

## Masterarbeit

# Continuous Scatterplotting von Tensorfeldern in 3D

Niklas Teichmann

28. Februar 2019

**Zusammenfassung:** Die Visualisierung von Tensorfeldern ist trotz vieler Jahrzehnte der Forschung noch immer ein aktives Thema. Komplexe Strukturen innerhalb von Tensorfeldern stellen hohe Anforderungen an Visualisierungssoftware, um Nutzern die Interpretation zu erleichtern. Innerhalb dieser Arbeit wird eine Erweiterung für die Visualisierungssoftware ‚FAnToM‘ vorgestellt, die Techniken aus dem Direct Volume Rendering und dem Continuous Scatterplotting verwendet, um Invarianten von symmetrischen Tensoren zweiten Grades und ihre Verteilung innerhalb eines Datensatzes darzustellen. Besonderer Wert wird auf Interaktivität gelegt, um Nutzern die explorative Analyse der Daten zu ermöglichen.

Hiermit erkläre ich, die vorliegende wissenschaftliche Arbeit selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, d. \_\_\_\_\_  
Ort, Datum

Matrikelnummer: 2878372

Unterschrift

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Verwandte Arbeiten</b>	<b>3</b>
<b>3 Grundlagen</b>	<b>4</b>
3.1 Mathematische Grundlagen . . . . .	4
3.1.1 Urbild und Bild einer Funktion . . . . .	5
3.1.2 Einschränkung einer Funktion . . . . .	5
3.2 Surjektivität . . . . .	5
3.3 Injektivität . . . . .	5
3.3.1 Dirac Delta . . . . .	6
3.3.2 Volumen einer Teilmenge von $\mathbb{R}^n$ . . . . .	6
3.3.3 Lineare Abbildungen . . . . .	6
3.3.4 Vektorraum . . . . .	7
3.3.5 Dualraum . . . . .	7
3.3.6 Multilineare Funktionen . . . . .	8
3.3.7 Tensor . . . . .	8
3.3.8 Jacobi Matrix . . . . .	9
3.3.9 Determinante einer Matrix . . . . .	9
3.3.10 Determinante der Jacobimatrix . . . . .	9
3.3.11 Rang einer Matrix . . . . .	10
3.3.12 Spur einer Matrix . . . . .	10
3.3.13 Norm einer Matrix . . . . .	10
3.3.14 Deviator einer Matrix . . . . .	11
3.3.15 Eigenwerte und Eigenvektoren einer Matrix . . . . .	11
3.3.16 Fraktionale Anisotropie einer Matrix . . . . .	11
3.3.17 Modus einer Matrix . . . . .	12
3.3.18 Gradient . . . . .	13
3.3.19 Orthogonalität von Matrizen . . . . .	13
3.3.20 Matrixinvarianten . . . . .	13
3.3.21 Invariantensätze . . . . .	14
3.3.22 Kontinuierliche Menge . . . . .	18
3.3.23 Diskrete Menge . . . . .	18
3.3.24 Faltung . . . . .	18
3.3.25 Gaußsche Glockenkurve . . . . .	18
3.4 Mechanische Grundlagen . . . . .	19
3.4.1 Physikalisches Feld . . . . .	19
3.4.2 Mechanische Spannung . . . . .	19
3.4.3 Mechanische Verformung . . . . .	20
<b>4 Verwendete Verfahren und Technologien</b>	<b>20</b>
4.1 FAnToM . . . . .	20

4.2	OpenGL . . . . .	21
4.3	Qt . . . . .	23
4.4	Continuous Scatterplotting . . . . .	23
4.4.1	Scatterplotting . . . . .	23
4.4.2	Erklärung des Continuous Scatterplotting . . . . .	24
4.5	Volumetrische Daten . . . . .	29
4.6	Volumenvisualisierung . . . . .	30
4.6.1	Isoflächen . . . . .	30
4.6.2	Slicing . . . . .	32
4.6.3	Direct Volume Rendering . . . . .	34
<b>5</b>	<b>Direct Volume Rendering Verfahren</b>	<b>38</b>
5.1	Texture Slicing . . . . .	38
5.2	Raycasting . . . . .	40
5.3	Cell Projection . . . . .	42
5.4	Splatting . . . . .	44
5.5	Shear Warp . . . . .	45
5.6	Wahl des DVR-Verfahrens . . . . .	46
<b>6</b>	<b>Umsetzung</b>	<b>47</b>
6.1	Umsetzung des Continuous Scatterplotting . . . . .	48
6.2	Datenvorbereitung . . . . .	49
6.2.1	Die Vorbereitungssession . . . . .	49
6.2.2	Die Visualisierungssession . . . . .	50
6.3	Voxelisierung . . . . .	51
6.4	Berechnung der maximalen Dichte . . . . .	55
6.5	Umsetzung des Raycastings . . . . .	57
6.6	Die Interaktionswidgets . . . . .	59
6.6.1	quaderförmiges Interaktionswidget . . . . .	60
6.6.2	Zylindrisches Interaktionswidget . . . . .	62
6.7	Labeling . . . . .	64
6.8	Weitere Interaktionen . . . . .	64
6.9	Optionen des Algorithmus . . . . .	65
6.9.1	Gemeinsame Optionen . . . . .	66
6.9.2	Optionen des Objektrenderings . . . . .	67
6.9.3	Optionen des Invariantenraumrenderings . . . . .	67
6.10	Das Intervallfenster . . . . .	69
<b>7</b>	<b>Ergebnisse</b>	<b>69</b>
7.1	Single Point Load . . . . .	70
7.2	Metallscheibe . . . . .	72
7.3	Biegebalken . . . . .	77
7.4	Metallkomponente . . . . .	82
7.5	Fazit der Ergebnisse . . . . .	85

<b>8 Ausblick</b>	<b>87</b>
<b>9 Zusammenfassung</b>	<b>88</b>
<b>Literatur</b>	<b>I</b>
<b>10 Anhang</b>	<b>IV</b>
10.1 Anhang A . . . . .	IV

## 1 Einleitung

Tensorfelder sind eine sowohl in der Forschung als auch in der Praxis häufig vorkommende Art von Datensätzen. Tensoren stellen, vereinfacht ausgedrückt, mathematische Funktionen dar, die eine Menge von Vektoren auf einen skalaren Wert abbilden. Beispiele für Tensorfelder findet man in der Diffusions-Tensor-Bildgebung[4], welche die Diffusionsbewegung von Wasser in Gewebe wie dem Hirn untersucht, oder bei Verformungs-[17, S. 122] und Spannungstensoren[17, S. 154] in der Mechanik, durch die Richtungen, Stärken und Wirkungen von Kräften innerhalb von Objekten ausgedrückt werden. Ein häufig gewählter Ansatz, um die Interpretation von Tensordaten zu erleichtern, ist die Tensorfeldvisualisierung, ein Teilgebiet der wissenschaftlichen Visualisierung, das sich mit der Erzeugung von für Menschen verständlichen visuellen Repräsentationen von Tensorfeldern beschäftigt.

Clemens Fritzsch stellt in seiner Diplomarbeit[14] ein Verfahren zur Tensorfeldvisualisierung vor. Dabei werden Kennwerte der Tensoren, sogenannte Invarianten, berechnet und das Tensorfeld durch Continuous Scatterplotting[3] visualisiert. Invarianten werden üblicherweise zu Invariantensätzen gruppiert. Bei dreidimensionalen Tensoren zweiten Grades, wie sie in der Praxis häufig vorkommen, besteht ein solcher Invariantensatz immer aus drei Invarianten. Das von Fritzsch implementierte Continuous Scatterplotting bildet ein Tensorfeld in einen zweidimensionalen kartesischen Raum ab, in dem die Koordinate eines Punktes zwei der Invarianten des Tensors an diesem Punkt entsprechen. Räume, in denen die Koordinaten den Invarianten an den jeweiligen Punkten entsprechen, werden im Folgenden als ‚Invariantenräume‘ bezeichnet.

In vielen praktischen Anwendungen sind jedoch alle drei Invarianten eines Invariantensatzes und ihr Verhältnis zueinander von Interesse. Ziel der vorliegenden Arbeit ist es daher, das Verfahren von Fritzsch auf ein dreidimensionales Continuous Scatterplotting zu verallgemeinern. Dazu müssen folgende Schritte erfüllt werden:

Zunächst muss ein Verfahren implementiert werden, durch das ein dreidimensionales Continuous Scatterplotting in geringer Rechenzeit erzeugt werden kann. Aufgrund der Menge an Daten und der Komplexität des Continuous Scatterplotting selbst ist dies keine triviale Aufgabe. Als zweites muss eine Möglichkeit gefunden werden, das dreidimensionale Ergebnis des Continuous Scatterplotting darzustellen. Einen geeigneten Ansatz dazu bietet das Direct Volume Rendering. Da viele Direct Volume Rendering Verfahren existieren, müssen diese untersucht und miteinander verglichen werden, um das für die Problemstellung am besten geeignete Verfahren zu finden und zu implementieren. Wie sich aus der Arbeit von Fritzsch ergeben hat, kann es schwierig sein, die Form des Continuous Scatterplots zu interpretieren und Rückschlüsse auf das Feld im Ortsraum zu ziehen. Um die Interpretation zu erleichtern werden Interaktionen implementiert, die es ermöglichen, Teile der Darstellung im Invariantenraum auszuwählen und die entsprechenden Bereiche im Ortsraum anzuzeigen. Um Nutzern die explorative Analyse von Datensätzen zu ermöglichen, darf die Zeit, die benötigt wird, um die Auswirkungen von Interaktionen darzustellen, nur Bruchteile von Sekunden betragen. Zum Abschluss

wird die erzeugte Visualisierung mit bestehenden Verfahren zur Tensorfeldvisualisierung verglichen, um die Korrektheit des neuen Verfahrens zu gewährleisten sowie Vor- und Nachteile zu zeigen.

Dabei stellen sich eine Reihe von Herausforderungen, von denen einige im Folgenden genannt werden[14, 18]:

**1. Menge an Daten pro Tensor** Ein einzelner Tensor kann, abhängig von Grad und Dimension, beliebig viele Datenwerte umfassen. Selbst ein Tensor zweiten Grades und dritter Dimension, der durch eine  $3 \times 3$  Matrix dargestellt wird, besteht bereits aus neun Werten, für die eine visuelle Codierung gefunden werden muss.

**2. Menge an Daten pro Datensatz** Häufig enthalten Datensätze Tausende oder Millionen von Tensoren, was Ansprüche an die Skalierbarkeit der Visualisierung stellt. Zusätzlich bestehen relevante Merkmale der Datensätze oft nur aus einem geringen Anteil der Menge von Tensoren. Bei der Entwicklung der Visualisierung muss daher auch die Sichtbarkeit solch kleiner Merkmale sichergestellt werden.

**3. Fehlende Intuition** Tensoren beschreiben im Allgemeinen Abbildungen zwischen Skalaren, Vektoren und höheren Tensoren. Während bei niedrigen Rängen, wie Skalaren oder Vektoren, noch intuitive Interpretationen existieren, fällt es Menschen erheblich schwerer, Matrizen oder Tensoren höheren Grades zu interpretieren. Die Repräsentation eines Tensors muss daher gut durchdacht sein, um relevante Eigenschaften interpretierbar darzustellen.

**4. Domänspezifische Informationen** Abhängig von der jeweiligen Domäne können unterschiedliche Informationen über die vorliegenden Tensoren von Interesse sein. So gibt es Anwendungsfälle, in denen Bereichen mit hoher Isotropie besondere Bedeutung zugemessen wird, während solche Bereiche in einem anderen Kontext eher uninteressant sind[18, S. 4]. Eine Anwendung zu entwickeln, die über Domänen hinweg verwendbar ist, ist daher eine Herausforderung.

Die vorliegende Arbeit beschränkt sich aus praktischen Gründen auf symmetrische Tensoren zweiten Grades im dreidimensionalen Raum, da diese Art von Tensoren in den meisten Anwendungsfällen verwendet wird.

Der Rest dieser Arbeit gliedert sich wie folgt: Zunächst werden im Kapitel 2 [Verwandte Arbeiten](#) Vor- und Nachteile von bekannten Verfahren zur Tensorfeldvisualisierung erläutert. Danach werden im Kapitel 3 [Grundlagen](#) die verwendeten Definitionen und Grundlagen aus der Mathematik und Kontinuumsmechanik vorgestellt. Als Nächstes werden in 4 [Verwendete Verfahren und Technologien](#) die benutzten Programmschnittstellen und Visualisierungstechniken erläutert. Insbesondere wird dort auch auf FAnToM als

Softwaregrundlage eingegangen. Eine Reihe von verbreiteten Direct Volume Rendering Verfahren wird in Kapitel 5 Direct Volume Rendering Verfahren vorgestellt und in Hinsicht auf die Eignung für das vorliegende Problem verglichen. Kapitel 6 Umsetzung beschreibt die konkrete Umsetzung der FAnToM-Erweiterung mit allen implementierten Funktionen. Nachfolgend wird in Kapitel 7 Ergebnisse die entwickelte Erweiterung exemplarisch auf einige Datensätze angewendet, und die Ergebnisse diskutiert. Zum Abschluss werden in Kapitel 8 Zusammenfassung die Arbeit zusammengefasst und in 9 Ausblick im Verlauf der Arbeit neu aufgetretene Problemstellungen sowie mögliche Herangehensweisen zu deren Lösung erörtert.

## 2 Verwandte Arbeiten

Es existiert bereits eine große Anzahl von Verfahren, die Visualisierungen von Tensorfeldern erzeugen. Nachfolgend werden, ohne Anspruch auf Vollständigkeit, einige der wichtigsten genannt und kurz beschrieben.

Eine relativ einfache Darstellung eines Tensorfeldes besteht darin, die einzelnen Komponenten eines Tensor zweiten Grades als Skalarfelder aufzufassen und als Grauwertbild zu zeichnen. Dabei ist der Grauwert an einem Datenpunkt abhängig vom Verhältnis des Wertes der Komponente des Tensors zu dem höchsten Wert dieser Komponente im Datensatz. Die früheste von uns gefundene Erwähnung dieses Verfahrens stammt aus einem Paper von Kindlmann und Weinstein aus dem Jahre 1999 [22], in dem es jedoch als weder neu noch besonders intuitiv beschrieben wird.

Die wohl am weitesten verbreitete Art von Tensorvisualisierungen sind die glyphenbasierten Verfahren. Diese Verfahren beschränken sich auf Tensoren zweiten Grades im dreidimensionalen Raum, die als  $3 \times 3$  Matrizen dargestellt werden können. Das umfasst einen großen Teil der Tensordaten aus Mechanik und Medizin. Eine Glyphe ist hierbei ein kleines Bild eines grafischen Primitive, z.B. eines Ellipsoiden, Kuboiden oder Superquadrics[21], das einen Tensor darstellt. Die Form der Primitives ist dabei abhängig von den Eigenwerten des jeweiligen, als Matrix dargestellten Tensors an dieser Stelle. Indem an jedem Datenpunkt eine solche Glyphe gezeichnet wird, erhält der Nutzer ein Bild von der Verteilung und Struktur der Tensoren im Datensatz. Glyphenbasierte Verfahren sind besonders in der Medizin beliebt, da sie leicht Rückschlüsse auf die Richtung von Nerven- und Muskelfasern zulassen.

Hyperstreamlines[9] bilden ein Analogon zu den Stromlinien bei Vektorfeldern. Als Eingabedaten sind nur Felder von reellen, symmetrischen, dreidimensionalen Tensoren zweiten Grades mit nichtnegativen Eigenwerten zugelassen, da so sichergestellt wird, dass die Eigenwerte ganzzahlig und größer 0 sind, sowie dass die Eigenvektoren paarweise orthogonal zueinander sind. Das Verfahren zeichnet ausgehend von festgelegten Punkten Tuben durch das Tensorfeld. Die Mittellinien dieser Tuben entsprechen dabei Stromlinien, deren Richtung vom Eigenvektor mit dem höchsten Eigenwert abhängt. Der

Durchschnitt durch den Schlauch orthogonal zur Mittellinie ist stets eine Ellipse, deren Halbachsen den zwei kleineren Eigenwerten entsprechen. Damit keine Informationen über den größten Eigenwert verloren gehen, werden Stücke der Tuben abhängig von dessen Betrag eigenfärbt.

Auch die Diplomarbeit von Clemens Fritzsch[14] muss erwähnt werden, auf der die vorliegende Arbeit direkt aufbaut. Fritzsch verwendet Methoden des Continuous Scatterplotting, um Invarianten von zweidimensionalen, symmetrischen Tensoren zweiten Grades darzustellen. Dabei beschränkt er sich jedoch auf zwei der drei Invarianten in jedem Invariantensatz, um einen zweidimensionalen Scatterplot zu erzeugen. Optional können sogenannte ‚Extremalkurven‘ oder ‚Extremalflächen‘ eingezeichnet werden, die die Positionen kritischer Punkte innerhalb der Darstellung anzeigen. Die vorliegende Arbeit erweitert den Ansatz von Fritzsch auf vollständige Invariantensätze, indem eine dreidimensionale Darstellung erzeugt wird. Das Einzeichnen von Extremalkurven und -flächen wird jedoch nicht weiter verfolgt.

Ein zu der vorliegenden Arbeit sehr ähnlicher Ansatz wurde von Raith et al. [33] verfolgt. Dabei werden Tensorfelder in den Invariantenraum überführt, in dem die Koordinaten eines Punktes den Invarianten des Tensors an diesem Punkt entsprechen. Das Feld wird im Invarianten- und Ortsraum als Gitternetz dargestellt. Durch Interaktionen können geometrische Objekte innerhalb des Invariantenraumes platziert werden. Wenn eine Seitenfläche eines dieser Objekte eine Zelle des Gitternetzes im Invariantenraum schneidet, wird der Verlauf des Schnitts im Ortsraum berechnet und angezeigt. Wenn dabei die Schnittfläche parallel zur xy-, xz- oder yz-Ebene verläuft, entspricht der Verlauf des Schnitts einer Isofläche der drei Invarianten im Ortsraum. Die wesentlichen Unterschiede zur vorliegenden Arbeit bestehen aus zwei Aspekten. Zum einen bringt die Darstellung des Feldes als Gitternetz Nachteile mit sich. Die Interpretation der Form des Feldes im dreidimensionalen Raum ist erschwert und Informationen über innere Strukturen sind schwer oder teilweise gar nicht ablesbar. Zum anderen benötigt die Interaktion bei Raith et al. bei manchen Datensätzen mehrere Sekunden, was explorative Analysen erschwert. Die vorliegende Arbeit versucht diese Probleme zu lösen. Die Interpretation der Form des Feldes und innerer Strukturen wird durch die Darstellung als Direct Volume Rendering erleichtert. Zusätzlich wird versucht, die von Interaktion verursachte Rechenzeit so gering wie möglich zu halten, um Nutzern die Exploration des Datensatzes zu ermöglichen.

## 3 Grundlagen

### 3.1 Mathematische Grundlagen

In diesem Teil der Arbeit werden mathematische Grundlagen zu Tensoren, Feldern und Invarianten erläutert. Insbesondere für die Definition von Tensoren sind Vorkenntnisse nötig, die ebenfalls erklärt werden.

Der Großteil der verwendeten Formeln und Definitionen stammt aus der Arbeit von R. M. Bowen und C. C. Wang[5].

### 3.1.1 Urbild und Bild einer Funktion

Zu jeder Funktion  $f : A \rightarrow B$ , die Objekten aus der Menge  $A$  Objekte aus der Menge  $B$  zuordnet, lässt sich das Bild einer Menge  $A' \subset A$  als

$$f(A') : \{b \in B | \exists a \in A' : f(a) = b\}, \quad (1)$$

und das Urbild einer Menge  $B' \subset B$  als

$$f^{-1}(B') : \{a \in A | \exists b \in B' : f(a) = b\} \quad (2)$$

bestimmen.  $f(A)$  wird hierbei die Bildfunktion,  $f^{-1}(B)$  die Urbildfunktion genannt.

### 3.1.2 Einschränkung einer Funktion

Gegeben sei eine Funktion  $f : A \rightarrow B$ . Dann ist  $f|_{A'} : A' \rightarrow B$ , die Einschränkung von  $f$  auf die Menge  $A' \subset A$ , definiert als

$$f|_{A'}(a) = f(a) \text{ für alle } a \in A'. \quad (3)$$

Für alle  $a \in A, a \notin A'$  ist  $f|_{A'}$  nicht definiert.

## 3.2 Surjektivität

Eine Funktion  $f : A \rightarrow B$  wird als injektiv bezeichnet, wenn gilt

$$\forall b \in B \exists a \in A : f(a) = b. \quad (4)$$

## 3.3 Injektivität

Eine Funktion  $f : A \rightarrow B$  wird als injektiv bezeichnet, für alle  $b_1, b_2 \in B$  gilt

$$f(b_1) = f(b_2) \Rightarrow b_1 = b_2. \quad (5)$$

### 3.3.1 Dirac Delta

Das Dirac Delta (auch Delta Distribution genannt) ist eine stetig lineare Abbildung  $\delta$ , die einer beliebig oft differenzierbaren Funktion  $f$ , die auf  $\mathbb{R}^n$  oder  $\mathbb{C}^n$  definiert ist, den Wert an der Stelle 0 zuordnet.

Formaler ausgedrückt bildet  $d$  Funktionen aus  $C^\infty(\Omega)$ , dem Raum der beliebig oft differenzierbaren Funktionen über  $\Omega \subset \mathbb{R}^n$  oder  $\Omega \subset \mathbb{C}^n$  mit  $0 \in \Omega$ , auf ihren Wert an der Stelle 0 ab. Mit 0 sind hierbei auch die entsprechenden höherdimensionale Nullvektoren aus  $\mathbb{R}^n$  und  $\mathbb{C}^n$  gemeint.

Das Dirac Delta ist formal gesehen keine Funktion. Es kann jedoch mithilfe des Lebesgue Integrals definiert werden. Dies würde hier jedoch zu weit führen. Genaueres kann in der Arbeit von Kusse et al. [25, S. 100 ff.] nachgelesen werden.

Auf diese Art definiert hat das Dirac Delta zwei wichtige Eigenschaften, die auch in der vorliegenden Arbeit verwendet werden:

$$\delta(x) = \begin{cases} +\infty, & \text{wenn } x = 0 \\ 0, & \text{sonst} \end{cases} \quad (6)$$

$$\int_{-\infty}^{+\infty} \delta(x) dx = 1 \quad (7)$$

### 3.3.2 Volumen einer Teilmenge von $\mathbb{R}^n$

Das  $n$ -dimensionale Volumen einer Menge  $Vol(A)$ ,  $A \subset \mathbb{R}^n$  wird über das Lebesgue-Stieltjes Maß definiert [24]. Intuitiv entspricht das Lebesgue-Stieltjes Maß in  $\mathbb{R}$  der Länge, in  $\mathbb{R}^2$  dem Flächeninhalt und in  $\mathbb{R}^3$  dem Volumen.

Falls für eine Menge  $A$  gilt  $Vol(A) = 0$ , so bezeichnet man diese als Nullmenge.

### 3.3.3 Lineare Abbildungen

Seien  $V, U$  zwei Vektorräume über demselben Körper  $K$ . Eine lineare Abbildung  $\varphi : V \rightarrow U$  ist eine Funktion, sodass für alle  $\lambda \in K$ ,  $v \in V$  und  $u \in U$  gilt [5, S. 85]:

$$\varphi(u + v) = \varphi(u) + \varphi(v) \quad (8)$$

$$\varphi(\lambda v) = \lambda \cdot \varphi(v) \quad (9)$$

### 3.3.4 Vektorraum

Sei  $V$  eine Menge,  $(K, +, \cdot)$  ein Körper,  $\oplus$  eine Abbildung  $V \times V \rightarrow V$  Vektoraddition und  $\odot$  eine Abbildung  $K \times V \rightarrow V$  genannt Skalarmultiplikation.  $(V, \oplus, \odot)$  wird dann als Vektorraum bezeichnet, wenn zusätzlich für die Vektoraddition die folgenden Eigenschaften gelten:

$$\forall u, v, w \in V : u \oplus (v \oplus w) = (u \oplus v) \oplus w \quad (\text{Assoziativit\"at}) \quad (10)$$

$$\exists 0_V \in V. \forall v \in V : v \oplus 0_V = 0_V \oplus v = 0_V \quad (\text{neutrales Element}) \quad (11)$$

$$\forall v_+ \in V. \exists v_- \in V : v_+ \oplus v_- = v_- \oplus v_+ = 0_V \quad (\text{inverse Elemente}) \quad (12)$$

$$\forall u, v \in V : u \oplus v \quad (\text{Kommutativit\"at}) \quad (13)$$

Und für die Skalarmultiplikation müssen zusätzlich folgende Eigenschaften gelten:

$$\forall k \in K. \forall u, v \in V : k \odot (u \oplus v) = (k \odot u) \oplus (k \odot v) \quad (\text{Distributivit\"at}) \quad (14)$$

$$\forall k, l \in K. \forall v \in V : (k + l) \odot v = (k \odot v) \oplus (l \odot v) \quad (\text{Distributivit\"at}) \quad (15)$$

$$\forall k, l \in K. \forall v \in V : (k \cdot l) \odot v = k \odot (l \odot v) \quad (\text{Assoziativit\"at}) \quad (16)$$

$$\exists k_1 \in K. \forall v \in V : k_1 \odot v = v \quad (\text{Einselement}) \quad (17)$$

### 3.3.5 Dualraum

Gegeben seien ein  $n$ -dimensionaler Vektorraum  $V$  und sein zugrundeliegender Körper  $K$ . Die lineare Abbildung  $\varphi : V \rightarrow K$  eines Vektors  $v = (v_1, \dots, v_n) \in V$  auf den skalaren Wert  $k \in K$  hat dann die Form

$$\varphi(v_1, \dots, v_n) = \varphi_1 \cdot v_1 + \dots + \varphi_n \cdot v_n = k. \quad (18)$$

Die  $\varphi_1, \dots, \varphi_n$  können wiederum als Vektor geschrieben werden. Der durch die Menge aller  $\varphi$  über  $V$  erzeugte,  $n$ -dimensionale Vektorraum  $V^*$  wird Dualraum genannt [5, S. 203].

Vektoren aus  $V$  werden als kovariant bezeichnet, Vektoren aus  $V^*$  als kontravariant [5, S. 205].

### 3.3.6 Multilineare Funktionen

Multilineare Funktionen über Vektorräumen sind Funktionen der Form  $\varphi : V_1 \times \cdots \times V_n \rightarrow K$ , wobei jedes  $V_i$  ein Vektorraum über  $K$  ist, und zusätzlich

$$\varphi(\lambda \cdot v_1 + \mu \cdot v'_1, \dots, v_n) = \lambda \cdot \varphi(v_1, \dots, v_n) + \mu \cdot \varphi(v'_1, v_2, \dots, v_n), \quad (19)$$

mit  $\lambda, \mu \in K$ ,  $v_i, v'_i \in V_i$  gilt (für jede weitere Variable analog). Intuitiv bedeutet das, dass  $\varphi$  linear in jeder Variable ist [5, S. 204, 218].

### 3.3.7 Tensor

Multilineare Funktionen der Form  $T : V^* \times \cdots \times V^* \times V \times \cdots \times V \rightarrow K$ , wobei  $V$  ein Vektorraum über  $K$  und  $V^*$  sein Dualraum ist, werden als Tensoren bezeichnet [5, S. 218]. Der Grad des Tensors ist definiert als die Anzahl an Variablen der Funktion. Die Tensoren über  $V$  bilden wiederum einen Vektorraum [5, S. 220]. Durch diese Definition ist ein Tensor immer invariant zur Basis des Vektorraums seiner Variablen. Egal in welche Basis er umgerechnet wird, er drückt stets dasselbe aus.

In der vorliegenden Arbeit werden ausschließlich Tensoren zweiten Grades in kartesischen, dreidimensionalen Koordinaten verwendet. Dabei ist zu beachten, dass in kartesischen Koordinaten die Basis eines Vektorraumes  $V$  und seines Dualraumes  $V^*$  die gleiche Darstellung haben, also  $V$  und  $V^*$  austauschbar sind. Wenn in einen Tensor zweiten Grades  $T$  die Basisvektoren  $e_{1,\dots,d}$  des zugrundeliegenden Vektorraumes  $V$  bzw.  $V^*$  der Dimension  $d$  in jeder möglichen Kombination eingesetzt werden, ergeben sich für  $1 \leq i, j \leq d$  folgende Komponenten:

$$c_{i,j} = \sum_{i=1}^d \sum_{j=1}^d T(e_i, e_j) \quad (20)$$

Diese basisabhängige Darstellung des Tensors bildet eine  $d \times d$  Matrix. Alle in der vorliegenden Arbeit verwendeten Tensoren liegen in dieser Form vor. Da durch das Matrix-Vektor-Produkt einer Matrix  $m$  mit einem Vektor  $v$

$$m \cdot v = u \quad (21)$$

eine Abbildung auf einen Vektor  $u$  desselben Vektorraumes wie  $v$  ausgedrückt werden kann, lassen sich mithilfe von Tensoren Abbildungen zwischen Vektorräumen unabhängig von der Basis des Raumes formulieren.

### 3.3.8 Jacobi Matrix

Die Jacobi Matrix  $J_f$  einer differenzierbaren Abbildung  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  ist eine  $m \times n$  Matrix, deren Komponenten die partiellen ersten Ableitungen von  $f$  sind. Formal geschrieben gilt also für die Koordinaten des Urbilds  $x_1, \dots, x_n$  und Abbildungen  $f_1, \dots, f_n$  der einzelnen Komponenten

$$J_f(a) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(a) & \dots & \frac{\partial f_1}{\partial x_n}(a) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(a) & \dots & \frac{\partial f_m}{\partial x_n}(a) \end{pmatrix}. \quad (22)$$

Sie entspricht damit der ersten Ableitung in der mehrdimensionalen Analysis. Die Determinante der Jacobi-Matrix  $\det(J_f)$  wird auch als Funktionaldeterminante bezeichnet, und beschreibt einige Eigenschaften der Funktion  $f$ .

### 3.3.9 Determinante einer Matrix

Die Determinante einer Matrix ist eine aus den Einträgen der Matrix berechnete Kennzahl. Für alle  $m \times n$  Matrizen mit  $n \neq m$  ist die Determinante 0. Für quadratische Matrizen  $m \in \mathbb{K}^{n \times n}$  über dem Körper  $\mathbb{K}$  ist die Funktion  $\det : \mathbb{K}^{n \times n} \rightarrow \mathbb{K}$ , die Determinante von  $m$  bestimmt, durch die Leibniz-Formel definiert:

$$\det(m) = \sum_{\sigma \in S_n} \left( sgn(\sigma) \prod_{i=1}^n m_{i,\sigma(i)} \right) \quad (23)$$

Dabei ist  $S_n$  die Menge aller Permutationen einer Menge mit  $n$  Elementen,  $sgn(\sigma)$  das Signum der Permutation  $\sigma$  und  $m_{ij}$  der Eintrag in der i-ten Zeile und j-ten Spalte der Matrix.

### 3.3.10 Determinante der Jacobimatrix

Die Determinante einer Matrix hat eine Reihe von interessanten Eigenschaften. Besonders relevant für die vorliegende Arbeit ist die folgende:

Sei  $|\det(J_f)|$  die Determinante der Jacobi-Matrix der Funktion  $f$  an einem Punkt  $p$ . Die Determinante kann dann als Wert der Expansion bzw. des Schrumpfens der Funktion in der Nähe von  $p$  aufgefasst werden. Für eine lineare Funktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ , deren Funktionaldeterminante in jedem Punkt  $p_n \in \mathbb{R}$  gleich ist, bedeutet das mit dem  $n$ -dimensionalen euklidischen Abstand  $\|p_1, p_2\|_n$

$$\|f(p_1), f(p_2)\|_n = \det(J_f) \cdot \|p_1, p_2\|_n. \quad (24)$$

Indem man das Lebesgue-Stieltjes Maß in  $\mathbb{R}^n$  mittels des euklidischen Abstands definiert, lassen sich so Volumenänderungen ausdrücken.

### 3.3.11 Rang einer Matrix

Der Zeilenraum einer Matrix ist der Raum, der aus Linearkombinationen ihrer Zeilenvektoren aufgespannt wird. Die Dimension des Zeilenraumes ist gleich der Anzahl linear unabhängiger Zeilenvektoren, und wird als Zeilenrang der Matrix bezeichnet. Analog lässt sich der Spaltenrang einer Matrix definieren. Es lässt sich zeigen, dass Zeilen- und Spaltenrang einer Matrix immer gleich sind und deshalb kurz als Rang  $\text{rang}(M)$  der Matrix  $M$  bezeichnet werden.

### 3.3.12 Spur einer Matrix

Die Spur (trace) einer  $n \times n$  Matrix  $A$  mit Komponenten  $a_{ij}$  mit  $1 \leq i, j \leq n$  ist definiert als

$$\text{tr}(A) = \sum_{i=1}^n a_{ii}, \quad (25)$$

also die Summe aller Elemente in der Hauptdiagonale. Eine wichtige Eigenschaft der Spur ist, dass sie bei der Überführung einer Matrix in eine andere Basis gleich bleibt (siehe auch Abschnitt 3.3.20).

### 3.3.13 Norm einer Matrix

Eine Norm ist eine Abbildung  $f : V \rightarrow \mathbb{R}$  eines Vektorraumes  $V$  über dem Körper  $K$  auf die reellen Zahlen, die folgende Bedingungen erfüllt:

$$f(kv) = |k|f(v) \quad (\text{Absolute Homogenität}) \quad (26)$$

$$f(u + v) \leq f(u) + f(v) \quad (\text{Erfüllung der Dreiecksungleichung}) \quad (27)$$

$$f(v) = 0 \iff v = 0 \text{ ist der Nullvektor} \quad (\text{Definitheit}) \quad (28)$$

mit  $k \in K$ ,  $u, v \in V$ .

Eine in kartesischen Koordinaten häufig eingesetzte Norm ist die euklidische Norm. Diese ist auf dem Vektorraum aller  $m \times n$  Matrizen  $K^{m \times n}$  mit  $A \in K^{m \times n}$  definiert als

$$\text{norm}(A) = \sqrt{\text{tr}(AA^T)}. \quad (29)$$

Die euklidische Norm wird auch als ‚Frobeniusnorm‘ bezeichnet.

### 3.3.14 Deviator einer Matrix

Eine Matrix  $A$  kann wie folgt in ihren isotropen Anteil  $\bar{A}$  und ihren anisotropen Anteil  $\tilde{A}$  zerlegt werden:

$$\tilde{A} = A - \bar{A} \quad (30)$$

$\tilde{A}$  wird auch als Deviator von  $A$  bezeichnet. Bei einer  $3 \times 3$  Matrix ergibt sich  $\bar{A}$  als

$$\bar{A} = \frac{1}{3} \text{tr}(A) I, \quad (31)$$

wobei  $I$  die Matrixdarstellung des Einheitstensors ist, also

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (32)$$

### 3.3.15 Eigenwerte und Eigenvektoren einer Matrix

Eigenvektoren  $v_i$ ,  $1 \leq i \leq n$  einer  $n \times n$  Matrix  $M$  sind vom Nullvektor verschiedene Vektoren, für die gilt

$$M \cdot v_i = \lambda_i v_i. \quad (33)$$

Intuitiv sind die Eigenvektoren dadurch definiert, dass sich durch Multiplikation mit  $M$  ihre Richtung nicht verändert. Die zugehörigen  $\lambda_i$  werden als Eigenwerte bezeichnet. Falls der Rang einer  $n \times n$  Matrix  $M$  kleiner ist als  $n$ , so hat diese Matrix  $n - \text{rang}(M)$  Eigenwerte, die 0 sind.

### 3.3.16 Fraktionale Anisotropie einer Matrix

Die fraktionale Anisotropie  $\text{FA}(M)$  einer Matrix, mit  $\bar{\lambda}$  als Mittelwert der Eigenwerte, ist definiert als

$$\text{FA}(M) = \sqrt{\frac{3((\lambda_1 - \bar{\lambda})^2 + (\lambda_2 - \bar{\lambda})^2 + (\lambda_3 - \bar{\lambda})^2)}{2(\lambda_1^2 + \lambda_2^2 + \lambda_3^2)}}. \quad (34)$$

Sie entspricht also der Standardabweichung der Eigenwerte, dividiert durch den Mittelwert ihrer Quadrate. Dadurch wird die Standardabweichung auf das Intervall  $[0; 1]$  normiert. Bei Matrizen mit hoher FA (nahe 1) ist ein Eigenwert um ein Vielfaches größer als die

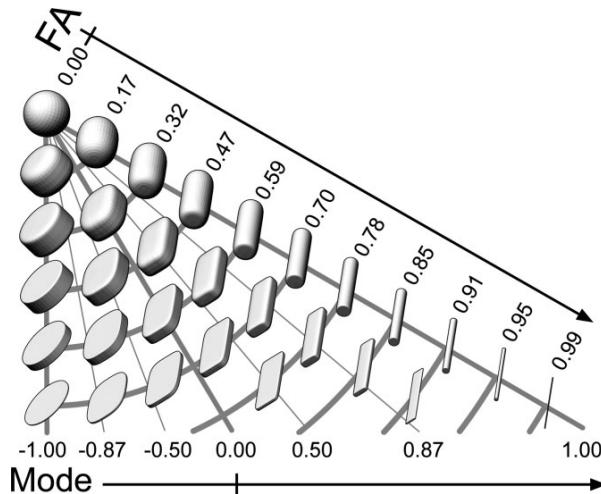


Abbildung 1: Darstellung der fraktionale Anisotropie und des Modus von Matrizen in Form von Superquadrics[21]. Die fraktionale Anisotropie nimmt mit größer werdender Entfernung zum obersten linken Superquadric zu. Der Modus des Deviators wird abhängig vom Winkel dargestellt, wobei er links -1 beträgt und rechts 1. Entnommen aus [11, S. 140].

anderen beiden. Hohe FA tritt in der Kontinuumsmechanik z.B. in Bereichen auf, in denen das Material in die Richtung eines Eigenvektors auseinandergezogen wird, während es in Richtung der anderen Eigenvektoren gleich bleibt oder sogar schrumpft. Eine niedrige FA dagegen drückt aus, dass die Eigenwerte etwa die gleichen Werte annehmen. In der Kontinuumsmechanik ist dies beispielsweise in Bereichen isoper Verformung, also gleichmäßiger Ausdehnung / gleichmäßigem Schrumpfen in alle Richtungen, der Fall.

### 3.3.17 Modus einer Matrix

Der Verformungs-Modus [7] einer Matrix  $A$ , im Folgenden kurz Modus genannt, ist definiert als

$$\text{mode}(A) = 3\sqrt{6} \det(A \setminus \text{norm}(A)). \quad (35)$$

Im Folgenden wird meistens der Modus des Deviators von  $A$  verwendet.

Der Modus liegt im Intervall  $[-1; 1]$  und drückt das Verhältnis der Eigenwerte der Matrix zueinander aus:

- $\text{mode}(A) = 1$ : ein hoher, zwei gleiche niedrige Eigenwerte; lineare Anisotropie
- $\text{mode}(A) = 0$ : ein hoher, ein niedriger und ein mittlerer Eigenwert; Orthotropie
- $\text{mode}(A) = -1$ : zwei gleiche hohe, ein niedriger Eigenwert: planare Anisotropie

Um die Intuition hinter Modus und fraktionaler Anisotropie zu verdeutlichen, sind in Abb. 1 Superquadrics von Matrizen unterschiedlicher Modi dargestellt. Insbesondere soll damit gezeigt werden, dass der Modus nicht von der Größe der Eigenwerte abhängt, sondern von deren Verteilung.

### 3.3.18 Gradient

Der Gradient einer skalaren Funktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  über einem kartesischen Koordinatensystem ist definiert als

$$\text{grad}(f) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}, \quad (36)$$

also als Vektor aller partiellen Ableitungen in die Richtungen  $x_i$ . Analog ist der Gradient einer Skalarfunktion  $g : K^{m \times n} \rightarrow \mathbb{R}$ , wobei  $K^{m \times n}$  der Raum aller  $m \times n$  Matrizen ist, definiert als

$$\text{grad}(g) = \begin{pmatrix} \frac{\partial f}{\partial a_{11}} & \cdots & \frac{\partial f}{\partial a_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial a_{m1}} & \cdots & \frac{\partial f}{\partial a_{mn}} \end{pmatrix}, \quad (37)$$

wobei  $a_{ij}$  die Komponenten der Matrizen sind.

### 3.3.19 Orthogonalität von Matrizen

Zwei Matrizen  $U, V$  werden als orthogonal zueinander bezeichnet, wenn [11] gilt:

$$\text{tr}(U, V^T) = 0 \quad (38)$$

### 3.3.20 Matrixinvarianten

Als Invarianten werden zu mathematischen Objekten zugeordnete Größen bezeichnet, die invariant gegenüber der Anwendung bestimmter Transformationen auf die Objekte sind. Invarianten eines Tensors in Matrixdarstellung, im Folgenden kurz Invarianten genannt, sind beispielsweise Größen, die sich unabhängig von der Wahl der Basis der Matrix nicht verändern[11]. Eine Invariante ist somit eine Funktion  $\Psi : M \rightarrow A$  die Objekten aus dem Vektorraum aller Matrizen  $M$  Objekte aus der Menge  $A$  zuordnet. In der Praxis wird für  $A$  meistens  $\mathbb{R}$  gewählt.

Da sich die vorliegende Arbeit überwiegend mit symmetrischen, dreidimensionalen Tensoren zweiten Grades und ausschließlich mit orthogonalen Transformationen beschäftigt, beschränkt sich auch die Betrachtung der Invarianten im Folgenden auf diese Art von Tensoren. Dabei gilt für solche Tensoren insbesondere, dass die Invarianten eines Tensors  $T$  durch seine Eigenwerte vollständig charakterisiert werden. Invarianten sind in diesem Spezialfall also auch als Funktion  $\Psi : \mathbb{R}^d \rightarrow \mathbb{R}$  der Form

$$\Psi : \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \rightarrow \mathbb{R} \quad (39)$$

darstellbar, wobei  $\lambda_1, \lambda_2, \lambda_3$  die Eigenwerte von  $T$  sind. Dies stimmt mit der Betrachtungsweise von Zobel und Scheuermann [41] überein.

### 3.3.21 Invariantensätze

Mengen von Invarianten werden als Invariantensätze bezeichnet. Matrixinvarianten  $\Psi_1$  und  $\Psi_2$  werden als orthogonal zueinander bezeichnet, wenn ihre Gradienten für jede mögliche Eingabematrix orthogonal sind. Die genauen Definitionen und Berechnungen dazu sind in den Arbeiten von Ennis, Kindlmann et al. [11] nachzulesen. Da Gradienten von skalarwertigen Funktionen auf Matrizen wiederum Matrizen sind [11, S. 137], genügt es zu zeigen, dass diese orthogonal zueinander sind. Invariantensätze, deren Elemente paarweise orthogonal sind, werden als orthogonale Invariantensätze bezeichnet.

Für  $3 \times 3$  Matrizen enthalten alle orthogonalen Invariantensätze höchstens drei Invarianten. In der Praxis spielt eine Vielzahl solcher Invariantensätze eine Rolle, von denen im Folgenden einige erläutert werden.

Wenn die zu einem Invariantensatz gehörenden Invarianten einer Matrix  $M$  berechnet wurden, so stellen diese eine Repräsentation der basisunabhängigen Eigenschaften von  $M$  dar. In dieser Arbeit wurden ausschließlich symmetrischen  $3 \times 3$  Matrizen visualisiert, die aus bis zu sechs nicht-doppelten Werten bestehen können. Die Beschreibung dieser sechs Werte durch die drei Invarianten ist nicht ohne Informationsverlust möglich. Die verlorenen drei Werte an Information sind jedoch ausschließlich Informationen über die Richtung der Eigenvektoren, was in vielen Anwendungsfällen nicht von Interesse ist.

Welcher der Invariantensätze für einen bestimmten Datensatz verwendet wird, kann abhängig von der Domäne ausgesucht werden. Die Wahl ist natürlich abhängig von den schon genannten Interpretationen der Invarianten in den jeweiligen Domänen.

**Die Eigenwerte** Die Eigenwerte bilden einen orthogonalen Invariantensatz. Sie sind insbesondere in der Medizin sehr beliebt, da hohe Eigenwerte von Diffusionstensoren auf die Bewegungsrichtung von Molekülen in Gewebe schließen lassen.

**Der I-Invariantensatz** Das charakteristische Polynom  $\chi$  einer  $3 \times 3$  Matrix  $A$  hat die Form

$$\begin{aligned}\chi_A(a) &= \det(a \cdot I - A) \\ \chi_A(a) &= -a^3 + I_1 \cdot a^2 - I_2 \cdot a + I_3,\end{aligned}\tag{40}$$

wobei  $\lambda$  ein Element aus dem Körper von  $A$  und  $I$  die dreidimensionale Einheitsmatrix ist. Es wird häufig verwendet, um die Eigenwerte von Matrizen zu bestimmen, da diese den Nullstellen des Polynoms entsprechen.

Eine weitere Eigenschaft ist, dass die Parameter  $I_1, I_2, I_3$  einen Invariantensatz darstellen. Wegen der Wichtigkeit des charakteristischen Polynoms werden sie häufig als ‚Hauptinvarianten‘ bezeichnet. Alternativ können sie auch berechnet werden als

- $I_1(A) = \text{tr}(A)$  (Spur von  $A$ )
- $I_2(A) = \frac{1}{2}(\text{tr}(A)^2 - \text{tr}(A^2))$  (Summe der Hauptminoren von  $A$ )
- $I_3(A) = \det(A)$  (Determinante von  $A$ )

Der I-Invariantensatz ist jedoch nicht orthogonal. Die Beweise oder Wiederlegungen der Orthogonalitätseigenschaft aller hier vorgestellter Invariantensätze sind in [11, S. 144] aufgeführt.

Während in der Mechanik für  $I_1$  eine Interpretation als Maß für den isotropen Anteil des Tensors existiert, fehlen eindeutige Interpretationen für  $I_2$  und  $I_3$ . Es gibt zwar einen Zusammenhang zwischen  $I_2$  und dem deviatorischen Anteil des Tensors, dieser ist jedoch nicht eindeutig genug um aus hohem  $I_2$  auf hohe Anisotropie schließen zu können. Für  $I_3$  existiert in der Mechanik keine verbreitete Interpretation.

**Der J-Invariantensatz** Die Berechnung der J-Invarianten ist identisch zum I-Invariantensatz, nur dass statt  $A$  der Deviator von  $A$  als Eingabe verwendet wird:

- $J_1(A) = \text{tr}(\tilde{A})$  (Spur des Deviators von  $A$ )
- $J_2(A) = \frac{1}{2}(\text{tr}(\tilde{A})^2 - \text{tr}(\tilde{A}^2))$  (Summe der Hauptminoren des Deviators von  $A$ )
- $J_3(A) = \det(\tilde{A})$  (Determinante des Deviators von  $A$ )

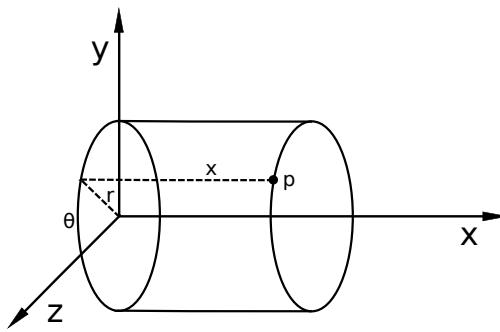


Abbildung 2: Darstellung eines zylindrischen Koordinatensystems.  $x, y$  und  $z$  entsprechen dabei den kartesischen Achsen. Die Koordinaten des Punktes  $p$  sind angegeben als  $x, r, \theta$ .  $x$  entspricht der kartesischen Koordinate,  $r$  ist die Entfernung zwischen der Projektion von  $p$  auf die von  $y$  und  $z$  Achse aufgespannte Fläche und dem Koordinatenursprung und  $\theta$  entspricht dem Winkel zwischen Projektion von  $p$ , dem Koordinatenursprung und der  $x$ -Achse.

Dabei ist jedoch zu beachten, dass

$$\begin{aligned}
 tr(\tilde{A}) &= tr(A - \frac{1}{3}tr(A)I) \\
 &= a_{11} - \frac{1}{3}tr(A) + a_{22} - \frac{1}{3}tr(A) + a_{33} - \frac{1}{3}tr(A) \\
 &= a_{11} + a_{22} + a_{33} - tr(A) \\
 &= tr(A) - tr(A) \\
 &= 0,
 \end{aligned} \tag{41}$$

weshalb statt  $J_1$  in der Regel  $I_1$  als Teil des Invariantensatzes verwendet wird. Ähnlich wie  $I$  ist auch  $J$  nicht orthogonal.

**Der K-Invariantensatz** Der K-Invariantensatz ist orthogonal und ist für eine Matrix  $A$  definiert als

- $K_1(A) = tr(A)$  (Spur von  $A$ )
- $K_2(A) = norm(\tilde{A})$  (Norm des Deviators von  $A$ )
- $K_3(A) = mode(\tilde{A})$  (Modus des Deviators von  $A$ ).

Da  $K_1 \in [-\infty; \infty]$ ,  $K_2 \in [0; \infty]$  und  $K_3 \in [-1; 1]$  ist, bietet sich für den K-Invariantensatz eine Darstellung in einem zylindrischen Koordinatensystem an, wobei  $K_1$  eine Position auf einer zentralen Achse beschreibt,  $K_2$  die orthogonale Entfernung zu diesem Punkt

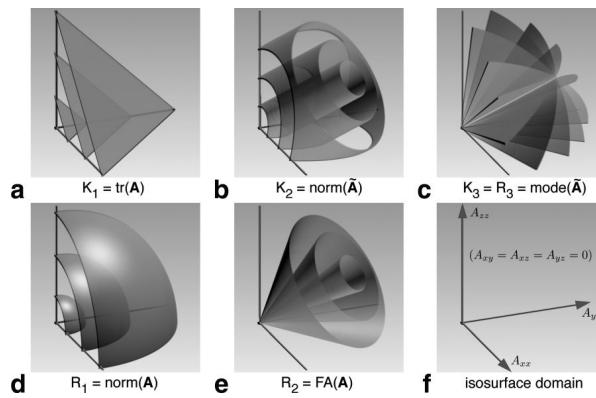


Abbildung 3: Dargestellt sind Isolänen der K- und R-Invarianten von diagonalisierten  $3 \times 3$  Matrizen (alle Werte außerhalb der Hauptdiagonale sind 0). Die Koordinatenachsen entsprechen den drei Eigenwerten der Matrizen. Entnommen aus [11, S. 139]

und  $K_3$  den Winkel zu einer festgelegten, zur zentralen Achse orthogonalen, zweiten Achse. Sowohl ein zylindrisches als auch ein kartesisches Koordinatensystem sind in Abb. 2 dargestellt.  $K_1$  entspricht dabei der zylindrischen  $x$ -Koordinate,  $K_2$  dem Radius  $r$  und  $K_3$  dem Winkel  $\theta$ .

Ein Vorteil des K-Invariantensatzes ist die relativ leichte Interpretierbarkeit in der Kontinuumsmechanik.  $K_1$  kann als Maß der Stärke des iostropen Anteils der Matrix,  $K_2$  als Maß des anisotropen Anteils angesehen werden [23]. Dagegen kann aus dem Wert von  $K_3$  die Verteilung der Eigenwerte und damit die Form der Anisotropie des Tensors geschlussfolgert werden.

**Der R-Invariantensatz** Die Invarianten des orthogonale R-Invariantensatz sind für eine Matrix  $A$  definiert als

- $R_1(A) = \text{norm}(A)$  (Norm von  $A$ )
- $R_2(A) = \sqrt{\frac{3}{2} \frac{\text{norm}(\tilde{A})}{\text{norm}(A)}}$  (fraktionale Anisotropie)
- $R_3(A) = \text{mode}(A)$  (Modus von  $A$ ).

Dabei fällt auf, dass gilt  $R_3(A) = K_3(A) = \text{mode}(A)$ . Ähnlich wie der K-Invariantensatz können auch die R-Invarianten in einem zylindrischen Koordinatensystem dargestellt werden.

Genauso wie für den K-Invariantensatz existiert auch für den R-Invariantensatz eine Interpretation in der Kontinuumsmechanik.

Sowohl K- als auch R-Invarianten beschreiben die Verteilung, die ‚Größe‘ und das Verhältnis der Eigenwerte zueinander. Dies ist in Abb. 3 dargestellt.

### 3.3.22 Kontinuierliche Menge

### 3.3.23 Diskrete Menge

### 3.3.24 Faltung

Die kontinuierliche Faltung  $*_c$  zweier Funktionen  $f, g : \mathbb{R}^n \rightarrow \mathbb{C}$  ist definiert als

$$(f *_c g)(x) = \int_{\mathbb{R}^n} f(\tau)g(x - \tau)d\tau. \quad (42)$$

Falls das Urbild beider Funktionen eine diskrete Menge  $\mathbb{D} \subset \mathbb{R}^n$  ist, wird stattdessen die diskrete Faltung  $*_d$  verwendet. Diese ist definiert als

$$(f *_d g)(n) = \sum_{k \in \mathbb{D}} f(k)g(n - k). \quad (43)$$

Das Ergebnis der Faltung ist eine neue Funktion, die bildlich betrachtet den um  $g$  gewichteten Mittelwert von  $f$  an jedem Punkt darstellt.  $g$  wird in diesem Fall als Kern oder Faltungskern bezeichnet. Um beispielsweise den Wert von  $(f * g)$  an der Stelle  $x$  zu bestimmen, wird  $g$  um  $x$  verschoben,  $f$  und  $g$  punktweise multipliziert und das Integral des Ergebnisses gebildet, was der Bestimmung des Mittelwerts entspricht. Dies geht aus dem Mittelwertsatz der Integralrechnung hervor.

### 3.3.25 Gaußsche Glockenkurve

Glockenkurven sind Funktionen der Form

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \quad (44)$$

Die Variablen  $\mu$  und  $\sigma$  dienen dazu, die Form der Glockenkurve und ihre Position im Koordinatensystem zu beeinflussen. Das Maximum der Glockenkurve liegt immer bei  $\mu$ . Mit  $\sigma$  lässt sich die Breite und Höhe der Glockenkurve beeinflussen, wobei die Fläche darunter gleich bleibt.

In Abb. 4 sind drei Glockenkurven mit unterschiedlichen Werten für  $\mu$  und  $\sigma$  dargestellt.

Im zweidimensionalen Fall existieren analoge Glockenkurven der Form

$$f(x, y) = \frac{1}{\sqrt{2\pi}\sigma_x^2} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}} \cdot \frac{1}{\sqrt{2\pi}\sigma_y^2} e^{-\frac{(y-\mu_y)^2}{2\sigma_y^2}}. \quad (45)$$

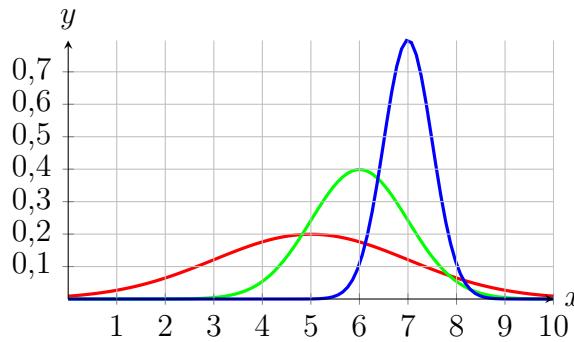


Abbildung 4: Darstellung von drei Gaußschen Glockenkurven mit unterschiedlichen  $\mu$  und  $\sigma$ : Rot:  $\mu = 5, \sigma = 2$ , Grün:  $\mu = 6, \sigma = 1$ , Blau:  $\mu = 7, \sigma = 0,5$

Die Variablen  $\mu_x$  und  $\mu_y$  dienen dabei dazu, die Positionen der Kurven in der jeweiligen Dimension festzulegen. Die Höhe und Form der Glockenkurve ergeben sich aus  $\sigma_x$  und  $\sigma_y$ . Eine Erhöhung von  $\sigma_x$  beispielsweise führt dazu, dass die Höhe des Extremums reduziert und die Glocke in x-Richtung gestreckt wird.

## 3.4 Mechanische Grundlagen

### 3.4.1 Physikalisches Feld

Ein physikalisches Feld ist eine physikalische Größe, die an verschiedenen Positionen im Raum unterschiedliche Werte annimmt[12, 1–2 Electric and magnetic fields]. Abstrahiert kann es als Funktion  $\mathbb{R}^n \rightarrow V$  beschrieben werden, wobei  $V$  eine beliebige Menge ist. Häufig wird für  $V$  jedoch  $\mathbb{R}$ , z.B. bei Temperaturen,  $\mathbb{R}^n$ , z.B. bei Strömungen, oder  $\mathbb{R}^{m \times n}$ , z.B. bei Verformungen, eingesetzt. Andere Beispiele für physikalische Felder sind elektromagnetische Felder oder Gravitationsfelder. Felder werden meist abhängig von ihrem Bild klassifiziert, so z.B. in Skalar-, Vektor- oder Tensorfelder.

Wenn Felder nicht in Form einer kontinuierlichen Funktion beschrieben werden können, weil z.B. nur für endlich viele Punkte Messwerte vorhanden sind, kann ein Feld auf einem Gitter, das aus diesen Messpunkten besteht, definiert werden. Die Werte innerhalb der Zellen lassen sich dann durch Interpolation der Werte an den Eckpunkten berechnen.

Innerhalb der vorliegenden Arbeit werden Spannung und Verformung an Punkten von Objekten als Felder  $\mathbb{R}^3 \rightarrow \mathbb{R}^{m \times n}$  beschrieben.

### 3.4.2 Mechanische Spannung

In der Mechanik beschreibt Spannung die Kräfte, die Partikel innerhalb eines Objektes aufeinander auswirken. Um die Spannung an einem Punkt zu bestimmen, wird die Kraft, die auf infinitesimal kleine Flächenteile von Schnitten durch das Objekt wirkt, berechnet.

Die Kraft ist dabei als Vektor formulierbar. Wenn die Schnitte entlang von drei zueinander orthogonalen Ebenen durchgeführt werden, erhält man so drei Vektoren, die zusammen die Matrixdarstellung des Cauchy Tensors  $\sigma$  bilden:

$$\sigma = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} \quad (46)$$

Jede Spalte des Cauchy Tensors entspricht dabei einem der Vektoren. Wie aus der Formel ersichtlich, ist die Matrix symmetrisch und enthält zwei Typen von Komponenten: Die  $\sigma$  entlang der Hauptdiagonale, die Kraft in Richtung der Normalen der jeweiligen Ebene angeben, und den  $\tau$ , die Scherspannungen angeben, die parallel zu den Schnittebenen verlaufen.

### 3.4.3 Mechanische Verformung

Wenn in einem Objekt Spannung vorliegt, so verformt es sich proportional zur Stärke der Spannung (Hooke'sches Gesetz). Durch einen Verformungstensor, auch Verzerrungstensor genannt, lässt sich die Verformung an einem Punkt durch Kraftanwendung angeben:

$$\epsilon = \begin{bmatrix} \epsilon_x & \frac{1}{2}\gamma_{xy} & \frac{1}{2}\gamma_{xz} \\ \frac{1}{2}\gamma_{xy} & \epsilon_y & \frac{1}{2}\gamma_{yz} \\ \frac{1}{2}\gamma_{xz} & \frac{1}{2}\gamma_{yz} & \epsilon_z \end{bmatrix} \quad (47)$$

Ähnlich wie beim Spannungstensor drücken die  $\epsilon$  in der Hauptdiagonale Dehnungen oder Stauchungen des Objektes entlang der Hauptachsen aus, die  $\gamma$  Werte Scherungen, also Verschiebungen der Seitenflächen zueinander. Dabei ändern sich die Winkel der Kanten des Objektes zueinander.

## 4 Verwendete Verfahren und Technologien

### 4.1 FAnToM

FAnToM[1, 40] („Field Analysis using Topological Methods“) ist ein Programm, dessen Entwicklung im Jahre 1999 begann. Obwohl es zu Anfang noch als Hilfsmittel für die Analyse von Vektorfeldtopologien konzipiert war, entwickelte es sich über die Jahre hinweg zu einer Plattform für diverse Visualisierungen.

Erweiterungen von FAnToM werden in Form von sogenannten ‚Algorithmen‘ entwickelt. Dabei unterscheidet man zwei Arten: den ‚Data Algorithmus‘ und den ‚Visualization Algorithmus‘. Data Algorithmen sind Erweiterungen, die Daten verarbeiten. Beispiele

dafür sind ‚Load VTK‘, eine Erweiterung, die Daten aus dem VTK Format einliest, oder ‚Combine Scalar to Vector‘, das mehrere Felder mit skalaren Werten zu einem einzigen Feld mit Vektorwerten kombiniert, wobei die Komponenten eines Vektors an einem Punkt den Skalaren der Eingabefelder am selben Punkt entsprechen. Visualization Algorithmen dagegen dienen dazu, Visualisierungen aus Daten zu erzeugen. Interaktionen mit den Visualisierungen können entweder über die Maus oder über Optionen stattfinden. Ein Beispiel für einen Visualization Algorithmus ist ‚Show Grid‘, durch das eine Zellstruktur als Gitternetz dargestellt wird.

Ein wesentliches Feature von FAnToM besteht darin, Algorithmen miteinander zu verknüpfen. Dies geschieht, indem Ausgabedaten von Algorithmen als Eingabe für andere dienen können. Da viele Algorithmen mit Blick auf Wiederverwendbarkeit entwickelt wurden, können mit wenig Aufwand aus bestehenden Algorithmen komplett neue Visualisierungspipelines entwickelt werden. Dies geschieht mittels des sogenannten ‚Flow Graphs‘, in dem Algorithmen als Knoten dargestellt sind, deren Aus- und Eingaben miteinander verbunden werden können, was als Verbindung zwischen den jeweiligen Knoten im Flow Graph dargestellt wird.

Diese Pipelines können als FAnToM-Session abgespeichert und später wiederhergestellt werden.

FAnToM bietet eine Vielzahl mächtiger Werkzeuge, auf die bei der Entwicklung neuer Erweiterungen zurückgegriffen werden kann. Besonders relevant für die vorliegende Arbeit waren dabei Funktionen zur effizienten Verarbeitung von Feldern und Tensoren, die Schnittstellen für Qt und OpenGL sowie die große Bibliothek bestehender Algorithmen.

## 4.2 OpenGL

OpenGL[19] ist eine Spezifikation, die das Verhalten eines rasterbasierten Renderingsystems beschreibt. Sie definiert eine Schnittstelle, gegen die von zwei Seiten entwickelt werden kann: Zum einen von Seiten der Hardwarehersteller, die Funktionen von OpenGL auf ihrer Hardware implementieren, zum anderen von Anwendungsprogrammierern, die unabhängig von der Hardware, auf der das Programm laufen soll, gegen die Spezifikation von OpenGL entwickeln können. Da in der vorliegenden Arbeit OpenGL nur aus Sicht eines Anwendungsprogrammierers verwendet wurde, beschränkt sich die folgende Beschreibung darauf.

Die Verarbeitungsschritte von der Übergabe von Daten aus einer Anwendung zu OpenGL bis zum fertig gerenderten Bild werden als Renderpipeline bezeichnet. Seit Version 2.0 unterstützt OpenGL sogenannte ‚Shader‘, kleine Programmstücke, die es Nutzern von OpenGL ermöglichen, die Renderpipeline sehr stark zu verändern. Shader bekommen drei Arten von Eingabedaten:

1. Die Ausgabe des vorherigen Schrittes in der Renderpipeline

2. Parameter aus dem Programm, das OpenGL verwendet („Uniforms“)
3. Daten, die aus dem vorhergehenden Shader übergeben wurden

Ein wichtiger Spezialfall von Uniforms sind Texturen, die in dem OpenGL verwendenden Programm geladen, an die Shader übergeben und dort verwendet werden können.

Für die vorliegende Arbeit wurden Vertex, Geometry und Fragment-Shader entwickelt. Diese drei Typen sind im Folgenden kurz erklärt.

**Vertex-Shader** Vertex-Shader bekommen als Eingabe aus der Renderpipeline die Punkte eines zu zeichnenden Grafik-Primitive, beispielsweise die Punkte eines Liniensegmentes oder die Eckpunkte eines Dreiecks. Abhängig von den eingegebenen Uniforms können die Koordinaten dieser Punkte dann durch den Shader verändert werden. Dies ist z.B. üblich, um die Sicht auf die Szene von der Position red Kamera aus zu berechnen. Vertex-Shader werden einmal pro Punkt ausgeführt. Die Ausgabe des Vertex-Shaders sind die neuen Koordinaten der Punkte der Primitives.

**Geometry-Shader** Die Renderpipeline übergibt dem Geometry-Shader für jedes Primitive eine Liste der zugehörigen Punkte, z.B. eine Liste von drei Punkten für ein Dreieck. Der Geometry-Shader ist in der Lage, die Position und den Typ dieser Primitives beliebig zu verändern und sogar neue Punkte zu erzeugen. So kann er beispielsweise aus einer Linie ein Dreieck erzeugen, sowie Uniforms pro Punkt definieren, welche an später ausgeführte Shader (z.B. den Fragment-Shader) weitergegeben werden. Pro Primitive wird der Geometry-Shader einmal ausgeführt.

**Fragment-Shader** Fragment-Shader bestimmen die Farben der Fragmente. Fragmente sind Stücke der projizierten Primitives, deren Größe von der Art der Rasterisierung abhängen. Aus den Fragmenten werden die Farben der Pixel des gerenderten Bildes bestimmt. In der Regel wird pro Pixel mindestens ein Fragment berechnet. Wenn mehrere Fragmente zu einem Pixel kombiniert werden bezeichnet man dies als Supersampling. Die Eingaben in den Fragment-Shader sind die Mittelpunkte der Fragmente und die für diese Punkte interpolierten Attribute der Eckpunkte des Primitive. Neben der Fragmentfarbe können auch andere Werte berechnet und ausgegeben werden, z.B. ein Tiefenattribut, das die Entfernung des Fragments zur Kamera angibt und damit korrekte Überlappung von Primitives ermöglicht.

Eine weitere im Folgenden verwendete Funktion von OpenGL ist die Möglichkeit, gerenderte Bilder mithilfe von „Framebuffern“ nicht anzuzeigen, sondern stattdessen in einer Textur zu speichern. Diese abgelegten Bilder können durch das sogenannte „Blending“ mit den Ergebnissen späterer Renderings kombiniert werden. Dabei wird ein weiteres Bild auf dem bestehenden gezeichnet und die Farbwerte entsprechend von Wichtungsfunktionen kombiniert. Dies ermöglicht beispielsweise die Darstellung transparenter Objekte.

Die meisten Implementierungen von OpenGL machen intensiven Gebrauch von Grafikkarten und deren Fähigkeit zur extremen Parallelisierung von Aufgaben. Dies, verbunden mit der Möglichkeit Ausgaben der Renderpipeline zurück in den Arbeitsspeicher zu laden und der Anpassbarkeit der Renderpipeline durch Shader, ermöglicht es, OpenGL generell für rechenintensive, gut parallelisierbare Aufgaben zu benutzen und so Berechnungen stark zu beschleunigen.

## 4.3 Qt

Qt[6] ist eine populäre, plattformübergreifende Bibliothek für die Entwicklung von Programmoberflächen. Die Popularität von Qt beruht auf der Plattformunabhängigkeit und der großen Vielfalt leicht zu verwendender Funktionen. Darunter sind beliebte Oberflächenelemente wie Knöpfe, Textfelder und Ankreuzkästen, wodurch die Gestaltung einer Nutzerschnittstelle sehr einfach wird.

FAnToM besitzt eine auf Qt basierende Oberfläche, die aus Algorithmen heraus um weitere Fenster und Elemente erweitert werden kann. Eine wesentliche Fähigkeit von Qt ist dabei, einen OpenGL Kontext zu erzeugen, sodass durch OpenGL gerenderte Bilder angezeigt werden können. Diese Fähigkeit von Qt wird innerhalb dieser Arbeit verwendet, um mehrere, unabhängige Datensichten zu implementieren.

## 4.4 Continuous Scatterplotting

### 4.4.1 Scatterplotting

Scatterplotting ist ein weit verbreitetes Verfahren, um Attribute von Objekten in Relation zueinander darzustellen. Es ist eine Funktion auf einer Menge von Objekten  $M$  als  $\tau : M \rightarrow \mathbb{R}^n, n \in \{1, 2, 3\}$ , wobei das Bild eine Projektion auf einzelne oder zusammengesetzte Attribute der Objekte darstellt. Dazu wird je Objekt ein Punkt in ein ein-, zwei- oder dreidimensionales Koordinatensystem eingetragen, dessen Koordinatenachsen Attributwerten entsprechen. Beispiele für Scatterplottings sind in Abb. 5 zu sehen. Das Bild von  $\tau$  ist auf maximal drei Dimensionen beschränkt, da höherdimensionale Daten visuell nur schwer zu interpretieren sind.

Ein Vorteil von Scatterplottings ist die leichte Erkennbarkeit von Strukturen in den Datensätzen: Bereiche hoher Punktdichte (Cluster), Punkte weit entfernt von allen anderen Punkten (Outlier) und funktionale Zusammenhänge zwischen den verwendeten Attributen lassen sich gut identifizieren.

Scatterplotting hat jedoch zwei gravierende Nachteile. Zum einen macht die geringe Dimensionalität des Bildes von  $\tau$  oft die Projektion auf eine Teilmenge der Attribute der dargestellten Objekte notwendig, zum anderen kann Scatterplotting nur endlich viele Objekte gleichzeitig darstellen.

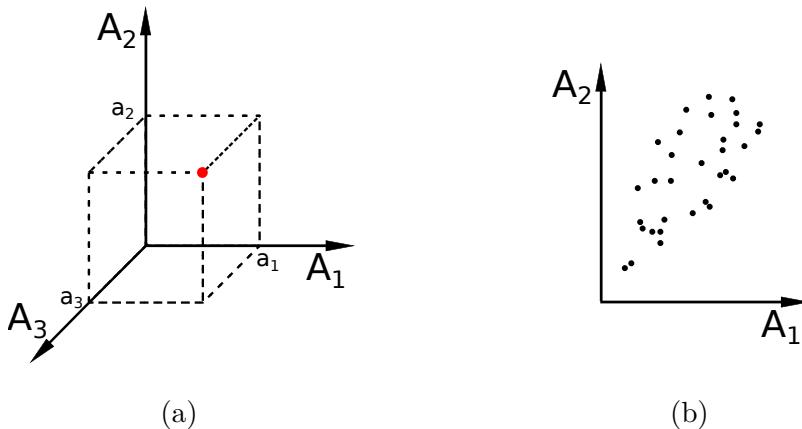


Abbildung 5: Zwei Beispiele für Scatterplottings. In (a) wird ein Objekt mit den Attributwerten  $a_1, a_2, a_3$  als roter Punkt iKCSGK6Q8rBsihln einem dreidimensionalen Scatterplot dargestellt. Um die Attribute besser ablesen zu können, wurden gestrichelte Linien eingezeichnet. In (b) wurden mehrere Objekte mit unterschiedlichen Attributen im selben, zweidimensionalen Scatterplot als schwarze Punkte eingezeichnet.

Ein praktisches Beispiel eines Urbildes mit unendlich vielen Elementen ist  $\mathbb{R}^n$  bei physikalischen Feldern. Dort sind an jedem der unendlich vielen Punkte im Raum Werte definiert. Ein Scatterplot dieser unendlich vielen Elemente ist nun weder in endlicher Laufzeit möglich, noch wären einzelne Punkte in der entstehenden Darstellung erkennbar, da in jedem Fall nur endlich viele Pixel zur Verfügung stehen. Eine Alternative besteht darin, nur den Scatterplot der Eckpunkte der Zellen zu erzeugen, doch damit gehen Informationen über das Innere der Zellen verloren.

#### 4.4.2 Erklärung des Continuous Scatterplotting

Das Continuous Scatterplotting stellt eine Erweiterung des Scatterplotting dar, sodass als Urbild statt einer endlichen Anzahl von Objekten auch kontinuierliche Mengen dienen können. Die von Bachtaler und Weiskopf[3] vorgestellte Definition wurde von Fritsch[14] um einige wichtige Punkte erweitert. Beim Continuous Scatterplotting wird ein Ansatz verwendet, der dem Verfahren bei der Herleitung der Kontinuumsmechanik aus Systemen mit diskreten Massenpunkten ähnelt [3, S. 1429]. Das Ziel des Continuous Scatterplotting ist es, eine Dichtefunktion  $\sigma : \mathbb{R}^m \rightarrow \mathbb{R}$  zu finden, die den Punkten des Bildraums, abhängig von  $\tau$ , Dichtewerte zuordnet.  $\sigma$  kann als Funktion aufgefasst werden, die jedem Pixel des Scatterplots einen skalaren Wert zuordnet, der z.B. als Farbe oder Opazität dargestellt werden kann. In Abb. 6 ist der Effekt, den  $\tau$  auf die Dichtefunktion hat, dargestellt. Um  $\sigma$  berechnen zu können, werden zwei Annahmen getroffen:

Die erste Annahme ist, dass eine Dichtefunktion  $s : \mathbb{R}^n \rightarrow \mathbb{R}$  existiert, die jedem Punkt des Urbilds von  $\tau$  einen skalaren Wert, die Dichte an diesem Punkt, zuordnet. Der Einfachheit halber wird im Folgenden eine uniforme Dichte von  $s(x) = 1$  angenommen.

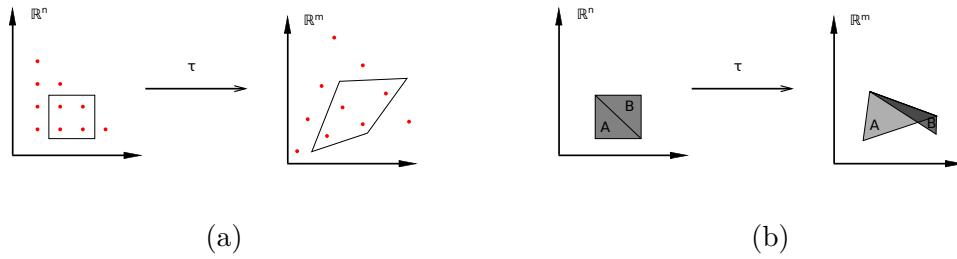


Abbildung 6: Darstellung der Dichteverteilung abhängig von  $\tau$ . (a) Die roten Punkte deuten die Dichte in den beiden Scatterplots an, die zwei Vierecke sind Teilmengen des Urbilds, die durch  $\tau$  verformt werden. Zu erkennen ist, dass die Streckung des Vierecks zur Erhöhung der Entfernung zwischen den Punkten führt, was einer Verringerung der Dichte im Viereck entspricht. (b) Der Continuous Scatterplot zweier Dreiecke. Der Grauwert gibt die Dichte an Punkten innerhalb der Dreiecke an.

Aus der Dichte an den Punkten lässt sich die Gesamtdichte  $M$ , auch bezeichnet als Masse, einer Teilmenge  $V \subset \mathbb{R}^n$  berechnen:

$$M = \int_V s(x) d^n x \quad (48)$$

Die zweite Annahme besagt, dass die Masse einer Teilmenge  $\phi \subset \mathbb{R}^m$  gleich der Masse ihres Urbildes ist, also  $\tau$  die Gesamtdichte nicht beeinflusst. Da  $\tau$  in der Regel nicht invertierbar ist, wird  $\tau^{-1}(\phi)$  verwendet, um das Urbild von  $\phi$  zu notieren.

Für  $\xi \in \Phi$  führen die Annahmen zu der Gleichung

$$\int_{\phi} \sigma(\xi) d^m \xi = \int_{\tau^{-1}(\phi)=V} s(x) d^n x. \quad (49)$$

Da  $\tau$  weder injektiv noch surjektiv ist, gilt der umgekehrte Fall nur, wenn zusätzlich  $\forall x \in V : f^{-1}(f(x)) \subset V$  gilt [14, S. 20]:

$$\int_V s(x) d^n x = \int_{\phi=\tau(V)} \sigma(\xi) d^m \xi \quad (50)$$

Fritsch [14, S. 20 f.] definiert zusätzlich ein  $\sigma_v$ , das die Dichtefunktion der Einschränkung von  $\tau$  auf  $V$  beschreibt, also den ‚Beitrag‘ von  $V$  zu  $\sigma$ :

$$\int_{\phi} \sigma_v(\xi) d^m \xi = \int_{\tau|_V^{-1}(\phi)} s(x) d^n x \quad (51)$$

Mithilfe der  $\sigma_v$  stellt Fritzsch fest, dass für eine Zerlegung des Definitionsbereichs  $U$  von  $\tau$  in eine Menge von disjunkten Teilmengen  $\hat{V} = V_i | V_i \in U$  gilt, dass

$$\begin{aligned}
 \int_{\phi} \sigma(\xi) d^m \xi &= \int_{\bigcup_{V_i} \tau|_{V_i}^{-1}(\phi)} s(x) d^n x \\
 &= \sum_{V_i} \int_{\tau|_{V_i}^{-1}(\phi)} s(x) d^n x \\
 &= \sum_{V_i} \int_{\phi} \sigma_{V_i}(\xi) d^m \xi \\
 &= \int_{\phi} \sum_{V_i} \sigma_{V_i}(\xi) d^m \xi.
 \end{aligned} \tag{52}$$

Da die Integranden des ersten und letzten Integrals gleich sein müssen, ergibt sich daraus

$$\sigma(\xi) = \sum_{V_i} \sigma_{V_i}(\xi). \tag{53}$$

Eine intuitive Interpretation von Gleichung (53) ist, dass die gesamte Dichtefunktion von  $\tau$  als Summe der Dichtefunktionen der Einschränkungen von  $\tau$  auf die  $V_i$  gebildet werden kann.

Die Berechnung von  $\sigma$  hängt vom Verhältnis von  $m$  und  $n$  zueinander ab. Da in der vorliegenden Arbeit nur der Fall  $m = n = 3$  auftritt, beschränkt sich die weitere Erläuterung auf diesen Fall. Für  $m = n$  unterscheiden Bachthaler und Weiskopf folgende Fälle für  $V \subseteq \mathbb{R}^n$  [3, S. 1430]:

### Fall 1: $\tau$ ist differenzierbar und ein Diffeomorphismus

Ein Diffeomorphismus ist eine bijektive, stetig differenzierbare Abbildung, deren Umkehrabbildung ebenfalls stetig differenzierbar ist. Wenn  $\tau$  ein Diffeomorphismus ist, lässt sich durch die Anwendung des Transformationssatzes von Integralen die Gleichung

$$\int_{\tau(V)} \sigma(\xi) d^{m=n} \xi = \int_V \sigma(\tau(x)) |\det(J_\tau)(x)| d^n x = \int_V s(x) d^n x \tag{54}$$

bilden, wobei  $J_\tau$  die Jacobimatrix von  $\tau$  ist. Da der Wert der Determinante der Jacobimatrix von der Position abhängig ist, wird der jeweilige Punkt  $x$  eingesetzt. Durch weitere Umformungen entsteht die Gleichung

$$\sigma(\xi) = \frac{s(\tau^{-1}(\xi))}{|\det(J_\tau)(\tau^{-1}(\xi))|}. \tag{55}$$

Wie schon in Kapitel 3 erwähnt, entspricht der Betrag der Determinanten der Jacobimatrix dem Betrag der Expandierung oder Schrumpfung der Funktion in der Nähe des jeweiligen Punktes. Wenn die Funktion expandiert, nimmt die Dichte ab und wenn sie schrumpft, nimmt die Dichte zu.

**Fall 2:  $\tau$  ist differenzierbar und konstant über  $V$ ,  $V$  ist keine Nullmenge**

Wenn  $\det(J_\tau) = 0$ , dann ist  $\tau$  kein Diffeomorphismus und  $\sigma$  kann nicht so wie in **Fall 1** definiert werden. Das ist z.B. der Fall, wenn  $\tau$  Bereiche mit konstanten Werten enthält und führt dazu, dass eine nicht leere Teilmenge von  $\mathbb{R}^n$  auf einen einzelnen Punkt  $\xi$  abgebildet wird. Eine Möglichkeit die unendlich hohe Dichte an  $\xi$  darzustellen und dennoch die Gesamtmasse nicht zu verändern ist es, bei  $\xi$  ein Dirac-Delta einzufügen, das mit dem Volumen aller konstanten Bereiche  $V_i \in V$ , die nach  $\xi$  abgebildet werden, skaliert wird:

$$\begin{aligned}\sigma(\xi) &= \sum_{V_i \in V} \delta(\xi - \tau(V_i)) \cdot \int_{V_i} s(x) d^n x \\ &= \sum_{V_i \in V} \delta(\xi - \tau(V_i)) \cdot \text{Vol}(V_i). \quad (\text{da } s(x) = 1)\end{aligned}\tag{56}$$

Aus den Eigenschaften des Dirac Deltas und unter der Annahme, dass  $s(x) = 1$ , ergibt sich, dass die Masseerhaltung erfüllt ist:

$$\begin{aligned}\int_{\phi} \sigma(\xi) d^n \xi &= \sum_{V_i} \int_{\phi} \delta(\xi - \tau(V_i)) \cdot \text{Vol}(V_i) d^n \xi \\ &= \sum_{\forall V_i : \tau(V_i) \in \phi} \text{Vol}(V_i) \\ &= \int_{\tau^{-1}(\phi)} 1 d^n x\end{aligned}\tag{57}$$

In Abb. 6b ist ein Beispielfall dafür angegeben. Die Dichte im überlappenden Bereich der beiden Dreiecke ist gleich der Summe der Dichten der einzelnen Dreiecke. Bei anderen Dichtefunktionen muss Gleichung (56) entsprechend angepasst werden.

**Fall 3:  $\tau$  ist differenzierbar und kein Diffeomorphismus,  $V$  ist Nullmenge**

Ein weiterer Fall, in dem  $\det(J_\tau) = 0$  ist, tritt auf, wenn  $V$  eine Nullmenge ist (also z.B. eine Linie im  $\mathbb{R}^3$ ). Da Integration über Nullmengen immer 0 ergibt, können die Nullmengen und ihr Bild bezüglich  $\tau$  einfach aus dem Scatterplotting entfernt werden, ohne die Masseerhaltung zu verletzen.

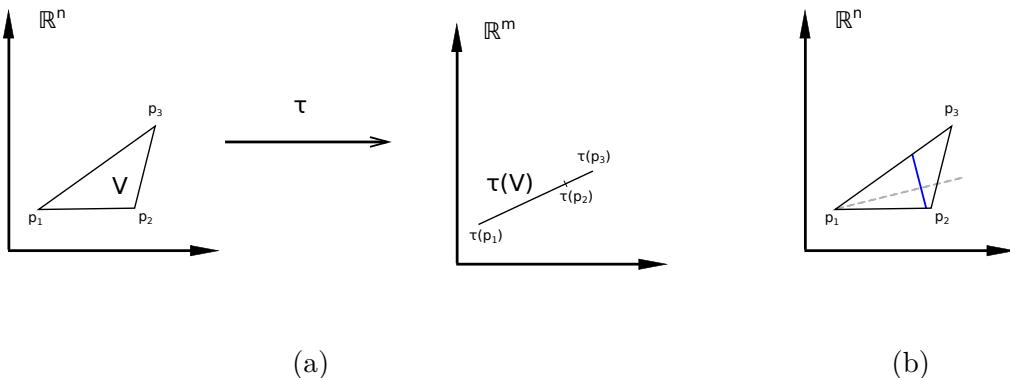


Abbildung 7: Darstellung von **Fall 5**. (a) Alle drei Punkte des Dreiecks  $V$  im linken Koordinatensystem werden auf eine Gerade abgebildet. Dadurch ist  $\tau(V)$  eine Nullmenge. (b) Eine Darstellung des von Fritzsch vorgeschlagenen Ansatzes. Die graue, gestrichelte Linie entspricht dem Gradienten, die Länge der blauen Linie ist proportional zur Dichte.

#### **Fall 4: $\tau$ ist nicht differenzierbar**

Wenn im Datensatz Unstetigkeiten auftreten, z.B. zwischen Zellen, ist  $\tau$  an diesen nicht differenzierbar. Da die unstetigen Bereiche ebenfalls Nullmengen bilden, kann die gleiche Lösung wie in **Fall 3** gewählt werden.

#### **Fall 5: $\tau$ ist differenzierbar und kein Diffeomorphismus, $V$ ist keine Nullmenge**

Von Bachthaler und Weiskopf wurde noch ein fünfter Fall genannt, den sie jedoch als nicht in der Praxis vorkommend ansahen. Fritzsch konnte jedoch Beispiele für diesen Fall angeben[14, S. 23 f.], die in der Praxis eine Rolle spielen. Ein solcher Fall tritt ein, wenn  $V$  keine Nullmenge ist, jedoch auf eine Nullmenge  $\tau(V)$  abgebildet wird (siehe Abb. 7a), wenn also entweder mindestens eine Komponente des Bildes konstant ist, oder eine funktionale Abhängigkeit zwischen mindestens zwei Komponenten existiert.

Die Jacobimatrix enthält in diesem Fall mindestens eine Nullzeile, wodurch gilt, dass  $\text{rang}(M)$  zwischen 0 und  $m$  liegt..

Fritzsch schlägt zur Lösung dieses Problems zwei Ansätze vor. Der Erste orientiert sich an **Fall 2** und ordnet dem Bildbereich sehr hohe Dichten zu, um die Masseerhaltung zu gewährleisten.

Der zweite Ansatz ist nur für den Fall  $m = n = 2$  beschrieben, somit muss  $\text{rang}(M) = 1$  sein. Er kann jedoch auch auf höhere Dimensionen übertragen werden. Ziel ist es, die Dichte an Punkten  $p \in \tau(V)$  proportional zur Gesamtmasse der darauf abgebildeten Punkte festzulegen. Dadurch wird zwar die Masseerhaltung nicht erfüllt, aber eine interpretierbare Darstellung erzeugt. Dazu werden zunächst die Gradienten für jede Komponente des Bildes von  $V$  berechnet. Die Größe der  $\dim(V) - \dim(\tau(V))$ -dimensionalen Teilmengen von  $V$ , die orthogonal zu den durch die Gradienten aufgespannten Räumen sind, ist

gleich der Masse, die auf die einzelnen Punkte abgebildet wird. Ein Beispiel für den Fall  $\dim(V) = 2$ ,  $\dim(\tau(V)) = 1$  ist in Abb. 7b zu sehen. Die Dichte an den Punkten von  $\tau(V)$  wird dann proportional zur Masse gewählt.

## 4.5 Volumetrische Daten

Felder, die Punkten im dreidimensionalen Raum bestimmte Werte zuordnen, sind ein häufig vorkommender Typ von Datensätzen, der oft unter dem Begriff ‚volumetrische Datensätze‘ zusammengefasst wird. Dieser Abschnitt beschreibt einen der verbreitetsten Ansätze um solche Felder abzuspeichern.

Ein häufiger Typ von volumetrischen Datensätzen beschreibt die Positionen, Formen und Eigenschaften von Objekten im dreidimensionalen Raum. Dazu werden Punkten im  $\mathbb{R}^3$  Werte aus einer Menge  $M$  zugeordnet, die den Eigenschaften der Objekten entsprechen. Oft werden bestimmte Werte aus  $M$  dafür verwendet, Punkte zu markieren, an denen sich kein Objekt befindet. Im Fall von  $M = \mathbb{R}^n$  für ein beliebiges, festes  $n$  ist ein solcher Wert, der Punkte außerhalb von Objekten markiert, meist das Nullelement. Umgekehrt bedeuten andere Werte, dass sich der Punkt innerhalb eines Objektes befindet.

Das Ergebnis eines Continuous Scatterplotting der Form  $m = n = 3$  stellt beispielsweise einen volumetrischen Datensatz dar, in dem Punkten ein Dichtewert größer 0 zugeordnet wird, falls sie sich innerhalb eines Objektes befinden. Ansonsten ist die Dichte 0. Da die Visualisierung genau dieser Art von Datensätzen die zentrale Aufgabe der vorliegenden Arbeit ist, ist die Abspeicherung und das Auslesen von volumetrischen Daten ein wichtiges Teilproblem.

Bei sehr einfachen Feldern kann es ausreichen, eine Repräsentation einer mathematischen Funktion, durch die das Feld approximiert wird, zu speichern. Bei komplexeren Feldern ist dies meistens nicht möglich, da entweder keine hinreichend gut passende Funktion bekannt ist, oder aber die Auswertung der Funktion an einem Punkt sehr viel Rechenzeit benötigen würde, wie z.B. bei aufwändigen Simulationen.

Eine beliebte Möglichkeit zur Modellierung von volumetrischen Datensätzen besteht darin, den Raum, in dem sich das Objekt befindet, in Volumenelemente, genannt ‚Voxel‘, zu zerlegen, denen Werte zugeordnet werden. Welchem Teil des Voxels, z.B. Mittelpunkt, Eckpunkte, oder Seitenflächen, die Werte zugewiesen werden, ist vom Anwendungsfall abhängig. Im einfachsten Fall geben diese Werte binär an, ob das Voxel ein Objekt schneidet oder nicht. Komplexere Beispiele sind Eigenschaften von Objektstücken oder Teilen des Raums innerhalb des Voxels. Beispiele für solche Werte sind die Farbe und Opazität eines durchsichtigen Objektes, die Dichte einer Gesteinsschicht oder die von einem Volumen abgegebene Energie bei einer Magnetresonanztomographie. Im Allgemeinen bilden Voxel ein dreidimensionales Äquivalent zu Pixeln.

Voxel haben einen entscheidenden Vorteil gegenüber anderen volumetrischen Datensätzen. Die Komplexität der Daten, und daraus folgend die Rechenzeit der meisten Algorithmen,

die Voxel als Eingabe verwenden, ist nicht abhängig von der Anzahl und Komplexität der in der ursprünglichen Szene dargestellten Objekte, sondern nur noch von der Anzahl der Voxel, auf die die Szene abgebildet wird. Durch Anpassung der Anzahl der Voxel ist es möglich, Rechenzeit und Qualität der Ergebnisse von Algorithmen gegeneinander abzuwagen.

In der Praxis werden Voxel in unterschiedlichen Formen verwendet. Die populärste Variante sind Datensätze, die ein quaderförmiges Volumen in gleichgroße quaderförmige Voxel zerlegen. Aber auch Datensätze mit tetraedrischen Voxeln oder solche, deren Voxel unterschiedliche Formen annehmen, existieren.

## 4.6 Volumenvisualisierung

Zur Darstellung von Objekten werden in der Computergrafik meist Dreiecke verwendet. In der Computergrafik werden dreidimensionale Objekte üblicherweise durch Dreiecke dargestellt, durch die die Oberflächen der Objekte approximiert werden. Dieses Vorgehen entfernt jedoch alle Informationen über Strukturen im Inneren der Objekte, wie beispielsweise Bereiche gleicher Materialien oder Dichte. Um volumetrische Datensätze, wie beispielsweise die Ergebnisse von Continuous Scatterplots der Form  $m = n = 3$ , visualisieren zu können ohne Informationen über innere Strukturen zu verlieren werden deshalb besondere Verfahren benötigt. Diese werden allgemein unter dem Begriff ‚Volumenvisualisierung‘ zusammengefasst. Genauer definiert bezeichnet Volumenvisualisierung Methoden der Extraktion von bedeutungsvollen Informationen aus volumetrischen Daten durch interaktive Grafiken und Bildgebung [15, S. 127]. Nachfolgend werden einige Volumenvisualisierung für skalare Felder mit Vor- und Nachteilen vorgestellt. Dabei beschränken sich die Erläuterungen auf Datensätze, die auf, meist quaderförmigen, Voxeln basieren, deren Volumen ein skalarer Wert zugeordnet wurde. Für die meisten Algorithmen existieren jedoch auch Varianten, die mit anderen Formen volumetrischer Daten umgehen können.

Nachteile der Speicherung durch Voxel sind unter anderem der hohe Verbrauch an Speicher, da möglicherweise viele tausende Voxel abgespeichert werden müssen, die aus der diskreten Approximation entstehenden Artefakte, z.B. Treppeneffekte in Bereichen mit plötzlichen Veränderungen, und der Informationsverlust über die ursprüngliche Form der abgebildeten Objekte.

### 4.6.1 Isoflächen

Volumetrische Datensätze beinhalten oft komplexe Strukturen. Die Position und Form sowie die Datenwerte innerhalb dieser Strukturen gut zu vermitteln stellt oft ein zentrales Ziel der Volumenvisualisierung dar. Eine Möglichkeit diese Informationen zu vermitteln ist die Berechnung und Darstellung von Isoflächen. Die Isofläche einer Funktion  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$  und eines Isowerts  $i \in \mathbb{R}$  ist definiert als eine Oberfläche, die den  $\mathbb{R}^3$  in zwei Regionen

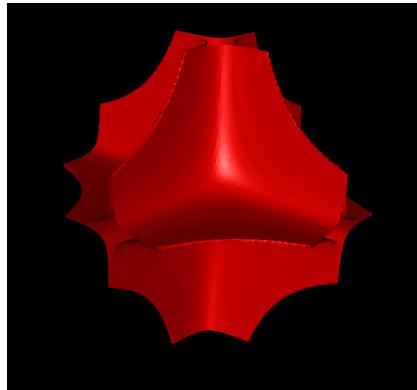


Abbildung 8: Darstellung einer Isofläche in einem automatisch generierten Datensatz. Erzeugt mithilfe von FAnToM [1].

$R_0, R_1$  teilt, wobei für jeden Punkt  $r_0 \in R_0$  gilt  $f(r_0) < i$  und für jeden Punkt  $r_1 \in R_1$  gilt  $f(r_1) > i$  [15, S. 7]. Die Isofläche muss dabei nicht zusammenhängend sein. Auf der Isofläche selbst ist  $f$  gleich dem Isowert.

In Abb. 8 ist ein Beispiel für eine Isofläche dargestellt. Der Datensatz ist ein Ausschnitt aus der Funktion  $f(x, y, z) = \cos(x \cdot y \cdot z)$ , wobei  $x, y$  und  $z$  im Intervall  $[-1; 1]$  liegen. Die Datenwerte sind den Eckpunkten der Voxel zugeordnet.

Die Berechnung von Isoflächen stellt eine Herausforderung dar, besonders bezüglich der Korrektheit des Verlaufs der Isoflächen innerhalb eines Voxels und der benötigten Rechenzeit. In der Vergangenheit wurden viele Verfahren entwickelt, die sich in Hinblick auf Korrektheit und Geschwindigkeit stark unterscheiden. Der wohl einflussreichste davon ist ‚Marching Cubes‘ [29]. Marching Cubes kategorisiert Voxel abhängig davon, welche der Werte ihrer Eckpunkte größer oder kleiner als der Isowert sind. Für jede so gebildete Kategorie von Voxeln existiert eine vordefinierte Menge von Dreiecken, die den Verlauf der Isolinie innerhalb dieser Voxel annähernd beschreiben. Für jedes Voxel werden die Dreiecke der jeweiligen Kategorie korrekt rotiert und zur Isofläche hinzugefügt.

Viele neuere Algorithmen bauen direkt auf Marching Cubes auf, indem sie entweder Fehler des ursprünglichen Algorithmus beheben, Optimierungen anbieten oder auf Voxeln anderer Formen (z.B. Tetraeder) anwendbar sind.

Die Visualisierung von volumetrischen Daten mithilfe von Isoflächen bietet eine Reihe von Vorteilen. Als Erstes muss die Isofläche nicht neu berechnet werden, solang sich der Isowert nicht ändert. Die erzeugten Dreiecke können mithilfe von 3D Bibliotheken und Grafikkarten sehr schnell und in sehr großer Anzahl dargestellt werden. Auch Rotation, Zoomen und andere Interaktionen brauchen nur vergleichsweise wenig Rechenaufwand. Des Weiteren bieten Isoflächen eine ausgezeichnete Darstellung der Form und Position von Strukturen innerhalb eines Datensatzes. Durch Schattierung und Beleuchtung der Dreiecke fällt es leicht, dem Benutzer eine Vorstellung der dreidimensionalen Form zu vermitteln.

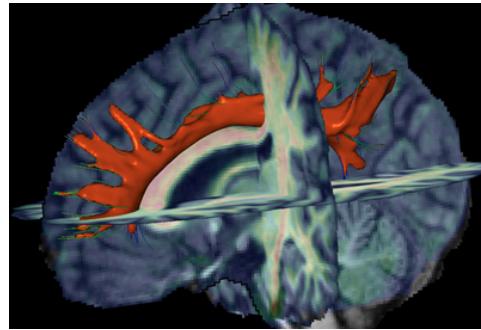


Abbildung 9: Darstellung von drei Slicings durch einen volumetrischen MRI Datensatz. In Orange ist zusätzlich eine Isofläche eingezeichnet. Entnommen aus [35].

Indem es dem Benutzer zusätzlich erlaubt wird, den Isowert interaktiv festzulegen, z.B. mithilfe eines Textfeldes oder eines Schiebereglers, wird es möglich, Entwicklungen der Werte über den Datensatz hinweg zu analysieren.

Isoflächen haben jedoch auch Nachteile. So kann es passieren, dass eine Isofläche, die einen geschlossenen dreidimensionalen Körper bildet, den Blick auf Strukturen im Inneren des Körpers blockiert. Und zuletzt stellen Isoflächen immer nur einen kleinen Teil des Datensatzes gleichzeitig dar, was es erschwert, einen Überblick des gesamten Volumens zu erhalten.

#### 4.6.2 Slicing

Ein einfacher Ansatz, um einen volumetrischen Datensatz zu visualisieren ist das sogenannte ‚Slicing‘[30]. Dabei wird der Schnitt zwischen dem Datensatz und einer Ebene berechnet, und dieser Schnitt durch eine Menge von verbundenen, texturierten Dreiecken, einem sogenannten ‚Dreieckszug‘ dargestellt. Form und Position des Dreieckzuges entsprechen dabei der Schnittebene. Mithilfe von Interpolation werden die Werte des volumetrischen Datensatzes an Punkten auf der Oberfläche der Dreiecke berechnet. Das Bestimmen von Werten an Punkten innerhalb der 3D Textur wird als ‚Abtastung‘, die Punkte als ‚Abtastpunkte‘ bezeichnet. Um aus den Abtastpunkten die Färbungen der Dreiecke zu berechnen, gibt es zwei Verfahren, bezeichnet als Präklassifizierung und Postklassifizierung.

Bei der Präklassifizierung wird zunächst die Farbe an den Abtastpunkten bestimmt. Dazu wird eine sogenannte ‚Transferfunktion‘  $T$ , verwendet die einem Punkt anhand seiner Datenwerte eine Farbe zuordnet. Ein Beispiel für eine Transferfunktion ist in Abb. 10 dargestellt. Die horizontale Achse entspricht den skalaren Werten zwischen Minimum und Maximum im Datensatz sowie den zugeordneten Farben. Die vertikale Achse entspricht der Opazität, von vollständig transparent am unteren Rand bis zu opak am oberen Rand. Durch Position und Farbe der Dreiecke am oberen Rand der Transferfunktion kann die Farbzuordnung festgelegt werden. Die Quadrate stellen interaktiv ausgewählte

Punkte dar, die Zuordnungen von Werten im Skalarfeld zu Farb- und Opazitätswerten entsprechen, zwischen denen entlang der verbindenden Linien interpoliert wird. So wird allen Werten im Feld eine Farbe und eine Opazität zugeordnet. Dabei ist zu beachten, dass Opazität im allgemeinen Fall bei Slicing nicht verwendet wird. Um die Farben an allen anderen Punkten der Schnittfläche zu bestimmen, werden die Farben der Abtastpunkte interpoliert. Ein Nachteil der Präklassifizierung liegt darin, dass durch die Interpolation Farben entstehen können, die nicht im Bild der Transferfunktion enthalten sind. Ein Beispiel: Sei  $f$  eine Funktion, die Punkten  $p \in \mathbb{R}^3$  einen skalaren Wert  $s \in \mathbb{R}$  zuordnet, also  $f(p) = s$ . Sei außerdem  $V$  ein volumetrischer Datensatz, der einen Ausschnitt von  $f$  enthält und  $T$  eine Transferfunktion, die Punkte dieses Datensatzes auf eine Farbe aus dem RGBA Farbraum abbildet, die durch einen Tupel  $(r, g, b, a)$  dargestellt wird. Die Komponenten des Tupels  $(r, g, b, a)$  seien zur Einfachheit auf den Bereich  $[0; 1]$  skaliert. Zusätzlich sei  $T(f(p)) = (r, g, b, a)$  definiert als

$$f(x) = \begin{cases} (1,0; 0,0; 0,0; 1,0) \text{ (Rot), wenn } x \leq 0,3 \\ (0,0; 1,0; 0,0; 1,0) \text{ (Grün), wenn } 0,3 < x \leq 0,6 \\ (0,0; 0,0; 1,0; 1,0) \text{ (Blau), wenn } 0,6 < x. \end{cases} \quad (58)$$

Seien nun drei Punkte  $p_0, p_1$  und  $p_2$  definiert, wobei  $p_0$  und  $p_2$  Abtastpunkte sind,  $p_1$  mittig zwischen  $p_0$  und  $p_2$  liegt sowie  $f(p_0) = 0,0$  und  $f(p_2) = 1,0$  gilt. Dann werden bei der Präklassifizierung zunächst die Farben an  $p_0$  und an  $p_2$  bestimmt und danach die Farbe an  $p_1$  durch Interpolation der Farben an  $p_0$  und  $p_2$  berechnet. In diesem Beispiel würde  $p_0$  das Tupel  $(1, 0; 0, 0; 0, 0; 1, 0)$ , und  $p_2$  das Tupel  $(0, 0; 0, 0; 1, 0; 1, 0)$  zugeordnet werden. Bei linearer, komponentenweiser Interpolation entsteht als Farbe von  $p_1$  das Tupel  $(0, 5; 0, 0; 0, 5; 1, 0)$ , ein dunkles Violett. Diese Farbe ist im Bild der Transferfunktion nicht vorhanden. Der Benutzer kann daher den Wert von  $f(p_1)$  nicht direkt ablesen, sondern muss zuerst bestimmen, welche Farben gemischt wurden und zu welchen Anteilen. Besonders problematisch ist dies, wenn die Mischfarbe in der Transferfunktion für einen komplett anderen Wert verwendet wurde, was zu falschen Schlüssen über die Verteilung der Werte im Datensatz führen kann.

Die Postklassifizierung löst diese Probleme, indem die Reihenfolge der Interpolation und Anwendung der Transferfunktion vertauscht wird. Somit werden zunächst die Datenwerte interpoliert und danach die Transferfunktion auf die interpolierten Werte angewendet. Dadurch werden Farbverläufe so wie in der Transferfunktion dargestellt. Es entsteht jedoch der Nachteil, dass durch die Interpolation neue Datenwerte entstehen können, die so im Datensatz nicht vorkommen. In den meisten Fällen wird dies jedoch als das geringere Übel angesehen, weshalb Postklassifizierung fast immer bevorzugt wird. In Abb. 9 sind drei Slicings zusammen mit einer orangenen Isofläche abgebildet.

Indem Interaktionen angeboten werden, durch die der Benutzer Position und Orientierung der Schnittfläche interaktiv anpassen kann, bietet Slicing eine gute Möglichkeit um die Verteilung von Werten innerhalb eines volumetrischen Datensatzes zu analysieren und Werte ablesen zu können. Der Rechenaufwand ist geringer als bei der Berechnung der

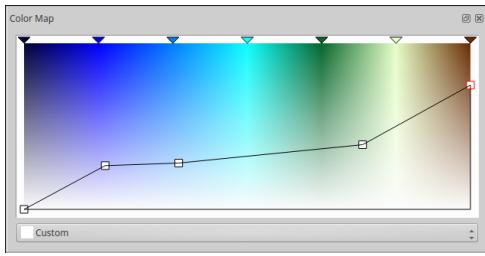


Abbildung 10: Eine Implementierung einer interaktiven Transferfunktion in FAnToM.

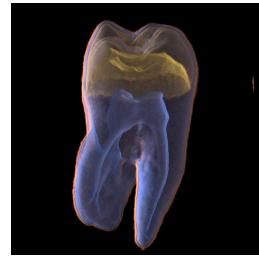


Abbildung 11: Ein Direct Volume Rendering eines Zahns. Entnommen aus [10, S. 6]

Isoflächen und macht es möglich, die Slicings ohne spürbare Verzögerung zu repositionieren. Slicings eignen sich auch, um andere Volumenvisualisierung zu ergänzen, wie z.B. die Isofläche in Abb. 9.

Die großen Nachteile von Slicings liegen in der Schwierigkeit, korrekte Positionen und Orientierungen zu finden, um interessante Bereiche sichtbar zu machen sowie darin, dass die Form von dreidimensionalen Strukturen relativ schlecht vermittelt wird.

#### 4.6.3 Direct Volume Rendering

Die bisher vorgestellten Volumenvisualisierung Verfahren stellten immer nur einen kleinen Teil des volumetrischen Datensatzes gleichzeitig dar. Das hat den Vorteil, dass ein Benutzer nur einen Teil der Daten gleichzeitig interpretieren muss, bringt jedoch eine Reihe von Nachteilen mit sich. Erstens wird vorausgesetzt, dass der Benutzer durch Interaktion unterschiedliche Bereiche auswählt, um sich einen Überblick über den Datensatz zu verschaffen, was abhängig von Größe und Komplexität viel Zeit und Aufmerksamkeit des Nutzers in Anspruch nehmen kann. Zweitens besteht dabei das Risiko, dass Merkmale des Datensatzes, die nur bei sehr bestimmten Einstellungen sichtbar sind, übersehen werden. Und drittens muss der Benutzer zusätzliche mentale Arbeit verrichten, um die jeweils angezeigten Teile in seiner Vorstellung zusammenzusetzen, damit er ein vollständiges Verständnis des Datensatzes entwickeln kann. Dies kann, wiederum abhängig von Größe und Komplexität, bei manchen Datensätzen fast unmöglich sein.

Der Begriff ‚Direct Volume Rendering‘ (DVR) bezeichnet eine Reihe von Verfahren der Volumenvisualisierung, die den gesamten Datensatz gleichzeitig darstellen können. Daher stammt auch das ‚Direct‘ im Namen dieser Gruppe von Verfahren: Anstelle den Datensatz durch einen Querschnitt (Slicing) oder durch eine Oberfläche darzustellen, wird versucht, den gesamten Datensatz direkt zu visualisieren. Ein großer Teil der Informationen in diesem Abschnitt stammt aus dem Buch „The Visualization Handbook“ [15].

Ähnlich wie beim Slicing wird eine Transferfunktion verwendet, um den Voxeln eine Farbe zuzuordnen. Zusätzlich legt die Transferfunktion auch noch einen Wert im Intervall  $[0; 1]$  als Opazität fest, die ausdrückt, wie stark ein Voxel hinter ihm gerenderte Voxel

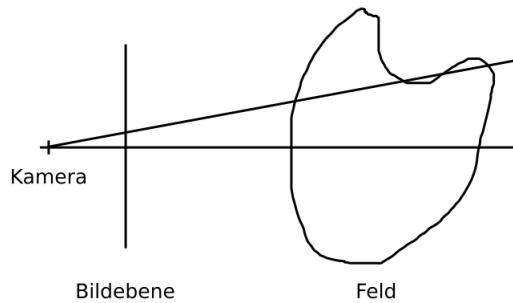


Abbildung 12: Schematische Darstellung der *image-order* Verfahren. Farbwerte an Positionen auf der Bildebene werden mittels Abtastung entlang von Strahlen, die an der Kameraposition beginnen und die Position auf der Bildebene sowie den Datensatz schneiden, bestimmt.

überdeckt, wie schon in Abschnitt 4.6.2 beschrieben. Eine Opazität von 1 bedeutet dabei, dass das Voxel komplett opak ist, eine Opazität von 0 dagegen dass das Voxel vollständig durchsichtig und damit in der Visualisierung unsichtbar ist. Ein Beispiel für ein durch Direct Volume Rendering erzeugtes Bild zeigt Abb. 11.

DVR-Verfahren gliedern sich in vier Gruppen, die als *image-order*, *object-order*, *hybride* und *Domain*-Verfahren bezeichnet werden.

Allen *image-order* Verfahren ist gemein, dass sie eine Position im Raum als ‚Kameraposition‘ festlegen, die Position, von der aus der Benutzer den Datensatz betrachtet. Diese Position befindet sich dabei im gleichen Koordinatensystem wie die Voxel selbst. Vor der Kamera, orthogonal zur Blickrichtung, wird eine imaginäre Ebene platziert, die sogenannte Bildebene. Ihre Größe und Position ist so gewählt, dass sie das gesamte Blickfeld der Kamera einnimmt (siehe Abb. 12). Indem Positionen der Bildebene auf Pixel des Bildschirms abgebildet werden, ist es möglich, durch Einfärbung der Bildebene eine Visualisierung zu erzeugen. Die Farbe eines Pixels ergibt sich dabei, indem ein Strahl berechnet wird, der sowohl durch die Position des Mittelpunkts des Pixels auf der Bildebene als auch durch die Kameraposition verläuft. Entlang des Strahls werden Abtastpunkte gewählt, an denen die Transferfunktion ausgewertet wird. Genau wie beim Slicing können dabei Prä- und Postklassifizierung verwendet werden. Abhängig davon, wie die Farb- und Opazitätswerte der Abtastpunkte kombiniert werden, können unterschiedliche Darstellungen entstehen. Eine Reihe davon wird später im Text vorgestellt. Da die Kombination von Farb- und Opazitätswerten in OpenGL fast immer durch die Funktion des Blendings implementiert wird, verwenden wir den Begriff ‚Blending‘ im Folgenden synonym dazu. Bei manchen Varianten ist es möglich, die Berechnung der Farbe eines Pixels frühzeitig zu beenden, wenn die berechneten Farb- und Opazitätswerte bestimmte Werte annehmen. In diesem Fall wird davon ausgegangen, dass der Einfluss der noch zu berechnenden Abtastpunkte zu klein ist, um vom Benutzer noch wahrgenommen zu werden. Diese frühzeitige Terminierung ist einer der wichtigsten Vorteile der *image-order*

Verfahren.

Im Gegensatz dazu zerlegen *object-order* Verfahren den volumetrischen Datensatz in eine Menge von Basiselementen oder Basisfunktionen, die auf eine Bildebene projiziert werden und dadurch eine Darstellung des Datensatzes bilden. Wenn sich Projektionen der Basiselemente überlappen, was für die meisten Datensätze praktisch der Standardfall ist, werden die Farbwerte abhängig von gewähltem Verfahren mittels Blending kombiniert. Genau wie bei den *image-order* Verfahren können unterschiedliche Varianten des Blendings gewählt werden, was zu unterschiedlichen Visualisierungen führt. Der größte Vorteil von *object-order* Verfahren liegt darin, dass nur die Voxel abgespeichert und verarbeitet werden müssen, die einen skalaren Wert ungleich 0 enthalten. Das ist besonders gut, da die Rechenzeit von *object-order* Verfahren in der Regel direkt von der Anzahl zu verarbeitender Voxel abhängt. Ein Problem liegt darin, dass es für viele Verfahren notwendig ist, die Voxel abhängig von ihrer Entfernung zur Kamera zu sortieren. Diese Sortierung muss aktualisiert werden, wenn die Kamera bewegt wird.

*Hybride* Verfahren versuchen Vorteile von *image-order*, wie die frühzeitige Terminierung, und *object-order* Verfahren, wie den geringen Speicherverbrauch und die Abhängigkeit der Rechenzeit von der Anzahl an Voxeln, zu kombinieren. Die Details der Implementierung unterscheiden sich jedoch stark zwischen den einzelnen Verfahren.

Zuletzt gibt es noch die *Domain*-Verfahren. Dabei wird der komplette Datensatz so transformiert, dass die DVR Visualisierung mittels mathematischer Eigenschaften des Wertebereichs (englisch ‚Domain‘) erzeugt werden kann. Ein Beispiel dafür ist, den Datensatz zunächst mittels der Fourier Transformation in den Frequenzraum zu überführen. Indem im Frequenzraum ein bestimmter zweidimensionaler Ausschnitt gewählt und zurück in den Ortsraum transformiert wird, kann eine Projektion des ursprünglichen Datensatzes erzeugt werden. Ein großer Vorteil von *Domain*-Verfahren liegt häufig in der sehr viel kürzeren Rechenzeit.

Bei einer Eingabe von  $N \times N \times N$  Voxeln hat die bereits erwähnten Variante im Frequenzraum beispielsweise eine Komplexität von  $O(N^2 \log(N))$  statt den bei *object-order* Verfahren üblichen  $O(N^3)$ . Leider schränkt dieses Verfahren die möglichen erzeugbaren Visualisierungen auf ein simples Integral entlang der Sichtstrahlen ein  $O(N^3)$  [15, S. 143]. Es existieren Ergebnisse, die versuchen andere Visualisierungen zu approximieren. Jedoch stellt dies eine deutliche Einschränkung des DVR dar, weshalb im Folgenden *Domain*-Verfahren nicht weiter betrachtet werden.

Es existieren vier populäre Arten von Visualisierungen, die durch DVR-Verfahren erzeugt werden können. In Abb. 13 sind diese Visualisierungen am Beispiel des gleichen Datensatzes, eines menschlichen Kopfes, dargestellt.

**Röntgenbild** Die Bezeichnung ‚Röntgenbild‘ stammt aus der Ähnlichkeit der entstehenden Bilder zu denen, die beim klassischen Röntgen in der bildgebenden Medizin erzeugt werden. Die Färbung eines Pixels ergibt sich in diesem Verfahren dadurch, dass

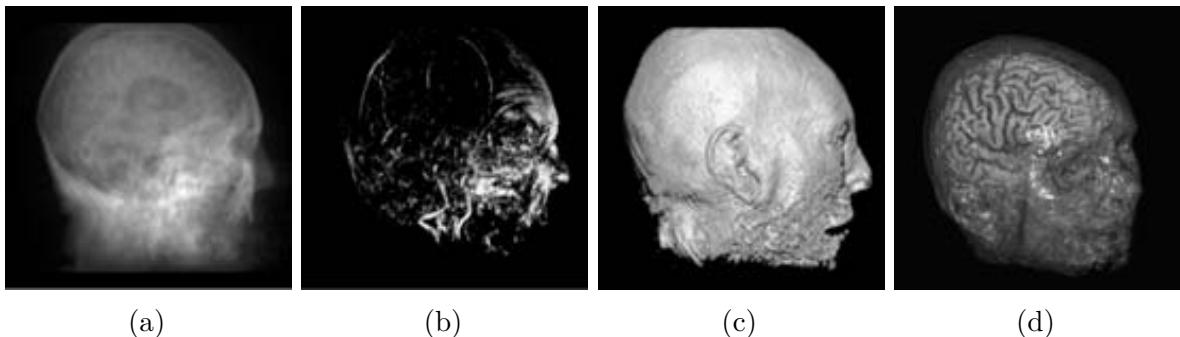


Abbildung 13: Arten von DVR Visualisierungen: (a) Röntgenbild, (b) Maximum Intensity Projection, (c) Isoflächenartig, (d) Transluzent. Entnommen aus [15, S. 134]

die ermittelten Farb- und Transparenzwerte aller zur Färbung des Pixels beitragenden Voxel aufaddiert werden. Damit sind die Voxel gemeint, die z.B. vom selben Strahl durch den Mittelpunkt des Pixels geschnitten werden (*image-order*) oder deren Basisfunktionen sich in diesem Pixel überschneiden (*object-order*). Das Blending entspricht in diesem Fall einer Addition, die die Farbe eines Voxels auf die bisher bestimmte Farbe eines Pixels aufaddiert. Ein Beispiel für ein so erzeugtes Bild ist in Abb. 13a zu sehen.

**Maximum Intensity Projection (MIP)** Bei der MIP wird zunächst das Voxel mit dem höchsten Skalarwert berechnet, der auf diesen Pixel abgebildet wird, z.B. per Strahl oder Projektion der Basisfunktion. Nachdem dieser Wert bestimmt wurde, wird auf ihn die Transferfunktion angewendet und das Ergebnis als Farbe des Pixels festgelegt. Das Blending entspricht einer Maximumsfunktion, die den höchsten bisher für einen Pixel gefundenen Skalarwert und den Skalarwert des Voxels entgegennimmt und das Maximum der beiden ausgibt. Abb. 13b zeigt ein durch MIP erstelltes Bild.

**Isoflächenartig** Wenn die Transferfunktion genau einen Wert  $x$  auf eine Opazität von 1,0 abbildet und alle anderen Werte auf Opazitäten von 0,0, so ähneln die erzeugten Bilder Isoflächen. Blending entspricht hierbei einer Minimumsfunktion, die anstelle der Farbwerte die Distanz zur Kamera zur Entscheidung verwendet. Abhängig vom Verfahren kann auch sofort terminiert werden, wenn ein opakes Voxel gefunden wurde. Dies erschwert jedoch das spätere Hinzufügen von z.B. Schatten. Durch geschickte Wahl der Reihenfolge, in der die Voxel verarbeitet werden, z.B. mit zunehmender Entfernung zur Kamera, kann weitere Beschleunigung erreicht werden. In Abb. 13c ist eine so erzeugte isoflächenartige Darstellung gezeigt.

**Transluzent** Ähnlich wie beim Röntgenbild können auch bei transluzenten Bildern die Transferfunktionen Werte zwischen 0 und 1 annehmen. Dabei werden Voxel, deren Opazität weder 0 noch 1 ist als teilweise transparent dargestellt. Dazu müssen die Farben und Opazitäten von Voxeln, die zur Farbe eines Pixels beitrete, korrekt kombiniert

werden, sodass transparente Voxel die Sicht auf dahinterliegende Strukturen freigeben. Die Formulierung des Blendings hängt hierbei von der Verarbeitungsreihenfolge der Voxel ab. In Abschnitt 5.2 wird dies genauer erläutert. Transluzente Visualisierungen sind besonders populär, da sie sowohl hohe Flexibilität durch Wahl der Transferfunktion aufweisen als auch den Grundgedanken des DVR, das gleichzeitige und vollständige Darstellen des Datensatzes, am besten umsetzen. Der Rest der Arbeit beschäftigt sich deshalb ausschließlich mit Verfahren zur Erzeugung von transluzenten Visualisierungen. Ein Beispiel für ein durch Transluzentes DVR erzeugtes Bild ist in Abb. 13d zu sehen. Transluzente Verfahren stellen eine Verallgemeinerung des Verfahrens zur Erzeugung von Isosflächen dar.

## 5 Direct Volume Rendering Verfahren

Es existieren eine Reihe von DVR-Verfahren, die die in Abschnitt 4.6.3 erläuterten Konzepte umsetzen. Diese Verfahren bieten dabei individuelle Vor- und Nachteile, die in Hinblick auf die Anforderungen der Anwendung verglichen werden müssen. In diesem Kapitel werden dazu zunächst einige der wichtigsten DVR-Verfahren vorgestellt und am Ende miteinander verglichen, um das für die Problemstellung geeignete Verfahren auszuwählen.

### 5.1 Texture Slicing

Eines der ersten *image-order* DVR-Verfahren wurde von Neumann und Cullip [8] entwickelt. Es wird aufgrund der darin verwendeten Mechanismen meist als Texture Slicing bezeichnet, in manchen Veröffentlichungen aber auch als Raycasting. Dieser Begriff wird jedoch auch für ein anderes Verfahren verwendet, das später in dieser Arbeit vorgestellt wird.

Beim Texture Slicing werden mehrere imaginäre Ebenen parallel zur Bildebene durch den Datensatz gelegt. Diese Schnittebenen werden nach einem Verfahren eingefärbt, das praktisch identisch zu dem in Abschnitt 4.6.2 beschriebenen Slicing ist. Die entstandenen Schnittbilder werden danach auf die Bildebene projiziert und durch Blending miteinander kombiniert. Die für das Blending verwendete Formel hängt von der Reihenfolge ab, in der die Schnitte miteinander kombiniert werden. Wenn die Schnitte mit zunehmender Entfernung zur Kamera (front-to-back) verarbeitet werden, entspricht das Blending den Formeln

$$c_n = c_{n-1} + c_{s_n} \cdot (1 - \alpha_{n-1}) \cdot \alpha_{s_n} \quad (59)$$

$$\alpha_n = \alpha_{n-1} + \alpha_{s_n} \cdot (1 - \alpha_{n-1}). \quad (60)$$

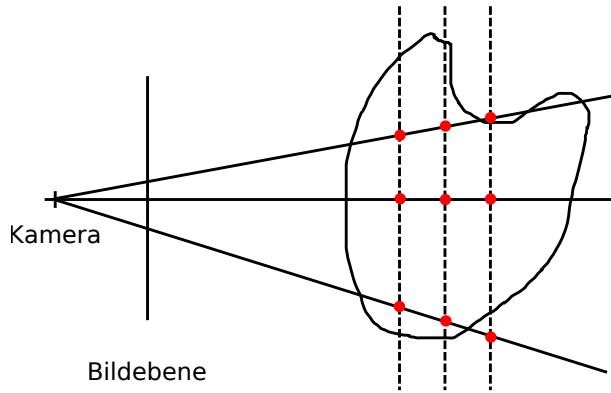


Abbildung 14: Schematische Darstellung des Texture Slicings. Die gestrichelten Linien entsprechen Schnittbildern durch den Datensatz. Die roten Punkte stellen die Positionen auf den Schnittbildern dar, die auf das Pixel des jeweiligen Strahls projiziert werden.

Dabei steht  $c_n$  für die nach dem Blending von  $n$  Schnittflächen akkumulierte Farbe,  $\alpha_n$  für die akkumulierte Opazität. Das Blending wird initialisiert mit den Werten  $c_0 = \alpha_0 = 0$ . Die Parameter  $c_{s_n}$  und  $\alpha_{s_n}$  entsprechen der Farbe und Opazität der  $n$ -ten Schnittfläche. Das Blending wird pixelweise durchgeführt. Wenn sich der Wert von  $\alpha_{n-1}$  in Pixeln nah genug an 1 annähert, können diese in späteren Blending Schritten übersprungen werden, da der Einfluss dieser Blendings auf die Farbe der Pixel sehr klein ist. Dies setzt jedoch voraus, dass das Blending dahingehend konfigurierbar ist, was nicht in allen Versionen von OpenGL oder anderen Renderingsystemen der Fall ist.

Wenn die Schnitte dagegen mit abnehmender Entfernung zur Kamera verarbeitet werden (back-to-front), ergeben sich für das Blending die Gleichungen

$$c_n = (1 - \alpha_{s_n}) \cdot c_{n-1} + c_{s_n} \cdot \alpha_{s_n} \quad (61)$$

$$\alpha_n = (1 - \alpha_{s_n}) \cdot \alpha_{n-1} + \alpha_{s_n}. \quad (62)$$

Dabei ist frühzeitiges Terminieren nicht möglich, weshalb die back-to-front Reihenfolge selten verwendet wird. Die Blending-Formeln werden auch in allen anderen vorgestellten DVR Verfahren verwendet, weshalb sie dort nicht noch einmal explizit vorgestellt werden.

Ein Problem des Texture Slicings liegt darin, dass die gewählte Anzahl der Schnittflächen einen Einfluss auf das erzeugte Bild hat. Wenn innerhalb der gleichen Distanz von der Kameraposition aus mehr Schnittflächen berechnet werden, muss das Blending mehr Summanden aufaddieren. Mehr Schnittflächen führen also zu höheren aufaddierten Opazitäten und damit zu anderen Bildern. Eine oft angewandte Lösung liegt darin, die von der Transferfunktion bestimmte Opazität mit dem Inversen der Anzahl der Schnittflächen zu skalieren.

Die Laufzeit des Texture Slicings ist hauptsächlich abhängig von der Anzahl der einzufärbenden Pixel und der Nummer der zu berechnenden Schnittbilder. Eine höhere Anzahl von Schnittbildern führt jedoch auch zu einer exakteren Repräsentation des Datensatzes, da sich dadurch die Chance, dass ein relevanter Teil des Datensatzes von keinem der Schnitte getroffen wird, reduziert. Durch Interaktionen kann es dem Benutzer ermöglicht werden, den Tradeoff zwischen Rechenzeit und Bildqualität anzupassen.

Der wichtigste Vorteil des Texture Slicings ist die für *image-order* Verfahren relativ geringe Rechenzeit. Sowohl das Erstellen der Schnittbilder als auch die Durchführung des Blendings können effizient von Grafikkarten parallelisiert werden. Ein wichtiger Nachteil ist jedoch, dass die Entfernung der Schnittpunkte von Strahl und Schnittebenen sich von Strahl zu Strahl unterscheiden: Im Zentrum des Sichtfeldes liegen die Schnittpunkte näher zusammen als am Rand. Dies führt dazu, dass die Bildqualität zum Rand hin abnimmt (siehe Abb. 14). Zudem werden dünne, zur Bildebene parallele Strukturen vom Texture Slicing oft zu schwach oder gar nicht dargestellt, da zu wenige Schnittpunkte mit den Schnittebenen existieren.

## 5.2 Raycasting

Historisch entwickelte sich das heutige Raycasting aus den Texture Slicing Verfahren. Es versucht, die Probleme des Texture Slicings, die durch die Verwendung der Schnittebenen entstehen, zu Lasten der Laufzeit zu beheben.

Dazu wird die Bestimmung der Farbwerte der Pixel als approximierte Berechnung von Linienintegralen aufgefasst. Der Datensatz wird zunächst als Wolke von Partikeln aufgefasst, wobei die Partikel Licht emittieren und absorbieren [15, s. 134 f.]. Wie viel Licht Partikel in einem Voxel absorbieren und emittieren sowie die Farbe des emittierten Lichts wird über die Transferfunktion beschrieben. Das aus Richtung der Strahlen zur Kameraposition hin auf der Bildebene eintreffende Licht entspricht dann dem Integral

$$I(x, r) = \int_0^L C(s)\mu(s)e^{-\int_0^s \mu(t)dt}ds. \quad (63)$$

Dabei entspricht  $x$  der Position auf der Bildebene,  $r$  der Richtung des Strahls von der Kameraposition aus durch  $x$ ,  $L$  der Länge des Strahls, bis er zum letzten Mal einen Voxel schneidet,  $C(s)$  dem an Position  $s$  emittierten Licht und  $\mu(s)$  dem Okklusionskoeffizienten an  $s$ , also dem Anteil von Licht, das an  $s$  absorbiert wird, wobei  $\mu(s) \in [0; 1]$ . Somit emittiert jeder Punkt  $p$  eines Strahls Licht und absorbiert gleichzeitig einen Teil des Lichts, das von den anderen Punkten des Strahls, die weiter von der Kamera entfernt sind als  $p$ , emittiert wird. Der Anteil des Lichts, den  $p$  absorbiert, entspricht der Opazität des Voxels, in dem  $p$  liegt.

Da das Integral die Akkumulation von Farbe entlang von Strahlen durch den Datensatz berechnet, wird das Verfahren als Raycasting bezeichnet. Dieses Integral lässt sich im

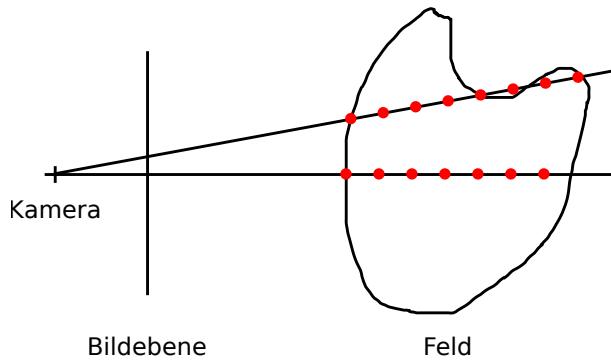


Abbildung 15: Schematische Darstellung des Raycastings. Die roten Punkte stellen Abtastpunkte dar, die in regelmäßigen Abständen auf den Strahlen liegen.

allgemeinen Fall nur mit viel Rechenaufwand, wenn überhaupt, automatisch berechnen [15, S. 136]. Um die Rechenzeit zu verringern, kann das Integral approximiert werden. Durch Vereinfachungsschritte, die hier aus Gründen der Länge übersprungen werden, ergibt sich die Formel

$$I(x, r) = \sum_{i=0}^{L/\Delta s-1} C(i\Delta s) \alpha(i\Delta s) \prod_{j=0}^{i-1} (1 - \alpha(j\Delta s)). \quad (64)$$

Hier entspricht  $\Delta s$  einem Bruchteil von  $L$ . Desto kleiner  $\Delta s$  wird, desto besser wird das Integral approximiert. Die Funktion  $\alpha(i\Delta s)$  ermittelt die Opazität an dem Punkt des Strahls, der  $i\Delta s$  von der Kamera entfernt ist. Die Gleichung kann äquivalent durch zwei rekursive Folgen ausgedrückt werden, die praktisch den Blending-Formeln des Texture Slicings entsprechen, wenn  $c_{s_n}$  durch  $C(n\Delta s)$  und  $\alpha_{s_n}$  durch  $\alpha(n\Delta s)$  ersetzt wird. Auch beim Raycasting wird front-to-back bevorzugt, da frühzeitige Terminierung möglich ist.

Durch Verwendung der rekursiven Gleichungen wird der Farbwert eines Pixels bestimmt ohne Zwischenergebnisse in eine Textur schreiben zu müssen. Daher wird beim Raycasting kein Blending im Sinne der Funktion von OpenGL benötigt. Die Farb- und Opazitätswerte müssen jedoch trotzdem durch Gleichungen kombiniert werden, die äquivalent zu denen beim Blending sind.

Abtastpunkten ausserhalb der Textur wird automatisch eine Opazität von 0 zugeordnet. Deshalb ist es sinnvoll, die Abtastpunkte erst am Rand des ersten vom Strahl geschnittenen Voxels zu beginnen zu lassen. In Abb. 15 wird Raycasting mit dieser Optimierung schematisch dargestellt.

Die Vorteile des Raycastings liegen darin, dass der Abstand der Abtastpunkte auf einem Strahl zueinander gleich bleibt. Im Gegensatz zum Texture Slicing nimmt die Bildqualität

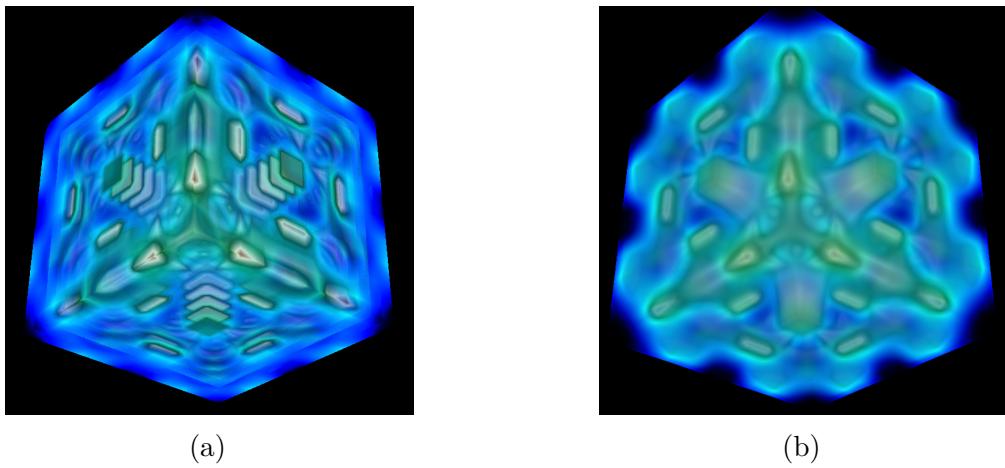


Abbildung 16: Zwei Raycastings desselben Skalarfeldes mit unterschiedlichen Abtastraten. (a) zeigt ein Raycasting mit 20 Abtastpunkten, (b) ein Raycasting mit 200 Abtastpunkten.

also zum Rand hin nicht ab. Die Distanz zwischen Ein- und Austrittspunkt eines Strahls in die 3D Textur geteilt durch die Anzahl der Abtastpunkte innerhalb dieser Distanz wird als ‚Abtastrate‘ bezeichnet.

Die Qualität eines Raycastings ist stark abhängig von der Abtastrate auf den Strahlen und somit vom Abstand zwischen den Abtastpunkten. Ein geringerer Abstand führt generell zu weniger Artefakten und macht kleine Strukturen besser erkennbar, erhöht jedoch den Rechenaufwand. Indem der Nutzer die Abtastrate selbst auswählt, kann er den Tradeoff zwischen Rechenzeit und Bildqualität an seine Bedürfnisse anpassen. In Abb. 16a und 16b sind zwei Darstellungen desselben Skalarfeldes  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ ,  $f(x, y, z) = \cos(x \cdot y \cdot z)$  zu sehen, die durch Raytracing mit einer unterschiedlichen Anzahl von Abtastpunkten erzeugt wurden. Durch die geringe Abtastrate in Abb. 16a entstehen scheibenartige visuelle Artefakte. Durch die höhere Abtastrate in Abb. 16b verschwinden diese Artefakte.

### 5.3 Cell Projection

Cell Projection zählt zu den *object-order* Verfahren. Es basiert auf der Arbeit von Lenz, Gudmundsson, Lindskog und Danielsson [28] sowie der von Frieder, Gordon und Reynolds [13]. Die grundlegende Idee der Cell Projection besteht darin, für jedes Voxel ein zweidimensionales Polygon zu berechnen und dieses auf die Bildebene zu projizieren. Form, Farbe und Transparenz des Polygons werden dabei durch die Form des Voxels sowie die relative Position der Kamera zum Voxel bestimmt. Punkten innerhalb des Polygons werden Dichtewerte relativ zur Dicke des Voxels an diesem Punkt zugeordnet. Die theoretische Grundlage dieses Vorgehens ähnelt stark dem Continuous Scatterplotting, insbesondere dem **Fall 5**.

Wenn sich mehrere Projektionen der Voxel auf der Bildebene überschneiden, werden sie mittels Blending kombiniert. Bei *object-order* Verfahren ist in der Regel keine frühzeitige

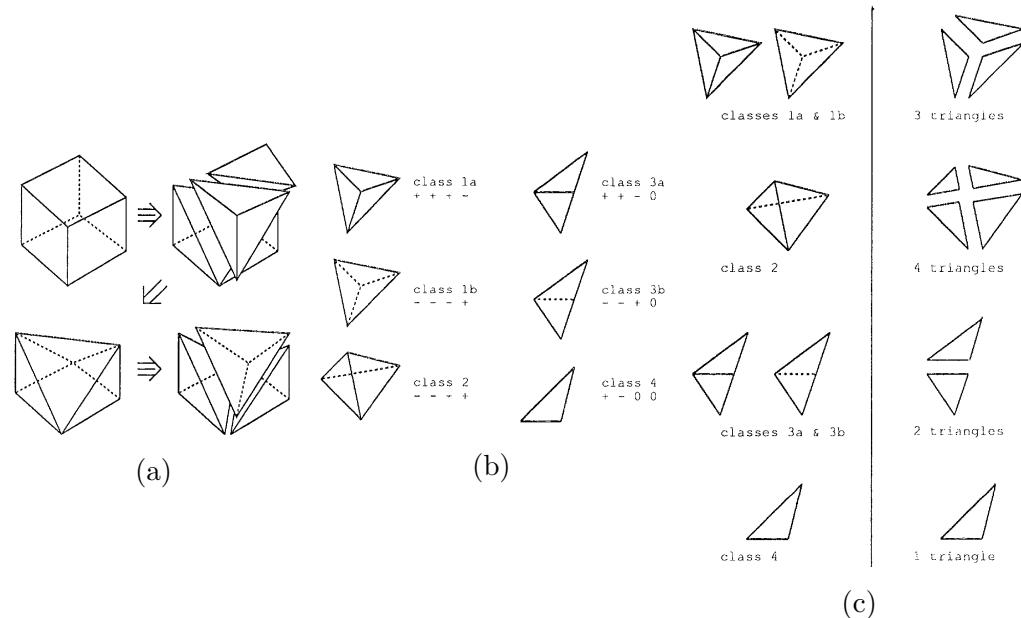


Abbildung 17: Teilaufgaben einer beispielhaften Cell Projection: (a) Zerlegung eines quaderförmigen Voxels in fünf tetraedrische Voxel. (b) Klassifizierung von Tetraeder abhängig von ihrer Ausrichtung zur Kamera. (c) Zerlegung der Projektion des Tetraeders in Dreiecke.

Terminierung möglich, beide Reihenfolgen gleich gut geeignet sind. Es muss jedoch darauf geachtet werden, dass die Voxel in einer konsistenten Reihenfolge verarbeitet werden.

Cell Projection Verfahren unterscheiden sich darin, welche Arten von Voxel sie darstellen können. Im Folgenden wird der von Shirley und Tuchman [36] entwickelte Algorithmus vorgestellt.

Im ersten Schritt zerlegt der Algorithmus die Voxel in disjunkte tetraedrische Teile (siehe Abb. 17a). Die daraus entstehenden Tetraeder werden anhand der Ausrichtung ihrer Flächen zur Blickrichtung der Kamera klassifiziert. Dazu wird das Skalarprodukt zwischen den Normalen der Flächen eines Tetraeders mit der Blickrichtung der Kamera berechnet. Die Klassen sind in Abb. 17b dargestellt. Die Symbole neben den Klassen weisen darauf hin, wie viele Skalarprodukte der Normalen positiv (+), negativ (-) oder 0 (0) sein müssen.

Nachdem die Tetraeder klassifiziert sind, werden die von ihnen durch Projektion auf die Bildebene erzeugten Flächen bestimmt und, abhängig von der Klasse, in Dreiecke zerlegt, wie in Abb. 17c dargestellt. Falls Eckpunkte der Dreiecke innerhalb der Fläche des Tetraeders liegen, wird die Opazität an diesen Punkten abhängig von der Dicke des Tetraeders an dieser Stelle berechnet. Danach werden die Dreiecke auf die Bildebene gerendert. Die Farb- und Opazitätswerte im Inneren ergeben sich durch Interpolation der Eckpunkte.

$\frac{1}{273}$	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>4</td><td>7</td><td>4</td><td>1</td></tr> <tr><td>4</td><td>16</td><td>26</td><td>16</td><td>4</td></tr> <tr><td>7</td><td>26</td><td>41</td><td>26</td><td>7</td></tr> <tr><td>4</td><td>16</td><td>26</td><td>16</td><td>4</td></tr> <tr><td>1</td><td>4</td><td>7</td><td>4</td><td>1</td></tr> </table>	1	4	7	4	1	4	16	26	16	4	7	26	41	26	7	4	16	26	16	4	1	4	7	4	1
1	4	7	4	1																						
4	16	26	16	4																						
7	26	41	26	7																						
4	16	26	16	4																						
1	4	7	4	1																						

Abbildung 18: Eine diskrete Abtastung einer zweidimensionalen Glockenkurve. Die Werte in den Zellen werden mit dem Faktor links multipliziert, sodass die Summe aller Zellen 1 entspricht.

Der Vorteil der Cell Projection liegt in der sehr exakten Darstellung des Datensatzes: Alle Voxel werden in korrekter Form und mit korrekten Dichtewerten gerendert, was bei den meisten anderen *object-order* Verfahren nicht der Fall ist. Diese Exaktheit wird durch zusätzliche Vorberechnungen erreicht, die die Gesamtrechenzeit des Algorithmus erhöhen. Zudem können die harten Kanten der projizierten Polygone visuelle Artefakte erzeugen, die bei anderen *object-order* Verfahren nicht vorkommen.

## 5.4 Splatting

Splatting basiert auf einem von Westover entwickelten Algorithmus[38, 39]. Es zählt, wie auch Cell Projection, zu den *object-order* Verfahren. Im Gegensatz zur Cell Projection wird jedoch nicht versucht den Datensatz so exakt wie möglich darzustellen, sondern das ursprüngliche Feld aus den diskreten Werten der Voxel so gut wie möglich zu rekonstruieren und darzustellen. Das Verfahren gleicht dabei der Rekonstruktion eines kontinuierlichen Signals aus einer diskreten Abtastung, wie sie in der Signalverarbeitung häufig angewendet wird.

Dazu wird eine Abtastung einer zweidimensionalen Gaußschen Glockenkurve an die Position eines Voxels gesetzt. Form, Drehung und Maximum der Glockenkurve werden so gut wie möglich an des Voxel angepasst. Damit nicht für jeden Voxel eine neue Glockenkurve berechnet werden muss, wird zu Beginn des Rendering ein eine Lookup-Tabelle mit den Werten einer Glockenkurve erzeugt. Eine solche Tabelle ist in Abb. 18 dargestellt. Danach werden die Glockenkurven auf die Bildebene projiziert und durch Blending und Interpolation die Farben der Pixel bestimmt.

Splatting bietet gegenüber der Cell Projection und ähnlichen Verfahren den Vorteil, dass im entstehenden Bild keine harten Voxelkanten zu sehen sind. Die aus der Signalverarbeitung übernommenen Methoden garantieren eine hohe Bildqualität mit relativ kurzen Rechenzeiten. Splatting ist daher eines der beliebtesten *object-order* Verfahren. Das Fehlen von harten Kanten stellt jedoch auch einen Nachteil von Splatting dar: Die Grenzen des Datensatzes werden weich gezeichnet. Ein zusätzliches Problem tritt auf,

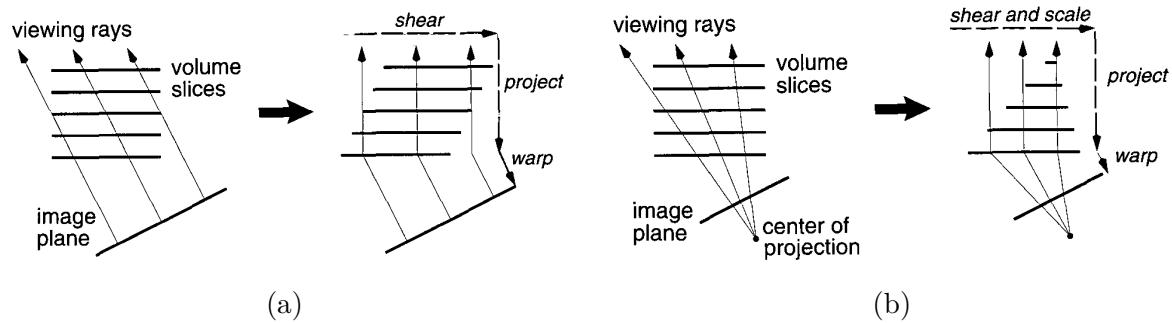


Abbildung 19: Darstellungen der im Shear Warp verwendeten Scherungs- und Verzerrungsoperationen auf dem Datensatz und den Sichtstrahlen in (a) paralleler Projektion und (b) perspektivischer Projektion. Entnommen aus [26].

wenn die Größe der Voxel stark variiert. Wenn beispielsweise einige wenige Voxel sehr viel größer sind als der Rest, können die für diese Voxel erzeugten Glockenkurven deutlich erkennbar sein. Da durch einzelne Glockenkurven die Form eines Voxels nur approximiert wird, kann dies zu deutlichen visuellen Artefakten im erzeugten Bild führen.

## 5.5 Shear Warp

Shear Warp zählt zu den *hybrid*en Verfahren. Es kombiniert Techniken aus *object*- und *image-order* Verfahren, wodurch es typische Vorteile beider Arten bietet. Entwickelt wurde Shear Warp von Philipp Lacroute und Marc Levoy, die es 1994 zuerst vorstellten [26].

Der Grundansatz des Shear Warp besteht darin, den volumetrischen Datensatz mittels einer Scherung (engl. „shear“) so zu transformieren, dass von der Kameraposition ausgehende und die Bildebene schneidende Strahlen parallel zu einer der Koordinatenachsen sind.

Bei orthogonaler Projektion genügt es, die Schichten der 3D Textur untereinander zu verschieben, wie in Abb. 19a dargestellt. Wenn die Projektion dagegen perspektivisch ist, müssen die Schichten zusätzlich so skaliert werden, dass ihre Größe mit zunehmender Entfernung zur Kamera abnimmt. Dies ist in Abb. 19b dargestellt.

Da die von den Pixeln ausgehenden Strahlen parallel zu einer der Koordinatenachsen sind, ist es leicht, die geschnittenen Voxel zu bestimmen. Die Punkte auf einem Strahl unterscheiden sich nur in einer Koordinate. Teure Projektionsoperationen wie bei *object-order* oder schrittweise Abtastung wie bei *image-order* Verfahren werden unnötig.

Pixel mit einer Opazität nah an 1,0 können übersprungen werden, was der frühzeitigen Terminierung bei *image-order* Verfahren entspricht. Zusätzlich können auch Voxel mit einer Opazität von 0,0 übersprungen werden, da diese nicht nur Farbe der Pixel beitragen. Abb. 20 zeigt diese beiden Optimierungen.

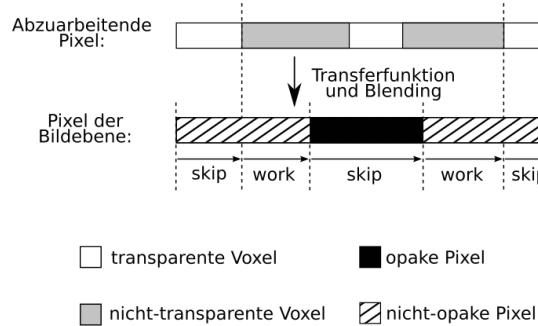


Abbildung 20: Darstellung der optimierten Abbildung von Voxeln auf Pixel. Wenn entweder die Voxel vollständig transparent oder die Zielpixel vollständig opak sind, werden sie übersprungen (‘skip’), ansonsten werden die Voxelwerte auf die Pixel abgebildet (‘work’). Erstellt nach einem Bild aus [26].

Im Gegensatz zu *object-order* Verfahren werden Voxel beim Shear Warp nicht in Basisfunktionen übersetzt, sondern pro Pixel die jeweiligen Werte durch Interpolation bestimmt. Da die Dicke der Voxel von der Blickrichtung abhängt, wird die Opazität mittels einer vorberechneten Tabelle skaliert. Es kann passieren, dass zwei oder mehr Voxel einer Schicht auf ein Pixel abgebildet werden, was korrekt behandelt werden muss.

Zum Abschluss wird eine zur Scherung inverse Verzerrung auf die Bildebene angewendet (engl. ‘warp’), um das endgültige Bild zu erzeugen. Diese Operation ist, wie in Abb. 19a und 19b dargestellt, identisch für parallele und perspektivische Projektion.

## 5.6 Wahl des DVR-Verfahrens

Die vorgestellten Verfahren unterscheiden sich in Hinsicht auf Laufzeit, Speicherverbrauch, Adaptierbarkeit und Qualität der erzeugten Visualisierung. Zum Abschluss des Kapitels werden sie deshalb anhand dieser Eigenschaften und den aus der Aufgabenstellung resultierenden Anforderungen verglichen und eins der Verfahren zur Umsetzung ausgewählt.

*Object-order* Verfahren bieten im Allgemeinen die Vorteile geringen Speicherverbrauchs und relativ geringer Laufzeit. Diese Vorteile werden jedoch durch eine Reihe von Nachteilen erkauft, die sie für die vorliegende Arbeit ungeeignet machen:

**Visuelle Artefakte** Aufgrund der typischen Formen der Basisfunktionen sind *object-order* Verfahren anfällig für die Erzeugung visueller Artefakte. Bei Datensätzen mit Voxeln gleichmäßiger Größe sind die Artefakte meist nicht wahrnehmbar. Wenn einzelne Voxel deutlich größer sind, werden auch die Basisfunktionen sichtbar, womit auch die Sichtbarkeit der Artefakte zunimmt. Durch die Transformation von Voxeln in die Invariantenräume können sehr große Voxel entstehen, weshalb eine Vorverarbeitung und Zerstückelung solcher Voxel notwendig werden würde.

**Teilweise Darstellung von Voxeln** Die Anforderungen spezifizieren Interaktionen, durch die Teile des Invariantenraumes und die diesen Teilen entsprechenden Teile des Objektes ausgeblendet werden. Um dies mit *object-order* Verfahren zu implementieren, müssen komplexe Operationen auf die Basisfunktionen angewendet werden, was die Laufzeit wahrscheinlich deutlich erhöhen würde.

**Keine frühzeitige Terminierung** Durch die Anwendung der Prinzipien des Continuous Scatterplotting entstehen möglicherweise Gebiete mit hoher Dichte. Da *object-order* Verfahren im Allgemeinen keine frühzeitige Terminierung unterstützen, kann dies nicht ausgenutzt werden um die Laufzeit zu verbessern.

Das Problem der teilweisen Darstellung von Voxeln tritt auch bei Shear Warp auf, weshalb es nicht ohne aufwändige Überarbeitung und mit erhöhter Laufzeit verwendet werden konnte.

Damit verbleiben als Optionen nur noch die *image-order* Verfahren. Unterschiede zwischen diesen lassen sich meistens auf einen Tradeoff zwischen Rechenzeit und Bildqualität zurückführen. Raycasting bietet dabei eine Möglichkeit für den Benutzer diesen Tradeoff selbst zu beeinflussen und, falls entsprechend eingestellt, die besten Ergebnisse. Auch die teilweise Darstellung von Voxeln ist leicht in Raycasting zu implementieren, indem die Werte der Varianten an Abtastpunkten interpoliert werden und Punkte mit Werten außerhalb des gewählten Bereichs übersprungen werden. Daher wird für diese Arbeit Raycasting als DVR-Verfahren gewählt.

## 6 Umsetzung

In diesem Kapitel wird beschrieben, wie die Grundlagen, Technologien und Verfahren aus den vorherigen Kapiteln eingesetzt wurden, um eine interaktive Tensorfeldvisualisierung in FAnToM zu erzeugen.

Die Grundidee der entwickelten Visualisierung besteht darin, zwei DVR zu erzeugen. Das Erste stellt die Zellen des ursprünglichen Feldes dar, über die eine konstante Dichte angenommen wird. Da das Zielgebiet der Anwendung die Materialforschung ist, wird das ursprüngliche Feld im Ortsraum im Folgenden als ‚Objekt‘ bezeichnet. Für das

zweite DVR werden die drei Invarianten eines Invariantensatzes der an den Eckpunkten der Zellen definierten Tensoren berechnet und diese als Koordinaten interpretiert. Die Zellstruktur wird somit in einen ‚Invariantenraum‘ überführt. Wegen der Popularität der K- und R-Invariantensätze werden die Koordinaten optional in ein zylindrisches Koordinatensystem übertragen. Die Dichten in den Zellen der neuen Zellstruktur werden durch Methoden des Continuous Scatterplotting berechnet.

Das Verfahren zur Erzeugung der Darstellungen und Interaktionen ist im Folgenden näher erläutert.

## 6.1 Umsetzung des Continuous Scatterplotting

Um ein vollständiges Continuous Scatterplotting zu implementieren, müssen drei Teilaufgaben erfüllt werden.

Als Erstes müssen die Volumen der Tetraeder im Orts- und Invariantenraum berechnet werden. Aus der Volumenänderung der Zellen bei der Abbildung vom Orts- in den Invariantenraum lassen sich die Dichten  $\sigma_{V_i}$  der Tetraeder  $V_i$  im Invariantenraum bestimmen. Dabei wird die für **Fall 1** angegebene Gleichung (55) verwendet. Die anderen Fälle können zwar theoretisch vorkommen, sind jedoch extrem unwahrscheinlich. Durch unvermeidbare Rundungsfehler werden Bereiche konstanter Invarianten (**Fall 2**), Nullmengen (**Fall 3**) und Abbildungen auf Teilräume mit geringerer Dimension (**Fall 5**) zerstört. Das Erkennen von diesen Rundungsfehlern wird dadurch erschwert, dass in den verwendeten Datensätzen tatsächlich Bereiche mit sehr geringen Veränderungen der Invarianten oder Abbildungen auf sehr flache Tetraeder vorkommen. Die Unterscheidung zwischen solchen extremen Formen von **Fall 1** und den anderen Fällen ist nicht ohne Weiteres möglich. **Fall 4** kommt nicht vor, da die verwendeten Felder überall differenzierbar sind.

Als zweites müssen die Dichtewerte in Bereichen in denen sich mehrere Zellen überlappen korrekt berechnet werden. Da die  $V_i$  eine vollständige Zerlegung des Feldes darstellen, lässt sich Gleichung (53) anwenden. Die  $\sigma_V$  entsprechen den berechneten Dichtewerten pro Tetraeder, die für jedes Voxel durch das Blending aufaddiert werden. Das Abspeichern der Ergebnisse dieser Teilaufgabe ist nicht trivial, da ihre Form dem Schnittraum beliebig vieler Tetraeder entsprechen kann.

Und als drittes muss das Continuous Scatterplotting in ein Format transformiert werden, das als Eingabe des später implementierten Raycastings dienen kann. Um Rechenzeit zu sparen sollte versucht werden, Teilaufgabe 2 und 3 zu kombinieren.

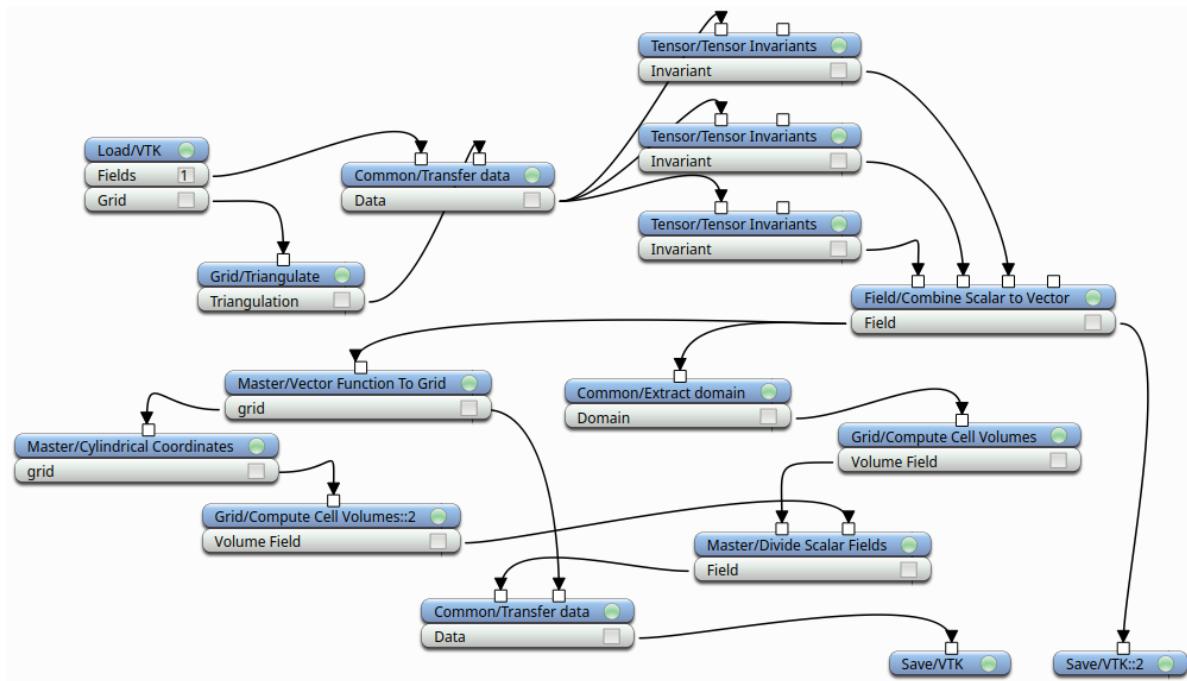


Abbildung 21: Der Flowgraph der FAnToM Session, die zur Vorbereitung der Daten verwendet wird.

## 6.2 Datenvorbereitung

### 6.2.1 Die Vorbereitungssession

Bevor die Visualisierung beginnen kann, müssen die Daten vorbereitet werden. Um Zeit zu sparen, wird dieser Schritt für jeden Datensatz nur einmal durchgeführt und die Ergebnisse abgespeichert. Ein Screenshot der für die Datenvorbereitung verwendeten Session ist in Abb. 21 zu sehen. Die in der Vorbereitungssession durchgeführten Berechnungen sind im folgenden kurz beschrieben. Die verantwortlichen Algorithmen werden in Klammern angegeben. Fast alle der verwendeten Algorithmen sind standardmäßig in FAnToM vorhanden. Die restlichen 3, ‚VectorFunctionToGrid‘, ‚CylindricalCoordinates‘ und ‚Divide Scalar Fields‘, sind sehr einfache Algorithmen, die für die vorliegende Arbeit implementiert wurden.

Die im VTK Format[2] vorliegenden Daten werden geladen (‚Load/VTK‘) und die Zellstruktur, auf der das Feld definiert ist, tetraedrisiert (‚Extract Domain‘, ‚Triangulate Grid‘ und ‚Transfer Data‘). Die Invarianten der Tensoren an den Eckpunkten der Zellen werden berechnet und zu einem Vektorfeld kombiniert (‚Tensor Invariants‘ und ‚Combine Scalar to Vector‘). Eine Kopie dieses Vektorfeldes wird abgespeichert (‚Save VTK‘), um als erste Eingabe der Visualisierungssession zu dienen. Indem das Vektorfeld als Koordinaten der Punkte interpretiert wird, wird eine Kopie der Zellstruktur im Invariantenraum erzeugt (‚Vector Function To Grid‘). Optional können für Invarianten, die sich gut in zylindrischen

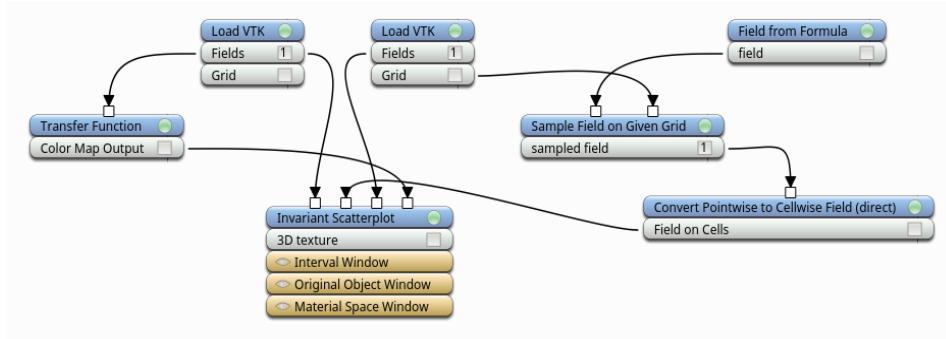


Abbildung 22: Der Flowgraph der Visualisierungssession.

Koordinaten darstellen lassen, die Punkte als zylindrische Koordinaten interpretiert und in kartesische Koordinaten umgerechnet werden (‘Cylindrical Coordinates’). Dadurch wird eine einheitliche Darstellung des Datensatzes in kartesischen Koordinaten für die Visualisierungssession geschaffen, wodurch Rechenzeit gespart werden kann. In jedem Fall werden die Volumen Zellen im Orts- und Invariantenraum berechnet (‘Compute Cell Volumes’). Unter der Annahme, dass die Dichten innerhalb der Zellen im Ortsraum konstant 1,0 waren, ergeben sich die Dichten im Invariantenraum als Quotient der Volumen im Ortsraum und der im Invariantenraum (‘Divide Scalar Fields’). Am Ende werden die berechneten Dichtewerte der Zellstruktur im Invariantenraum zugewiesen und das Ergebnis abgespeichert, um als zweite Eingabe der Visualisierungssession zu dienen.

### 6.2.2 Die Visualisierungssession

Zwei Teile der Datenvorbereitung müssen in der gleichen Session stattfinden wie die Visualisierung selbst, um dem Anwender die Möglichkeit der Interaktion zu geben. Das sind zum einen die Erzeugung der Transferfunktion, zum anderen die Berechnung der Dichte innerhalb der Voxel im Objekt. Die Session ist in Abb. 22 zu sehen.

Der linke ‘VTK Load’ Algorithmus lädt dabei das skalare Dichtefeld im Invariantenraum, der rechte das Vektorfeld auf den ursprünglichen Punkten, in dem die Invarianten gespeichert sind. Vom Algorithmus ‘Transfer Function’ wird eine Transferfunktion für das Feld im Invariantenraum erzeugt. Wie in Abb. 10 beschrieben, ist diese vom Nutzer frei konfigurierbar und wird in Echtzeit auf die Darstellung angewendet. Die Dichten der Voxel im Objekt werden durch eine Funktion bestimmt, die in den Algorithmus ‘Field from Formula’ eingegeben wird (standardmäßig konstant 1). Die geladenen VKT Dateien, die Transferfunktion und die neuen Voxeldichten werden am Ende dem Visualisierungsalgorithmus übergeben

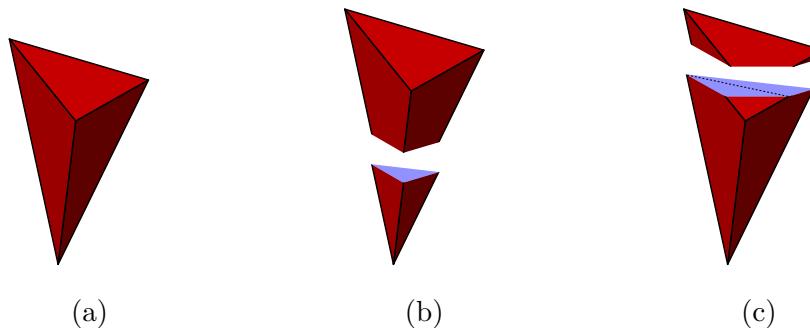


Abbildung 23: Darstellungen von möglichen Schnitten durch einen Tetraeder. (a) Der Tetraeder ohne Schnitte. (b) Ein Schnitt, der durch ein einzelnes Dreieck dargestellt werden kann. (b) Ein Schnitt, der ein Viereck bildet. Die gestrichelte Linie deutet eine mögliche Unterteilung in Dreiecke an.

### 6.3 Voxelisierung

Um Teilschritte 2 und 3 des beschriebenen Verfahrens zur Umsetzung eines Continuous Scatterplotting zu implementieren, wird aus den Tetraedern mit den in der Vorverarbeitung berechneten Dichtewerten eine 3D Textur erzeugt. Eine 2D Textur kann im Allgemeinen als Tabelle aufgefasst werden, in der Farbwerte codiert sind. 3D Texturen setzen sich aus vielen 2D Texturen zusammen, deren Felder die Eigenschaften von Voxeln einer Schicht der 3D Textur beschreiben.

Um die Tetraeder und ihre Dichtewerte durch Positionen und Eigenschaften von Voxeln zu repräsentieren, wird ein Verfahren implementiert, das für jedes Voxel die geschnittenen Tetraeder bestimmt und deren Dichtewerte auf die Dichte des Voxels aufaddiert. Dies wird als Voxelisierung bezeichnet. Initial ist die Dichte eines Voxels 0. FAnToM besitzt zwar eine effiziente Funktion, um Werte innerhalb der Zellen eines Feldes zu berechnen, diese ist jedoch nicht in der Lage, mit selbstdurchdringenden Feldern, wie sie bei der Überführung in den Invariantenraum entstehen können, umzugehen. Es war deshalb notwendig, eine effiziente Methode zur Erzeugung der 3D Texturen zu implementieren.

OpenGL stellt verschiedene Optionen für Farbcodierung zur Auswahl, womit Speicherbedarf und Präzision eingestellt werden können. Für die vorliegende Arbeit wurde „GL\_RGBA16“ gewählt, die vier Werte mit jeweils 16 Bit Präzision zur Verfügung stellt. Üblicherweise wird in einem Feld der Tabelle die Werte für den roten, grünen und blauen Anteil der Farbe sowie die Transparenz („Alpha“) an dieser Stelle codiert. Das DVR-Verfahren interpretiert diese Werte jedoch anders: Der Alphawert entspricht der Dichte und die restlichen drei Farbwerte den interpolierten Invarianten der Zelle, in der das Voxel liegt. Die Invarianten spielen allerdings nur bei der Visualisierung im Ortsraum eine Rolle, weshalb die Werte für Rot, Grün und Blau in der Textur im Invariantenraum leer gelassen werden.

Aufgrund der leichten Parallelisierbarkeit bot es sich an, die Voxelisierung durch OpenGL

und somit auf der Grafikkarte durchführen zu lassen. Das Verfahren basiert auf der Arbeit von Rueda et al.[34]. Dabei werden Schichten der 3D Textur berechnet, indem die Schnittflächen zwischen den Tetraedern und Ebenen bestimmt werden. Die Ebenen verlaufen dabei durch die Mittelpunkte der Voxel einer Schicht der Textur. Da der Schnitt zwischen einer Ebene und einem Tetraeder immer entweder ein Dreieck oder ein Viereck ist, kann er als Folge von einem oder zwei Dreiecken dargestellt werden (siehe Abb. 23). Diese Dreiecke wiederum werden durch den standardmäßigen Rasterisierer von OpenGL in die fertigen Voxel unterteilt. Das komplette Verfahren ist in Form einer Shaderpipeline implementiert, die im Folgenden kurz erläutert wird. Falls der Shader nicht-triviale Algorithmen enthält, ist zusätzlich der Algorithmus in vereinfachter Form angegeben.

Die Eingabe des Vertex-Shaders besteht aus den Punkten der Tetraeder, ihren jeweiligen Dichten und optional den an den Punkten berechneten Invarianten. OpenGL lässt allerdings in den von FAnToM unterstützten Versionen keine Tetraeder als Eingabe zu. Deshalb müssen diese auf eine andere Art übergeben werden. Eine Möglichkeit dazu sind Liniensegmente mit Nachbarschaft (‘GL\_LINES\_ADJACENCY\_EXT’). Das Format von Liniensegmenten mit Nachbarschaft besteht aus vier Punkten pro Primitive: Der erste Punkt entspricht dem Beginn des vorhergehenden Liniensegments, der zweite und dritte Punkt entsprechen dem aktuellen Liniensegment und der vierte Punkt ist der Endpunkt des nächsten Liniensegments. Da es immer genau vier Punkte pro Primitive gibt, eignet das Format sich auch, um Tetraeder an die Pipeline zu übergeben.

Da die zylindrische Darstellung in Richtung der x-Achse sehr groß werden kann, wurde ein Verfahren entwickelt und implementiert, das das Feld entlang der x-Achse unterteilt und in mehrere einzelne Texturen voxelisiert. Dasselbe Verfahren kann theoretisch auch für die anderen Richtungen implementiert werden, das war jedoch für keinen der verwendeten Datensätze nötig.

Wie schon im Abschnitt 4.2 angesprochen, ist es möglich, die Ausgabe der Renderpipeline in einer Textur zu speichern. Dazu wird ein Framebuffer verwendet, der Schichten der 3D Textur als Ausgabe der Pipeline bindet. Die Voxelisierung wird also für jede Schicht einzeln durchgeführt.

Die Anzahl an Voxeln pro Dimension in der 3D Textur kann vom Nutzer selbst festgelegt werden. Falls in Richtung der x-Achse ein Wert über 2048 gewählt wird, werden mehrere Texturen erzeugt. 2048 ist die in der von FAnToM verwendeten OpenGL Version die maximale Ausdehnung einer 3D Textur in jede Richtung.

Eine wichtige Eigenschaft der Voxelisierung liegt darin, dass die Komplexität des Datensatzes auf ein vom Benutzer festgelegtes Maß reduziert wird. Unabhängig davon, aus wie vielen Zellen der zu visualisierende Datensatz besteht, wird immer die vom Benutzer vorgegebene Anzahl von Voxeln erzeugt. Dies bringt Vor- und Nachteile mit sich.

Zu den Vorteilen gehört, dass die Voxelisierung nur ausgeführt werden muss, wenn neue Daten geladen werden oder sich Parameter der Voxelisierung selbst ändern. Die Laufzeit des Raycastings ist ausschließlich abhängig von der Anzahl der einzufärbenden Pixel und der Menge an Abtastpunkten pro Strahl. Durch diese beiden Eigenschaften ist

es möglich, auch sehr große Datensätze explorativ zu analysieren, nachdem sie einmal voxelisiert wurden. Dadurch stellt die Voxelisierung einen wichtigen Ansatz dar, um **Herausforderung 2** aus der Einleitung, die Menge der Daten pro Datensatz, zu lösen.

Dafür werden eine Reihe von Nachteilen in Kauf genommen. Ein Nachteil liegt darin, dass durch die Voxelisierung leicht Stufenartefakte entstehen. Dieser Effekt tritt im zweidimensionalen Fall häufig auf und wird dort mit speziellen Verfahren abgemildert. Varianten dieser Verfahren im Dreidimensionalen könnten möglicherweise auch auf die 3D Texturen angewendet werden, dies ist jedoch im Verlauf der vorliegenden Arbeit noch nicht untersucht worden. Weitere Probleme treten auf, wenn Zellen sehr klein sind und dadurch nur auf wenige Voxel abgebildet werden. In diesem Fall gehen Informationen über die Form der Zellen verloren.

Die Dichte innerhalb eines Voxels entspricht der Summe der Tetraeder die das Voxel schneiden, unabhängig davon, wie viel des Voxels vom Tetraeder eingenommen wird. Dadurch enthält die 3D Textur insgesamt mehr Masse als das ursprüngliche Feld, was gegen einen Grundsatz des Continuous Scatterplotting, die Masserhaltung, verstößt. Eine Möglichkeit diesen Fehler zu reduzieren oder sogar komplett zu vermeiden wäre es, pro Voxel die Dichtewerte an mehreren Punkten innerhalb des Voxels zu berechnen und die Dichte des Voxels auf den Mittelwert der einzelnen Dichtewerte zu setzen. OpenGL bietet standardmäßig eine Option dieses Verfahrens, das als Supersampling bezeichnet wird, zu implementieren. Da aber Supersampling die Rechenzeit der Voxelisierung stark erhöhen würde und die Voxelisierung bereits einen Großteil der Rechenzeit der Visualisierung ausmacht, wurde dies noch nicht implementiert. Teilaufgabe 2 der Umsetzung des Continuous Scatterplotting ist daher erfüllt, die Implementierung erzeugt jedoch bei kleinen Tetraedern und an Rändern von Tetraedern fehlerhafte Werte.

Die durch die Voxelisierung erzeugte 3D Textur kann direkt an das Raycasting weitergegeben werden, wodurch Teilaufgabe 3 des Continuous Scatterplotting erfüllt ist.

**Vertex-Shader** Der Vertex-Shader erhält als Eingabe die Eckpunkte der Tetraeder, codiert in Form von Liniensegmenten mit Nachbarschaft. Falls das Feld in mehrere Texturen voxelisiert werden soll, wird eine Verschiebung und Skalierung durchgeführt, um nur die korrekten Tetraeder zu voxelisieren. Dichten und Invarianten werden als Variablen an den nächsten Teil der Shaderpipeline weitergegeben.

**Geometry-Shader** Die von FAnToM unterstützte OpenGL Version unterstützt keine Geometry-Shader. Mithilfe einer Erweiterung kann das jedoch umgangen werden.

Der Geometry-Shader bekommt als Eingabe alle Punkte eines Primitive und die vom Vertex-Shader übergebenen Variablen für diese Punkte. Ziel des Geometry-Shaders ist es, die Tetraeder mit Ebenen, die parallel zur xy-Ebene verlaufen, zu schneiden und die Schnittflächen als Dreiecke zurück in die Renderpipeline zu übergeben.

Dazu berechnet der Geometry-Shader zunächst die Schnittpunkte der Tetraederkanten mit der Ebene. Wenn mindestens drei Schnittpunkte existieren, werden daraus Dreiecke konstruiert und weitergegeben. Wenn vier Schnittpunkte gefunden wurden, müssen sie zuvor noch nach ihrer Position im Raum geordnet werden, um daraus zwei überlappungsfreie Dreiecke zu konstruieren und weiterzugeben. In Algorithmus 1 wird das Verfahren dargestellt.

```

Data :  $p_i$  Eckpunkte des Tetraeders,  $z$  Parameter der Schnittebene
Result : Schnittfläche zwischen Tetraeder und Schnittebene
schnittZahl  $\leftarrow 0$ ;
schnittPunkte  $\leftarrow []$ ;
for  $\forall (p_m, p_n), n < m$  do // Für jede Kante des Tetraeders
|  $p \leftarrow \text{schnittpunktBerechnen}(z, p_m, p_n)$ ;
| if  $p$  liegt zwischen  $p_m, p_n$  then
| | schnittPunkte.append(p);
| | schnittZahl++;
| end
end
if schnittZahl == 4 then
| sortieren(schnittPunkte);
end
erzeugeDreiecke(schnittPunkte);

```

**Algorithmus 1 :** Die Berechnung der Tetraederschnittflächen im Geometry-Shader.

**Fragment-Shader** Der Fragment-Shader normalisiert die Invarianten auf das Intervall  $[0; 1]$ , wobei 0 dem niedrigsten und 1 dem höchsten vorkommenden Wert entspricht. Damit wird sichergestellt, dass die Invarianten mit maximal möglicher Präzision in der Textur gespeichert werden.

Zudem wird die Dichte der Voxel modifiziert. Wenn der maximale vorkommende Dichtewert im Voraus bekannt ist, kann die Dichte ähnlich wie die Invarianten auf das Intervall  $[0; 1]$  normalisiert werden. Dies ist jedoch durch die mögliche Überlappung beliebig vieler Tetraeder kein triviales Problem. Wie es gelöst wird, ist in Abschnitt 6.4 beschrieben. Die Dichtewerte können auch durch eine Reihe von Optionen vom Benutzer manipuliert werden. Erstens kann sie mit einem festgelegten Faktor, standardmäßig 1, multipliziert werden, um die Dichte der gesamten 3D Textur zu erhöhen, falls gewünscht. Zweitens kann vom Nutzer eine minimale Dichte im Intervall  $[0; 1]$  festgelegt werden, unter der kein Dichtewert eines von einem Tetraeder getroffenen Voxels liegen kann. Standardmäßig ist die minimale Dichte 0. Und drittens kann die dritte Wurzel aus der berechneten Dichte gezogen werden. Dies hat einen erheblichen Vorteil bei Datensätzen, in denen sowohl sehr große als auch sehr kleine Dichtewerte vorkommen, da alle Werte an 1 angenähert werden. Datensätze dieser Art sind nicht selten, teilweise betragen die Unterschiede der

Dichtewerte 10 oder mehr Größenordnungen. Da die Dichtewerte nicht direkt abgelesen werden müssen, führt das Ziehen der Wurzel zu keinen erheblichen Problemen in der Interpretierbarkeit der Darstellung.

Durch den Fragment-Shader werden im Invariantenraum mehrere der erzeugten Dreiecke auf dieselben Voxel abgebildet. Die Kombination der Werte pro Voxel erfolgt durch die schon mehrmals erwähnte von OpenGL bereitgestellte Funktion des Blendings. Im Objekt können Tetraeder nicht überlappen, weshalb die Invariantenwerte in der Textur davon nicht betroffen sind. Durch das Blending wird  $\sigma_{V_i}$  für jeden Tetraeder  $V_i$  berechnet und pro Voxel aufaddiert.

## 6.4 Berechnung der maximalen Dichte

Texturen können nur Werte im Intervall  $[0; 1]$  enthalten. Um andere Werte darstellen zu können, muss ein Faktor  $f$  ermittelt werden, durch den die Werte, die in der Textur gespeichert werden sollen, geteilt werden. Wenn wiederum aus der Textur gelesen werden soll, werden die in der Textur gespeicherten Werte mit diesem Faktor multipliziert, um die ursprünglichen Werte zu ermitteln.

Um die Präzision der Textur möglichst gut auszunutzen, sollte  $f$  dem Maximum der in der Textur gespeicherten Werte entsprechen. Dies stellt die Voxelisierung vor ein Problem: Das Maximum der 3D Textur kann erst bestimmt werden nachdem die Voxelisierung abgeschlossen ist, ist jedoch nötig, um die Voxelisierung korrekt durchzuführen. Um dieses Problem zu lösen wurde in dieser Arbeit ein Verfahren entwickelt, um das korrekte Maximum schon im Voraus zu berechnen.

In einem Vorverarbeitungsschritt wird die höchste potentiell mögliche Dichte in einem Voxel berechnet. Diese käme genau dann vor, wenn alle Tetraeder  $V_i, i = 1, \dots, n$  sich in einem Voxel überschneiden würden. Somit entspricht sie der Summe der Dichten aller Tetraeder.

Danach wird die Voxelisierung einmal ausgeführt, wobei jeder Dichtewert durch die berechnete maximal mögliche Dichte geteilt und somit garantiert in das Intervall  $[0; 1]$  abgebildet wird.

Als Nächstes wird das Maximum der gerade erzeugten 3D Textur gesucht. Da die 3D Textur relativ groß werden kann (bis zu mehreren Gigabyte), die Übertragung von Daten aus der Grafikkarte zum Arbeitsspeicher relativ langsam ist und das Finden des Maximums gut parallelisiert werden kann, wurde auch dieses Problem durch eine Shaderpipeline gelöst.

Die Daten in der 3D Textur sind nicht geordnet und es existiert auch keine andere Struktur, durch die die Berechnung des Maximums vereinfacht werden könnte. Deshalb musste sie als lineare Suche implementiert werden. Um die potentiell hohe Laufzeit von  $O(n)$  zu verringern, wurde die Suche mithilfe eines ‚Divide-and-Conquer‘ Ansatzes parallelisiert. Divide-and-Conquer beschreibt ein häufig angewendetes Verfahren zur

Lösung von Problemen. Dabei werden komplexe Probleme in einfachere Teilprobleme aufgeteilt, deren Lösungen so kombiniert werden, dass sie eine Lösung des ursprünglichen komplexen Problems darstellen [20].

In diesem konkreten Fall wird das komplexe Problem, die Berechnung der maximalen Dichte in der 3D Textur, in einfachere Teilprobleme, die Berechnung der maximalen Dichten für jede einzelne Schicht, zerlegt. Diese Teilaufgaben sind voneinander unabhängig und können deshalb parallel gelöst werden.

Die Eingabe der Shaderpipeline ist die 3D Textur, sowie eine Linie zwischen den Punkten  $(-1,0; 0,0; 0,0)$  und  $(1,0; 0,0; 0,0)$ . Berechnet wird eine Darstellung der Linie aus  $z$  Pixeln, wobei  $z$  die Anzahl von Schichten der 3D Textur ist. Der Wert jedes Pixels entspricht dem Maximum der jeweiligen Schicht. Durch Verwendung eines Framebuffers wird das Ergebnis in eine Textur gerendert, die erheblich weniger Speicherplatz verbraucht als die 3D Textur. Diese wird zurück in den Arbeitsspeicher geladen und das Maximum durch linearen Durchlauf gefunden.

Indem das Maximum der 3D Textur, ein Wert zwischen 0 und 1, mit dem Wert der größten möglichen Dichte multipliziert wird, erhält man die korrekte maximale Dichte. Diese wird verwendet, um die Werte einer weiteren Voxelisierung zu normalisieren. Dadurch wird die Präzision in der 3D Textur erhöht.

**Data :**  $T$  3D Textur,  $z$  interpolierte z-Koordinate der Schicht,  
 $x_{max}, y_{max}$  Ausdehung der Schicht in x- und y-Richtung

**Result :** Maximum der Schicht

```

max ← 0;
for x ← 0 to  $x_{max}$  do
    for y ← 0 to  $y_{max}$  do
        p ← punktAnKoordinaten( $\frac{x}{x_{max}}$ ,  $\frac{y}{y_{max}}$ , z)
        voxelDichte ← texturwertAnPunkt( $T$ , p)
        max ← maximum(max, voxelDichte)
    end
end
return max;
```

**Algorithmus 2 :** Die Bestimmung der Maxima aller Schichten im Fragment-Shader.

**Vertex-Shader** Der Vertex-Shader ordnet  $(-1,0; 0,0; 0,0)$  den Wert 0 und  $(1,0; 0,0; 0,0)$  den Wert 1 zu und gibt diese Werte an den Fragment-Shader weiter.

**Fragment-Shader** Für jedes Fragment werden die Werte der Endpunkte der Linie aus dem Vertex-Shader interpoliert. Diese Werte entsprechen der Schicht, deren Maximum berechnet werden soll. Danach wird über alle Voxel dieser Schicht iteriert und das

Maximum der Schicht zurückgegeben. Beim Auslesen der Werte aus der 3D Textur ist Interpolation deaktiviert, damit die Werte nicht verfälscht werden. Algorithmus 2 stellt die Funktion des Fragment-Shaders dar.

## 6.5 Umsetzung des Raycastings

Für die beiden DVR Darstellungen wird eine Variante des in Abschnitt 5.2 beschriebenen Raycastings verwendet. Dies geschieht wiederum in Form einer Shaderpipeline.

Die Eingabe der Pipeline ist die vorberechnete 3D Textur und ein Rechteck bestehend aus zwei Dreiecken. Das Rechteck entspricht der Bildebene in Abschnitt 5.2. Die Bildebene befindet sich so vor der Kamera, dass sie das gesamte Blickfeld einnimmt. Das Raycasting wird durchgeführt, indem die Shader die Farben der Pixel des Rechtecks, also der Bildebene, berechnen.

Um den räumlichen Eindruck des DVR zu erhöhen, wird eine perspektivische Projektion mit einem Blickfeld von 90° verwendet.

**Vertex-Shader** Im Vertex-Shader werden lediglich die x- und y-Koordinaten der Punkte in einer Variablen gespeichert und an den Fragment-Shader übergeben.

**Fragment-Shader** Die ‚Model-View-Projection Matrix‘, kurz MVP, ist eine Matrix, die die Abbildung von Punkten im Raum auf die Bildebene ausdrückt. Dazu gehören Translationen und Rotationen der Szene sowie die perspektivische Projektion von 3D zu 2D. Mithilfe des Inversen der MVP lässt sich für jeden Punkt auf der Bildebene ein Strahl konstruieren, der diesen Punkt und die Kameraposition schneidet.

Danach wird der Schnitt zwischen den die 3D Textur begrenzenden, achsenparallelen Ebenen, genannt ‚Bounding Box‘ und diesem Strahl berechnet. Aus der Reihenfolge, in der der Strahl die Ebenen der Bounding Box schneidet, kann bestimmt werden, ob die 3D Textur vom Strahl durchquert wird oder nicht. Falls nicht, kann sofort die Hintergrundfarbe als Farbe des Fragments zurückgegeben werden. Falls der Strahl die 3D Textur schneidet, wird der Abstand zwischen den Abtastpunkten berechnet.

Um Artefakte zu verhindern, kann pro Strahl optional ein pseudo-zufälliger Abstand berechnet werden. In diesem Fall beginnen die Abtastpunkte erst in diesem Abstand zum Punkt, an dem der Strahl die 3D Textur zum ersten mal schneidet.

Die eigentliche Berechnung der Fragmentfarbe geschieht in einer Schleife. Für jeden Abtastpunkt entlang des Strahls wird die Dichte an diesem Punkt gemessen, durch die Transferfunktion in Farbe und Transparenz übersetzt und nach **????** zu einem Farbwert kombiniert.

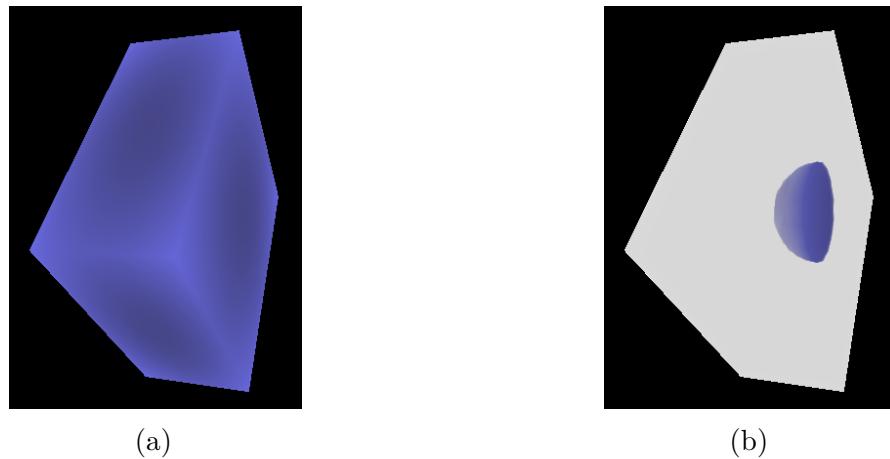


Abbildung 24: Ergebnisse des Raycastings auf einem quaderförmigen Feld von  $3 \times 3$  Matrizen. In (a) werden alle Voxel des Feldes in Blau angezeigt, in (b) nur ein kleiner, halbkugelförmiger Teilbereich. Zu beachten ist, dass in (a) die Farben von Punkten näher am Zentrum des Feldes dunkler und in (b) die ausgeblendeten Bereiche als stark transparent und weiß dargestellt werden.

Auch die bereits angesprochenen Optimierungen, durch die Abtastpunkte nur innerhalb der 3D Textur liegen und bei hoher akkumulierter Opazität frühzeitig terminiert werden kann, sind implementiert.

Wenn beim Rendering des Feldes im Invariantenraum ein Abtastpunkt außerhalb der ausgewählten Intervalle von Invarianten liegt, werden die Farbe und Dichte als 0 angenommen. Wenn dagegen beim Rendern des Objekts ein Abtastpunkt die Invarianten der Tensoren an diesem Punkt außerhalb des Intervalls liegen, wird ein stark transparentes Weiß als Farbe festgelegt. Dadurch entsteht ein Schema des restlichen Objektes, wodurch die Position und Form der noch sichtbaren Bereiche besser einschätzbar wird. Dieser Effekt ist in Abb. 24 zu sehen.

Um die Tiefenwahrnehmung zu verbessern, werden zusätzlich Farben an Punkten, die näher am Zentrum des Objekts liegen, als dunkler angezeigt, als die an weiter vom Zentrum entfernten Punkten (siehe Abb. 24a). Dadurch sind Löcher im Volumen leichter zu erkennen, ohne Lichtquellen und Schattenbildung berechnen zu müssen.

Algorithmus 3 beschreibt die Funktionsweise des Raycasting Algorithmus für das DVR im Orts- und Invariantenraum. Teile des Algorithmus, gekennzeichnet durch Kommentare, werden dabei ausschließlich in einem der beiden Räume ausgeführt. Die Implementierung basiert auf dem FAnToM Algorithmus ‚Volume Renderer GLSL‘, verändert diesen jedoch deutlich um mehrere 3D Texturen in x-Richtung als Eingabe zuzulassen, Bereiche des Volume Rendering ausblenden zu können, korrekte Überlappung mit dahinter gezeichneten Objekten sicherzustellen, die in Abschnitt 6.9 beschriebenen Optionen zu ermöglichen und die benötigte Rechenzeit zu verkürzen.

**Data :**  $s$  Strahlvektor der Länge 1,  $t$  Schrittweite,  
 $T$  3D Textur,  $p_0$  erster Abtastpunkt,  $B$  ausgewählter Invariantenbereich  
**Result :** akkumulierte Farbe entlang des Strahls

```

transparenz ← 1,0;
fragmentfarbe ← (0; 0; 0; 0);
for i ← 0 to n do
    if transparenz < grenzwert then
        | break;
    end
    invarianten ← invarianten( $T, p$ );
    dichte ← dichte( $T, p$ );
    // Nur im Objekt
    if invarianten ∉ B then
        | fragmentFarbe ← (1,0; 1,0; 1,0; 0,05);
        | transparenz ← transparenz - 0,05;
        | continue;
    end
    // Nur im Invariantenraum
    if  $p \notin B$  then
        | continue;
    end
    voxelFarbe ← transferfkt(dichte);
    fragmentFarbe ← fragmentFarbe + voxelFarbe · transparenz;
    transparenz ← transparenz - voxelFarbe.transparenz;
end

```

**Algorithmus 3 :** Die Berechnung der akkumulierten Farbe eines Strahls durch die 3D Textur.

## 6.6 Die Interaktionswidgets

Um in der Darstellung des Feldes im Invariantenraum Intervalle von Invarianten auswählen zu können, wurden Interaktionflächen implementiert. Leiner et al. [27] stellen an solche Elemente, die sie als ‚3D Widgets‘ bezeichnen, eine Reihe von Kriterien:

**1. Klare Erkennbarkeit der Widgets** Widgets müssen leicht als solche erkennbar sein. Unter anderem müssen sie sich eindeutig von dem nicht-interaktiven Teil der Visualisierung unterscheiden lassen. Dies kann durch die Form, die Farbe oder die Positionierung der Widgets geschehen.

**2. Zerlegung in Handles** Da Tastatur und Maus oft nicht genügend Freiheitsgrade bieten um dreidimensionale Interaktionen direkt durch sie umsetzen zu können, wird die

Interaktion in Teile zerlegt. Für jeden Teil sollte eine Komponente des Widgets reserviert werden, der durch Maus und Tastatur manipuliert werden kann. Diese Komponenten werden von Leiner et al. als ‚Handles‘ bezeichnet. Die Interaktion mit einem Widget gliedert sich damit in zwei Teile: Die Selektion des gewünschten Handles und die Manipulation des Handles.

**3. Hervorhebung des selektierten Handles** Damit der Benutzer sicher sein kann, das korrekte Handle selektiert zu haben, sollte dieses visuell hervorgehoben werden.

Preim et al. [32, S. 340] fügen noch ein weiteres Kriterium hinzu:

**4. Affordances der Handles** Mit dem Begriff ‚Affordances‘ beschreiben Preim et al. [31, S. 137] Eigenschaften von Objekten, die auf die Interaktionsmöglichkeiten dieser Objekte hinweisen. Beispielsweise sollte ein Stuhl Affordances besitzen, die darauf schließen lassen, dass es möglich ist, sich auf diesen zu setzen. Welche Eigenschaften als Affordances erkannt werden, hängt von einer Vielzahl von physikalischen und logischen Faktoren ab, sowie davon, welche anderen Interaktionen der Benutzer bereits kennt.

Da in der Literatur kein für die gewünschten Interaktionen geeignetes Widget beschrieben ist, wurden im Rahmen dieser Arbeit eigene Widgets entwickelt, durch die Intervalle von einer oder mehreren der drei Invarianten eines Variantensatzes im Variantenraum ausgewählt werden können. Da die Variantenräume entweder zylindrische oder kartesische Koordinaten besitzt, werden für beide Koordinatensysteme jeweils eigene Widgets entwickelt. Die Auswahl des angezeigten Widgets muss dabei der Benutzer treffen, da FAnToM keine Möglichkeit unterstützt, das verwendete Koordinatensystem abzuspeichern.

Beide Widgets kombinieren die Konzepte des Cutaways [32, S. 406 f.], einer Interaktion, bei der Teile des Datensatzes ausgeblendet werden um das Innere des Datensatzes zu zeigen, mit den Interaktionen zur Positionierung von Schnittebenen, wie sie auch in Abschnitt 4.6.2 beschrieben werden.

An die Interaktionen wurde eine weitere Anforderung gestellt. Die Interaktionen sollten so implementiert werden, dass die Auswirkung einer Interaktion mit so kurzer Verzögerung wie möglich angezeigt wird. Die Implementierung der Auswirkungen der Interaktionen ist deshalb innerhalb des Raycastings umgesetzt. Tatsächlich wird die Laufzeit durch Interaktion mit dem quaderförmigen Interaktionswidget generell und durch Interaktion mit der ersten Variante beim zylindrischen Interaktionswidget reduziert, da die Bounding Box kleiner und damit die Länge des Schnitts zwischen Sichtstrahlen und Bounding Box geringer wird.

### **6.6.1 quaderförmiges Interaktionswidget**

Das erste Interaktionswidget hat die Form eines Quaders, der das erzeugte DVR vollständig umfasst. Die Seitenflächen sind parallel zu paarweise aus den kartesischen

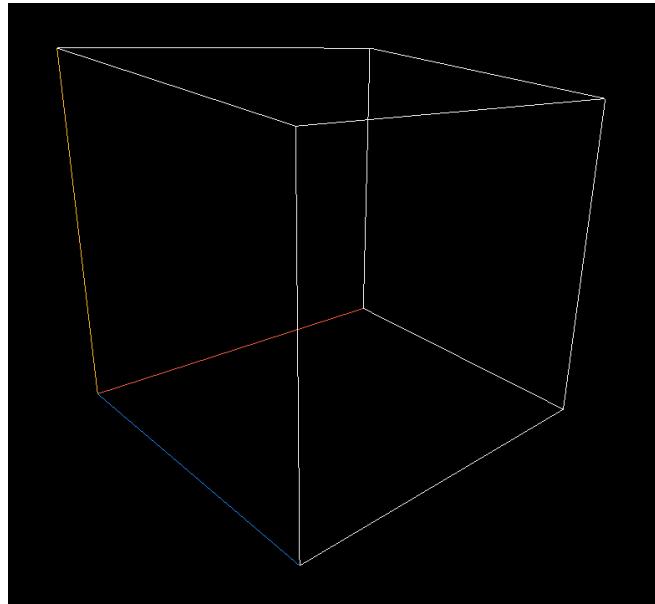


Abbildung 25: Eine Darstellung des quaderförmigen Interaktionswidgets. Die drei eingefärbten Kanten schneiden sich in dem Punkt, dessen Koordinaten die Minima aller drei Invarianten sind. Die Farben dienen der Orientierung im Raum und weisen darauf hin, zu welcher Achse die jeweilige Kante parallel ist: Rot zur x-, Gelb zur y- und Blau zur z-Achse.

Koordinatenachsen gebildeten Ebenen (xy-, xz-, yz-Ebene) und vollständig transparent. Nur die Kanten des Quaders sind sichtbar, sie werden durch Linien dargestellt. Um die Orientierung des Quaders im dreidimensionalen Raum erkennbar zu machen, sind die drei Kanten, die zum Knoten links unten hinten zusammenlaufen, eingefärbt: Die Kante parallel zur x-Achse in rot, die Kante parallel zur y-Achse in gelb und die Kante parallel zur z-Achse in blau. Die Farben wurden so gewählt, dass sie auch bei Farbenblindheit noch unterscheidbar sind. Die Koordinaten des Schnittpunkts der drei farbigen Kanten entsprechen den Minima der einzelnen Invarianten: Die x- dem der ersten, die y- dem der zweiten und die z-Koordinaten dem der dritten Invariante. Entsprechend verlaufen die farbigen Kanten von diesem Schnittpunkt aus in Richtung der steigenden Invarianten: Die rote in Richtung der ersten, die gelbe in Richtung der zweiten und die blaue Kante in Richtung der dritten Invariante. Die Positionen der Seitenflächen entsprechen im Ausgangszustand den Minima und Maxima der einzelnen Invarianten.

Die Geraden unterscheiden sich deutlich von der Darstellung des DVR, wodurch **Kriterium 1** erfüllt wird. Ein Problem kann jedoch auftreten, wenn die Farben der Transferfunktion sich mit denen der Kanten überschneiden. Abb. 25 zeigt das quaderförmige Interaktionswidget.

Die Interaktion findet statt, indem eine Seitenfläche durch Klicken und Halten der mittleren Maustasten ausgewählt und durch Bewegung der Maus entlang der zur Fläche orthogonalen Achse verschoben wird. Jede Seitenfläche stellt einen Handle dar, durch

den das Minimum oder Maximum des angezeigten Intervalls einer einzelnen Invariante eingestellt wird. Dadurch ist **Kriterium 2** erfüllt. Die Position der Seitenflächen folgt dabei der Maus, sodass immer die aktuellen Intervallgrenzen gezeigt werden. Dies erfüllt **Kriterium 3** zum Teil: Das aktuelle Handle liegt immer unter dem Mauszeiger und bewegt sich mit ihm, wodurch das ausgewählte Handle in der Regel erkennbar ist.

**Kriterium 4** ist ebenfalls nur teilweise erfüllt. Das quaderförmige Interaktionswidget kann als Teilmenge des kartesischen dreidimensionalen Raums aufgefasst werden, dessen Seitenflächen bewegt werden können. Dies stellt eine Affordance dar. Jedoch muss davon ausgegangen werden, dass diese Affordance nicht ausreicht, um Benutzer auf die möglichen Interaktionen mit den Handles hinzuweisen.

Ob und wie die Widgets verändert werden können, um die Kriterien besser zu erfüllen, ist Teil der weiteren Forschung und wird sich insbesondere aus Tests mit Anwendern ergeben.

Das quaderförmige Interaktionswidget kann für jeden Variantensatz verwendet werden, ist jedoch besonders für solche geeignet, bei denen alle drei Varianten beliebige Werte in  $\mathbb{R}$  annehmen können, z.B. Eigenwerte sowie I- und J-Variantensätze.

Die Berechnung, ob ein Mausklick eine Handle trifft, geschieht mittels ‚Ray-Picking‘. Dabei wird, ähnlich wie beim Raycasting, ein Strahl ausgehend von der Kamera durch die Szene berechnet. Wenn der Strahl ein Handle trifft, wird die jeweilige Intervallschranke ausgewählt. Wenn mehr als ein Handle getroffen wird, wird das zur Kamera nähste Handle bevorzugt. Preim et al. vergleichen diese Art der Auswahl von Widgets als Metapher zum Zeigen auf ein Objekt mit einem Laserpointer oder dem Greifen eines Objekts mit der Hand [32, S. 344].

Um die vom Interaktionswidget ausgeblendeten Bereiche besser einschätzen zu können, existiert die Option, zusätzlich zu den aktuellen Positionen der Kanten des Interaktionswidgets auch deren Position im Ausgangszustand einzuziehen. Die Darstellung dieser Kanten wird als Gitternetz oder ‚Wireframe‘ bezeichnet.

### **6.6.2 Zylindrisches Interaktionswidget**

Wie in Abschnitt 3.3.21 beschrieben, kann es für manche Variantensätze, z.B. die K- und R-Variantensätze, sinnvoll sein, sie in einem zylindrischen Koordinatensystem darzustellen. Dafür müssen eigene Interaktionswidgets entwickelt werden, durch die, äquivalent zum quaderförmigen Widget, die Intervallgrenzen der Varianten manipuliert werden können.

Die implementierten Widgets haben die Form von Kreissegmenten mit unterschiedlich eingefärbten Komponenten, die parallel zur yz-Ebene gezeichnet werden. Ein zylindrisches Interaktionswidget ist in Abb. 26a dargestellt. Jede farbige Komponente markiert einen Teil des Widgets, durch den eine andere Variante manipuliert werden kann. Diese Teile gliedern sich wiederum in Handles für das Minimum und Maximum der jeweiligen

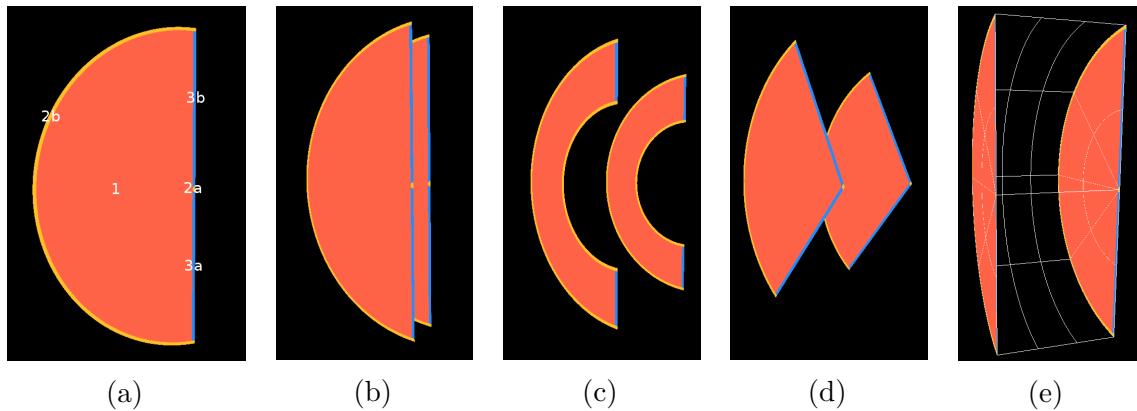


Abbildung 26: Die Darstellungen der zylindrischen Interaktionswidgets. (a) Ein Widget mit den einzelnen Handles. Ein Klick auf die Komponenten 1, 2 und 3 des Interaktionswidgets wählt das Handle der jeweils ersten, zweiten oder dritten Invariante aus. Die Komponenten 2a und 3a sind dabei die Minima der Invariante, 2b und 3b die Maxima. (b) bis (d) zeigen veränderte Positionen und Formen der Interaktionswidgets durch gewählte Invariantenbereiche, wobei (b) der ersten, (c) der zweiten und (d) der dritten Invariante entspricht. In (e) ist der zylindrische Wireframe dargestellt.

Invariante auf. Die Wahl der Farben folgt der gleichen Logik wie beim quaderförmigen Widget. Dadurch wird **Kriterium 1** erfüllt.

Die Interaktion selbst ist analog zum quaderförmigen Widget: Durch Klicken der Maustaste wird ein Handle ausgewählt, dessen Position der Bewegung der Maus folgt. Für die erste Invariante entspricht dies der x-Koordinate des jeweiligen Widgets (siehe Abb. 26b). Bei der zweiten und dritten Invariante wird die Form des Widgets angepasst: Da die zweite Invariante der Entfernung zur x-Achse entspricht, werden innerer und äußerer Radius des Kreissegments angepasst. Wenn der innere Radius größer als null ist, wird somit aus dem Kreissegment ein Kreisringsektor (siehe Abb. 26c). Die dritte Invariante entspricht dem Skalarprodukt zwischen der Projektion des Punktes auf die yz-Ebene und der y-Achse. Die zugehörigen Handles sind die beiden geraden Seiten des Kreissegments, deren Winkel zur y-Achse verändert wird (siehe Abb. 26d). Durch diese Aufteilung wird **Kriterium 2** erfüllt.

**Kriterien 3 und 4** werden genau wie beim quaderförmigen Widget nur teilweise erfüllt und sind Thema weiterer Forschung.

Die zylindrischen Widgets stellen somit die Seitenflächen eines Ausschnitts aus einem Zylinder dar, der den dargestellten Teil des Invariantenraums umfasst. Um diese Eigenschaft zu verdeutlichen, kann ein zusätzliches Gitter aus Linien auf der Oberfläche des Zylinders gezeichnet werden, äquivalent zum Wireframe beim quaderförmigen Interaktionswidget (siehe Abb. 26e).

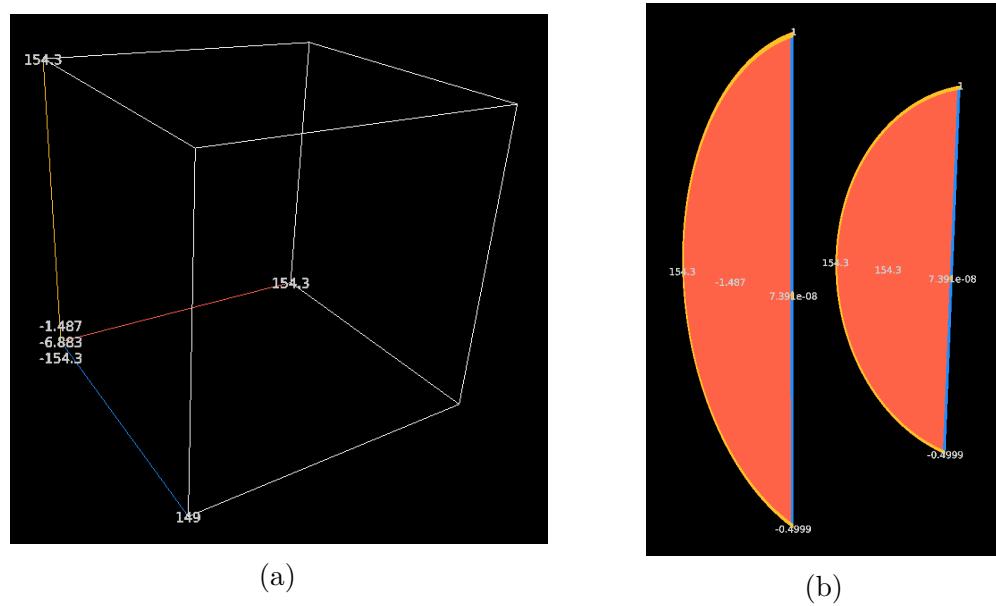


Abbildung 27: Labels an den Interaktionswidgets: (a) am quaderförmigen Interaktionswidget, (b) an den zylindrischen Interaktionswidgets

## 6.7 Labeling

Um die Position der Interaktionshandles im Raum ablesen zu können, ist es möglich, die eingestellten Minimal- und Maximalwerte der Invarianten an den Interaktionswidgets einzuzeichnen. Solche Beschriftungen in grafischen Oberflächen werden als ‚Labels‘ bezeichnet.

Beim quaderförmigen Interaktionswidget werden die Labels an den Endpunkten der farbigen Kanten angezeigt. An dem Punkt, an dem sich die drei Kanten schneiden, werden die Labels untereinander angeordnet, mit dem Label der ersten Invariante oben und dem der dritten unten. Abb. 27a zeigt die Positionierung der Labels.

Die Labels am zylindrischen Interaktionswidget sind an den Handles eingezeichnet. Für die erste Invariante befinden sich die Labels im Zentrum der zugehörigen Handles, der großen roten Flächen. Auch die Labels der zweiten Invariante sind an den entsprechenden Handles, den gelben Bögen an der Innen- und Außenseite der Kreisringausschnitte, eingezeichnet. Für die Position der Labels der dritten Invariante werden die äußeren Eckpunkte der geraden Seiten der Kreisringausschnitte gewählt. Die Positionen sind in Abb. 27b zu sehen.

## 6.8 Weitere Interaktionen

Es existieren noch eine Reihe weiterer Interaktionen, mit deren Hilfe die Visualisierung angepasst werden kann:

**Rotation des DVR** Indem die linke Maustaste gedrückt und gehalten wird, kann durch die Bewegung der Maus die Visualisierung um das aktuelle Rotationszentrum gedreht werden. Die Rotation ist in Form eines sogenannten ‚Trackballs‘ implementiert. Preim et al. beschreiben den Trackball als imaginäre, das Objekt umschließende Kugel, die durch Mausbewegung gedreht wird und diese Bewegung auf das dargestellte Objekt, in diesem Fall das DVR und die Interaktionswidgets, überträgt.

**Setzen des Rotationszentrums** Das Rotationszentrum des Trackballs kann, wenn die zylindrischen Interaktionswidgets aktiviert sind, mittels doppeltem Linksklick an einen Punkt entlang der x-Achse gesetzt werden. Wenn dagegen das quaderförmige Interaktionswidget aktiviert ist, ist das Rotationszentrum immer im Zentrum des Quaders und bewegt sich bei Einschränkung der Invariantenintervallgrenzen mit.

**Bewegung der Kameraposition** Ähnlich wie bei der Rotation der Szene, kann durch Drücken und Halten der rechten Maustaste die Kameraposition mittels Bewegung der Maus innerhalb der Szene verschoben werden.

**Zurücksetzen von Rotation und Kameraposition** Durch Drücken der ‚c‘-Taste wird die Kameraposition und die Rotation des Feldes zurückgesetzt.

**Zurücksetzen der Invariantenbereiche** Die Schranken des ausgewählten Invariantenbereichs können durch Drücken der ‚r‘-Taste zurückgesetzt werden.

**Zoomen** Durch Drehen des Mausrades nach vorn wird die Kamera um eine Schrittänge in Blickrichtung nach hinten bewegt, durch Drehen des Mausrades nach hinten eine Schrittänge nach vorn. Jedes Mal wenn die Kamera auf diese Weise nach vorn bewegt wird, wird die Schrittänge um einen Prozentsatz verringert, wenn die Kamera nach hinten bewegt wird um einen Prozentsatz erhöht. Durch diese Funktion wird eine Art von Zoomen realisiert. Die Geschwindigkeit des Zoomens reduziert sich, desto näher herangezoomt wurde und beschleunigt sich, desto weiter herausgezoomt wurde. Dadurch ist es sowohl möglich, auf einen bestimmten Punkt der Visualisierung heranzuzoomen als auch herauzuzoomen und sich einen Überblick über die Visualisierung zu verschaffen.

## 6.9 Optionen des Algorithmus

Der Visualisierungsalgorithmus bietet eine Reihe von Optionen an, durch die beide DVR Darstellungen angepasst werden können. Die Optionen gliedern sich dabei auf in solche, die für beide Darstellungen existieren und solche, die nur für eine von beiden vorhanden sind.

### 6.9.1 Gemeinsame Optionen

**Anzahl von Abtastpunkten** Die Anzahl der Abtastpunkt pro Strahl kann für beide Darstellungen unabhängig voneinander mittels eines Textfeldes gewählt werden.

**Jittering** Um visuelle Artefakte zu verringern, kann durch eine Option ein pseudozufälliger Abstand pro Strahl berechnet werden, um den der erste Abtastpunkt verschoben wird. Dadurch werden manche Artefakte, die beim Raycasting entstehen können, vermieden.

**Helligkeit** Ein Schieberegler ermöglicht die Einstellung der Helligkeit der Farben des Volume Rendering, unabhängig von der Transferfunktion.

**Dichte** Durch einen weiteren Schieberegler kann ein Faktor festgelegt werden, mit dem die Dichte der Voxel multipliziert wird. Alternativ existiert dafür auch ein Textfeld, falls exakte Werte oder Werte außerhalb der Reichweite des Schiebereglers benötigt werden.

**Offset** Der dritte Schieberegler legt einen Grenzwert für die Dichtewerte fest. Voxel, deren Dichte unterhalb dieses Grenzwerts liegt, werden als komplett transparent angenommen.

**Lineare Interpolation** Standardmäßig misst die Abtastung die Dichtewerte der Textur an einem vorgegebenen Punkt. Da die Textur durch Rasterisierung entstanden ist, kann dies zu visuellen Artefakten führen, die die Darstellung ‚blockig‘ erscheinen lassen. Es wurde eine Option implementiert, durch die stattdessen lineare Interpolation verwendet werden kann, um die Dichte an einem Punkt zu bestimmen. Dabei wird neben der Dichte des Voxels selbst auch die Dichte der angrenzenden Voxel verwendet, und zwischen diesen der Wert am jeweiligen Abtastpunkt interpoliert.

**Skalierung der Transparenz mit der Anzahl der Abtastpunkte im Feld** Diese Option macht es möglich, die beim Raycasting pro Pixel akkumulierte Transparenz durch die Anzahl der Abtastpunkte des Strahls zu teilen. Dies hat den Vorteil, dass stark transparente Voxel am Rand des Datensatzes deutlicher dargestellt werden, was die Erkennung von Strukturen in diesen Bereichen erleichtert, jedoch das Ablesen konkreter Dichtewerte aus der Visualisierung erschwert.

**Anzahl der Voxel in x/y/z-Richtung** Die Anzahl der Voxel, in die das Feld voxelisiert werden soll, kann für die einzelnen Dimensionen eingestellt werden. Höhere Werte führen zu einer besseren Darstellung kleiner Tetraeder, erhöhen jedoch die Rechenzeit und den benötigten Speicher.

### 6.9.2 Optionen des Objektrenderings

**Bounding Box** Um die Größe und Form des Objekts besser einschätzbar zu machen, können weiße Linien an den Kanten der begrenzenden Flächen eingezeichnet werden, also die Bounding Box der 3D Textur dargestellt werden.

### 6.9.3 Optionen des Invariantenraumrenders

**Ausgewähltes Interaktionswidget** Der Benutzer kann entscheiden, ob das quaderförmige oder die zylindrischen Interaktionswidgets angezeigt werden sollen. Dies ist notwendig, da es nicht möglich ist aus der internen Repräsentation eines Feldes in FAnToM zu bestimmen, ob dieses in zylindrischen oder kartesischen Koordinaten liegt.

**Wireframe** Für beide Interaktionswidgets können optional Wireframes angezeigt werden, Liniengitter an den Grenzen der Invariantenintervalle im Ausgangszustand. Die Position der Linien macht es einfacher, die ausgewählten Bereiche einzuschätzen. Form und Position des Wireframes sind nicht abhängig vom ausgewählten Invariantenbereich. Der Wireframe des zylindrischen Interaktionswidgets ist in Abb. 26e zu sehen.

**Labels einzeichnen** Es ist möglich auszuwählen, ob die Minimal- und Maximalwerte der jeweiligen Invarianten an den Widgets in Form von Labels eingezeichnet werden sollen oder nicht.

**Interaktionswidgets ausblenden** Da die Interaktionswidgets einen großen Teil des Objekts verdecken können, besteht die Möglichkeit, diese auszublenden. Während sie ausgeblendet sind, ist keine Interaktion mit ihnen möglich, die Invariantenbereiche können also nicht durch Mausinteraktion verändert werden.

**Wurzelskalierung** Eine Fragment-Shader der Voxelisierung durchgeführte Skalierung durch Ziehen der dritten Wurzel kann an- und ausgeschaltet werden. Die Begründung dafür ist in Abschnitt 5.2 näher beschrieben.

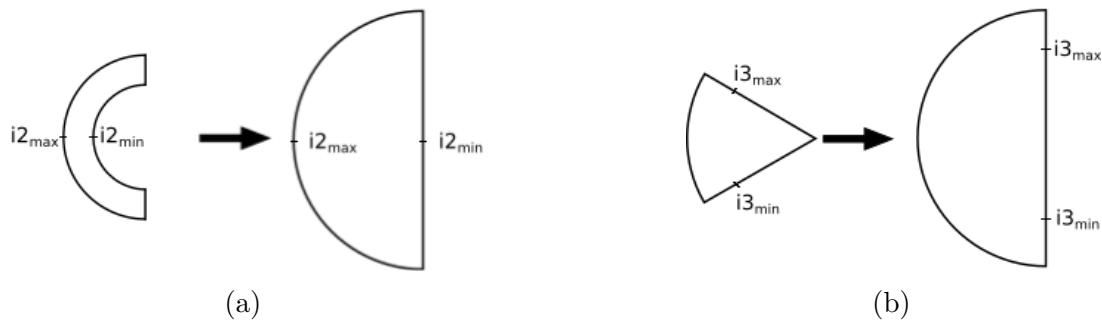


Abbildung 28: Darstellung der Normalisierung bei aktivierten zylindrischen Interaktionen: (a) Normalisierung der Darstellung der zweiten Invariante und (b) Normalisierung der dritten Invariante.

**Normalisierung des Feldes** Im Invariantenraum kann das Feld beliebige Formen annehmen. Wenn beispielsweise eine Invariante nur Werte innerhalb eines sehr viel kleineren Intervalls annimmt als die anderen beiden, wird die 3D Textur als sehr flach dargestellt. Dies kann die Interpretation der Visualisierung erschweren. Deshalb ist es möglich, die Darstellung des Feldes auf eine feste Form zu skalieren. Die Form ist abhängig davon, ob zylindrische oder quaderförmige Interaktionen ausgewählt sind. Wenn das quaderförmige Interaktionswidget aktiviert ist, wird die Bounding Box der 3D Textur einfach auf einen Würfel mit einer Kantenlänge von 1 skaliert. Bei aktiviertem zylindrischen Interaktionswidget dagegen wird die Darstellung des Feldes auf einen Halbzylinder mit einer Höhe und einem Durchmesser von 1 skaliert. Dazu wird die 3D Textur entlang der drei zylindrischen Koordinatenachsen verformt. Für die erste Invariante entspricht dies einer Skalierung auf das Intervall  $[0;1]$  entlang der Mittelachse des Zylinders. Bei der zweiten Invariante wird die 3D Textur so skaliert, dass Punkte, deren zweite Invariante im Datensatz minimal ist, auf der Mittelachse des Halbzylinders und Punkte mit maximaler im Datensatz vorkommender zweiter Invariante auf der Mantelfläche des Halbzylinders liegen. Eine Seitendarstellung der Skalierung der zweiten Invariante ist in Abb. 28a dargestellt. Ähnlich geschieht auch die Skalierung der dritten Invariante. Punkte mit minimalen Werten der dritten Invariante werden so skaliert, dass sie auf der unteren Hälfte der quadratischen Seitenfläche des Halbzylinders, Punkte mit maximalen Werten so, dass sie auf der oberen Hälfte liegen. Wiederum in einer Seitendarstellung ist dies in Abb. 28b dargestellt.

**Minimale Dichte** Die Dichtewerte der Voxel im Invariantenraum können sich extrem unterscheiden. Abhängig von der Größe der jeweiligen Voxel traten in Testdatensätzen Werte zwischen  $10^{-4}$  und  $10^{22}$  auf. Damit auch Bereiche mit sehr geringer Dichte sichtbar sind, ist es möglich, eine minimale Dichte im Intervall  $[0; 1]$  anzugeben. Wenn der in der 3D Textur abgespeicherte Wert die angegebene minimale Dichte unterschreitet, wird stattdessen dieser Wert verwendet.

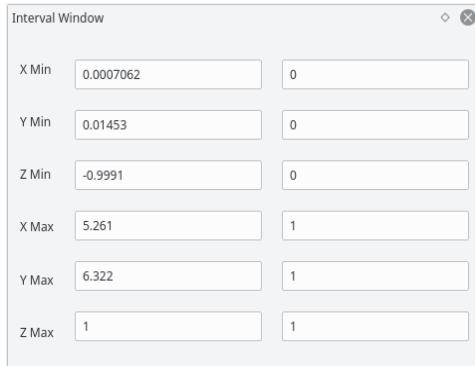


Abbildung 29: Das Intervall-Fenster.

## 6.10 Das Intervallfenster

Um die Invariantenintervalle auf exakte Werte einstellen zu können, existiert ein weiteres Fenster (siehe Abb. 29). Darin sind die sechs Schranken der Invariantenintervalle tabellarisch angegeben. Das linke Textfeld gibt den absoluten, das rechte den relativen Wert im Intervall  $[0; 1]$  an. Alle Textfelder sind editierbar und Änderungen werden direkt auf die DVR angewendet.

## 7 Ergebnisse

Die Visualisierung wurde auf eine Reihe von Testdatensätzen angewendet, um die Korrektheit und Qualität der Darstellung zu bewerten. In diesem Abschnitt werden die Datensätze vorgestellt, die Ergebnisse der Visualisierungen gezeigt und mit bestehenden Visualisierungen verglichen sowie mögliche Interpretationen erläutert.

Als Vergleichsdarstellung und um eine Vorstellung über die Struktur der Datensätze zu vermitteln, werden für die ersten beiden Datensätze Visualisierungen der Tensoren als Ellipsoidglyphen gezeigt. Die Glyphen werden abhängig von den Eigenwerten des jeweiligen Tensors unterschiedlich dargestellt. Die Länge der Achsen eines Ellipsoids entspricht den Eigenwerten und die längste Achse zeigt in Richtung des dazugehörigen Eigenvektors.

Die Ergebnisbilder wurden auf einem Rechner mit 8 GB Arbeitsspeicher, einem Intel i5-3570K Prozessor mit 4 Kernen und einer GeForce GTX 970 Grafikkarte in einer Auflösung von  $1920 \times 1080$  erzeugt.

Das Raycasting wurde bei allen folgenden Ergebnissen mit 5000 Abtastpunkten pro Strahl durchgeführt. Die Unterschiede in der Laufzeit des Raycastings liegen damit hauptsächlich an der Anzahl an Pixeln, von denen aus Strahlen durch die 3D Textur geschickt werden. Die Anzahl der Pixel entspricht der Größe der auf die Bildebene projizierten Bounding

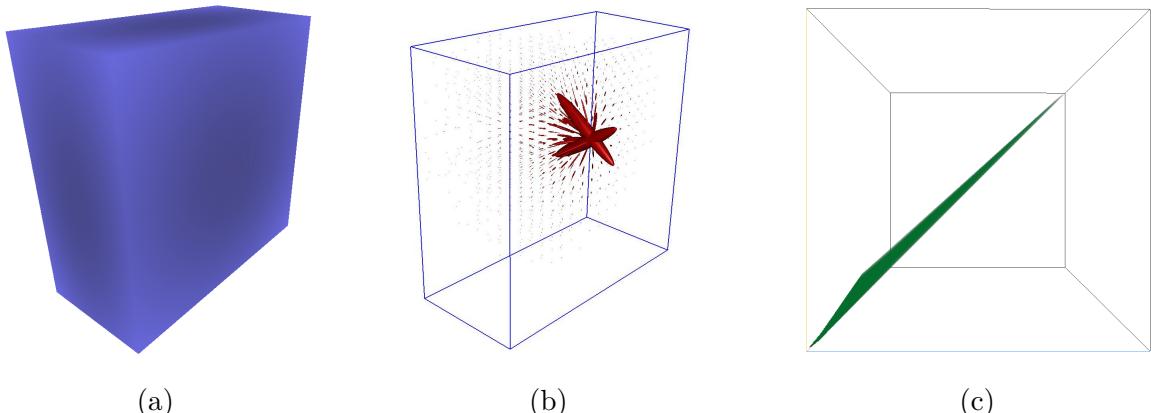


Abbildung 30: Darstellung des Single Point Load Datensatzes: (a) als DVR des Objekts, (b) als Ellipsoide und (c) im Invariantenraum, berechnet mit dem I-Invariantensatz.

Box um die 3D Textur und ändert sich abhängig von der Perspektive und der Form der Bounding Box.

## 7.1 Single Point Load

Um die grundlegenden Funktionen der implementierten Visualisierung zu erläutern, wird als erster Testdatensatz ein von FAnToM erzeugtes Tensorfeld verwendet. Dieses ist das Ergebnis einer Simulation, in der auf einen unendlichen Halbraum an einem Punkt eine Kraft einwirkt. Diese Art von Datensätzen wird auch als ‚Single Point Load‘ bezeichnet. Die entstehende mechanische Spannung wird an Punkten innerhalb des Objekts gemessen und die Tensoren in Form von  $3 \times 3$  Matrizen gespeichert. Darstellungen des Datensatzes sind in Abb. 30 zu sehen. Der Punkt, an dem die Kraft einwirkt, befindet sich in der linken Hälfte der vorderen, quadratischen Seite. Der Datensatz besteht aus 19.494 tetraederförmigen Zellen.

Ein DVR eines quaderförmigen Ausschnitts des Halbraums in der Nähe des Einwirkungspunktes der Kraft ist in Abb. 30a zu sehen. Die verwendete Transferfunktion ist konstant vollständig opak und blau für alle Eingabewerte, um die Form des Halbraumausschnitts deutlich darzustellen. Die Darstellung soll vor allem als Vergleichspunkt für die später gezeigten Visualisierungen des Halbraums dienen, bei denen Teile abhängig von den gewählten Invarianten ausgeblendet wurden.

Abb. 30b zeigt eine Darstellung der Tensoren als Ellipsoidglyphen. Deutlich erkennbar ist der Punkt an dem die Kraft einwirkt, denn dort sind die Glyphen am größten. Die Form der Ellipsoide ist länglich und zeigt weg vom Einwirkungspunkt der Kraft.

Für die Visualisierung wurde der I-Invariantensatz gewählt. Abb. 30c zeigt das DVR des Datensatzes im Invariantenraum des I-Invariantensatzes. Die Transferfunktion ordnet Bereichen hoher Dichte die Farbe Grün und hohe Opazitätswerte zu, Bereichen geringer

Dichte die Farbe Weiß und geringe Opazitätswerte. Da der I-Invariantensatz keiner der Invariantensätze ist, die sich für die Darstellung in zylindrischen Koordinaten eignen, wird ein kartesisches Koordinatensystem und das quaderförmige Interactionswidget verwendet. Auch wird die Darstellung durch Mausinteraktion um  $90^\circ$  im Uhrzeigersinn um die y-Achse rotiert. Dadurch wird die Form des DVR deutlicher sichtbar. Der Punkt, an dem die drei farbigen Kanten zusammentreffen liegt nach der Drehung vorn links unten.

Durch die Normalisierungsfunktion wurden die Dimensionen des Invariantenraumes so skaliert, dass die Intervalle der drei Invarianten gleich groß dargestellt werden. Ohne Skalierung ist die Ausdehnung Bounding Box in Richtung der z-Achse relativ gering.

Die Form des DVR im Invariantenraum ist länglich und verläuft entlang einer imaginären Geraden zwischen zwei Punkten, die jeweils das Minimum und Maximum aller drei Invarianten darstellen. Dies zeigt eine starke lineare Abhängigkeit zwischen den drei Invarianten im Datensatz. Zu erklären ist dies teilweise durch die fehlende Orthogonalität des I-Invariantensatzes: Desto höher die Eigenwerte einer Matrix, desto höher sind auch die Werte der I-Invarianten.

Da nur für  $I_1$  eine eindeutige Interpretation in der Mechanik existiert, beschränkt sich die Demonstration der Funktion des Interactionswidgets in diesem Datensatz auf  $I_1$ . In den Abb. 31a, 31c und 31e wird die untere Schranke des Intervalls der ersten Invariante erhöht. Im unrotierten Zustand wird dies durch die Position der linken Fläche des quaderförmigen Interactionswidgets dargestellt. Da jedoch das DVR rotiert wird, ist diese Fläche nun die vordere des Interactionswidgets. Dies erschwert zwar die Interpretation, wird jedoch von der besser sichtbaren Darstellung des DVR nach der Rotation mehr als aufgewogen. Um den ausgeblendeten Bereich des DVR besser einschätzen zu können, wird das Wireframe in einem hellen Grau eingeblendet. Die Abb. 31b, 31d und 31f stellen die Bereiche des Ausschnitts des Halbraums in blau dar, in denen die Invarianten innerhalb der gewählten Intervalle liegen. Der Rest des Halbraumausschnitts wird schemenhaft in grau dargestellt.

In Abb. 31a wird das Intervall von  $I_1$  um etwa 1% verkleinert, was in der Darstellung des Invariantenraumes kaum erkennbar ist. Dennoch wird in Abb. 31b ein großer Teil des Halbraums ausgeblendet. Der minimale Wert von  $I_1$  im Datensatz liegt bei 0,009, der maximale Wert bei 44,03. Aus diesen beiden Beobachtungen und der Interpretation von  $I_1$  als Maß für die Stärke der isotropen Spannung folgt, dass in einem großen Teil des Datensatzes fast keine isotrope Spannung vorliegt. Nur in direkter Nähe zum Einwirkungspunkt der Kraft nimmt  $I_1$  höhere Werte an.

Abb. 31c zeigt eine Darstellung des Invariantenraumes, in dem die untere Schranke von  $I_1$  um 10% der Intervalllänge erhöht ist. Dadurch ist die untere Spitze des DVR abgeschnitten. Die Wirkung auf die angezeigten Bereiche im DVR des Halbraums in Abb. 31d ist deutlich erkennbar. Diese sind im Vergleich zu Abb. 31d deutlich kleiner und wieder in der Nähe des Einwirkungspunktes der Kraft.

Das gleiche Verhalten kann auch in Abb. 31e abgelesen werden, wo im Invariantenraum die unteren 50% des Intervalls von  $I_1$  ausgeblendet sind. Abb. 31f stellt die zugehörigen Bereiche im Halbraum dar.

Die Anzahl der erzeugten Voxel wird im Folgenden immer in der Reihenfolge x-Richtung  $\times$  y-Richtung  $\times$  z-Richtung angegeben. Im Falle des Single Point Load betrug sie  $400 \times 400 \times 400$  für das DVR im Invariantenraum und  $400 \times 400 \times 200$  im Ortsraum. Die geringere Anzahl von Voxeln in z-Richtung des Ortsraums ist möglich, da der Halbraumausschnitt in diese Richtung dünner ist. Dadurch wird die Größe der 3D Textur reduziert und Speicherplatz gespart.

Die Erzeugung 3D Texturen benötigte für beide DVR zusammen etwa 3 Sekunden, das Raycasting etwa 0,02 Sekunden.

## 7.2 Metallscheibe

Als zweites Beispiel werden die Ergebnisse einer thermo-mechanischen Simulation einer Metallscheibe verwendet. Die Simulation wurde von Prof. Dr. Thomas Nagel vom Helmholtz-Zentrum für Umweltforschung durchgeführt. Dieselben Daten bilden auch die Grundlage für ein Anwendungsbeispiel in der Arbeit von Fritzsch [14, S.15, 45 ff.]. Fritzsch beschrieb die Datensätze dort wie folgt:

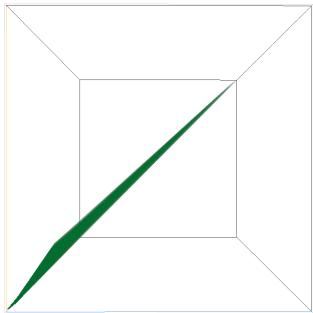
„Die Metallscheibe liegt flach auf und ist entlang ihres äußeren Rands fixiert. Zwecks Umformung wirkt von oben ein hoher Druck ein. Zusätzlich wird sie von unten erwärmt und von oben gekühlt.“[14, S. 15]

Die ‚obere‘ und ‚untere‘ Seite der Metallscheibe sind dabei die flachen Seiten, deren Normalen parallel zur z-Achse sind. Die z-Achse selbst zeigt nach unten. Die Anzahl der Tetraederzellen beträgt 26.984.

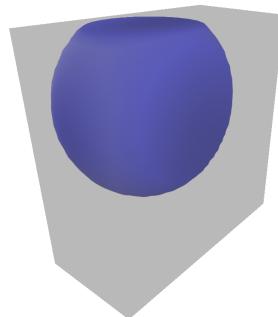
Da die Metallscheibe spiegelsymmetrisch entlang der x- und y-Achse ist, genügt es, ein Viertel der Scheibe zu simulieren. Als Ergebnisse der Simulation wurden zwei Datensätze erzeugt, die jeweils die mechanische Spannung und die Verformung an den Punkten der Metallscheibe enthalten.

Beide Datensätze werden von Fritzsch untersucht. Dabei werden als Invarianten  $I_1$  und  $\sqrt{2J}$  verwendet. Da bei symmetrischen Tensoren gilt  $I_1 = K_1$  und  $\sqrt{2J_2} = K_2$  und die Tensoren beider Datensätze dreidimensional sind, ist im Folgenden stattdessen vom K-Invariantensatz die Rede. Um die Korrektheit der in der vorliegenden Arbeit implementierten Visualisierung zu zeigen und die Ergebnisse mit denen von Fritzsch vergleichen zu können, werden Visualisierungen des K-Invariantensatzes für die mechanische Verformung erzeugt, die im Folgenden denen von Fritzsch gegenübergestellt werden. Für die Erstellung der DVR wird als dritte Koordinate  $K_3$  verwendet.

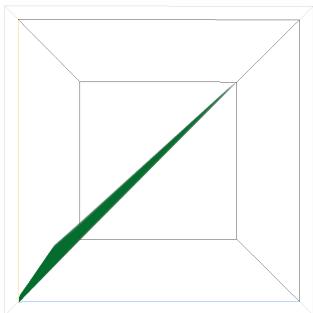
Ein DVR der Metallscheibe ist in Abb. 32a in blau dargestellt. Abb. 32b zeigt eine Darstellung der Verformungstensoren als Ellipsoidglyphen. Am inneren Rand der Scheibe,



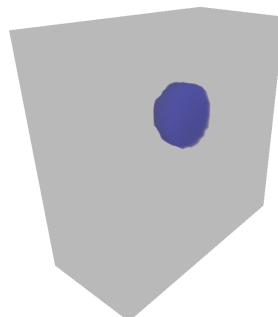
(a)



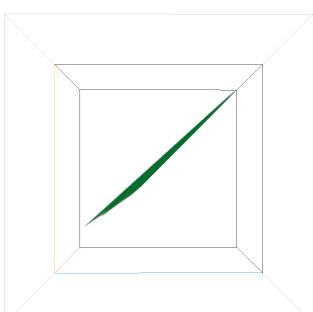
(b)



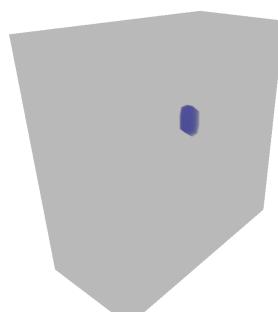
(c)



(d)

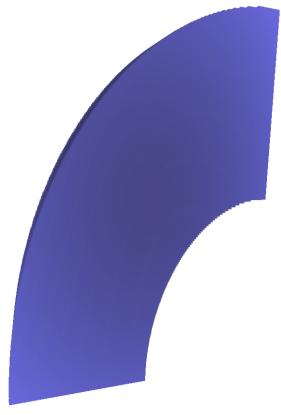


(e)

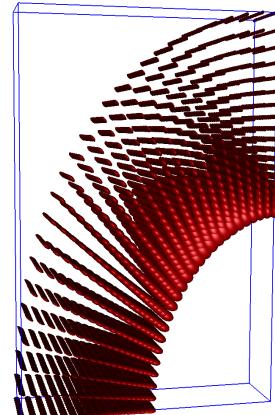


(f)

Abbildung 31: Darstellungen der Interaktion mittels des quaderförmigen Interaktionswidets für den Single Point Load Datensatz. Angezeigt werden links der Invariantenraum des I-Invariantensatzes und rechts Darstellungen des Halbraumes. In (a) und (b) ist die untere Schranke um 1% der Intervalllänge erhöht, in (c) und (d) um 10% und in (e) und (f) um 50%.

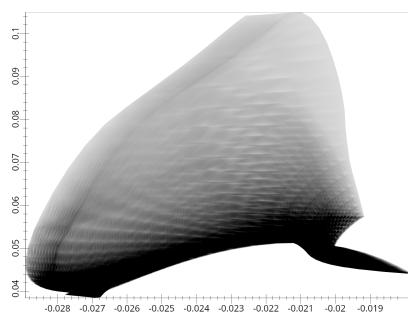


(a)

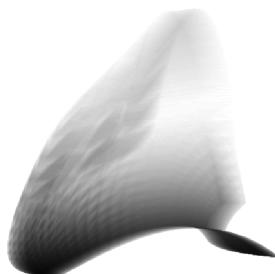


(b)

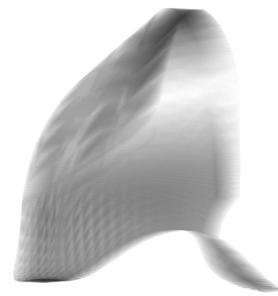
Abbildung 32: Der Metallscheiben-Datensatz: (a) ein DVR des Objektes, (b) Tensorglyphen der mechanischen Verformung.



(a)



(b)



(c)

Abbildung 33: Darstellungen des Invariantenraums des K-Invariantensatzes für die mechanische Verformung in der Simulation der Metallscheibe: (a) Die von Fritzsch vorgestellte Visualisierung [14, S. 45], (b) ein DVR mit orthogonaler Projektion in kartesischen Koordinaten, (c) ein DVR mit perspektivischer Projektion in zylindrischen Koordinaten.

insbesondere im oberen Bereich, sind die Ellipsoide am größten. Dies lässt auf hohe Beträge der Eigenwerte dort schließen. Das Vorzeichen der Eigenwerte lässt sich jedoch nicht ableiten. Zum äußeren Rand hin nimmt die Größe der Ellipsoide ab. Eine weitere Eigenschaft lässt sich aus der Form der Ellipsoide ablesen. Diese scheint am unteren Ende des Scheibenausschnitts eher länglich, am oberen Ende eher rund zu sein. Scheinbar ist die Verformung am oberen Ende stärker isotrop als am unteren. Ebenfalls nimmt die Größe der Ellipsoide von oben rechts nach unten links zu, die Beträge der Eigenwerte werden also scheinbar von unten nach oben größer.

Abb. 33a zeigt die von Fritzsch erzeugte Visualisierung der mechanischen Verformung. Die x-Achse entspricht dem Wert von  $K_1$ , die y-Achse dem von  $K_2$ . Da seine Visualisierung ebenfalls auf den Prinzipien des Continuous Scatterplotting basiert, weisen dunklere Bereiche auf höhere Dichten hin.

Damit die Ähnlichkeit zwischen der Darstellung von Fritzsch und den Ergebnissen der vorgestellten Arbeit deutlich werden, wurde ein DVR mit orthogonaler Projektion in kartesischen Koordinaten erzeugt, das in Abb. 33b zu sehen ist. Für die Verwendung von orthogonaler Projektion existiert im implementierten FAnToM Plugin noch keine Option, diese könnte jedoch in Zukunft hinzugefügt werden. Auch die verwendete Transferfunktion wurde absichtlich so gewählt, um die Darstellung des DVR an die von Fritzsch anzulegen: Bereiche mit hoher Dichte werden als opak und schwarz dargestellt, Bereiche geringer Dichte als transparent und weiß. Die x- und y-Achsen entsprechen beim orthogonalen DVR jeweils  $K_1$  und  $K_2$ , analog der Darstellung von Fritzsch. Die z-Achse entspricht zusätzlich  $K_3$ .

Die Abb. 33a und 33b ähneln sich stark, weisen jedoch eine Reihe deutlicher Unterschiede auf. Zunächst ist die Abb. 33a deutlich breiter, was an einer unterschiedlichen Skalierung in Richtung der x-Achse liegt. Des Weiteren ist der rechte obere Teil von Abb. 33b deutlich heller als der von Abb. 33a. Das ist darauf zurückzuführen, dass die Tetraeder in diesem Bereich sehr flach und fast parallel zur xy-Ebene sind. Dadurch werden sie nur von wenigen oder gar keiner der bei der Voxelisierung verwendeten Schnittebenen getroffen, wodurch nur wenige Voxel Dichtewerte erhalten. Durch die Skalierung der Dichte mit der Anzahl der Abtastpunkte wird der Bereich deutlich heller als in der Darstellung von Fritzsch. Ein weiterer wichtiger Faktor ist, dass das von Fritzsch implementierte Verfahren die Dichtewerte der projizierten Tetraeder einfach aufaddiert. Im DVR würde dies dem Röntgenbildverfahren entsprechen. Da stattdessen ein transluzentes Verfahren implementiert wurde, erscheinen dichte Bereiche wie an der unteren Kante des DVR heller als dieselben Stellen in der Darstellung von Fritzsch, da helle Bereiche an der Vorderseite die dahinterliegenden Bereiche überdecken.

Der K-Invariantensatz eignet sich für die Darstellung in zylindrischen Koordinaten. Daher bietet sich die mechanische Verformung der Metallscheibe an, um das zylindrische Interaktionswidget zu demonstrieren. Abb. 33c zeigt ein DVR desselben Datensatzes wie Abb. 33b, bei dem jedoch die Koordinaten der Eckpunkte der Tetraeder in Zylinderkoordinaten interpretiert wurden.

Für alle vom Invariantenraum der Metallscheibe erzeugten DVR wurden die Koordinaten mit der in 6.9.3 vorgestellten Option normalisiert, um die Form des Feldes im Invariantenraum besser sichtbar zu machen.

Abb. 34 demonstriert die Funktionsweise des zylindrischen Interaktionswidgets. Abb. 34a, 34c und 34e zeigen Darstellungen der Metallscheibe im Invariantenraum. Links und rechts des DVR sind die Interaktionswidgets eingeblendet. Durch Interaktion wurden die Intervalle der anzugezeigenden Invarianten begrenzt, was durch Position und Form der Interaktionswidgets angezeigt wird. Allgemein entspricht der Teil des Invariantenraums, der innerhalb der ausgewählten Invariantenintervalle liegt, dem Bereich zwischen den roten Teilen der zylindrischen Interaktionswidgets. Teile der DVR, die außerhalb der Intervalle liegen, werden ausgeblendet. Abb. 34b, Abb. 34d und 34f zeigen DVR der Metallscheibe, bei denen die Bereiche mit Invarianten innerhalb des ausgewählten Intervalls in blau und außerhalb in einem transparenten Grau dargestellt werden.

In Abb. 34a wurde das linke Interaktionswidget nach rechts bewegt, wodurch die untere Schranke des Intervalls von  $K_1$  erhöht wurde während die obere Schranke gleich blieb. Die Länge des Intervalls wurde insgesamt um 40% reduziert. Der Teil des DVR außerhalb des eingeschränkten Intervalls, also links vom linken Interaktionswidget, ist ausgeblendet. Abb. 34b zeigt das zugehörige DVR im Ortsraum, in dem die Bereiche mit erster Invariante außerhalb des eingeschränkten Intervalls ausgeblendet wurden. Nur die untere Hälfte der Metallscheibe sowie der innere Rand sind noch blau. Da  $K_1$  als Maß des isotropen Anteils der Verformungstensoren interpretiert werden kann, lässt sich daraus schlussfolgern, dass in diesen Bereichen starke isotrope Verformung stattfindet. Ein interessantes Detail ist, dass auf der Rückseite der Metallscheibe der blaue Bereich leicht größer zu sein scheint.

Abb. 34c zeigt die Einschränkung des Intervalls der zweiten Invariante. Hier ist wiederum die untere Schranke erhöht. Dies ist an der Form der Interaktionswidgets erkennbar. Im zylindrischen Koordinatensystem entspricht die zweite Invariante der Entfernung zur zentralen x-Achse, weshalb der ausgeblendete Bereich als halbkreisförmiger weißer Ausschnitt auf den Interaktionswidgets dargestellt ist. Dies umfasst wiederum etwa 40% des Invariantenintervalls. Die Bereiche des Objekts, deren Invarianten im selektierten Intervall liegen, sind in Abb. 34d zu sehen. Sie befinden sich in einem halbkreisförmigen Bereich an einer Stelle am inneren Rand der Scheibe. Aus der Interpretation von  $K_2$  als Maß für die Anisotropie lässt sich schließen, dass dort starke anisotrope Verformung stattfindet.

Zuletzt zeigt Abb. 34e die Einschränkung des Intervalls der dritten Invariante. Diese entspricht in zylindrischen Koordinaten dem Winkel zur xx-Ebene. Die Einschränkung wird durch die Winkel der geraden Seitenflächen der Interaktionswidgets dargestellt. Die untere Schranke wird dabei um 80% der Intervalllänge erhöht. Abb. 34f zeigt die Bereiche des Objekts mit hoher dritter Invariante. Diese befinden sich in einem halbkreisförmigen Bereich im Inneren der Scheibe sowie entlang des gesamten inneren Randes.

Aus den Darstellungen der Invariantenbereiche im Objekt lassen sich eine Reihe von Schlüssen ziehen. Zunächst bestätigen sich die aus der Ellipsoiddarstellung abgelesenen

Informationen: Die Tensoren am inneren Rand und insbesondere im oberen Bereich des inneren Randes sind sehr viel stärker als im Rest der Metallscheibe. Auch, dass die Tensoren im unteren Bereich des Scheibenausschnitts stärker sind als im oberen stimmt überein. Die zusätzliche Zunahme der Anisotropie von oben nach unten kann auf den gezeigten Bildern nicht abgelesen werden. Dies liegt daran, dass die Anisotropie im unteren Bereich erheblich schwächer ist als am inneren Rand, weshalb die Interaktion diese mit ausblendet.

Zusätzlich lassen sich Informationen ablesen, die die Ellipsoiddarstellung nicht bietet. Aus der Kombination aus hohem  $K_2$  und  $K_3$  am inneren Rand beispielsweise folgt, dass dort hohe orthotrope Anisotropie vorliegt. Interessant sind auch die leicht unterschiedlichen Verteilungen von  $K_1$  auf der Vorder- und Rückseite der Scheibe. Dies ist wahrscheinlich auf die Temperaturunterschiede zurückzuführen.

Die erzeugten 3D Texturen bestehen aus  $400 \times 400 \times 400$  Voxeln im Invariantenraum und  $600 \times 600 \times 100$  Voxeln im Ortsraum. Wie schon zuvor wurde die Anzahl der Voxel im Ortsraum an die Form des Objektes angepasst, um Speicher zu sparen und die Qualität des DVR zu erhöhen.

Die Voxelisierung zu 3D Texturen benötigt insgesamt etwa 3,3 Sekunden, die Rechenzeit für ein einzelnes Bild betrug im Durchschnitt ca. 0,1 Sekunden.

### 7.3 Biegebalken

Die letzten zwei Testdatensätze stammen aus der Arbeit von Raith et al. [33]. Der erste der beiden Datensätze beschreibt die Eigenwerte innerhalb eines sogenannten Biegebalkens. Raith et al. beschreiben den Biegebalken wie folgt[33, S. 1128]:

Der Datensatz ist die Simulation eines waagerechten Balkens, der an einer Seite befestigt ist, während auf die andere Seite eine Kraft einwirkt. Generiert wurde er durch die Simulationssoftware Abaqus[37]. Die Dimensionen des Balkens sind  $30\text{ mm} \times 30\text{ mm} \times 100\text{ mm}$ . Die einwirkende Kraft beträgt  $1000\text{ N}$  und wirkt in negative y-Richtung. Der Balken besteht aus einem Metall mit einem Elastizitätsmodul von  $210\text{ GPa}$  und einer Poissonzahl von  $0,3$ . Der Datensatz besteht aus 4320 Tetraederzellen.

Der Biegebalken wird im Folgenden immer so dargestellt, dass das befestigte Ende auf den Bildern rechts liegt und die Kraft nach unten wirkt. Als Invarianten werden die Werte der drei Hauptspannungen verwendet. Diese entsprechen den Eigenwerten der Spannungstensoren an den jeweiligen Punkten, geordnet nach ihrem Betrag. Die erste Invariante ist also stets der höchste, die zweite der mittlere und die dritte der kleinste der drei Eigenwerte. Da die Tensoren symmetrisch sind und nur reelle Werte enthalten, existieren immer drei Eigenwerte.

Ziel des Tests dieses Datensatzes ist es, die Ergebnisse von Raith et al. zu reproduzieren sowie Vor- und Nachteile der erzeugten Visualisierungen mit denen von Raith et al. zu vergleichen.

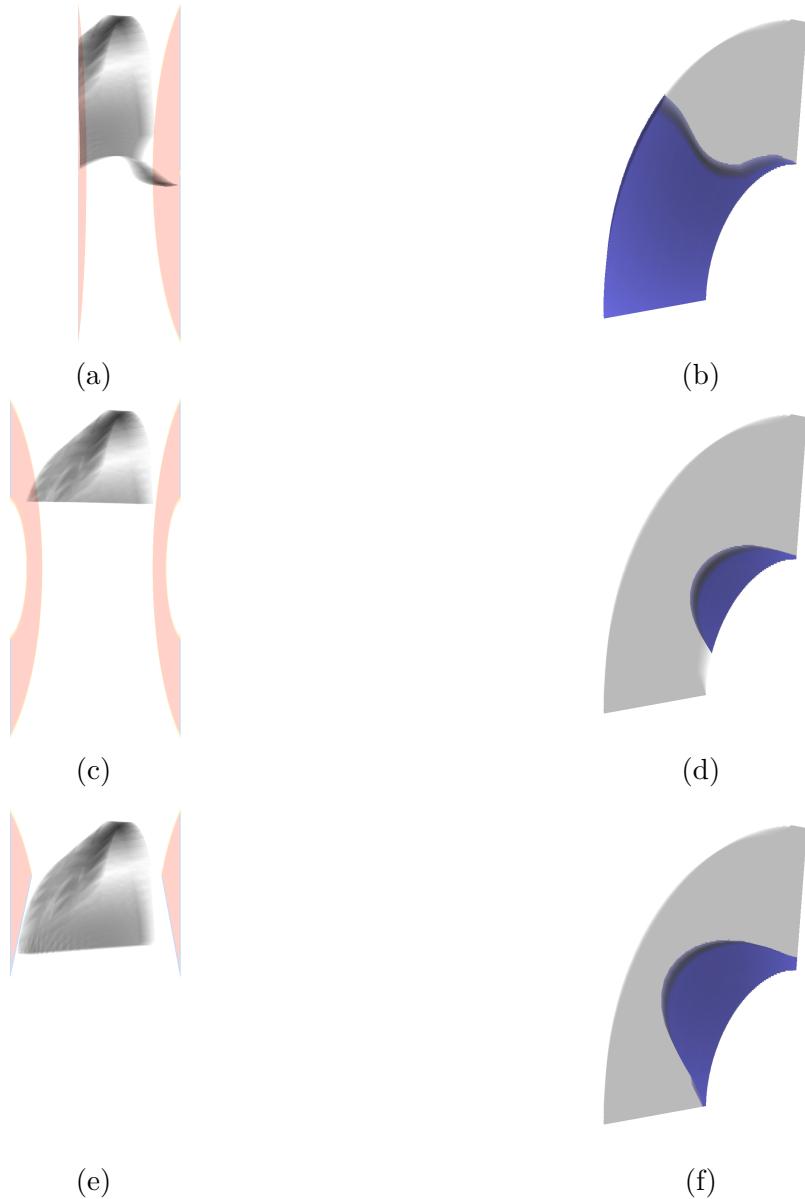


Abbildung 34: Die Verformung des Metallscheiben-Datensatzes mit ausgewählten Invariantenbereichen. In (a) und (b) ist wurden die unteren 40% des Intervalls von  $K_1$  ausgeblendet, in (c) und (d) die unteren 40% von  $K_2$ , in (e) und (f) die unteren 80% von  $K_3$ .

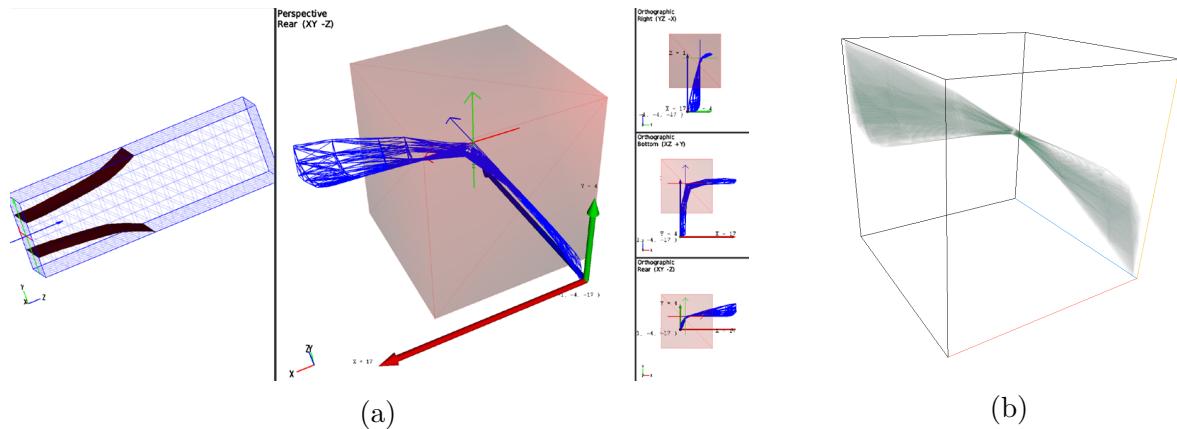


Abbildung 35: Darstellungen des Biegebalken-Datensatzes: (a) die Darstellungen von Raith et al., entnommen aus [33], (b) eine Darstellung als DVR im Invariantenraum

In Abb. 35a ist die Visualisierung von Raith et al. abgebildet: Auf der rechten Seite werden die Positionen und Formen der Zellen des Datensatzes im Invariantenraum durch eine Wireframe-Darstellung (WF-Darstellung) angezeigt. Der transparente, rote Quader zeigt die Grenzen eines durch Interaktion ausgewählten Teils des Invariantenraumes, sehr ähnlich zu dem innerhalb der vorliegenden Arbeit implementierten quaderförmigen Interaktionswidget. Gut sichtbar ist, dass der Quader die WF-Darstellung nur in positiver Richtung der x-Achse (roter Pfeil) und negativer Richtung der z-Achse (blauer Pfeil) schneidet. Kleine achsenparallele Sichten auf die WF-Darstellung und den Quader sind am rechten Rand abgebildet.

Die linke Seite zeigt eine WF-Darstellung, die die Zellen des Biegebalkens im Ortsraum visualisiert. Indem die Schnittflächen zwischen dem Quader und den Zellen der WF-Darstellung im Invariantenraum berechnet und zurück in den Ortsraum transformiert werden, entstehen die beiden gebogenen, roten Flächen in der WF-Darstellung. Da die Schnittflächen im Invariantenraum jeweils parallel zur xy- und yz-Ebene sind, stellen die roten Flächen Isosurfaces der ersten und zweiten Invariante dar, deren Isowerte durch die Positionen der Seitenflächen des Quaders spezifiziert werden.

Abb. 35b ist die Darstellung desselben Datensatzes im Invariantenraum als DVR. Das DVR wurde so gedreht, dass die Kameraposition und Perspektive dem der Visualisierung von Raith et al. ähnelt. Die unterschiedliche Form ist überwiegend auf Unterschiede in der Position der Kamera und bei der Implementierung der perspektivischen Projektion zurückzuführen.

Bei der Erstellung des DVR trat ein Problem auf. Sehr hohe Dichtewerte in dem Bereich, in dem sich die beiden v-förmigen Komponenten im Invariantenraum treffen führten zu Problemen mit dem Skalierungsfaktor der 3D Textur. Um die Auswirkungen auf die Visualisierung zu minimieren, verwenden wir die in 6.9.3 beschriebene Option, um die minimale Dichte eines Voxels auf 5% der maximal vorkommenden Dichte zu begrenzen.

Dadurch werden Fehler minimiert und die Analyse vereinfacht. Diese Option wurde in Abb. 35b, 36a, 36c und 36e verwendet.

In dem Bereich, an dem sich die beiden v-förmigen Komponenten im Invariantenraum treffen nimmt die durch die Voxelisierung berechnete Dichte im Vergleich zum Rest der 3D Textur um bis zu 20 Größenordnungen höhere Werte an. Um diese hohen Werte dennoch abspeichern zu können, muss ein sehr hoher Skalierungsfaktor verwendet werden. Dies führte wiederum dazu, dass die abgespeicherten Dichtewerte im Rest der 3D Textur sehr klein wurden, weshalb OpenGL sie auf 0 abrundet. Dadurch waren die v-förmigen Bereiche im DVR unsichtbar. Um dies zu verhindern, wurde die in Abschnitt 6.9.3 beschriebene Funktion zur Festlegung einer minimalen Dichte benutzt. Indem die minimale Dichte auf 5% der maximal vorkommenden Dichte gesetzt wird, werden die v-förmigen Bereiche sichtbar.

Dadurch tritt jedoch ein weiteres Problem auf: Die v-förmigen Komponenten werden opaker gezeichnet als der Bereich, in dem sie zusammentreffen. Da Opakheit durch die Transferfunktion mit Werten hoher Dichte assoziiert wird, führt dies möglicherweise zu falschen Annahmen über die Dichteverteilung im Invariantenraum. Tatsächlich ist dies jedoch auf die Skalierung der Dichtewerte mit der Anzahl der Abtastpunkte pro Strahl im Raycasting zurückzuführen.

Tatsächlich liegt der Bereich, an dem die beiden v-förmigen Komponenten zusammentreffen, nah am Punkt (0;0;0). Das erklärt auch, warum die Dichtewerte dort so hoch sind: Im Großteil des Biegebalkens nehmen die Eigenwerte nur sehr kleine Beträge an, da hier nur relativ wenig Kraft wirkt. Dadurch wird ein Großteil des Volumens des Biegebalkens auf diesen sehr kleinen Bereich abgebildet, was durch das Continuous Scatterplotting zu sehr hohen Dichtewerten führt. Wie zuvor wird bei den DVR im Invariantenraum das Wireframe des quaderförmigen Interaktionswidgets eingeblendet, um die Größe der ausgeblendeten Bereiche besser einschätzen zu können.

Die von Raith et al. erzeugte Visualisierung wählt durch das von ihnen implementierte Interaktionswidget gleichzeitig eingeschränkte Intervalle aller drei Invarianten aus. Es bietet sich daher an, diesen Datensatz zu nutzen, um die Funktion des gleichzeitigen Einschränkens mehrerer Intervalle mithilfe des quaderförmigen Interaktionswidgets zu demonstrieren. Dafür ist in Abb. 36a die obere Schranke des größten Eigenwerts auf den Wert 9 festgesetzt, wie es auch in der Visualisierung von Raith et al. der Fall ist. Der Wert wird im Artikel von Raith et al. nicht explizit erwähnt und wurde mündliche nachgefragt. Im Invariantenraum entspricht dies der Bewegung der vorderen linken Seitenfläche des Interaktionswidgets nach hinten rechts. Erkennbar ist, dass, wie auch bei Raith et al., die linke v-förmige Komponente geschnitten und der Teil außerhalb des Intervalls ausgeblendet wird. Der durch die Einschränkung des Intervalls ausgeblendeten Bereich des Objekt DVR ist in Abb. 36b dargestellt, ein Bereich in der linken Hälfte der oberen Kante des Balkens. Deutlich erkennbar ist die Ähnlichkeit der Form der Grenze zwischen dem blauen und dem grauen Bereich des DVR und dem Verlauf der oberen Invarianten-Isofläche in der Visualisierung in Abb. 35a. Die von Raith et al. vorgeschlagene Erklärung für Form und Position dieses Bereichs liegt in der Bedeutung

der Eigenwerte bzw. Hauptspannungen. Hauptspannungen mit hohen positiven Werten weisen dabei auf Bereiche mit Zugspannungen hin, also Spannungen, die das Material in diesen Bereichen auseinanderziehen, was intuitiv im Biegebalken an dieser Stelle der Fall ist. Ein wichtiger Vorteil gegenüber der Visualisierung von Raith liegt darin, dass der Nutzer direkt erkennen kann, welche Bereiche innerhalb des Interaktionswidgets liegen, anstatt nur Schnittflächen angezeigt zu bekommen. Besonders bei komplexen Datensätzen kann dies die Analyse vereinfachen.

Bei der zweiten Visualisierung, dargestellt in Abb. 36c, wird die untere Schranke des kleinsten Eigenwertes auf -9 festgelegt. Auch dies entspricht dem im Artikel von Raith et al. verwendeten Wert und wurde ebenfalls mündlich erfragt. Im Invariantenraum ist dies dargestellt durch die Bewegung der rechten vorderen Seitenfläche des Interaktionswidgets nach hinten links. Wie zuvor wird dadurch eine der v-förmigen Komponenten geschnitten, dieses Mal ist es jedoch die rechte. Die zugehörige Darstellung der ausgeblendeten Bereiche im Objekt sind in Abb. 36d abgebildet. Es handelt sich um einen Bereich in der linken Hälfte der unteren Kante des Balkens. Wiederum zeigt sich die Ähnlichkeit zu einer der Isoflächen in der Visualisierung von Raith et al. deutlich, dieses Mal betrifft es die untere Isofläche. Die von Raith et al. vorgeschlagene Interpretation dieses Bereichs lässt sich auch auf die Werte der Hauptspannungen zurückführen. Hauptspannungen mit negativen Werten und hohen Beträgen treten in Bereichen auf, in denen das Material zusammengedrückt wird. Man spricht auch von Druckspannungen. Die Interpretation stimmt mit der Intuition überein, dass das Material des Biegebalkens an dieser Stelle zusammengepresst wird.

Abb. 36e zeigt die Kombination beider zuvor erläuterter Intervalleinschränkungen. Eine Einschränkung in Richtung des mittleren Eigenwerts ist nicht notwendig, da wie schon erwähnt das von Raith et al. implementierte Interaktionswidget das DVR des Biegebalkens im Invariantenraum in Richtung der y-Achse nicht schneidet. Abb. 36f zeigt die Wirkung auf das DVR im Ortsraum. Dabei wird deutlich, dass die ausgeblendeten Bereiche spiegelsymmetrisch gegenüber einer horizontalen Achse durch das Objekt sind. Auch dies stimmt mit den Ergebnissen von Raith et al. überein.

Durch die Form des DVR im Invariantenraum bietet sich der Biegebalken-Datensatz dazu an, den explorativen Anteil der Interaktionen zu demonstrieren. Hier zeigt sich der Vorteil der implementierten Interaktionen: Bei einer Verschiebung des Interaktionswidgets wird der angezeigte Bereich im Ortsraum direkt angepasst. Im Gegensatz dazu benötigten die Interaktionen in der Visualisierung von Raith et al. für diesen Datensatz bereits ein vielfaches länger. Zur Demonstration wurde in Abb. 39 in Anhang 10.1 die obere Intervallgrenze schrittweise reduziert, sodass der Verlauf der Werte der ersten Invariante im DVR des Ortsraums sichtbar wird. In den ersten Schritten in Abb. 39a bis 39d vergrößert sich der im Biegebalken ausgeblendete Bereich nur langsam. Die Beträge der durch das Einwirken der Kraft erzeugten Spannungen nehmen scheinbar mit größerer Entfernung zum Einwirkungspunkt stark ab. Das Wachstum des ausgeblendeten Bereichs des Biegebalkens zwischen den Abbildungen nimmt über Abb. 39e bis 39n hinweg zu. Wenn sich die obere Schranke der ersten Invariante jedoch an den Wert 0 annähert, zu

sehen in Abb. 39o bis 39q, wird das Wachstum wieder langsamer. Somit hat scheinbar ein erheblicher Teil des Biegebalkens eine mittelgroße erste Invariante. Zwischen Abb. 39q bis 39t sind deutliche Sprünge in der Ausdehnung der ausgeblendeten Bereiche des DVR im Ortsraum zu erkennen. Da, wie schon beschrieben, in einem großen Teil des Biegebalkens nur sehr kleine Kräfte wirken, ist dieses Verhalten zu erwarten.

Interessant ist besonders der nicht ausgeblendete Bereich in Abb. 39t. Die obere Schranke der ersten Invariante hat dort einen negativen Wert. Da die erste Invariante jedoch der höchsten Hauptspannung entspricht, bedeutet dies, dass dort alle drei Hauptspannungen negativ sein müssen. Somit wird in einem kleinen Bereich am linken Rand des Balkens das Material aus allen Richtungen zusammengedrückt.

Die erzeugten 3D Texturen hatten eine Größe von  $400 \times 400 \times 400$  im Invariantenraum und  $400 \times 400 \times 800$  im Ortsraum. Das Erzeugen der 3D Texturen benötigte etwa 3,1 Sekunden und die Rechenzeit für ein einzelnes Bild betrug etwa 0,05 Sekunden.

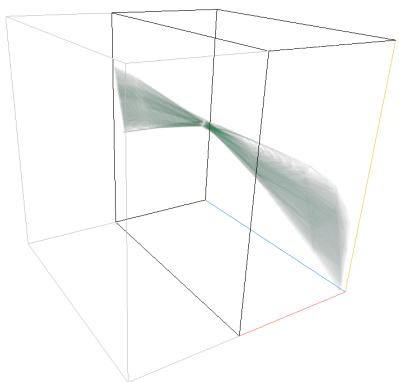
## 7.4 Metallkomponente

Wie auch der Biegebalken-Datensatz stammt der letzte Datensatz aus der Arbeit von Raith et al. Dort wird er wie folgt beschrieben [33, S. 1129]:

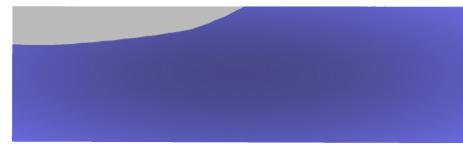
Der Datensatz repräsentiert ein aus drei Komponenten zusammengesetztes Werkstück. Die erste Komponente ist ein Metalleinsatz, von dem drei Seitenbalken ausgehen. Der Metalleinsatz wird von der zweiten Komponente, einem Polymer, umschlossen. Das Polymer wiederum ist von Fasern durchzogen, die die dritte Komponente bilden. Der Datensatz ist eine Simulation, die mithilfe von Abaqus[37] und Helios[16] erstellt wurde. Da das Werkstück symmetrisch entlang einer Schnittfläche parallel zur yz-Ebene ist, wurde nur die rechte Hälfte simuliert. Um Fehler durch die Halbierung zu verhindern, wurden entlang der Schnittfläche spezielle symmetrische Bedingungen simuliert. An der oberen und rechten Kante wird das Werkstück als fixiert angenommen, während an der unteren Kante des Metalleinsatzes eine Kraft von 3000 N in negative y-Richtung einwirkt. Durch die Form und das Material der Komponenten sowie das spezielle Verhalten an der Schnittfläche ist das Verhalten des Werkstücks unter Krafteinwirkung erheblich komplexer und schwieriger zu verstehen als in den vorherigen Beispielen, was auch zu einer komplexeren Form des Datensatzes im Invariantenraum führt.

Die verwendeten Invarianten sind wieder die Werte der Hauptspannungen, geordnet nach ihrer Größe.

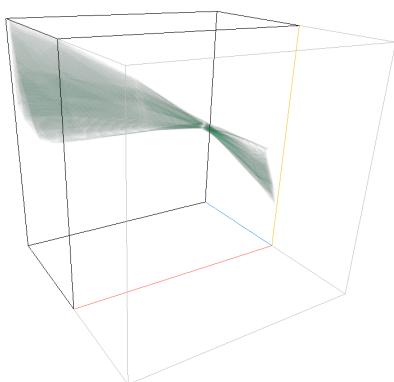
Dieser Datensatz wurde aus mehreren Gründen ausgewählt. Erstens ist er deutlich komplexer als die bisher vorgestellten Datensätze. Diese Komplexität stellt höhere Anforderungen an die Visualisierung, wie sie auch bei realen Anwendungen auftreten würden. Zweitens enthält der Datensatz mit 2.541.049 Tetraederzellen deutlich mehr Zellen als die anderen vorgestellten Datensätze. Und drittens existiert mit der Arbeit



(a)



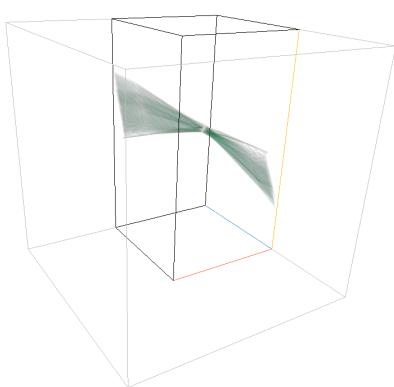
(b)



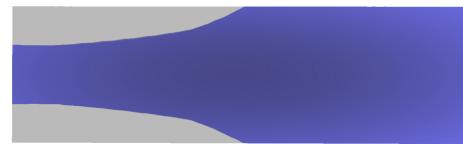
(c)



(d)



(e)



(f)

Abbildung 36: Der Biegebalken-Datensatz mit quaderförmigen Interaktionen: (a) und (b) zeigen Bereiche, deren höchster Eigenwert kleiner als 9 ist, (c) und (d) Bereiche in denen der kleinste Eigenwert größer als -9 ist, (e) und (f) Bereiche in denen der größte Eigenwert kleiner als 9 und der kleinste größer als -9 ist

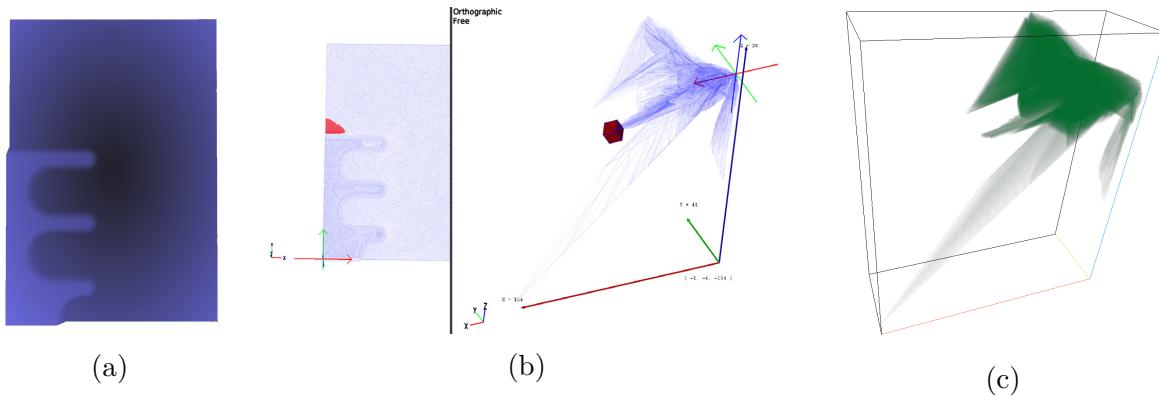


Abbildung 37: Darstellungen des Metallkomponenten-Datensatzes: (a) ein DVR des Datensatzes im Ortsraum, (b) die von Raith et al. erzeugten Visualisierungen und (c) ein DVR des Datensatzes im Invariantenraum.

von Raith et al. ein Referenzpunkt, mit dem die erzeugten Ergebnisse verglichen werden können.

In Abb. 37 sind Visualisierungen des Metallkomponenten-Datensatzes abgebildet. Abb. 37a zeigt ein DVR des Objekts. Unten links ist die Hälfte des Metalleinsatzes mit seinen drei Seitenbalken zu erkennen. Abb. 37b zeigt die von Raith et al. erzeugten Visualisierungen. Wie auch beim Biegebalken-Datensatz werden die Zellen durch eine WF-Darstellung visualisiert. Auf der rechten Seite ist die WF-Darstellung im Invariantenraum zu sehen. Interessant dabei ist, dass das Koordinatensystem linkshändig ist, also x- und y-Achse vertauscht sind. Ein kleiner Bereich des Invariantenraumes ist ausgewählt, dargestellt durch das rote, quaderförmige Interaktionswidget.

Im linken Teil von Abb. 37b ist die WF-Darstellung des Objekts zu sehen. Die Schnittflächen zwischen den Seitenflächen des Quaders und den Zellen der WF-Darstellung im Invariantenraum wurden zurück in den Ortsraum transformiert und sind in rot eingezeichnet. Der dadurch markierte Bereich wird von Raith et al. als eine Region beschrieben, in der üblicherweise Beschädigungen im Material auftreten.

Abb. 37c zeigt den Datensatz als DVR im Invariantenraum. Das in dieser Arbeit implementierte DVR-Verfahren verwendet ein rechtshändiges Koordinatensystem. Um die Ähnlichkeit zum rechten Teil von Abb. 37b, der in linkshändigen Koordinaten erzeugt wurde, zu erhöhen, wurde Abb. 37c vertikal gespiegelt.

Wegen der Komplexität des Datensatzes ist es ohne gründliche Kenntnisse der physikalischen Prozesse schwierig, Informationen aus der Form des DVR im Invariantenraum oder aus den Formen der ein- und ausgeblendeten Teile des DVR im Ortsraum zu ziehen. Die vorliegende Arbeit beschränkt sich daher darauf, die von Raith et al. erzeugten Visualisierungen zu reproduzieren und zu vergleichen.

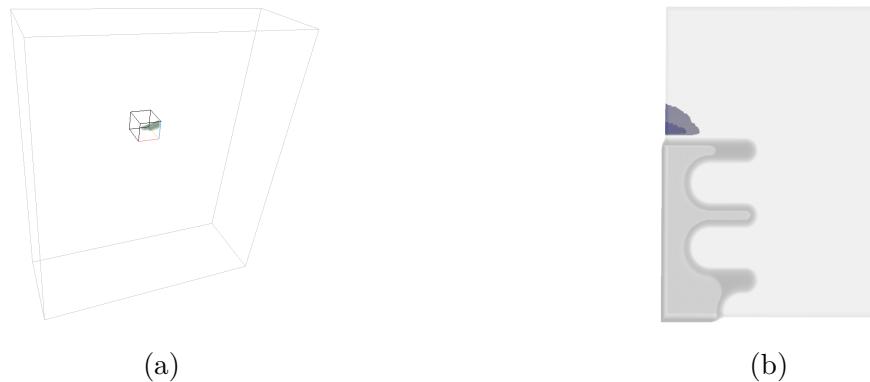


Abbildung 38: Der Metallkomponenten-Datensatz, eingeschränkt auf einen kleinen quaderförmigen Bereich mit hohen Werten der ersten und zweiten sowie mittleren Werten der dritten Invariante: (a) Darstellung im Invariantenraum und (b) Darstellung im Ortsraum.

Abb. 38 zeigt den Metallkomponenten-Datensatz, in dem die Intervalle aller drei Invarianten eingeschränkt wurden. Durch Interaktion mit dem quaderförmigen Interaktionswidget wurden die oberen und unteren Schranken aller Invariantenintervalle eingeschränkt. Dies ist in Abb. 38a dargestellt. Der ausgewählte Bereich zeigt große Ähnlichkeit zu der Form und Position des Interaktionswidgets in der Visualisierung von Raith et al. in Abb. 37b. Der Bereich im Ortsraum, in dem Invarianten innerhalb des gewählten Intervalls auftreten, ist in Abb. 38b dargestellt: Ein kleiner Bereich am oberen Rand der Metallkomponente. Dieser entspricht in Form und Position den Ergebnissen von Raith et al., wie sie in Abb. 37b zu sehen sind.

Die 3D Textur des Invariantenraumes besteht aus  $400 \times 400 \times 200$  Voxeln, die des Ortsraums aus  $600 \times 600 \times 100$  Voxeln. Die Berechnung der 3D Texturen benötigte insgesamt ca. 30 Sekunden, was auf die sehr viel größere Anzahl an Tetraedern in diesem Datensatz zurückzuführen ist. Um ein einzelnes Bild zu erzeugen wurden durchschnittlich ca. 0,08 Sekunden benötigt.

## 7.5 Fazit der Ergebnisse

Die implementierte Visualisierung wurde anhand der vorgestellten Beispiele demonstriert.

Es konnte gezeigt werden, dass die gleichzeitige Darstellung eines Datensatzes im Orts- und Invariantenraum es ermöglicht, aus der Position eines Punktes im Invariantenraum Informationen über den an diesem Punkt definierten Tensor im Ortsraum zu ziehen. Auch die Form des Feldes im Invariantenraum kann interessante Einblicke in die Struktur des Feldes bieten. Da die Invarianten domänen spezifische Eigenschaften der Tensoren repräsentieren, ist es somit möglich, Bereichen der Objekte Eigenschaften zuzuordnen. Die Kernaufgabe der Visualisierung, Informationen über die Struktur von Tensorfeldern

zu vermitteln, ist somit erfüllt. Die Korrektheit der Visualisierung wurde durch Vergleich mit der Arbeit von Fritzsch und Raith et al. nachgewiesen.

Zusätzlich konnte gezeigt werden, dass die vorgestellte Visualisierung einige Vorteile gegenüber anderen Tensorfeldvisualisierungen bietet. Verglichen mit der Darstellung durch Ellipsoidglyphen wurde deutlich, dass es in der implementierten Visualisierung durch die Interaktionen deutlich einfacher war Eigenschaften der Tensoren und ihre Position im Objekt abzulesen. Auch kleine Veränderungen, wie die unterschiedlichen Spannungen an Vorder- und Rückseite der Metallscheibe, die bei der Ellipsoidglyphendarstellung nicht erkennbar waren, wurden sichtbar dargestellt.

Die Verbesserungen gegenüber der Arbeit von Fritzsch liegen in der gleichzeitigen Darstellung aller drei Invarianten und den implementierten Interaktionen. Dadurch wurde die Interpretierbarkeit der Darstellung stark erhöht. Im Vergleich mit der Visualisierung von Raith et al. konnten die Vorteile der Darstellung als DVR und der kürzeren Rechenzeit der Interaktionen gezeigt werden. Während bei Raith et al. die Visualisierung ca 3,0 Sekunden benötigt, um den Effekt einer Interaktion darzustellen, geschieht dies in der von uns implementierten Visualisierung in weniger als 0,1 Sekunden. Dadurch wird die explorative Analyse des Datensatzes möglich.

Wie für alle vorgestellten Datensätze gezeigt werden konnte, stellt die Repräsentation von Tensoren durch Invarianten eine sinnvolle Reduktion der Datenmenge pro Tensor dar. **Herausforderung 1** ist somit erfüllt. Dadurch, dass die Invarianten abhängig von der Domäne des Datensatzes ausgewählt werden können sind ebenfalls **Herausforderungen 3 und 4** gelöst. Dies wird besonders deutlich, wenn man die Visualisierungen des Metallscheiben Datensatzes mit dem des Biegebalkens vergleicht. Dort veränderte sich die Interpretation der Visualisierung durch die Wahl unterschiedlicher Invariantensätze stark.

Die **Herausforderung 2** kann nur als teilweise gelöst betrachtet werden. Zwar konnte am Beispiel des Metallkomponenten-Datensatzes gezeigt werden, dass auch Datensätze mit mehreren Millionen Tensoren dargestellt und durch Interaktionen in akzeptabler Zeit manipuliert werden können. Die Voxelisierung skaliert jedoch schlecht mit der Anzahl an Tetraedern, sowohl in Hinblick auf Laufzeit als auch auf die Erkennbarkeit der Form einzelner Tetraeder. Insbesondere extrem hohe Dichtewerte durch Überlappung von Tetraedern stellen ein Problem dar, wie am Beispiel des Biegebalkens gezeigt wurde. Die verwendete Lösung manipuliert die Dichtewerte im Datensatz so stark, dass die Visualisierung einen falschen Eindruck der Dichteverteilung im Invariantenraum vermittelt.

Auch die implementierten Interaktionen wurden ausführlich demonstriert. Es haben sich zwei unterschiedliche Anwendungsfälle für die Interaktionen ergeben. Der erste Fall, demonstriert im Biegebalken- und Metallkomponenten-Datensatz, besteht darin, einen spezifischen Bereich im Invariantenraum auszuwählen und die zugehörigen Bereiche im Ortsraum zu betrachten. Dazu muss der Nutzer die spezifischen Bereiche des Invariantenraumes bereits kennen oder aus der Form des DVR im Invariantenraum ablesen können. Wenn dies nicht gegeben ist, können die Interaktionen nur für den zweiten Anwendungsfall

verwendet werden. Dieser entspricht der explorativen Analyse des Datensatzes durch Interaktion. Durch die Interaktionen werden beliebige Intervalle von Invarianten ausgewählt und angezeigt. Indem die Form und Verteilung der Invarianten interpretiert wird, kommt es zu einem Gewinn an Informationen. Dies wurde im Single Point Load-, Metallscheiben- und Biegebalken-Datensatz demonstriert. Ein Sonderfall der explorativen Analyse ist beim Biegebalken Datensatz dargestellt. Indem der Ortsraum betrachtet wird während die Invariantenintervalle schrittweise eingeschränkt werden, werden Informationen über die Verteilung der Invarianten innerhalb des Objekts gewonnen. Insgesamt konnte gezeigt werden, dass die implementierten Interaktionen dabei helfen **Herausforderungen 2 und 3** zu lösen.

Obwohl die Interpretation der Form des Feldes im Invariantenraum noch immer ein Problem darstellt, konnte beim Biegebalken- und Metallkomponenten-Datensatz gezeigt werden, dass sichtbare Strukturen im Invariantenraum Bereichen mit interessanten Eigenschaften im Ortsraum entsprechen.

## 8 Ausblick

Die Ergebnisse wurden Experten für Kontinuumsmechanik von der Technischen Universität Dortmund und dem Helmholtz-Zentrum für Umweltforschung vorgeführt, wobei die Reaktionen positiv waren. Es wurde jedoch kritisiert, dass die explorative Analyse intensive Kenntnisse der Datensätze voraussetzt. In der Praxis werden sogenannte Materialmodelle eingesetzt, die das Verhalten von Materialien beschreiben. Ausgehend davon lassen sich Invariantenbereiche definieren. Diese sind jedoch nicht immer zylinderachsenparallel oder quaderförmig wie die bereits implementierten Interaktionswidgets. Dies wäre z.B. bei einem Material der Fall, das bei höheren isotropen Spannungen auch höhere anisotrope Spannungen aushält. Ein solches Materialmodell würde im K-Invariantenraum die Form eines Kegels annehmen.

Das Laden von Materialmodellen und Auswählen der entsprechenden Bereiche könnte eine lohnenswerte Erweiterung des Systems darstellen. Die Interpretation durch den Nutzer würde erleichtert, da er die Bereiche nicht mehr per Hand einstellen müsste und daher weniger Kenntnisse über die Invarianten und ihre Bedeutung notwendig wären. Gleichzeitig wären die Ergebnisse exakter und besser in der Praxis anwendbar. Dadurch würden **Herausforderungen 3 und 4** besser gelöst werden.

Ein weiterer Ansatzpunkt für Verbesserungen wäre das Voxelisierungsverfahren. Da die Voxelisierung für einen Großteil der Rechenzeit der Visualisierung verantwortlich ist, würden Optimierungen dort die Gesamlaufzeit deutlich reduzieren. Dadurch wäre es auch möglich, die Masseerhaltung durch Supersampling zu implementieren, ohne die benötigte Zeit zu stark zu erhöhen. Auch Probleme wie dünne, ebenenparallele Bereiche könnten durch Verbesserungen der Voxelisierung behoben werden, genauso wie Verfahren zur

Reduktion der Treppeneffekte. **Herausforderung 2** könnte durch diese Verbesserungen wirksamer bearbeitet werden als es im Moment der Fall ist.

Wie in den Vergleichen mit den Ergebnissen von Fritzsch erwähnt ist die Implementierung einer Option zum Umschalten auf orthogonale Projektion im DVR eine möglicherweise lohnenswerte Erweiterung. Dadurch würde das Ablesen von Koordinaten aus der Visualisierung erleichtert werden. Auch Optionen zur Erstellung von Röntgenbildern, MIPs oder Isoflächen sind vorstellbar.

Ebenfalls vorstellbar ist die Verwendung eines alternativen DVR-Verfahrens, um Probleme der Voxelisierung komplett zu umgehen. Eine Möglichkeit dafür könnte eine stark modifizierte Cell Projection sein. Dadurch würde sich jedoch wahrscheinlich die für Interaktion benötigte Rechenzeit deutlich erhöhen.

Zuletzt bieten auch die Interaktionswidgets Raum für Weiterentwicklung, besonders in Hinblick auf Affordances, Hervorhebung der Handles bei Selektion und der visuellen Abhebung der Widgets von den DVRs.

Weitere Verbesserungsmöglichkeiten werden wahrscheinlich deutlich werden, wenn das System in der Praxis eingesetzt wird.

## 9 Zusammenfassung

Innerhalb der vorliegenden Arbeit wurde ein System entwickelt und evaluiert, das die Untersuchung von Tensorfeldern mithilfe von Invariantensätzen ermöglicht. Es wurden mathematische, physikalische und technische Grundlagen erläutert und verschiedene DVR-Verfahren in Hinblick auf die Problemstellung verglichen. Ein Verfahren zur Erstellung dreidimensionaler Continuous Scatterplots wurde vollständig implementiert. Ebenfalls wurde ein Raycasting-Algorithmus implementiert, um die erzeugten dreidimensionalen Continuous Scatterplot darzustellen. Mithilfe von Interaktionen konnten die Zusammenhänge zwischen dem Invarianten- und dem Ortsraum dargestellt werden. Eine Vielzahl weiterer Interaktionen und Optionen, die es Nutzern ermöglichen, die Visualisierung an ihre Bedürfnisse anzupassen, wurden ebenfalls realisiert und vorgestellt. Zum Abschluss wurde die Visualisierung auf vier Datensätze angewendet, die Ergebnisse präsentiert und mit denen anderer Tensorfeldvisualisierungen verglichen. Dabei konnten die Ergebnisse der anderen Visualisierungen reproduziert und Vorteile der Visualisierung demonstriert werden. Anwendungsfälle für die Interaktionen und ihr Nutzen zur Analyse der vorgestellten Datensätze wurde gezeigt. Jedoch wurden auch Nachteile und Probleme des Verfahrens sichtbar, von denen ein Teil durch künftige Arbeit behoben werden könnte. Die in der Einleitung gestellten Herausforderungen konnten zum Großteil zufriedenstellend gelöst werden.

## Literatur

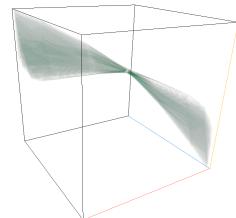
- [1] Universität Leipzig Abteilung für Bild- und Signalverarbeitung. *FAnToM Website*. URL: <http://www.informatik.uni-leipzig.de/fantom/content/about-fantom> (besucht am 04.02.2019).
- [2] Lisa S Avila u. a. *The VTK User's Guide*. Kitware New York, 2010.
- [3] Sven Bachthaler und Daniel Weiskopf. „Continuous scatterplots“. In: *IEEE transactions on visualization and computer graphics* 14.6 (2008), S. 1428–1435.
- [4] Peter J Bassler, James Mattiello und Denis LeBihan. „MR diffusion tensor spectroscopy and imaging“. In: *Biophysical journal* 66.1 (1994), S. 259–267.
- [5] Ray M Bowen und Chao-Cheng Wang. *Introduction to vectors and tensors*. Bd. 1. Courier Corporation, 2008.
- [6] The Qt Company. *Qt Website*. URL: <https://www.qt.io/> (besucht am 04.02.2019).
- [7] John C Criscione u. a. „An invariant basis for natural strain which yields orthogonal stress response terms in isotropic hyperelasticity“. In: *Journal of the Mechanics and Physics of Solids* 48.12 (2000), S. 2445–2465.
- [8] Timothy J Cullip und Ulrich Neumann. „Accelerating volume reconstruction with 3D texture hardware“. In: (1993).
- [9] Thierry Delmarcelle und Lambertus Hesselink. „Visualizing second-order tensor fields with hyperstreamlines“. In: *IEEE Computer Graphics and Applications* 13.4 (1993), S. 25–33.
- [10] Robert A Drebin, Loren Carpenter und Pat Hanrahan. „Volume rendering“. In: *ACM Siggraph Computer Graphics*. Bd. 22. 4. ACM. 1988, S. 65–74.
- [11] Daniel B Ennis und Gordon Kindlmann. „Orthogonal tensor invariants and the analysis of diffusion tensor magnetic resonance images“. In: *Magnetic resonance in medicine* 55.1 (2006), S. 136–146.
- [12] Richard P Feynman, Robert B Leighton und Matthew Sands. *The Feynman lectures on physics, Vol. I: The new millennium edition: mainly mechanics, radiation, and heat*. Bd. 1. Basic books, 2011.
- [13] Gideon Frieder, Dan Gordon und R Anthony Reynolds. „Back-to-front display of voxel based objects“. In: *IEEE Computer Graphics and Applications* 5.1 (1985), S. 52–60.
- [14] Clemens Fritzsch. „Visuelle Analyse kontinuumsmechanischer Simulationen durch kontinuierliche Streudiagramme“. Diplomarbeit. Universität Leipzig, 2016.
- [15] Charles D Hansen und Chris R Johnson. *The Visualization handbook*. Elsevier, 2005.
- [16] *Helios Website*. URL: <https://www.autodesk.de/products/helius-composite> (besucht am 06.02.2019).

- [17] Keith D. Hjelmstad. *Fundamentals of Structural Mechanics*. 10. Aufl. Springer US Verlag KG, 2005. ISBN: 978-3-13-477010-0.
- [18] Mario Hlawitschka u. a. „Top Challenges in alization of Engineering Tensor Fields“. In: *Visualization and Processing of Tensors and Higher Order Descriptors for Multi-Valued Data*. Springer, 2014, S. 3–15.
- [19] The Khronos™ Group Inc. *OpenGL Website*. URL: <https://www.khronos.org/opengl/> (besucht am 04.02.2019).
- [20] Michael I Jordan und Robert A Jacobs. „Hierarchical mixtures of experts and the EM algorithm“. In: *Neural computation* 6.2 (1994), S. 181–214.
- [21] Gordon Kindlmann. „Superquadric tensor glyphs“. In: *Proceedings of the Sixth Joint Eurographics-IEEE TCVG conference on Visualization*. Eurographics Association. 2004, S. 147–154.
- [22] Gordon Kindlmann und David Weinstein. „Hue-balls and lit-tensors for direct volume rendering of diffusion tensor fields“. In: *Proceedings of the conference on Visualization'99: celebrating ten years*. IEEE Computer Society Press. 1999, S. 183–189.
- [23] Gordon Kindlmann u. a. „Diffusion tensor analysis with invariant gradients and rotation tangents“. In: *IEEE Transactions on Medical Imaging* 26.11 (2007), S. 1483–1499.
- [24] Norbert Kusolitsch. *Maß-und Wahrscheinlichkeitstheorie: Eine Einführung*. Springer-Verlag, 2014.
- [25] Bruce R Kusse und Erik A Westwig. *Mathematical physics: applied mathematics for scientists and engineers*. John Wiley & Sons, 2010.
- [26] Philippe Lacroute und Marc Levoy. „Fast volume rendering using a shear-warp factorization of the viewing transformation“. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM. 1994, S. 451–458.
- [27] Ulrich Leiner, Bernhard Preim und Stephan Ressel. „Entwicklung von 3D-Widgets-Überblicksvortrag“. In: *Proceedings of Simulation und Animation: SCS Europe, Erlangen*, S (1997), S. 170–188.
- [28] Reiner Lenz u. a. „Display of Density Volumes“. In: *IEEE Computer Graphics and Applications* 6.7 (1986), S. 20–29. DOI: [10.1109/MCG.1986.276813](https://doi.org/10.1109/MCG.1986.276813). URL: <https://doi.org/10.1109/MCG.1986.276813>.
- [29] William E Lorensen und Harvey E Cline. „Marching cubes: A high resolution 3D surface construction algorithm“. In: *ACM siggraph computer graphics*. Bd. 21. 4. ACM. 1987, S. 163–169.
- [30] Tamara Munzner. *Visualization analysis and design*. AK Peters/CRC Press, 2014.
- [31] Bernhard Preim und Raimund Dachselt. *Interaktive Systeme: Band 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung*. Springer-Verlag, 2010.

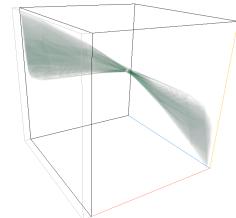
- [32] Bernhard Preim und Raimund Dachselt. *Interaktive Systeme: Band 2: User Interface Engineering, 3D-Interaktion, Natural User Interfaces*. Springer-Verlag, 2015.
- [33] Felix Raith u. a. „Tensor Field Visualization using Fiber Surfaces of Invariant Space“. In: *IEEE transactions on visualization and computer graphics* 25.1 (2019), S. 1122–1131.
- [34] Antonio J Rueda u. a. „Voxelization of solids using simplicial coverings“. In: (2004).
- [35] Gerik Scheuermann und Matthias Goldau. *Vorlesung Visualisierung in Naturwissenschaft und Technik*. 2015.
- [36] Peter Shirley und Allan Tuchman. *A polygonal approximation to direct scalar volume rendering*. Bd. 24. 5. ACM, 1990.
- [37] Dassault Systèmes. *Abaqus Website*. URL: <https://www.3ds.com/products-services/simulia/products/abaqus/> (besucht am 04.02.2019).
- [38] Lee Westover. „Footprint evaluation for volume rendering“. In: *ACM Siggraph Computer Graphics* 24.4 (1990), S. 367–376.
- [39] Lee Westover. „Interactive volume rendering“. In: *Proceedings of the 1989 Chapel Hill workshop on Volume visualization*. ACM. 1989, S. 9–16.
- [40] Alexander Wiebel u. a. „Fantom-lessons learned from design, implementation, administration, and use of a visualization system for over 10 years“. In: (2009).
- [41] Valentin Zobel und Gerik Scheuermann. „Extremal curves and surfaces in symmetric tensor fields“. In: *The Visual Computer* (2017), S. 1–16.

## 10 Anhang

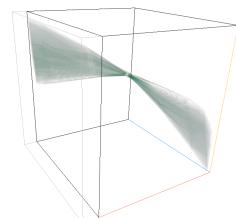
### 10.1 Anhang A



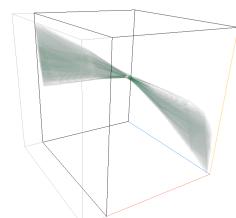
(a) 0%



(b) 5%

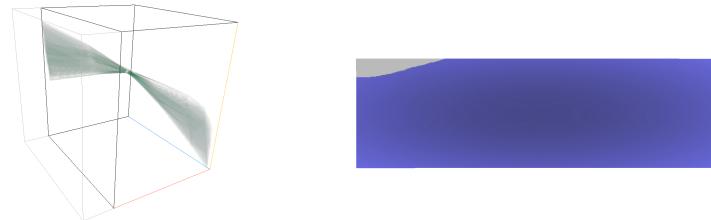


(c) 10%



(d) 15%

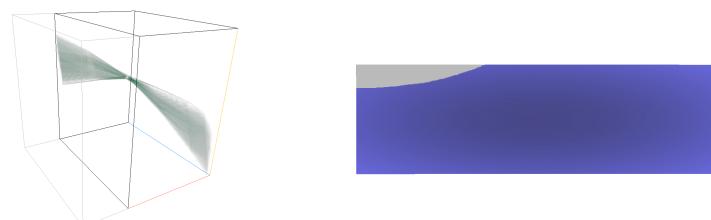
Abbildung 39: Darstellungen des Biegebalken Datensatzes. Die obere Schranke des größten Eigenwerts wurde jeweils um den angegebenen Prozentsatz der gesamten Intervalllänge verringert.



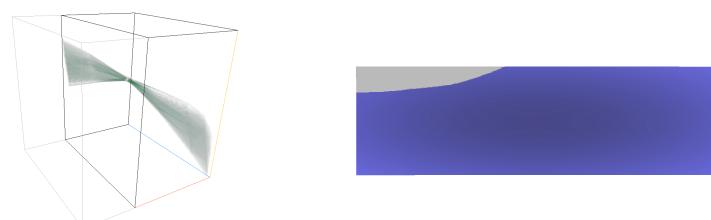
(e) 20%



(f) 25%

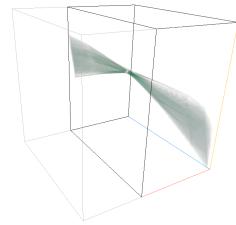


(g) 30%

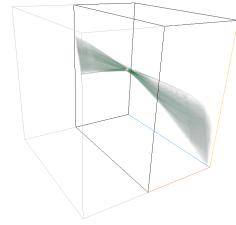


(h) 35%

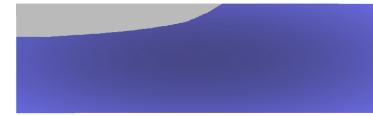
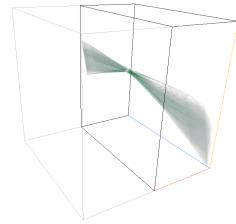
Abbildung 39: Darstellungen des Biegebalken Datensatzes. Die obere Schranke des größten Eigenwerts wurde jeweils um den angegebenen Prozentsatz der gesamten Intervalllänge verringert.



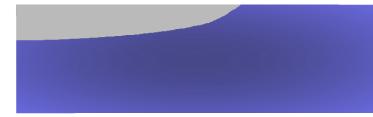
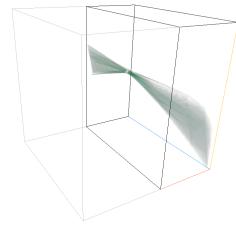
(i) 40%



(j) 45%



(k) 50%



(l) 55%

Abbildung 39: Darstellungen des Biegebalken Datensatzes. Die obere Schranke des größten Eigenwerts wurde jeweils um den angegebenen Prozentsatz der gesamten Intervalllänge verringert.

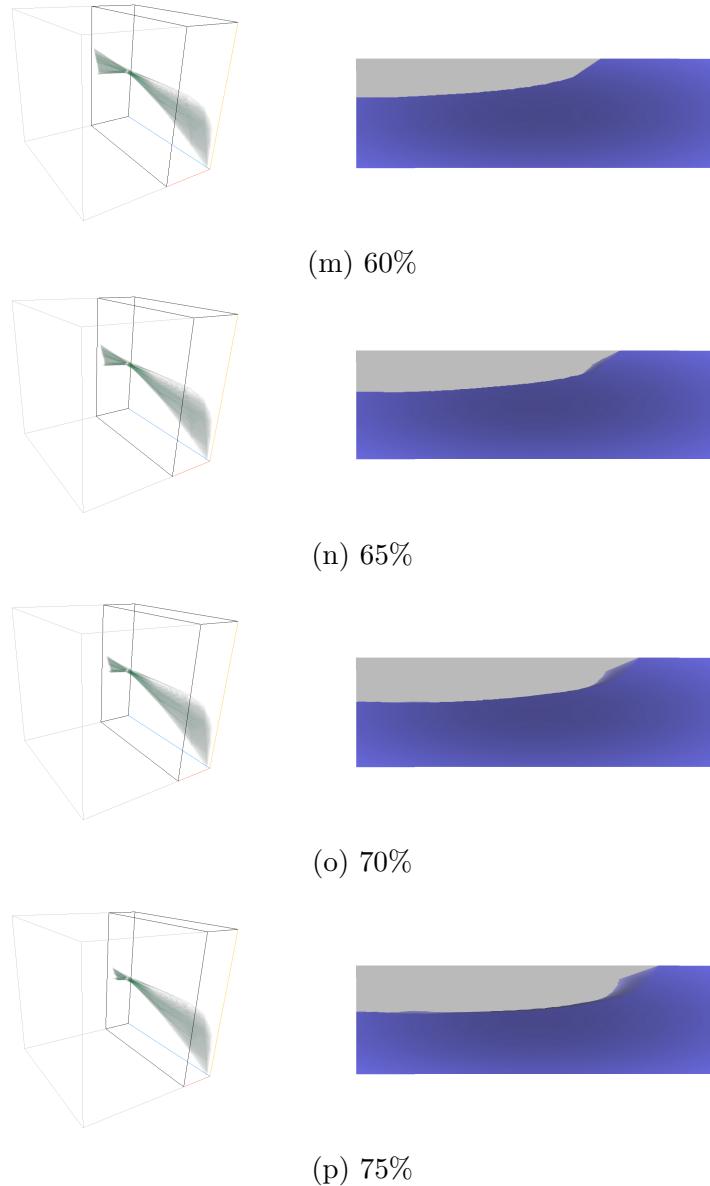


Abbildung 39: Darstellungen des Biegebalken Datensatzes. Die obere Schranke des größten Eigenwerts wurde jeweils um den angegebenen Prozentsatz der gesamten Intervalllänge verringert.

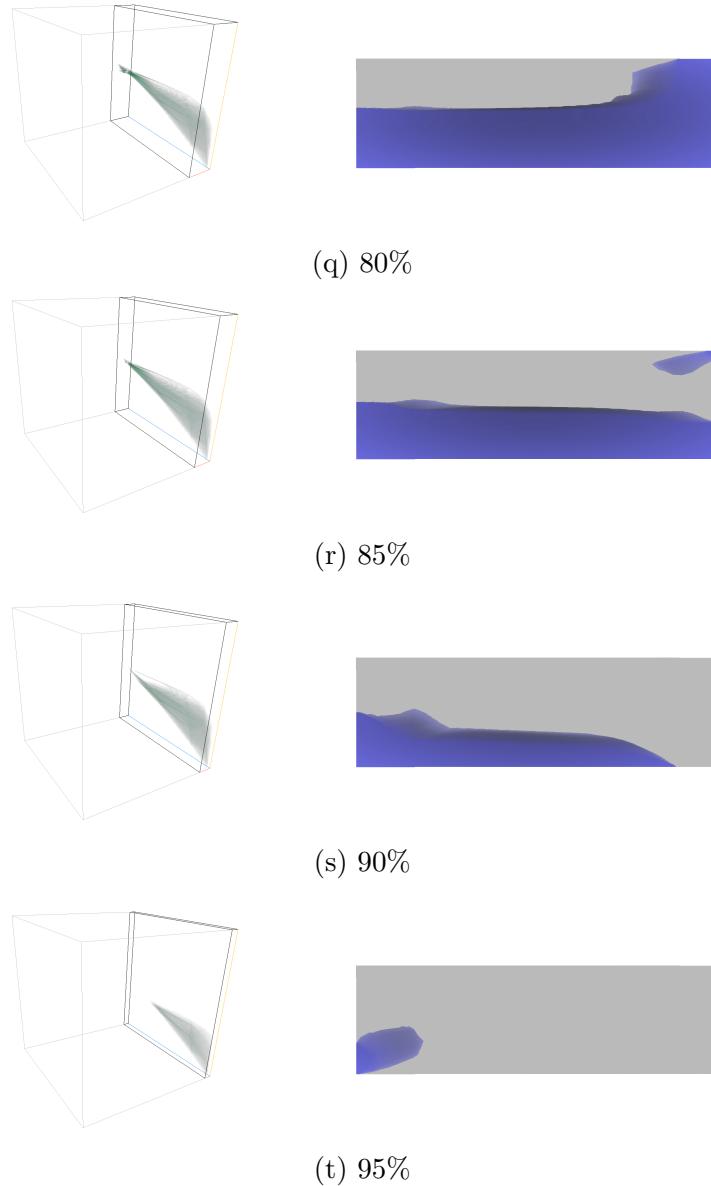


Abbildung 39: Darstellungen des Biegebalken Datensatzes. Die obere Schranke des größten Eigenwerts wurde jeweils um den angegebenen Prozentsatz der gesamten Intervalllänge verringert.