`	achsenparallele Tetraederzellen
`HEXAHEDRON`	Hexaederzellen
`AXIS_PARALLEL_HEX`	achsenparallele Hexaederzellen
`PRISM`	Prismenzellen
`AXIS_PARALLEL_PRISM`	achsenparallele Prismenzellen
`PYRAM`	Pyramidenzellen

Die jeweiligen achsenparallelen Zelltypen können, da sie für die Tetrahedrisierung nur ein Spezialfall sind, zusammen mit den beliebig geformten Zellen ihrer Art behandelt werden. Eine Tetraederzelle muss natürlich nicht verändert werden. Es bleiben also drei zu zerlegende Zelltypen (Hexaeder, Prismen und Pyramiden).

6.1 Zerlegung von beliebigen Zellen in Tetraeder

Bei der Zerlegung der Zellen in Tetraeder muss beachten werden, dass Hexaeder, Prismen und Pyramiden dreieckige und viereckige Seitenflächen besitzen können. In dem zu konstruierenden Tetraedergitter können nur dreieckige Seitenflächen auftauchen. Eine viereckige Fläche muss also in zwei dreieckige aufgeteilt werden. Dies wird leicht durch Legen einer

Diagonale von einem Eckpunkt zum gegenüberliegenden Eckpunkt der Fläche erreicht. Auf diese Weise gibt es genau zwei Möglichkeiten eine Seitenfläche zu teilen. Die Betrachtung der Lage der Diagonalen auf den viereckigen Seitenflächen der Zellen dient nun als Basis für die Zerlegungsstrategie.

Wie in Kapitel 4 beschrieben, hat jede Position eines Gitters innerhalb von *FAnToM* einen eindeutigen Index. Dieser Index wir benutzt, um die Lage der Diagonalen zu bestimmen. Es wird festgelegt, dass eine Diagonale in einer Seitenfläche immer vom Eckpunkt mit dem kleinsten Index ausgeht. Es gibt nun vier Möglichkeiten, wo der Eckpunkt mit dem kleinsten Index liegen kann. Sind die Eckpunkte wie in Abb. 6.1 links mit i, j, k und l bezeichnet, so ergeben die Fälle, in denen i oder l der Eckpunkt mit dem niedrigsten Index sind, dieselbe Diagonale. Das gleiche gilt für die Fälle mit j oder k als Ecke mit kleinstem Index.

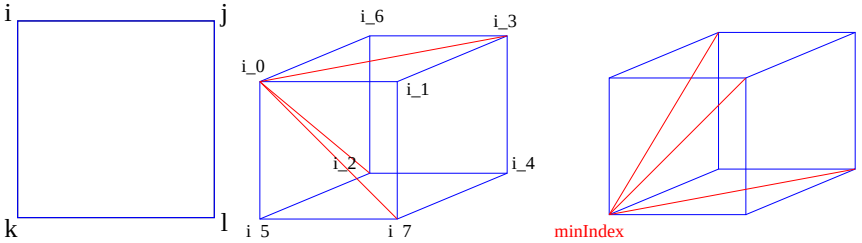


Abbildung 6.1: **links:** Seitenfläche eines Hexaeders, Prismas oder einer Pyramide mit Indizes für jeden Eckpunkt. **mitte**: Hexaeder mit Indizes für jeden Eckpunkt und vorgegebenen Diagonale für den Falls, dass i_0 der kleinste Index ist. **rechts:** Hexaeder mit vorgegebenen Diagonalen zu *minIndex* (wird im Weiteren als Beispielfall benutzt).

Da alle viereckigen Flächen auf diese Art zerlegt werden, kann es nie vorkommen, dass in zwei Zellen, welche eine Seitenfläche teilen, unterschiedliche Zerlegungen dieser Fläche entstehen. Beide Zellen haben dieselben Positionen, die diese Fläche bilden, und somit auch dieselben Indizes an den Eckpunkten, was wiederum unbedingt zur selben Diagonale führt. Die gewählte Strategie zur Zerlegung führt also zu einer konsistenten Lage der Diagonalen zueinander und somit dazu, dass später jede Seitenfläche eines Tetraeders nur *eine* Seitenfläche *eines* anderen Tetraeders berührt.

Eine Anwendung dieser Strategie auf alle Seitenflächen eines Hexaeders liefert auf den ersten Blick $64 = 2^6 = (\#\text{Diagonalen})^{\#\text{Viereckssseiten}}$ verschiedene Fälle, wie die Diagonalen angeordnet sein können. Für ein Prisma ergeben sich auf diese Weise $8 = 2^3$ Fälle. Einige dieser Fälle könnten dazu führen, dass eine Zelle nicht in Tetraeder zerlegt werden kann, ohne innere Punkte in der Zelle einzufügen und diese als Eckpunkte mit zu benutzen.

In jeder Zelle gibt es einen Eckpunkt, dessen Index am kleinsten ist. Bei dem Hexaeder[1] in Abb. 6.1 in der Mitte ist angenommen, dass i_0 der kleinste Index ist. Aus dieser Annahme

[1] Alle Abbildungen zeigen Würfel und regelmäßige Prismen, gemeint sind aber beliebige Hexaeder und beliebige Prismen

ergibt sich eindeutig die Lage der drei rot eingezeichneten Diagonalen. Diese Beobachtung ist für die Effizienz und Einfachheit der Zellzerlegungsstrategie essentiell, denn diese festgelegten Diagonalen tragen nicht mehr zu den oben genannten Fällen bei. Es wird sich zeigen, dass damit keine Fälle auftreten, die innere Punkte nötig machen.

Um die Zerlegung auf Basis der Diagonalen zu erläutern, wird diese nun an einem Beispiel durchgespielt. In diesem Beispiel wird o.B.d.A angenommen, dass sich der kleinste Index eines Hexaeders, wie in Abb. 6.1 rechts, in der vorne, links, unten liegenden Ecke befindet. Alle anderen Positionen können durch Rotation erhalten werden, das Beispiel wird dann analog funktionieren.

Es können nun zwei primäre Fälle der Lage der Diagonalen unterschieden werden. Der *erste* Fall ist in Abb. 6.2 dargestellt. Hier haben die Diagonalen zweier gegenüberliegender Seiten jeweils eine entgegengesetzte Orientierung (in einem Würfel wären sie senkrecht zueinander). In diesem Fall kann die Hexaederzelle in fünf Tetraeder zerlegt werden (siehe Abb. 6.2).

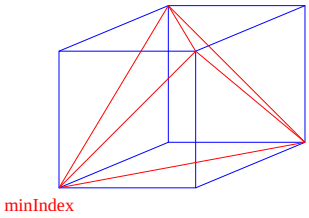


Abbildung 6.2: Alle Diagonalen von gegenüberliegenden Seiten liegen entgegengesetzt, somit ist der Hexaeder in 5 Tetraeder aufgeteilt.

Beim *zweiten* Fall gibt es mindestens ein gegenüberliegendes Seitenpaar, bei dem die Diagonalen die gleiche Orientierung haben. Die Zerlegung wird hier komplizierter. Es wir jetzt angenommen, dass zumindest die Diagonalen auf der Vorder- und Rückseite dieselbe Orientierung haben, wie in Abb. 6.3 links die gelben Linien. Der Hexaeder wir entlang dieser beiden Diagonalen in zwei Prismen unterteilt (Abb. 6.3 rechts). Sind die Diagonalen auf anderen Seitenpaaren gleich orientiert, so kann die Beispielkonfiguration wieder durch Rotation des Hexaeders erreicht werden.

Beide entstandenen Prismen haben wieder eine Ecke mit niedrigstem Index. Da die Prismen nur Positionen des Hexaeders enthalten und die Position des Hexaeders die den kleinsten Index, hat in beiden Prismen vorkommt, hat genau diese Position auch den kleinsten Index innerhalb jedes Prismas. Die entsprechende Ecke und die dadurch festgelegten Diagonalen sind in Abb. 6.4 links eingezeichnet.

Wird ein Prisma entlang dieser Diagonalen wieder in zwei Teile zerschnitten, so ergibt dies, wie in Abb. 6.4 rechts dargestellt, einen Tetraeder und eine Pyramide mit viereckiger Grundseite.

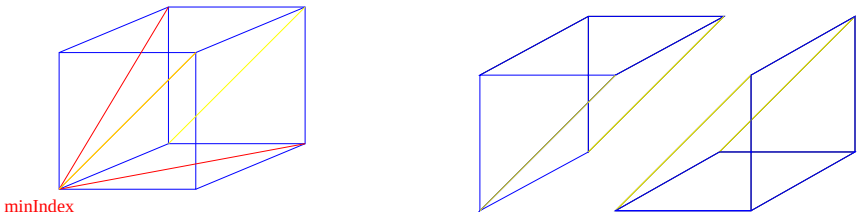


Abbildung 6.3: Ein Hexaeder mit gleich orientierten Diagonalen auf der Vorder- und Rückseite in zwei Prismen zerlegt.

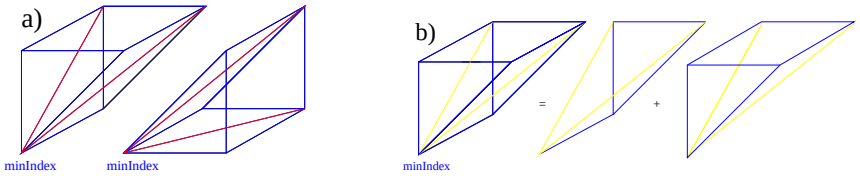


Abbildung 6.4: a) Ein Hexaeder in zwei Prismen zerlegt und die vorgegebenen Diagonalen wieder eingezeichnet. b) Ein Prisma in einen Tetraeder und eine Pyramide zerlegt.

Für den Tetraeder bleibt nichts zu tun, er kann in das Gitter aufgenommen werden. Um die Pyramide zu zerlegen wird, nach der oben beschriebenen Strategie, die Lage der Diagonalen bestimmt. In Abb. 6.5 links wurde angenommen, dass der kleinste Index der Viereckfläche vorne links oder hinten rechts liegt. Die Pyramide wird entlang der Diagonalen in zwei Tetraeder zerschnitten (Abb. 6.5 links).

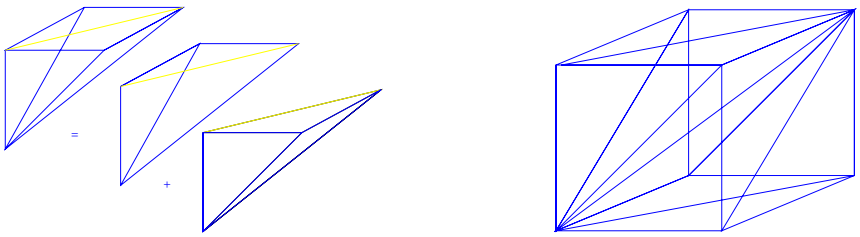


Abbildung 6.5: **links:** Eine Pyramide wird in zwei Tetraeder Zerlegt. **rechts:** Ein vollständig aufgeteilter Hexaeder.

Wird ein Hexaeder in Prismen zerlegt, diese dann jeweils in einen Tetraeder und eine Pyramide und die Pyramiden jeweils in zwei Tetraeder, so ist der Hexaeder vollständig in sechs Tetraeder zerlegt (Abb. 6.5 rechts). Die Zerlegung von Pyramiden und Prismen ist mit diesem Beispiel ebenfalls erklärt. Eine Pyramide ergibt zwei Tetraeder und ein Prisma drei Tetraeder.

Dass sich die Prismen immer in eine Pyramide und einen Tetraeder zerlegen lassen, liegt an der gleichen Beobachtung, die auch zur Reduzierung der Fälle beim Hexaeder führten: Zum minimalen Index laufen immer 2 Diagonalen (beim Hexaeder waren es drei), die damit festgelegt sind.

Die Tetrahedrisierung des gesamten Gitters wird durch eine Tetrahedrisierung aller Zellen einzeln und in der Reihenfolge ihres Indexes durchgeführt. Es ist zu erkennen, dass mit der gewählten Strategie tatsächlich keine inneren Punkte zur Tetrahedrisierung benötigt werden. Dies ist, wie schon angemerkt, für die Geschwindigkeit der Tetrahedrisierung sehr positiv, denn es müssen keine neuen Punkte in das Gitter eingefügt werden und vor allem müssen während der Tetrahedrisierung nicht wiederholt die Nachbarschaftsinformationen der Zellen abgefragt werden, was sehr viel Zeit kosten würde.

6.2 Implementierung

Als erstes wird jede Zelle des zu bearbeitenden Gitters durchgegangen, und es wird gezählt, welche Zelltypen wie oft auftreten. Anhand dieser Anzahlen kann abgeschätzt werden, wieviele Zellen nach der Tetrahedrisierung vorhanden sind. Bei Tetraedern, Prismen und Pyramiden ist die Abschätzung exakt, bei Hexaedern wird eine Zerlegung in sechs Tetraeder angenommen. Die Länge der Liste zum Speichern der Zellen ist also höchstens zu groß und kann im Nachhinein noch reduziert werden.

Wie oben bereits erwähnt, werden alle Zellen nach ihrem Index durchgegangen. Für jede Zelle wird getestet, welchen Typ sie hat, und dann die entsprechende Methode zur Zerlegung aufgerufen (vgl. Anhang C.2).

Die in den Methoden festgelegten Punkte, die zu einem Tetraeder gehören, werden zur Erzeugung desselben verwendet. Der erzeugte Tetraeder wird an die Liste mit den Tetraedern für das Gitter angehängt und, falls alle Zellen bearbeitet wurden, wird das Gitter erzeugt. Das Gitter wird mit dem TensorSet zu einem TensorField zusammengebaut und das Tensorfeld dem DataSet hinzugefügt.

6.2.1 Zerlegung der Hexaeder

Um zu erkennen, welche Diagonalenkonfiguartion vorliegt, wird zunächst kontrolliert welche Diagonalen gegenüberliegender Seiten „parallel“ sind. Sind alle nicht „parallel“, handelt es sich um einen der beiden Fälle, die in fünf Tetraeder zerlegt werden können, und die Tetraeder werden direkt erzeugt. Aus welchen Eckpunkten jeder Tetraeder dann besteht, wird nach der Falltabelle in Anhang C.1 bestimmt. Im `hexFiveTetsTable` gibt die erste Koordinate des Arrays an, welcher 5-Tetraeder-Fall vorliegt, die zweite um welchen Tetraeder es sich handelt und die dritte welcher Eckpunkt des Tetraeders gemeint ist. Die Zahlen in der Tabelle geben an, der wievielte Punkt des Hexaeders als Eckpunkt des Tetraeders zu verwenden ist.

Wird erkannt, dass ein „paralleles“ Diagonalenpaar existiert, dann liegt einer der 6-Tetraeder-Fälle vor, bei denen der Hexaeder in zwei Prismen zerlegt wird. Die Eckpunkte werden wieder aus einer Falltabelle aus Anhang C.1 ermittelt. Die erste Koordinate des `hexPrismsVertsTable` gibt an welcher 6-Tetraeder-Fall vorliegt und die zweite, für welches Prisma die Punkte, die unter Koordinate drei zu finden sind, bestimmt sind. Mit den aus dieser Tabelle ermittelten Punkte wird dann die Methode zum Zerlegen von Prismen aufgerufen.

6.2.2 Zerlegung der Prismen

Bei den Prismen wird, wie bei den Hexaedern, zuerst festgestellt, welche Diagonalenkonfiguration vorliegt. Zur Bestimmung der zu einem Hexaeder gehörenden Punkte werden hier zwei Tabellen benutzt. Der `prismTetsTable` bestimmt, welche Tetraeder zur Zerlegung benötigt werden. Mit der erhaltenen Nummer als Index liefert der `prismTetsVertsTable` die dem Tetraeder entsprechenden Eckpunkte des Prismas. Die Tetraeder werden dann direkt erzeugt und an die Liste[2] angehängt.

6.2.3 Zerlegung der Pyramide

Da es bei Pyramiden nur zwei Möglichkeiten der Tetrahedrisierung gibt, wird hier keine Tabelle benutzt. Die zu einem Tetraeder gehörigen Punkte werden innerhalb der Zerlegungsmethode bestimmt und auch gleich zur Tetraeder-Liste hinzugefügt.

6.2.4 Tetraedererzeugung

Die erzeugten Tetraeder können zwei verschiedene Orientierungen haben, das heißt, das Volumen kann positiv oder negativ sein. Um nur Tetraeder mit positiver Orientierung zu haben, wird bei der Erzeugung der Tetraeder das Volumen ausgerechnet. Falls dieses negativ ist, werden zwei Punkte miteinander vertauscht. Nach der Vertauschung hat ein vorher negativ orientierter Tetraeder ein positive Volumen.

Entstehen bei der Zerlegung eines Prismas oder einer Pyramide Tetraeder mit unterschiedlicher Orientierung, so werden diese zwar erzeugt, aber dem Benutzer wird durch Zeichnen dieser Tetraeder gezeigt, dass die zu zerlegende Zelle sehr seltsam geformt ist.

[2]Die „Liste“ ist als `vector` der C++-Standardbibliothek realisiert, soll hier aber, um nicht mit mathematischen Vektoren verwechselt zu werden, Liste genannt werden.

Kapitel 7

Diskrete Helmholtz-Hodge-Zerlegung

7.1 Voraussetzungen

Die diskrete Zerlegung, wie sie in [8] beschrieben ist, arbeitet auf einem Vektorfeld mit irregulärem 3D-Gitter. Das Gitter darf allerdings nur aus Tetraedern bestehen, und das Vektorfeld (ξ) muss zellzentriert sein. Zellzentiert heißt, dass ξ innerhalb einer Zelle konstant und durch einen Vektor ξ_k bestimmt ist.

Bei der Zerlegung sollen wie im kontinuierlichen Fall (siehe Abschnitt 2.5) zwei Potentialfelder u und $\mathbf{v}$ entstehen, deren Gradient bzw. Rotation Komponenten von ξ sind. Die Potentialfelder sind an den N Gitterpunkten definiert und in den Zellen linear interpoliert. Weil die Ableitungen in der Zelle dann konstant sind, bilden Gradient bzw. Rotation von u bzw. $\mathbf{v}$ wieder zellzentrierte Felder.

Die Potentialfelder können als $f(x) = \sum_{j=1}^{N} \varphi_j(x) f_j$ dargestellt werden, wobei die φ_j stückweise lineare Basisfunktionen sind, welche am Gitterpunkt x_j den Wert eins und an allen anderen Gitterpunkten den Wert null haben. Für die Eindeutigkeit der Zerlegung müssen die Vektorfelder, wie in Abschnitt 2.5 schon bemerkt, Randbedingungen erfüllen. Im diskreten Fall genügt es zu fordern, dass die Potentialfeldern an den Randpunkten null sind.

7.2 Berechnung der Potentialfelder

Hier wird auf eine genaue Herleitung der Gleichungssysteme zur Bestimmung der Potentialfelder verzichtet, da diese leicht in [8] nachgeschlagen werden kann. Das skalare Potentialfeld u kann durch Lösen des folgenden Gleichungssystems bestimmt werden:

$$\forall i, \sum_{T_k \in \mathcal{N}(i)} \nabla \varphi_{ik} \cdot (\nabla u)_k |T_k| \quad = \quad \sum_{T_k \in \mathcal{N}(i)} \nabla \varphi_{ik} \cdot \xi_k |T_k|$$

$\mathcal{N}(i)$ ist hierbei die Menge aller Tetraeder, die den Punkt i als Eckpunkt haben, $|T_k|$ ist das Volumen des Tetraeders T_k und φ_{ik} die auf den k-ten Tetraeder beschränkte i-te Basisfunkti-

on. Wird $(\nabla u)_k$ mit den Basisfunktionen dargestellt, so ergibt sich für die Koeffizienten der Matrix die Darstellung $\nabla\varphi_{ik} \cdot \nabla\varphi_{jk}$. Um die Gradienten der Basisfunktionen zu berechnen, ist die in [8] beschriebene geometrische Interpretation sehr hilfreich. $\nabla\varphi_{ik}$ ist ein Vektor senkrecht zur der dem Gitterpunkt i gegenüberliegenden Seitenfläche f_{ik} des Tetraeders T_k. Der Vektor zeigt in Richtung des Gitterpunktes i und hat die Länge $\frac{area(f_{ik})}{3|T_k|}$. Das Vorgehen bei der Berechnung der Gradienten der Basisfunktionen ist folgendes: Zuerst wird die Fläche bestimmt, auf welcher der Vektor senkrecht stehen soll. Dann wird durch das Kreuzprodukt zweier die Fläche begrenzender Vektoren die Richtung des Vektors bestimmt. Der Flächeninhalt der Fläche und das Volumen des Tetraeders werden ebenfalls unter Verwendung der jeweils begrenzenden Vektoren ermittelt. Aus der Fläche und dem Volumen wird dann der Betrag des Vektors berechnet. Als letztes wird der Vektor normiert und mit dem errechten Betrag auf die richtige Länge skaliert.

Zur Berechnung des Vektorpotentialfeldes $\mathbf{v}$ muss das folgende Gleichungssystem gelöst werden:

$$\forall i, \sum_{T_k \in \mathcal{N}(i)} \nabla\varphi_{ik} \times (\nabla \times \mathbf{v})_k |T_k| \quad = \quad \sum_{T_k \in \mathcal{N}(i)} \nabla\varphi_{ik} \times \xi_k |T_k|$$

Um die Koeffizienten des Gleichungssystems zu erhalten, müssen noch einige Umformungen vorgenommen werden. Zu den Umformungen gehört wieder das Darstellen des Potentialfeldes durch die Basisfunktionen.

$$\begin{aligned}
\nabla\varphi_i \times (\nabla \times \mathbf{v}) &= \nabla\varphi_i \times (\nabla \times \sum_{j=1}^{N} \varphi_j v_j) \\
&= \sum_{j=1}^{N} \nabla\varphi_i \times (\nabla\varphi_j \times v_j) \\
&= \sum_{j=1}^{N} \mathbf{A}(\nabla\varphi_i, \nabla\varphi_j) \cdot v_j
\end{aligned}$$

Die (3×3)-Matrix $\mathbf{A}(\nabla\varphi_i, \nabla\varphi_j)$ ist in Anhang B.2 nachzulesen. Es ist jetzt klarer zu erkennen, dass das Gleichungssystem einer $(3n \times 3n)$-Matrix entspricht. Dass dies so sein muss, wird auch durch die Überlegung verständlich, dass aus den drei Koordinaten von ξ wieder drei Koordinaten für jeden Punkt aus $\mathbf{v}$ erzeugt werden müssen. Die Matrix des Gleichungssystems besteht aus Blöcken, die das für die entsprechenden Koordinaten passende $\mathbf{A}(\nabla\varphi_i, \nabla\varphi_j)$ enthalten.

Es bleibt noch anzumerken, dass die Gleichungssysteme dünn besetzt sind, da sich jede Zeile nur auf einen Punkt bezieht und dieser Punkt nur zu einer kleinen Anzahl von Tetraedern gehört.

7.3 Berechnung der Komponentenfelder aus den Potentialfeldern

Die Felder ∇u und $\nabla \times \mathbf{v}$ ergeben sich direkt durch Berechnung des Gradienten bzw. der Rotation der entsprechenden Potentialfelder.

7.4 Implementierung

7.4.1 Behandlung der Randpunkte

Zur Bestimmung der Randpunkte wird jede Zelle des Datensatzes durchlaufen. Für jede Zelle werden die Randflächen festgestellt und über diese die Nachbarn der Zelle bestimmt. Falls eine Zelle an einer Seite keinen Nachbarn hat, so liefert die Nachbarnsuche einen nicht gültigen Index. Hat eine Zelle auf einigen Seiten keine Nachbarn, so sind alle Punkte der entsprechenden Flächen Randpunkte.

Wenn bei der Überprüfung auf Nachbarn also ein nicht gültiger Index auftaucht, so werden alle Punkte der entsprechenden Fläche im `is_boundary` Array auf `true` gesetzt.

Bei der Berechnung der Potentiale werden die Randpunkte vollständig ignoriert. Die Gleichungssysteme werden nur für innere Punkte aufgestellt und aufgelöst. Die Randpunkte werden nach den Berechnungen mit dem Wert null wieder einsortiert und zusammen mit den inneren Punkten dem TensorSet hinzugefügt.

7.4.2 Berechnung des Ausgangsfeldes

Alle Vektorfelder in *FAnToM* sind auf den Gitterpunkten definiert. Da die Zerlegung aber zellzentrierte Felder als Eingabe braucht, müssen die Vektorfelder umgerechnet werden. Zur Zeit wird die Umrechnung durch eine Berechnung des Vektors am Schwerpunkt der Zelle gemacht, das heißt, die Vektoren an den Eckpunkten der Zelle werden arithmetisch gemittelt. Der erhaltene Vektor ξ_k gilt dann für die ganze Zelle.

7.4.3 Berechnen der Potentiale

Das oben beschriebene Verfahren zum Aufstellen der Gleichungssysteme wäre in *FAnToM* sehr langsam. In jedem Schritt, also für jeden einzelnen Punkt, müsste die Nachbarschaftsinformation ausgewertet werden, um alle Nachbartetraeder zu finden. Um dies zu umgehen, wird zellweise vorgegangen.

Bei der punktweisen Vorgehensweise wird für jeden Punkt der Beitrag jedes angrenzenden Tetraeders berechnet und die Beiträge werden addiert. Wird zellweise vorgegangen, so kann für jeden Tetraeder der Beitrag, den er zu jedem seiner Eckpunkte liefert ausgerechnet

werden. Falls der Wert jedes Punktes am Anfang auf null gesetzt wurde und bei der Behandlung jedes Tetraeders dessen Beitrag zu den Werten seiner Eckpunkte addiert wird, ist die Addition der Anteile einfach nur über die Zeit verteilt worden. Die Punkte erhalten also bei beiden Vorgehensweisen, über den gesamten Durchlauf gesehen, dieselben Beiträge und haben somit am Ende den gleichen Wert.

7.4.4 Berechnen der zellzentrierten Felder

Das Berechnen der Komponenten des Ausgangsfeldes aus den Potentialen ist dank der von *FAnToM* bereitgestellten Infrastruktur relativ leicht zu realisieren. *FAnToM* stellt für Tetraederzellen eine Funktion zur Berechnung von Ableitungen in der Zelle zu Verfügung. Beim skalaren Potentialfeld ist die bereitgestellte Ableitung schon der Gradient und kann direkt übernommen werden. Die Ableitungsfunktion liefert für eine Zelle im Vektorpotential eine Matrix der partiellen Ableitungen. Die Einträge dieser Matrix können dann zur Berechnung der diskreten Rotation verwendet werden.

7.4.5 Umrechnen der zellzentrierten Felder

Wie schon erwähnt, kann *FAnToM* keine zellzentrierten Felder behandeln. Eine Umrechnung in punktbasierte Felder ist nötig. Die Umrechnung wird hier mit einem Kleinste-Quadrate-Ansatz durchgeführt.

Zur Berechnung des zellzentrierten Feldes $\mathbf{c}$ aus den Werten an den Gitterpunkten $\mathbf{p}$ kann eine Matrix $\mathbf{B}$ aufgestellt werden, so dass $\mathbf{Bp} = \mathbf{c}$ gilt. Die Matrix hat als Dimensionen die Anzahl der Punkte N und die Anzahl der Zellen M. Sie enthält einen Eintrag an der Stelle (n, m), falls der Punkt n Eckpunkt des Tetraeders m ist. Alle Einträge der Matrix, die nicht null sind, haben, da der Wert für eine Zelle aus den vier Eckpunkten gemittelt wird, den Wert $\frac{1}{4}$.

Das Gleichungssystem $\mathbf{Bp} = \mathbf{c}$ wird im Kleinste-Quadrate-Sinn nach $\mathbf{p}$ aufgelöst, um das punktbasierte Feld zu errechnen. Lösen im Kleinste-Quadrate-Sinn bedeutet das Gleichungssystem

$$\mathbf{B}^T\mathbf{Bp} = \mathbf{B}^T\mathbf{c}$$

zu lösen.

Da die Multiplikationen $\mathbf{B}^T\mathbf{B}$ und $\mathbf{B}^T\mathbf{c}$ sehr aufwändig sind, werden sie nicht explizit durchgeführt, sondern beim Durchlaufen aller Zellen Eintrag für Eintrag befüllt. $\mathbf{c}$ wird befüllt, indem für alle vier Punkte i, j, k, l eines Tetraeders zu c_i, c_j, c_k und c_l jeweils $\frac{1}{4} \cdot b_k$ addiert wird. $\mathbf{E} = \mathbf{B}^T\mathbf{B}$ wird befüllt, indem für alle vier Punkte eines Tetraeders zu $\mathbf{E}_{ab}$ jeweils $\frac{1}{16}$ addiert wird ($a, b \in \{i, j, k, l\}$).

Diese Vorgehensweise leitet sich wie folgt her:

$$\begin{aligned}
(\mathbf{B}^T\mathbf{B})_{ab} &= \sum_{k=1}^{M} \mathbf{B}^T_{ak}\mathbf{B}_{kb} \\
&= \sum_{k=1}^{M} \mathbf{B}_{ka}\mathbf{B}_{kb} \\
\mathbf{B}_{ka} &= \begin{cases} \frac{1}{4} & \text{falls Zelle } k \text{ Eckpunkt } a \text{ hat} \\ 0 & \text{sonst} \end{cases} \\
\mathbf{B}_{ka}\mathbf{B}_{kb} &= \begin{cases} \frac{1}{16} & \text{falls Zelle } k \text{ Eckpunkte } a \text{ und } b \text{ hat} \\ 0 & \text{sonst} \end{cases}
\end{aligned}$$

Es ist zu erkennen, dass nur bei den oben erwähnten Fällen Beiträge ungleich null für einen Matrixeintrag entstehen. Die Befüllung von $\mathbf{c}$ erschließt sich in ähnlicher Weise.

7.4.6 Zusammengefasstes Vorgehen

Als erstes werden die Randpunkte bestimmt und aus der Liste der zu berechnenden Punkte entfernt. Dann werden in einer Schleife alle Zellen durchlaufen. In der Schleife werden alle bis jetzt erwähnten Matrizen befüllt. Zuerst die, welche zur Umrechnung der zellzentrierten Felder benötigt werden. Danach werden die Tensoren der Zelle gemittelt und die benötigten Normalen aus den Seitenflächen berechnet. Im Folgenden werden das Volumen des Tetraeders und die Größe der Seitenfläche berechnet. Aus diesen vier Größen werden dann die $\nabla\varphi_i$ berechnet. Die Matrizen und rechten Seiten zur Ermittelung der Potentiale werden nun mittels des Volumens und der $\nabla\varphi_i$ befüllt.

Nach Beendigung der Schleife werden die Randpunkte wieder einsortiert, sowie die Potentialfelder errechnet und dem DataSet hinzugefügt. Die Gleichungssysteme werden mit den in Kapitel 3 beschriebenen Verfahren gelöst.

Sind die Potentialfelder berechnet, so werden daraus die zellzentrierten Felder berechnet und die rechten Seiten für deren Umwandlung in punktbasierte Felder befüllt. Als letztes folgt die Umwandlung der Felder und das Hinzufügen der umgewandelten Felder zum DataSet.

Die rotationsfreie und die divergenzfreie Komponente des Ausgangsvektorfeldes sind dann von anderen Algorithmen in *FAnToM* benutzbar.

Kapitel 8

Resultate

In dieser Projekt-Arbeit sind zwei in *FAnToM* eingebettete Algorithmen entstanden. Die Helmholtz-Hodge-Zerlegung und die Tetrahedrisierung wurden beide unter den DataAlgos eingeordnet und sind in *FAnToM* über das entsprechende Menü zugreifbar. Die Arbeit stellt somit eine Erweiterung der von *FAnToM* angebotenen Werkzeuge dar. Die Tetrahedrisierung wird wahrscheinlich nicht nur zum Vorbereiten von Datensätzen für die Helmholtz-Hodge-Dekomposition dienen, sondern auch für andere Algorithmen wie Marching Tetrahedra eine Erweiterung des Spektrums der zu verarbeitenden Datensätze ermöglichen.

8.1 Bilder

Die Ergebnisse von Durchläufen der Algorithmen sind auf den Bildern im Anhang A zu sehen. Zwei Bilder zeigen die Veränderung des Bluntfin-Gitters durch die Tetrahedrisierung. Andere Bilder geben durch Isoflächen oder einen Hedgehog Eindrücke der errechneten Potentialfelder.

8.2 Lösen der Gleichungssysteme

Die Berechnung des Potentialfeldes für die divergenzfreie Komponente führt über das Lösen von nichtsymmetrischen Matrizen. Da für nichtsymmetrische Matrizen die Methode der konjugierten Gradienten nicht anwendbar ist, musste hierfür ein anderes Verfahren (bikonjugierte Gradienten) angewendet werden. Die Konvergenz dieses Verfahrens ist nicht so gut wie die des konjugierte Gradienten-Verfahrens. Es zeigte sich, dass die Konvergenz, die anfangs nicht in akzeptabler Zeit zu Lösungen führte, durch Wahl eines anderen Verfahrens zur Prekonditionierung erheblich verbessert werden konnte. Die Wahl der Prekonditionierung spielt hier also eine noch größere Rolle als bei symmetrischen Matrizen.

8.3 Zu den Datensätzen

Das Gitter des Bluntfin (nicht der innere Teil, der in den Bildern verwendet wurde) stellte bei der Tetrahedrisierung anfangs eine Hürde dar, da er Hexaederzellen enthält, welche die Form von Prismen haben, bei denen also zweimal jeweils zwei Punkte auf einer Position zusammen fallen. Das Problem wurde allerdings mit einer entsprechenden Sonderfallbehandlung bei doppelten Punkten behoben.

Kapitel 9

Ausblick

Für die Tetrahedrisierung wird in Zukunft nicht mehr viel Arbeit nötig sein. Sie dient, wie in den Resultaten schon erwähnt, in erster Linie als Vorstufe für andere Algorithmen. Es bleibt, sich Gedanken über sehr ungünstig geformte Zellen zu machen. Da die Zellen in *FAnToM* von bilinearen Patches als Seitenflächen begrenzt sind, können unglücklich gewählte Eckpunkte zu sehr unschönen Zellen führen. So könnten bei der Tetrahedrisierung einer ungünstigen Pyramide zwei sich überschneidende Tetraeder entstehen. Das macht für den Algorithmus selbst zwar keine Probleme, aber Algorithmen, die das erzeugte Gitter benutzen könnten Probleme bekommen. Diese Probleme können zum Beispiel bei der Helmholtz-Hodge-Zerlegung auftreten, weil dort das Volumen der Tetraeder benutzt wird. Wenn sich zwei Tetraeder überschneiden, ist es schwierig, das Volumen einem der beiden Tetraeder zuzuordnen.

Die Helmholtz-Hodge-Zerlegung bietet alleine durch die Interpretation der produzierten Vektorfelder auch in Zukunft ein interessantes Betätigungsfeld. Die Vektorfelder können zur Detektion von Merkmalen des Vektorfeldes benutzt werden. Da die Felder nur jeweils Rotation oder Divergenz enthalten sind die Merkmale, wie im Zweidimensionalen Quellen, Senken und Zykel, unter Umständen leichter zu finden. Im Dreidimensionalen gibt es mehr Arten kritischer Punkte als im Zweidimensionalen (z.B. Quellen: *Repelling Star, Repelling Focus,...*). Es wird interessant sein, zu untersuchen, wie diese Punkte nach der Zerlegung in den einzelnen Komponenten aussehen und welche Schlüsse dann daraus für die Strömung im Ausgangsfeld zu ziehen sind. Eine Untersuchung der Potentiale kann ebenfalls erwogen werden.

Anhang A

Bilder

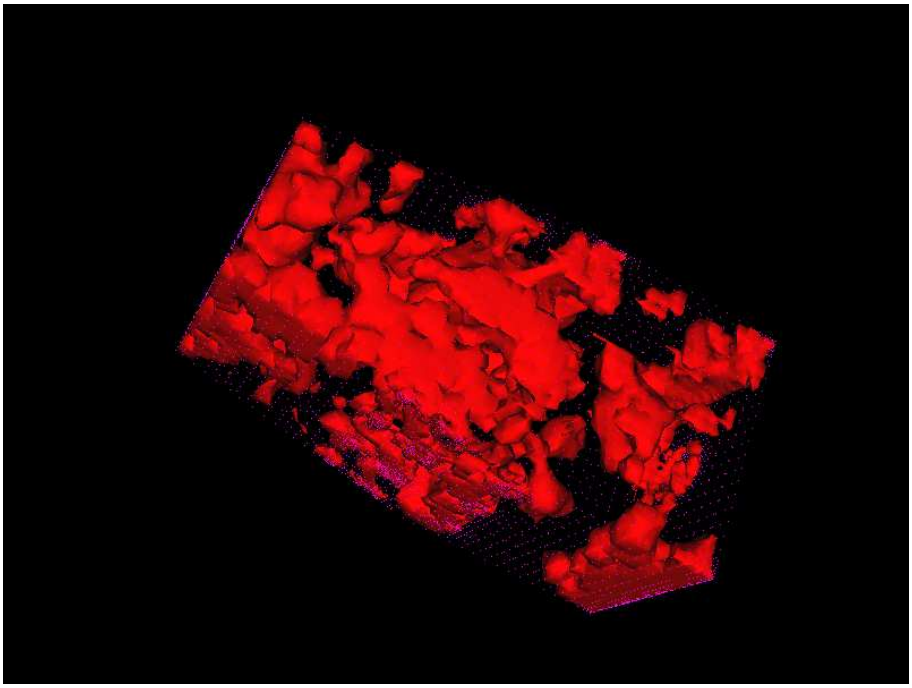


Abbildung A.1: Eine Isofläche im Potentialfeld der rotationsfreien Komponente des Gasbrennkammer-Datensatzes. Isowert=0,1.

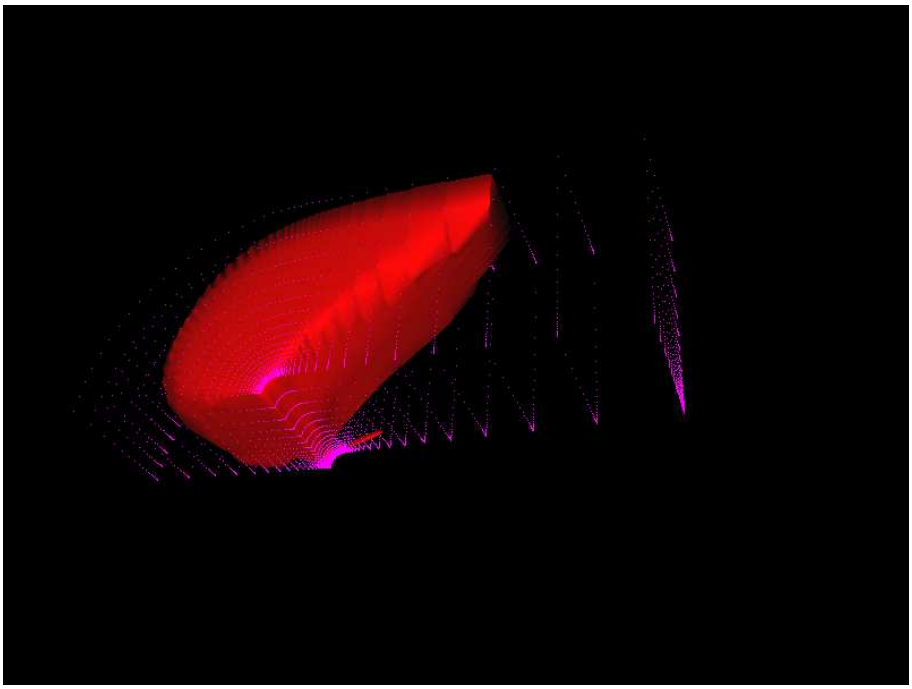


Abbildung A.2: Eine Isofläche im Potentialfeld der rotationsfreien Komponente des Bluntfin-Datensatzes. Isowert=0,01.

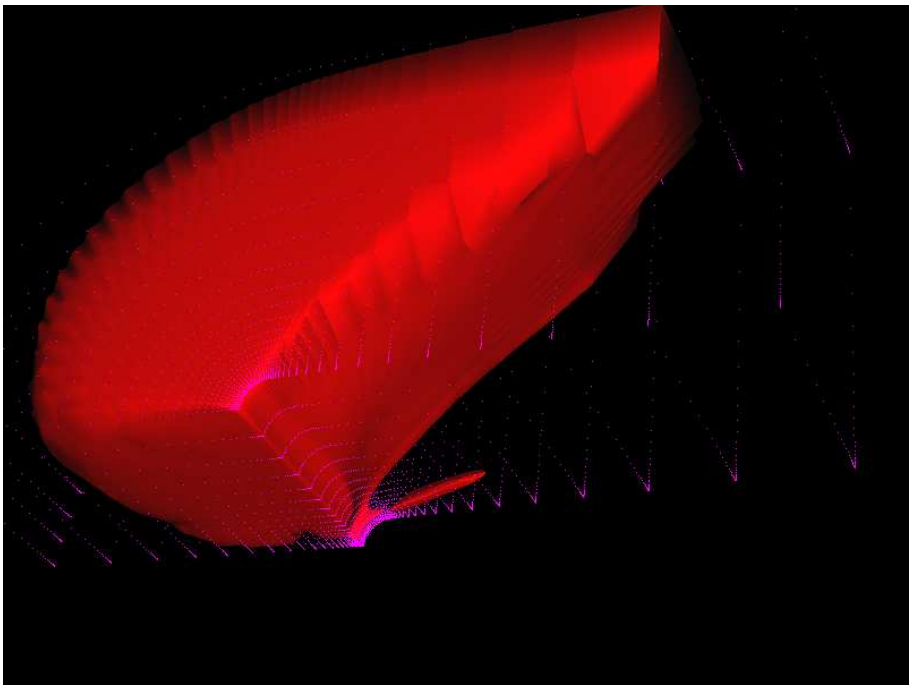


Abbildung A.3: Eine Isofläche im Potentialfeld der rotationsfreien Komponente des Bluntfin-Datensatzes. Isowert=0,005.

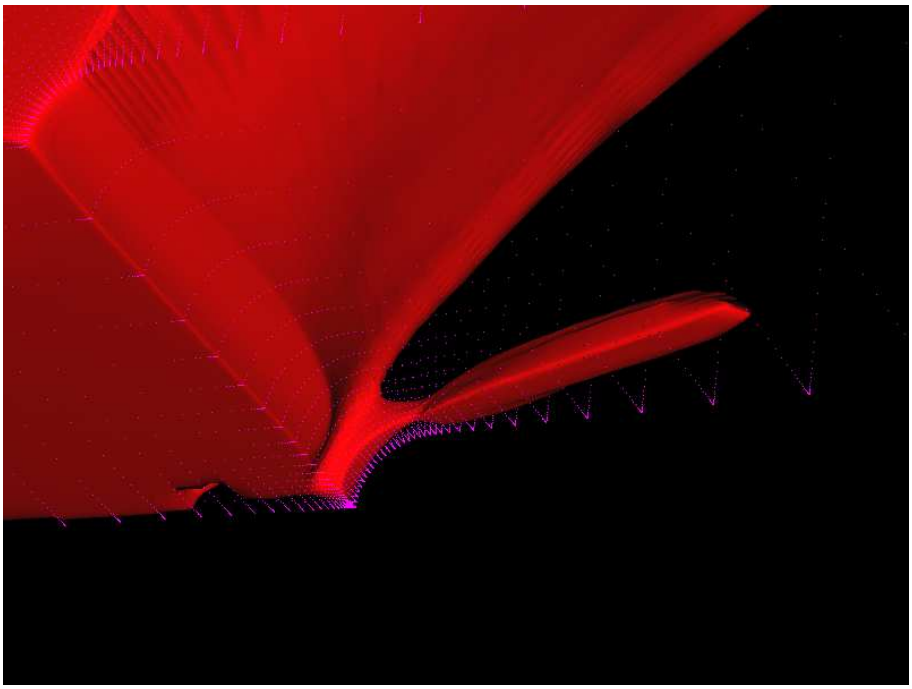


Abbildung A.4: Eine Isofläche im Potentialfeld der rotationsfreien Komponente des Bluntfin-Datensatzes. Isowert=0,001.

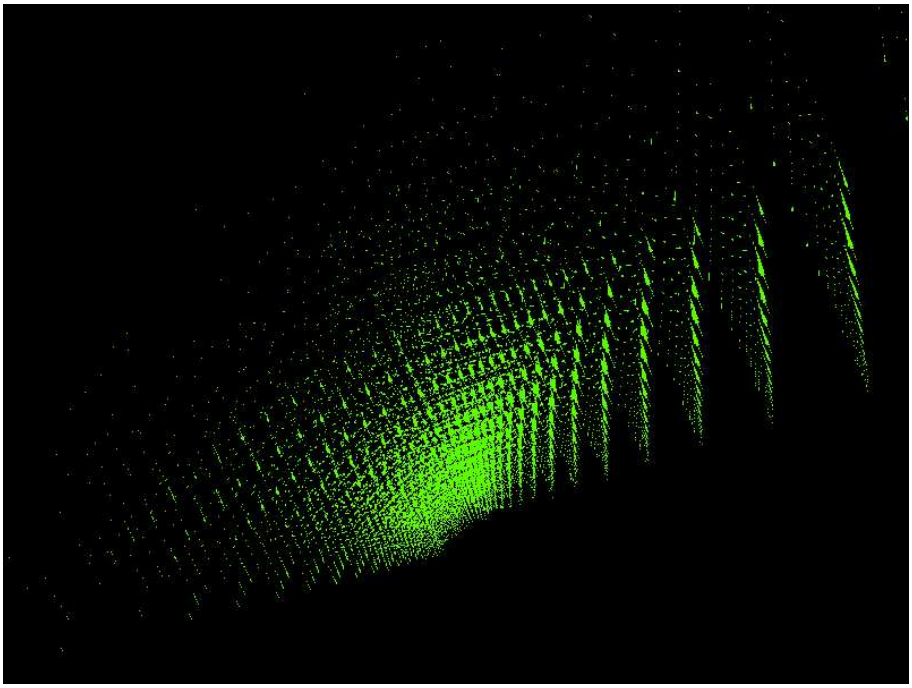


Abbildung A.5: Hedgehog des Vektorpotentials der divergenzfreien Komponente des Bluntfin-Datensatzes.

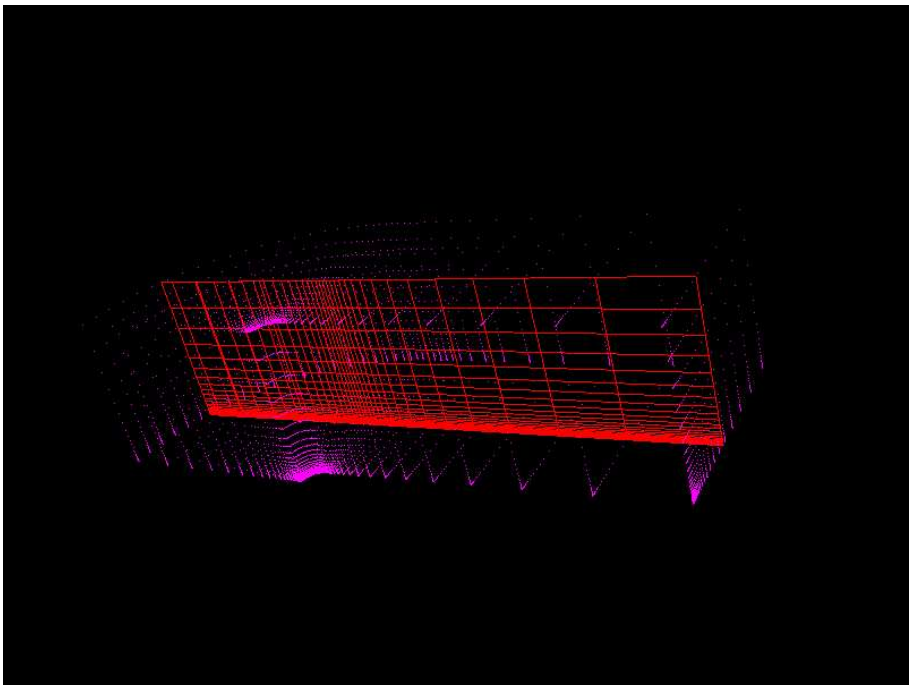
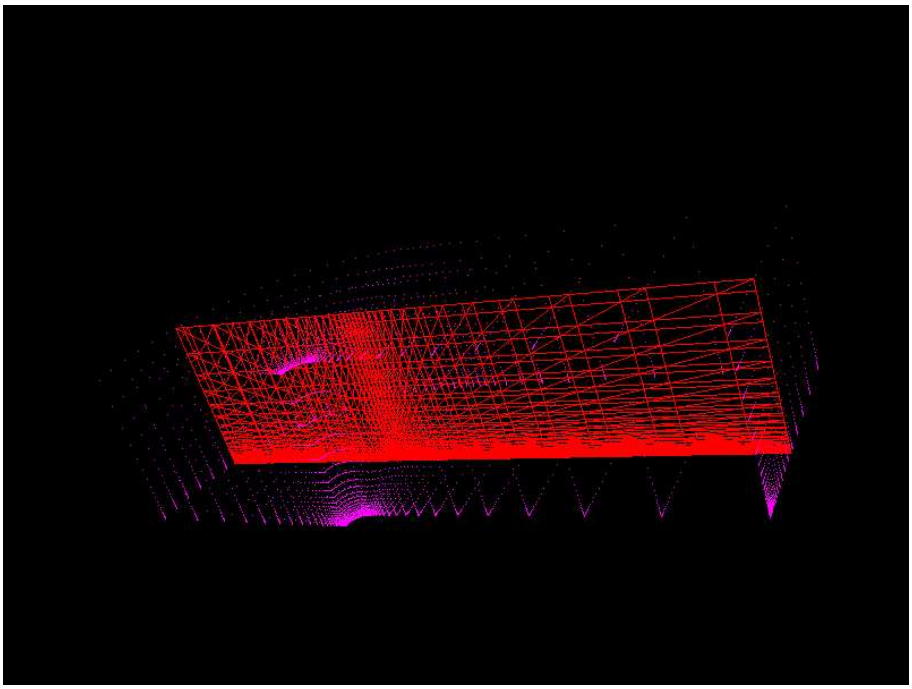


Abbildung A.6: Schnitte durch das Gitter des Bluntfin-Datensatzes mit blau eingezeichneten Randpunkten des Gitters. **oben**: vor der Tetrahedrisierung. **unten**: nach der Tetrahedrisierung.



Anhang B

Formel-Sammlung

B.1 Formeln für die Konjugierte Gradienten Methode

$$
\begin{aligned}
d_0 &= r_0 = b - \mathbf{A}x_0 \\
\alpha_i &= \frac{r_i^T r_i}{d_i^T \mathbf{A} d_i} \\
x_{i+1} &= x_i + \alpha_i d_i \\
r_{i+1} &= r_i - \alpha_i \mathbf{A} d_i \\
\beta_{i+1} &= \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i} \\
d_{i+1} &= r_{i+1} + \beta_{i+1} d_i
\end{aligned}
$$

B.2 Formeln zur Helmholtz-Hodge-Zerlegung

$$
\mathbf{A}(\nabla\varphi_i, \nabla\varphi_j) \cdot v_j = \mathbf{A}(\nabla\varphi_i, \nabla\varphi_j) \cdot \begin{pmatrix} v_{jx} \\ v_{jy} \\ v_{jz} \end{pmatrix}
$$

mit

$$
\mathbf{A}(\nabla\varphi_i, \nabla\varphi_j) = \begin{pmatrix} -(\partial y\varphi_i)(\partial y\varphi_j) - (\partial z\varphi_i)(\partial z\varphi_j) & (\partial y\varphi_i)(\partial x\varphi_j) & (\partial z\varphi_i)(\partial x\varphi_j) \\ (\partial x\varphi_i)(\partial y\varphi_j) & -(\partial x\varphi_i)(\partial x\varphi_j) - (\partial z\varphi_i)(\partial z\varphi_j) & (\partial z\varphi_i)(\partial y\varphi_j) \\ (\partial x\varphi_i)(\partial z\varphi_j) & (\partial y\varphi_i)(\partial z\varphi_j) & -(\partial x\varphi_i)(\partial x\varphi_j) - (\partial y\varphi_i)(\partial y\varphi_j) \end{pmatrix}
$$

Anhang C

Code-Fragmente

C.1 Falltabelle

FConvertToTetrahedronGridLookupTable.hh:

```
/*
       6--------7
      /|       /|
     / |      / |
    4--------5  |
    |  |     |  |
    |  2-----|--3
    | /      | /
    |/       |/
    0--------1

   Faces:   0=LEFT, 1=RIGHT,  2=FRONT,
   3=BACK, 4=DOWN, 5=UP

*/
static const int hexFiveTetsTable[2][5][4]={
    {
{0,1,3,5},
{0,3,2,6},
{0,5,6,4},
{5,3,6,7},
{0,5,3,6}
    },
    {
```

```
{0,1,2,4},
{1,3,2,7},
{1,7,4,5},
{4,7,2,6},
{1,2,4,7}
    }
};

static const int hexPrismsVertsTable[6][2][6]={
    {
{0,6,4,1,7,5},
{0,2,6,1,3,7}
    },
    {
{0,2,4,1,3,5},
{4,2,6,5,3,7}
    },
    {
{0,5,1,2,7,3},
{0,4,5,2,6,7}
    },
    {
{0,4,1,2,6,3},
{1,4,5,3,6,7}
    },
    {
{0,1,3,4,5,7},
{0,3,2,4,7,6}
    },
    {
{0,1,2,4,5,6},
{1,3,2,5,7,6}
    }
};

/*
      5
     /|\
    / | \
   3-----4
   |  |  |
```

```
   |  |  |
   |  2  |
   | / \ |
   |/   \|
   0-----1

A 0 in the binary index stands for a diagonal from the lowest vertex-index
of the side

 */
static const int prismTetsTable[8][3]={
    { 2, 4, 7}, /*000*/
    { 2, 3,10}, /*001*/
    { 1, 6, 7}, /*010*/
    {-1,-1,-1}, /*011*/   // The cases "100" and "011"
    {-1,-1,-1}, /*100*/   // dont't appear
    {10, 0, 9}, /*101*/
    { 1, 5,11}, /*110*/
    { 0, 8,11}  /*111*/
};

static const int prismTetsVertsTable[12][4]={
    {0,1,2,3}, /*0*/
    {0,1,2,4}, /*1*/
    {0,1,2,5}, /*2*/
    {0,3,1,5}, /*3*/
    {0,4,1,5}, /*4*/
    {0,2,3,4}, /*5*/
    {0,4,2,5}, /*6*/
    {0,4,5,3}, /*7*/
    {1,2,3,4}, /*8*/
    {1,2,3,5}, /*9*/
    {1,3,4,5}, /*10*/
    {2,3,4,5}, /*11*/
};
```

C.2 Auswahl der Zerlegung nach Zelltyp

```
  switch(cellType){
      case FCell::HEXAHEDRON:
```

```
    case FCell::AXIS_PARALLEL_HEX:
        decomposeHexahedron(myInVerts);
        hexs++;
        break;
    case FCell::PRISM:
    case FCell::AXIS_PARALLEL_PRISM:
        decomposePrism(myInVerts);
        prisms++;
        break;
    case FCell::PYRAM:
        decomposePyramid(myInVerts);
        pyramids++;
        break;
    case FCell::TETRAHEDRON:
    case FCell::AXIS_PARALLEL_TET:
        addTetrahedron(vertexIndices);
        tets++;
        break;
    default:
        cout<<"UNEXPECTED CELLTYPE APPEARED "
            <<"at FConvertToTetrahedronGridAlgorithm::execute"
            <<endl;
}
```

Literaturverzeichnis

[1] R. Abraham, J. Marsden, and T. Ratiu, editors. *Manifolds, Tensor Analysis and Applications*, volume 75 of *Applied Mathematical Science*. Springer, 1988.

[2] Il'ja N. Bronstein, Konstantin A. Semendjajev, Gerhard Musiol, and Heiner Mühlig. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Thun und Frankfurt am Main, fünfte edition, 2000.

[3] Holger Burbach, Gerik Scheuermann, Xavier Tricoche, and Thomas Wischgoll. *FAnToM (Field Analysis by Topological Methods)*. AG Graphische Datenverarbeitung und Computergeometrie, Universität Kaiserslautern, 2002.

[4] R. Fletcher. Conjugate gradient methods for indefinite systems. In G. Watson, editor, *Numerical Analysis Dundee 1975*, pages 73–89. Springer Verlag, Berlin, New York, 1976.

[5] Otto Forster. *Analyasis 2*. Number 31 in vieweg Studium, Grundkurs Mathematik. Vieweg, fünfte edition, 1996.

[6] Max Langbein, Gerik Scheuermann, and Xavier Tricoche. An efficient point location method for visualization in large unstructured grids. preprint.

[7] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, August 1994.
`http://www-2.cs.cmu.edu/~jrs/jrspapers.html`.

[8] Yiying Tong, Santiago Lombeyda, Anil N. Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. 2003.
`http://www-grail.usc.edu/pubs/TLHD03.pdf`.