Entwicklung eines effizienten Verfahrens zur Lösung hyperbolischer Differentialgleichungen

Joachim Becker
Institut für Angewandte Mathematik
Universität Freiburg

Dissertation
zur Erlangung des Doktorgrades
vorgelegt bei der
Mathematischen Fakultät
Albert-Ludwigs-Universität
Freiburg

Betreuer: Prof. Dr. D. Kröner

29. Oktober 2000

Dekan:	Prof. Dr. W. Soergel
Referenten:	Prof. Dr. D. Kröner
	Prof. Dr. M. Rumpf, Bonn
Datum der Promotion:	30.01.2000

Zusammenfassung

In dieser Arbeit wollen wir ein effizientes Verfahren zur Lösung hyperbolischer Differentialgleichungen entwickeln. Wir betrachten sowohl skalare Gleichungen als auch Systeme, hier speziell die Euler-Gleichungen der Gasdynamik, in 2D und in 3D. Dabei rechnen wir auf verschiedenen Gittertypen. Um ein Verfahren mit möglichst wenig Viskosität zu erhalten, setzt unsere Methode auf den so genannten Discontinuous Galerkin Methoden auf. Dabei wird die exakte Lösung stückweise durch ein Polynom höheren Grades anstatt einer stückweise konstanten Funktion approximiert. Dieses Verfahren wird an mehreren Testbeispielen getestet. Schliesslich wird diese Methode angewendet, um ein zeitabhängiges Vektorfeld zu visualisieren. Dabei wird eine neu entwickelte Methode, der Texturtransport, vorgestellt.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Zur Beschreibung der Natur werden hyperbolische ”Erhaltungssätze” hauptsächlich in der Strömungsmechanik angewendet. Hier seien einige Beispiele erwähnt, bei denen hyperbolische Differentialgleichungen auftreten können.
Die Umströmung von Autos und die Strömungsvorgänge im Motor sind entscheidend für Leistung, Geräuschentwicklung, Brennstoffverbrauch und Schadstoffausstoß.
In der Luft- und Raumfahrttechnik ist die Entwicklung komplexer Fluggeräte und Trägersysteme ohne intensiven Einsatz numerischer Simulationen nicht denkbar. Optimierungen nach Aspekten der Sicherheit, der Wirtschaftlichkeit und der Umweltbelastung sind im Rahmen einer zukunftsverträglichen Lösung der Probleme des Luft- und Orbitalverkehrs zwingend. Derartige Untersuchungen beziehen die gesamte Wechselwirkung zwischen Strömung und Struktur, die Strömungs- und Verbrennungsvorgänge der Antriebssysteme mit ein. Die benötigten Simulationen können heute nur eingeschränkt (z.T. nur mit vereinfachten Modellen) durchgeführt werden, da die Leistung der verfügbaren Rechner nicht ausreicht.
Andere Anwendungsgebiete von hyperbolischen Differentialgleichungen neben der Luftfahrt sind z.B. die Meteorologie und Wettervorhersage, die Astrophysik, die Ausbreitung von Wellen in elastischen Körpern und der Fluss von Gletschern.
Sehr große Teile der physikalischen Phänomene können durch die Euler-Gleichungen der Gasdynamik beschrieben werden. Die Euler-Gleichungen sind ein Spezialfall der Navier-Stokes-Gleichungen für reibungsfreie Medien. Wir werden uns in dieser Arbeit auf diese Gleichungen beschränken.

In dieser Arbeit betrachten wir die Diskretisierung für Systeme von hyperbolischen Erhaltungssätzen

$$\partial_t \mathbf{w} + \operatorname{div} \mathbf{f}(\mathbf{w}) = 0 \quad \text{in} \quad \mathbb{R}^{\mathrm{d}} \times \mathbb{R}^{+} \tag{1.1}$$

mit Anfangswerten

$$\mathbf{w}(\mathbf{x}, 0) = \mathbf{w_0}(\mathbf{x}) \quad \text{in} \quad \mathbb{R}^{\mathrm{d}}\,. \tag{1.2}$$

Dabei seien

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}, \mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix} \text{ und } \mathbf{f} = \begin{pmatrix} f_1 \\ \vdots \\ f_d \end{pmatrix}.$$

d bezeichnet die Raumdimension und m die Anzahl der Komponenten.
Der Divergenzoperator ist definiert durch

$$\text{div } \mathbf{f}(\mathbf{w}) = \sum_{i=1}^{d} \partial_{x_i} f_i(\mathbf{w}),$$

wobei $\mathbf{w} : \mathbb{R}^{\mathrm{d}} \times \mathbb{R}^{+} \to \mathbb{R}^{\mathrm{m}}$ und $f_1, \ldots, f_d : \mathbb{R}^{\mathrm{m}} \to \mathbb{R}^{\mathrm{m}}$.
Im Fall $m = 1$ reden wir vom skalaren Fall. In dieser Arbeit betrachten wir nur die Fälle $d = 2, 3$. Die f_i nennen wir auch Flussfunktionen.
Wir nehmen an, dass die Nichtlinearitäten f_1 , f_2 und f_3 in (1.1) glatt sind. Im Wesentlichen gibt es drei Methoden, um (1.1) und (1.2) zu lösen. Das sind die Methode der Finiten Differenzenverfahren auf kartesischen Gittern in Form von Dimensionssplitting [18], [20], [21], [47], [57], [63], die Finiten Elemente Verfahren [32], [33], [34], [35] und die Verfahren der Finiten Volumen [25], [37], [42], [44], [61], [69] auf beliebigen Gittern. Bis jetzt gibt es noch keine generellen Konvergenzsätze für numerische Schemata für Systeme in 2-D. Für kleine Zeiten liegen Existenzsätze vor [36]. Für grosse Zeiten jedoch gibt es für die Euler-Gleichungen in mehreren Raumdimensionen keinen Existenzsatz.

Ziel dieser Arbeit ist die Entwicklung eines effizienten Verfahren höherer Ordnung im Ort. Dazu werden die von Cockburn, Shu, Hou und Chavent [8], [14], [15], [16], [17] entwickelten Discontinuous Galerkin Verfahren in mehreren Raumdimensionen betrachtet. Jaffre, Johnson und Szepessy [31] zeigen für diese Verfahren die Konvergenz im skalaren Fall. Es wird untersucht, wie sich die sehr theoretischen Aspekte algorithmisch umsetzen lassen, speziell in $3D$, und wie man sie effizient auf einem Parallelrechner implementiert.
Ein weiteres Hauptziel dieser Arbeit ist die Entwicklung einer Visualisierungsmethode für zeitabhängige Vektorfelder. Die Visualisierung von Vektorfeldern ist eine der wichtigsten Aufgaben mathematischer Visualisierung. Bislang gibt es nur auf stationären Vektorfeldern zufriedenstellende Methoden. Diese fehlten bislang bei instationären Vektorfeldern. Hier wird nun eine neue Methode vorgestellt, bei der ein numerisches Verfahren vom hyperbolischen Typ zu lösen ist. Dabei kommen dann auch die Discontinuous Galerkin Verfahren zum Einsatz.
Ein spezielles Phänomen hyperbolischer Differentialgleichungen ist, dass auch bei glatten Anfangsdaten Unstetigkeiten in der Lösung entstehen können. Numerische Verfahren haben meistens die Eigenschaft, Unstetigkeiten zu verschmieren. Diese Verschmierungen sind bei Verfahren erster Ordnung oft sehr groß. Bei Verfahren erster Ordnung wird auf einem Kontrollvolumen die gesuchte Lösung durch einen konstanten Wert approximiert.

Mit diesen konstanten Werten wird dann der numerische Fluss ausgewertet. Um diese Verschmierungen zu minimieren, müssen Verfahren höherer Ordnung betrachtet werden. Bei den verwendeten Verfahren höherer Ordnung wie MUSCL (Monotonic Upstream Centered Schemes for Conservation Laws) [38], [26] oder ENO (Essentially Non Oscillatory) [38], [55] wird in jedem Zeitschritt aus den konstanten Werten in den Kontrollvolumen in Abhängigkeit der Werte in den Nachbarzellen ein Polynom höheren Grades rekonstruiert. Mit diesem rekonstruierten Polynom wird dann die Flussberechnung durchgeführt. Danach hat man jedoch wieder die Lösung als konstanten Wert auf dem Kontrollvolumen vorliegen.

Bei den Discontinuous Galerkin Verfahren [8], [14], [15], [16], [17] versucht man nun, einen anderen Weg zu gehen. Die gesuchte Lösung auf einem Kontrollvolumen wird durch ein Polynom höheren Grades approximiert. Mit diesem Polynom wird dann der numerische Fluss ausgewertet. Wie sich im Laufe dieser Arbeit herausstellen wird, ist dieses Verfahren auf einem festen Gitter sehr viel langsamer als die Verfahren erster Ordnung oder die anderen Verfahren höherer Ordnung. Dieser Vergleich ist aber nur bedingt zulässig. Um nämlich eine bestimmte Fehlertoleranz zu unterschreiten, kann bei den Discontinuous Galerkin Verfahren auf einem sehr viel gröberen Gitter gerechnet werden, womit die reine Rechenzeit wesentlich reduziert wird. Auch muss man die Rechenzeit pro Freiheitsgrad betrachten. Dann stellt sich heraus, dass die Rechenzeit der Discontinuous Galerkin Verfahren doch nicht so groß ist. Darüberhinaus liefern die Discontinuous Galerkin Verfahren sehr beeindruckende Ergebnisse.

Die Arbeit gliedert sich wie folgt: In Kapitel 2 werden die Finiten Volumen Verfahren in allgemeiner Art erklärt. Zusätzlich werden hier die Notationen eingeführt. In Kapitel 3 werden Verfahren zur Lösung skalarer Gleichungen vorgestellt und die Euler-Gleichungen der Gasdynamik besprochen. Weiter werden numerische Flussfunktionen zur Lösung der Euler-Gleichungen vorgestellt. In Kapitel 4 werden dann die Discontinuous Galerkin Verfahren vorgestellt. Dabei werden Verfahren zweiter und dritter Ordnung auf Dreiecken, Quadraten, Tetraedern und Hexaedern analysiert. In Kapitel 5 soll die Parallelisierung der Discontinuous Galerkin Verfahren betrachtet werden. Aufgrund der hohen Laufzeit der Verfahren ist eine Parallelisierung unumgänglich, wenn man auf hinreichend feinen Gittern arbeiten will. Darüberhinaus eignen sich die Discontinuous Galerkin Verfahren besonders gut zum Parallelisieren. Da die Polynomordnung lokal begrenzt ist, ist der Aufwand nicht sehr viel höher als bei Verfahren erster Ordnung. In Kapitel 6 soll die numerische Konvergenzordnung der Verfahren analysiert werden. In Kapitel 7 werden weitere numerische Beispiele vorgestellt. In Kapitel 8 wird eine Anwendung vorgestellt, die wesentlichen Gebrauch der Discontinuous Galerkin Verfahren macht. Bei dieser neu entwickelten Visualisierungsmethode für zeitabhängige Vektorfelder soll ein vorgegebenes Muster (wir sprechen von Textur) mit einem gegebenen oder gleichzeitig gerechneten Vektorfeld transportiert werden. Für diese Transportrechnungen war es notwendig, ein Verfahren mit möglichst wenig numerischer Viskosität zu entwickeln. In

Kapitel 9 wird eine weitere Visualisierungsmethode vorgestellt. Für skalare Datenfelder ist Volumenvisualisierung eine beeindruckende Methode. Hier wird die Volumenvisualisierung in der Strömungsmechanik angewendet, wobei die Numerik wie in Kapitel 8 benutzt wird. In Kapitel 10 schließlich fassen wir die Arbeit zusammen und geben einen Ausblick auf weitere Forschungsmöglichkeiten.
Als Programmiersprache haben wir C gewählt, weil diese Sprache auf allen Computersystemen verfügbar ist und sie geeignete Datenstrukturen besitzt. Ferner gibt es schon genügend Unterroutinen, die in C geschrieben sind und die man deswegen leicht in eigene Programme einbinden kann. So haben wir in dieser Arbeit beim Programmieren das Grafiksystem GRAPE [51],[28] für Triangulierungen, Verfeinerungen und graphische Darstellung der Ergebnisse genutzt.
Noch eine technische Bemerkung: Da wir in dieser Arbeit sowohl den 2D als auch den 3D Fall betrachten, haben wir uns beim Notieren von Gleichungen für folgenden Weg entschieden: Auftretende Matrizen, Vektoren und Gleichungen erscheinen in der 3D Version. Ist keine explizite 2D Version angegeben, so sind die 2D Versionen in einfacher Weise auf ähnlichem Weg zu erhalten. Dazu ist in der Regel nur der f_3 Teil wegzulassen, bei (5x5)-Matrizen die vierte Spalte und Zeile zu streichen und bei 5er-Vektoren die vierte Zeile zu streichen. Im verbleibendem Teil ist zum Beispiel n_3 und u_3 gleich Null zu setzen.

Kapitel 2

Finite Volumen

In diesem Kapitel werden zunächst die Notationen eingeführt. Dann werden allgemeine Eigenschaften hyperbolischer Differentialgleichungen besprochen und schließlich die Finiten Volumen Verfahren zur numerischen Lösung hergeleitet.

2.1 Notationen

DEFINITION 2.1.1 ***(unstrukturiertes Gitter)***
(2D) Sei ein k-Polygon ein abgeschlossenes, konvexes Polygon mit k Knoten. Die Menge

$$\mathcal{T} := \{T_i \mid T_i \quad \textit{ist ein k-Polygon für} \quad i \in I \subseteq \mathbb{N}\},$$

(wobei $I \subseteq \mathbb{N}$ eine Indexmenge ist) wird unstrukturiertes Gitter von $\Omega \subset \mathbb{R}^2$ genannt, falls die folgenden Eigenschaften erfüllt sind:

$$1) \quad \Omega = \bigcup_{i \in I} T_i \,,$$

$$2) \quad \textit{Für zwei verschiedene} \quad T_i, T_j \textit{ gilt entweder}$$

$$T_j \cap T_i = \emptyset \textit{ oder}$$

$$T_j \cap T_i = \textit{ ein gemeinsamer Knoten von } T_i, T_j \textit{ oder}$$

$$T_j \cap T_i = \textit{ eine gemeinsame Kante von } T_i, T_j.$$

BEMERKUNG 2.1.2 *In 3D erweitert sich Eigenschaft 2 von Definition 2.1.1 um*

$$T_j \cap T_i = \textit{ eine gemeinsame Fläche von } T_i, T_j \,.$$

Ein Gitter (wir sprechen auch von **Triangulierung**), das Definition 2.1.1 erfüllt, heißt auch konforme oder zulässige Triangulierung. Wenn im folgenden ein Knoten diese Definition verletzt, so sprechen wir von einem nichtkonformen (oder auch hängenden) Knoten. Die Abbildung 2.1 zeigt einen nichtkonformen Knoten.

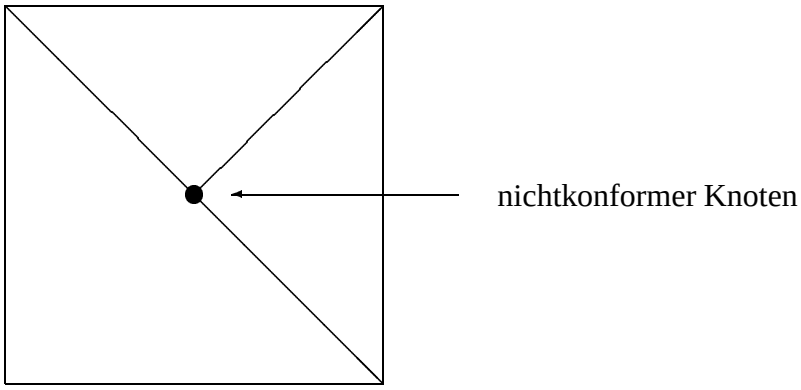

Abbildung 2.1: Nichtkonformer Knoten

Sei $\mathcal{T} = (T_j)_j$ ein unstrukturiertes Gitter im $\mathbb{R}^d$. Im folgenden benutzen wir folgende Notationen:

$\mathbf{x}$	:	in (2D): (x_1, x_2) bzw. (x, y)
		in (3D): (x_1, x_2, x_3) bzw. (x, y, z)
		die Raumvariable
t	:	die Zeitvariable
$\mathbf{w}(\mathbf{x}, t)$	:	$(w_1(\mathbf{x}, t), \ldots, w_m(\mathbf{x}, t))$
		m Anzahl der Komponenten im Vektor $\mathbf{w}(\mathbf{x}, t)$
		$m = 1$ ist der skalare Fall
$\mathbf{w}$	:	Abkürzung für $\mathbf{w}(\mathbf{x}, t)$
$\mathbf{f}(\mathbf{w})$	:=	$\begin{pmatrix} f_1(\mathbf{w}) \\ f_2(\mathbf{w}) \end{pmatrix}$ in (2D)
		$\begin{pmatrix} f_1(\mathbf{w}) \\ f_2(\mathbf{w}) \\ f_3(\mathbf{w}) \end{pmatrix}$ in (3D)

T_j	:	die j-te Zelle vom unstrukturierten Gitter.
$\mid T_j \mid$	:	Fläche (2D) bzw. Volumen (3D) von T_j.
$T_{jl}, l = 1, \ldots, N$	:	Nachbarzellen von T_j.
N	:	Anzahl der Nachbarelemente (Dreiecke N=3, Tetraeder N=4, Rechtecke N=4, Hexaeder N=6)
$\alpha(j,l)$	:	globale Nummer von der l-ten Nachbarzelle T_{jl} von T_j, so dass $T_{jl} = T_{\alpha(j,l)}$.
t^n	:	Zeit nach n Zeitschrittiterationen
Δt^n	:	Mit $\Delta t^n := t^n - t^{n-1}$ (kurz Δt) bezeichnen wir den Zeitschritt von t^{n-1} nach t^n.
w_j^n	:	Approximation der exakten Lösung $\mathbf{w}$ auf T_j zur Zeit t^n.
w_{jl}^n	:	Approximation der exakten Lösung $\mathbf{w}$ auf T_{jl} zur Zeit t^n; $w_{jl}^n = w_{\alpha(j,l)}^n$.
S_{jl}	:	l-te Rand-Kante (2D) bzw. Rand-Fläche (3D) von T_j.
$\mid S_{jl} \mid$	:	Länge (2D) bzw. Fläche (3D) von S_{jl}.
ν_{jl}	:	äußere Normale zu S_{jl} mit der Länge $\mid S_{jl} \mid$.
n_{jl}	:=	$\frac{\nu_{jl}}{\mid \nu_{jl} \mid} = (n_{jl}^x, n_{jl}^y, n_{jl}^z)^t$. Mit $n_{jl}^x, n_{jl}^y, n_{jl}^z$ bezeichnen wir jeweils die $x-$,$y-$ bzw. $z-$ Koordinate von n_{jl}. Es gilt $\mid n_{jl} \mid = 1$. Wenn klar ist, welche Kante gemeint ist, schreiben wir auch $n = (n_1, n_2, n_3)^t$
h	:=	$\sup_j \text{diam}(T_j)$. Bei Dreiecken ist der Durchmesser gleich der längsten Seite.
$w_h(z,t)$	:=	w_j^n, falls $z \in T_j$ und $t^n \leq t < t^{n+1}$, $\quad n = 0, 1, \ldots$

2.2 Schwache Lösung

Wenn wir das System (1.1)

$$\partial_t \mathbf{w} + \partial_x f_1(\mathbf{w}) + \partial_y f_2(\mathbf{w}) + \partial_z f_3(\mathbf{w}) = 0$$

mit einer Testfunktion $\phi(x, y, z, t)$ mit $\phi \in C_0^\infty(\mathbb{R}^3 \times \mathbb{R}_0^+)$ multiplizieren und dann über Raum und Zeit integrieren, erhalten wir

$$\int_{\mathbb{R}^3 \times \mathbb{R}_0^+} [\phi \partial_t \mathbf{w} + \phi \partial_x f_1(\mathbf{w}) + \phi \partial_y f_2(\mathbf{w}) + \phi \partial_z f_3(\mathbf{w})] dx dy dz dt = 0\,.$$

Durch partielle Integration erhalten wir

$$\begin{aligned} \int_{\mathbb{R}^3 \times \mathbb{R}_0^+} &[\partial_t \phi \mathbf{w} + \partial_x \phi f_1(\mathbf{w}) + \partial_y \phi f_2(\mathbf{w}) + \partial_z \phi f_3(\mathbf{w})] dx dy dz dt = \\ &- \int_{\mathbb{R}^3} \phi(x, y, z, 0) \mathbf{w}(x, y, z, 0) dx dy dz\,. \end{aligned} \tag{2.1}$$

DEFINITION 2.2.1 ***(schwache Lösung)***
Die Funktion $\mathbf{w}(x, y, z, t)$ wird schwache Lösung der Differentialgleichung genannt, wenn (2.1) für alle Funktionen $\phi \in C_0^\infty(\mathbb{R}^3 \times \mathbb{R}_0^+)$ erfüllt ist.

2.3 Generelle Eigenschaften hyperbolischer Differentialgleichungen

In diesem Abschnitt werden allgemeine Eigenschaften über hyperbolische Differentialgleichungen erörtert. Wir machen dies für den 3D Fall. Die Reduzierung auf den 2D Fall ist, wenn nicht explizit angegeben, offensichtlich.
Für die Definition hyperbolischer Differentialgleichungen benötigen wir zunächst folgende

DEFINITION 2.3.1 ***(Jacobimatrix)***
Die Jacobimatrix von $f_i(\mathbf{w})$ bezeichnen wir mit $(\partial f_i(\mathbf{w}))$ oder kurz $f_i'(\mathbf{w})$.

$$f_i'(\mathbf{w}) = \begin{pmatrix} \frac{\partial f_{i1}(\mathbf{w})}{\partial w_1} & \cdots & \frac{\partial f_{i1}(\mathbf{w})}{\partial w_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{im}(\mathbf{w})}{\partial w_1} & \cdots & \frac{\partial f_{im}(\mathbf{w})}{\partial w_m} \end{pmatrix}\,.$$

Sei

$$C_{jl}(\mathbf{w}) := (\partial_n \mathbf{f}(\mathbf{w})) = n_{jl}^x f_1'(\mathbf{w}) + n_{jl}^y f_2'(\mathbf{w}) + n_{jl}^z f_3'(\mathbf{w})\,.$$

DEFINITION 2.3.2 ***(Hyperbolische Erhaltungsgleichung)***
Sei Ω eine offene Teilmenge des $\mathbb{R}^m$ und seien $f_1, f_2, f_3 : \Omega \to \mathbb{R}^m$ hinreichend glatte Funktionen (mindestens $\in C^2$). Das Cauchy-Problem

$$\partial_t \mathbf{w} + \partial_x f_1(\mathbf{w}) + \partial_y f_2(\mathbf{w}) + \partial_z f_3(\mathbf{w}) = 0 \quad \textit{mit} \quad (x, y, z, t) \in \mathbb{R}^3 \times]0, \mathrm{T}[\,, \tag{2.2}$$

wobei $\mathbf{w} : \mathbb{R}^3 \times \mathbb{R}^+ \to \Omega$ mit den Anfangswerten

$$\mathbf{w}(x, y, z, 0) = \mathbf{w}_0(x, y, z) \quad \textit{in} \quad \mathbb{R}^3\,, \tag{2.3}$$

wird (strikt) hyperbolisch genannt, wenn $\lambda f_1'(\mathbf{w}) + \mu f_2'(\mathbf{w}) + \xi f_3'(\mathbf{w})$ für alle $(\lambda, \mu, \xi, \mathbf{w}) \in \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \Omega$ mit (paarweise verschiedenen) reellen Eigenwerten diagonalisierbar ist.

Seien $\lambda_1(\mathbf{w}) \leq \ldots \leq \lambda_m(\mathbf{w})$ die Eigenwerte von $(\partial_n \mathbf{f}(\mathbf{w}))$, $\mid n_{jl} \mid = 1$, mit dazugehörigen Rechts-Eigenvektoren $r_1(\mathbf{w})$, ... , $r_m(\mathbf{w})$. Da die zu lösende Differentialgleichung als hyperbolisch angenommen wurde, ist $(\partial_n \mathbf{f}(\mathbf{w}))$ mit reellen Eigenwerten diagonalisierbar. Sei die Matrix $Q(\mathbf{w})$ so definiert, dass die j-te Spalte von Q der Eigenvektor $r_j(\mathbf{w})$ ist. Deshalb folgt

$$Q^{-1}(\mathbf{w}) \cdot (\partial_n \mathbf{f}(\mathbf{w})) \cdot Q(\mathbf{w}) = diag\{\lambda_k(\mathbf{w})\} \tag{2.4}$$

$$:= \begin{pmatrix} \lambda_1(\mathbf{w}) & 0 & \ldots & 0 \\ 0 & \lambda_2(\mathbf{w}) & \ldots & 0 \\ \vdots & & \ddots & 0 \\ 0 & 0 & & \lambda_m(\mathbf{w}) \end{pmatrix}.$$

Dann definieren wir die Matrix

$$\chi(\mathbf{w}) := Q(\mathbf{w}) \cdot \mathrm{diag}\{\frac{1}{2} + \frac{1}{2} \cdot \mathrm{sign}(\lambda_k(\mathbf{w}))\} \cdot Q^{-1}(\mathbf{w})\,. \tag{2.5}$$

Deshalb haben wir aufgrund von (2.4) und (2.5)

$$\chi(\mathbf{w}) \cdot (\partial_n \mathbf{f}(\mathbf{w})) = Q(\mathbf{w}) \cdot \mathrm{diag}\{\max(\lambda_k(\mathbf{w}), 0)\} \cdot Q^{-1}(\mathbf{w}) =: (\partial_n \mathbf{f}(\mathbf{w}))^+ \tag{2.6}$$

und

$$(Id - \chi(\mathbf{w})) \cdot (\partial_n \mathbf{f}(\mathbf{w})) = Q(\mathbf{w}) \cdot \mathrm{diag}\{\min(\lambda_k(\mathbf{w}), 0)\} \cdot Q^{-1}(\mathbf{w}) =: (\partial_n \mathbf{f}(\mathbf{w}))^-\,. \tag{2.7}$$

Deshalb gilt unter Ausnutzung von (2.6) und (2.7)

$$(\partial_n \mathbf{f}(\mathbf{w})) = (\partial_n \mathbf{f}(\mathbf{w}))^+ + (\partial_n \mathbf{f}(\mathbf{w}))^-\,. \tag{2.8}$$

Für weitere Definitionen und Eigenschaften von hyperbolischen Differentialgleichungen verweisen wir auf die Bücher von Smoller [56], LeVeque [43] und Kröner [38].

2.4 Finite Volumen Verfahren in allgemeiner Form

In diesem Abschnitt leiten wir ein Finite Volumen Verfahren zur Lösung der hyperbolischen Differentialgleichung auf unstrukturierten Gittern für glatte $\mathbf{w}$ her:
Es gilt nach Integration von (1.1) für alle $T_j \in \mathcal{T}$

$$\frac{\partial}{\partial t}\int\limits_{T_j} \mathbf{w} = -\int\limits_{T_j} (\frac{\partial}{\partial x} f_1(\mathbf{w}) + \frac{\partial}{\partial y} f_2(\mathbf{w}) + \frac{\partial}{\partial z} f_3(\mathbf{w}))\,.$$

Mit

$$w_j(t) := \frac{1}{\mid T_j \mid}\int\limits_{T_j} \mathbf{w}(\mathbf{x},t) \quad \text{(Mittelung von } w \text{ auf } T_j)$$

erhalten wir

$$\frac{\partial}{\partial t} w_j(t) = -\frac{1}{\mid T_j \mid}\int\limits_{T_j} (\frac{\partial}{\partial x} f_1(\mathbf{w}) + \frac{\partial}{\partial y} f_2(\mathbf{w}) + \frac{\partial}{\partial z} f_3(\mathbf{w}))\,.$$

Nach dem Integralsatz von Gauss gilt

$$\frac{\partial}{\partial t} w_j(t) = -\frac{1}{\mid T_j \mid}\sum_{l=1}^{N}\int\limits_{S_{jl}} (n_{jl}^x f_1(\mathbf{w}) + n_{jl}^y f_2(\mathbf{w}) + n_{jl}^z f_3(\mathbf{w}))\,. \tag{2.9}$$

Nun wollen wir das Randintegral in Abhängigkeit der Werte in den Nachbarzellen approximieren. Dazu benutzen wir die

DEFINITION 2.4.1 ***(numerische Flussfunktion)***
Sei $f \in C^1(\mathbb{R}^m)$. Eine Funktion $g_{jl} \in C^{0,1}(\mathbb{R}^m \times \mathbb{R}^m)$ heisst numerische Flussfunktion, wenn sie folgende Bedingungen erfüllt:
1) ***Lipschitz-Stetigkeit:*** *Für $g_{jl}, l = 1, \ldots, N$ nehmen wir an, dass für irgendein $R > 0$ und für alle $u, v, u', v' \in B_R(0)$ gilt (dabei ist $B_R(0)$ eine Kugel mit Radius R um 0) :*

$$\mid g_{jl}(u,v) - g_{jl}(u',v') \mid \leq c(R)\ \mid S_{jl} \mid\ (\mid u - u' \mid + \mid v - v' \mid)\,. \tag{2.10}$$

2) ***Konsistenz:***

$$g_{jl}(u,u) = \mathbf{f}(u)\cdot\nu_{jl} \quad \text{, wobei} \quad \mathbf{f}(u) := \begin{pmatrix} f_1(u) \\ f_2(u) \\ f_3(u) \end{pmatrix}\,. \tag{2.11}$$

3) ***Erhaltungseigenschaft:*** *Für $T_{jl} = T_k$ und $T_{km} = T_j$ gelte*

$$g_{jl}(u,v) = -g_{km}(v,u)\,. \tag{2.12}$$

Sei nun $g_{jl}(u,v)$ eine Approximation von $\int\limits_{S_{jl}} \mathbf{f}(u)\cdot n_{jl}$.

Insgesamt gilt also

$$\frac{\partial}{\partial t}w_j(t) \approx -\frac{1}{\mid T_j \mid}\sum_{l=1}^{N} g_{jl}(w_j, w_{jl})\,.$$

Nun wollen wir die Zeitableitung durch einen geeigneten Ausdruck approximieren: Für eine glatte Funktion w liefert die Taylorentwicklung von $w(t^{n+1}) = w(t^n + \Delta t)$ durch $w(t^n)$

$$w(t^{n+1}) = w(t^n) + \Delta t \frac{\partial}{\partial t}w(t)\Big|_{t^n} + \mathcal{O}(\Delta t^2)$$

$$\Rightarrow \frac{\partial}{\partial t}w(t)\Big|_{t^n} = \frac{w(t^{n+1}) - w(t^n)}{\Delta t} + \mathcal{O}(\Delta t)\,. \tag{2.13}$$

Somit erhalten wir als Approximation erster Ordnung

$$w(t^{n+1}) \approx w(t^n) + \Delta t \frac{\partial}{\partial t}w(t)\Big|_{t^n}\,.$$

Im diskreten Fall schreiben wir

$$\partial_t w_j(t) = \frac{w_j^{n+1} - w_j^n}{\Delta t^n} + \mathcal{O}(\Delta t)\,. \tag{2.14}$$

Damit ergibt sich

DEFINITION 2.4.2 ***(Explizites Finite Volumen Verfahren erster Ordnung)***
Für gegebene Anfangswerte $w_0 \in L^\infty(\mathbb{R}^3)$ sei w_j^n definiert durch das folgende numerische Schema:

$$w_j^0 := \frac{1}{\mid T_j \mid}\int_{T_j} w_0(x)dx$$

$$w_j^{n+1} := w_j^n - \frac{\Delta t^n}{\mid T_j \mid}\sum_{l=1}^{N} g_{jl}(w_j^n, w_{jl}^n)\,.$$

DEFINITION 2.4.3 ***(CFL-Bedingung)***
Die Zeitschrittweite wird kontrolliert durch die Courant-Friedrichs-Lewy-Bedingung (CFL) [19], [43]

$$\frac{\Delta t^n}{h} sup_s \mid f'(s) \mid < \kappa \text{ für } s \text{ im Wertebereich der Lösung} \tag{2.15}$$

mit $\kappa < 1$. κ nennen wir auch die ***CFL-Zahl****.*

2.5 Verfahren höherer Ordnung in der Zeit

In Gleichung (2.14) und somit auch in Definition 2.4.2 wurde eine Approximation erster Ordnung benutzt und wir haben nach Einsetzen der diskreten Werte

$$u^{n+1} := u^n - \Delta t G(u^n)$$

erhalten. Dabei ist $G(u)$ eine diskrete Approximation des Divergenzoperators aus Gleichung (1.1). Mit Hilfe von Runge-Kutta-Verfahren ist es möglich, Zeitintegrationen höherer Ordnung zu entwickeln. In [55] werden diese hergeleitet. Im Wesentlichen steckt nur Taylorentwicklung und Koeffizientenvergleich dahinter.
Eine Approximation zweiter Ordnung erhält man durch das folgende Verfahren:

$$\begin{aligned} u^{n+\frac{1}{2}} &:= u^n - \Delta t G(u^n) \\ u^{n+1} &:= \tfrac{1}{2}u^n + \tfrac{1}{2}u^{n+\frac{1}{2}} - \tfrac{1}{2}\Delta t G(u^{n+\frac{1}{2}})\,. \end{aligned} \tag{2.16}$$

Eine Approximation dritter Ordnung erhält man durch das folgende Verfahren:

$$\begin{aligned} u^{n+\frac{1}{3}} &:= u^n - \Delta t G(u^n) \\ u^{n+\frac{2}{3}} &:= \frac{3}{4}u^n + \frac{1}{4}u^{n+\frac{1}{3}} - \frac{1}{4}\Delta t G(u^{n+\frac{1}{3}}) \\ u^{n+1} &:= \frac{1}{3}u^n + \frac{2}{3}u^{n+\frac{2}{3}} - \frac{2}{3}\Delta t G(u^{n+\frac{2}{3}})\,. \end{aligned} \tag{2.17}$$

Somit ergibt sich

DEFINITION 2.5.1 ***(Explizites Finite Volumen Verfahren erster Ordnung im Ort und zweiter Ordnung in der Zeit)***
Für gegebene Anfangswerte $w_0 \in L^\infty(\mathbb{R}^3)$ sei w_j^n definiert durch das folgende numerische Schema:

$$w_j^0 := \frac{1}{\mid T_j \mid} \int_{T_j} w_0(x)dx$$

$$w_j^{n+\frac{1}{2}} := w_j^n - \frac{\Delta t^n}{\mid T_j \mid} \sum_{l=1}^{N} g_{jl}(w_j^n, w_{jl}^n)$$

$$w_j^{n+1} := \frac{1}{2}w_j^n + \frac{1}{2}w_j^{n+\frac{1}{2}} - \frac{1}{2}\frac{\Delta t^n}{\mid T_j \mid} \sum_{l=1}^{N} g_{jl}(w_j^{n+\frac{1}{2}}, w_{jl}^{n+\frac{1}{2}})$$

und

DEFINITION 2.5.2 ***(Explizites Finite Volumen Verfahren erster Ordnung im Ort und dritter Ordnung in der Zeit)***
Für gegebene Anfangswerte $w_0 \in L^\infty(\mathbb{R}^3)$ sei w_j^n definiert durch das folgende numerische Schema:

$$w_j^0 := \frac{1}{\mid T_j \mid} \int_{T_j} w_0(x)dx$$

$$w_j^{n+\frac{1}{3}} := w_j^n - \frac{\Delta t^n}{\mid T_j \mid} \sum_{l=1}^{N} g_{jl}(w_j^n, w_{jl}^n)$$

$$w_j^{n+\frac{2}{3}} := \frac{3}{4}w_j^n + \frac{1}{4}w_j^{n+\frac{1}{3}} - \frac{1}{4}\frac{\Delta t^n}{\mid T_j \mid} \sum_{l=1}^{N} g_{jl}(w_j^{n+\frac{1}{3}}, w_{jl}^{n+\frac{1}{3}})$$

$$w_j^{n+1} := \frac{1}{3}w_j^n + \frac{2}{3}w_j^{n+\frac{2}{3}} - \frac{2}{3}\frac{\Delta t^n}{\mid T_j \mid} \sum_{l=1}^{N} g_{jl}(w_j^{n+\frac{2}{3}}, w_{jl}^{n+\frac{2}{3}}) \,.$$

Kapitel 3

Spezielle numerische Flussfunktionen

In diesem Kapitel werden zunächst numerische Flussfunktionen für skalare Probleme, die zur Berechnung des numerischen Transports bei der Visualisierungsmethode (siehe Kapitel 8) benutzt werden, vorgestellt. Weiterhin wird ein spezielles hyperbolisches System, die Euler-Gleichungen der Gasdynamik, vorgestellt. Für diese werden ebenfalls numerische Flussfunktionen angegeben.

3.1 Numerische Flussfunktionen für skalare Gleichungen

Hier seien die in dieser Arbeit verwendeten numerischen Flussfunktionen für skalare Probleme erwähnt:
Der erste verwendete numerische Fluss ist das Verfahren von **Lax-Friedrichs** [38]:

$$g_{jl}^{LF}(w_j, w_{jl}) = \mid S_{jl} \mid \frac{1}{2}[n_{jl}\mathbf{f}(w_j) + n_{jl}\mathbf{f}(w_{jl}) - \frac{1}{\lambda_{jl}}(w_{jl} - w_j)]\,.$$

Die Konstante erfüllt die folgende Bedingung

$$\lambda_{jl} = \lambda_{lj} > \tilde{c} > 0\ ,\ \lambda_{jl}\text{sup}_u(n_{jl}\mathbf{f}'(u)) \leq 1.$$

Die Zeitschrittweite des numerischen Verfahren wird durch die CFL-Bedingung begrenzt:

$$\text{sup}_j \frac{\Delta t}{\mid T_j \mid} \sum_{l=1}^{N} (\frac{\mid S_{jl} \mid}{2}(n_{jl}\mathbf{f}'(w_j) + \frac{1}{\lambda_{jl}})) < \kappa$$

mit $\kappa < 1$.
Der zweite verwendete numerische Fluss ist das Verfahren von **Engquist-Osher** [24],[38]:

Sei

$$c_{jl}(u) := n_{jl}\mathbf{f}(u)$$

und

$$c_{jl}^{+}(u) := c_{jl}(0) + \int_0^u \max(c'_{jl}(s), 0)ds \,,\; c_{jl}^{-}(u) := \int_0^u \min(c'_{jl}(s), 0)ds \,.$$

Dann ist der Engquist-Osher Fluss folgendermaßen definiert:

$$g_{jl}^{EO}(w_j, w_{jl}) = \mid S_{jl} \mid [c_{jl}^{+}(w_j) + c_{jl}^{-}(w_{jl})].$$

Die Zeitschrittweite des numerischen Verfahren wird durch die CFL-Bedingung begrenzt:

$$\sup_j \frac{\Delta t}{\mid T_j \mid} \sum_{l=1}^{N} \max(\mid S_{jl} \mid n_{jl}\mathbf{f}'(w_j), 0) < \kappa$$

mit $\kappa < 1$.

3.2 Euler-Gleichungen

Wir wollen in aller Kürze die Euler-Gleichungen motivieren. Die in diesem und späteren Kapiteln vorkommenden Bezeichner sind in Tabelle 3.1 angegeben.
Zunächst betrachten wir die Massenerhaltung

$$\partial_t \rho + \text{div}\,(\rho\vec{u}) = \partial_t \rho + \partial_x(\rho u_1) + \partial_y(\rho u_2) + \partial_z(\rho u_3) = 0\,.$$

Die drei Gleichungen für die Impulserhaltung in 3D sind

$$\partial_t(\rho u_1) + \partial_x(\rho u_1^2 + p) + \partial_y(\rho u_1 u_2) + \partial_z(\rho u_1 u_3) = 0\,,$$

$$\partial_t(\rho u_2) + \partial_x(\rho u_2 u_1) + \partial_y(\rho u_2^2 + p) + \partial_z(\rho u_2 u_3) = 0$$

und

$$\partial_t(\rho u_3) + \partial_x(\rho u_3 u_1) + \partial_y(\rho u_3 u_2) + \partial_z(\rho u_3^2 + p) = 0$$

Die Gleichung für die Energieerhaltung lautet

$$\partial_t e + \partial_x(u_1(p+e)) + \partial_y(u_2(p+e)) + \partial_z(u_3(p+e)) = 0\,.$$

ρ	Dichte
u_1	Geschwindigkeit in x-Richtung
u_2	Geschwindigkeit in y-Richtung
u_3	Geschwindigkeit in z-Richtung
$\vec{u} = (u_1, u_2, u_3)^t$	Geschwindigkeitsvektor
p	Druck
$\sigma = \rho u_1$	x-Impuls
$\tau = \rho u_2$	y-Impuls
$\varpi = \rho u_3$	z-Impuls
e	Energiedichte
c	Schallgeschwindigkeit
m	Machzahl
H	Enthalpie
γ	adiabatische Konstante

Tabelle 3.1: Bezeichner der Euler-Gleichungen.

Diese fünf partiellen Differentialgleichungen heißen Euler-Gleichungen der Gasdynamik. Die sechste Gleichung ist die Zustandsgleichung und lautet

$$p = p(\rho, u_1, u_2, u_3, e)\,.$$

Wenn diese fünf Gleichungen auftreten, spricht man vom nicht-isentropischen Fall, den wir in dieser Arbeit betrachten.

In Vektorschreibweise heißt dies also in Verbindung mit (1.1)

$$\mathbf{w} = \begin{pmatrix} \rho \\ \sigma \\ \tau \\ \varpi \\ e \end{pmatrix} = \begin{pmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ e \end{pmatrix}\,,$$

$$f_1(\mathbf{w}) = \begin{pmatrix} \rho u_1 \\ \rho u_1^2 + p \\ \rho u_1 u_2 \\ \rho u_1 u_3 \\ (e+p)u_1 \end{pmatrix}, f_2(\mathbf{w}) = \begin{pmatrix} \rho u_2 \\ \rho u_2 u_1 \\ \rho u_2^2 + p \\ \rho u_2 u_3 \\ (e+p)u_2 \end{pmatrix}, f_3(\mathbf{w}) = \begin{pmatrix} \rho u_3 \\ \rho u_3 u_1 \\ \rho u_3 u_2 \\ \rho u_3^2 + p \\ (e+p)u_3 \end{pmatrix}.$$

Die Zustandsgleichung für ein ideales polytropes Gas lautet

$$p = (\gamma - 1)\left(e - \frac{\rho}{2}(u_1^2 + u_2^2 + u_3^2)\right).$$

Dabei ist γ eine Konstante, für die $2 > \gamma > 1$ gilt. Bei allen folgenden Beispielen ist die Konstante $\gamma = 1.4$ gewählt. Dies entspricht dem Wert für ein 2-atomiges Gas. Da Luft im Wesentlichen aus Stickstoff (N_2) und Sauerstoff (O_2) besteht, gilt dieser Wert also auch für Luft.
Die weiteren auftretenden Größen sind die Schallgeschwindigkeit

$$c := \sqrt{\frac{\gamma p}{\rho}},$$

die Enthalpie

$$H := \frac{e+p}{\rho}$$

und die Machzahl

$$m := \frac{\sqrt{u_1^2 + u_2^2 + u_3^2}}{c}.$$

Eine ausführliche Erläuterung der Euler-Gleichungen mit den physikalischen Hintergründen kann man z.B. im Buch von LeVeque [43] nachlesen. Die Variablen $\rho, \sigma, \tau, \varpi, e$ heissen die konservativen Variablen. Die Variablen ρ, u_1, u_2, u_3, p heissen die primitiven Variablen.

Jacobimatrix, Eigenwerte und Eigenvektoren

Die Jacobimatrizen der Euler-Gleichungen lauten folgendermaßen:

$$\partial f_1(\mathbf{w}) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ \frac{\gamma-3}{2}u_1^2 + \frac{\gamma-1}{2}(u_2^2+u_3^2) & (3-\gamma)u_1 & -(\gamma-1)u_2 & -(\gamma-1)u_3 & (\gamma-1) \\ -u_1u_2 & u_2 & u_1 & 0 & 0 \\ -u_1u_3 & u_3 & 0 & u_1 & 0 \\ -\gamma u_1\frac{\varepsilon}{\rho} + (\gamma-1)u_1(u_1^2+u_2^2+u_3^2) & \gamma\frac{\varepsilon}{\rho} - \frac{\gamma-1}{2}(3u_1^2+u_2^2+u_3^2) & -(\gamma-1)u_1u_2 & -(\gamma-1)u_1u_3 & \gamma u_1 \end{pmatrix}.$$

$$\partial f_2(\mathbf{w}) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ -u_1u_2 & u_2 & u_1 & 0 & 0 \\ \frac{\gamma-3}{2}u_2^2 + \frac{\gamma-1}{2}(u_1^2+u_3^2) & -(\gamma-1)u_1 & (3-\gamma)u_2 & -(\gamma-1)u_3 & (\gamma-1) \\ -u_1u_3 & u_3 & 0 & u_1 & 0 \\ -\gamma u_2\frac{\varepsilon}{\rho} + (\gamma-1)u_2(u_1^2+u_2^2+u_3^2) & -(\gamma-1)u_1u_2 & \gamma\frac{\varepsilon}{\rho} - \frac{\gamma-1}{2}(3u_2^2+u_1^2+u_3^2) & -(\gamma-1)u_1u_3 & \gamma u_2 \end{pmatrix}.$$

$$\partial f_3(w) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ -u_1u_2 & u_2 & u_1 & 0 & 0 \\ -u_1u_3 & u_3 & 0 & u_1 & 0 \\ \frac{\gamma-3}{2}u_3^2 + \frac{\gamma-1}{2}(u_1^2+u_2^2) & -(\gamma-1)u_1 & -(\gamma-1)u_2 & (3-\gamma)u_3 & (\gamma-1) \\ -\gamma u_3\frac{\varepsilon}{\rho} + (\gamma-1)u_3(u_1^2+u_2^2+u_3^2) & -(\gamma-1)u_1u_2 & -(\gamma-1)u_1u_3 & \gamma\frac{\varepsilon}{\rho} - \frac{\gamma-1}{2}(3u_2^2+u_1^2+u_3^2) & \gamma u_3 \end{pmatrix}.$$

Die Eigenwerte der Euler-Gleichungen lauten

$$\lambda_1(\mathbf{w}, n) = (n_1 u_1 + n_2 u_2 + n_3 u_3) - c\,, \tag{3.1}$$
$$\lambda_2(\mathbf{w}, n) = \lambda_3(\mathbf{w}, n) = \lambda_4(\mathbf{w}, n) = (n_1 u_1 + n_2 u_2 + n_3 u_3)\,, \tag{3.2}$$
$$\lambda_5(\mathbf{w}, n) = (n_1 u_1 + n_2 u_2 + n_3 u_3) + c\,. \tag{3.3}$$

Die Rechtseigenvektoren, die die Spalten von Q bilden, lauten

$$r_1(\mathbf{w}, n) = \begin{pmatrix} \frac{\rho}{\sqrt{2}c} \\ \frac{\rho}{\sqrt{2}}(u_1/c - n_1) \\ \frac{\rho}{\sqrt{2}}(u_2/c - n_2) \\ \frac{\rho}{\sqrt{2}}(u_3/c - n_3) \\ \frac{\rho}{\sqrt{2}}(-n_1 u_1 - n_2 u_2 - n_3 u_3 + \frac{u_1^2+u_2^2+u_3^2}{2c} + \frac{c}{\gamma-1}) \end{pmatrix},$$

$$r_2(\mathbf{w}, n) = \begin{pmatrix} n_1 \\ n_1 u_1 \\ n_1 u_2 + n_3 \rho \\ n_1 u_3 - n_2 \rho \\ \frac{1}{2} n_1 (u_1^2 + u_2^2 + u_3^2) + \frac{n_3 u_2 - n_2 u_3}{\rho} \end{pmatrix},$$

$$r_3(\mathbf{w}, n) = \begin{pmatrix} n_2 \\ n_2 u_1 - n_3 \rho \\ n_2 u_2 \\ n_2 u_3 + n_1 \rho \\ \frac{1}{2} n_2 (u_1^2 + u_2^2 + u_3^2) + \frac{-n_3 u_1 + n_1 u_3}{\rho} \end{pmatrix},$$

$$r_4(\mathbf{w}, n) = \begin{pmatrix} n_3 \\ n_3 u_1 + n_2 \rho \\ n_3 u_2 - n_1 \rho \\ n_3 u_3 \\ \frac{1}{2} n_3 (u_1^2 + u_2^2 + u_3^2) + \frac{n_2 u_1 - n_1 u_2}{\rho} \end{pmatrix},$$

$$r_5(\mathbf{w}, n) = \begin{pmatrix} \frac{\rho}{\sqrt{2}c} \\ \frac{\rho}{\sqrt{2}}(u_1/c + n_1) \\ \frac{\rho}{\sqrt{2}}(u_2/c + n_2) \\ \frac{\rho}{\sqrt{2}}(u_3/c + n_3) \\ \frac{\rho}{\sqrt{2}}(n_1 u_1 + n_2 u_2 + n_3 u_3 + \frac{u_1^2+u_2^2+u_3^2}{2c} + \frac{c}{\gamma-1}) \end{pmatrix}.$$

Die Linkseigenvektoren, die die Zeilen von Q^{-1} bilden, lauten

$$l_1(\mathbf{w},n)=\begin{pmatrix} n_1+\frac{-n_3u_2+n_2u_3}{\rho}-\frac{n_1(\gamma-1)(u_1^2+u_2^2+u_3^2)}{2c^2} \\ n_2+\frac{n_3u_1-n_1u_3}{\rho}-\frac{n_2(\gamma-1)(u_1^2+u_2^2+u_3^2)}{2c^2} \\ n_3+\frac{-n_2u_1+n_1u_2}{\rho}-\frac{n_3(\gamma-1)(u_1^2+u_2^2+u_3^2)}{2c^2} \\ \frac{-n_1u_1-n_2u_2-n_3u_3+\frac{(\gamma-1)(u_1^2+u_2^2+u_3^2)}{2c}}{\rho\sqrt{2}} \\ \frac{n_1u_1+n_2u_2+n_3u_3+\frac{(\gamma-1)(u_1^2+u_2^2+u_3^2)}{2c}}{\rho\sqrt{2}} \end{pmatrix},$$

$$l_2(\mathbf{w},n)=\begin{pmatrix} n_1u_1\frac{\gamma-1}{c^2} \\ \frac{-n_3}{\rho}+n_2u_1\frac{\gamma-1}{c^2} \\ \frac{n_2}{\rho}+n_3u_1\frac{\gamma-1}{c^2} \\ \frac{n_1-\frac{(\gamma-1)u_1}{c}}{\rho\sqrt{2}} \\ \frac{-n_1-\frac{(\gamma-1)u_1}{c}}{\rho\sqrt{2}} \end{pmatrix},$$

$$l_3(\mathbf{w},n)=\begin{pmatrix} \frac{n_3}{\rho}+n_1u_2\frac{\gamma-1}{c^2} \\ n_2u_2\frac{\gamma-1}{c^2} \\ \frac{-n_1}{\rho}+n_3u_2\frac{\gamma-1}{c^2} \\ \frac{n_2-\frac{(\gamma-1)u_2}{c}}{\rho\sqrt{2}} \\ \frac{-n_2-\frac{(\gamma-1)u_2}{c}}{\rho\sqrt{2}} \end{pmatrix},$$

$$l_4(\mathbf{w},n)=\begin{pmatrix} \frac{-n_2}{\rho}+n_1u_3\frac{\gamma-1}{c^2} \\ \frac{n_1}{\rho}+n_2u_3\frac{\gamma-1}{c^2} \\ n_3u_3\frac{\gamma-1}{c^2} \\ \frac{n_3-\frac{(\gamma-1)u_2}{c}}{\rho\sqrt{2}} \\ \frac{-n_3-\frac{(\gamma-1)u_2}{c}}{\rho\sqrt{2}} \end{pmatrix},$$

$$l_5(\mathbf{w},n)=\begin{pmatrix} \frac{-(\gamma-1)n_1}{c^2} \\ \frac{-(\gamma-1)n_2}{c^2} \\ \frac{-(\gamma-1)n_3}{c^2} \\ \frac{(\gamma-1)}{\rho c\sqrt{2}} \\ \frac{(\gamma-1)}{\rho c\sqrt{2}} \end{pmatrix}.$$

3.3 Rotationsinvarianz der Euler-Gleichungen

2D:
In Gleichung (2.9) ist das Integral

$$\int_{S_{jl}} (n_1 f_1(\mathbf{w}) + n_2 f_2(\mathbf{w}))$$

auszuwerten. Um die Rechnung zu vereinfachen, kann man die Rotationsinvarianz der Euler-Gleichungen ausnutzen. Es gilt

$$f_1(\mathbf{w})n_1 + f_2(\mathbf{w})n_2 = T^{-1}(\vec{n}) \cdot f_1(T(\vec{n})\mathbf{w})$$

mit

$$T(\vec{n}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & n_1 & n_2 & 0 \\ 0 & -n_2 & n_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} .$$

Hier sei bemerkt, dass $\vec{n} = (n_1, n_2) = (\cos(\alpha), \sin(\alpha))$.
Dabei ist α der Winkel zwischen x_1 und der Normalen $\vec{n}$ (siehe Abb. 3.1).

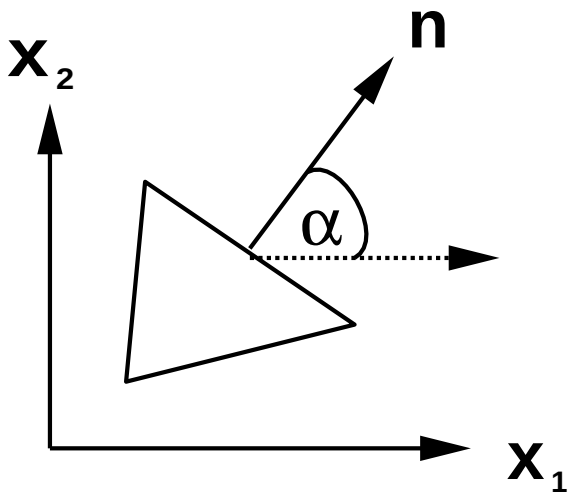

Abbildung 3.1: Ausnutzen der Rotationsinvarianz der Euler-Gleichungen

3D:
In Gleichung (2.9) ist das Integral

$$\int_{S_{jl}} (n_1 f_1(\mathbf{w}) + n_2 f_2(\mathbf{w}) + n_3 f_3(\mathbf{w}))$$

auszuwerten. Um die Rechnung zu vereinfachen, kann man wiederum die Rotationsinvarianz der Euler-Gleichungen ausnutzen. Es gilt

$$f_1(\mathbf{w})n_1 + f_2(\mathbf{w})n_2 + f_3(\mathbf{w})n_3 = T^{-1}(\vec{n}) \cdot f_1(T(\vec{n})\mathbf{w})$$

mit

$$T(\vec{n}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos(\alpha)\cos(\beta) & \sin(\alpha)\cos(\beta) & \sin(\beta) & 0 \\ 0 & -\sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & -\cos(\alpha)\sin(\beta) & -\sin(\alpha)\sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} .$$

Hier sei bemerkt, dass $(n_1, n_2, n_3) = (\cos(\alpha)\cos(\beta), \sin(\alpha)\cos(\beta), \sin(\beta))$.
Es gilt in 2D und 3D natürlich $det(T) = 1$ und $T^{-1}(\alpha, \beta) = T^t(\alpha, \beta)$.
Der Beweis folgt durch einfaches Einsetzen und Nachrechnen. So können die mehrdimensionalen Euler-Gleichungen zu einem eindimensionalem System transformiert werden.

3.4 Numerische Flussfunktionen für Systeme

Hier seien die in dieser Arbeit verwendeten numerischen Flussfunktionen erwähnt. Für die Herleitung der Verfahren siehe Kröner [38].
Lax-Friedrichs [38]:

$$g_{jl}^{LF}(w_j, w_{jl}) =\mid S_{jl} \mid \frac{1}{2}[n_{jl}\mathbf{f}(w_j) + n_{jl}\mathbf{f}(w_{jl}) - \alpha_{jl}(w_{jl} - w_j)] .$$

Dabei ist α_{jl} so definiert, dass

$$\infty > \tilde{c} \geq \alpha_{jl} \geq \lambda_{max} .$$

Hier ist λ_{max} der betragsmäßig größte Eigenwert der Jacobimatrix $(\partial\mathbf{f}(w))$ in der Umgebung einer Kante S_{jl}.
Steger-Warming [61],[38]:

$$g_{jl}^{SW}(w_j, w_{jl}) =\mid S_{jl} \mid [C_{jl}^{+}(w_j)w_j + C_{jl}^{-}(w_{jl})w_{jl}] .$$

Vijayasundaram [69],[38]:

$$g_{jl}^{V}(w_j, w_{jl}) =\mid S_{jl} \mid [C_{jl}^{+}(\frac{w_j + w_{jl}}{2})w_j + C_{jl}^{-}(\frac{w_j + w_{jl}}{2})w_{jl}] .$$

Osher-Solomon [4],[58],[59]:

$$g_{jl}^{OS}(w_j, w_{jl}) :=\mid S_{jl} \mid \left(T^{-1}(n_{jl}) \cdot g_{jl}^{OS*}(T(n_{jl})w_j, T(n_{jl})w_{jl})\right)$$

mit

$$g_{jl}^{OS*}(w_j, w_{jl}) := f_1(w_j) + \int_{\Gamma^{jl}} (\partial f_1(w))^- dw \,. \tag{3.4}$$

Dabei verbindet ein bestimmter Pfad Γ^{jl} die Punkte w_j und w_{jl} in eindeutiger Weise. Die Definition und Herleitung des Verfahren kann man in [4] nachlesen.
Der Zeitschritt wird aufgrund der CFL-Bedingung durch folgende Beziehung kontrolliert:

$$\Delta t^n := \kappa \cdot \min_{\{T_j \in \mathcal{T}\}} \left\{ \frac{\mid T_j \mid}{\max\limits_{l=1,..,N} \max\limits_{k=1,\cdots,5} \mid \lambda_k(w_j, n_{jl}) \mid \cdot \max\limits_{l=1,..,N} \mid S_{jl} \mid} \right\} , \tag{3.5}$$

wobei $\kappa < 1$ die CFL-Zahl ist.

3.5 Implementierung von Randbedingungen

In diesem Abschnitt sollen die verschiedenen Randbedingungen behandelt werden. Das Gebiet sei Ω. Den Rand von Ω bezeichnen wir mit $\partial\Omega$. Im Inneren des Gebietes wird der numerische Fluss zwischen dem Element T_j und dem Element T_{jl} wie oben beschrieben mit der Flussfunktion $g_{jl}(w_j, w_{jl})$ ausgewertet. Dabei sind $w_j = (\rho_j, \sigma_j, \tau_j, \varpi_j, e_j)^T$ konstant auf T_j und $w_{jl} = (\rho_{jl}, \sigma_{jl}, \tau_{jl}, \varpi_{jl}, e_{jl})^T$ konstant auf T_{jl}. Wenn gilt $T_j \cap \partial\Omega \neq \emptyset$ und $T_j \cap \partial\Omega$ mehr als einen Punkt umfasst, so ist diese Kante von T_j eine Randkante.
Die verschiedenen Randbedingungen lauten:

- Einflussbedingung: Zu jedem T_j mit $\bar{T}_j \cap \partial\Omega \neq 0$ nehme eine Zelle $\tilde{T}_j$ zum Gebiet hinzu, die außerhalb des Gebietes liegt und eine gemeinsame Kante mit der Zelle T_j hat. Diese Kante ist S_{jl}. Diese Zelle bezeichnen wir auch als Ghostzelle. In dieser Zelle gelten zur Zeit t die Werte

  $$w_{ein}(t) = (\rho_{ein}(t), \sigma_{ein}(t), \tau_{ein}(t), \varpi_{ein}(t), e_{ein}(t))^T \,.$$

  Dann wertet man den numerischen Fluss über diese Einflusskante mit der Flussfunktion $g_{jl}(w_j, w_{ein}(t))$ aus. Bei unseren Beispielen gelten in der Ghostzelle für alle Zeiten t dieselben festen Werte $w_{ein} = (\rho_{ein}, \sigma_{ein}, \tau_{ein}, \varpi_{ein}, e_{ein})^T$.

- Ausflussbedingung: Man nehme wie oben wieder zu T_j eine Ghostzelle $\tilde{T}_j$ zum Gebiet hinzu. Definiere $w_{aus}(t) := w_j(t)$. Dann lautet der Fluss $g_{jl}(w_j, w_{aus}) = \nu_{jl} \cdot \mathbf{f}(w_j)$.

- Reflektionsbedingung: Man nehme wieder eine Ghostzelle zum Gebiet hinzu. Man berechnet in Abhängigkeit der äußeren Normalen die aus den Geschwindigkeitskomponenten resultierende Geschwindigkeit bzw. Impuls $\vartheta_j = n_1 \cdot \sigma_j + n_2 \cdot \tau_j + n_3 \cdot \varpi_j$. Daraus ergeben sich in der Ghostzelle die Werte

  $$w_{ref} = (\rho_j, \sigma_j - 2 \cdot \vartheta_j \cdot n_1, \tau_j - 2 \cdot \vartheta_j \cdot n_2, \varpi_j - 2 \cdot \vartheta_j \cdot n_3, e_j)^T \,.$$

Wenn für eine Randfläche $x = const$ gilt, so reduziert sich dies zu

$$w_{ref} = (\rho_j, -\sigma_j, \tau_j, \varpi_j, e_j)^T .$$

Wenn für eine Randfläche $y = const$ gilt, so reduziert sich dies zu

$$w_{ref} = (\rho_j, \sigma_j, -\tau_j, \varpi_j, e_j)^T .$$

Wenn für eine Randfläche $z = const$ gilt, so reduziert sich dies zu

$$w_{ref} = (\rho_j, \sigma_j, \tau_j, -\varpi_j, e_j)^T .$$

Dann lautet der Fluss $g_{jl}(w_j, w_{ref})$.

- Undurchlässige Wand: Für den Fluss über die Randkante hat man die Bedingung $\vec{n} \cdot \vec{u} = n_1 u_1 + n_2 u_2 + n_3 u_3 = 0$. Dadurch wird eine undurchlässige Wand simuliert. Hier wertet man den Fluss sofort aus. Sei nun $S_{jl} = T_j \cap \partial\Omega$. Dann berechnet sich der Fluss folgendermaßen:

$$\int_{S_{jl}} (n_1 \cdot f_1 + n_2 \cdot f_2 + n_3 \cdot f_3) ds = \int_{S_{jl}} \begin{pmatrix} 0 \\ n_1 \cdot p \\ n_2 \cdot p \\ n_3 \cdot p \\ 0 \end{pmatrix} ds = \mid S_{jl} \mid \begin{pmatrix} 0 \\ n_1 \cdot p \\ n_2 \cdot p \\ n_3 \cdot p \\ 0 \end{pmatrix} .$$

Kapitel 4

Discontinuous Galerkin Verfahren

Der Inhalt dieses Kapitels ist die Herleitung, Motivation und Anwendung der Discontinuous Galerkin Verfahren. Da in diesem Kapitel die verschiedensten Integrationsformeln auf den verschiedensten Elementen gebraucht werden, sollen diese zunächst in einem Unterkapitel gesammelt werden, damit man den eigentlichen Teil ohne jeweilige Unterbrechung lesen kann.

4.1 Integrationsformeln

Zunächst bezeichnen wir mit

$$P^k := \{p(x) \mid p(x) \text{ ist Polynom mit Grad } \leq k\}$$

den Raum der Polynome vom Grad $\leq k$.

Später werden wir Basisfunktionen benutzen. Dazu bieten sich auf Intervallen die Legendre-Polynome an. Diese sind bezüglich des Intervalls $[-1;1]$ orthogonal (siehe Stoer [62]).

Die Beziehung für die Legendre Polynome lautet

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}((x^2-1)^n) \quad , n = 0, 1, 2, \ldots .$$

$P_n(x)$ ist ein Polynom n-ten Grades und hat genau n einfache Nullstellen im Intervall $-1 < x < 1$.

Die ersten Legendre Polynome lauten:

$$
\begin{aligned}
P_0(x) &= 1 \\
P_1(x) &= x \\
P_2(x) &= x^2 - \tfrac{1}{3} \\
P_3(x) &= x^3 - \tfrac{3}{5}x \\
P_4(x) &= x^4 - \tfrac{6}{7}x^2 + \tfrac{3}{35} \\
P_5(x) &= x^5 - \tfrac{10}{9}x^3 + \tfrac{5}{21}x \\
P_6(x) &= x^6 - \tfrac{315}{231}x^4 + \tfrac{105}{231}x^2 - \tfrac{5}{231} \\
P_7(x) &= x^7 - \tfrac{693}{429}x^5 + \tfrac{315}{429}x^3 - \tfrac{35}{429}x \ .
\end{aligned}
$$

Die Nullstellen x_i der Legendre-Polynome sind gerade die Stützstellen der Gauss-Quadraturregeln. Für Polynome $p(x) \in P^k$ gilt:

$$\int_{-1}^{1} p(x) = \sum_i \omega_i p(x_i) \text{ für Polynome } p(x) \in P^k \,.$$

exakt für $p(x) \in P^k$	Integrationspunkte x_i	Gewichte ω_i
$k = 1$, Nullstelle von $P_1(x)$	$x_1 = 0.0$	$\omega_1 = 2.0$
$k = 3$, Nullstellen von $P_2(x)$	$x_1 = -\frac{1}{\sqrt{3}} \approx -0.5773502691896$	$\omega_1 = 1.0$
	$x_2 = +\frac{1}{\sqrt{3}} \approx +0.5773502691896$	$\omega_2 = 1.0$
$k = 5$, Nullstellen von $P_3(x)$	$x_1 = -\sqrt{\frac{3}{5}} \approx -0.7745966692415$	$\omega_1 = \frac{5}{9}$
	$x_2 = 0.0$	$\omega_2 = \frac{8}{9}$
	$x_3 = +\sqrt{\frac{3}{5}} \approx +0.7745966692415$	$\omega_3 = \frac{5}{9}$
$k = 7$, Nullstellen von $P_4(x)$	$x_1 \approx -0.8611363115941$	$\omega_1 \approx 0.3478548451$
	$x_2 \approx -0.3399810436849$	$\omega_2 \approx 0.6521451549$
	$x_3 \approx +0.3399810436849$	$\omega_3 \approx 0.6521451549$
	$x_4 \approx +0.8611363115941$	$\omega_4 \approx 0.3478548451$
$k = 9$, Nullstellen von $P_5(x)$	$x_1 \approx -0.906179845938664$	$\omega_1 \approx 0.2369268851$
	$x_2 \approx -0.538469310105683$	$\omega_2 \approx 0.4786286705$
	$x_3 = 0.0$	$\omega_3 \approx 0.5688888889$
	$x_4 \approx +0.538469310105683$	$\omega_4 \approx 0.4786286705$
	$x_5 \approx +0.906179845938664$	$\omega_5 \approx 0.2369268851$

$k = 11$,Nullstellen von $P_6(x)$	$x_1 \approx -0.9324695142$	$\omega_1 \approx 0.1713244924$
	$x_2 \approx -0.6612093847$	$\omega_2 \approx 0.3607615730$
	$x_3 \approx -0.2386191861$	$\omega_3 \approx 0.4679139346$
	$x_4 \approx +0.2386191861$	$\omega_4 \approx 0.4679139346$
	$x_5 \approx +0.6612093847$	$\omega_5 \approx 0.3607615730$
	$x_6 \approx +0.9324695142$	$\omega_6 \approx 0.1713244924$
$k = 13$,Nullstellen von $P_7(x)$	$x_1 \approx -0.9491079123$	$\omega_1 \approx 0.1294849662$
	$x_2 \approx -0.7415311856$	$\omega_2 \approx 0.2797053915$
	$x_3 \approx -0.4058451514$	$\omega_3 \approx 0.3818300505$
	$x_4 = 0.0$	$\omega_4 \approx 0.4179591837$
	$x_5 \approx +0.4058451514$	$\omega_5 \approx 0.3818300505$
	$x_6 \approx +0.7415311856$	$\omega_6 \approx 0.2797053915$
	$x_7 \approx +0.9491079123$	$\omega_7 \approx 0.1294849662$

Tabelle 4.1: Integrationsformeln auf Intervallen. $\int_{-1}^{1} p(x)dx = \sum_i \omega_i p(x_i)$ **für Polynome** $p \in P^k$**.**

Auf Dreiecken in 2D verwenden wir folgende Integrationsformeln:

exakt für $p(x,y) \in P^k$	Integrationspunkte baryzentrische Koordinaten λ_i	Gewichte ω_i
$k=2$	$\lambda_1 = 0.0, \lambda_2 = \frac{1}{2}, \lambda_3 = \frac{1}{2}$ $\lambda_1 = \frac{1}{2}, \lambda_2 = 0.0, \lambda_3 = \frac{1}{2}$ $\lambda_1 = \frac{1}{2}, \lambda_2 = \frac{1}{2}, \lambda_3 = 0.0$	$\omega_1 = \frac{1}{3}$ $\omega_2 = \frac{1}{3}$ $\omega_3 = \frac{1}{3}$
$k=4$	$\lambda_1 = \alpha, \lambda_2 = \beta, \lambda_3 = \beta$ $\lambda_1 = \beta, \lambda_2 = \alpha, \lambda_3 = \beta$ $\lambda_1 = \beta, \lambda_2 = \beta, \lambda_3 = \alpha$ $\lambda_1 = \gamma, \lambda_2 = \delta, \lambda_3 = \delta$ $\lambda_1 = \delta, \lambda_2 = \gamma, \lambda_3 = \delta$ $\lambda_1 = \delta, \lambda_2 = \delta, \lambda_3 = \gamma$ mit $\alpha \approx 0.8168475729804585$ $\beta = \frac{1}{2}(1-\alpha)$ $\gamma \approx 0.1081030181680702$ $\delta = \frac{1}{2}(1-\gamma)$	$\omega_1 = \mu$ $\omega_2 = \mu$ $\omega_3 = \mu$ $\omega_4 = \frac{1}{3} - \mu$ $\omega_5 = \frac{1}{3} - \mu$ $\omega_6 = \frac{1}{3} - \mu$ mit $\mu \approx 0.1099517436553219$

Tabelle 4.2: Integrationsformeln auf Dreiecken. $\int_T p(x,y)dxdy = \mid T \mid \sum_i \omega_i p(\lambda)$ **für Polynome** $p \in P^k$**.**

Auf dem Quadrat $[-1;1]^2$ in 2D verwenden wir folgende Integrationsformeln: Für Polynome $p(x,y) \in P^k$ gilt:

$$\int_{-1}^{1}\int_{-1}^{1} p(x,y) = \sum_i \omega_i p(x_i,y_i) \text{ für Polynome } p(x,y) \in P^k \,.$$

exakt für $p(x,y) \in P^k$	Integrationspunkte x_i, y_i	Gewichte ω_i
$k=1$	$x_1 = 0.0\,,\, y_1 = 0.0$	$\omega_1 = 4.0$
$k=3$	$x_1 = -\frac{1}{\sqrt{3}}\,,\, y_1 = -\frac{1}{\sqrt{3}}$	$\omega_1 = 1.0$
	$x_2 = -\frac{1}{\sqrt{3}}\,,\, y_2 = +\frac{1}{\sqrt{3}}$	$\omega_2 = 1.0$
	$x_3 = +\frac{1}{\sqrt{3}}\,,\, y_3 = -\frac{1}{\sqrt{3}}$	$\omega_3 = 1.0$
	$x_4 = +\frac{1}{\sqrt{3}}\,,\, y_4 = +\frac{1}{\sqrt{3}}$	$\omega_4 = 1.0$
$k=5$	$x_1 = -\sqrt{\frac{3}{5}}, y_1 = -\sqrt{\frac{3}{5}}$	$\omega_1 = \frac{25}{81}$
	$x_2 = -\sqrt{\frac{3}{5}}, y_2 = 0.0$	$\omega_2 = \frac{40}{81}$
	$x_3 = -\sqrt{\frac{3}{5}}, y_3 = +\sqrt{\frac{3}{5}}$	$\omega_3 = \frac{25}{81}$
	$x_4 = 0.0, y_4 = -\sqrt{\frac{3}{5}}$	$\omega_4 = \frac{40}{81}$
	$x_5 = 0.0, y_5 = 0.0$	$\omega_5 = \frac{64}{81}$
	$x_6 = 0.0, y_6 = +\sqrt{\frac{3}{5}}$	$\omega_6 = \frac{40}{81}$
	$x_7 = +\sqrt{\frac{3}{5}}, y_7 = -\sqrt{\frac{3}{5}}$	$\omega_7 = \frac{25}{81}$
	$x_8 = +\sqrt{\frac{3}{5}}, y_8 = 0.0$	$\omega_8 = \frac{40}{81}$
	$x_9 = +\sqrt{\frac{3}{5}}, y_9 = +\sqrt{\frac{3}{5}}$	$\omega_9 = \frac{25}{81}$

Tabelle 4.3: Integrationsformeln auf Quadraten. $\int_{-1}^{1}\int_{-1}^{1} p(x,y)dxdy = \sum_i \omega_i p(x_i,y_i)$ **für Polynome** $p \in P^k$**.**

Auf Tetraedern in 3D verwenden wir folgende Integrationsformeln:

exakt für $p(x,y,z) \in P^k$	Integrationspunkte baryzentrische Koordinaten λ_i	Gewichte ω_i
$k = 2$	$\lambda_1 = \alpha$, $\lambda_2 = \beta$, $\lambda_3 = \beta$, $\lambda_4 = \beta$	$\omega_1 = \frac{1}{4}$
	$\lambda_1 = \beta$, $\lambda_2 = \alpha$, $\lambda_3 = \beta$, $\lambda_4 = \beta$	$\omega_2 = \frac{1}{4}$
	$\lambda_1 = \beta$, $\lambda_2 = \beta$, $\lambda_3 = \alpha$, $\lambda_4 = \beta$	$\omega_3 = \frac{1}{4}$
	$\lambda_1 = \beta$, $\lambda_2 = \beta$, $\lambda_3 = \beta$, $\lambda_4 = \alpha$	$\omega_4 = \frac{1}{4}$
	mit $\alpha = \frac{5+3\sqrt{5}}{20} \approx 0.58541020$ $\beta = \frac{5-\sqrt{5}}{20} \approx 0.13819660$	

Tabelle 4.4: Integrationsformeln auf Tetraedern. $\int_T p(x,y,z)dxdydz = \mid T \mid \sum_i \omega_i p(\lambda)$ **für Polynome** $p \in P^k$**.**

Auf dem Hexaeder $[-1;1]^3$ in 3D verwenden wir folgende Integrationsformeln: Für Polynome $p(x,y,z) \in P^k$ gilt:

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} p(x,y,z) = \sum_i \omega_i p(x_i,y_i,z_i) \text{ für Polynome } p(x,y,z) \in P^k \,.$$

Tabelle 4.5: Integrationsformeln auf Hexaedern.

Integrationsformeln auf Hexaedern		
exakt für $p(x,y,z) \in P^k$	Integrationspunkte x_i, y_i, z_i	Gewichte ω_i
$k=1$	$x_1 = 0.0\,,\ y_1 = 0.0\,,\ z_1 = 0.0$	$\omega_1 = 8.0$
$k=2$	$x_1 = \frac{1}{\sqrt{1.5}}\,,\ y_1 = 0.0\,,\ z_1 = \frac{1}{\sqrt{3}}$	$\omega_1 = 2.0$
	$x_2 = 0.0\,,\ y_2 = \frac{1}{\sqrt{1.5}}\,,\ z_2 = -\frac{1}{\sqrt{3}}$	$\omega_1 = 2.0$
	$x_3 = -\frac{1}{\sqrt{1.5}}\,,\ y_3 = 0.0\,,\ z_3 = \frac{1}{\sqrt{3}}$	$\omega_1 = 2.0$
	$x_4 = 0.0\,,\ y_4 = -\frac{1}{\sqrt{1.5}}\,,\ z_4 = -\frac{1}{\sqrt{3}}$	$\omega_1 = 2.0$
$k=3$	$x_1 = -\frac{1}{\sqrt{3}}\,,\ y_1 = -\frac{1}{\sqrt{3}}\,,\ z_1 = -\frac{1}{\sqrt{3}}$	$\omega_1 = 1.0$
	$x_2 = -\frac{1}{\sqrt{3}}\,,\ y_2 = +\frac{1}{\sqrt{3}}\,,\ z_2 = -\frac{1}{\sqrt{3}}$	$\omega_2 = 1.0$
	$x_3 = +\frac{1}{\sqrt{3}}\,,\ y_3 = -\frac{1}{\sqrt{3}}\,,\ z_3 = -\frac{1}{\sqrt{3}}$	$\omega_3 = 1.0$
	$x_4 = +\frac{1}{\sqrt{3}}\,,\ y_4 = +\frac{1}{\sqrt{3}}\,,\ z_4 = -\frac{1}{\sqrt{3}}$	$\omega_4 = 1.0$
	$x_5 = -\frac{1}{\sqrt{3}}\,,\ y_5 = -\frac{1}{\sqrt{3}}\,,\ z_5 = +\frac{1}{\sqrt{3}}$	$\omega_5 = 1.0$
	$x_6 = -\frac{1}{\sqrt{3}}\,,\ y_6 = +\frac{1}{\sqrt{3}}\,,\ z_6 = +\frac{1}{\sqrt{3}}$	$\omega_6 = 1.0$
	$x_7 = +\frac{1}{\sqrt{3}}\,,\ y_7 = -\frac{1}{\sqrt{3}}\,,\ z_7 = +\frac{1}{\sqrt{3}}$	$\omega_7 = 1.0$
	$x_8 = +\frac{1}{\sqrt{3}}\,,\ y_8 = +\frac{1}{\sqrt{3}}\,,\ z_8 = +\frac{1}{\sqrt{3}}$	$\omega_8 = 1.0$
$k=5$	$x_1 = -\sqrt{\frac{3}{5}},\ y_1 = -\sqrt{\frac{3}{5}}\,,\ z_1 = -\sqrt{\frac{3}{5}}$	$\omega_1 = \frac{125}{729}$
	$x_2 = -\sqrt{\frac{3}{5}},\ y_2 = 0.0\,,\ z_2 = -\sqrt{\frac{3}{5}}$	$\omega_2 = \frac{200}{729}$
	$x_3 = -\sqrt{\frac{3}{5}},\ y_3 = +\sqrt{\frac{3}{5}}\,,\ z_3 = -\sqrt{\frac{3}{5}}$	$\omega_3 = \frac{125}{729}$
	$x_4 = 0.0,\ y_4 = -\sqrt{\frac{3}{5}}\,,\ z_4 = -\sqrt{\frac{3}{5}}$	$\omega_4 = \frac{200}{729}$
	$x_5 = 0.0,\ y_5 = 0.0\,,\ z_5 = -\sqrt{\frac{3}{5}}$	$\omega_5 = \frac{320}{729}$

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

exakt für $p(x,y,z) \in P^k$	Integrationspunkte x_i, y_i, z_i	Gewichte ω_i
	$x_6 = 0.0, y_6 = +\sqrt{\frac{3}{5}}\,, z_6 = -\sqrt{\frac{3}{5}}$	$\omega_6 = \frac{200}{729}$
	$x_7 = +\sqrt{\frac{3}{5}}, y_7 = -\sqrt{\frac{3}{5}}\,, z_7 = -\sqrt{\frac{3}{5}}$	$\omega_7 = \frac{125}{729}$
	$x_8 = +\sqrt{\frac{3}{5}}, y_8 = 0.0\,, z_8 = -\sqrt{\frac{3}{5}}$	$\omega_8 = \frac{200}{729}$
	$x_9 = +\sqrt{\frac{3}{5}}, y_9 = +\sqrt{\frac{3}{5}}\,, z_9 = -\sqrt{\frac{3}{5}}$	$\omega_9 = \frac{125}{729}$
	$x_{10} = -\sqrt{\frac{3}{5}}, y_{10} = -\sqrt{\frac{3}{5}}\,, z_{10} = 0.0$	$\omega_{10} = \frac{200}{729}$
	$x_{11} = -\sqrt{\frac{3}{5}}, y_{11} = 0.0\,, z_{11} = 0.0$	$\omega_{11} = \frac{320}{729}$
	$x_{12} = -\sqrt{\frac{3}{5}}, y_{12} = +\sqrt{\frac{3}{5}}\,, z_{12} = 0.0$	$\omega_{12} = \frac{200}{729}$
	$x_{13} = 0.0, y_{13} = -\sqrt{\frac{3}{5}}\,, z_{13} = 0.0$	$\omega_{13} = \frac{320}{729}$
	$x_{14} = 0.0, y_{14} = 0.0\,, z_{14} = 0.0$	$\omega_{14} = \frac{512}{729}$
	$x_{15} = 0.0, y_{15} = +\sqrt{\frac{3}{5}}\,, z_{15} = 0.0$	$\omega_{15} = \frac{320}{729}$
	$x_{16} = +\sqrt{\frac{3}{5}}, y_{16} = -\sqrt{\frac{3}{5}}\,, z_{16} = 0.0$	$\omega_{16} = \frac{200}{729}$
	$x_{17} = +\sqrt{\frac{3}{5}}, y_{17} = 0.0\,, z_{17} = 0.0$	$\omega_{17} = \frac{320}{729}$
	$x_{18} = +\sqrt{\frac{3}{5}}, y_{18} = +\sqrt{\frac{3}{5}}\,, z_{18} = 0.0$	$\omega_{18} = \frac{200}{729}$
	$x_{19} = -\sqrt{\frac{3}{5}}, y_{19} = -\sqrt{\frac{3}{5}}, z_{19} = +\sqrt{\frac{3}{5}}$	$\omega_{19} = \frac{125}{729}$
	$x_{20} = -\sqrt{\frac{3}{5}}, y_{20} = 0.0\,, z_{20} = +\sqrt{\frac{3}{5}}$	$\omega_{20} = \frac{200}{729}$
	$x_{21} = -\sqrt{\frac{3}{5}}, y_{21} = +\sqrt{\frac{3}{5}}, z_{21} = +\sqrt{\frac{3}{5}}$	$\omega_{21} = \frac{125}{729}$
	$x_{22} = 0.0, y_{22} = -\sqrt{\frac{3}{5}}, z_{22} = +\sqrt{\frac{3}{5}}$	$\omega_{22} = \frac{200}{729}$
	$x_{23} = 0.0, y_{23} = 0.0\,, z_{23} = +\sqrt{\frac{3}{5}}$	$\omega_{23} = \frac{320}{729}$
	$x_{24} = 0.0, y_{24} = +\sqrt{\frac{3}{5}}, z_{24} = +\sqrt{\frac{3}{5}}$	$\omega_{24} = \frac{200}{729}$
	$x_{25} = +\sqrt{\frac{3}{5}}, y_{25} = -\sqrt{\frac{3}{5}}, z_{25} = +\sqrt{\frac{3}{5}}$	$\omega_{25} = \frac{125}{729}$
	$x_{26} = +\sqrt{\frac{3}{5}}, y_{26} = 0.0\,, z_{26} = +\sqrt{\frac{3}{5}}$	$\omega_{26} = \frac{200}{729}$
	$x_{27} = +\sqrt{\frac{3}{5}}, y_{27} = +\sqrt{\frac{3}{5}}, z_{27} = +\sqrt{\frac{3}{5}}$	$\omega_{27} = \frac{125}{729}$

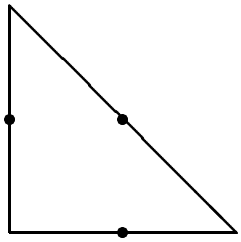

Abbildung 4.1: Quadraturpunkte: Exakt für Polynome $p \in P^2$ auf Dreiecken.

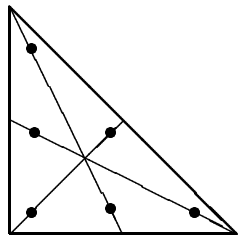

Abbildung 4.2: Quadraturpunkte: Exakt für Polynome $p \in P^4$ auf Dreiecken.

4.2 Notationen

In diesem Abschnitt sind zum besseren Verständnis schon einige Notationen vorweggenommen.

$u(x,y,t)$ $u(x,y,z,t)$	exakte Funktion in 2D exakte Funktion in 3D
u_x,u_y,u_z,u_t	partiellen Ableitungen der exakten Funktion u nach x,y,z,t
$u_h(\mathbf{x},t) = \sum_{i=1}^{n} u_i(t)v_i(\mathbf{x})$	u_h Näherungslösung von u u_i Freiheitsgrad v_i Basisfunktion
$\bar{u}_h := \frac{1}{\mid T \mid} \int_T u_h$	Mittelwert von u_h in Zelle T
$\bar{u}_h^n := \frac{1}{\mid T \mid} \int_T u_h^n$	Mittelwert von u_h^n in Zelle T nach n Zeitschritten
(2D, Quadrate, $(x,y) \in K$) (K eine Zelle des Gebiets) $u_h(x,y,t) = \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y)$	$\overline{u}(t)$ Mittelwert von u_h in K u^x,u^y Freiheitsgrade $\phi_i(x)$,$\psi_j(y)$ Basisfunktionen

Tabelle 4.6: Notationen.

(2D, Quadrate, eine spezielle Zelle (i,j)) $\overline{u}_{i,j}$	 Mittelwert von u_h in Zelle (i,j)
(3D, Hexaeder, $(x,y,z) \in K$) (K eine Zelle des Gebiets) $u_h(x,y,z,t) = \overline{u}(t) + u^x(t)\phi_i(x)$ $+ u^y(t)\psi_j(y) + u^z(t)\xi_k(z)$	 $\overline{u}(t)$ Mittelwert von u_h in K u^x,u^y,u^z Freiheitsgrade $\phi_i(x)$,$\psi_j(y)$,$\xi_k(z)$ Basisfunktionen
(3D, Hexaeder, eine spezielle Zelle (i,j,k)) $\overline{u}_{i,j,k}$	 Mittelwert von u_h in Zelle (i,j,k)
(3D, Hexaeder, eine spezielle Zelle (i,j,k)) $u^x_{i,j,k}$	 Freiheitsgrad u^x von u_h in Zelle (i,j,k)
(1D) u_i^n	 numerische Lösung von u im Gitterpunkt i zum Zeitpunkt t^n

Tabelle 4.7: Notationen.

4.3 MUSCL-Verfahren und Limitierung

Ziel des Abschnitts ist die begriffliche Klärung der Begriffe MUSCL-Verfahren und Limitierung.
Betrachten wir die skalare Erhaltungsgleichung in $1D$

$$u_t + f(u)_x = 0$$

mit geeigneten Anfangsdaten. u_j^n sei die konstante Approximation der exakten Lösung u in der Zelle j zum Zeitpunkt t^n. x_j sei der Mittelpunkt der Zelle j.
Seien die Werte u_{j-1}^n , u_j^n, u_{j+1}^n , u_{j+2}^n zum Zeitpunkt t^n gegeben.
Das Gitter sei äquidistant, d.h. $\Delta x_j = x_{j+1} - x_j = const.$ $\Delta t^n := t_{n+1} - t_n$ bezeichnet die Zeitschrittweite.
In Definition 2.4.2 hatten wir ein explizites Finite Volumen Verfahren erster Ordnung definiert. Hier, für den skalaren Fall, ergibt sich

$$u_j^0 := \frac{1}{\Delta x_j} \int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u_0(x)dx$$

$$u_j^{n+1} := u_j^n - \frac{\Delta t^n}{\Delta x_j}(g(u_j^n, u_{j+1}^n) - g(u_{j-1}^n, u_j^n)) .$$

Unser Ziel ist es nun, aus den stückweise konstanten Werten stückweise lineare Werte zu rekonstruieren. Definieren wir dazu

$$u_{j+\frac{1}{2}}^L = u_j^n + \frac{1}{2}\phi(u_j^n - u_{j-1}^n, u_{j+1}^n - u_j^n)$$

und

$$u_{j+\frac{1}{2}}^R = u_{j+1}^n - \frac{1}{2}\phi(u_{j+1}^n - u_j^n, u_{j+2}^n - u_{j+1}^n) .$$

Die Funktion ϕ im allgemeinen wird **Limiter** genannt. Die folgende spezielle Funktion

$$\phi(x_1, x_2) := \begin{cases} sign(x_1)min(\mid x_1 \mid, \mid x_2 \mid) & \text{, falls } x_1x_2 > 0 \\ 0 & \text{, falls } x_1x_2 \leq 0 \end{cases}$$

ist der sogenannte **minmod**-Limiter.
Seine Aufgabe ist es, bei der Rekonstruktion das Entstehen neuer Extrema zu unterdrücken. Mit den so definierten Zwischenwerten $u_{i+\frac{1}{2}}^L$ und $u_{i+\frac{1}{2}}^R$ können wir nun das sogenannte **MUSCL**-Verfahren, ein Verfahren höherer Ordnung im Ort definieren:

$$\begin{aligned} u_j^0 &:= \frac{1}{\Delta x_j} \int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u_0(x)dx \\ u_j^{n+1} &:= u_j^n - \frac{\Delta t^n}{\Delta x_j}(g(u_{j+\frac{1}{2}}^L, u_{j+\frac{1}{2}}^R) - g(u_{j-\frac{1}{2}}^L, u_{j-\frac{1}{2}}^R)) . \end{aligned} \tag{4.1}$$

Weitere Limiter, die Anwendung in mehreren Raumdimensionen und die Erweiterung auf Systeme kann man in Kröner [38] nachlesen.

4.4 Herleitung der Discontinuous Galerkin Verfahren

In diesem Abschnitt betrachten wir ein Anfangswertproblem. Dieses Problem ist gegeben durch die hyperbolische Differentialgleichung

$$\partial_t u + \text{div}\, \mathbf{f}(u) = 0 \text{ , in } \Omega \times (0,T)\,, \tag{4.2}$$

wobei $\Omega \subset \mathbb{R}^{\mathrm{d}}$ mit
der Anfangsbedingung

$$u(\mathbf{x},0) = u_0(\mathbf{x}) \text{ in } \Omega\,. \tag{4.3}$$

Betrachten wir zur Herleitung des Discontinuous Galerkin Verfahren u zunächst als skalare Funktion. Im Abschnitt über die Limitierung 4.8 werden wir dann zwischen skalarer und vektorwertiger Funktion u unterscheiden.
In einem Verfahren erster Ordnung wird die Lösung u dieser Gleichung durch stückweise konstante Werte approximiert. Bei den verwendeten Verfahren höherer Ordnung wie MUSCL [38],[26] oder ENO [38],[55] wird die Lösung u dieser Gleichung ebenfalls durch stückweise konstante Werte approximiert. Dabei wird aber in jedem Zeitschritt aus dem konstanten Wert in einer Zelle und den dazugehörigen Nachbarzellen ein Polynom höheren Grades rekonstruiert. Dieses Polynom wird dann noch geeignet limitiert und mit diesem dann die numerische Flussberechnung durchgeführt.
Unser Ziel ist es u, statt wie im Verfahren erster Ordnung durch stückweise konstante Werte, durch Polynome höheren Grades zu approximieren. Wir suchen ein

$$u_h(\mathbf{x},t) = \sum_{i=1}^{n} u_i(t) v_i(\mathbf{x}). \tag{4.4}$$

Hier seien u_i die Unbekannten und $v_i \in L^\infty(\Omega)$ geeignete Basisfunktionen. Wir betrachten den Raum der unstetigen Funktionen

$$V_h = \{v_h \in L^\infty(\Omega) : v_h \mid_K \in V(K), \forall K \in \mathcal{T}_h\}\,, \tag{4.5}$$

wobei $\mathcal{T}_h$ die Triangulierung des Gebietes Ω und $V(K)$ der sogenannte lokale Raum ist. Die Basisfunktionen v_i sollen aus diesem Raum V_h sein. In dieser Arbeit wird als Raum $V(K)$ der Raum $P^k = \{p \mid p \text{ ist Polynom vom Grad} \leq k\}$ genommen.
Zur Herleitung des Verfahrens multiplizieren wir Gleichung (4.2) mit $v_h \in V_h$, integrieren über ein Testvolumen $T_j \in \mathcal{T}_h$ und ersetzen die exakte Lösung u durch die Approximation $u_h \in V_h$. So erhalten wir

$$\frac{d}{dt}\int_{T_j} u_h(\mathbf{x},t)v_h(\mathbf{x})d\mathbf{x} + \sum_{S_{jl}\in\partial T_j}\int_{S_{jl}} \mathbf{f}(u_h(\mathbf{x},t))\cdot n_{jl}v_h(\mathbf{x})dS_{jl} - \int_{T_j}\mathbf{f}(u_h(\mathbf{x},t))\cdot\nabla v_h(\mathbf{x})d\mathbf{x} = 0\,. \tag{4.6}$$

Wir ersetzen die Integrale durch die Quadraturregeln wie folgt:

$$\int_{S_{jl}} \mathbf{f}(u_h(\mathbf{x},t))\cdot n_{jl}v_h(\mathbf{x})dS_{jl} \approx \sum_{q=1}^{L}\omega_q\mathbf{f}(u_h(\mathbf{x}_q,t))\cdot n_{jl}v_h(\mathbf{x}_q)\mid S_{jl}\mid, \tag{4.7}$$

wobei $\mathbf{x}_q \in S_{jl}$ die Integrationspunkte und ω_q die Gewichte der verwendeten Integrationsformeln sind.

$$\int_{T_j} \mathbf{f}(u_h(\mathbf{x},t))\cdot\nabla v_h(\mathbf{x})d\mathbf{x} \approx \sum_{q=1}^{M}\underline{\omega}_q\mathbf{f}(u_h(\mathbf{x}_q,t))\cdot\nabla v_h(\mathbf{x}_q)\mid T_j\mid, \tag{4.8}$$

wobei $\mathbf{x}_q \in T_j$ die Integrationspunkte und $\underline{\omega}_q$ die Gewichte der verwendeten Integrationsformeln sind.

Die in (4.7) und (4.8) benutzten Integrationspunkte $\mathbf{x}_q$ müssen noch auf das Referenzelement transformiert werden, um die Integrationsformeln aus 4.1 anwenden zu können.
Bevor wir das numerische Schema definieren können, wollen wir folgende Definition treffen.

DEFINITION 4.4.1 ***(L^2-Projektion)***
Auf einem Testvolumen T_j sei eine beliebige Funktion u_h gegeben. Gesucht ist eine Funktion $u^{(1)} \in P^1$, für die gilt:

$$\int_{T_j}\mid u_h - u^{(1)}\mid^2 = \min_{p\in P^1}\int_{T_j}\mid u_h - p\mid^2\ . \tag{4.9}$$

Diese Funktion $u^{(1)}$ ist dann die L^2-Projektion in den Raum der stückweise linearen Funktionen.

Der Operator P_{V_h} sei die L^2-Projektion

$$P_{V_h}(u_h) = u^{(1)}\,,$$

wobei $u^{(1)}$ durch (4.9) gegeben ist.
Dann ersetzen wir den Fluss $\mathbf{f}(u_h(\mathbf{x},t))\cdot n_{jl}$ in (4.7) durch die numerische Flussfunktion $g_{jl}(u_h^j(\mathbf{x},t), u_h^{jl}(\mathbf{x},t))$ und erhalten folgendes Schema:

$$u_h(\mathbf{x},0) = P_{V_h}(u_0(\mathbf{x}))\,,$$

$$\frac{d}{dt}\int_{T_j} u_h(\mathbf{x},t)v_h(\mathbf{x})dx + \sum_{S_{jl}\in\partial T_j} \mid S_{jl} \mid \sum_{q=1}^{L} \omega_q g_{jl}(u_h^j(\mathbf{x}_q,t), u_h^{jl}(\mathbf{x}_q,t))v_h(\mathbf{x}_q)$$

$$- \mid T_j \mid \sum_{q=1}^{M} \underline{\omega}_q \mathbf{f}(u_h(\tilde{\mathbf{x}}_q,t)) \cdot \nabla v_h(\tilde{\mathbf{x}}_q) = 0\,, \forall v_h \in V_h,\ \forall T_j \in \mathcal{T}_h. \quad (4.10)$$

Dabei seien $\mathbf{x}_q$ Integrationspunkte der verwendeten Integrationsformel auf dem Rand eines Elementes. Die $\tilde{\mathbf{x}}_q$ seien Integrationspunkte der verwendeten Integrationsformel auf dem Element.
Wir definieren nun für $v_i \in V_h$

$$\tilde{L}_j(u_h, v_i) := - \sum_{S_{jl}\in\partial T_j} \mid S_{jl} \mid \sum_{q=1}^{L} \omega_q g_{jl}(u_h^j(\mathbf{x}_q,t), u_h^{jl}(\mathbf{x}_q,t))v_i(\mathbf{x}_q)$$

$$+ \mid T_j \mid \sum_{q=1}^{M} \underline{\omega}_q \mathbf{f}(u_h(\tilde{\mathbf{x}}_q,t)) \cdot \nabla v_i(\tilde{\mathbf{x}}_q)\,. \quad (4.11)$$

So können wir Gleichung (4.10) schreiben als

$$\frac{d}{dt}\int_{T_j} u_h(\mathbf{x},t)v_i(\mathbf{x})dx = \tilde{L}_j(u_h, v_i)\,. \quad (4.12)$$

Seien $v_1, \ldots, v_n$ die Basisfunktionen des Raumes V_h. Dann müssen n Gleichungen vom Typ (4.12) betrachtet werden. Die numerische Lösung von (4.10) ist gegeben durch

$$u_h(\mathbf{x},t) = u_1(t)v_1(\mathbf{x}) + u_2(t)v_2(\mathbf{x}) + \ldots + u_n(t)v_n(\mathbf{x}) = u_1v_1 + u_2v_2 + \ldots + u_nv_n\,.$$

Wir erhalten

$$\begin{array}{rcl}
\int_{T_j} \partial_t(u_1v_1v_1 + u_2v_2v_1 + \ldots + u_nv_nv_1) & = & \tilde{L}_j(u_h, v_1) \\
\int_{T_j} \partial_t(u_1v_1v_2 + u_2v_2v_2 + \ldots + u_nv_nv_2) & = & \tilde{L}_j(u_h, v_2) \\
\ldots & & \ldots \\
\int_{T_j} \partial_t(u_1v_1v_n + u_2v_2v_n + \ldots + u_nv_nv_n) & = & \tilde{L}_j(u_h, v_n)\,.
\end{array} \quad (4.13)$$

Definieren wir nun die Matrix $\tilde{M}_j$ auf folgende Weise:

$$\tilde{M}_j := \begin{pmatrix}
\int_{T_j} v_1^2 & \int_{T_j} v_2v_1 & \ldots & \int_{T_j} v_nv_1 \\
\int_{T_j} v_1v_2 & \int_{T_j} v_2^2 & \ldots & \int_{T_j} v_nv_2 \\
\vdots & \vdots & \ddots & \vdots \\
\int_{T_j} v_1v_n & \int_{T_j} v_2v_n & \ldots & \int_{T_j} v_n^2
\end{pmatrix}.$$

Somit erhalten wir

$$\tilde{M}_j \cdot \begin{pmatrix} \partial_t u_1 \\ \partial_t u_2 \\ \vdots \\ \partial_t u_n \end{pmatrix} = \begin{pmatrix} \tilde{L}_j(u_h, v_1) \\ \tilde{L}_j(u_h, v_2) \\ \vdots \\ \tilde{L}_j(u_h, v_n) \end{pmatrix} .$$

Man sieht, dass man für jedes Element die Matrix $\tilde{M}_j$ zu invertieren hat, um dieses Gleichungssystem zu lösen. Die Matrix $\tilde{M}_j$ wird als Massenmatrix bezeichnet. Wenn man nun orthogonale Basisfunktionen verwendet, d.h. es gilt

$$\int_{T_j} v_i(\mathbf{x}) v_k(\mathbf{x}) = 0 \; \forall i \neq k \, , \forall T_j \in \mathcal{T}_h \, , \tag{4.14}$$

dann ist die Matrix $\tilde{M}_j$ eine Diagonalmatrix, so dass nur noch jede Zeile des Gleichungssystems durch das Diagonalelement geteilt werden muss.
Zunächst wollen wir zum besseren Verständnis weitere Definitionen machen.

DEFINITION 4.4.2 ***(Total Variation in 1D)***
Für $f : [a, b] \to \mathbb{R}$ definieren wir die totale Variation von f durch

$$TV_{[a,b]}(f) := \sup_{a=x_0<x_1<\ldots<x_n=b} \sum_{k=0}^{n-1} \mid f(x_{k+1}) - f(x_k) \mid , n \in \mathbb{N} \, .$$

Im diskreten Fall definieren wir für eine Sequenz $v = (u_j)_{j \in \mathbb{N}}$ von diskreten Werten u_j

$$TV(v) := \sum_{j=0}^{\infty} \mid u_{j+1} - u_j \mid \; .$$

DEFINITION 4.4.3 ***(Total Variation Diminishing)***
Seien die u^n durch ein Verfahren wie (4.1) gegeben.
Dann wird dieses Verfahren ein TVD-Verfahren (TVD = Total Variation Diminishing) genannt, wenn für alle $n \in \mathbb{N}$ gilt

$$TV(u^{n+1}) \leq TV(u^n) \leq TV(u^0) < \infty \, .$$

DEFINITION 4.4.4 ***(Total Variation Bounded (TVB))***
Wenn es eine Konstante $B > 0$ gibt, die nur von den Anfangswerten u^0 , $TV(u^0)$ und allen möglichen n und Δt, so dass $n\Delta t \leq T$, abhängt, nennen wir ein Verfahren ein TVB-Verfahren in $0 \leq t \leq T$, falls gilt:

$$TV(u^n) \leq B \, .$$

Es ist klar, dass ein TVD-Verfahren automatisch auch ein TVB-Verfahren ist.

DEFINITION 4.4.5 ***(TVDM und TVBM)***
Gegeben sei eine Funktion $u_h(\mathbf{x},t) = \sum_{i=1}^{n} u_i(t)v_i(\mathbf{x})$. *Sei*

$$\bar{u}_h := \frac{1}{\mid T \mid} \int_T u_h \, .$$

a) Gilt dann

$$TV(\bar{u}_h^{n+1}) \leq TV(\bar{u}_h^n) \leq TV(\bar{u}_h^0) < \infty \, ,$$

so ist das Verfahren ein TVDM-Verfahren. TVDM bedeutet, dass das Verfahren ein TVD Verfahren bzgl. der Mittelwerte ist.

b) Gilt dann

$$TV(\bar{u}_h^n) \leq B$$

mit der Konstanten B *aus Definition 4.4.4, so ist das Verfahren ein TVBM-Verfahren. TVBM bedeutet, dass das Verfahren ein TVB Verfahren bzgl. der Mittelwerte ist.*

DEFINITION 4.4.6 ***(TVD-Limiter)***
Wenn in einem Verfahren ein Limiter benutzt wird und dieses Verfahren ein TVD*-Verfahren ist, so nennen wir diesen Limiter auch einen* TVD*-Limiter.*

DEFINITION 4.4.7 ***(TVB-Limiter)***
Wenn in einem Verfahren ein Limiter benutzt wird und dieses Verfahren ein TVB*-Verfahren ist, so nennen wir diesen Limiter auch einen* TVB*-Limiter.*

DEFINITION 4.4.8 ***(TVDM-Limiter)***
Wenn in einem Verfahren ein Limiter benutzt wird und dieses Verfahren ein $TVDM$*-Verfahren ist, so nennen wir diesen Limiter auch einen* $TVDM$*-Limiter.*

DEFINITION 4.4.9 ***(TVBM-Limiter)***
Wenn in einem Verfahren ein Limiter benutzt wird und dieses Verfahren ein $TVBM$*-Verfahren ist, so nennen wir diesen Limiter auch einen* $TVBM$*-Limiter.*

Nach diesen kurzen Definitionen können wir die höhere Ordnung in der Zeit bei den Discontinuous Galerkin Verfahren betrachten.
Die höhere Ordnung in der Zeit wird durch eine TVD Runge-Kutta Zeitdiskretisierung erreicht. Die TVD Runge-Kutta Verfahren der Ordnung zwei und drei wurden in Kapitel 2 definiert.

4.5 Theoretische Resultate über Discontinuous Galerkin Verfahren

In diesem Abschnitt sollen theoretische Ergebnisse zu den Discontinuous Galerkin Verfahren aus Veröffentlichungen zitiert werden. Speziell wird die Konvergenz der Verfahren im skalaren Fall gezeigt.

Konvergenz der Discontinuous Galerkin Verfahren im skalaren Fall

Jaffre, Johnson und Szepessy beweisen in [31] die Konvergenz der Discontinuous Galerkin Verfahren im skalaren Fall. Die Beweistechniken laufen dabei ähnlich wie bei der Stromliniendiffusionsmethode [35].

Betrachte die Gleichung

$$u_t + \sum_{i=1}^{2} f_i(u)_{x_i} \text{ in } \mathbb{R}^2 \times \mathbb{R}_+ \tag{4.15}$$

mit

$$u(\cdot, 0) = u_0 \text{ in } \mathbb{R}^2 \,. \tag{4.16}$$

Dabei sei $u_t = \frac{\partial u}{\partial t}$, $f_i : \mathbb{R} \to \mathbb{R}$ glatte Flüsse mit beschränkten Ableitungen f_i'. Weiter haben die Anfangsdaten $u_0 \in L_2(\mathbb{R}^2)$ kompakten Träger.

LEMMA 4.5.1 *Nehmen wir an, dass $f_i' \in C(\mathbb{R})$ beschränkt ist und dass $u_0 \in L_2(\mathbb{R}^2)$ kompakten Träger hat. Dann konvergieren die Lösungen u_h des Discontinuous Galerkin Verfahren im P^q-Fall ($q \geq 0$) stark in $L_p^{loc}(\mathbb{R}^2 \times \mathbb{R}_+), 1 \leq \mathrm{p} < 2$ gegen die eindeutige Entropielösung u von (4.15),(4.16) für $h \to 0$.*

Beweis: Siehe Jaffre, Johnson und Szepessy [31].

BEMERKUNG 4.5.2 *In einer Arbeit von Despres [22] wird eine Entropieungleichung für eine höhere Ordnung Approximation mittels Discontinuous Galerkin Verfahren für die Euler-Gleichungen gegeben.*

Ordnung der Quadraturregeln

Nun stellt sich die Frage, von welcher Ordnung die verwendeten Quadraturregeln sein müssen, um eine gewisse Ordnung zu erreichen. Folgender Satz, der in einer der Arbeiten von Cockburn und Shu [16] bewiesen wird, liefert uns die Antwort darauf.

Betrachten wir die gewöhnliche Differentialgleichung

$$\frac{d}{dt}u_h = L_h(u_h, \gamma_h)\,. \tag{4.17}$$

Der Operator $L_h(u_h)$ ist eine diskrete Approximation von $-\text{div}\, f(u)$. Das folgende Ergebnis gibt einen Einblick in die Qualität dieser Approximation.

LEMMA 4.5.3 *Seien $u : \Omega \times (0,T) \longrightarrow \mathbb{R}$ und $f_i : \mathbb{R} \longrightarrow \mathbb{R}$ hinreichend glatt. Sei $f \circ u = f(u(\cdot,t)) \in W^{k+2,\infty}(\Omega)$. Sei $\gamma = spur(u)$. Sei die Quadraturformel über die Kanten exakt für Polynome vom Grad $\leq (2k+1)$, also $p \in P^{2k+1}$. Sei die Quadraturformel über die Elemente exakt für Polynome vom Grad $\leq (2k)$, also $p \in P^{2k}$. Nehmen wir an, dass die Familie der Triangulierungen $\mathcal{F} = \{\mathcal{T}_h\}_{h>0}$ regulär ist, z.B. es existiert eine Konstante σ, so dass gilt:*

$$\frac{h_{T_j}}{\rho_{T_j}} \geq \sigma\ , \forall T_j \in \mathcal{T}_h\ , \forall \mathcal{T}_h \in \mathcal{F}\ ,$$

wobei h_{T_j} der Durchmesser von T_j ist, und ρ_{T_j} der Durchmesser der größten Kugel in T_j ist. Falls $P^k(T_j) \subset V(T_j)\ \forall T_j \in \mathcal{T}_h$, gilt dann:

$$||\ L_h(u,\gamma) + div f(u)\ ||_{L^\infty(\Omega)} \leq C h^{k+1}\ |\ f(u)\ |_{W^{k+2,\infty}(\Omega)}\ .$$

Beweis: Siehe Cockburn und Shu [16].

Dieser Satz sagt uns, mit welchen Quadraturformeln wir jeweils arbeiten müssen, um eine bestimmte Ordnung zu erreichen.

In Cockburn et al.[17] werden folgende Theoreme zu Fehlerabschätzungen bewiesen. Die folgenden Sätze gelten in 1-D im skalaren, linearen Fall ($f(u) = cu$ mit $c \in \mathbb{R}$).

LEMMA 4.5.4 ***(Erste L^2-Fehler Abschätzung)***
Nehmen wir an, dass die Anfangswertefunktion u_0 aus $H^{k+1}(0,1)$ ist. Dann gilt für den Approximationsfehler

$$\|u - u_h\|_{L^2(0,1)} \leq C\ |\ u_0\ |_{H^{k+1}(0,1)}\ (\Delta x)^{k+1/2}\ .$$

Dabei hängt C nur von k, $|\ c\ |$ und dem Endzeitpunkt T ab.

Beweis: Siehe Cockburn [17].

LEMMA 4.5.5 ***(Zweite L^2-Fehler Abschätzung)***
Nehmen wir an, dass die Anfangswertefunktion u_0 aus $H^{k+2}(0,1)$ ist. Dann gilt für den Approximationsfehler

$$\|u - u_h\|_{L^2(0,1)} \leq C\ |\ u_0\ |_{H^{k+2}(0,1)}\ (\Delta x)^{k+1}\ .$$

Dabei hängt C nur von k, $|\ c\ |$ und dem Endzeitpunkt T ab.

Beweis: Siehe Cockburn [17].

Wir treffen jetzt folgende

DEFINITION 4.5.6 ***(P^k - Fall)***
Wenn die exakte Lösung u durch ein Polynom aus P^k approximiert wird, sprechen wir vom P^k - Fall.

Zusammenfassend kann man also sagen:
Betrachtet man den P^k- Fall, d.h. die gesuchte Funktion wird durch ein Polynom vom Grad k approximiert, so ist folgendes zu beachten:

- Die Integrationsformel für den Rand eines Elementes muss von der Ordnung $2k+1$ sein.
- Die Integrationsformel für ein Element muss von der Ordnung $2k$ sein. Unter diesen Annahmen wird in ([16],Theorem 2.10) bewiesen, dass die formale Konsistenzordnung (unter hinreichenden Regularitätsannahmen) des RKDG Verfahren von der Ordnung $(k+1)$ ist.
  (RKDG = Runge-Kutta Discontinuous Galerkin)
- Die CFL-Zahl muss kleiner $\frac{1}{2k+1}$ sein. Verlangt die Art der Runge-Kutta Zeitschritt-Diskretisierung auch eine CFL Zahl kleiner 1, so ist diese Zahl noch mit der CFL-Zahl des Ortes zu multiplizieren. ([14],Lemma 2.10) Der Beweis gilt für den Fall $k = 1, 2$ (second und third-order Fall). Für $k \geq 3$ ist dies eine Vermutung, die in numerischen Tests auch so implementiert wurde.

4.6 Verfahren zweiter Ordnung

Wenn man sich auf Rechtecke (2D) und Quader (3D) anstatt auf Dreiecke, Tetraeder und beliebige Hexaeder beschränkt, kann man als Basisfunktionen Legendre-Polynome benutzen. Damit hat man automatisch orthogonale Basisfunktionen und man erhält Diagonalmatrizen als Massenmatrizen. Man kann also auf die aufwendige Invertierung der lokalen Massenmatrizen verzichten.

4.6.1 Dreiecke in 2D

Für den P^1 Fall benutzen wir den folgenden Ausdruck für die Näherungslösung im Dreieck K:

$$u_h(x, y, t) = \sum_{i=1}^{3} u_i(t)\phi_i(x, y) \, .$$

Dabei sind die $u_i(t)$ die Freiheitsgrade in den Mittelpunkten der Kanten. Die Basisfunktionen $\phi_i(x, y)$ sind so gewählt, dass sie im Mittelpunkt der Kante m_i den Wert 1 und

in den anderen Mittelpunkten den Wert 0 haben. Wir wählen hier die Mittelpunkte anstatt die Eckpunkte aus folgendem Grund: Seien die $\phi_i(x,y) \in P^1, i = 1..3$, dann sind $\phi_i(x,y)\phi_j(x,y) \in P^2$. Für Dreiecke gibt es eine Quadraturformel, die die Quadraturpunkte auf den Mittelpunkten der Kanten hat und für Elemente aus P^2 exakt ist (siehe Tabelle 4.2). Somit sind bei dieser Wahl der Basisfunktionen diese orthogonal. Deshalb ist die entstehende Massenmatrix diagonal und hat folgende Gestalt:

$$\tilde{M}_K = \mid K \mid diag(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) \,.$$

Dabei ist

$$diag(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) = \begin{pmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} \end{pmatrix} \,.$$

Explizit werden die drei Basisfunktionen so definiert:

$$\begin{array}{c} \phi_1(\lambda_0 = 0.0, \lambda_1 = \frac{1}{2}, \lambda_2 = \frac{1}{2}) = 1.0 \\ \phi_2(\lambda_0 = 0.0, \lambda_1 = \frac{1}{2}, \lambda_2 = \frac{1}{2}) = 0.0 \\ \phi_3(\lambda_0 = 0.0, \lambda_1 = \frac{1}{2}, \lambda_2 = \frac{1}{2}) = 0.0 \end{array}$$

$$\begin{array}{c} \phi_1(\lambda_0 = \frac{1}{2}, \lambda_1 = 0.0, \lambda_2 = \frac{1}{2}) = 0.0 \\ \phi_2(\lambda_0 = \frac{1}{2}, \lambda_1 = 0.0, \lambda_2 = \frac{1}{2}) = 1.0 \\ \phi_3(\lambda_0 = \frac{1}{2}, \lambda_1 = 0.0, \lambda_2 = \frac{1}{2}) = 0.0 \end{array} \qquad (4.18)$$

$$\begin{array}{c} \phi_1(\lambda_0 = \frac{1}{2}, \lambda_1 = \frac{1}{2}, \lambda_2 = 0.0) = 0.0 \\ \phi_2(\lambda_0 = \frac{1}{2}, \lambda_1 = \frac{1}{2}, \lambda_2 = 0.0) = 0.0 \\ \phi_3(\lambda_0 = \frac{1}{2}, \lambda_1 = \frac{1}{2}, \lambda_2 = 0.0) = 1.0 \,. \end{array}$$

Die Quadraturformel über den Rand muss exakt sein für Polynome $p \in P^3$. Wir nehmen die Zweipunkt-Formel

$$\int_{-1}^{1} p(x) = p(-\frac{1}{\sqrt{3}}) + p(\frac{1}{\sqrt{3}}) \text{ für } p \in P^3$$

aus Tabelle 4.1.
Die Quadraturformel auf dem Element muss exakt sein für Polynome $p \in P^2$. Hier wird die Formel mit drei Punkten aus Tabelle 4.2 benutzt.

4.6.2 Quadrate in 2D

Für den P^1 Fall benutzen wir den folgenden Ausdruck für die Näherungslösung $u_h(x,y,t)$ im Quadrat (oder allgemein im achsenparallelen Rechteck) $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}] \times [y_{j-\frac{1}{2}}, y_{j+\frac{1}{2}}]$ (dieses werden wir im folgenden auch mit Zelle (i,j) bezeichnen) :

$$u_h(x,y,t) = \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y)\,,$$

wobei

$$\phi_i(x) = \frac{x - x_i}{\Delta x_i/2}\,,\ \psi_j(y) = \frac{y - y_j}{\Delta y_j/2}\,,$$

und

$$\Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}\,, \Delta y_j = y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}}\,.$$

Die Freiheitsgrade sind dann

$$\overline{u}(t)\,, u^x(t)\,, u^y(t).$$

Die Basisfunktionen sind

$$1\,, \phi_i(x)\,, \psi_j(y)\,.$$

Diese sind wieder orthogonal. Die Massenmatrix ist diagonal und lautet

$$\tilde{M}_{ij} = \Delta x_i \Delta y_j diag(1, \frac{1}{3}, \frac{1}{3})\,.$$

Die Quadraturformel über den Rand muss exakt sein für Polynome $p \in P^3$. Wir nehmen die Zweipunkt-Formel

$$\int_{-1}^{1} p(x) = p(-\frac{1}{\sqrt{3}}) + p(\frac{1}{\sqrt{3}}) \text{ für } p \in P^3$$

aus Tabelle 4.1.
Die Quadraturformel auf dem Element muss exakt sein für Polynome $p \in P^2$. Wir nehmen die für Polynome $p \in P^3$ exakte Formel aus Tabelle 4.3.

$$\int_{-1}^{1}\int_{-1}^{1} p(x,y)dxdy = p(-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}) + p(-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}) + p(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}) + p(\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}})$$

für $p \in P^3$.

Numerisches Verfahren

Zusammengefasst ergibt sich im Falle von Rechtecken in 2D folgendes numerisches Verfahren. In der Zelle (i,j) gilt

$$u_h(x,y,t) = \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y)\,.$$

Um besser kenntlich zu machen, dass es sich um die Zelle (i,j) handelt, schreiben wir nun

$$(u_h)_{(i,j)}(x,y,t) = (\overline{u})_{(i,j)}(t) + (u^x)_{(i,j)}(t)\phi_i(x) + (u^y)_{(i,j)}(t)\psi_j(y)$$

und nennen die Zelle (i,j) nun auch K. Die Nachbarn der Zelle K nennen wir K_l. Die gemeinsame Kante von der Zelle K und einer Zelle K_l nennen wir S_l.
Zum diskreten Zeitpunkt $t = t^n$ seien die Unbekannten gegeben durch

$$(\overline{u})^n_{(i,j)}\,,\,(u^x)^n_{(i,j)}\,,\,(u^y)^n_{(i,j)}\,.$$

Dann gilt

$$\begin{aligned}
(\overline{u})^{n+\frac{1}{2}}_{(i,j)} &= (\overline{u})^n_{(i,j)} - \tfrac{\Delta t}{\Delta x_i \Delta y_j}\cdot(update)_{(i,j)}\\
(u^x)^{n+\frac{1}{2}}_{(i,j)} &= (u^x)^n_{(i,j)} - \tfrac{3\Delta t}{\Delta x_i \Delta y_j}\cdot(update_x)_{(i,j)}\\
(u^y)^{n+\frac{1}{2}}_{(i,j)} &= (u^y)^n_{(i,j)} - \tfrac{3\Delta t}{\Delta x_i \Delta y_j}\cdot(update_y)_{(i,j)}
\end{aligned}$$

mit

$$\begin{aligned}
(update)_{(i,j)} &= \sum_{\{K_l|K_l \text{ ist Nachbar von } K\}} |\,S_l\,| \sum_{q=1}^{L}\omega_q\\
&\qquad g((u_h)^n_{(i,j)}(x_q,y_q,t^n),(u_h)^n_{K_l}(x_q,y_q,t^n))\\
(update_x)_{(i,j)} &= \sum_{\{K_l|K_l \text{ ist Nachbar von } K\}} |\,S_l\,| \sum_{q=1}^{L}\omega_q\\
&\qquad g((u_h)^n_{(i,j)}(x_q,y_q,t^n),(u_h)^n_{K_l}(x_q,y_q,t^n))\phi_i(x_q)\\
&\quad - |\,K\,| \sum_{q=1}^{M}\underline{\omega}_q \mathbf{f}((u_h)^n_{(i,j)}(\tilde{x}_q,\tilde{y}_q,t^n))\cdot\nabla\phi_i(\tilde{x}_q)\\
(update_y)_{(i,j)} &= \sum_{\{K_l|K_l \text{ ist Nachbar von } K\}} |\,S_l\,| \sum_{q=1}^{L}\omega_q\\
&\qquad g((u_h)^n_{(i,j)}(x_q,y_q,t^n),(u_h)^n_{K_l}(x_q,y_q,t^n))\psi_j(y_q)\\
&\quad - |\,K\,| \sum_{q=1}^{M}\underline{\omega}_q \mathbf{f}((u_h)^n_{(i,j)}(\tilde{x}_q,\tilde{y}_q,t^n))\cdot\nabla\psi_j(\tilde{y}_q)
\end{aligned}$$

und weiter

$$\begin{aligned}
(\overline{u})^{n+1}_{(i,j)} &= \tfrac{1}{2}(\overline{u})^n_{(i,j)} + \tfrac{1}{2}(\overline{u})^{n+\frac{1}{2}}_{(i,j)} - \tfrac{1}{2}\tfrac{\Delta t}{\Delta x_i \Delta y_j}\cdot(update)_{(i,j)}\\
(u^x)^{n+1}_{(i,j)} &= \tfrac{1}{2}(u^x)^n_{(i,j)} + \tfrac{1}{2}(u^x)^{n+\frac{1}{2}}_{(i,j)} - \tfrac{1}{2}\tfrac{3\Delta t}{\Delta x_i \Delta y_j}\cdot(update_x)_{(i,j)}\\
(u^y)^{n+1}_{(i,j)} &= \tfrac{1}{2}(u^y)^n_{(i,j)} + \tfrac{1}{2}(u^y)^{n+\frac{1}{2}}_{(i,j)} - \tfrac{1}{2}\tfrac{3\Delta t}{\Delta x_i \Delta y_j}\cdot(update_y)_{(i,j)}
\end{aligned}$$

mit

$$
\begin{aligned}
(update)_{(i,j)} &= \sum_{\{K_l | K_l \text{ ist Nachbar von } K\}} | S_l | \sum_{q=1}^{L} \omega_q \\
&\qquad g((u_h)_{(i,j)}^{n+\frac{1}{2}}(x_q, y_q, t^{n+\frac{1}{2}}), (u_h)_{K_l}^{n+\frac{1}{2}}(x_q, y_q, t^{n+\frac{1}{2}})) \\
(update_x)_{(i,j)} &= \sum_{\{K_l | K_l \text{ ist Nachbar von } K\}} | S_l | \sum_{q=1}^{L} \omega_q \\
&\qquad g((u_h)_{(i,j)}^{n+\frac{1}{2}}(x_q, y_q, t^{n+\frac{1}{2}}), (u_h)_{K_l}^{n+\frac{1}{2}}(x_q, y_q, t^{n+\frac{1}{2}}))\phi_i(x_q) \\
&\quad - | K | \sum_{q=1}^{M} \underline{\omega}_q \mathbf{f}((u_h)_{(i,j)}^{n+\frac{1}{2}}(\tilde{x}_q, \tilde{y}_q, t^{n+\frac{1}{2}})) \cdot \nabla \phi_i(\tilde{x}_q) \\
(update_y)_{(i,j)} &= \sum_{\{K_l | K_l \text{ ist Nachbar von } K\}} | S_l | \sum_{q=1}^{L} \omega_q \\
&\qquad g((u_h)_{(i,j)}^{n+\frac{1}{2}}(x_q, y_q, t^{n+\frac{1}{2}}), (u_h)_{K_l}^{n+\frac{1}{2}}(x_q, y_q, t^{n+\frac{1}{2}}))\psi_j(y_q) \\
&\quad - | K | \sum_{q=1}^{M} \underline{\omega}_q \mathbf{f}((u_h)_{(i,j)}^{n+\frac{1}{2}}(\tilde{x}_q, \tilde{y}_q, t^{n+\frac{1}{2}})) \cdot \nabla \psi_j(\tilde{y}_q)
\end{aligned}
$$

4.6.3 Tetraeder in 3D

Für den P^1 Fall benutzen wir den folgenden Ausdruck für die Näherungslösung im Tetraeder K:

$$u_h(x, y, z, t) = \sum_{i=1}^{4} u_i(t)\phi_i(x, y, z) \, .$$

Dabei werden die Freiheitsgrade $u_i(t)$ in den vier Quadraturpunkten gewählt, dessen Quadraturformel mit vier Punkten exakt für Polynome aus P^2 ist. Die Basisfunktionen $\phi_i(x, y, z)$ werden so gewählt, dass sie in einem der vier Quadraturpunkte den Wert 1 annehmen und in den anderen Quadraturpunkten 0 sind (siehe Abbildung 4.3). Somit sind bei dieser Wahl der Basisfunktionen diese orthogonal. Deshalb ist die entstehende Massenmatrix diagonal und hat folgende Gestalt:

$$\tilde{M}_K = | K | \, diag(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}) \, .$$

Die Quadraturformel über den Rand muss exakt sein für Polynome $p \in P^3$. Auf der Dreiecksrandfläche nehmen wir eine Quadraturformel, die mit sechs Punkten exakt für Polynome $p \in P^4$ ist (siehe Tabelle 4.2).
Die Quadraturformel auf dem Element muss exakt sein für Polynome $p \in P^2$. Hier wird die Formel mit vier Punkten aus Tabelle 4.4 benutzt.

3D (Zweite Ordnung)

	λ_0	λ_1	λ_2	λ_3	ϕ_1	ϕ_2	ϕ_3	ϕ_4
Punkt 1	α	β	β	β	1	0	0	0
Punkt 2	β	α	β	β	0	1	0	0
Punkt 3	β	β	α	β	0	0	1	0
Punkt 4	β	β	β	α	0	0	0	1

mit

$\alpha = \frac{5+3\sqrt{5}}{20} \approx 0.58541020$
$\beta = \frac{5-\sqrt{5}}{20} \approx 0.13819660$

Abbildung 4.3: Quadraturpunkte: Exakt für Polynome $p \in P^2$ auf Tetraedern.
In dieser Tabelle kann man ablesen, wie die Basisfunktionen ϕ_i,$i = 1, \ldots, 4$ in den jeweiligen Punkten gewählt werden. Die λ_j sind die jeweiligen baryzentrischen Koordinaten.

4.6.4 Hexaeder in 3D

Für den P^1 Fall benutzen wir den folgenden Ausdruck für die Näherungslösung $u_h(x, y, z, t)$ im Hexaeder $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}] \times [y_{j-\frac{1}{2}}, y_{j+\frac{1}{2}}] \times [z_{k-\frac{1}{2}}, z_{k+\frac{1}{2}}]$:

$$u_h(x, y, z, t) = \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y) + u^z(t)\xi_k(z)\,,$$

wobei

$$\phi_i(x) = \frac{x - x_i}{\Delta x_i/2}\,,\ \psi_j(y) = \frac{y - y_j}{\Delta y_j/2}\,,\ \xi_k(z) = \frac{z - z_k}{\Delta z_k/2}\,.$$

und

$$\Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}\,, \Delta y_j = y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}}\,, \Delta z_k = z_{k+\frac{1}{2}} - z_{k-\frac{1}{2}}\,.$$

Die Freiheitsgrade sind dann

$$\overline{u}(t)\,, u^x(t)\,, u^y(t)\,, u^z(t).$$

Die Basisfunktionen sind

$$1\,, \phi_i(x)\,, \psi_j(y)\,, \xi_k(z)\,.$$

Diese sind wieder orthogonal. Die Massenmatrix ist diagonal und lautet

$$\tilde{M}_{ijk} = \Delta x_i \Delta y_j \Delta z_k diag(1, \frac{1}{3}, \frac{1}{3}, \frac{1}{3})\,.$$

Die Quadraturformel über den Rand muss exakt sein für Polynome $p \in P^3$. Auf der Randfläche nehmen wir eine Quadraturformel, die mit vier Punkten exakt für Polynome $p \in P^3$ ist (siehe Tabelle 4.3).

$$\int_{-1}^{1}\int_{-1}^{1} p(x,y)dxdy = p(-\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}}) + p(-\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}) + p(\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}) + p(\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}})$$

für $p \in P^3$.

Die Quadraturformel auf dem Element muss exakt sein für Polynome $p \in P^2$. Hier wird die Formel mit vier Punkten aus Tabelle 4.5 benutzt.

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} p(x,y,z)dxdydz =$$
$$2\Big(p(\tfrac{1}{\sqrt{1.5}},0,\tfrac{1}{\sqrt{3}}) + p(0,\tfrac{1}{\sqrt{1.5}},-\tfrac{1}{\sqrt{3}}) + p(-\tfrac{1}{\sqrt{1.5}},0,\tfrac{1}{\sqrt{3}}) + p(0,-\tfrac{1}{\sqrt{1.5}},-\tfrac{1}{\sqrt{3}})\Big) \text{ für } p \in P^2\,.$$

4.7 Verfahren dritter Ordnung

4.7.1 Dreiecke in 2D

Für den P^2 Fall benutzen wir den folgenden Ausdruck für die Näherungslösung im Dreieck K:

$$u_h(x,y,t) = \sum_{i=1}^{6} u_i(t)\phi_i(x,y)\,.$$

Dabei werden die Freiheitsgrade $u_i(t)$ in den sechs Quadraturpunkten gewählt, dessen Quadraturformel mit sechs Punkten exakt für Polynome $p \in P^4$ ist. Die Basisfunktionen $\phi_i(x,y)$ werden so gewählt, dass sie in je einem der sechs Quadraturpunkte den Wert 1 und in den anderen Quadraturpunkten den Wert 0 annehmen (siehe Abbildung 4.4). Bei dieser Wahl der Basisfunktionen $\phi_i(x,y)$ sind diese orthogonal. Deshalb ist die entstehende Massenmatrix diagonal und hat folgende Gestalt:

$$\tilde{M}_K = \mid K \mid diag(\mu,\mu,\mu,\frac{1}{3}-\mu,\frac{1}{3}-\mu,\frac{1}{3}-\mu)\,.$$

Die Quadraturformel über den Rand muss exakt sein für Polynome $p \in P^5$. Wir nehmen die Dreipunkt-Formel aus Tabelle 4.1

$$\int_{-1}^{1} p(x)dx = \frac{5}{9}[p(-\sqrt{\frac{3}{5}}) + p(\sqrt{\frac{3}{5}})] + \frac{8}{9}p(0) \text{ für } p \in P^5\,.$$

Die Quadraturformel auf den Elementen muss exakt sein für Polynome $p \in P^4$. Hier nehmen wir die Formel mit sechs Punkten aus Tabelle 4.2.

2D (Vierte Ordnung)

	λ_0	λ_1	λ_2	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6
Punkt 1	α	β	β	1	0	0	0	0	0
Punkt 2	β	α	β	0	1	0	0	0	0
Punkt 3	β	β	α	0	0	1	0	0	0
Punkt 4	γ	δ	δ	0	0	0	1	0	0
Punkt 5	δ	γ	δ	0	0	0	0	1	0
Punkt 6	δ	δ	γ	0	0	0	0	0	1

mit

$\alpha \approx 0.8168475729804585$
$\beta = \frac{1}{2}(1-\alpha)$
$\gamma \approx 0.1081030181680702$
$\delta = \frac{1}{2}(1-\gamma)$
$\mu \approx 0.1099517436553219$

Abbildung 4.4: Quadraturpunkte: Exakt für Polynome $p \in P^4$ auf Dreiecken.
In dieser Tabelle kann man ablesen, wie die Basisfunktionen ϕ_i,$i = 1,\ldots,6$ in den jeweiligen Punkten gewählt werden. Die λ_j sind die jeweiligen baryzentrischen Koordinaten.

4.7.2 Quadrate in 2D

Für den P^2 Fall benutzen wir den folgenden Ausdruck für die Näherungslösung $u_h(x,y,t)$ im Quadrat (oder allgemein im achsenparallelen Rechteck) $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}] \times [y_{j-\frac{1}{2}}, y_{j+\frac{1}{2}}]$:

$$\begin{aligned} u_h(x,y,t) &= \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y) + u^{xy}(t)\phi_i(x)\psi_j(y) \\ &\quad + u^{xx}(t)(\phi_i^2(x) - \tfrac{1}{3}) + u^{yy}(t)(\psi_j^2(y) - \tfrac{1}{3})\,, \end{aligned}$$

wobei

$$\phi_i(x) = \frac{x - x_i}{\Delta x_i/2}\,,\ \psi_j(y) = \frac{y - y_j}{\Delta y_j/2}\,,$$

und

$$\Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}\,,\ \Delta y_j = y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}}\,.$$

Die Freiheitsgrade sind dann

$$\overline{u}(t)\,,\, u^x(t)\,,\, u^y(t)\,,\, u^{xy}(t)\,,\, u^{xx}(t)\,,\, u^{yy}(t)\,.$$

Die Basisfunktionen sind

$$1\,,\, \phi_i(x)\,,\, \psi_j(y)\,,\, \phi_i(x)\psi_j(y)\,,\, \phi_i^2(x) - \frac{1}{3}\,,\, \psi_j^2(y) - \frac{1}{3}\,.$$

Diese sind wieder orthogonal. Die Massenmatrix ist diagonal und lautet

$$\tilde{M}_{ij} = \Delta x_i \Delta y_j diag(1, \frac{1}{3}, \frac{1}{3}, \frac{1}{9}, \frac{4}{45}, \frac{4}{45}) .$$

Die Quadraturformel über den Rand muss exakt sein für Polynome $p \in P^5$. Wir nehmen die Dreipunkt-Formel aus Tabelle 4.1

$$\int_{-1}^{1} p(x)dx = \frac{5}{9}[p(-\sqrt{\frac{3}{5}}) + p(\sqrt{\frac{3}{5}})] + \frac{8}{9}p(0) \text{ für } p \in P^5 . \tag{4.19}$$

Die Quadraturformel auf den Elementen muss exakt sein für Polynome $p \in P^4$. Wir nehmen die Neunpunkt-Formel aus Tabelle 4.3, die sogar exakt ist für Polynome $p \in P^5$.

4.7.3 Hexaeder in 3D

Für den P^2 Fall benutzen wir den folgenden Ausdruck für die Näherungslösung $u_h(x, y, t)$ im Hexaeder $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}] \times [y_{j-\frac{1}{2}}, y_{j+\frac{1}{2}}] \times [z_{k-\frac{1}{2}}, z_{k+\frac{1}{2}}]$:

$$\begin{aligned} u_h(x,y,z,t) \quad = \quad & \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y) + u^z(t)\xi_k(z) \\ & + u^{xy}(t)\phi_i(x)\psi_j(y) + u^{xz}(t)\phi_i(x)\xi_k(z) + u^{yz}(t)\psi_j(y)\xi_k(z) \\ & + u^{xx}(t)(\phi_i^2(x) - \tfrac{1}{3}) + u^{yy}(t)(\psi_j^2(y) - \tfrac{1}{3}) + u^{zz}(t)(\xi_k^2(z) - \tfrac{1}{3}) , \end{aligned}$$

wobei

$$\phi_i(x) = \frac{x - x_i}{\Delta x_i/2} , \ \psi_j(y) = \frac{y - y_j}{\Delta y_j/2} , \ \xi_k(z) = \frac{z - z_k}{\Delta z_k/2} .$$

und

$$\Delta x_i = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}} , \Delta y_j = y_{j+\frac{1}{2}} - y_{j-\frac{1}{2}} , \Delta z_k = z_{k+\frac{1}{2}} - z_{k-\frac{1}{2}} .$$

Die Freiheitsgrade sind dann

$$\overline{u}(t) , u^x(t) , u^y(t) , u^z(t) , u^{xy}(t) , u^{xz}(t) , u^{yz}(t) , u^{xx}(t) , u^{yy}(t) , u^{zz}(t) .$$

Die Basisfunktionen sind

$$1 , \phi_i(x) , \psi_j(y) , \xi_k(z) , \phi_i(x)\psi_j(y), \phi_i(x)\xi_k(z), \psi_j(y)\xi_k(z),$$
$$\phi_i^2(x) - \frac{1}{3} , \psi_j^2(y) - \frac{1}{3} , \xi_k^2(z) - \frac{1}{3} .$$

Diese sind wieder orthogonal. Die Massenmatrix ist diagonal und lautet

$$\tilde{M}_{ijk} = \Delta x_i \Delta y_j \Delta z_k diag(1, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{9}, \frac{1}{9}, \frac{1}{9}, \frac{4}{45}, \frac{4}{45}, \frac{4}{45}) .$$

Die Quadraturformel über den Rand muss exakt sein für Polynome $p \in P^5$ (siehe Tabelle 4.3).
Die Quadraturformel auf den Elementen muss exakt sein für Polynome $p \in P^4$ (siehe Tabelle 4.5).

4.8 Limitierung

In diesem Abschnitt soll die Limitierung bei den Discontinuous Galerkin Verfahren besprochen werden. Diese Limitierung ist notwendig, um Oszillationen in den numerischen Verfahren zu vermeiden. Bei den MUSCL (siehe 4.3) oder ENO Verfahren [38] wird in jedem Zeitschritt aus stückweise konstanten Werten in jeder Zelle in Abhängigkeit der Werte in den Nachbarzellen ein Polynom von höherem Grad approximiert. Vor der Flussberechnung wird dieses Polynom dann geeignet limitiert, um Oszillationen zu vermeiden. Bei den Discontinuous Galerkin Verfahren wird nach jedem Zeitschritt eine Projektion durchgeführt, um ebenfalls Oszillationen zu vermeiden. Siehe dazu die folgende Skizze (4.5).

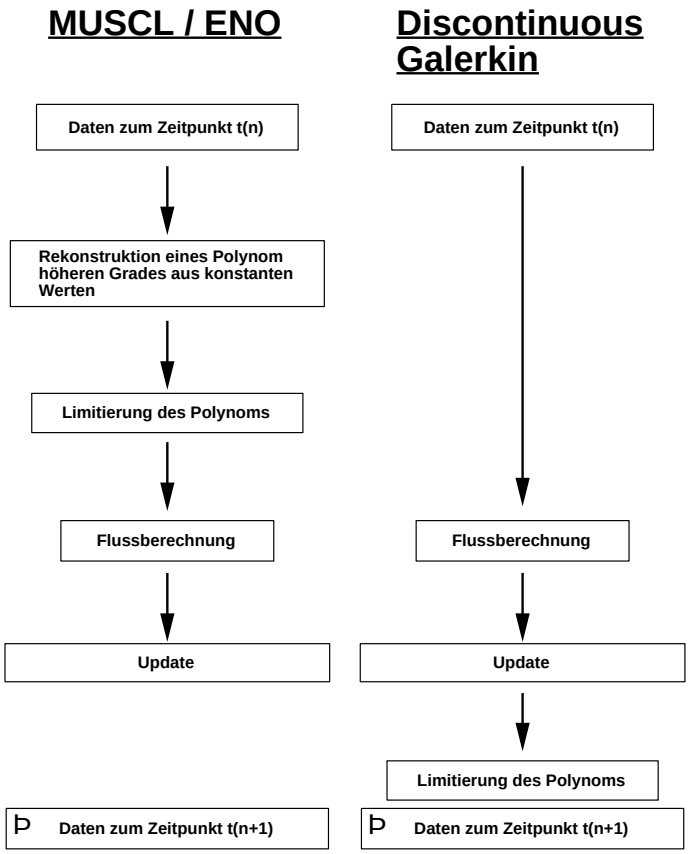

Abbildung 4.5: Vergleich zwischen MUSCL/ENO Verfahren und Discontinuous Galerkin Verfahren

Formal ist das Runge-Kutta Discontinuous Galerkin Verfahren (RKDG) folgendermaßen definiert:

DEFINITION 4.8.1 ***(RKDG-Verfahren)***

- *Setze* $u_h^0 = \Lambda\Pi_h P_{V_h}(u_0)$ *;*
- *Für die Zeitschritte* $n = 0, 1, \ldots, Z_T - 1$ *berechne* u_h^{n+1} *wie folgt:*
  - *Setze* $u_h^{(0)} = u_h^n$ *;*
  - *für* $i = 1, \ldots, k+1$ *berechne die Zwischenwertfunktionen:*

$$u_h^{(i)} = \Lambda\Pi_h\Big(\sum_{l=0}^{i-1} \alpha_{il} u_h^{(l)} + \beta_{il}\Delta t^n L_h(u_h^{(l)})\Big)\,; \tag{4.20}$$

  - *Setze* $u_h^{n+1} = u_h^{(k+1)}$.

Der Projektionsoperator $\Lambda\Pi_h$ *zur Limitierung wird im folgenden ausführlich behandelt. Um deutlich zu machen, dass wir uns in einem* P^k*-Fall befinden, schreiben wir auch* $\Lambda\Pi_h^{(P^k)}$ *statt* $\Lambda\Pi_h$*. Die Koeffizienten* α_{il} *und* β_{il} *sind definiert durch das Runge-Kutta Verfahren (siehe Abschnitt 2.5), um die höhere Ordnung in der Zeit zu erreichen.*

4.8.1 Projektion mittels Maximumsprinzip für skalare Gleichungen

Bei skalaren Gleichungen kann man unter Ausnutzung des Maximumprinzips eine Art von Projektion benutzen, bei der zwar auch limitiert wird, aber auf eine explizite Definition eines Limiters verzichtet werden kann. Diese Projektion geschieht nach dem folgenden Algorithmus.

- Die Approximation $(u_h)_j^n$ der exakten Lösung u sei in der Zelle T_j zum Zeitpunkt t^n gegeben.
- Bestimme für jede Zelle T_j
  $max_j^n = \max((u_h)_j^n, (u_h)_{j1}^n, \cdots, (u_h)_{jN}^n)$ und
  $min_j^n = \min((u_h)_j^n, (u_h)_{j1}^n, \cdots, (u_h)_{jN}^n)$.
  Dabei sind $(u_h)_{j1}^n, \cdots, (u_h)_{jN}^n$ die Werte in den N Nachbarzellen von T_j.
- Nun folgt die Berechnung eines Runge-Kutta-Teilzeitschrittes gemäss Definition 4.8.1.
- Gilt
  $min_j^n \le (u_h^{(i)})_j \le max_j^n$ für alle Zellen T_j ,
  so braucht hier nicht limitiert zu werden. Fahre in diesem Fall mit dem nächsten Runge-Kutta-Teilzeitschritt fort.

- Gilt
  $min_j^n \leq (u_h^{(i)})_j \leq max_j^n$ in einer Zelle T_j nicht, so muss die neue Lösung dort limitiert werden. Würde man hier nichts unternehmen, würden eventuell neue Extremwerte entstehen und außerdem wäre das Maximumsprinzip für skalare Funktionen [38] verletzt.

  – Konstruiere eine neue Lösung $\tilde{u}_h^{(i)}$ aus $u_h^{(i)}$ mit folgenden Eigenschaften:

  * Die Erhaltungseigenschaft [38] muss gewährleistet sein:
    $\int_{T_j} \tilde{u}_h^{(i)} = \int_{T_j} u_h^{(i)}$
  * Das Maximumsprinzip muss erfüllt sein:
    $min_j^n \leq \tilde{u}_h^{(i)} \leq max_j^n$
  * Eine sehr einfache Limitierung erreicht man z.B. so: Gilt

$$min_j^n \leq (u_h^{(i)})_j \leq max_j^n \tag{4.21}$$

    in einer Zelle T_j nicht, so halbiere in der Gleichung

$$u_h(x,y,z,t) = \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y) + u^z(t)\xi_k(z)$$

    die Steigungsfaktoren $u^x(t), u^y(t), u^z(t)$, bis Bedingung (4.21) gilt. Sollte die Bedingung (4.21) nach einer gewissen Anzahl von Schritten nicht erfüllt sein, so setze $u^x(t) = u^y(t) = u^z(t) = 0$. Damit wird unter Erhaltungseigenschaft auf das konstante Polynom zurückgeschaltet.

  – Definiere nun $u_h^{(i)} := \tilde{u}_h^{(i)}$ und fahre mit dem nächsten Runge-Kutta-Teilzeitschritt fort.

Man mache sich deutlich, dass dieses Verfahren nur das globale, nicht aber das lokale Maximumsprinzip erfüllt. Somit ist dies kein TVD-Verfahren, sondern nur ein TVB-Verfahren.

4.8.2 Projektion bei skalaren Gleichungen und Systemen

Bei dieser Möglichkeit des Limitierungsschritt folgen wir dem Vorschlag von Cockburn und Shu [17].
Hier wollen wir eine Funktion im P^1-Fall limitieren.
Zunächst konstruieren wir einen Limitoperator $\Lambda\Pi_h^{(P^1)}$ auf stückweise linearen Funktionen, so dass die folgenden Eigenschaften erfüllt sind:

- Falls u_h konstant ist , gilt $\Lambda\Pi_h^{(P^1)} u_h = u_h$.

- Die Erhaltungseigenschaft muss gewährleistet sein. Für jedes Element K der Triangulierung $\mathcal{T}_h$ muss gelten:

$$\int_K \Lambda\Pi_h^{(P^1)} u_h = \int_K u_h \,.$$

- Auf jedem Element K von $\mathcal{T}_h$ darf der Gradient von $\Lambda\Pi_h^{(P^1)} u_h$ nicht größer als der von u_h sein.

Rechtecke und Hexaeder

Hier ist die Limitierung für den Hexaeder Fall notiert. Die Reduzierung auf Rechtecke ist offensichtlich bzw. kann in [17] nachgelesen werden. Betrachten wir zunächst den P^1-Fall in der skalaren Version. Limitiert werden die Steigungsvorfaktoren u^x, u^y, u^z der Gleichung

$$u_h(x,y,z,t) = \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y) + u^z(t)\xi_k(z)\,.$$

Um eine TVD-Limitierung zu erhalten, müssten wir u^x durch

$$m(u^x, \overline{u}_{i+1,j,k} - \overline{u}_{i,j,k}, \overline{u}_{i,j,k} - \overline{u}_{i-1,j,k}) \tag{4.22}$$

ersetzen.
Dabei ist m die gewöhliche *minmod* Funktion

$$m(a_1,\ldots,a_m) := \begin{cases} s \min_i \mid a_i \mid & \text{, falls } s = sign(a_1) = \ldots = sign(a_m) \\ 0 & \text{, sonst} \end{cases} . \tag{4.23}$$

Unglücklicherweise degeneriert dies wie jedes andere TVD-Verfahren zu einem Verfahren erster Ordnung in Umgebung von kritischen Punkten [46].
Cockburn und Shu machen in [14] den Vorschlag, u^x statt dessen durch

$$\overline{m}(u^x, \overline{u}_{i+1,j,k} - \overline{u}_{i,j,k}, \overline{u}_{i,j,k} - \overline{u}_{i-1,j,k})\,. \tag{4.24}$$

zu ersetzen.
Dabei ist $\overline{m}$ die korrigierte *minmod* Funktion [14], die definiert ist durch

$$\overline{m}(a_1,\ldots,a_m) := \begin{cases} a_1 & \text{, falls } \mid a_1 \mid \leq M\Delta x^2 \\ m(a_1,\ldots,a_m) & \text{, sonst} \end{cases} . \tag{4.25}$$

In $1D$ beweisen Cockburn und Shu folgendes Lemma, das eine Aussage über die Konstante M macht.

LEMMA 4.8.2 *Betrachten wir Regionen, in denen $u \in C^2$ und $\mid u_{xx} \mid \leq M_2$ ist. Dabei ist M_2 eine Konstante.*
Falls

$$M = \frac{2}{3} M_2 \,,$$

oder

$$M = \frac{2}{9}(3 + 10M_2) \cdot M_2 \cdot \frac{h^2}{h^2 + \mid \Delta_+ u_j^0 \mid + \mid \Delta_- u_j^0 \mid} \,,$$

dann beeinflusst die Limitierung die Genauigkeit der approximierten Lösung in diesen Regionen nicht.

Beweis: Siehe Cockburn und Shu [14].
Dadurch ist dieses Verfahren kein TVD-Verfahren mehr, sondern nur noch ein TVB-Verfahren [14].
Es ist klar, dass gilt: Je kleiner M, desto weniger Oszillationen werden zugelassen. In der Praxis wird man $M_2 = max_{D_j} \mid u_{xx}^0 \mid$, wobei D_j eine Umgebung von glatten kritischen Punkten von $u_0(x)$ ist, wählen. In dieser Arbeit haben wir mit $M = 50$ gerechnet.
Jetzt wollen wir noch zwei theoretische Resultate von Cockburn [13] zitieren. Darin wird speziell eine Aussage zur Konvergenz im nichtlinearen Fall gegen die Entropielösung gemacht.

LEMMA 4.8.3 ***(Die TVBM Eigenschaft)***
Wir nehmen an, dass der Limiter $\Lambda\Pi_h$ ein TVBM-Limiter (siehe Definition 4.4.9) ist. Weiter seien alle Koeffizienten α_{il} des Runge-Kutta Verfahrens (4.20) nichtnegativ und erfüllen folgende Bedingung:

$$\sum_{l=0}^{i-1} \alpha_{il} = 1 \,, \;\; i = 1, \ldots, k+1 \,.$$

Dann gilt

$$\mid \bar{u}_h^n \mid_{TV(0,1)} \leq \mid \bar{u}_0 \mid_{TV(0,1)} + CM \,, \;\; \forall n \geq 0 \,,$$

wobei C nur von k abhängt.

Mittels des Lemmas von Arzelá-Ascoli kann folgendes Lemma bewiesen werden.

LEMMA 4.8.4 ***(Konvergenz gegen die Entropie Lösung)***
Sei der Limiter $\Lambda\Pi_h$ ein TVDM- oder TVBM-Limiter (siehe Definitionen 4.4.8,4.4.9) . Weiter seien alle Koeffizienten α_{il} des Runge-Kutta Verfahrens (4.20) nichtnegativ und erfüllen folgende Bedingung:

$$\sum_{l=0}^{i-1} \alpha_{il} = 1\ ,\ i = 1, \ldots, k+1\ .$$

Dann existiert eine Teilfolge $\{\bar{u}_{h'}\}_{h'>0}$ der Folge $\{\bar{u}_h\}_{h>0}$, die durch das RKDG Verfahren erzeugt worden ist und in $L^\infty(0,T;L^1(0,1))$ gegen eine schwache Lösung des Problems (4.2),(4.3) konvergiert.

Falls die TVBM Version des Limiters $\Lambda\Pi_h$ benutzt wird, konvergiert die ganze Folge und ist diese schwache Lösung die Entropielösung.

Weiter gilt: Gilt für den Limiter $\Lambda\Pi_h$

$$|| \bar{u}_h - \Lambda\Pi_h(u_h) ||_{L^1(0,1)} \leq C\Delta x \mid \bar{u}_h \mid_{TV(0,1)}\ ,$$

dann gilt das obere Resultat nicht nur für die Mittelwerte der Folge $\{\bar{u}_h\}_{h>0}$, sondern für die Folge der Funktionen $\{u_h\}_{h>0}$.

Dieser Satz verwendet einen Limiter in seiner Konvergenzaussage, der Satz 4.5.1 jedoch macht seine Konvergenzaussage für den Fall ohne Limiter.
Ähnlich wie in (4.24) wird u^y ersetzt durch

$$\overline{m}(u^y, \overline{u}_{i,j+1,k} - \overline{u}_{i,j,k}, \overline{u}_{i,j,k} - \overline{u}_{i,j-1,k})\ ,$$

wobei in (4.25) Δx durch Δy ersetzt wird.
Ähnlich wie in (4.24) wird u^z ersetzt durch

$$\overline{m}(u^z, \overline{u}_{i,j,k+1} - \overline{u}_{i,j,k}, \overline{u}_{i,j,k} - \overline{u}_{i,j,k-1})\ ,$$

wobei in (4.25) Δx durch Δz ersetzt wird.
Somit ergibt sich also insgesamt die Definition des Projektionsoperator wie folgt:

$$\begin{aligned} \Lambda\Pi_h^{(P^1)} u_h(x,y,z,t) \;=\; & \overline{u}(t) \\ & + \overline{m}(u^x, \overline{u}_{i+1,j,k} - \overline{u}_{i,j,k}, \overline{u}_{i,j,k} - \overline{u}_{i-1,j,k})\phi_i(x) \\ & + \overline{m}(u^y, \overline{u}_{i,j+1,k} - \overline{u}_{i,j,k}, \overline{u}_{i,j,k} - \overline{u}_{i,j-1,k})\psi_j(y) \\ & + \overline{m}(u^z, \overline{u}_{i,j,k+1} - \overline{u}_{i,j,k}, \overline{u}_{i,j,k} - \overline{u}_{i,j,k-1})\xi_k(z)\ . \end{aligned} \tag{4.26}$$

Mit diesen Definitionen kann man nun skalare Gleichungen limitieren.
Zum Limitieren im Fall von Systemen benötigen wir zunächst folgende

DEFINITION 4.8.5 ***(charakteristische Variablen)***
Sei $w = (\rho, \rho u_1, \rho u_2, \rho u_3, e)^t$ der Vektor der Euler-Gleichungen in konservativen Variablen. Die Matrix $(\partial_n f(w))$ genügt der Gleichung $Q^{-1}(w)(\partial_n f(w))Q(w) = \Lambda(w)$ (siehe (2.4)). $Q^{-1}(w)$ bestehe aus den linken Eigenvektoren von $(\partial_n f(w))$. Dann nennen wir die Einträge des Vektors

$$\tilde{w} := Q^{-1}(w) \cdot w$$

charakteristische Variablen.

Betrachten wir nun die Freiheitsgrade $\overline{u}(t), u^x(t), u^y(t), u^z(t)$ der Gleichung

$$u_h(x, y, z, t) = \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y) + u^z(t)\xi_k(z)$$

als vektorwertige Funktionen.
Für Systeme schlagen Cockburn und Shu [17] das Limitieren der lokalen charakteristischen Variablen vor. Um den Vektor u^x zu limitieren, gehen wir wie folgt vor:

- Bestimme die Matrizen Q und die Inverse Q^{-1}, die die Jacobimatrix bzgl. des Mittelwertes im Element ijk in x-Richtung diagonalisiert (siehe (2.4))

  $$Q^{-1}\frac{\partial f_1(\overline{u}_{ijk})}{\partial u}Q = \Lambda\,,$$

  wobei Λ die Diagonalmatrix ist, die aus den Eigenwerten der Jacobimatrix besteht. Hier sei wiederholt, dass die Spalten von Q die rechten Eigenvektoren von $\frac{\partial f_1(\overline{u}_{ijk})}{\partial u}$ und die Zeilen von Q^{-1} die linken Eigenvektoren sind.

- Transformiere alle benötigten zu limitierende Größen, z.B. die drei Größen u^x_{ijk}, $\overline{u}_{i+1,j,k} - \overline{u}_{i,j,k}$ und $\overline{u}_{i,j,k} - \overline{u}_{i-1,j,k}$ in Richtung der charakteristischen Variablen. Dies erreicht man, indem man jeweils den Vektor von links mit Q^{-1} multipliziert.

  $$\tilde{u}_1 = Q^{-1}u^x_{ijk}\,,\ \tilde{u}_2 = Q^{-1}(\overline{u}_{i+1,j,k} - \overline{u}_{i,j,k})\,,\ \tilde{u}_3 = Q^{-1}(\overline{u}_{i,j,k} - \overline{u}_{i-1,j,k})$$

- Wende nun den skalaren Limiter (4.25) auf jede Komponente des transformierten Vektors an.

- Das Ergebnis der Limitierung wird in den ursprünglichen Raum zurücktransformiert, indem mit Q von links multipliziert wird.

  $$u^x = Q \cdot \overline{m}_{vek}(\tilde{u}_1, \tilde{u}_2, \tilde{u}_3)\,.$$

  Dabei ist $\overline{m}_{vek}$ so definiert: Für jede Komponente der Vektoreinträge nehme die skalare modifizierte minmod-Funktion aus (4.25).

Dreiecke und Tetraeder

Für die Limitierung bei Dreiecken oder Tetraedern verweisen wir zunächst auf die Idee von Cockburn [17]. In dieser Arbeit wird die oben beschriebene Methode für Rechtecke auf Dreieckstriangulierungen angepasst. Dabei wird Gebrauch von sogenannten B-Triangulierungen [16] gemacht. Die Dreiecke erfüllen dabei gewisse Winkelbedingungen. So sind z.B. Triangulierungen, die durch Teilen von Rechtecken an der Diagonalen entstanden sind, B-Triangulierungen. Teilt man diese Dreiecke jedoch noch ein weiteres Mal nach dem Halbierungsalgorithmus [4], so ist die entstehende Triangulierung keine B-Triangulierung mehr.

Hier in dieser Arbeit wollen wir einen Weg vorschlagen, der in gewisser Weise die Idee der ENO-/MUSCL- Verfahren [38] aufgreift. Dieser Algorithmus funktioniert für beliebige Triangulierungen. Die Näherungslösung im P^1-Fall sei gegeben durch:

$$u_h(x, y, t) = \sum_{i=1}^{3} u_i(t)\phi_i(x, y) \, .$$

Wir nehmen hier an, dass ein Dreieck drei Nachbardreiecke hat, somit also nicht eine oder mehr Kanten auf dem Rand hat. Sollte das Dreieck am Rand liegen, so kann man den folgenden Algorithmus abändern oder in diesem Dreieck unter Berücksichtigung der Erhaltungseigenschaft die konstante Lösung nehmen. Betrachten wir das Dreieck K_0. Die Nachbarn von K_0 seien K_1, K_2, K_3. Bilde in jedem Dreieck $K_i, i = 0, 1, 2, 3$ das Polynom

$$u_h(x, y, t) \mid_{K_i} = \overline{u}_i(t) + a_i(t)(x - x_{s_i}) + b_i(t)(y - y_{s_i}) \, .$$

Dabei bezeichnet (x_{s_i}, y_{s_i}) den Schwerpunkt des Dreiecks K_i. $a_i(t), b_i(t)$ sind $\in \mathbb{R}$. $\overline{u}_i(t)$ ist der Mittelwert von u_h im Dreieck K_i. Die Werte werden durch Lösen eines linearen Gleichungssystems bestimmt.

Bilde nun neue Dreiecke K_{ijk}, indem die Schwerpunkte der Dreiecke K_i, K_j, K_k miteinander verbunden werden.

Bezüglich des Dreiecks K_0 ermittele zusätzlich die drei Polynome

$$q_{012}(x,y,t)\mid_{K_{012}} = \overline{u}_0(t) + a_{012}(t)(x - x_{s_0}) + b_{012}(t)(y - y_{s_0})$$
aus den Bedingungen
$$q_{012}(x_{s_0}, y_{s_0}, t)\mid_{K_{012}} = \overline{u}_0(t)\,,$$
$$q_{012}(x_{s_1}, y_{s_1}, t)\mid_{K_{012}} = \overline{u}_1(t)\,,$$
$$q_{012}(x_{s_2}, y_{s_2}, t)\mid_{K_{012}} = \overline{u}_2(t)\,,$$

$$q_{013}(x,y,t)\mid_{K_{013}} = \overline{u}_0(t) + a_{013}(t)(x - x_{s_0}) + b_{013}(t)(y - y_{s_0})$$
aus den Bedingungen
$$q_{013}(x_{s_0}, y_{s_0}, t)\mid_{K_{013}} = \overline{u}_0(t)\,,$$
$$q_{013}(x_{s_1}, y_{s_1}, t)\mid_{K_{013}} = \overline{u}_1(t)\,,$$
$$q_{013}(x_{s_3}, y_{s_3}, t)\mid_{K_{013}} = \overline{u}_3(t)\,,$$

$$q_{023}(x,y,t)\mid_{K_{023}} = \overline{u}_0(t) + a_{023}(t)(x - x_{s_0}) + b_{023}(t)(y - y_{s_0})$$
aus den Bedingungen
$$q_{023}(x_{s_0}, y_{s_0}, t)\mid_{K_{023}} = \overline{u}_0(t)\,,$$
$$q_{023}(x_{s_2}, y_{s_2}, t)\mid_{K_{023}} = \overline{u}_2(t)\,,$$
$$q_{023}(x_{s_3}, y_{s_3}, t)\mid_{K_{023}} = \overline{u}_3(t)\,.$$

Danach benutzen wir die normale minmod-Limiter-Funktion aus Gleichung (4.23)

$$m_x = m(a_0, a_{012}, a_{013}, a_{023})$$

und

$$m_y = m(b_0, b_{012}, b_{013}, b_{023})\,.$$

Danach setzen wir

$$\Lambda\Pi_h^{(P^1)} u_h(x,y,t)\mid_{K_0} := \overline{u}_0(t) + m_x(t)(x - x_{s_0}) + m_y(t)(y - y_{s_0})\,.$$

Dies entspricht im Wesentlichen der Limitierung von Durlofsky, Engquist und Osher [23]. Es ist klar, dass dieser Algorithmus die Erhaltungseigenschaft erfüllt. Der Wert im Schwerpunkt des Dreiecks K_0 bleibt nämlich unverändert. Lediglich die Steigungsfaktoren des linearen Polynoms werden eventuell verändert. Auch werden keine neuen Extrema erzeugt. Dieser im skalaren Fall beschriebene Algorithmus kann sehr einfach auf Systeme angewendet werden, indem man jede Komponente getrennt betrachtet.

Limitierung bei quadratischen Funktionen

Am Anfang von Abschnitt 4.8.2 wurde die Limitierung einer Funktion aus P^1

$$u_h(x,y,z,t) = \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y) + u^z(t)\xi_k(z)$$

mittels des Operators $\Lambda\Pi_h^{(P^1)}$ besprochen. Hier soll nun beschrieben werden, wie Funktionen aus P^2

$$\begin{aligned} u_h(x,y,z,t) &= \overline{u}(t) + u^x(t)\phi_i(x) + u^y(t)\psi_j(y) + u^z(t)\xi_k(z) \\ &\quad + u^{xy}(t)\phi_i(x)\psi_j(y) + u^{xz}(t)\phi_i(x)\xi_k(z) + u^{yz}(t)\psi_j(y)\xi_k(z) \\ &\quad + u^{xx}(t)(\phi_i^2(x) - \tfrac{1}{3}) + u^{yy}(t)(\psi_j^2(y) - \tfrac{1}{3}) + u^{zz}(t)(\xi_k^2(z) - \tfrac{1}{3}) \end{aligned}$$

mittels eines Operators $\Lambda\Pi_h^{(P^2)}$ limitiert werden.
Sei das Polynom $u_h = u_h^0 + u_h^1 + u_h^2$. Dabei sei

$$u_h^0 = \overline{u}(t)\,,$$

$$u_h^1 = u^x(t)\phi_i(x) + u^y(t)\psi_j(y) + u^z(t)\xi_k(z)$$

und

$$\begin{aligned} u_h^2 &= u^{xy}(t)\phi_i(x)\psi_j(y) + u^{xz}(t)\phi_i(x)\xi_k(z) + u^{yz}(t)\psi_j(y)\xi_k(z) \\ &\quad + u^{xx}(t)(\phi_i^2(x) - \tfrac{1}{3}) + u^{yy}(t)(\psi_j^2(y) - \tfrac{1}{3}) + u^{zz}(t)(\xi_k^2(z) - \tfrac{1}{3}) \end{aligned}\,.$$

Um nun die Projektion $\Lambda\Pi_h^{(P^2)}u_h$ auszurechnen, machen wir die Annahme, dass Oszillationen, die in u_h auftauchen, in u_h^1 zu beobachten sind. Deswegen machen wir im Fall

$$(u_h^0 + u_h^1) = \Lambda\Pi_h^{(P^1)}(u_h^0 + u_h^1)$$

keine Limitierung, weil wir annehmen, dass keine Oszillationen auftreten und setzen

$$\Lambda\Pi_h^{(P^2)}u_h := u_h\,.$$

Im anderen Fall

$$(u_h^0 + u_h^1) \neq \Lambda\Pi_h^{(P^1)}(u_h^0 + u_h^1)$$

löschen wir den u_h^2 Teil, setzen also $u_h = u_h^0 + u_h^1$ und limitieren den verbleibenden P^1 Teil wie oben beschrieben (4.8.2). Wir erhalten

$$\Lambda\Pi_h^{(P^2)}u_h := \Lambda\Pi_h^{(P^1)}(u_h^0 + u_h^1)\,.$$

Wenn wir die in $1D$ gültigen Eigenschaften für Limiter auch in $2D$ annehmen, erhalten wir folgende Aussage: Die Limiter $\Lambda\Pi_h^{(P^1)}$ und $\Lambda\Pi_h^{(P^2)}$ sind TVB-Limiter, jedoch keine TVD-Limiter. Dies folgt aus der Definition des Limiters (4.26), der Definition 4.25 und Satz 4.8.2. Numerische Tests lassen uns annehmen, dass diese Eigenschaften aus $1D$ auch in $2D$ gelten.

Kapitel 5

Parallelisierung

Die numerische Simulation von komplexen Strömungen mit den hier vorgestellten expliziten numerischen Verfahren erfordert sehr lange Rechenzeiten, wenn man auf einem sehr feinen Gitter arbeitet. Wenn man das Gitter immer feiner wählt, verkleinert sich ebenfalls aufgrund der CFL-Bedingung die Zeitschrittweite der Verfahren, was zusätzlich zu einer längeren Laufzeit führt, weil bis zu einem definierten Endzeitpunkt mehr Zeitschritte gerechnet werden müssen. Dies führt dazu, dass komplexe Simulationen auf einzelnen Workstations nicht mehr in annehmbarer Zeit gerechnet werden können. Die Idee liegt nun nahe, dass die Arbeit des Verfahrens auf mehrere Prozessoren verteilt wird. Diese Idee wird auf einem Parallelrechner umgesetzt. Deshalb sprechen wir von Parallelisierung.
Die Vorteile der Parallelisierung von hyperbolischen Problemen mit expliziten Verfahren gegenüber der Parallelisierung mit impliziten Verfahren oder auch elliptischen oder parabolischen Problemen sind offensichtlich. Es sind keine Gleichungssysteme mit Iterationsverfahren zu lösen. Die numerische Lösung zu einem neuen Zeitschritt kann durch die Lösung zu einem älteren Zeitpunkt definiert werden (siehe Definitionen 2.4.2 , 2.5.1 , 2.5.2 oder 4.8.1). Dabei liegt der eigentliche Vorteil in der Beschränktheit des Abhängigkeitsgebietes.

5.1 Message Passing Interface

Zunächst betrachten wir die Parallelisierung unter dem Parallelisierungswerkzeug ”Message Passing Interface” (MPI). Dies ist ein Beispiel eines verteilten Systems. Zunächst muss das Rechengitter partitioniert werden. Diese grundlegende Idee ist die Gebietszerlegung: Das Rechengitter wird in Teilgebiete zerlegt, die dann auf die jeweiligen Prozessoren des Parallelrechners verteilt werden. Die einzelnen Prozessoren rechnen dann parallel zueinander auf den entsprechenden Teilgebieten. Die Partitionierung des Gebietes wurde mit der Partitionierungssoftware PARTY [48] unternommen. Mit verschiedenen Partitionierungsalgorithmen, die auch in [53] beschrieben sind, werden dabei jedem Element des

Gesamtgebietes ein Prozessor zugeordnet. Da zum Berechnen der numerischen Flüsse die Werte auf den Nachbarelementen auch benötigt werden, wird ein Teilgebiet um jeweils alle Nachbarelemente der zu diesem Teilgebiet gehörigen Elemente erweitert. Diese dazukommenden Elemente nennen wir *Geisterelemente* , weil sie eigentlich zu einem anderen Teilgebiet gehören und hier nur als Kopie existieren. Abbildung 5.1 verdeutlicht dies.

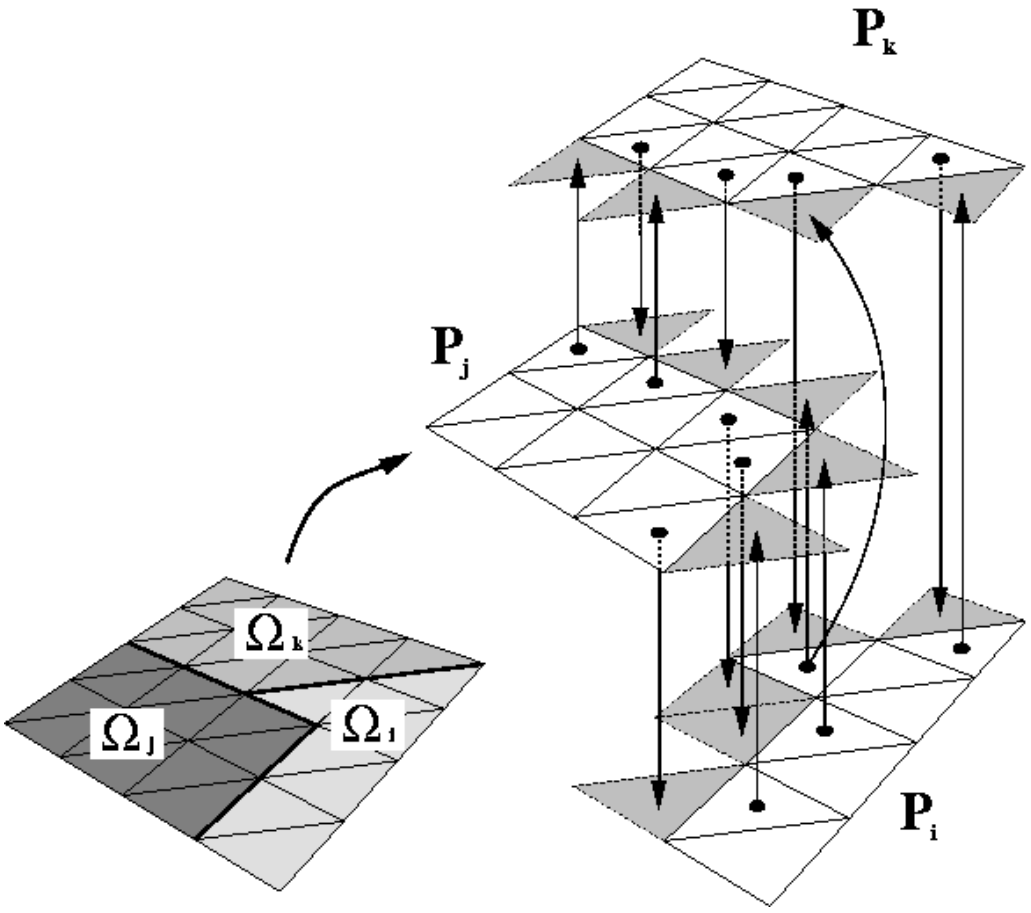

Abbildung 5.1: Datenaustausch zwischen den Teilgebieten

Abb. 5.1 zeigt die Kommunikation zwischen den Teilgebieten und den damit verbundenen Datenaustausch. Die Grafik wurde freundlicherweise zur Verfügung gestellt von R. Schwörer [53].

So kann jeder Prozessor unabhängig von den anderen seine Berechnungen durchführen. Lediglich nach jedem Zeitschritt ist ein Kommunizierungsschritt erforderlich. Bei diesem legt jedes Element, das Kopien auf anderen Teilgebieten hat, den Originalwert auf diese Kopien.
Algorithmus (5.1) beschreibt das serielle numerische Verfahren erster Ordnung.
Algorithmus (5.2) beschreibt das parallele numerische Verfahren erster Ordnung mit MPI.

Algorithmus (5.3) beschreibt das serielle numerische Verfahren höherer Ordnung.
Algorithmus (5.4) beschreibt das parallele numerische Verfahren höherer Ordnung mit MPI.

serielles Verfahren erster Ordnung

```
∀ Elemente i setze Anfangswerte u(i)
t = 0
do
{
        bestimme Δt durch CFL-Bedingung

        ∀ Elemente i
        {
                update(i)=0
        }

        ∀ Elemente i
        {
                ∀ Nachbarn j=neighbour(i,l)
                {
                                Falls (j<0)
                                {
                                        berechne Fluss über Randkante von Element i
                                        update(i) += fluss
                                }
                                Falls (j>i)
                                {
                                        berechne Fluss über Kante zwischen (i,j)
                                        update(i) += fluss
                                        update(j) -= fluss
                                }
                }
        }

        ∀ Elemente i
        {
                u(i) += update(i)
        }

        t += Δt
} Falls t < t_end, gehe an Schleifenanfang.
```

Tabelle 5.1: Pseudo-Algorithmus für serielles Verfahren erster Ordnung.

paralleles Verfahren erster Ordnung auf k Prozessoren (MPI)

zerlege Gesamtgitter in r Teilgitter $G_1, \ldots, G_r$
jedes Teilgitter ist für sich konsistent
ein Teilgitter G_k bestehe aus N_k Elementen
dabei sind die Elemente mit den Indizes 0 bis $firstghost - 1$ Originale
die Elemente mit den Indizes $firstghost$ bis $N_k - 1$ sind Kopien

$\forall$ Elemente $i = 0, \ldots, N_k - 1$ setze Anfangswerte u(i)

```
t = 0
do
{
    bestimme (Δt)_k durch CFL-Bedingung
    Kommunikation
     Δt := min_k(Δt)_k

    ∀ Elemente i = 0, ..., N_k - 1
    {
        update(i)=0
    }

    ∀ Elemente i = 0, ..., firstghost - 1
    {
        ∀ Nachbarn j=neighbour(i,l)
        {
                Falls (j<0)
                {
                    berechne Fluss über Randkante von Element i
                    update(i) += fluss
                }
                Falls (j>i)
                {
                    berechne Fluss über Kante zwischen (i,j)
                    update(i) += fluss
                    update(j) -= fluss
                }
        }
    }

    ∀ Elemente i = 0, ..., firstghost - 1
```

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

{
u(i) += update(i)
}

Kommunikation
$\forall$ **Elemente** $i = 0, \ldots, firstghost - 1$
{
kopiere Einträge auf die auf anderen Prozessoren liegenden Kopien (sofern solche existieren)
}

oder

Kommunikation
$\forall$ **Elemente** $i = firstghost, \ldots, N_k - 1$
{
hole Einträge von den auf anderen Prozessoren liegenden Originalen
}

t += Δt
} Falls $t < t_{end}$, gehe an Schleifenanfang.

Tabelle 5.2: Pseudo-Algorithmus für paralleles Verfahren erster Ordnung mit MPI.

serielles Discontinuous Galerkin Verfahren höherer Ordnung

```
∀ Elemente i und Basisfunktionen b setze Anfangswerte u(i,b)
t = 0
do
{
    bestimme Δt durch CFL-Bedingung

    Für Runge-Kutta r = 0, ..., R
      (R=1 für Verfahren zweiter Ordnung,R=2 für Verfahren dritter
      Ordnung)
    {
        ∀ Elemente i und Basisfunktionen b
        {
            update(i,b)=0
        }

        ∀ Elemente i
        {
            ∀ Nachbarn j=neighbour(i,l)
            {
                Falls (j<0)
                {
                    berechne Fluss über Randkante von Element i
                    update(i,b) += fluss
                }
                Falls (j>i)
                {
                    berechne Fluss über Kante zwischen (i,j)
                    update(i,b) += fluss
                    update(j,b) -= fluss
                }
            }
        }

        ∀ Elemente i und Basisfunktionen b
        {
            u(i,b) += update(i,b) (Abhängigkeit von r beachten)
        }
```

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

∀ Elemente i
{
Projektion
}

} (Ende der Runge-Kutta-Schleife)
t += Δt
} Falls $t < t_{end}$, gehe an Schleifenanfang.

Tabelle 5.3: Pseudo-Algorithmus für serielles Discontinuous Galerkin Verfahren höherer Ordnung.

paralleles Discontinuous Galerkin Verfahren höherer Ordnung auf k Prozessoren

zerlege Gesamtgitter in r Teilgitter $G_1, \ldots, G_r$
jedes Teilgitter ist für sich konsistent
ein Teilgitter G_k bestehe aus N_k Elementen
dabei sind die Elemente mit den Indizes 0 bis $firstghost - 1$ Originale
die Elemente mit den Indizes $firstghost$ bis $N_k - 1$ sind Kopien

$\forall$ Elemente $i = 0, \ldots, N_k - 1$ und Basisfunktionen b setze Anfangswerte u(i,b)

```
t = 0
do
{
    bestimme (Δt)_k durch CFL-Bedingung
    Kommunikation
     Δt := min_k(Δt)_k

    Für Runge-Kutta r = 0,...,R
       (R=1 für Verfahren zweiter Ordnung,R=2 für Verfahren dritter
       Ordnung)
    {
        ∀ Elemente i = 0,...,N_k − 1 und Basisfunktionen b
        {
            update(i,b)=0
        }

        ∀ Elemente i = 0,...,firstghost − 1
        {
            ∀ Nachbarn j=neighbour(i,l)
            {
                Falls (j<0)
                {
                    berechne Fluss über Randkante von Element i
                    update(i,b) += fluss
                }
                Falls (j>i)
                {
                    berechne Fluss über Kante zwischen (i,j)
                    update(i,b) += fluss
                    update(j,b) -= fluss
```

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

```
                        }
                }
        }

        ∀ Elemente i = 0, ..., firstghost − 1
        {
                u(i,b) += update(i,b) (Abhängigkeit von r beachten)
        }

        Kommunikation
        ∀ Elemente i = 0, ..., firstghost − 1
        {
                kopiere Einträge auf die auf anderen Prozessoren
                liegenden Kopien (sofern solche existieren)
        }
```

oder

```
        Kommunikation
        ∀ Elemente i = firstghost, ..., N_k − 1
        {
                hole Einträge von den auf anderen Prozessoren
                liegenden Originalen
        }

        ∀ Elemente i
        {
                Projektion
        }

        Kommunikation
        ∀ Elemente i = 0, ..., firstghost − 1
        {
                kopiere Einträge auf die auf anderen Prozessoren
                liegenden Kopien (sofern solche existieren)
        }
```

oder

```
        Kommunikation
        ∀ Elemente i = firstghost, ..., N_k − 1
```

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

{

hole Einträge von den auf anderen Prozessoren liegenden Originalen

}

} (Ende der Runge-Kutta-Schleife)

t += Δt

} Falls $t < t_{end}$, gehe an Schleifenanfang.

Tabelle 5.4: Pseudo-Algorithmus für paralleles Verfahren höherer Ordnung mit MPI.

5.2 Shared Memory

Das Konzept bei der Parallelisierung unter "Shared Memory" ist auf den ersten Blick sehr viel einfacher. Jeder Prozessor kann zu jeder Zeit auf den kompletten Hauptspeicher zugreifen. Deshalb muss das Rechengebiet nicht explizit partitioniert werden. Sei $\#P$ die Anzahl der Prozessoren und $\#N$ die Anzahl der Elemente. Dann rechnet jeder Prozessor die Flüsse für $\left[\frac{\#N}{\#P}\right]$ Elemente aus. Dabei ist $\left[\frac{\#N}{\#P}\right]$ der ganzzahlige Anteil von $\frac{\#N}{\#P}$. Sei $p := \left[\frac{\#N}{\#P}\right] + 1$. Dann rechnet der k-te Prozessor die Flüsse für Element $k \cdot p$ bis Element $\min((k+1) \cdot p, \#N)$, $k = 0, \ldots, \#P - 1$ aus. Hierbei ist nun folgendes zu beachten. Wenn ein Prozessor auf eine Speicheradresse, die zum selben Zeitpunkt auch von einem anderen Prozessor genutzt werden kann, schreiben will, so muss diese blockiert werden.

Durch den Einsatz von Parallelrechnern kann man nun die Laufzeit der Verfahren sehr vermindern. Im Idealfall würde man erwarten, dass bei der Verwendung von N mal so vielen Prozessoren die Laufzeit mit dem Faktor $1/N$ multipliziert wird. Dies ist allerdings nur theoretisch möglich und wird in der Praxis nicht erreicht, weil man zwischen den Zeitschritten die Kommunikation (MPI) bzw. das Blockieren von Speicheradressen beachten muss. Je mehr Prozessoren verwendet werden, desto mehr Kommunikation untereinander ist von Nöten. In Abbildung 5.2 ist die Skalierung des Discontinuous Galerkin Verfahrens unter MPI abzulesen. In Tabelle 5.5 sind die jeweiligen CPU-Zeiten bei Rechnung auf 2 bis zu 128 Prozessoren abzulesen.

Jeder Prozessor muss vom jeweiligen Prozessortyp abhängend ein gewisses Maß an numerischen Rechnungen zu bewältigen haben, sodass nicht die meiste Zeit wieder in die Kommunikation investiert werden muss.

Auf den Prozessoren, auf denen wir gerechnet haben, ergab sich aus Test eine Größenordnung von ca. 10000 Elementen. In diesem Fall dauerten die Rechnungen zwischen den Kommunikationsschritten so lang, dass nicht die meiste Zeit in die Kommunikation investiert werden musste.

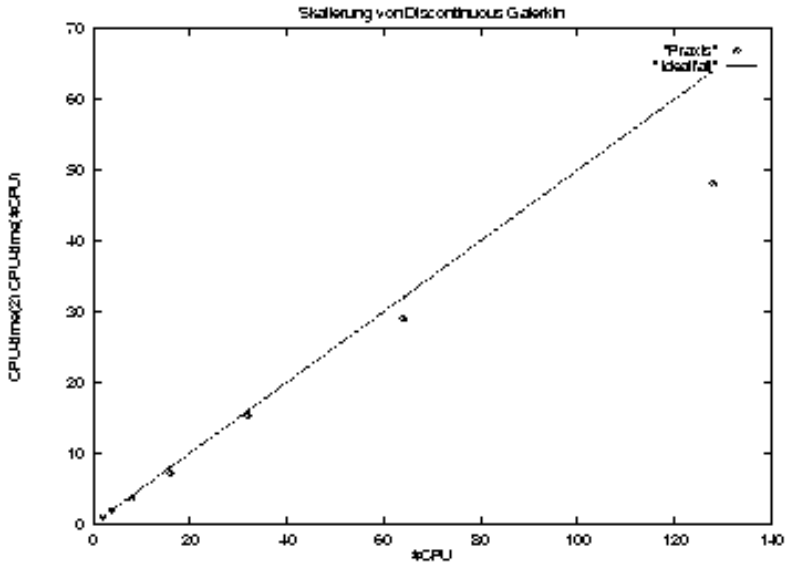

Abbildung 5.2: Skalierung des Discontinuous Galerkin Verfahren zweiter Ordnung unter MPI.

Abb. 5.2 zeigt die Skalierung des Discontinuous Galerkin Verfahren zweiter Ordnung unter MPI. Es wurde das Forward Facing Step Problem (siehe Kapitel 7) mit 258048 Elementen betrachtet. Ausgewertet wurde nach 527 Zeitschritten zur Zeit $t = 0.080032$ und wurde auf 2 bis 128 Prozessoren gerechnet. Siehe auch die Tabelle. Auf der x-Achse ist die Anzahl der Prozessoren aufgetragen. Auf der y-Achse ist das Verhältnis $\frac{CPU-time(2)}{CPU-time(P)}$ aufgetragen. An der Lage der Punkte kann man sehen, dass das Verfahren fast optimal skaliert.

Wenn wir nun erneut Abbildung 5.1 betrachten, so sagen wir, das hier eine *Geisterebene* der Größe eins vorliegt. Bei der Verwendung eines Verfahrens erster Ordnung benötigt man zur Flussberechnung nur die Informationen in den direkten Nachbarzellen. D.h. man muss wie oben beschrieben nur nach jedem Zeitschritt die Kopien mit den Originalwerten aktualisieren. Will man nun ein Verfahren höherer Ordnung wie MUSCL oder ENO parallelisieren, so kann die Größe der Geisterebene sehr schnell sehr groß werden, weil man zur Rekonstruktion der Polynome höheren Grades aus den stückweise konstanten Werten z.B. auch die Nachbarn der Nachbarn benötigt. So kann je nach Ordnung des Verfahrens die Anzahl der Geisterelemente sehr groß werden, was natürlich einen Mehraufwand an Kommunikation bedeutet. Außerdem können die partitionierten Gitter des Verfahrens erster Ordnung nicht benutzt werden.
Bei den Discontinuous Galerkin Verfahren besteht dieses Problem nicht. Da die Information über das jeweilige Polynom höheren Grades immer lokal in einer Zelle steckt, benötigt man wie im Verfahren erster Ordnung nur die Nachbarn eines Elementes zur Flussberechnung und kommt somit mit einer Geisterebene aus, d.h man kann die Partitionierungen der Verfahren erster Ordnung weiter benutzen. Dies ist ein weiterer sehr großer Vorteil der Discontinuous Galerkin Verfahren gegenüber den anderen Verfahren höherer

Anzahl Prozessoren	CPU-Zeit in sec.	ungefähre Anzahl der Elemente pro Prozessor
2	8650	129128-129391
4	4371	64512-65024
8	2338	32289-32594
16	1227	16145-16668
32	569	8071-8566
64	299	4037-4351
128	193	2019-2210

Tabelle 5.5: CPU-Zeiten bei Rechnungen mit verschiedener Anzahl von Prozessoren mit dem MPI-Verfahren.

Ordnung wie MUSCL oder ENO.
Algorithmus (5.6) beschreibt das parallele numerische Verfahren erster Ordnung mit dem Shared Memory Modell.

paralleles Verfahren erster Ordnung auf $NCPU$ Prozessoren (Shared Memory)

keine explizite Zerlegung des Gitters nötig
Gitter bestehe aus N Elementen

$\forall$ Elemente $i = 0, \ldots, N-1$ setze Anfangswerte u(i)
$ncpu = N/NCPU + 1$

$t = 0$
do
{
(seien wir nun auf dem k-ten Prozessor)
(Blockiere Speicherstelle von X bedeutet: Nur Prozessor k darf auf diese Stelle zugreifen)
(Barriere: Erst, wenn alle Prozessoren diese Stelle errreichen, geht der Algorithmus weiter)

bestimme $(\Delta t)_k$ durch CFL-Bedingung
Blockiere Speicherstelle von Δt
$\Delta t := \min((\Delta t)_k, \Delta t)$
Gebe Speicherstelle von Δt frei

Barriere
$\forall$ Elemente $i = k * ncpu, \ldots, \min((k+1) * ncpu, N) - 1$
{
update(i)=0
}

Barriere
$\forall$ Elemente $i = k * ncpu, \ldots, \min((k+1) * ncpu, N) - 1$
{
$\forall$ Nachbarn j=neighbour(i,l)
{
Falls (j<0) oder (j>i)
{
Falls (j<0)
{
berechne Fluss über Randkante von Element i
Blockiere Speicherstelle von update(i)
update(i) += fluss

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Gebe Speicherstelle von update(i) frei
}
Falls (j>i)
{
berechne Fluss über Kante zwischen (i,j)
Blockiere Speicherstelle von update(i)
update(i) += fluss
Gebe Speicherstelle von update(i) frei
Blockiere Speicherstelle von update(j)
update(j) -= fluss
Gebe Speicherstelle von update(j) frei
}
}
}
}

Barriere
$\forall$ Elemente $i = k * ncpu, \ldots, \min((k+1) * ncpu, N) - 1$
{
u(i) += update(i)
}

Barriere
t += Δt
} Falls $t < t_{end}$, gehe an Schleifenanfang.

Tabelle 5.6: Pseudo-Algorithmus für paralleles Verfahren erster Ordnung mit Shared Memory.

Partitionierungen der verschiedenen in dieser Arbeit verwendeten Gitter sind bei den Lösungen im Kapitel über die gerechneten Beispiele zu finden.

Kapitel 6

Konvergenzbestimmung in L^1

Bei der Programmierung und der Darstellung der Ergebnisse haben wir, wie schon im ersten Kapitel erwähnt, das Grafikpaket GRAPE [51], [28] benutzt, welches am Sonderforschungsbereich 256 der Universität Bonn gemeinsam mit dem Institut für Angewandte Mathematik in Freiburg seit Jahren entwickelt wird.

6.1 Shock Tube

6.1.1 Beschreibung des Problems

Das Shock Tube-Problem nach Sod [57], besteht darin, dass ein Gebiet in zwei Teilgebiete zerlegt wird, in denen jeweils zur Zeit $t = 0$ konstante Zustände vorliegen. Diese beiden Gebiete werden durch eine dünne Membran getrennt. Nun platzt diese Membran und es entstehen drei Wellen. Diese Wellen heißen Verdünnungswelle, Kontaktunstetigkeit und Schock. (siehe Abb.6.1)

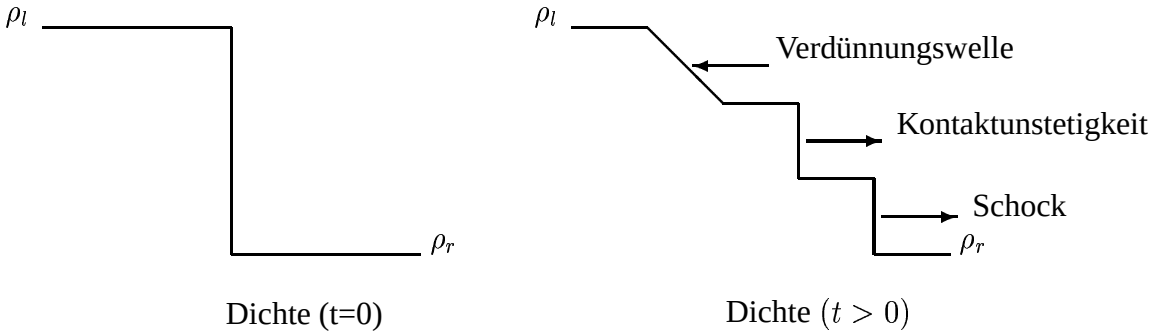

Abbildung 6.1: Wellen bei Shock Tube

Ziel numerischer Verfahren ist es z.B., den Schock so gut darzustellen, dass er über möglichst wenig Gitterpunkte verschmiert. Dieses Problem wird in 1D gerechnet, wobei z.B. für alle $x < 0$ Zustand 1 und für alle $x > 0$ Zustand 2 gilt. Man kann dieses Beispiel aber auch in 2D oder 3D rechnen. Dort nimmt man z.B. für festes x für alle y, z den gleichen Wert. Wir haben das folgende Beispiel gerechnet.
$\Omega = [-1.0; 1.0] \times [0.0; 1.0] \times [0.0; 1.0]$. Dabei gelten folgende Anfangswerte :

$$\rho(x, y, z, 0) = \begin{cases} 4 & , x < 0 \\ 1 & , x > 0 \end{cases} ,$$

$$p(x, y, z, 0) = \begin{cases} 1.6 & , x < 0 \\ 0.4 & , x > 0 \end{cases} ,$$

$$u_1(x, y, z, 0) = u_2(x, y, z, 0) = u_3(x, y, z, 0) = 0 .$$

Bei diesem Beispiel werten wir die numerische Lösung in $t = 0.75$ aus und vergleichen sie mit der ”exakten” Lösung 6.1.2.
Die anderen Größen können aus diesen vorgegebenen ausgerechnet werden. Wir nehmen folgende Randbedingungen: Am linken und rechten Rand nehmen wir Ausfluss-Bedingungen. An den anderen Rändern nehmen wir die Bedingung $\vec{n} \cdot \vec{u} = n_1u_1 + n_2u_2 + n_3u_3 = 0$. Dadurch wird eine undurchlässige Wand simuliert.

6.1.2 Exakte Lösung nach Chorin

Für das Riemann-Problem in 1D hat Chorin [9] einen Algorithmus vorgestellt, mit dem man zu einer Zeit t die Werte am Ort x iterativ mit vorgegebener Genauigkeit bestimmen kann. Diese Werte werden dann als ”exakte” Lösung von Shock Tube verwandt. Dieses Verfahren haben wir benutzt, um im folgenden die numerische Konvergenzordnung zu bestimmen. Im folgenden nennen wir die Lösung nach Chorin die exakte Lösung.

6.1.3 Numerische Konvergenzordnung in L^1

Wir bestimmen die numerische Konvergenzordnung für die Verfahren erster Ordnung und für die Discontinuous Galerkin Verfahren zweiter und dritter Ordnung. Dazu vergleichen wir die numerische Lösung mit der exakten Lösung nach Chorin. Dann haben wir die Möglichkeit, die numerische bzw. experimentelle Konvergenzordnung (EOC) zu bestimmen.
Für jede Zelle wird der x-Wert des Schwerpunktes bestimmt. In diesem x-Wert wird dann die exakte Lösung mit dem Algorithmus von Chorin bestimmt und dieser Wert mit der numerischen Lösung in dieser Zelle verglichen. So haben wir den Fehler auf dem gesamten Gebiet mit Hilfe einer Integrationsformel bestimmt, die für Polynome vom Grad ≤ 1 exakt ist.

$$\int_T \hat{\phi}(x)dx \sim \mid T \mid \hat{\phi}(\hat{a}) .$$

Dabei bezeichnet T das Volumen der Zelle , über die integriert wird, $\hat{a}$ den Schwerpunkt der Zelle und $\hat{\phi}$ die Funktion auf dieser Zelle. In unserem Fall ist dies die Differenz zwischen exakter und numerischer Lösung.
Die EOC wird dann folgendermaßen bestimmt:
Für die Normabschätzung nehmen wir als repräsentative Größe die Dichte, da bei ihr alle drei Wellen ausgeprägt sind [38]. Der Druck und die Geschwindigkeit sind links und rechts der Kontaktunstetigkeit gleich. Im folgenden bezeichnen wir die Lösung nach Chorin mit ρ. Die numerische Lösung mit den verschiedenen Verfahren bezeichnen wir mit ρ_h.
Für ein Gitter mit Gitterweite h nimmt man folgende Abschätzung für den Fehler an:

$$\|\rho - \rho_h\|_{L^1(\Omega)} = C \cdot h^\beta \,. \tag{6.1}$$

Für ein Gitter mit doppelter Gitterweite nimmt man analog

$$\|\rho - \rho_{2h}\|_{L^1(\Omega)} = C \cdot 2^\beta \cdot h^\beta \tag{6.2}$$

an. Nun teilt man Gleichung (6.1) durch Gleichung (6.2). Dabei fällt die Konstante C und der h-Term weg und man erhält

$$\frac{\|\rho - \rho_h\|_{L^1(\Omega)}}{\|\rho - \rho_{2h}\|_{L^1(\Omega)}} = \frac{1}{2^\beta} \,. \tag{6.3}$$

Wenn man nun die Gleichung (6.3) nach β auflöst, erhält man die experimentelle Konvergenzordnung (EOC)

$$\beta = \frac{ln(\frac{\|\rho-\rho_{2h}\|_{L^1(\Omega)}}{\|\rho-\rho_h\|_{L^1(\Omega)}})}{ln(2)} \,. \tag{6.4}$$

Dabei ist die L^1-Norm in 1D bestimmt durch

$$\|\rho - \rho_h\|_{L^1(\Omega)} = \int_\Omega \mid \rho - \rho_h \mid d\mu \approx \sum_i \mid \rho^i - \rho_h^i \mid \cdot h \,. \tag{6.5}$$

Hier gilt $h = \Delta x$. ρ^i ist die exakte Lösung in der Dichte im Gitterpunkt x_i.
Für Zellen haben wir entsprechend

$$\|\rho-\rho_h\|_{L^1(\Omega)} = \int_\Omega \mid \rho-\rho_h \mid d\mu = \sum_j \int_{T_j} \mid \rho-\rho_h^j \mid d\mu \approx \sum_j \mid \rho^j-\rho_h^j \mid \cdot \mid T_j \mid \,. \tag{6.6}$$

Hier ist ρ^j die exakte Lösung in der Dichte im Schwerpunkt der Zelle T_j.
Die CPU-Zeiten sind dabei relativ. Wenn man das jeweilige Programm mit anderen Optionen compiliert, so ist die CPU-Zeit noch zu reduzieren. Außerdem gibt es heute schon extrem schnellere Rechner. Man kann an den CPU-Zeiten nur ablesen, dass sich die Faustregel bestätigt: Wenn man die Anzahl der Zellen (Quadrate in 2D) vervierfacht, so halbiert sich bei gleicher CFL-Zahl die Zeitschrittweite und somit wird die CPU-Zeit verachtfacht.

Wenn man die Anzahl der Zellen (Hexaeder in 3D) verachtfacht, so halbiert sich bei gleicher CFL-Zahl die Zeitschrittweite und somit wird die CPU-Zeit versechzehnfacht.
Der Faktor siebzehn in den Tabellen erklärt sich z.B. daraus, dass Programme mit weniger Elementen weniger Hauptspeicher benötigen und deshalb eventuell im Cache des Rechners die meisten Daten liegen bleiben.
Die folgenden Tabellen enthalten den EOC-Wert und den L^1-Fehler. Die CPU-Zeiten sind auf einer r10000 Workstation von SGI mit einem 200 MHz Prozessor gerechnet worden.

Steger-Warming (erste Ordnung) nach t=0.75			CFL-Zahl=0.48
Anzahl der Elemente	16000	128.000	1.024.000
entspricht Gitterweite	h=0.05	h=0.025	h=0.0125
L^1-Fehler in 3D	0.305139	0.209788	0.166239
EOC	—	0.54053	0.33567
CPU-Zeit	0:12 min	3:27 min	60:30 min

Tabelle 6.1: Testergebnisse mit Verfahren erster Ordnung in 3D auf Hexaedern.

Steger-Warming (Discontinuous Galerkin 2.Ordnung) nach t=0.75			CFL-Zahl=0.2
Anzahl der Elemente	16000	128.000	1.024.000
entspricht Gitterweite	h=0.05	h=0.025	h=0.0125
L^1-Fehler in 3D	0.074889	0.038426	0.020399
EOC	—	0.962671	0.91358
CPU-Zeit	8:56 min	152:19 min	2596:56 min

Tabelle 6.2: Testergebnisse mit Verfahren zweiter Ordnung in 3D auf Hexaedern.

Durch Benutzen eines Verfahrens höherer Ordnung wird der L^1-Fehler absolut gesehen bei gleichbleibendem Gitter erheblich verkleinert und die experimentelle Konvergenzordnung erhöht sich von 0.54 (first order) auf 0.96 (DG 2.Ordnung). Wenn man allerdings nicht die Laufzeiten innerhalb eines Verfahrens, sondern die verschiedenen Verfahren miteinander vergleicht, so muss man noch berücksichtigen, dass bei den Verfahren höherer Ordnung pro Zeitschritt zwei bzw. drei Berechnungsschritte durch die Verwendung der Runge-Kutta-Zeitdiskretisierung erforderlich sind.
Zum Schluss dieses Kapitel noch eine Bemerkung. Unserer Meinung nach sagt der Wert der experimentellen Konvergenzordnung zwar etwas über das verwendete Verfahren aus,

Steger-Warming (Discontinuous Galerkin 3.Ordnung) nach t=0.75			CFL-Zahl=0.18
Anzahl der Elemente	16000	128.000	1.024.000
entspricht Gitterweite	h=0.05	h=0.025	h=0.0125
L^1-Fehler in 3D	0.075598	0.038645	0.020754
EOC	—	0.97348	0.88959
CPU-Zeit	39:13 min	693:02 min	ca. 204 h

Tabelle 6.3: Testergebnisse mit Verfahren dritter Ordnung in 3D auf Hexaedern.

aber bei weitem halten wir ihn für nicht so wichtig, wenn man ihn als alleinige Information über ein Verfahren erhält. Man muss parallel immer auch den absoluten Fehler eines Verfahren auf einem bestimmten Gitter betrachten. Es kann nämlich vorkommen, dass bei Erweiterung eines Verfahrens von erster Ordnung auf höhere Ordnung die numerische Konvergenzordnung nur unwesentlich verbessert wird, der absolute Fehler aber fast halbiert wird. Dies wäre dann unserer Meinung nach ein gutes Ergebnis, was durch die Konvergenzordnung alleine nicht erkennbar ist.

Wenn man sich Qualität der Lösungen, die Rechenzeiten und den Speicherbedarf der einzelnen verwendeten Verfahren anschaut, so muss man sagen, dass der Aufwand für die Erweiterung von erster auf zweite Ordnung durch die eindrucksvollen Lösungen schon gerechtfertigt scheint. Die Erweiterung auf dritte Ordnung erscheint jedoch nicht mehr ganz so effektiv, weil der Gewinn durch die sehr große Laufzeit des Verfahrens eher gering ist. In Tabelle 6.3 sieht man sogar, dass sich die Größenordnung der L^1-Fehler im Vergleich zu Tabelle 6.2 nicht mehr verändert, ja sogar im Einzelfall schlechter wird.

6.1.4 Vergleich verschiedener Verfahren höherer Ordnung

In diesem Abschnitt sollen die Discontinuous Galerkin Verfahren mit den Verfahren erster Ordnung und den MUSCL-Verfahren explizit verglichen werden. Die Frage ist zum Beispiel folgende: Rechne auf einem Gitter mit den Discontinuous Galerkin Verfahren und erhalte einen L^1-Fehler. Wie fein und wie lange muss man rechnen, um mit dem Verfahren erster Ordnung und dem MUSCL-Verfahren diesen L^1-Fehler auch zu erreichen. Betrachte dazu die folgenden Tabellen. Alle Rechnungen wurden auf einem Rechteckgitter in 2D mit der Flussfunktion von Steger und Warming auf einer SGI r10000 Workstation gemacht. Dabei wurde das oben beschriebene Shock Tube Problem nach $t = 0.75$ ausgewertet und mit der exakten Lösung nach Chorin verglichen.
Nehmen wir als Referenz den L^1-Fehler vom Discontinuous Galerkin Verfahren zweiter Ordnung bei $\Delta x = \frac{1}{40}$. Dafür benötigt das Programm 54 Sekunden. Um die vergleichbare Genauigkeit beim MUSCL-Verfahren zu erreichen, müsste man die Gitterweite von ca.

$\Delta x = \frac{1}{130}$ nehmen und ca. 8 Minuten rechnen. Beim Verfahren erster Ordnung müsste man eine Gitterweite von $\Delta x = \frac{1}{640}$ nehmen und ca. 2 Stunden und 10 Minuten rechnen. Wie die Rechnungen mit dem Forward Facing Step Problem (siehe nächstes Kapitel) aber zeigen, kann man bei Verfahren erster Ordnung durch ein sehr feines Gitter die Schärfe von den Schocks erhöhen, aber in der Kontaktunstetigkeit ist das Phänomen des Wirbels nicht auflösbar. Dies ist ein weiteres Argument für die Discontinuous Galerkin Verfahren. Bei Verfahren mit zuviel numerischer Viskosität werden Details einfach weggeglättet.

Discontinuous Galerkin Verfahren zweiter Ordnung (CFL-Zahl 0.21)

$\Delta x = \Delta y$	$\frac{1}{20}$	$\frac{1}{40}$	$\frac{1}{80}$	$\frac{1}{160}$
Anzahl der Elemente	800	3200	12800	51200
Anzahl der Zeitschritte	84	169	340	682
L^1-Fehler in 2D	0.076061	0.036802	0.019854	0.010985
EOC	—	1.047373	0.890354	0.853895
CPU-Zeit (h:min:sec)	0:07	0:54	7:55	1:06:26

$\Delta x = \Delta y$	$\frac{1}{320}$	$\frac{1}{640}$
Anzahl der Elemente	204800	819200
Anzahl der Zeitschritte	1365	2732
L^1-Fehler in 2D	0.005251	0.003005
EOC	1.064871	0.805227
CPU-Zeit (h:min:sec)	9:01:03	73:51:16

Tabelle 6.4: Ergebnisse mit Discontinuous Galerkin Verfahren zweiter Ordnung

Tabelle 6.4 zeigt die Ergebnisse des Discontinuous Galerkin Verfahrens zweiter Ordnung in 2D auf einem Quadratgitter.

MUSCL Verfahren zweiter Ordnung (CFL-Zahl 0.48)

$\Delta x = \Delta y$	$\frac{1}{20}$	$\frac{1}{40}$	$\frac{1}{80}$	$\frac{1}{160}$	$\frac{1}{320}$	$\frac{1}{640}$
Anzahl der Elemente	800	3200	12800	51200	204800	819200
Anzahl der Zeitschritte	36	74	149	298	597	1195
L^1-Fehler in 2D	0.169211	0.093124	0.053351	0.029601	0.016655	0.009788
EOC	—	0.861598	0.803638	0.849869	0.829691	0.766869
CPU-Zeit (h:min:sec)	<0:01	0:06	1:04	9:37	ca. 1:21:00	ca. 9:50:00

Tabelle 6.5: Ergebnisse mit MUSCL Verfahren

Tabelle 6.5 zeigt die Ergebnisse des MUSCL Verfahrens zweiter Ordnung in 2D auf einem Quadratgitter.

Verfahren erster Ordnung (CFL-Zahl 0.48)

$\Delta x = \Delta y$	$\frac{1}{20}$	$\frac{1}{40}$	$\frac{1}{80}$	$\frac{1}{160}$	$\frac{1}{320}$	$\frac{1}{640}$
Anzahl der Elemente	800	3200	12800	51200	204800	819200
Anzahl der Zeitschritte	36	73	148	297	596	1194
L^1-Fehler in 2D	0.304227	0.209298	0.139433	0.090630	0.058376	0.037387
EOC	—	0.539590	0.585986	0.621511	0.634613	0.642839
CPU-Zeit (h:min:sec)	<0:01	0:03	0:24	3:07	25:51	ca.2:10:00

Tabelle 6.6: Ergebnisse mit Verfahren erster Ordnung

Tabelle 6.6 zeigt die Ergebnisse des Verfahrens erster Ordnung in 2D auf einem Quadratgitter.

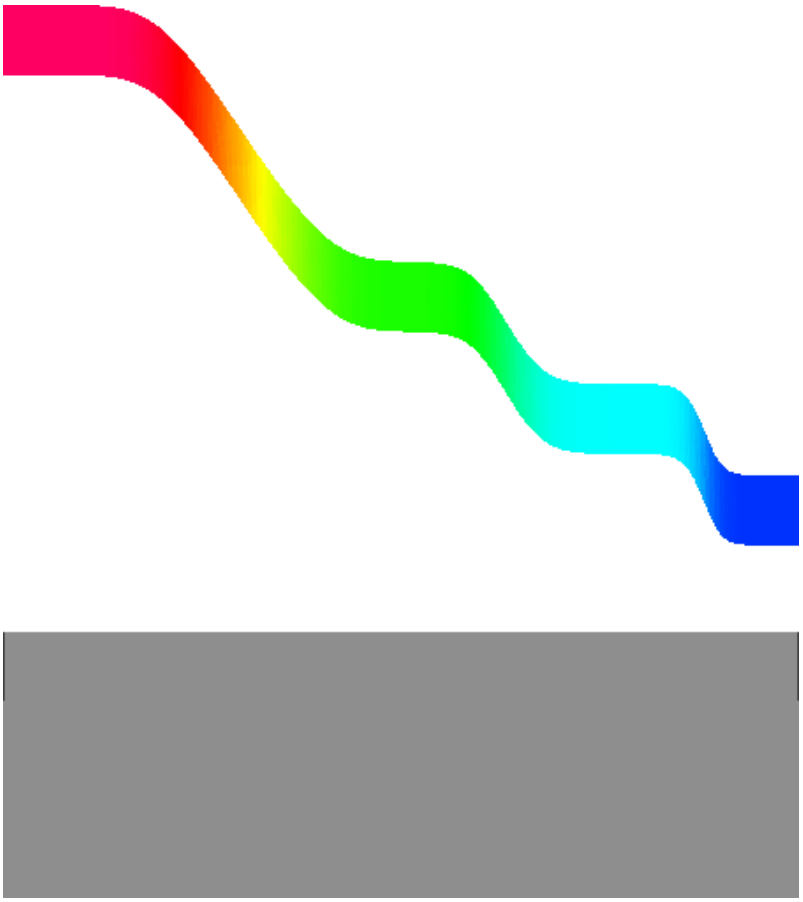

Abbildung 6.2: Dichte mit Verfahren erster Ordnung.

Abb. 6.2 zeigt einen 2D-Schnitt der numerischen Lösung der Dichte beim Shock Tube Problem in 3D. Die Rechnung wurde auf einem Hexaeder-Gitter mit 1.024.000 Elementen gemacht. Benutzt wurde ein Verfahren erster Ordnung mit numerischer Flussfunktion von Steger und Warming. Ausgewertet wurde die Lösung zum Endzeitpunkt t=0.75.

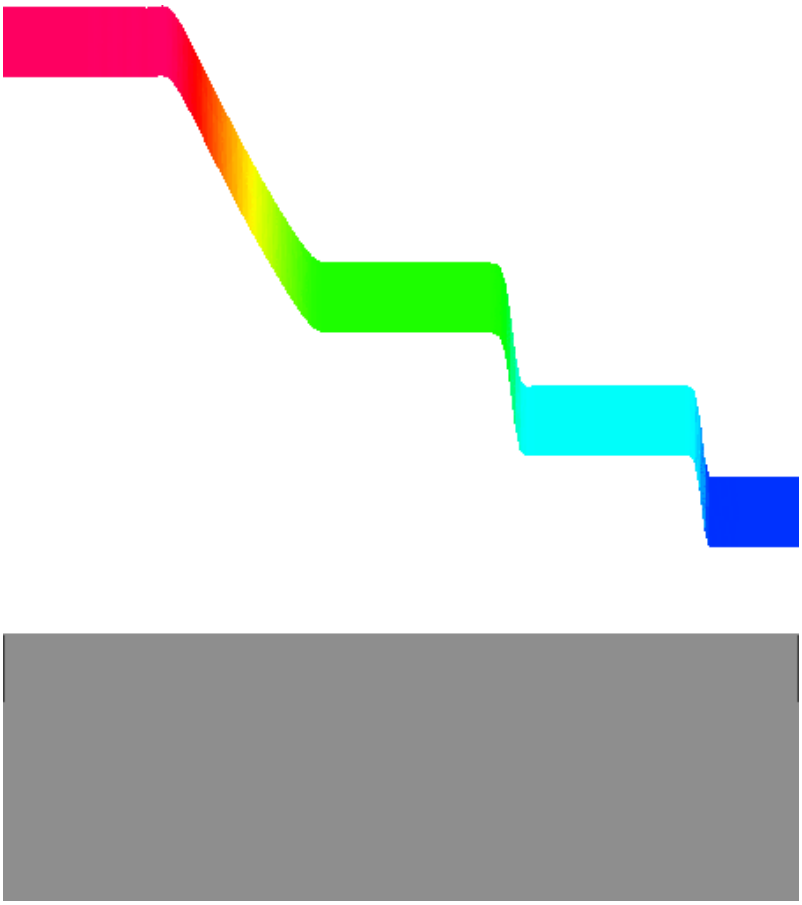

Abbildung 6.3: Dichte mit Discontinuous Galerkin Verfahren zweiter Ordnung.

Abb. 6.3 zeigt einen 2D-Schnitt der numerischen Lösung der Dichte beim Shock Tube Problem in 3D. Die Rechnung wurde auf einem Hexaeder-Gitter mit 1.024.000 Elementen gemacht. Ausgewertet wurde die Lösung zum Endzeitpunkt t=0.75. Im Gegensatz zur Lösung mit dem Verfahren erster Ordnung (siehe Abb. 6.2) sind die Kontaktunstetigkeit und der Schock viel schärfer wiedergegeben. Da sich die minimalen und maximalen Werte zwischen 0.999679 bzw. 4.010479 befinden, sind leichte Oszillationen zu erkennen. Der Parameter M aus dem Abschnitt über Limitierung wurde in dieser Rechnung 50 gewählt.

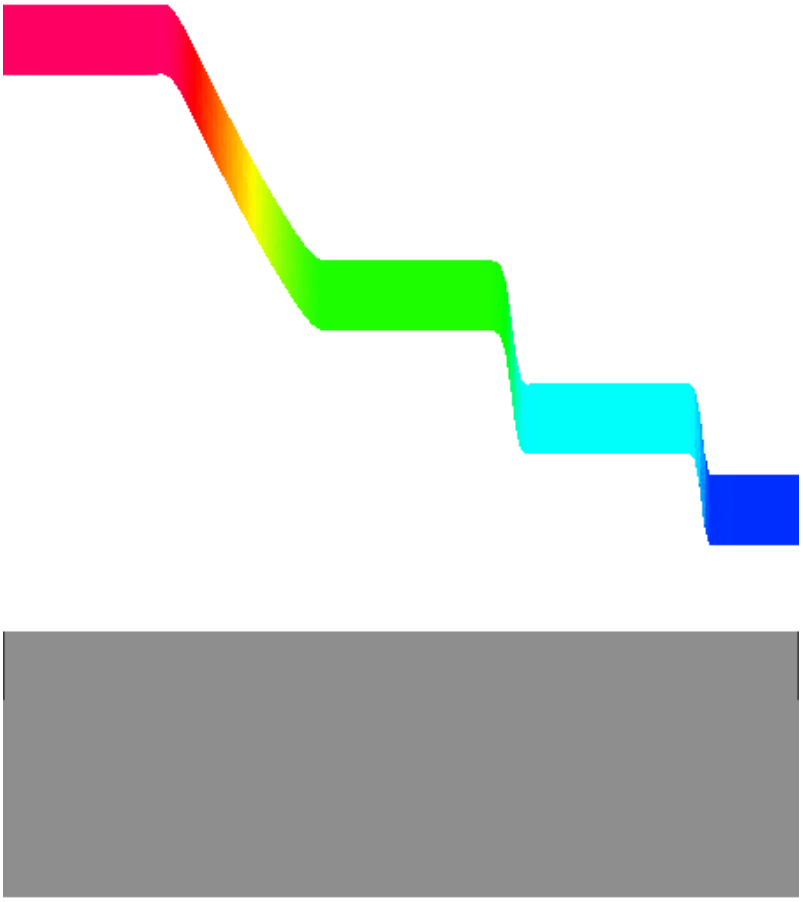

Abbildung 6.4: Dichte mit Discontinuous Galerkin Verfahren dritter Ordnung.

Abb. 6.4 zeigt einen 2D-Schnitt der numerischen Lösung der Dichte beim Shock Tube Problem in 3D. Die Rechnung wurde auf einem Hexaeder-Gitter mit 1.024.000 Elementen gemacht. Ausgewertet wurde die Lösung zum Endzeitpunkt t=0.75. Im Gegensatz zur Lösung mit dem Verfahren erster Ordnung (siehe Abb. 6.2) sind die Kontaktunstetigkeit und der Schock viel schärfer wiedergegeben. Da sich die minimalen und maximalen Werte zwischen 0.999609 bzw. 4.009027 befinden, sind leichte Oszillationen zu erkennen. Der Parameter M aus dem Abschnitt über Limitierung wurde in dieser Rechnung gleich 50 gewählt.

Kapitel 7

Beispiele und numerische Ergebnisse

In diesem Kapitel werden wir weitere Testprobleme rechnen. Wenn im folgenden von Anfangswerten die Rede ist, so schreiben wir sie z.B. als $\rho_0(x, y)$. Mit diesen Werten initialisieren wir die Werte zum Zeitpunkt $t = 0$ und es gilt $\rho(x, y, 0) := \rho_0(x, y)$.

7.1 Skalarer Fall: Rotating Cone

Dieses skalare Beispiel soll die Qualität der Discontinuous Galerkin Verfahren bei skalaren Gleichungen demonstrieren. Als Beispiel haben wir das Rotating Cone Problem gewählt.

$$u(x, y, t)_t - yu(x, y, t)_x + xu(x, y, t)_y = 0$$
$$u(x, y, 0) = u_0(x, y)$$

mit Anfangswerten

$$u_0(x, y) = \begin{cases} 1 - 20r(x, y) \text{ , falls } r(x, y) < 0.05 \\ 0 \text{ sonst} \end{cases}$$

mit $r(x, y) = (x - 0.5)^2 + y^2$.

Das Rotating Cone Problem wird folgendermaßen beschrieben: Ein Kegel mit einer maximalen Höhe 1 startet zur Zeit $t = 0$ rechts vom Ursprung und dreht sich in $t = 2\pi$ einmal gegen den Uhrzeigersinn um den Ursprung. Dieses Problem kann sowohl exakt als auch numerisch gelöst werden. Bei der numerischen Simulation verliert der Kegel an Höhe und wird immer mehr verschmiert werden. Die Höhe und der Grad der Verschmierung ist ein guter Indikator für die verwendeten Verfahren. Gerechnet haben wir dieses Problem auf einem Dreiecksgitter mit 51200 Zellen.

Bei einem Verfahren erster Ordnung startet der Kegel mit der Höhe 0.996875. Nach einer Viertel-Drehung beträgt die Höhe 0.883282, nach einer halben Drehung 0.773495. Nach $t = 2\pi$ (Endzeitpunkt, eine Drehung) ist die Höhe 0.601209. Siehe das linke Bild in Abb. 7.1.
Bei einem Verfahren höherer Ordnung startet der Kegel mit der Höhe 0.996875. Nach einer Viertel-Drehung beträgt die Höhe 0.996875, nach einer halben Drehung 0.987962. Nach $t = 2\pi$ (Endzeitpunkt, eine Drehung) ist die Höhe 0.987962. Siehe das rechte Bild in Abb. 7.1. Je dichter die Isolinien, desto steiler der Gradient. Hier wurde ein Discontinuous Galerkin Verfahren zweiter Ordnung mit der in Abschnitt 4.8.1 beschriebenen Projektionsmethode für skalare Gleichungen gerechnet.

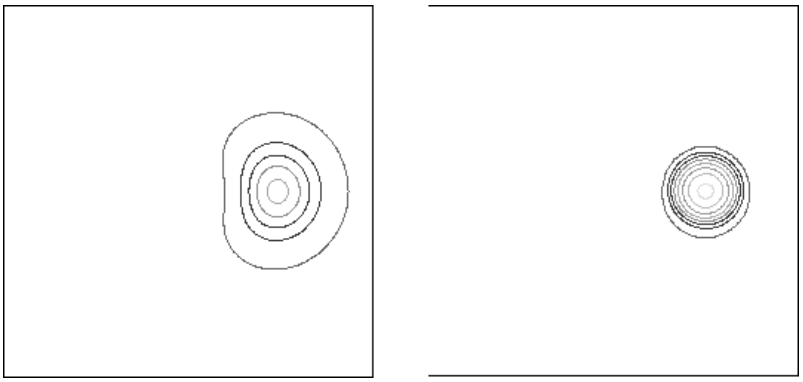

Abbildung 7.1: Vergleich zwischen Verfahren erster und höherer Ordnung beim Rotating Cone Problem.

Die Abbildung links zeigt den Verlauf des Kegels nach einer Drehung zum Endzeitpunkt. Durch den weiteren Abstand der Höhenlinien kann man erkennen, dass mit einem Verfahren erster Ordnung die Lösung sehr stark verschmiert wird. Die Abbildung rechts zeigt den Verlauf des Kegels nach einer Drehung mit dem Discontinuous Galerkin Verfahren zweiter Ordnung. Es werden fast die Anfangsdaten reproduziert. Die Isolinien sind sehr dicht.

7.2 Systeme: Forward Facing Step

2D

Das Mach 3 Wind Tunnel with a Step Problem, auch Problem der einspringenden Ecke oder Forward Facing Step genannt, ist folgendes: Das Gebiet ist

$$\Omega = [0.0; 3.0] \times [0.2; 1.0] \cup [0.0; 0.6] \times [0.0; 0.2] \,.$$

Als Anfangswerte gelten im ganzen Gebiet

$$\rho_0(x, y) := 1.4 \,,\, (u_1)_0(x, y) := 3.0 \,,\, (u_2)_0(x, y) := 0.0 \,,\, p_0(x, y) := 1.0 \,.$$

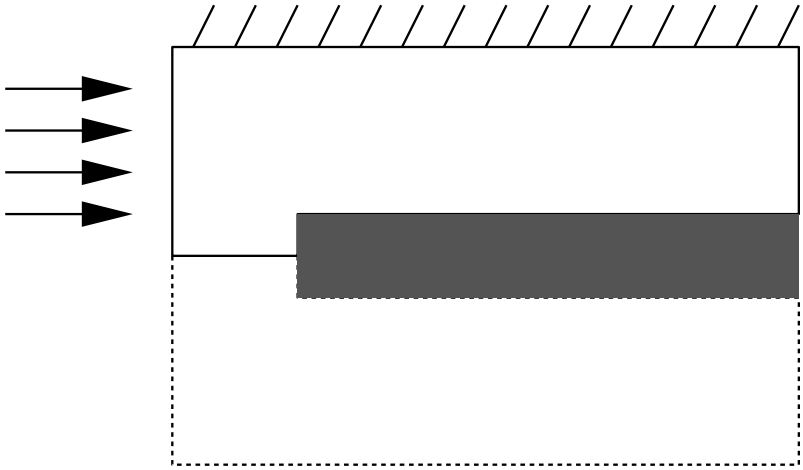

Abbildung 7.2: Gebiet beim Forward Facing Step Problem.

Abb. 7.2 zeigt das Gebiet des Forward Facing Step Problems. Die Strömung fliesst von links in das Gebiet hinein. Am oberen Rand hat man eine feste Wand. Aus Symmetriegründen rechnet man nur das halbe Problem und nimmt an der Spiegelungsachse Reflektionsrandbedingung an.

Daraus ergeben sich die anderen Anfangswerte, insbesondere die Machzahl $m = 3.0$. Diese Werte fließen über die linke Kante

$$\{(x, y) \in \mathbb{R}^2 \mid x = 0.0 \wedge y \in [0.0; 1.0]\}$$

zu jeder Zeit t weiterhin ein. Auf der rechten Seite

$$\{(x, y) \in \mathbb{R}^2 \mid x = 3.0 \wedge y \in [0.2; 1.0]\}$$

hat man Ausflussbedingungen. An der unteren Seite

$$\{(x, y) \in \mathbb{R}^2 \mid y = 0.0 \wedge x \in [0.0; 0.6]\}$$

hat man aus Symmetriegründen Reflektionsbedingung. Hier wird also in einer Ghostzelle die y-Komponente der Geschwindigkeit mit -1 multipliziert. Für die anderen drei Ränder

$$\begin{gathered}\{(x, y) \in \mathbb{R}^2 \mid x = 0.6 \wedge y \in [0.0; 0.2]\}, \\ \{(x, y) \in \mathbb{R}^2 \mid y = 0.2 \wedge x \in [0.6; 3.0]\}, \\ \{(x, y) \in \mathbb{R}^2 \mid y = 1.0 \wedge x \in [0.0; 3.0]\}\end{gathered}$$

hat man mehrere Möglichkeiten. Zum einen kann man bei allen drei Rändern Reflektionsbedingung annehmen oder man nimmt, wie wir, bei allen drei anderen Rändern die undurchlässige Wand an. Man hat also als Randbedingung $\vec{n} \cdot \vec{u} = n_1 u_1 + n_2 u_2 = 0$. Siehe auch Abbildung 7.2. Dieses Problem wird nach $t = 4.0$ ausgewertet. Hier haben wir mit der CFL-Zahl 0.4 gerechnet.

Das Problem wurde auf einem Parallelrechner unter MPI in mehreren Varianten gerechnet. Die Abbildung 7.3 zeigt ein Dreiecksgitter, auf dem mit ca. 1 Mio. Elementen die Rechnungen des Verfahrens erster Ordnung und die Variante höherer Ordnung mittels eines MUSCL-Verfahren gerechnet wurde. Abbildung 7.4 zeigt eine mögliche Partitionierung des Rechengitters. Dabei steht jede Farbe für einen Prozessor. Abbildung 7.5 zeigt den Lösungsverlauf der Dichte zum Endzeitpunkt mit dem Verfahren erster Ordnung. Abbildung 7.6 zeigt den Lösungsverlauf der Dichte zum Endzeitpunkt mit dem MUSCL-Verfahren. Abbildung 7.7 zeigt ein Rechtecksgitter, auf dessen Struktur die Discontinuous Galerkin Verfahren mit bis zu 250.000 Elementen gerechnet worden sind. Abbildung 7.8 zeigt den Lösungsverlauf der Dichte zum Endzeitpunkt mit dem Discontinuous Galerkin Verfahren zweiter Ordnung. Das Gitter besteht aus 258048 Elementen ($\Delta x = \Delta y = 1/320$). Abbildung 7.9 zeigt den Lösungsverlauf der Dichte zum Endzeitpunkt mit dem Discontinuous Galerkin Verfahren dritter Ordnung. Das Gitter besteht aus 64512 Elementen ($\Delta x = \Delta y = 1/160$).

Während man bei dem Verfahren erster Ordnung durch Erhöhen der Elementanzahl die Schockfronten sehr gut auflösen kann, weil die Elementdurchmesser kleiner als die Zeichengenauigkeit sind, so bleibt die Kontaktunstetigkeit hinter dem oberen Machstamm doch immer noch schlecht aufgelöst. Bei den Discontinuous Galerkin Verfahren zweiter Ordnung erkennt man ab der Diskretisierungsweite $\Delta x = \Delta y = 1/320$ in der Kontaktunstetigkeit eine Wirbelstruktur. Bei den Discontinuous Galerkin Verfahren dritter Ordnung erkennt man schon ab der Diskretisierungsweite $\Delta x = \Delta y = 1/160$ in der Kontaktunstetigkeit eine Wirbelstruktur. Dieses ist bei der höheren Ordnung Variante mit MUSCL nicht zu beobachten. Dies ist ein weiteres Zeichen für die Qualität der Discontinuous Galerkin Verfahren.

Betrachtet man jetzt ein Isolinienbild des Drucks (siehe Abb. 7.10), kann man folgende Schlussfolgerungen ziehen. Hinter dem oberen Machstamm bildet sich eine Kontaktunstetigkeit. Die Wirbel sind beim Druck nicht zu sehen, da der Druck in der Kontaktunstetigkeit konstant ist. Da die Geschwindigkeit in der Kontaktunstetigkeit ebenfalls konstant ist, ist die Wirbelstruktur in den Dichtebildern mit den Discontinuous Galerkin Verfahren nicht auf unterschiedlche Geschwindigkeiten zurückzuführen. Vielmehr darauf, dass schwere Teilchen, die nach unten ”drängen”, mit leichteren Teilchen von links nach rechts transportiert werden.

Die querverlaufenden Isolinien am unteren Rand in den Dichtebildern 7.8,7.9 lassen nicht auf eine Grenzschicht schliessen, was wiederum ein Zeichen von Viskosität im Verfahren sein würde, sondern es handelt sich wiederum um eine Kontaktunstetigkeit, die sich hinter dem unteren Machstamm bildet, was durch das Dichtebild deutlich wird.

3D

Im Gegensatz zu vielen Arbeiten, in denen das 2D-Gebiet in der dritten Komponente geshiftet wird, rechnen wir hier eine wirkliche 3D Anwendung. Deswegen kann man die Lösungen auch nur bedingt mit den 2D Lösungen vergleichen.

Das Mach 3 Wind Tunnel with a Step Problem, auch Problem der einspringenden Ecke

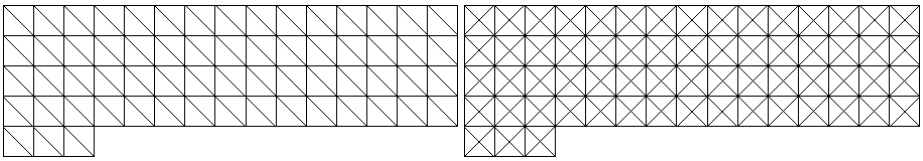

Abbildung 7.3: Dreiecksgitter der einspringenden Ecke.

Abb. 7.3 zeigt links das Makro-Gitter mit 126 Elementen. Rechts ist das Gitter mit 252 Elementen nach einer globalen Verfeinerung zu sehen. Bei einer Verfeinerung wird jeweils in der Mitte der längsten Kante eines Dreiecks ein neuer Knoten eingefügt. Gerechnet wurde auf 258048 Elementen (11 globale Verfeinerungen) beim MUSCL Verfahren und 1.032.192 Elementen (13 globale Verfeinerungen) beim Verfahren erster Ordnung.

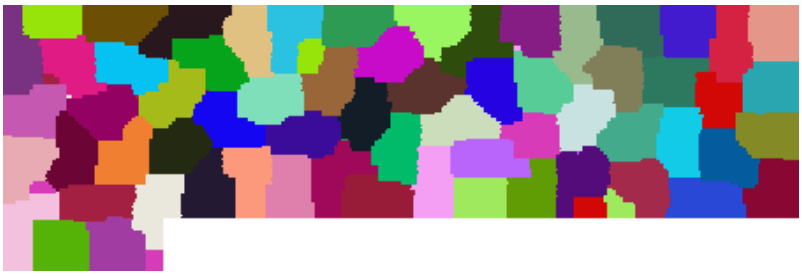

Abbildung 7.4: Partitionierung des Gebietes.

Abb. 7.4 zeigt die Partitionierung eines Rechteckgitters mit 16128 Elementen in 64 Teilgebiete. Gerechnet wurde aber auf bis zu 128 Prozessoren.

oder Forward Facing Step genannt, ist folgendes: Das Gebiet ist

$$\Omega = ([0.0; 3.0] \times [0.0; 1.0] \times [0.0; 1.0]) \setminus ([0.6; 3.0] \times [0.0; 0.4] \times [0.6; 1.0]) .$$

Als Anfangswerte gelten im ganzen Gebiet

$$\rho_0(x, y, z) := 1.4 \, , \, (u_1)_0(x, y, z) := 3.0 \, , \, (u_2)_0(x, y, z) := 0.0 \, ,$$
$$(u_3)_0(x, y, z) := 0.0 \, , \, p_0(x, y, z) := 1.0 \, .$$

Daraus ergeben sich die anderen Anfangswerte, insbesondere die Machzahl $m = 3.0$. Diese Werte fließen über die linke Kante

$$\{(x, y, z) \in \mathbb{R}^3 \mid x = 0.0 \wedge y \in [0.0; 1.0] \wedge z \in [0.0; 1.0]\}$$

zu jeder Zeit t weiterhin ein. Auf der rechten Seite

$$\{(x, y, z) \in \mathbb{R}^3 \mid x = 3.0 \wedge y \in [0.0; 1.0] \wedge z \in [0.0; 1.0]\}$$

hat man Ausflussbedingungen. An den übrigen Rändern kann man z.B. Reflektionsrandbedingungen oder undurchlässige Wände ($n_1u_1 + n_2u_2 + n_3u_3 = 0$) annehmen.
Die hier gezeigten Abbildungen zeigen die Ergebnisse der Rechnungen auf Hexaedern. Dabei wurden die Verfahren auf einem ”Shared Memory” Rechner parallelisiert. Die Lösungen zeigen die Dichte auf Schnittebenen.

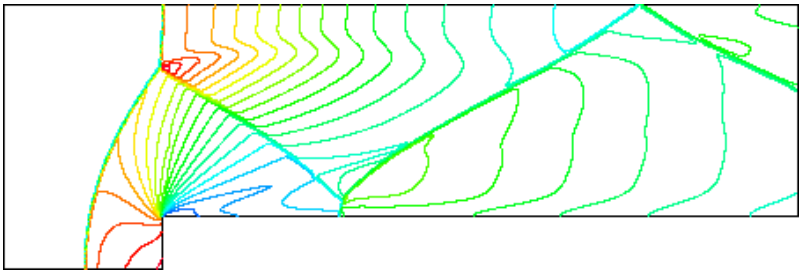

Abbildung 7.5: Dichte mit Verfahren erster Ordnung.

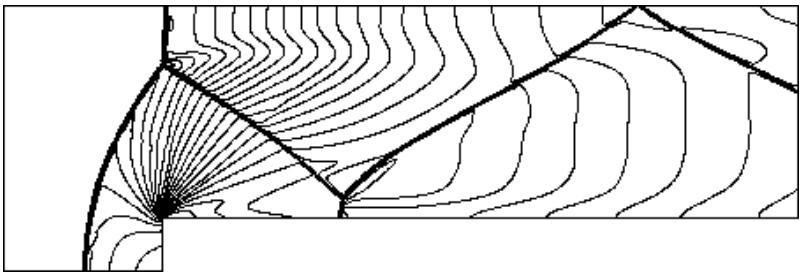

Abbildung 7.6: Dichte mit Verfahren höherer Ordnung (MUSCL).

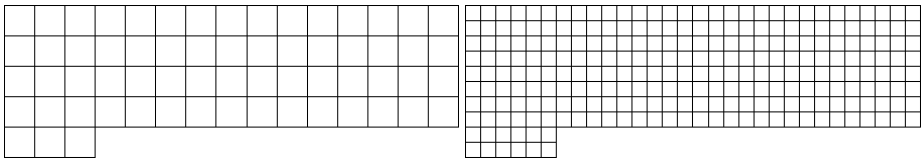

Abbildung 7.7: Rechtecksgitter.

Abb. 7.7 zeigt links das Makro-Gitter mit 63 Elementen. Dabei ist $\Delta x = \Delta y = 1/5$. Rechts ist das Gitter mit 252 Elementen nach einer globalen Verfeinerung zu sehen. Bei einer Verfeinerung wird jeweils in der Mitte einer Kante eines Quadrats ein neuer Knoten eingefügt. Gerechnet wurde auf 129024 Elementen (5 globale Verfeinerungen) bei Discontinuous Galerkin dritter Ordnung und 258048 Elementen (6 globale Verfeinerungen) bei Discontinuous Galerkin zweiter Ordnung.

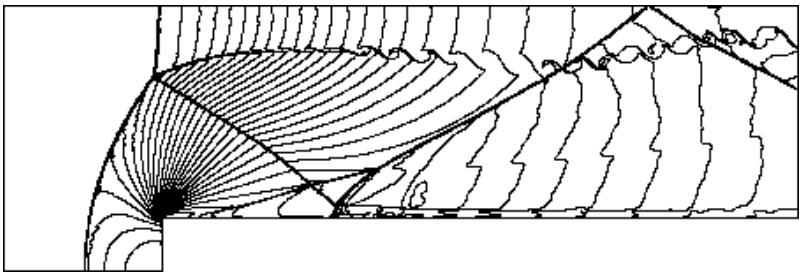

Abbildung 7.8: Dichte mit Discontinuous Galerkin Verfahren zweiter Ordnung.

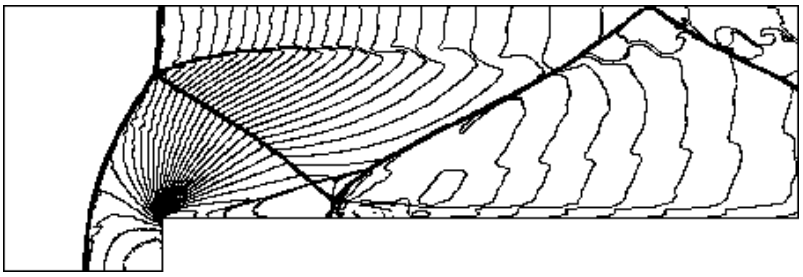

Abbildung 7.9: Dichte mit Discontinuous Galerkin Verfahren dritter Ordnung.

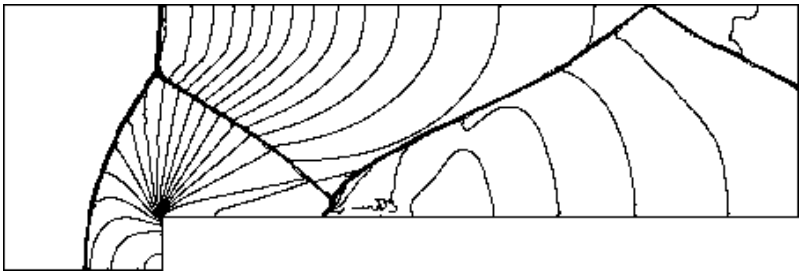

Abbildung 7.10: Druck mit Discontinuous Galerkin Verfahren dritter Ordnung.

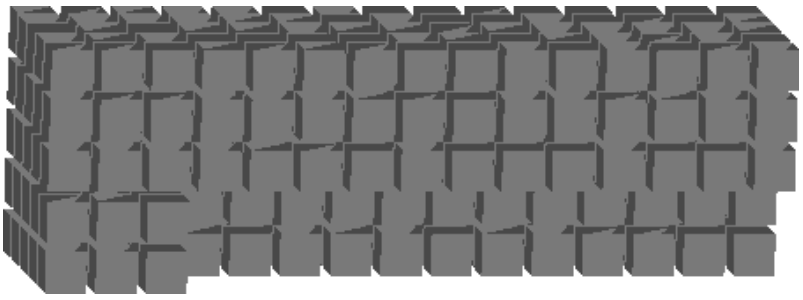

Abbildung 7.11: Hexaedergitter.

Abb. 7.11 zeigt das Makro-Gitter mit 327 Elementen. Dabei ist $\Delta x = \Delta y = \Delta z = 1/5$. Bei einer globalen Verfeinerung wird ein Hexaeder in 8 Hexaeder geteilt. Die Rechnungen wurden mit 1.339.392 Elementen (4 globale Verfeinerungen) gemacht.

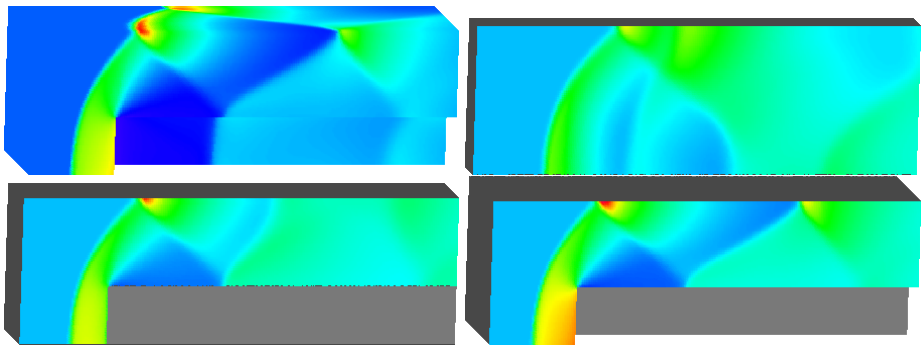

Abbildung 7.12: Dichte mit Verfahren erster Ordnung.

Abb. 7.12 zeigt die Dichte zum Zeitpunkt $t = 0$ auf vier Schnittebenen. Die Schnittebenen sind durch die Gleichung $n_x + n_y + n_z + d = 0$ gegeben. Hier wurde $n_x = n_y = 0, n_z = 1$ und $d = 0.0, 0.3, 0.6, 0.99$ gewählt.

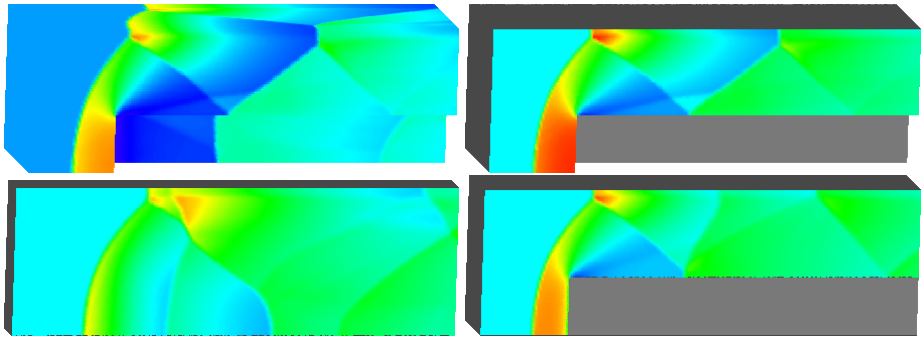

Abbildung 7.13: Dichte mit Discontinuous Galerkin Verfahren zweiter Ordnung.

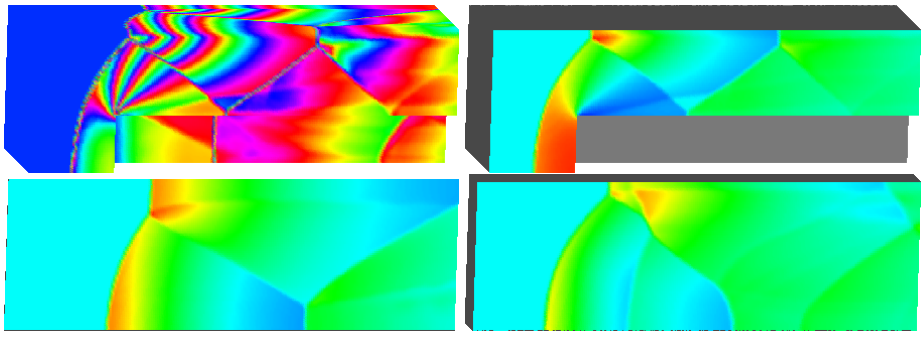

Abbildung 7.14: Dichte mit Discontinuous Galerkin Verfahren dritter Ordnung.

Kapitel 8

Visualisierung zeitabhängiger Vektorfelder mit Textur Transport

In diesem Kapitel wird eine Anwendung der Discontinuous Galerkin Verfahren gegeben. Es wird eine neue Visualisierungsmethode für zeitabhängige Vektorfelder vorgestellt. [1]

Am Anfang dieses Kapitels werden schon existierende Methoden zur Vektorfeldvisualisierung vorgestellt. Diese Verfahren weisen jedoch Schwächen auf, d.h sie sind zum Teil nur auf stationäre und nicht auf instationäre Vektorfelder anwendbar. Dann wird unsere neue Methode ausführlich vorgestellt. Bei dieser Methode wird ein vorgegebenes Muster in einem Geschwindigkeitsfeld transportiert.

Das daraus resultierende Transportproblem wird mit dem Discontinuous Galerkin Verfahren zweiter Ordnung in der skalaren Version berechnet.

8.1 Einleitung

Vektorfeldvisualisierung, besonders die Darstellung der Geschwindigkeitsfelder von CFD-Rechnungen, ist eines der fundamentalen Ziele wissenschaftlicher Visualisierung.

Die einfachste Methode, Vektorfelder darzustellen, ist das Zeichnen von Vektorpfeilen an Knoten eines über das Rechengitter gelegten Visualisierungsgitters. Diese Methode gibt einen Überblick zu einem festem Zeitpunkt, hat also keine Information

[1] Über dieses Thema ist auch ein gemeinsamer Artikel ”Visualization of time-dependent velocity fields by texture transport” [5] mit Martin Rumpf veröffentlicht worden.

über zeitliche Abläufe und hat ein Skalierungsproblem. D.h. Regionen mit sehr kleinen Geschwindigkeiten (z.B. in der Nähe eines Wirbels) oder sehr große Geschwindigkeiten in anderen Gebieten können nicht gleichzeitig aufgelöst werden. Skaliert man nun die Vektorlänge sehr groß, um in Wirbelnähe die Information zu verbessern, so werden die übrigen Vektoren so groß, dass der Informationsgrad der entstehenden Bilder sehr gering wird.

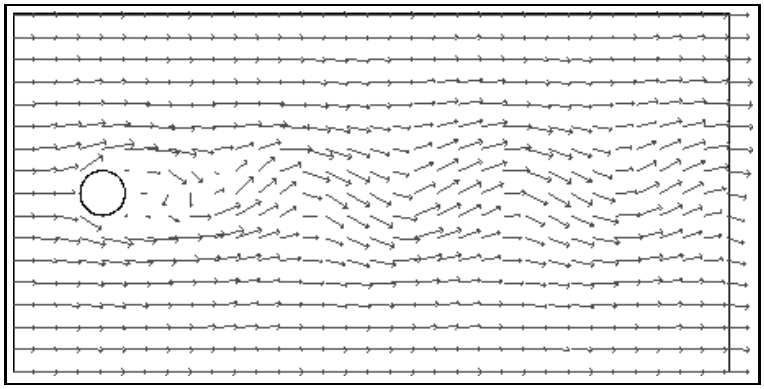

Abbildung 8.1: Vektorpfeile zu einem festem Zeitpunkt

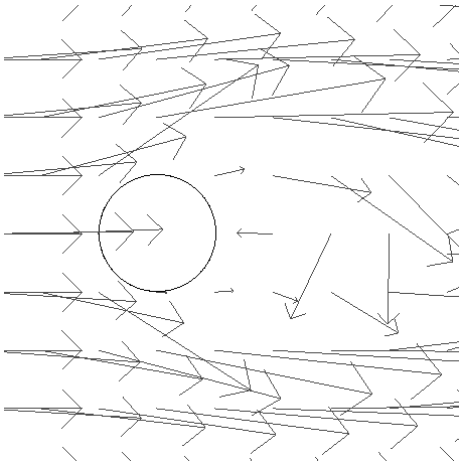

Abbildung 8.2: Zoom in die Wirbelzone hinter dem Hindernis.

Die Abbildungen 8.1 und 8.2 zeigen Vektorpfeile zu einem festen Zeitpunkt in der Karmanschen Wirbelstraße.
Das Ziel dieser Arbeit ist es, eine intuitivere und bessere Methode der Darstellung zu entwickeln. Sei ein Vektorfeld $v : \Omega \times \mathbb{R}_0^+ \to \mathbb{R}^n$ für ein Gebiet $\Omega \subset \mathbb{R}^n$ gegeben. Der zugehörige Fluss $\phi : \Omega \times \mathbb{R}_0^+ \to \Omega$ wird beschrieben durch das System gewöhnlicher Differentialgleichungen

$$\partial_t \phi(x,t) = v(\phi(x,t),t)$$

mit Anfangswerten $\phi(x,0) = x_0$.

Zum besseren Verständnis wollen wir zunächst einige Definitionen treffen.

Sei $0 \leq t_a \leq t_b \leq t_e$. In den folgenden drei Definitionen sei ein Vektorfeld $v(\mathbf{x}, t)$ für $\mathbf{x} \in \mathbb{R}^2$ und $t \in [0, t_e]$ gegeben. Wir betrachten die in den folgenden Definitionen definierten Mengen im Intervall $[t_a, t_b]$.

DEFINITION 8.1.1 ***(Partikelbahn)*** *(Bahnlinie,Partikelverfolgung)*
Gegeben sei ein zeitabhängiges Vektorfeld. Es werden die Positionen eines Partikels zu verschiedenen, aufeinanderfolgenden Zeitpunkten dargestellt.

$$M_1 := \{\mathbf{x}(t) \mid \mathbf{x}'(t) = v(\mathbf{x}(t), t) \text{ mit } \mathbf{x}(t_a) = \mathbf{x_0} \text{ für } t \in [t_a, t_b] \text{ und } \mathbf{x_0} \in \mathbb{R}^2\} .$$

DEFINITION 8.1.2 ***(Stromlinie)*** *(Streamline)*
Gegeben sei ein stationäres Vektorfeld. Bei einem zeitabhängigen Vektorfeld betrachten wir das Vektorfeld zu einem festen Zeitpunkt. Es werden die Positionen eines Partikels zu verschiedenen, aufeinanderfolgenden Zeitpunkten dargestellt. D.h. bei stationären Vektorfeldern ist Partikelbahn und Stromlinie identisch.
Sei $\bar{v}(\mathbf{x}(\bar{t})) := v(\mathbf{x}(\bar{t}), \bar{t})$ für $\bar{t} \in [0, t_e]$ fest.
Betrachte die Menge

$$M_2 := \{\mathbf{x}(s) \mid \mathbf{x}'(s) = \bar{v}(\mathbf{x}(s)) \text{ mit } \mathbf{x}(t_a) = \mathbf{x_0} \text{ für } s \in [t_a, t_b] \text{ und } \mathbf{x_0} \in \mathbb{R}^2\} .$$

DEFINITION 8.1.3 ***(Streichlinie)*** *(Streakline)*
Es werden in regelmäßigen kurzen Zeitabständen (oder auch kontinuierlich) an einer bestimmten Stelle der Strömung Partikel "injiziert" und die zu einem Zeitpunkt gehörenden Positionen der verschiedenen Partikel dargestellt. [2]
Betrachte die Menge

$$M_3 := \{\mathbf{x}_\tau(t_b) \mid \forall \tau \in [t_a, t_b] \text{ und } t \in [\tau, t_b] \text{ löse } \mathbf{x}'_\tau(t) = v(\mathbf{x}_\tau(t), t) \text{ mit } \mathbf{x}_\tau(\tau) = \mathbf{x_0} \text{ für } \mathbf{x_0} \in \mathbb{R}^2\} .$$

Spot Noise

Eine erste Methode zum Visualisieren eines (stationären) Vektorfeldes mit Hilfe von Texturen war die Spot Noise Methode von van Wijk [66].
Zunächst wollen wir den Begriff Textur definieren:

DEFINITION 8.1.4 ***(Textur)***
Eine Textur ist stets eine Funktion

$$T : N \to L .$$

Dabei ist L die Menge der möglichen Grau- oder Farbwerte der Textur. Häufig ist der Definitionsbereich der Textur diskretisiert:

[2] Die Bezeichnung "Streichlinie" rührt daher, dass man Partikel beobachten möchte, die mit einem *Pinselstrich* auf der Flüssigkeit angebracht wurden.

$$N = [0, n_x] \times [0, n_y] \subset \mathbb{N}^2 .$$

*Die Position (i, j) heißt im Zweidimensionalen **Pixel**.*
In 3D gilt

$$N = [0, n_x] \times [0, n_y] \times [0, n_z] \subset \mathbb{N}^3 .$$

*Die Position (i, j, k) heißt im Dreidimensionalen **Voxel**.*

DEFINITION 8.1.5 ***(Spot)***
Ein Spot ist gegeben durch folgende Funktion:

$$\textit{Spot} : N \longrightarrow \{0, 1\} .$$

Dabei repräsentiert 1 die Farbe Weiß und 0 die Farbe Schwarz.

Abb. 8.3 zeigt in der ersten Zeile solche Spots.
Ein Spot Noise Bild entsteht dann dadurch, dass eine bestimmte Anzahl von Spots eines Durchmessers übereinander gelagert werden.

$$T(i,j) = \begin{cases} 0.0 & \text{, falls Pixel } (i,j) \text{ von keinem Spot getroffen wird } (k = 0) \\ \frac{k}{2^{k-1}} & \text{, falls Pixel } (i,j) \text{ von } k \text{ Spots getroffen wird } (k > 0) \end{cases} .$$

Ergebnisse dieser Methode sind in Abb. 8.3 zu sehen.
Diese Spots können natürlich durch ein beliebiges Muster dargestellt werden. Speziell zur Visualisierung von Vektorfeldern sind bei van Wijk [66] zwei Varianten zu finden. Die erste verwendet zur Definition des Spots eine Ellipse, deren längere Achse parallel zum Vektor v ausgerichtet ist (siehe Abb. 8.4). Bei der zweiten Variante werden die Spots entlang der durch das Vektorfeld definierten Stromlinien bewegt. Beide Varianten schließen sich natürlich nicht aus.

Line Integral Convolution

Cabral und Leedom [7] stellten für stationäre Vektorfelder die Linienintegralfaltungsmethode (Line Integral Convolution, LIC) vor. Dieses Verfahren wurde von Hege und Stalling [60] sehr stark verbessert (Fast Line Integral Convolution, FLIC). Diese zunächst einmal nur in 2D und auf Ebenenschnitten in 3D bei stationären Vektorfeldern funktionierende Methode wollen wir kurz vorstellen.
Wir betrachten ein stationäres Vektorfeld, das durch eine Abbildung $v : \mathbb{R}^2 \to \mathbb{R}^2$ gegeben ist. Eine Integralkurve oder auch Stromlinie ist ein Pfad $\sigma(u)$, dessen Tangentialvektoren in einem festen Punkt mit dem Vektorfeld in diesem Punkt übereinstimmen.

$$\frac{d}{du}\sigma(u) = v(\sigma(u)) . \tag{8.1}$$

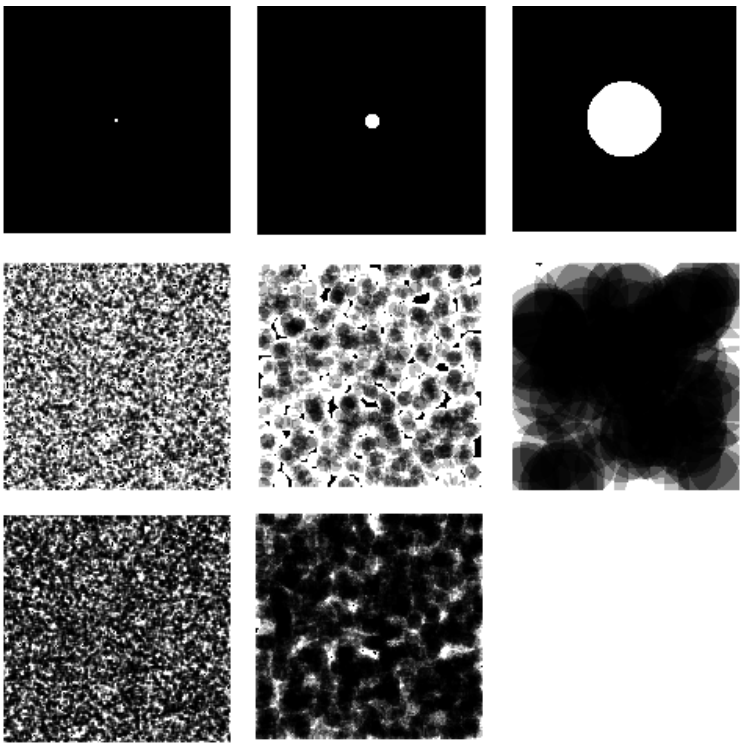

Abbildung 8.3: Spots und spot noise.

Abb. 8.3 zeigt in der ersten Zeile Spots mit einem Durchmesser von 5,20,100 Pixeln. Die Bilder in der zweiten und dritten Zeile sind mögliche Ergebnisse von "spot noise". In der ersten Spalte sieht man Überlagerungen von Spots mit Durchmesser 5 Pixel (1,15000,22000 Spots). In der zweiten Spalte sieht man Überlagerungen von Spots mit Durchmesser 20 Pixel (1,1000,2500 Spots). In der dritten Spalte sieht man Überlagerungen von Spots mit Durchmesser 100 Pixel (1,120 Spots).

Wie jeder Pfad kann $\sigma(u)$ parametrisiert werden durch eine stetige, monoton wachsende Funktion, ohne dabei den Verlauf und die Orientierung zu wechseln. Hier nehmen wir die Parametrisierung nach der Bogenlänge s.
Mit $\frac{ds}{du} = \mid v(\sigma(u)) \mid$ haben wir

$$\frac{d}{ds}\sigma(s) = \frac{d\sigma}{du}\frac{du}{ds} = \frac{v}{\mid v \mid} =: f(\sigma(s))\,. \tag{8.2}$$

Diese Reparametrisierung ist natürlich nur dort gültig, wo $\mid v \mid$ nicht verschwindet. Um eine Stromlinie durch einen Punkt x_0 zu finden, müssen wir die gewöhnliche Differentialgleichung (8.2) mit Anfangswerten $\sigma(0) = x_0$ lösen. Es kann gezeigt werden, dass eine eindeutige Lösung existiert, wenn die rechte Seite f einer Lipschitz-Bedingung genügt. Dies folgt aus dem Existenz- und Eindeutigkeitssatz von Picard-Lindelöf. Dazu muss $v \neq 0$ sein und v muss beschränkt sein.
Gegeben sei nun eine Input-Textur $T : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$. Wir nennen die Input-Textur

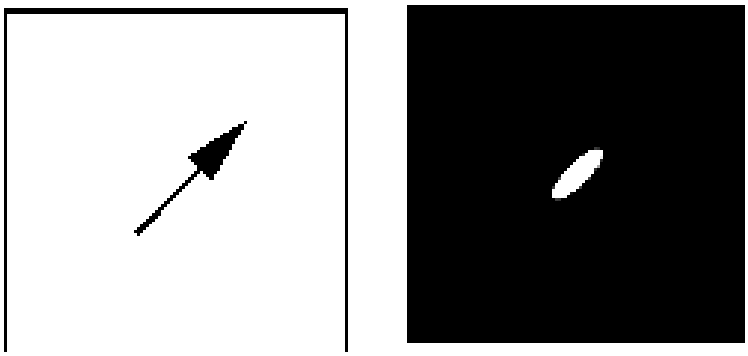

Abbildung 8.4: Vektorpfeil und Ellipse als Spot.

Abb. 8.4 zeigt eine Ellipse als Spot. Die längere Achse der Ellipse ist parallel zum Vektor v ausgerichtet.

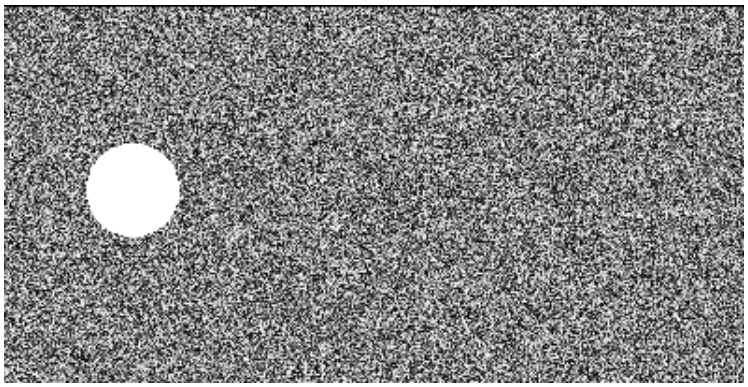

Abbildung 8.5: Weißes Rauschen über das Gebiet gelegt.

weißes Rauschen, wenn sie eine Normalverteilung von Grauwerten in $[0, 1]$ ist (siehe Abbildung 8.5). Für jeden Punkt (Pixel eines Bildes) hat man einen Intensitätswert.
Um das gegebene Vektorfeld darzustellen, geht man folgendermaßen vor. Man bestimmt die Intensität an einem Punkt $x_0 = \sigma(s_0)$ durch

$$I(x_0) = \int_{s_0-L}^{s_0+L} k(s - s_0)T(\sigma(s))ds\,. \tag{8.3}$$

Dabei ist k ein Faltungskern, der gewöhnlich vom Gauss-Typ oder Block-Typ ist und hier Filter-Kern genannt wird. L ist die Länge der Stromlinie und wird Filterlänge genannt.
Wenn wir $k(s) \equiv 1$ setzen, erhalten wir

$$I(x_0) = \int_{s_0-L}^{s_0+L} T(\sigma(s))ds = \frac{1}{2n+1}\sum_{i=-n}^{n} T(x_i)$$

mit $x_i = x(s_0 + ih_t)$. Dabei ist h_t der Abstand zwischen zwei Pixeln.

Wir betrachten zwei Punkte, $x_1 = \sigma(s_1)$ und $x_2 = \sigma(s_2)$, auf der gleichen Stromlinie und nehmen an, dass diese Punkte sehr nah beieinander liegen. Sei $\Delta s = s_2 - s_1$. Angenommen, wir hätten die Intensität $I(x_1)$ am Punkt x_1 schon berechnet, dann können wir die Intensität $I(x_2)$ am Punkt x_2 sehr effektiv daraus berechnen. Bei der Integration wird nämlich über fast genau denselben Teil integriert. Auf der einen Seite der Stromlinie fällt ein Stück weg und auf der anderen Seite der Stromlinie kommt ein Stück hinzu.
Gleichung (8.4) zeigt dies für den normierten Filter-Kern.

$$I(x_2) = I(x_1) - \int_{s_1-L}^{s_1-L+\Delta s} T(\sigma(s))ds + \int_{s_1+L}^{s_1+L+\Delta s} T(\sigma(s))ds\,. \tag{8.4}$$

Hierdurch haben die Punkte auf einer Stromlinie eine sehr starke Korrelation zueinander. Weicht man jedoch nur ein wenig von dieser einen Partikelbahn ab, so haben die Punkte keine Korrelation mehr zueinander, weil über einen völlig anderen Bereich integriert wird. Abbildung 8.6 verdeutlicht dies.

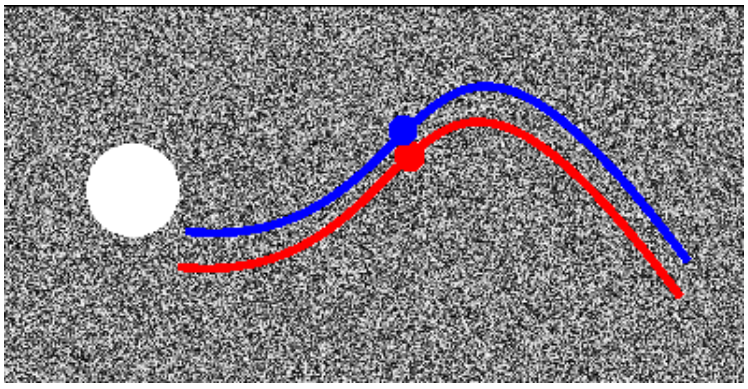

Abbildung 8.6: Zwei Partikelbahnen eines stationären Vektorfeldes.

Abbildung 8.7 zeigt das Ergebnis der LIC-Visualisierungsmethode zu einem festen Zeitpunkt der Karmanschen Wirbelstraße.
Diese Verfahren arbeiten zunächst einmal nur für stationäre Vektorfelder und auch nur in 2D bzw. in 3D auf Ebenenschnitten.
Eine Erweiterung auf beliebige Oberflächen in 3D ist in [3] zu finden.

Particle Tracing

Will man ein zeitabhängiges Geschwindigkeitsfeld in 2D oder 3D graphisch darstellen, so ist das Partikel Tracing ein sehr oft eingesetztes Werkzeug. Leider erhält man immer nur Informationen über wenige Partikel. Man erhält also nur lokale Aussagen. Abbildung 8.8 zeigt dies.

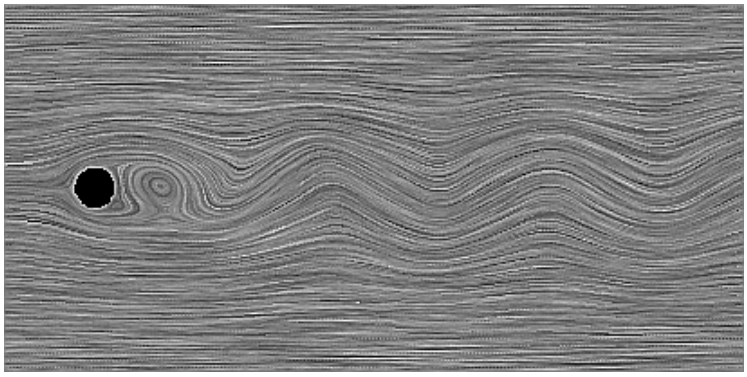

Abbildung 8.7: Ergebnis mit der LIC-Visualisierungsmethode.
Abb. 8.7 wurde erzeugt nach dem Algorithmus von Hege und Stalling [60].

Unsere neue Methode, die Textur Transport Methode, weist diese Einschränkungen nicht auf. Sie ist ein globales Partikelverfolgungsverfahren für zeitabhängige Vektorfelder. Im folgenden werden wir diese Methode beschreiben.

8.2 Lagrange Koordinaten und Transport Gleichung

Geschwindigkeitsfelder in numerischen Simulationen sind meistens gegeben in einem räumlich fixierten Euler Koordinaten System, wobei die physikalische Bedeutung der bewegten Teilchen in einem Lagrange Sinne gesehen werden kann. Diese Beobachtung ist der Ausgangspunkt vieler verschiedener Visualisierungstechniken. Die Methode, die wir vorstellen, stellt Lagrange Koordinaten dar, wobei eine Texturabbildung, die ein vorgegebenes Muster vom Lagrange Koordinatensystem auf ein Euler Koordinatensystem abbildet, benutzt wird. Sei nun $\Omega \subset I\!R^2$ ein Gebiet, das das Strömungsgebiet beschreibt, mit einem Einflussrand $\Gamma^+ \subset \partial\Omega$ und einem Ausflussrand $\Gamma^- \subset \partial\Omega$. Weiter nehmen wir an, dass die Geschwindigkeiten $v : \Omega \times [0, \hat{T}] \to I\!R^2$ bis zu einer fixierten Zeit $\hat{T}$ gegeben sind. In den Anwendungen wird diese Geschwindigkeit durch eine numerische Simulation (Euler- oder Navier-Stokes-Gleichungen) gegeben. Diese Berechnung des Geschwindigkeitsfeldes kann simultan zum Texturtransport geschehen oder die Werte werden in Dateien geschrieben, welche später vom Texturtransport eingelesen werden. Der Texturtransport wird auf dem gleichen Gitter wie die numerische Simulation berechnet. Interpretieren wir nun die Koordinaten X auf dem Einflussrand Γ^+, respektive die Einflusszeit T als abhängige Größen, die mit dem Geschwindigkeitsfeld transportiert werden. Dann ist die Bewegung durch die folgende Transport Gleichung für eine Dichte ρ (diese Dichte ist unabhängig von der Dichte der Euler- oder Navier-Stokes-Gleichungen) gegeben:

$$\begin{aligned} \partial_t \rho + v \cdot \nabla \rho &= 0 \quad \text{in } \Omega\,, \\ \rho &= \rho_\Gamma \quad \text{auf } \Gamma^+\,. \end{aligned} \tag{8.5}$$

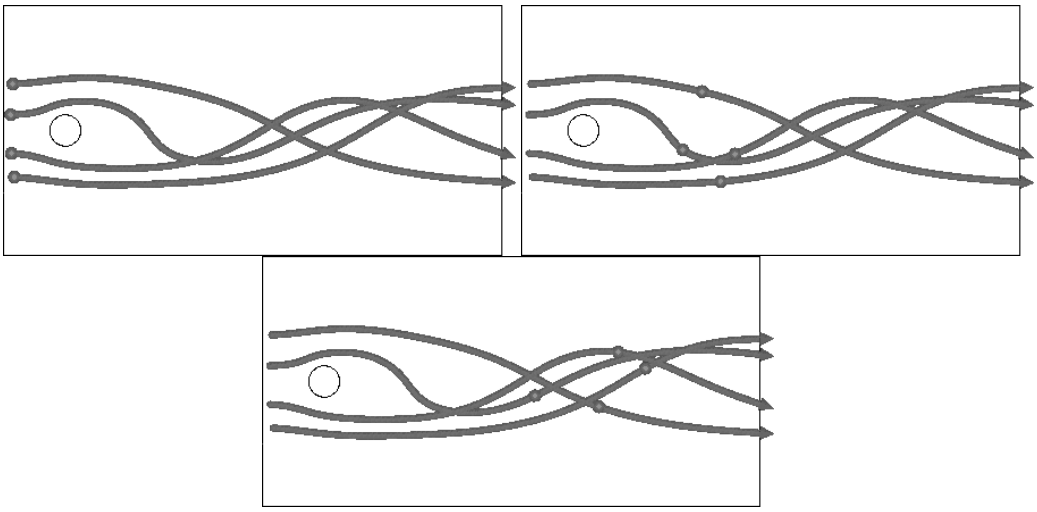

Abbildung 8.8: Partikelbahnen im zeitabhängigen Vektorfeld.

Abb. 8.8 zeigt vier Partikelbahnen im zeitabhängigen Vektorfeld der Karmanschen Wirbelstraße. Die Lage der vier Punkte auf der Partikelbahn gibt die Position des jeweiligen Partikels zu dem betrachteten Zeitpunkt an.

Dabei ist $\rho = X$ für $\rho_\Gamma = X$ auf Γ^+, respektive $\rho = T$ für $\rho_\Gamma = T$ auf Γ^+. Auf dem Ausflussrand Γ^- muss keine Randbedingung angegeben werden, falls $v \cdot \nu \geq 0$ für alle Zeiten. Hier ist ν die äussere Normale des Gebiets Ω. Dieser Transport kann interpretiert werden als ein simultanes und globales Partikelverfolgungsverfahren. Auf der Streichlinie $x(t)$ ist die Lösung ρ der obigen Transport Gleichung konstant, weil $\dot{x}(t) = v(x(t), t)$ und

$$\begin{aligned} \frac{d}{dt}\rho(x(t),t) &= \partial_t \rho(x(t),t) + \dot{x}(t) \cdot \nabla \rho(x(t),t) \\ &= 0\,. \end{aligned}$$

Deswegen sind Punkte mit konstantem X Wert auf der Partikelbahn, die an Position X auf Γ^+ startet, angeordnet. Analog indiziert ein konstanter T Wert die Fläche (in 3D, in 2D die Kurve), die das Bild der zugehörigen Fläche (in 3D, in 2D der Kurve) auf dem Einflussrand unter dem Fluss $\phi(\cdot, T)$ ist. In diesem Sinne können X, T als Funktionen auf $\Omega \times [0, \hat{T}]$ als Lagrange Koordinaten, die die Bewegung von Teilchen, die durch Γ^+ geflossen sind, beschreiben, betrachtet werden. Partikel, die früher ins Strömungsgebiet geflossen sind, werden nicht betrachtet.
Die Transport Gleichung ist ein gut-gestelltes Problem für die beschriebenen Anfangsbedingungen. Falls jede Partikelbahn, die an einer Position in Ω startet, das Gebiet verlassen hat, hängt die Lösung ρ nicht länger von diesen Anfangswerten ab. Für moderate Werte von $\hat{T}$ kann dies nicht der Fall sein und für einige Anwendungen ist besonders die Anfangsphase der physikalischen Simulation von großer Bedeutung.

Zusammengefasst müssen wir in 2D folgende zwei Gleichungen lösen: Sei $X : \mathbb{R}^2 \times \mathbb{R}^+ \rightarrow \mathbb{R}$, $T : \mathbb{R}^2 \times \mathbb{R}^+ \rightarrow \mathbb{R}$ und $x = (x_1, x_2)$. Für den Einflussrand gelte z.B. $\{x \mid x_1 = const\}$ (vergleiche Abbildung 8.9), dann lauten die beiden Gleichungen:

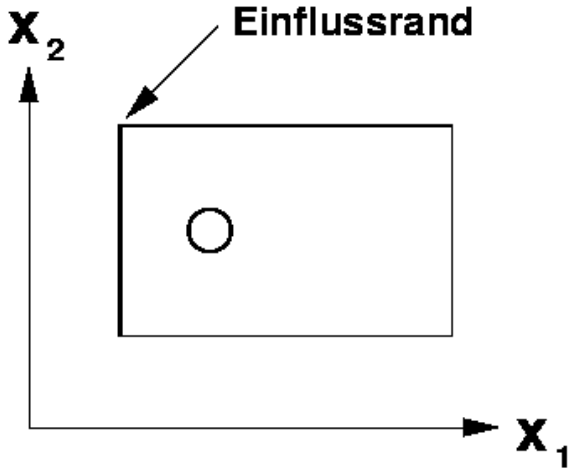

Abbildung 8.9: Lage des Einflussrandes bei der Karmanschen Wirbelstraße.

$$\begin{array}{rcll} \partial_t \mathbf{T} + v \cdot \nabla \mathbf{T} & = & 0 & \text{in } \Omega \times \mathbb{R}^+ , \\ \mathbf{T}(x,t) & = & t & \text{auf } \Gamma^+ \times \mathbb{R}^+ , \\ \mathbf{T}(x,0) & = & 0 & \text{in } \Omega \end{array} \tag{8.6}$$

und

$$\begin{array}{rcll} \partial_t \mathbf{X} + v \cdot \nabla \mathbf{X} & = & 0 & \text{in } \Omega \times \mathbb{R}^+ , \\ \mathbf{X}(x,t) & = & x_2 & \text{auf } \Gamma^+ \times \mathbb{R}^+ , \\ \mathbf{X}(x,0) & = & 0 & \text{in } \Omega . \end{array} \tag{8.7}$$

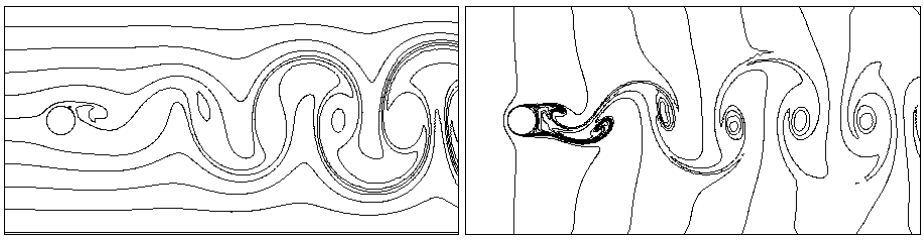

Abbildung 8.10: Isolinienbild der Komponenten X und T zu festem Zeitpunkt.

Abbildung 8.10 zeigt links ein Isolinienbild der X Komponente zu einem festen Zeitpunkt. Hier sind die Isolinien Streichlinien. Rechts ist ein Isolinienbild der T Komponente zu einem festen Zeitpunkt zu sehen.

Abbildung 8.10 zeigt Lösungen der beiden Gleichungen (8.6),(8.7) zu einem festen Zeitpunkt. Diese beiden Bilder wollen wir nun geeignet in einem Bild kombinieren.

Dazu müssen wir ein geeignetes Muster im Texturraum $\Gamma^+ \times [0, \hat{T}]$ definieren.

8.3 Textur Visualisierung

Nach numerischer Berechnung der beiden Transportprobleme haben wir für alle Punkte $x \in \Omega$ und alle Zeiten $t \in [0, \hat{T}]$ Werte X, T vorliegen. Diese Werte X, T wollen wir benutzen, um ein vorgegebenes Muster auf unser Rechengebiet Ω abzubilden.

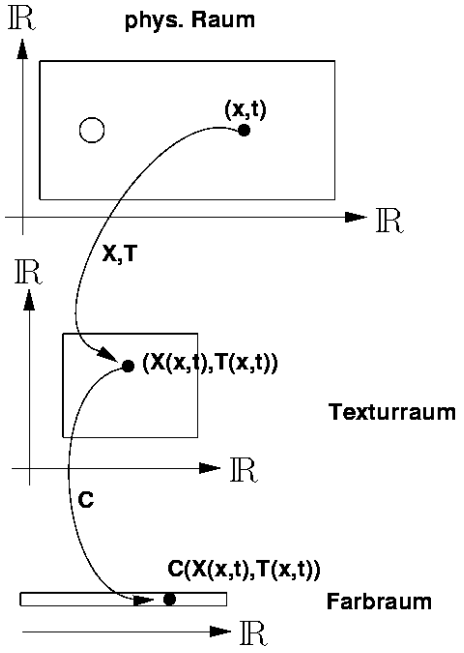

Abbildung 8.11: Eine Skizze der Abbildung vom Texturraum in den physikalischen Raum Ω.

Es gibt einige wünschenswerte Eigenschaften, die durch die strukturelle Darstellung der Lagrange Koordinaten verwirklicht werden sollte. Die Darstellung sollte simultan die Zeit und Einfluss-Koordinaten kodieren. Weiterhin sollte eine lange Zeitanimation der bewegten Teilchen möglich sein. Deshalb muss die Textur periodisch in T sein. Auch sollte man in detailiertere Regionen hochskalieren können. Also muss die Textur eine Skalierbarkeitseigenschaft haben. Diese Anforderungen werden durch folgende Konstruktion der Textur gesichert:

- Wähle ein weißes Rauschen auf einem gerasterten Gebiet $[0, 1]^2$. Dies ist gegeben durch eine Funktion $F : [0, 1] \times [0, 1] \to [0, 1]^3$, wobei der Vektor $F(x_1, x_2) = (y_1, y_2, y_3)^T$ den Farbwert eines Pixel bezeichnet. Der Vektor (y_1, y_2, y_3) bestimmt den RGB-Wert (R=rot,G=grün,B=blau) dieses Pixels. Da wir zunächst nur Grau-

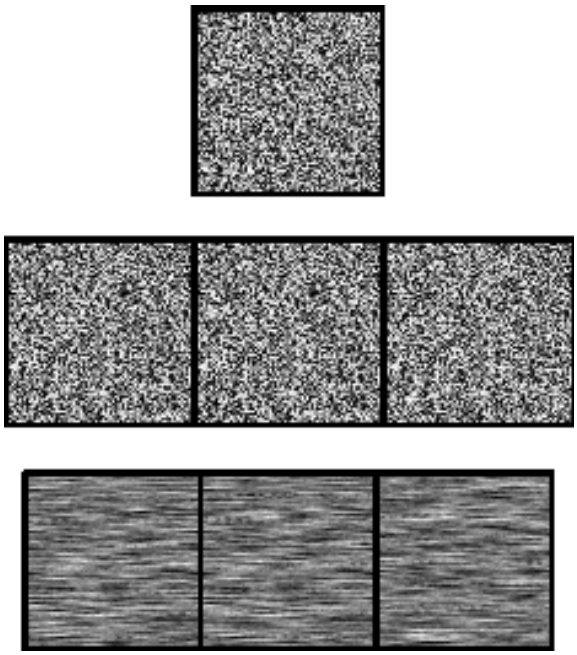

Abbildung 8.12: Erzeugung der Input-Textur (Teil 1).

werte zulassen, gilt $y_2 := y_1$ und $y_3 := y_1$, wobei $y_1 \in [0,1]$ durch einen Zufallszahlenalgorithmus bestimmt wird (vergleiche oberen Teil von Abb. 8.12).

- Dieses Gebiet wird dupliziert und dreimal in x_1-Richtung geshiftet (vergleiche mittleren Teil von Abb. 8.12).

- Benutze die LIC -ähnliche Faltung entlang der x_1 Komponente mit einem Vektorfeld $(v_1, v_2) = (\alpha, 0)$ mit $0 < \alpha < 1$. Als Ergebnis erhalten wir eine LIC-Textur dieses Geschwindigkeitsfeldes (vergleiche unteren Teil von Abb. 8.12). Dies ist gegeben durch eine Funktion $G : [0,1] \times [0,1] \rightarrow [0,1]^3$, wobei der Vektor $G(x_1, x_2) = (y_1, y_2, y_3)^T$ den Farbwert eines Pixel bezeichnet. Wieder gilt $y_1 = y_2 = y_3$ (vergleiche unteren Teil von Abb. 8.12).

- Nehme den mittleren Block aus dem Dreier-Block hinaus. Dieser ist nach Konstruktion periodisch (vergleiche oberen Teil von Abb. 8.13).

- Obwohl wir in dieser Textur schon eine Korrelation entlang der x_1 Komponente haben, wollen wir die Aussagekraft dieser Textur noch erhöhen. Durch Einfärben können wir eine zweite skalare Komponente , z.B. den Druck, darstellen. Alternativ können wir die Bewegung der Teilchen besser darstellen. Deswegen wird die Textur mit Grauton durch eine zeitperiodische Färbung erweitert (siehe mittleren Teil von Abb. 8.13). Die Färbung ist gegeben durch eine Multiplikation von zwei RGB-Werten. Der RGB-Wert (y_1, y_2, y_3) der LIC-Textur wird multipliziert mit dem RGB-

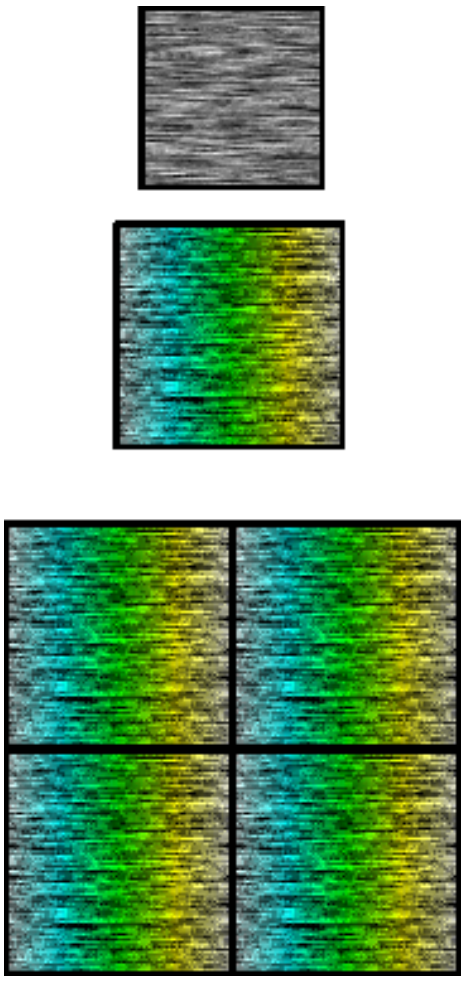

Abbildung 8.13: Erzeugung der Input-Textur (Teil 2).

Wert (z_1, z_2, z_3) der Farbe. Da $y_1 = y_2 = y_3$ gilt, ist der resultierende Farbwert $(y_1 z_1, y_1 z_2, y_1 z_3)$.

- Diese Textur ist nach Konstruktion periodisch in x_1- und x_2- Richtung. Um dies zu verdeutlichen wird dieser Block mehrmals kopiert (siehe unteren Teil von Abb. 8.13). Wir erhalten eine Textur über $\mathbb{R}^2$ mit einer fixierten Rasterung. Diese Funktion nennen wir $H : \mathbb{R} \times \mathbb{R} \to [0, 1]^3$.

- Abhängend von der Projektion von Welt- zu Bildschirm-Koordinaten skalieren wir die berechneten Lagrange Koordinaten X und T mit einem Faktor λ. Sei λ_0 eine Anfangsskalierung, die von der Größe des Gebiets $\Omega \times [0, \hat{T}]$ abhängt und $s = (\det P)^{\frac{1}{3}}$, wobei P die 3×3 Projektionsmatrix, die den linearen Teil der affinen Abbildung von Welt- zu Bild-Koordinaten beschreibt, dann ist $\lambda{:=}\lambda_0\, s$ eine akzeptable Wahl für den Skalierungsfaktor.

- Schliesslich erhalten wir als Texturkoordinaten $\lambda X, \lambda T$. D.h. nach Berechnung von X, T für alle Zeiten $t \in [0, \hat{T}]$ werten wir $H(\lambda T, \lambda X)$ aus.

Zusammengefasst ergibt sich folgendes Prinzip für die Erzeugung der Textur Transport-Bilder. Siehe dazu Abbildung 8.11. Die beiden Transport Gleichungen (8.6),(8.7) liefern für alle Zeiten $t \in [0, \hat{T}]$ für alle Punkte $(x_1, x_2) \in \Omega$ Werte $X(x_1, x_2, t)$ und $T(x_1, x_2, t)$. Dadurch ist eine Abbildung vom physikalischen Raum in den sogenannten Texturraum definiert. Durch eine weitere Abbildung C wird nun jedem Punkt des physikalischen Raumes ein Farbwert zugeordnet.
Durch diese Konstruktion hängt die resultierende Textur auf Ω zur Zeit $t \in [0, \hat{T}]$ stetig von t ab und die Skalierung von λ. Weiter ist das resultierende Muster unabhängig von dieser Skalierung.
Bei dieser Art der Konstruktion der Textur ist die T Komponente der Lagrange Koordinaten zweimal repräsentiert, durch die periodische Struktur der Textur und durch die Färbung.
Diese Art der Textur führte in den Anwendungen schon zu respektablen Ergebnissen, doch wollten wir noch weiter gehen. Die T Komponente der Textur wollten wir dazu benutzen, einen Zuverlässigkeitsparameter für die numerischen Rechnungen darzustellen.
Im folgenden soll die Konstruktion dieser Textur beschrieben werden. Obwohl wir für den Texturtransport ein Finite Volumen Verfahren höherer Ordnung benutzen, ist die Lösung mit einigen unvermeidbaren Fehlern versehen. Zunächst einmal ist schon die aus CFD Rechnungen resultierende Größe v mit einem Fehler versehen. Diese Größe verwenden wir in unseren numerischen Transportverfahren. Durch Viskosität im numerischen Verfahren erhalten wir einen zusätzlichen Fehler. Die Textur soll möglichst so beschaffen sein, dass sie einen scharfen Teil besitzt, den wir auf Bereiche abbilden, in denen wir uns der Güte der Lösung sicher sein können, und einen ausgewaschenen Teil, den wir auf Bereiche abbilden wollen, wo wir keine sichere Aussage zur Güte der Lösung machen können. Wir benötigen also einen Fehlerindikator, mit dem wir einen Fehler in Raum und Zeit messen können. Dieses Maß nennen wir $\eta(X(x,t), T(x,t))$ mit $x \in \Omega$, $t \in [0, \hat{T}]$ und betrachten es als Funktion im linearen Finite Elemente Raum. Wir benutzten nun die $\eta(X, T)$ Information für die Erzeugung des Vektorfeldbildes. In Bereichen, wo $\eta(X, T)$ klein ist, ist die numerische Lösung der Transport Gleichung zuverlässig und deswegen wird im scharfen Teil der Textur abgegriffen. Dort, wo $\eta(X, T)$ groß ist, ist die Bedeutung der Textur unklar und macht möglicherweise keinen Sinn.
Es soll also eine zweidimensionale Textur $\pi(x_1, x_2)$ erzeugt werden. Die eindimensionale Textur $\pi(0, \cdot)$ stellt einen scharfen Übergang von Grautönen dar. Die zweidimensionale Textur wird aus einer gegebenen eindimensionalen so erzeugt, das in x_1-Richtung von den unterschiedlichen Grautönen zu einem Einheitsgrauton für alle x_2 übergegangen wird. Wenn man die Annahme macht, dass die Farbe Weiß durch den numerischen Wert 1 und die Farbe Schwarz durch den numerischen Wert 0 beschrieben wird, so werden die verschiedensten Graustufen durch Werte $\in [0; 1]$ beschrieben. Der Einheitsgrauton besitze

den Wert 0.5. Für festes p sei der Wert $\pi(0,p)$ durch einen Zufallswert $r(p) \in [0;1]$ gegeben (siehe Abbildung 8.14).

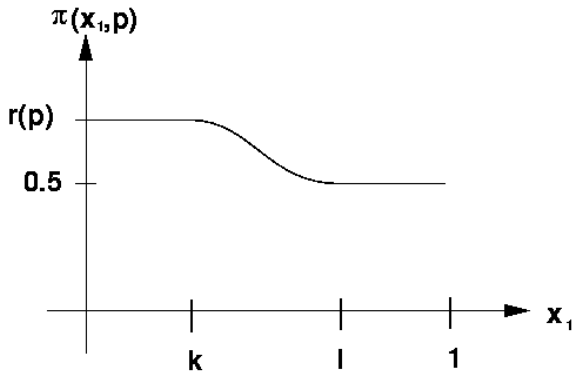

Abbildung 8.14: Eine mögliche Funktion $\pi(x_1,p)$.

Für Werte $x_1 < k$ sei $\pi(x_1,p) \equiv r(p)$ und $x_1 > l$ sei $\pi(x_1,p) \equiv 0.5$. Für $x_1 \in [k;l]$ wird durch einen kubischen Spline interpoliert. Dies sorgt schon für eine Auswaschung in x_1-Richtung. Eine ebenfalls wünschenswerte Auswaschung in x_2-Richtung wird dadurch realisiert, dass für größer werdendes x_1 über einen größeren x_2-Bereich gemittelt wird. Abbildung 8.15 zeigt das Ergebnis. Diese Textur ist wieder skalierbar.
Nehmen wir an, dass $\eta(X,T)$ eine Funktion mit Werten in $[0,1]$ ist. Kleine Werte indizieren kleine Fehler und Werte nahe der 1 indizieren große Fehler und somit keine Zuverlässigkeit der numerischen Ergebnisse. Dann nehmen wir $(\eta, \lambda X)$ als Texturkoordinaten, welche vom Texturraum auf das Numerikgitter abgebildet werden. Zusammengefasst ergibt sich bei dieser Art der Textur: Nach Berechnung von T, X für alle $x \in \Omega$ und alle Zeiten t wird für alle $x \in \Omega$ und alle Zeiten t der Fehlerindikator $\eta(X,T)$ berechnet und die erzeugte Textur $\pi(x_1,x_2)$ durch $\pi(\eta(X,T),\lambda X)$ ausgewertet (vergleiche oberen Teil von Abbildung 8.15).
Für die Einfärbung der Textur betrachten wir eine Funktion $C_2 : \mathbb{R} \to [0,1]^3$, d.h. jedem reellen Wert wird ein Farbwert zugeordnet. Hier wird nun C_2 bzgl. T ausgewertet, d.h. wir suchen $C_2(T)$.
Danach werden die RGB-Werte von $\pi(\eta(X,T),\lambda X)$ mit $C_2(T)$ kombiniert.
Eine mögliche Funktion η kann man bzgl. des Gradienten von X konstruieren. Zusätzlich wird die Komponente T noch überprüft. In den Regionen, in denen $T < \epsilon$ mit $\epsilon > 0$ ($\epsilon \approx 10^{-6}$) gilt, ist noch nichts ins Gebiet "hinein geflossen". Deshalb wird auch in diesem Teil die Textur in dem ausgewaschenen Bereich abgegriffen.
Sei $\mid \nabla X \mid \in [a_1,b_1]$ und sei $[a,b] \subset [a_1,b_1]$. Dann sei z.B.

$$\eta(X,T) = \begin{cases} 1.0 & \text{, falls } \mid \nabla X \mid \leq a \text{ oder } T < \epsilon \\ 1 - \frac{1}{b-a}(\mid \nabla X \mid -a) & \text{, falls } \mid \nabla X \mid \in [a,b] \\ 0.0 & \text{, falls } \mid \nabla X \mid \geq b \end{cases} .$$

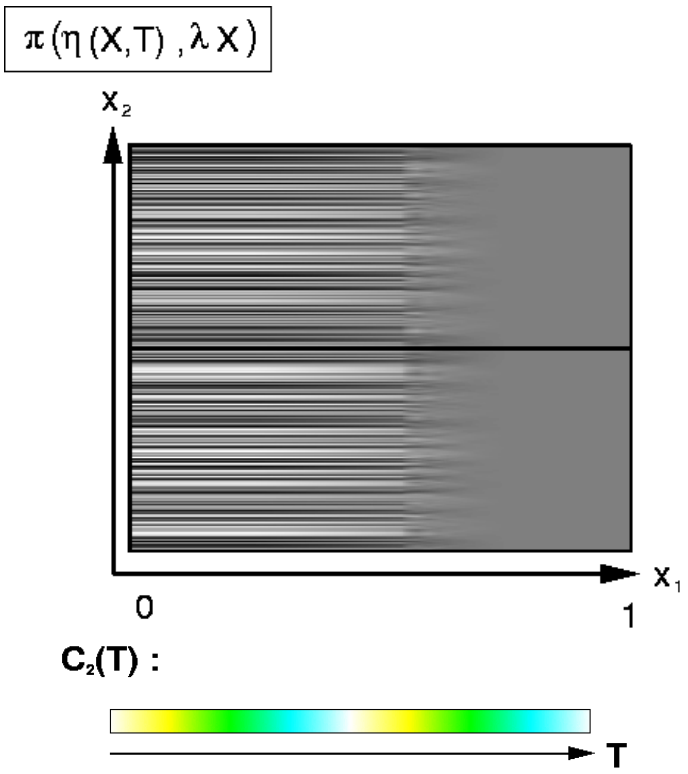

Abbildung 8.15: Fundamentale Zelle des Texturraums mit dem ausgewaschenen (blurring) Teil und eine Farbskala zum Kodieren der Zeit.

8.4 Numerisches Transport Schema

Numerische Verfahren für hyperbolische Erhaltungsgleichungen sind immer mit einer gewissen numerischen Viskosität versehen. Dies hat ein Verschmieren der numerischen Lösungsstruktur zur Folge. Diese Phänomene tauchen nicht nur bei Schocks, sondern auch bei linearen Transportgleichungen auf. Bei Verfahren höherer Ordnung hat man es eventuell mit Oszillationen zu tun. Gerade aber bei dem Texturtransportverfahren als Anwendungsbeispiel würde zuviel Viskosität die Entwicklung der interessanten Strömungsmuster zerstören. Deswegen verwarfen wir nach kurzen Tests die Standard Finite Volumen Verfahren erster Ordnung und wählten als Verfahren höherer Ordnung die Discontinuous Galerkin Verfahren, die wir in dieser Arbeit ausführlich bearbeitet haben, als numerischen Löser zum Lösen von Gleichung (8.5). Diese weisen sehr viel weniger numerische Viskosität als andere Löser auf. Die numerischen Oszillationen werden durch einen Limitierungsschritt nach jedem Zeitschritt vermieden. In aller Kürze werden wir nun nochmal die Discontinuous Galerkin Verfahren beschreiben, weil man hier eventuell (wenn das Vektorfeld nicht divergenzfrei ist) eine rechte Seite zu betrachten hat.

Nehmen wir an, dass $\mathcal{M}$ ein unstrukturiertes Gitter ist, das das Rechengebiet Ω überdeckt

und aus regulären Elementen E_i für i aus einer Indexmenge $I_{\mathcal{M}}$ besteht. Auf diesem Gitter führen wir den Raum $\mathcal{V}$ der stückweise polynomialen Funktionen, die nicht zwingend stetig über die Kanten sind, ein. Dann betrachten wir die Transportgleichung (8.5), geschrieben in Erhaltungsform

$$\frac{\partial}{\partial t}\rho + \text{div } f(\rho) = \rho \,\text{div } v\,,$$

wobei $f : I\!R \to I\!R^n$ durch $f_i(\rho) := v_i\rho$ gegeben ist. Multipliziert man mit $\psi \in \mathcal{V}$ und integriert über $E \in \mathcal{M}$, so erhält man

$$\frac{\partial}{\partial t}\int_E \rho\,\psi + \int_E \text{div } f(\rho)\,\psi = \int_E \rho \,\text{div } v\,\psi\,.$$

Nach partieller Integration erhalten wir

$$\frac{\partial}{\partial t}\int_E \rho\,\psi + \int_{\partial E} f(\rho)\cdot\nu\,\psi = \int_E \rho \,\text{div } v\,\psi + f(\rho)\nabla\psi\,.$$

Wir ersetzen den Fluss $f(\rho)\cdot\nu$, der den Fluss über die Kanten von E beschreibt, durch $g(\rho^-,\rho^+)$, wobei ρ^- und ρ^+ die stückweise polynomialen, aber eventuell unstetigen Funktionen ρ in E bzw. in $\tilde{E}$, der gegenüberliegenden Zelle von E, sind.
Dabei ist

$$\begin{aligned} g(\rho,\rho) &= f(\rho)\cdot\nu \\ g(\rho^-,\rho^+) &= -g(\rho^+,\rho^-)\,. \end{aligned}$$

So erhalten wir die in dieser Arbeit beschriebenen Discontinuous Galerkin Verfahren. Als numerische Flussfunktion nehmen wir die von Engquist-Osher [24]. Die Zeitableitung diskretisieren wir durch ein Runge-Kutta Schema[55]. Durch Verwendung eines Limiters (siehe Abschnitt 4.8.1) werden künstliche Oszillationen in der numerischen Lösung ausgeschlossen.
In diesen Anwendungen haben wir die Funktionen $\rho = T, X$ auf jedem Volumen E durch eine lineare Funktion approximiert. Abbildung 8.16 zeigt einen Vergleich zwischen einem Verfahren erster Ordnung und einem Discontinuous Galerkin Verfahren höherer Ordnung bei der Komponente X.

8.5 Beispiele in 2 Raumdimensionen

Zunächst betrachten wir den inkompressiblen Fluss um ein zylinderförmiges Hindernis in einem rechteckigen Kanal. Das Modell ist durch die inkompressiblen Navier-Stokes Gleichungen gegeben. Bei moderaten Reynoldszahlen erwarten wir das Strömungsbild der

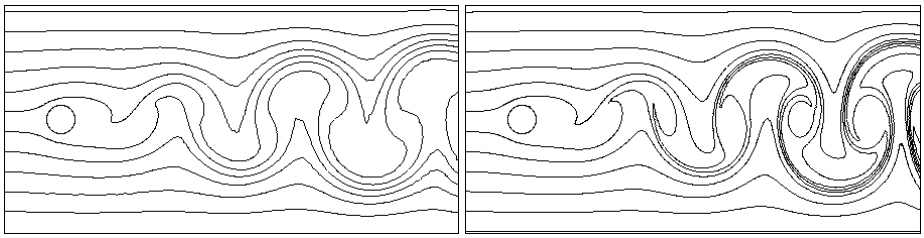

Abbildung 8.16: Ein Vergleich zwischen Verfahren erster und höherer Ordnung für den numerischen Transport der X Komponente.
Dabei werden Isolinien im physikalischen Gebiet Ω zu einer festen Zeit betrachtet.

Karmanschen Wirbelstraße. Die numerischen Geschwindigkeiten v, die wir für die Transportgleichungen benötigen, wurden vorher mit einer gemischten Finite Element Methode mit quadratischen Ansatzfunktionen für v berechnet [2]. Um die Güte der Transportgleichung zu verbessern, wurde jedes Dreieck des Rechengebiets in 16 Dreiecke zerlegt und dann mit der Discontinuous Galerkin Methode die X, T Koordinaten berechnet. Abb.8.17 zeigt zu verschiedenen Zeiten die Entwicklung. Abb. 8.18 zeigt die Skalierbarkeit der Textur zu einem festen Zeitpunkt.
Das nächste Beispiel ist ein Problem ohne Einfluss- und Ausflussrand. Ein interessantes Beispiel ist die Bénard Konvektion in einer rechteckigen Box, die von unten erhitzt und von oben gekühlt wird. Die Formation der Konvektionsrollen wird einen Austausch der Temperatur vornehmen. Wir definieren eine innere Linie L als künstlichen Rand für das Transportproblem. Dabei wird in Abhängigkeit von $v \cdot \nu$ der Einfluss-und Ausflussrand festgelegt. Abb. 8.19 zeigt das Ergebnis des Texturtransport zu verschiedenen Zeiten. Der graue Bereich ist nicht im Bild der Lagrange Koordinaten. Deswegen setzen wir hier den Fehlerindikator $\eta(X, T) = 1$ für $x \notin X(L, t)$.
Das nächste Beispiel ist der Texturtransport für eine kompressible Strömung, die durch die numerische Simulation der Euler-Gleichungen in 2D gegeben ist. In einem Kanal sind zwei zylinderförmige Hindernisse angebracht. Abb. 8.20 zeigt den Texturtransport zu verschiedenen Zeiten.
Das letzte 2D Beispiel ist der kompressible Fluss durch einen Kanal mit einspringender Stufe, das Forward Facing Step Problem. Wie schon im Bild der Dichte in Abbildung 7.8 zu sehen, bilden sich hinter dem Machstamm in der Kontaktunstetigkeit Wirbel, die das Verfahren erster Ordnung (Abb. 7.5) nicht auflöst. Wendet man nun das Texturtransportverfahren auf dieses Beispiel an, so werden diese Wirbel hier ebenfalls sehr gut aufgelöst. Abbildung 8.21 zeigt dies.

8.6 Beispiele in 3 Raumdimensionen

Die Textur Transport Methode soll nun auf den dreidimensionalen Fall übertragen werden. Im 2D Fall hatte man zwei Transportgleichungen zu lösen (für X, T). Nun sind es

drei Transportgleichungen (für $X = (X_1, X_2), T$). Dabei ist speziell der Transport einer zweidimensionalen Einflusskoordinate $X \in \Gamma^+$ zu berechnen. Für die Visualisierung nehmen wir die Idee der impliziten Stromflächen von J. van Wijk [67] auf. Betrachten wir eine implizit definierte Kurve $\gamma = \{X \in \Gamma^+ \mid g(X) = 0\}$ für eine reguläre Funktion g auf Γ^+. Mit $s : [0,1] \to \Gamma^+$ bezeichnen wir eine Parametrisierung von γ, welche wir als periodisch annehmen, falls γ geschlossen ist. Dann ist das Bild $X(\gamma, \cdot)$ von γ unter der Koordinaten Abbildung X eine Stromfläche. Diese Oberfläche kann auf einem diskreten Gitter durch einen diskreten Isoflächen Algorithmus extrahiert werden. Abhängend von der Parametrisierung s kann dieselbe Textur wie in den 2D Anwendungen benutzt werden. Wenn wir weiter Familien von implizit parametrisierten Kurven betrachten, erhalten wir einen stetigen Übergang im Texturbild. Anstatt nach impliziten Kurven auf Γ^+ können wir auch nach Bildern von impliziten Oberflächen auf $\Gamma^+ \times [0, \hat{T}]$ fragen.
Die folgende Tabelle 8.1 soll verdeutlichen, welche Möglichkeiten es für die Wahl einer Oberfläche gibt. Sei die Oberfläche durch die Menge $\{(x,y,z) \mid f(X,T) = lev\}$ gegeben. Die erste Abbildung 8.22 zeigt eine Oberflächendeformation durch die Lagrange Koordinaten Abbildung. Ein Ellipsoid wird auf $\Gamma^+ \times [0, \hat{T}]$ beschrieben.
In diesem 3D Testbeispiel sei $v = v(x,y,z,t)$ gegeben durch

$$v_1(x,y,z,t) = 0.2$$

$$v_2(x,y,z,t) = -zZ(t)$$

$$v_3(x,y,z,t) = yZ(t)$$

mit

$$Z(t) = \begin{cases} 1.5 \text{ , falls } t < 3.0 \\ 4.5 - t \text{ , falls } t \in [3.0; 6.0] \\ -1.5 \text{ , falls } t > 6.0 \end{cases} .$$

Dargestellt ist die Oberfläche $\sqrt{(X_1 - X_{10})^2 + (X_2 - X_{20})^2 + a(T - T_0)^2}$ mit fixierten Werten X_{10},X_{20},T_0 und a. Die Textur wird benutzt abhängend von $(\alpha(X_1, X_2), T)$.
Das nächste Beispiel ist die Deformation einer Fläche beim Forward Facing Step Beispiel in 3D (Abb. 8.23). Gerechnet wurden die Euler-Gleichungen mit den Discontinuous Galerkin Verfahren. Simultan dazu wurde mit dem berechneten Vektorfeld der Texturtransport berechnet. Siehe auch Abbildung 7.13, wo die Dichte dargestellt ist.
Im abschließenden Beispiel ist die Karmansche Wirbelstraße in 3D zu sehen (Abb.8.24). Dargestellt ist eine Kurve auf der Einlassfläche, die durch das Geschwindigkeitsfeld transportiert wird. Über diese Kurve fliessen die Partikel für alle Zeiten $t \in [0, \hat{T}]$ nach. Als Ergebnis ist eine "Röhre" zu sehen, auf der wir wieder das aus der 2D Version bekannte Bild der Karmanschen Wirbelstraße erkennen. Die Fläche wird hier in diesem Fall in Abhängigkeit aller Lagrange Koordinaten bestimmt (siehe dazu die fünfte Zeile in Tabelle 8.1). Die Textur wird ebenfalls in Abhängigkeit aller Lagrange Koordinaten bestimmt.

Isofläche $f(X,T)$	Textur
X_1	(X_2, T)
X_2	(X_1, T)
T	(X_1, X_2)
T	(X_2, X_1)
$\sqrt{(X_1 - X_{10})^2 + (X_2 - X_{20})^2 + a(T - T_0)^2}$	$(\alpha(X_1, X_2), T)$

Tabelle 8.1: Mögliche Wahl von Level-Funktionen und Texturen.

Tabelle 8.1 zeigt fünf Möglichkeiten, bestimmte Level mit ihren zugehörigen Texturen zu belegen. Möglichkeit 1: Die Isofläche $X_1(x_1, x_2, x_3, t) = lev$ wird dargestellt. Als Textur wird $\pi(\eta(X,T), \lambda X_2)$ und $C_2(T)$ ausgewertet.

Möglichkeit 2: Die Isofläche $X_2(x_1, x_2, x_3, t) = lev$ wird dargestellt. Als Textur wird $\pi(\eta(X,T), \lambda X_1)$ und $C_2(T)$ ausgewertet.

Möglichkeit 3: Die Isofläche $T(x_1, x_2, x_3, t) = lev$ wird dargestellt. Als Textur wird $\pi(\eta(X,T), \lambda X_1)$ und $C_2(X_2)$ ausgewertet.

Möglichkeit 4: Die Isofläche $T(x_1, x_2, x_3, t) = lev$ wird dargestellt. Als Textur wird $\pi(\eta(X,T), \lambda X_2)$ und $C_2(X_1)$ ausgewertet.

Möglichkeit 5: Die Isofläche $\sqrt{(X_1 - X_{10})^2 + (X_2 - X_{20})^2 + a(T - T_0)^2} = lev$ wird dargestellt. Als Textur wird $\pi(\eta(X,T), \lambda\alpha(X_1, X_2))$ und $C_2(T)$ ausgewertet.

Dabei sind $X_{10}, X_{20}, T_0, lev, a$ frei wählbare Parameter und α ein Winkel. Dieser Winkel erkennt z.B., an welcher Stelle auf der Kreisfunktion auf dem Einflussrand ein Partikel eingeflossen ist ($\alpha = arccos(X_1 - X_{10})$ und $\alpha = arcsin(X_2 - X_{20})$).

Mit der fünften Möglichkeit wird eine Kurve auf der Einlassfläche beschrieben, die durch das Geschwindigkeitsfeld transportiert wird. Über diese Kurve fliessen die Partikel für alle Zeiten $t \in [0, \hat{T}]$ nach. Als Ergebnis ist eine ”Röhre” zu sehen.

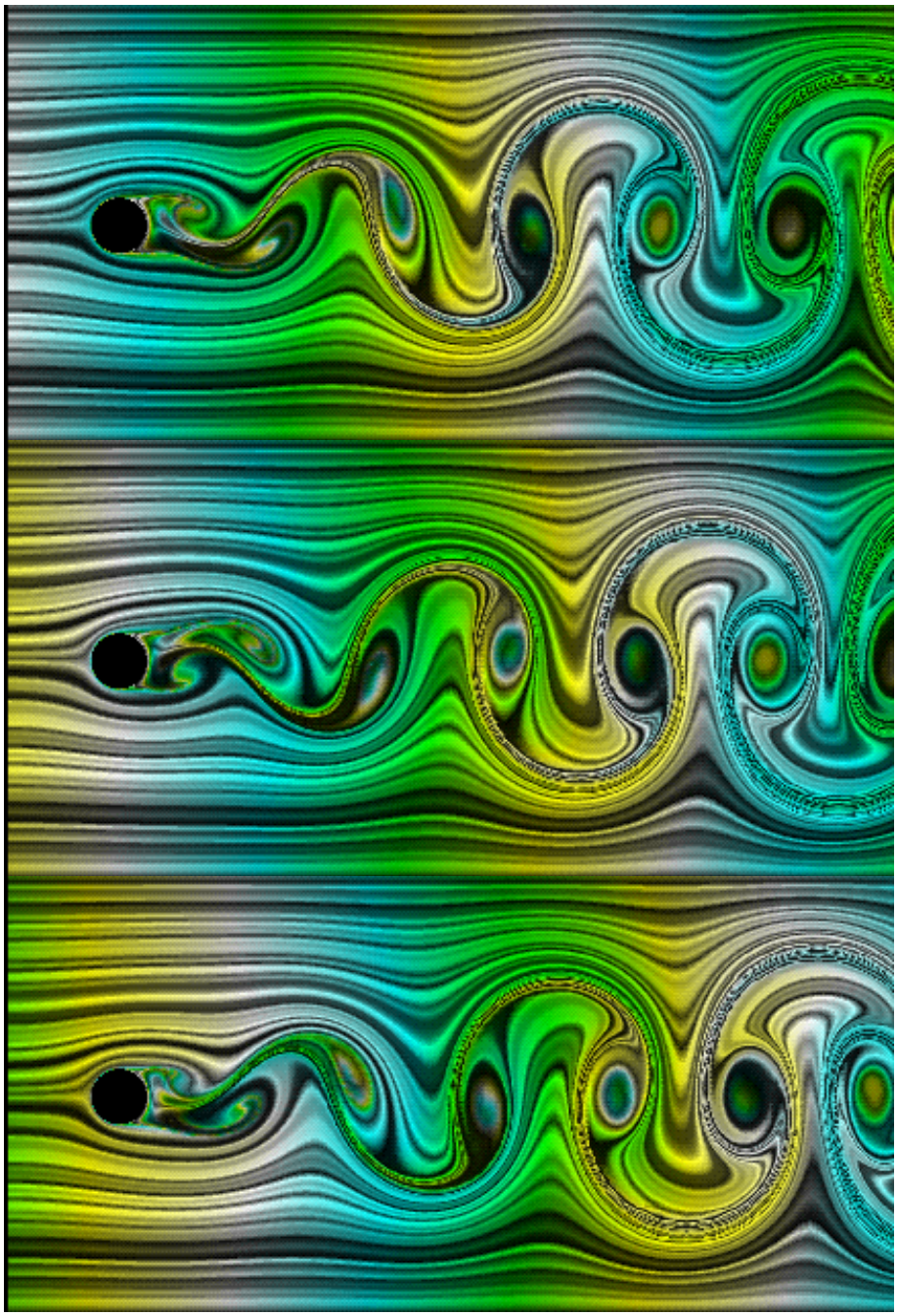

Abbildung 8.17: Texturtransport in der Karmanschen Wirbelstraße.

Abb. 8.17 zeigt den Texturtransport in der Karmanschen Wirbelstraße zu drei verschiedenen Zeitpunkten. Das Vektorfeld ist gegeben durch die inkompressiblen Navier-Stokes Gleichungen und wurde zur Verfügung gestellt von [2].

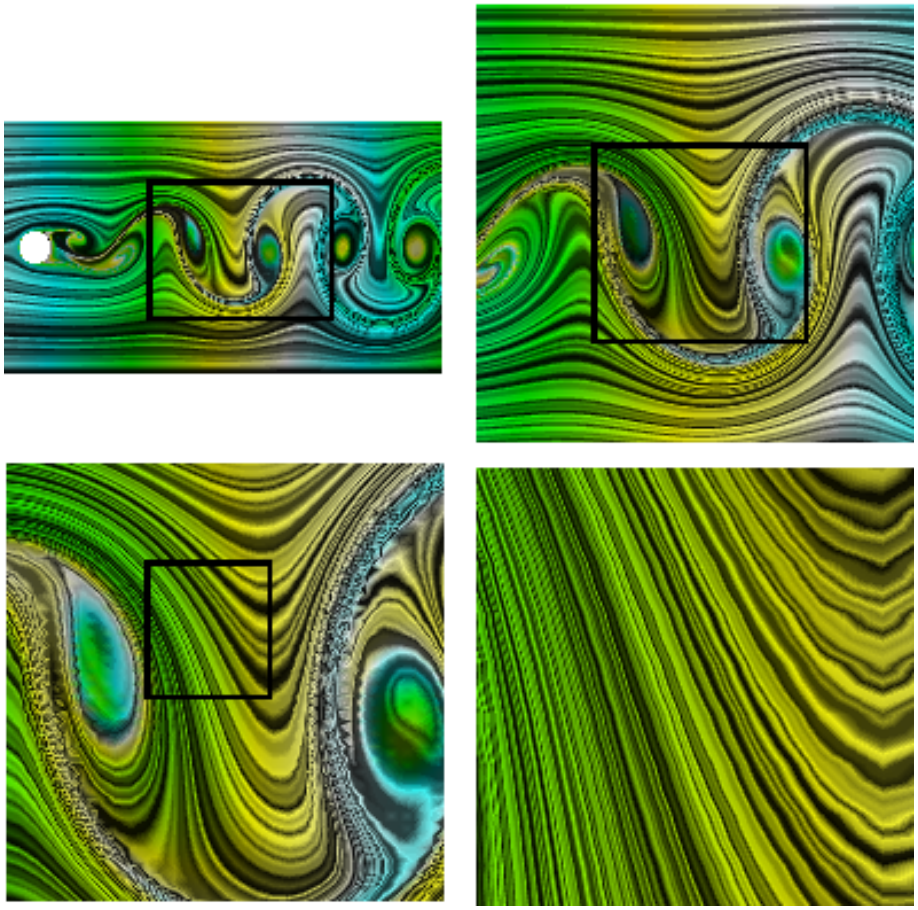

Abbildung 8.18: Verschiedene Vergrößerungen im Gebiet Ω.

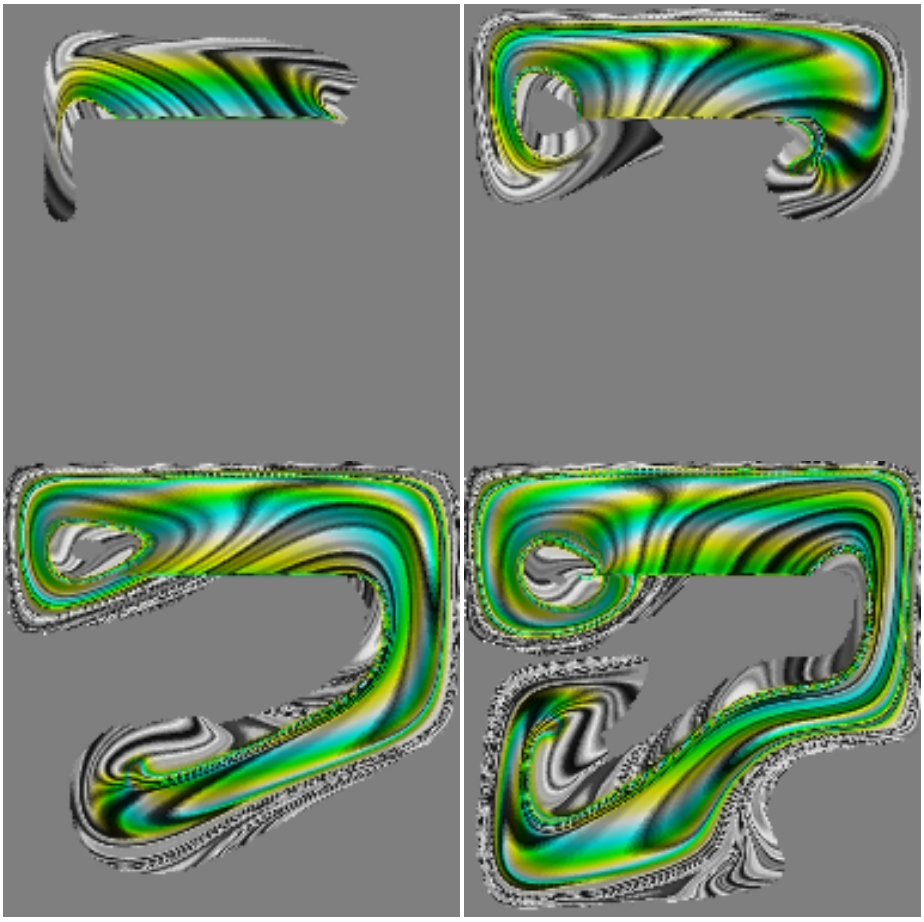

Abbildung 8.19: Texturtransport beim Bernard-Problem.

Abb. 8.19 zeigt den Texturtransport beim Bernard-Problem zu vier verschiedenen Zeitpunkten. Das Vektorfeld ist gegeben durch die Maragoni Konvektion.

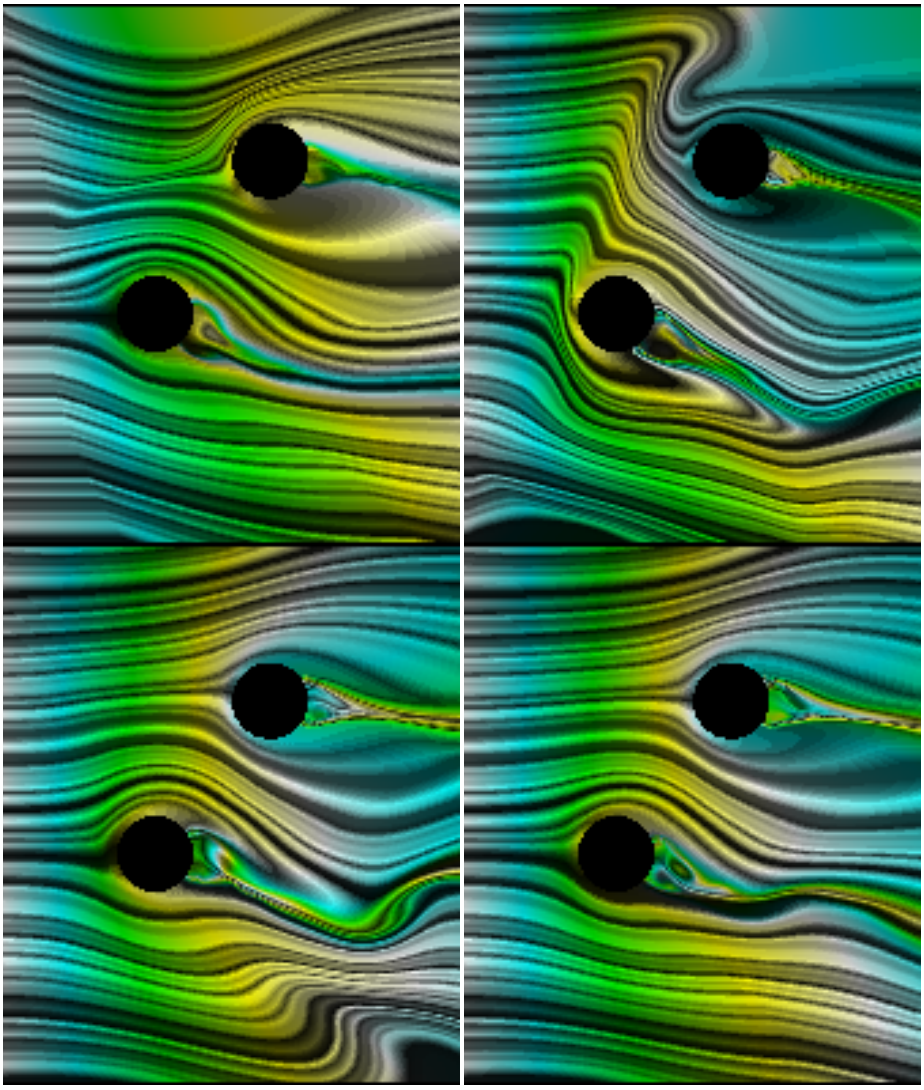

Abbildung 8.20: Texturtransport bei der Strömung um zwei Zylinder.

Abb. 8.20 zeigt den Texturtransport bei der Strömung um zwei Zylinder zu vier verschiedenen Zeitpunkten. Das Vektorfeld ist gegeben durch die kompressiblen Euler-Gleichungen und wurde mit einem Verfahren erster Ordnung berechnet.

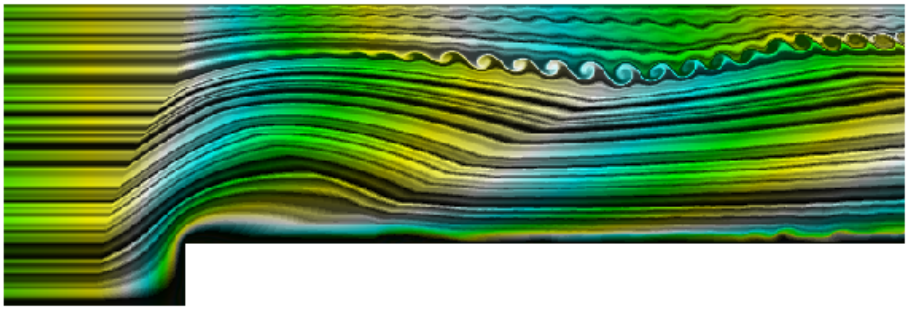

Abbildung 8.21: Texturtransport beim Forward Facing Step-Problem.

Abb. 8.21 zeigt den Texturtransport beim Forward Facing Step-Problem. Das Vektorfeld ist gegeben durch die kompressiblen Euler-Gleichungen und wurde mit einem Verfahren zweiter Ordnung (siehe Dichtebild 7.8 zum Vergleich) berechnet.

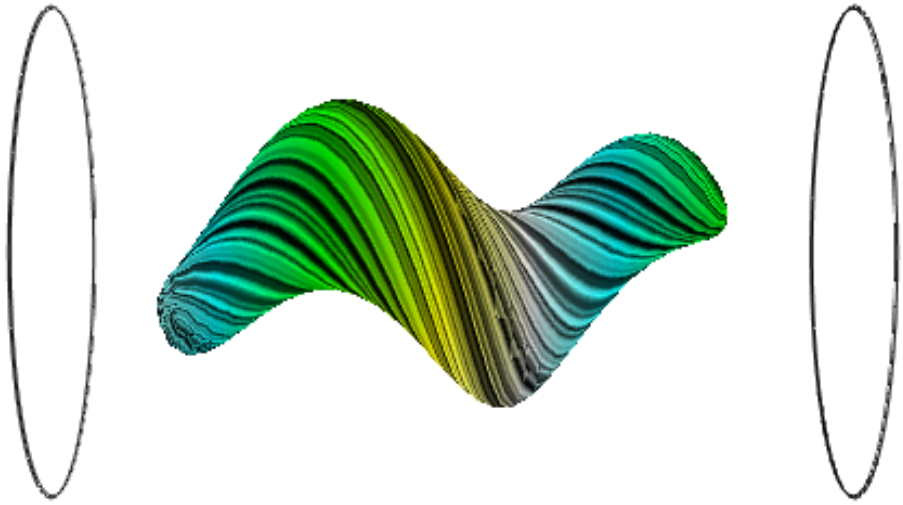

Abbildung 8.22: 3D Textur Transport (Röhre)

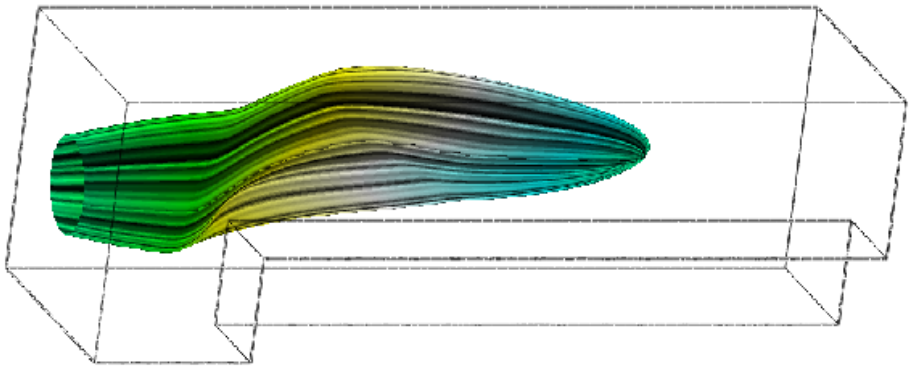

Abbildung 8.23: 3D Textur Transport (Forward Facing Step)

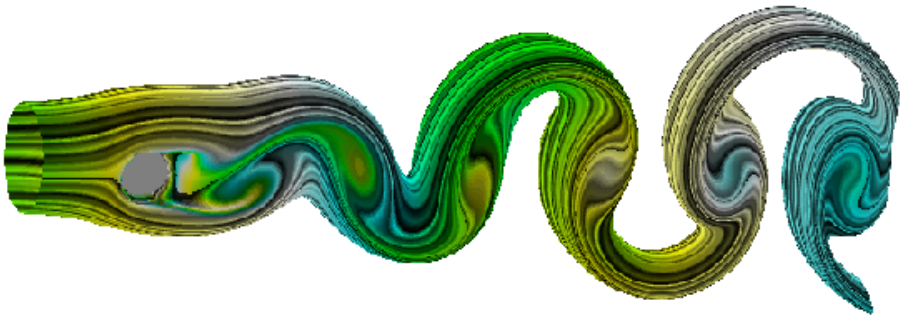

Abbildung 8.24: 3D Textur Transport (Karmansche Wirbelstraße)

Kapitel 9

Volumenvisualisierung (Volume Rendering)

Das Problem der graphischen Darstellung von Volumendaten gilt als zentrales Forschungsgebiet der wissenschaftlichen Visualisierung. Die heute bekannten Verfahren zur Visualisierung dreidimensionaler Skalarfelder, wie sie aus Messungen (speziell den bildgebenden Verfahren der Medizin) oder aus Simulationsrechnungen (z.B. aus der Strömungsmechanik) herrühren, zerfallen in zwei Klassen: Konturflächenbestimmung und direkte Volumenvisualisierung. Die Vorteile der ersten, nämlich die Reduktion der gesamten Volumeninformation auf eine Flächenbeschreibung werden durch die Vernachlässigung kleiner, amorpher und nicht auf der Isofläche liegender Strukturen, sowie durch algorithmische Artefakte erkauft. Dieser Nachteil führte zur Entwicklung der direkten Volume-Rendering-Verfahren, bei denen im Prinzip jedes Volumenelement einen Beitrag zum endgültigen Bild liefert und tiefer liegende Schichten durch Transparenz sichtbar gemacht werden. Dabei können, wenn auch mit enorm hohem Berechnungsaufwand, durch die flexible Abbildung der Datenwerte auf Farbe und Opazität (Transferfunktionen), sowie durch Voxel-Shading unterschiedliche Strukturen und Bereiche ausdrucksstark visualisiert werden [40]. Ein Voxel ist ein Punkt im Dreidimensionalen (vergleiche Definition 8.1.4 von Pixel im Zweidimensionalen). Die verwendeten Methoden werden in zwei große Gruppen aufgeteilt: Projektionsverfahren und Strahlverfolgungsverfahren. Bei den ersteren werden die Daten, bzw. der ”Fußabdruck” des Voxels auf die Bildebene projiziert, bei der zweiten Gruppe werden gemäß dem Raytracing-Verfahren für jeden Bildpunkt Strahlen durch das Volumen geschickt und die Intensitäten und Opazitäten entlang dieses Sehstrahls aufintegriert.
Die im späteren Verlauf des Kapitel zu sehenden Volume Rendering Bilder wurden wurden nach einem Algorithmus von Gerstner und anderen [29], [30] unter der Visualisierungsplattform Grape [28] gemacht. Bei diesem Algorithmus arbeitet man mit Würfelpackungen, die hierarchisch angeordnet sind. Durch die spezielle Struktur des Gitters (man hat immer $n^3, n = 2^k, k \in \mathbb{N}$ Würfel) kann man einen sehr schnellen Algorithmus bei der Strahlverfolgung anwenden. Ein Würfel wird dabei jeweils in sechs Tetraeder ge-

teilt.
In diesem Kapitel soll eine weitere Möglichkeit beschrieben werden, die Lagrange Koordinaten $X = (X_1(x, y, z, t), X_2(x, y, z, t))$ und $T = T(x, y, z, t)$ sinnvoll darzustellen. Betrachten wir den Einheitswuerfel $[0; 1]^3$. In zufällig gewählte Punkte (x_0, y_0, z_0) werden Ellipsoide gelegt.
Seien Konstanten $\epsilon_x, \epsilon_y, \epsilon_z$ mit

$$\epsilon_x >> \epsilon_y \text{ und } \epsilon_x >> \epsilon_z$$

gegeben.
Betrachte die Funktion

$$\hat{\rho}(x, y, z) = e^{-\frac{(x-x_0)^2}{\epsilon_x^2}} e^{-\frac{(y-y_0)^2}{\epsilon_y^2}} e^{-\frac{(z-z_0)^2}{\epsilon_z^2}} .$$

Mit Hilfe dieser Funktion definieren wir

$$\rho(x, y, z) = \sum_{(x_0,y_0,z_0)} \hat{\rho}(x, y, z) .$$

Nun betrachten wir folgende Mengen

$$\{(x, y, z) | \rho(x, y, z) = c \text{ mit } c \in \mathbb{R}\} .$$

Abb. 9.1 zeigt eine solche Menge für ein festes c.

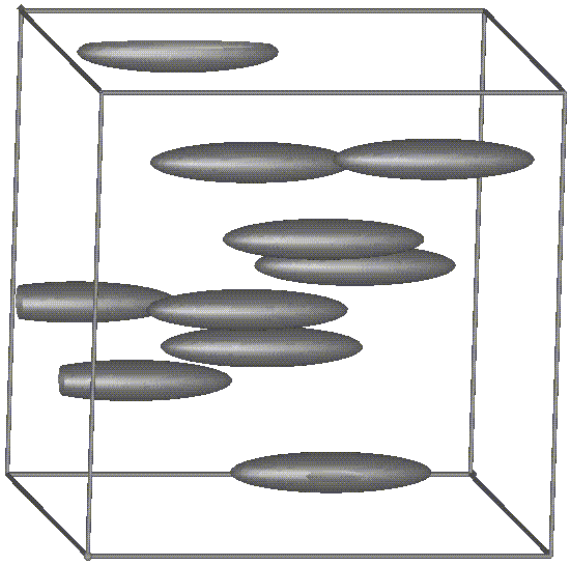

Abbildung 9.1: Ellipsoide im Einheitswürfel

Abb. 9.1 zeigt die Isosurface $\{(x, y, z) | \rho(x, y, z) = 0.5\}$.

Wenn man durch geeignete Wahl der Ellipsoid-Zentren (x_0, y_0, z_0) erreicht, dass die Ellipsoide nicht den Rand "durchstoßen", kann man an allen Seiten dieses Würfels einen Würfel derselben Art ansetzen. Man hat somit eine periodische Funktion.

Alternativ kann auch man auch die Funktionen

$$\phi_1(x, y, z) = x \text{ , } \phi_2(x, y, z) = y \text{ und } \phi_3(x, y, z) = z$$

betrachten.

Das erste Anwendungsbeispiel ist die inkompressible Strömung in der Karmanschen Wirbelstraße. Dabei wurde der 2D-Datensatz in die dritte Dimension geshiftet.
Als Isosurface stellen wir dann die Menge

$$\{(x, y, z) \mid \rho(T, X_1, X_2) = const\}$$

dar.
Abb. 9.2 zeigt die mögliche Anwendung in der dreidimensionalen Karmanschen Wirbelstraße.

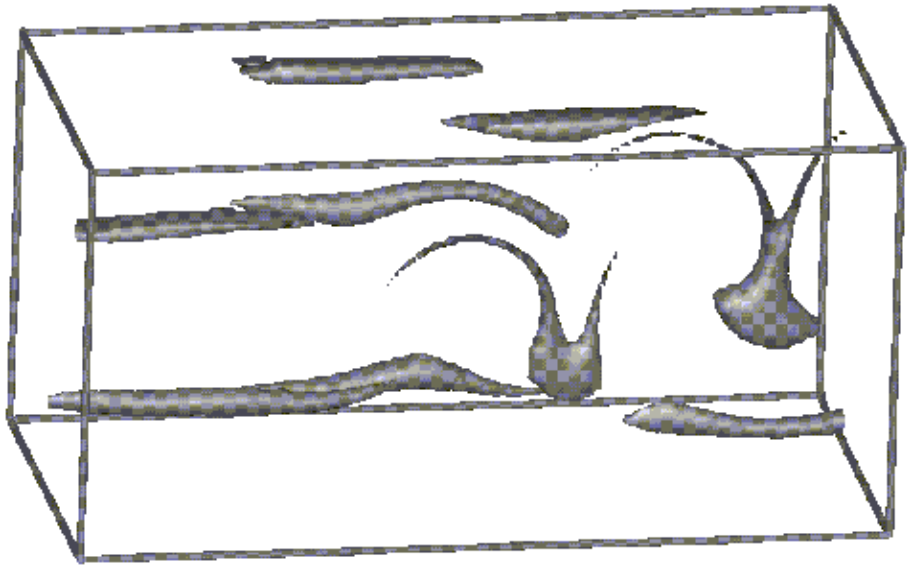

Abbildung 9.2: Ellipsoide (Karmansche Wirbelstraße) (Isosurface).

Abb. 9.2 zeigt die Isosurface $\{(x, y, z) | \rho(T, X_1, X_2) = 0.5\}$ zu einer festen Zeit. Die Ellipsoide werden dabei durch das Vektorfeld transportiert und somit verformt.

Bei der Visualisierung nach der Volume Rendering Methode ergibt sich folgendes Bild (siehe Abb. 9.3). Jeder Voxel liefert einen Eintrag zum Bild.
Die Abbildungen 9.4,9.5 zeigen die Funktion $\phi_3(T, X_1, X_2)$. Zunächst wird zu einem konstanten Wert eine feste Isosurface und dann das Volume Rendering dargestellt.

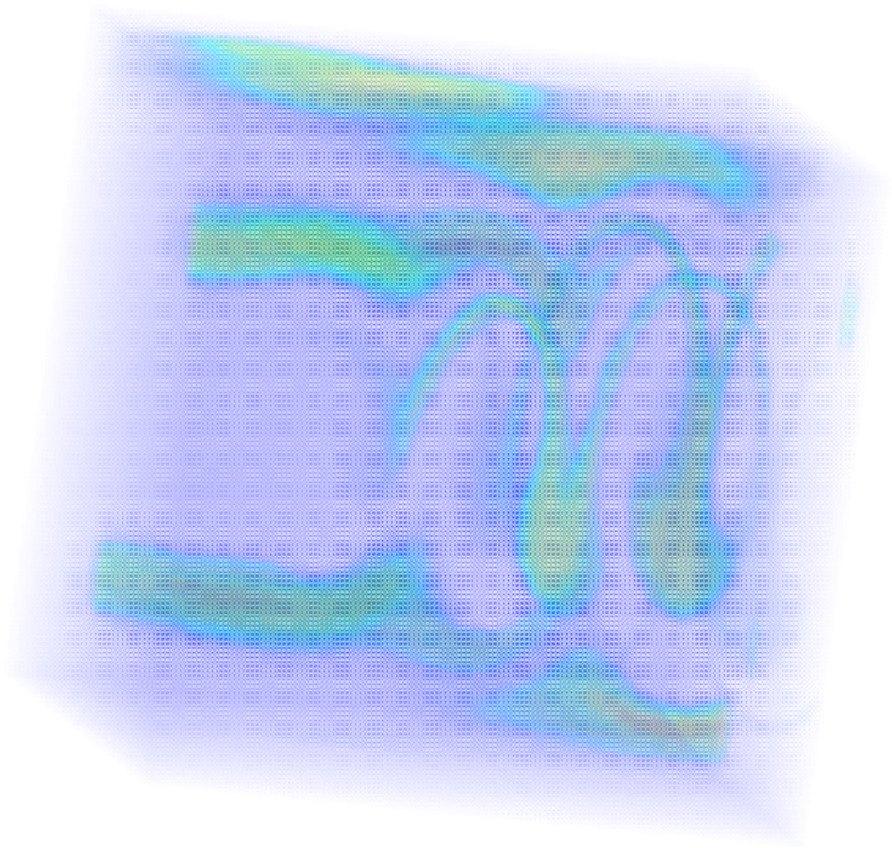

Abbildung 9.3: Ellipsoide (Karmansche Wirbelstraße) (Volume Rendering).
Abb. 9.3 zeigt das Volume Rendering zu der Funktion $\rho(T, X_1, X_2)$ zu einer festen Zeit. Man sieht deutlich die dreidimensionale Struktur. Hinten liegende Teile werden durch Transparenz sichtbar gemacht.

Die entprechende Transferfunktion (Abb. 9.6) sorgt für die Durchlässigkeit oder Undurchlässigkeit der entsprechenden Dichten. Wir machen die Annahme, dass die Dichten sich im Intervall $[0; 1]$ befinden. Wenn die Undurchlässigkeit den Wert 1 annimmt, so ist dieser Dichtebereich sehr ausgeprägt zu sehen. Wenn die Undurchlässigkeit den Wert 0 annimmt, so ist dieser Dichtebereich nicht zu sehen.
Das zweite Anwendungsbeispiel zeigt die kompressible Strömung in drei Raumdimensionen um drei Hindernisse. Das Strömungsgebiet ist folgendermaßen gegeben:

$$\begin{aligned}\Omega = \quad & ([0.0; 1.0] \times [0.0; 1.0] \times [0.0; 1.0]) \\ & \setminus([0.1; 0.3] \times [0.4; 0.6] \times [0.4; 0.6]) \\ & \setminus([0.2; 0.4] \times [0.65; 0.85] \times [0.65; 0.85]) \\ & \setminus([0.2; 0.4] \times [0.15; 0.35] \times [0.15; 0.35])\,.\end{aligned}$$

Gerechnet wurden die Euler-Gleichungen der Gasdynamik in drei Raumdimensionen. Die Ellipsoide wurden wieder durch das berechnete Geschwindigkeitsfeld transportiert. Die

Abbildung 9.4: Isosurfaces in der Karmanschen Wirbelstraße.
Abb. 9.4 zeigt die Isosurface $\{(x, y, z)|\phi_3(T, X_1, X_2) = 0.5\}$ zu einer festen Zeit.

Abbildungen 9.7, 9.8, 9.9 zeigen zunächst das Gebiet und dann wieder einen Vergleich zwischen einer festen Isosurface und dem Volume Rendering.

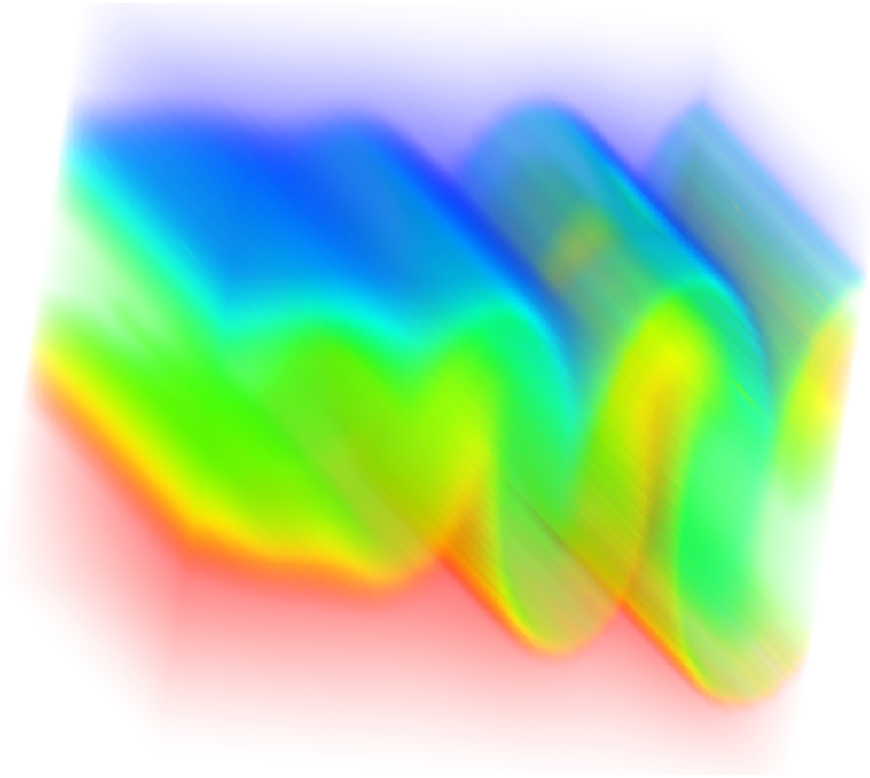

Abbildung 9.5: Volume Rendering in der Karmanschen Wirbelstraße.

Abb. 9.5 zeigt das Volume Rendering der Funktion $\phi_3(T, X_1, X_2)$ zu einer festen Zeit.

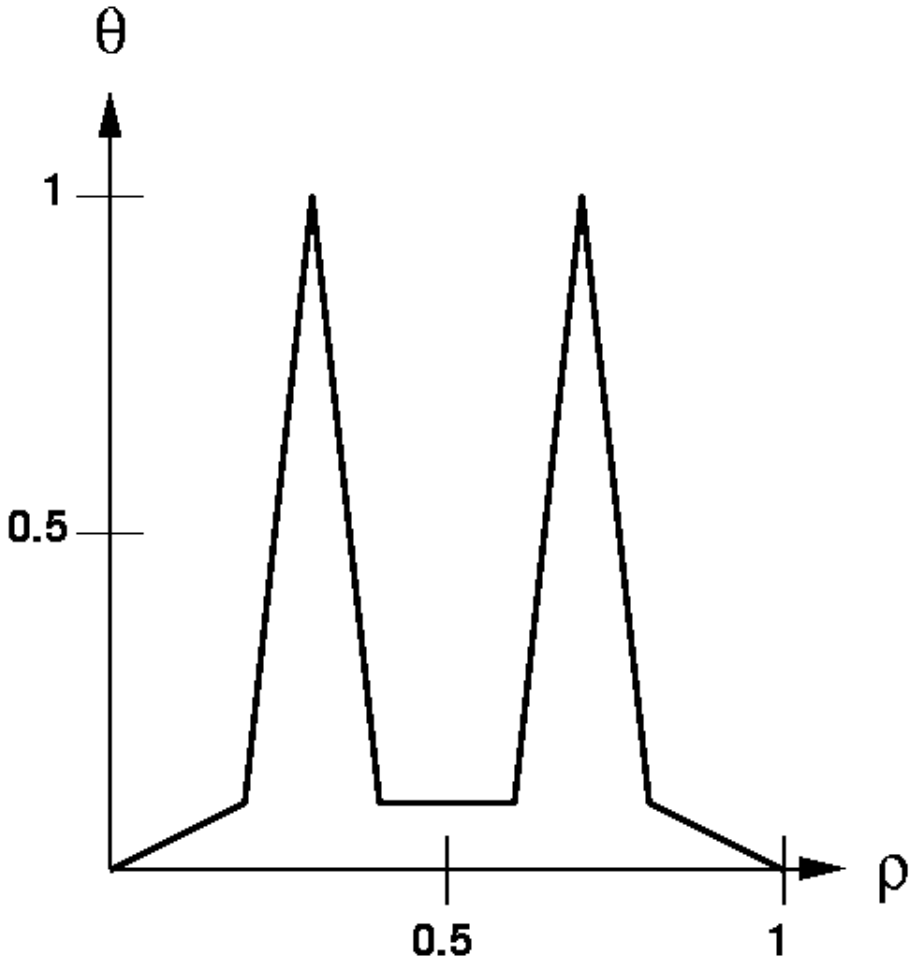

Abbildung 9.6: Transferfunktion für das Volume Rendering der Funktion $\phi_3(X_1, X_2, T)$.

Abb. 9.6 zeigt die Transferfunktion für Abb. 9.5. Hohe Werte von θ bedeuten wenig Durchlässigkeit. Deshalb werden die Farben hellblau und gelb in Abb. 9.5 sehr ausgeprägt gezeichnet. θ nahe der Null bedeutet viel Durchlässigkeit. Für $\theta = 0$ liefert der entsprechende Dichtewert keinen Eintrag zum Gesamtbild.

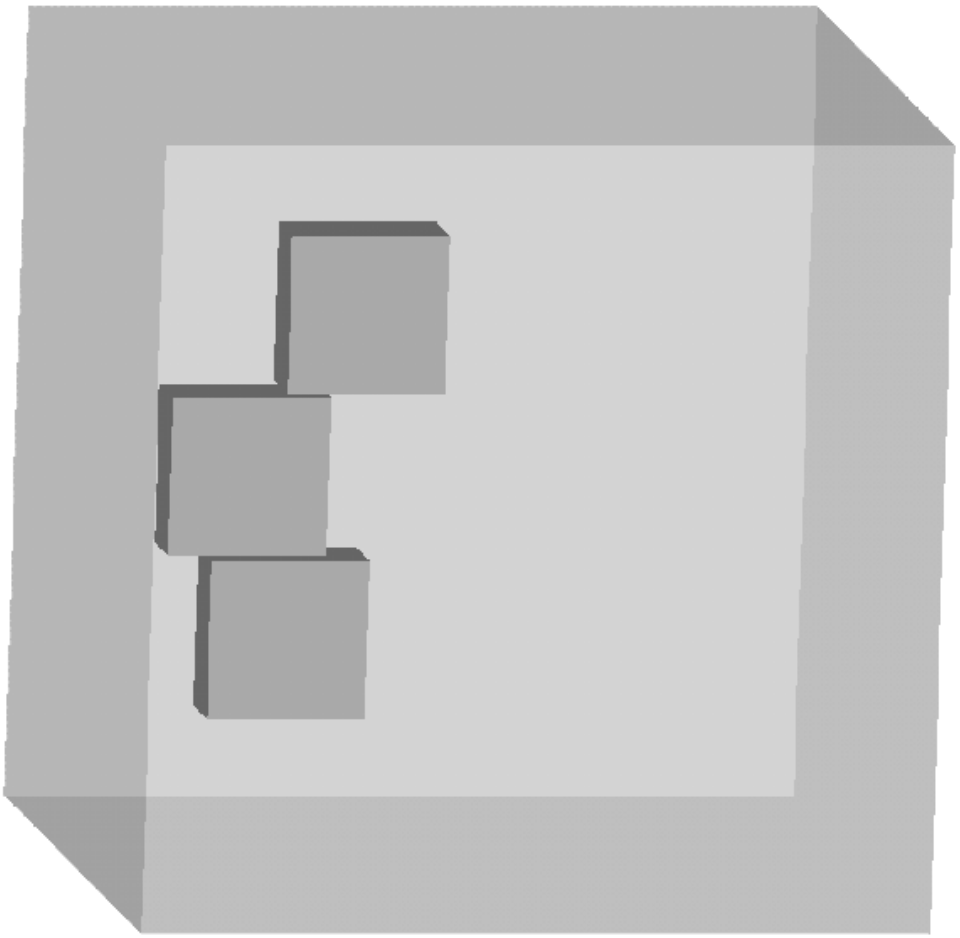

Abbildung 9.7: Strömungsgebiet.

Abb. 9.7 zeigt das Gebiet. Hier wurden die kompressiblen Euler-Gleichungen in drei Raumdimensionen gerechnet. Mit dem berechneten Geschwindigkeitsfeld wurden die Transportgleichungen für die Koordinaten X_1, X_2, T berechnet.

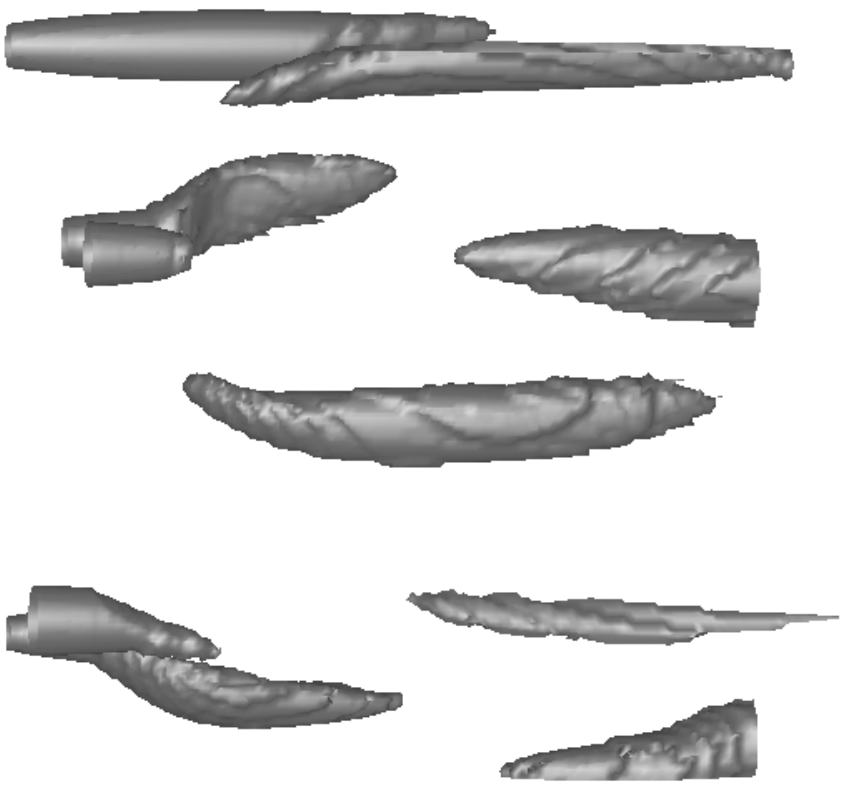

Abbildung 9.8: Isosurface im Strömungsgebiet.

Abb. 9.8 zeigt die Isosurface $\{(x, y, z) \mid \rho(T, X_1, X_2) = const\}$ zu einer festen Zeit.

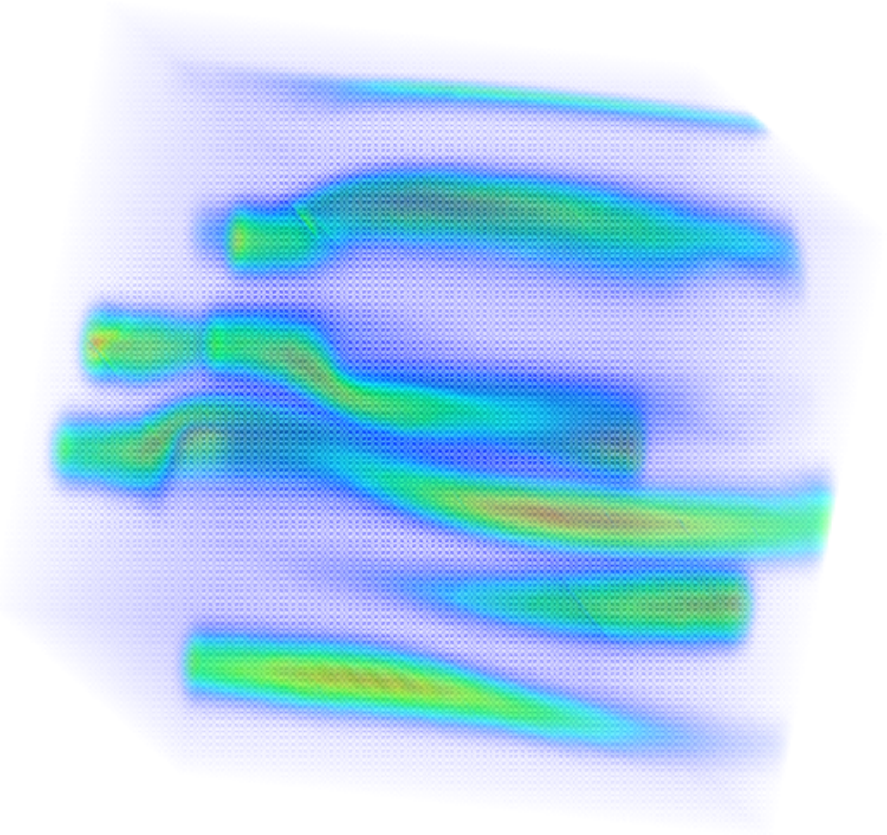

Abbildung 9.9: Volume Rendering im Strömungsgebiet.

Abb. 9.9 zeigt das Volume Rendering der Funktion $\rho(X_1, X_2, T)$ zu einer festen Zeit.

Kapitel 10

Zusammenfassung und Ausblick

In dieser Arbeit haben wir die Discontinuous Galerkin Verfahren für Systeme von hyperbolischen Differentialgleichungen betrachtet. Wir haben die Verfahren ausführlich in zwei und drei Raumdimensionen analysiert und mit den Standard-Verfahren höherer Ordnung wie MUSCL verglichen. Die 3D Version der Discontinuous Galerkin Verfahren wurde in dieser Arbeit zum ersten Mal analysiert und auf einem Parallelrechner umgesetzt. Dieses Verfahren lieferte eindrucksvolle Ergebnisse, die allerdings teuer erkauft wurden. Die Discontinuous Galerkin Verfahren sind sehr speicher- und rechenintensiv. Deswegen war es unvermeidbar, die Verfahren auf einem Parallelrechner zu parallelisieren. Dabei wurde sowohl die Variante unter MPI auf einem verteilten Rechner als auch die Shared Memory Variante umgesetzt. In dieser Arbeit haben wir die kompressiblen Euler-Gleichungen der Gasdynamik betrachtet. In Zukunft könnte man sich vorstellen, die kompressiblen Navier-Stokes-Gleichungen [53] ebenfalls mit den Discontinuous Galerkin Verfahren zu lösen. Dabei würde man den konvektiven Teil mit den in dieser Arbeit beschriebenen Methoden lösen. Den diffussiven Teil würde man wie bisher auf direktem Weg lösen. Eine weitere Anwendungsmöglichkeit ergibt sich bei den Gleichungen der Magnetohydrodynamik. Hier wäre die Adaption noch sehr viel einfacher, weil man im Wesentlichen nur die numerischen Flussfunktionen anpassen müsste.
Um die numerischen Lösungen dieser Differentialgleichungen zu verstehen und auch dem Ungeübten verständlich zu machen, ist es nötig, diese zu visualisieren. Dazu haben wir in dieser Arbeit einen Ansatz unternommen. Wir haben in dieser Arbeit ein neues Verfahren zur Visualisierung zeitabhängiger Vektorfelder entwickelt. Dabei wird eine vorgegebene Textur mit einem Geschwindigkeitsfeld transportiert. Dieses Verfahren gibt dem Betrachter die Möglichkeit, das Geschwindigkeitsfeld global auch für lange Zeiten zu verstehen. Mit den bisher zur Verfügung stehenden Verfahren hatte man nur für stationäre Vektorfelder sinnvolle Visualisierungstechniken. Bei diesen Texturtransportrechnungen waren auch numerische Verfahren vom hyperbolischen Typ zu betrachten. Da man hier besonders Verfahren ohne viel numerische Viskosität benötigte, kamen hier die Discontinuous Galerkin Verfahren in der skalaren Version zur Anwendung.

Anhang A

Danksagung

Folgenden Personen möchte ich besonders für die Hilfe bei der Erstellung dieser Dissertation danken:

- Professor Dr. Dietmar Kröner: Für die Ermöglichung der Arbeit, das interessante Thema und die Betreuung des numerischen Teils während dieser Zeit.
- Professor Dr. Martin Rumpf: Für die Betreuung des graphischen Teils der Arbeit. In Zusammenarbeit mit ihm wurde das Texturtransportverfahren entwickelt.
- Allen Mitarbeitern und Studenten des Institut für Angewandte Mathematik der Albert-Ludwigs-Universität, die mir während der Promotion mit Rat und Tat zur Seite standen. Besonders ist hier mein langjähriger Zimmerkollege Dipl.-Phys. Bernhard Schupp zu nennen, der mir bei der Umsetzung der parallelen Algorithmen eine sehr große Hilfe war. Weiterhin möchte ich Dr. Christian Rohde und Dipl.-Math. Mario Ohlberger für das Korrekturlesen der Arbeit danken.

Anhang B

Listings

Während dieser Arbeit wurde eine sehr große Anzahl von Programmen entwickelt. Diese hier alle zu erwähnen oder gar abzudrucken, würde den Rahmen sprengen. Diese wurden dann sowohl seriell als auch parallel unter MPI oder Shared Memory entwickelt. Zu nennen währen u.a:

- (2D): skalarer Fall : Quadrate : Verfahren erster Ordnung und Discontinuous Galerkin Verfahren der Ordnung 2. bis 7.
- (2D): skalarer Fall : Dreiecke : Verfahren erster Ordnung und Discontinuous Galerkin Verfahren der Ordnung 2. bis 3.
- (3D): skalarer Fall : Hexaeder : Verfahren erster Ordnung und Discontinuous Galerkin Verfahren der Ordnung 2. bis 3.
- (3D): skalarer Fall : Tetraeder : Verfahren erster Ordnung und Discontinuous Galerkin Verfahren der Ordnung 2.
- (2D): Systeme (Euler-Gleichungen) : Quadrate : Verfahren erster Ordnung und Discontinuous Galerkin Verfahren der Ordnung 2. bis 3.
- (2D): Systeme (Euler-Gleichungen) : Dreiecke : Verfahren erster Ordnung und Discontinuous Galerkin Verfahren der Ordnung 2.
- (3D): Systeme (Euler-Gleichungen) : Hexaeder : Verfahren erster Ordnung und Discontinuous Galerkin Verfahren der Ordnung 2. bis 3.
- (3D): Systeme (Euler-Gleichungen) : Tetraeder : Verfahren erster Ordnung und Discontinuous Galerkin Verfahren der Ordnung 2.

Exemplarisch wird hier das Programmpaket zur Lösung der Euler-Gleichungen in 2D auf Quadraten mit den Discontinuous Galerkin Verfahren der Ordnung 2 in der seriellen Version abgedruckt.

Literaturverzeichnis

[1] E. Bänsch: Local Mesh Refinement in 2 and 3 Dimensions. Report 6, SFB 256, Bonn 1990.

[2] E. Bänsch: Simulation of instationary, incompressible flows. Submitted to Acta Math. Univ. Comenianae.

[3] H. Battke, D. Stalling, H.-C. Hege: Fast Line Integral Convolution for Arbitrary Surfaces in 3D. Preprint SC 96-59 ZIB (Dez.1996)

[4] J. Becker: Finite Volumen Verfahren in 2-D für Systeme von hyperbolischen Differentialgleichungen mit Flussfunktion von Osher und Solomon. Institut für Angewandte Mathematik, Universität Bonn, Diplomarbeit,1995.

[5] J. Becker, M. Rumpf: Visualization of time-dependent velocity fields by texture transport. Preprint 14/1998,Mathematische Fakultät, Universität Freiburg. Springer Lecture Notes von Eurographics 1998 in Blaubeuren.

[6] N. Botta, R. Jeltsch: A numerical method for unsteady flows. Research Report No. 94-11 (Sept.1994)

[7] B. Cabral, L. Leedom: Imaging Vector Field Using Line Integral Convolution. Computer Graphics Proceedings, Annual Conference Series 1993.

[8] G. Chavent, B. Cockburn: The Local Projection P0-P1-Discontinuous-Galerkin Finite Element Method For Scalar Conservation Laws. Mathematical Modelling and Numerical Analysis Vol.23,N 4,1989,p.565-592.

[9] A.J. Chorin: Random Choice Solution of Hyperbolic Systems. J. of Comp. Physics 22 (1976), 517-533.

[10] S. Chakravarthy, S. Osher: Numerical Experiments with the Osher Upwind Scheme for the Euler Equations. AIAA Paper-82-0975.

[11] S. Chakravarthy, S. Osher: High Resolution Applications of the Osher Upwind Scheme for the Euler Equations. AIAA Paper-83-1943

[12] P.G. Ciarlet: The finite element method for elliptic problems. North Holland, Amsterdam 1978

[13] B. Cockburn: An Introduction to the Discontinuous Galerkin Method for Convection-Dominated Problems. In B. Cockburn, C. Johnson, C.-W. Shu, E. Tadmor: Advanced Numerical Approximation of Nonlinear Hyperbolic Equations. Cetraro,Italien 1997. Springer Lecture Notes (1997).

[14] B. Cockburn, C.-W. Shu: TVB Runge-Kutta Local Projection Discontinuous-Galerkin Finite Element Method For Conservation Laws II: General Framework. Mathematics of Computation, Vol.52,Nu.186,April 1989,p.411-435.

[15] B. Cockburn, S.-Y. Lin, C.-W. Shu: TVB Runge-Kutta Local Projection Discontinuous-Galerkin Finite Element Method For Conservation Laws III: One-Dimensional Systems. Journal of Computational Physics 84,90-113 (1989).

[16] B. Cockburn, S. Hou, C.-W. Shu: TVB Runge-Kutta Local Projection Discontinuous-Galerkin Finite Element Method For Conservation Laws IV: The Multidimensional Case. Mathematics of Computation, Vol.54,Nu.190,April 1990,p.545-581.

[17] B. Cockburn, C.-W. Shu: The P1-RKDG Method for Twodimensional Euler Equations of Gas Dynamics. ICASE-Report No.91-32,199.

[18] P. Colella, P. Woodward: The piecewise parabolic method for gas-dynamical simulation. J. of Comp. Physics 54 (1984), 174-201.

[19] R. Courant, K.O. Friedrichs, H. Lewy: Über die partiellen Diffentialgleichungen der mathematischen Physik. Math.Ann., 100 (1928) , 32-74

[20] M.C. Crandall, A. Majda: The method of fractional step for conservation laws. Num. Math. 34 (1980), 285-314.

[21] M.C. Crandall, A. Majda: Monotone difference approximation for scalar conservation laws. Math. of Comp. 34 (1980), 1-21.

[22] B. Despres: Entropy Inequality for High Order Discontinuous Galerkin Approximation of Euler Equations. Hyperbolic Problems: Theory, Numerics, Applications. Seventh International Conference in Zürich, Februar 1998.

[23] L.J. Durlofsky, S. Osher, B. Engquist: Triangle Based TVD Schemes for Hyperbolic Conservation Laws. J. Comp. Phys. 98 (1992) 64-73.

[24] B. Engquist, S. Osher: One sided difference approximations for nonlinear conservation laws. Math. of Comp. 36 (1981), 321-351.

[25] L. Fezoui, B. Stoufflet: A Class of Implizit Upwind Schemes for Euler Simulations with Unstructured Meshes. J. of Computational Physics 84 (1989) 174-206

[26] M. Geiben: Convergence of MUSCL-Type upwind finite volume schemes on unstructured grids. SFB 256, Preprint 278 (1991) , Bonn.

[27] M. Geiben, D. Kröner, M. Rokyta: A Lax-Wendroff type theorem for cell-centered finite volume schemes in 2-D. Preprint (1991)

[28] GRAPE, GRAphics Programming Environment: Reference Manual. SFB256

[29] T. Gerstner, M. Rumpf, U. Weikard: A Comparison of Error Indicators on Nested Grids for Multilevel Visualization. Report no. 24, SFB256 , Bonn 1999.

[30] T. Gerstner, M. Rumpf: Multiresolutional Parallel Isosurface Extraction based on Tetrahedral Bisection. Report no. 23, SFB256 , Bonn 1999.

[31] J. Jaffre, C. Johnson, A. Szepessy: Convergence of the Discontinuous Galerkin Finite Element Method for hyperbolic Conservation Laws. Math. Models and Methods in Applied Science, Vol.5 No.3 (1995) 367-386

[32] C. Johnson, J. Pitkäranta: An analysis of the discontinuous Galerkin method for a scalar hyperbolic equation. Math. Comput. 46 (1986), 1-26.

[33] C. Johnson, J. Saranen: Streamline diffusion methods for the incompressible Euler and Navier Stokes equations. Math. Comput. 47 (1986),1-18.

[34] C. Johnson, A. Szepessy: Convergence of a finite element method for a nonlinear hyperbolic conservation law. Math. Comput. 49 (1987), 427-444.

[35] C. Johnson, A. Szepessy, P. Hansbo: On the convergence of shock-capturing streamline diffusion finite element methods for hyperbolic conservation laws. Math. of Comp. 54 (1990), 107-129.

[36] T. Kato: The Cauchy problem for quasi-linear symmetric hyperbolic systems. Arch. Rat. Mech. Anal. 58, 181-205 (1975).

[37] D. Kröner: Directionally adapted upwind schemes in 2-D for the Euler equations. Priority Research Program, Results 1986-1988. Notes on numerical fluid mechanics II., Vol.25.

[38] D. Kröner: Numerical Schemes for Conservation Laws. Wiley Teubner, 1996.

[39] D. Kröner, M. Rokyta: Convergence of Upwind Finite Volume Schemes for Scalar Conservation Laws in 2-D. Prepint Nr.208 (1992)

[40] W. Krüger: The Application of Transport Theory to Visualization of 3-D Scalar Data Fields. Computers in Physics, Vol.5, No.4,1991.

[41] P. Lax, B. Wendroff: Systems of conservation laws. Comm. on Pure and Appl. Math. 13 (1960), 217-237.

[42] R.J. LeVeque: High Resolution Finite Volume Methods on Arbitrary Grids via Wave Propagation. J. Comp. Physics, 78 (1988), 36-63.

[43] R.J. LeVeque: Numerical Methods for Conservation Laws. Lectures in Mathematics, ETH Zürich. Birkhäuser Verlag.

[44] K.W. Morton: A finite volume scheme with shock fitting for the steady Euler equations. J. of Comp. Physics 80 (1989).

[45] S. Osher: Convergence of Generalized MUSCL Schemes. SIAM J. Numer. Anal. 22 (1985), 947-961.

[46] S. Osher, S. Chakravarthy.: High Resolution Schemes and the Entropy Condition. SIAM J. Numer. Anal. 21 (1984), 955-984.

[47] S. Osher, F. Solomon: Upwind Difference Schemes for Hyperbolic Systems of Conservation laws. Math. Comptut. 38 (1982) , 339-374

[48] R. Preis, R. Diekmann: The PARTY Partitioning - Library User Guide - Version 1.1. Institut für Mathematik und Computerwissenschaften, Universität Paderborn, 1996.

[49] M.C. Rivara: Algorithms for refining triangular grids suitable to adaptive and multigrid technique. Int. Journal for Num. Methods in Eng. 20,745-756 (1984).

[50] P.L. Roe: Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. J. of Comp. Physics 43 (1981), 357-372.

[51] M. Rumpf, A. Schmidt: GRAPE, GRAphics Programming Environment. Report 8, SFB 256, Bonn 1990.

[52] Sanders , Weiser : High Resolution Staggered Mesh. J. of Comp. Physics 101 (1992)

[53] R. Schwörer: Entwicklung und Parallelisierung von Finite-Volumen-Verfahren zur Lösung der kompressiblen Navier-Stokes-Gleichungen in 2-D. Institut für Angewandte Mathematik, Universität Freiburg, Diplomarbeit,1997.

[54] R.A. Shapiro: Adaptive finite element solution algorithm for the Euler equations. Notes on Numerical Fluid Mechanics, vol. 32, 1991.

[55] C.-W. Shu, S. Osher: Efficient Implementation of Essentially Non-oscillatory Shock-Capturing Schemes II. J. of Comp. Physics 83 (1989), 32-78

[56] J. Smoller: Shock waves and reaction-diffusion equations. Springer-Verlag, New York, 1983.

[57] G.A. Sod: A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws. J. of Comp. Physics 27 (1978), 1-31

[58] P. Solin: Three-dimensional Euler Equations and their Numerical Solution, Moving Particle Scheme for Grid Generation. Institut für Mathematik und Physik, Universität Prag Diplomarbeit, 1996.

[59] T. Sonar: On the Design of an Upwind Scheme for Compressible Flow on General Triangulations. Deutsche Forschungsanstalt für Luft- und Raumfahrt DLR, Institut für Theoretische Strömungsmechanik.

[60] D. Stalling, H.-C. Hege: Fast and Resolution Independent Line Integral Convolution. Proceedings SIGGRAPH '95, 1995.

[61] J.L. Steger, R.F. Warming: Flux vector splitting of the inviscid gasdynamic equations with application to finite difference methods. J. Comput. Phys. 40 (1981), 263-293.

[62] J. Stoer: Numerische Mathematik 1. Springer Verlag.

[63] G. Strang: On the construction and comparison of difference schemes. SIAM J. Numer. Anal. 5 (1968), 506-517.

[64] P.K. Sweby: High resolution Schemes using flux limiters for conservation laws. SIAM J. Numer. Anal. 21 (1984), 995 ff.

[65] A. Szepessy: Convergence of a shock capturing streamline diffusion finite element method for a scalar conservation law in two space dimensions. Math of Comp. 53 (1989), 527-545.

[66] J.J. van Wijk: Spot Noise – Texture Synthesis for Data Visualization. Computer Graphics, Volume 25, Number 4, 1991.

[67] J.J. van Wijk: Implicit Stream Surfaces. IEEE Visualization '93,245–252, 1993.

[68] J.J. van Wijk: Flow Visualization with Surface Particles. IEEE Computer Graphics and Applications Vol. 13, No.4, 1993 , pp.18–24.

[69] G. Vijayasundaram: Transonic flow simulations using an upstream centered scheme of Godunov in finite elements. J. of Comp. Physics 63 (1986), 416-433.

[70] R.F. Warming, R.M. Beam: On the construction and application of implicit factoredschemes for conservation laws. SIAM - AMS Proceedings, Volume 1, 1978.

[71] R.F. Warming, R.M. Beam, B.J. Hyett: Diagonalization and simultaneous symmetrization of the gas - dynamic matrices. Mathematics of Computation, Volume 29, number 132, October 1975, 1037-1045.

[72] P. Woodward, P. Colella: The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks. J. of Comp. Physics 54 (1984)