

Masterarbeit

Continuous Scatterplotting von Tensorfeldern in 3D

Niklas Teichmann

December 3, 2018

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 die Interpretation durch Nutzer zu erleichtern. Innerhalb dieser Arbeit wurde eine Erweiterung für die Visualisierungssoftware ‘FAnToM’ entwickelt, die Techniken aus dem Direct Volume Rendering und dem Continuous Scatterplotting verwendet, um Invarianten von symmetrischen Tensoren 2. Grades und ihre Verteilung innerhalb eines Datensatzes darzustellen. Dabei wurde besonderer Wert 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

Contents

1 Einleitung	1
2 Verwandte Arbeiten	2
3 Grundlagen	3
3.1 Mathematische Grundlagen	3
3.1.1 Urbild und Bild einer Funktion	4
3.1.2 Einschränkung einer Funktion	4
3.1.3 Dirac Delta	4
3.1.4 Volumen einer Teilmenge von \mathbb{R}^n	5
3.1.5 Jacobi Matrix	5
3.1.6 Lineare Abbildungen	5
3.1.7 Vektorraum	6
3.1.8 Dualraum	6
3.1.9 Multilineare Funktionen	7
3.1.10 Tensor	7
3.1.11 Rang einer Matrix	8
3.1.12 Spur einer Matrix	8
3.1.13 Norm einer Matrix	8
3.1.14 Deviator einer Matrix	9
3.1.15 Eigenwerte und Eigenvektoren einer Matrix	9
3.1.16 Fraktionale Anisotropie einer Matrix	9
3.1.17 Modus einer Matrix	10
3.1.18 Gradient	11
3.1.19 Orthogonalität von Matrizen	11
3.1.20 Matrixinvarianten	11
3.1.21 Invariantensätze	12
3.2 Mechanische Grundlagen	15
3.2.1 Physikalisches Feld	15
3.2.2 Mechanische Spannung	16
3.2.3 Mechanische Verformung	16
4 Verwendete Verfahren und Technologien	17
4.1 FAnToM	17
4.2 OpenGL	17
4.3 Qt	19
4.4 Volumetrische Daten	19
4.5 Volume Visualization	20
4.5.1 Isoflächen	20
4.5.2 Slicing	22
4.5.3 Direct Volume Rendering	24
4.5.4 Raycasting	26

4.6	Continuous Scatterplotting	27
4.6.1	Scatterplotting	27
4.6.2	Erklärung des Continuous Scatterplottings	28
5	Umsetzung	33
5.1	Datenvorbereitung	34
5.1.1	Die Vorbereitungssession	34
5.1.2	Die Visualisierungssession	35
5.2	Voxelisierung	36
5.3	Umsetzung des Continuous Scatterplottings	39
5.4	Berechnung der maximalen Dichte	40
5.5	Umsetzung des Raycastings	41
5.6	Die Interaktionsflächen	44
5.7	Weitere Interaktionen	45
5.8	Optionen des Algorithmus	45
5.8.1	Gemeinsame Optionen	45
5.8.2	Optionen des Objektrenderings	46
5.9	Optionen des Invariantenraumrenderings	47
5.10	Das Intervallfenster	47
6	Ergebnisse	48
6.1	Single Point Load	48
6.2	Metallscheibe	49
6.2.1	Spannung	51
6.2.2	Verformung	54
6.3	Bremshebel	56
6.3.1	K-Invarianten	57
6.3.2	R-Invarianten	57
6.4	Fazit	60
7	Ausblick	60
References		I

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. Skalare, Vektoren und Matrizen können ebenfalls als Tensoren aufgefasst werden. Die Auswertung der Tensoren ist dann mittels des inneren Produkts möglich. Im Abschnitt [wird](#) dies näher erklärt.

Beispiele für Tensorfelder findet man in der Diffusions-Tensor-Bildgebung[[4](#)], welche die Diffusion von Wasser in Gewebe wie z.B. dem Hirn untersucht, oder bei Verformungs-[[13](#), S. 122] und Spannungstensoren[[13](#), S. 154] in der Mechanik. 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. Dabei stellen sich eine Reihe von Herausforderungen, von denen einige im Folgenden genannt werden[[14](#)][[12](#)]:

Menge an Daten pro Tensor Ein einzelner Tensor kann, abhängig von Grad und Dimension, beliebig viele Datenwerte umfassen. Selbst ein Tensor 2. Grades und 3. Dimension, der durch eine 3×3 Matrix dargestellt wird, besteht bereits aus 9 Werten, für die eine visuelle Codierung gefunden werden muss.

Menge an Daten pro Datensatz Häufig enthalten Datensätze Tausende oder Millionen von Tensoren, was Anspüche 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 solcher kleiner Merkmale sichergestellt werden.

Fehlende Intuition Tensoren beschreiben im Allgemeinen Abbildungen zwischen Skalaren, Vektoren und höheren Tensoren. Während bei niedrigen Rängen (Skalare/Vektoren) noch intuitive Interpretationen existieren (Zahlenwert/Punkt im mehrdimensionalen Raum), fällt es Menschen erheblich schwerer, Matrizen oder Tensoren höheren Grades zu interpretieren. Die Repräsentation eines Tensors muss daher sehr gut durchdacht sein und relevante Eigenschaften interpretierbar darstellen.

Domänenspezifische Informationen Abhängig von der jeweiligen Domäne können unterschiedliche Informationen über die vorliegenden Tensoren von Interesse sein. Zum Beispiel kann isotropen oder degenerierten Punkten (Punkte, in denen die Eigenvektoren nicht eindeutig definiert sind) in manchen Anwendungsfällen besondere Bedeutung zugemessen werden, während sie in anderen Kontexten nur Punkte hoher Symmetrie

ohne besondere Bedeutung sind [14, S. 4]. Eine Anwendung zu entwickeln, die über Domänen hinweg verwendbar ist, ist daher eine Herausforderung.

Im Zuge der vorliegenden Arbeit wurde ein neues Verfahren zur Visualisierung von symmetrischen Tensoren 2. Grades im dreidimensionalen Raum entwickelt, das versucht, die genannten Probleme zu lösen. Es wurde als Erweiterung der Visualisierungssoftware ‘FAnToM’ implementiert. Im Speziellen wurden Invariantenfelder aus den Matrixdarstellungen der Tensoren berechnet und mithilfe von Techniken aus dem Continuous Scatterplotting und dem Direct Volume Rendering dargestellt. Durch Mausinteraktionen ist es möglich, Bereiche von Invarianten auszuwählen und im ursprünglichen Feld darzustellen. Das Ziel der Anwendung ist es, kontinuumsmechanische Untersuchungen von Materialien und Werkstücken zu erleichtern.

Der Rest dieser Arbeit ist wie folgt gegliedert: Zunächst werden im Kapitel [2 Verwandte Arbeiten](#) bekannte Verfahren zur Tensorfeldvisualisierung mit Vor- und Nachteilen 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. Kapitel [5 Umsetzung](#) beschreibt die konkrete Umsetzung der FAnToM-Erweiterung mit allen implementierten Funktionen. Nachfolgend wird in Kapitel [6 Ergebnisse](#) die entwickelte Erweiterung exemplarisch auf einige Datensätze angewendet, und die Ergebnisse diskutiert. Zum Abschluss werden in Kapitel [7 Ausblick](#) im Verlauf dieser Arbeit neu aufgetretene Problemstellungen sowie weitere Verbesserungsmöglichkeiten 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 Vektorfelds besteht darin, die einzelnen Komponenten eines Tensor 2. Grades als Skalarfelder aufzufassen und als Grauwertbild zu zeichnen. Dabei ist der Grauwert an einem Datenpunkt abhängig vom Verhältnis des Wertes der Komponente des Vektors zu dem höchsten Wert dieser Komponente im Datensatz. Die früheste von mir gefundene Erwähnung dieses Verfahrens stammt aus einem Paper von Kindlmann und Weinstein aus dem Jahre 1999 [18], 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 Matrizen dargestellt werden können und einen großen Teil der Daten aus Mechanik und Medizin ausmachen. Eine Glyphe ist hierbei ein kleines

Bild eines grafischen Primitivs, z.B. eines Ellipsoiden, Kuboiden oder Superquadrics[17], das einen Tensor darstellt. Die Form der Primitive 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[8] bilden ein Analogon zu den Stromlinien bei Vektorfeldern. Als Eingabedaten sind nur Felder von reellen, symmetrischen, dreidimensionalen Tensoren 2. 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 Schläuche durch das Vektorfeld. Die Mittellinien dieser Schläuche entsprechen dabei Stromlinien, deren Richtung vom Eigenvektor mit dem höchsten Eigenwert abhängt. Der Durchschnitt durch den Schlauch orthogonal zur Mitellinie ist stets eine Ellipse, deren Halbachsen den zwei kleineren Eigenwerten entsprechen. Da so aber Informationen über den Wert des größten Eigenwertes verloren gehen, werden einzelne Stücke des Schlauchs abhängig von diesem Wert eingefärbt.

Weiterhin muss die Diplomarbeit von Clemens Fritzsch, ‘Visuelle Analyse kontinuumsmechanischer Simulationen durch kontinuierliche Streudiagramme’[12], erwähnt werden, auf der die vorliegende Arbeit direkt aufbaut. Fritzsch verwendet Methoden des Continuous Scatterplotting, um Invarianten von zweidimensionalen, symmetrischen Tensoren 2. Grades darzustellen. Dabei beschränkt er sich jedoch auf zwei der drei Invarianten in jedem Invariantensatz, um einen zweidimensionalen Scatterplot zu erzeugen. Die vorliegende Arbeit erweitert diesen Ansatz auf vollständige Invariantensätze indem eine dreidimensionale Darstellung erzeugt wird.

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 dem Buch ‘Introduction to vectors and tensors’ 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 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

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

und auf allen $a \in A, a \notin A'$ nicht definiert.

3.1.3 Dirac Delta

Das Dirac Delta (auch Delta Distribution genannt) ist eine Funktion, die folgende Eigenschaften erfüllt:

$$\delta(x) = \begin{cases} +\infty, & \text{wenn } x = 0 \\ 0, & \text{sonst} \end{cases} \quad (4)$$

$$\int_{-\infty}^{+\infty} \delta(x) dx = 1 \quad (5)$$

Da diese beiden Eigenschaften für die vorliegende Arbeit ausreichen, wird auf eine genaue Definition von δ verzichtet. Diese kann jedoch in [21, S. 100 ff.] nachgelesen werden.

3.1.4 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[20]. Intuitiv entspricht es 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.

3.1.5 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_m der einzelnen Komponenten

$$J_f(a) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(a) & \dots & \frac{\partial f_1}{\partial x_n}(a) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(a) & \dots & \frac{\partial f_m}{\partial x_n}(a) \end{pmatrix} \quad (6)$$

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 . Die für diese Arbeit wichtigste Rolle spielt der absolute Wert $|det(J_f)|$ an einem Punkt p , die die Expansion bzw. das Schrumpfen der Funktion in der Nähe von p beschreibt. 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 $\|p_1, p_2\|_n$

$$\|f(p_1), f(p_2)\|_n = det(J_f) \cdot \|p_1, p_2\|_n \quad (7)$$

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

3.1.6 Lineare Abbildungen

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

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

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

3.1.7 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:

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

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

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

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

und für die Skalarmultiplikation folgende Eigenschaften:

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

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

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

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

3.1.8 Dualraum

Sei V ein n -dimensionaler Vektorraum und K sein zugrundeliegender Körper. 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 φ 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].

3.1.9 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 φ linear in jeder Variable ist [5, S. 204, 218].

3.1.10 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 2. Grades verwendet, die in kartesischen, dreidimensionalen Koordinatensystemen erzeugt wurden. 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 2. 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.

3.1.11 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.

3.1.12 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

$$\text{tr}(A) = \sum_{i=1}^n a_{ii} \quad (22)$$

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 3.1.20).

3.1.13 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:

$$f(kv) = |k|f(v) \quad (\text{Absolute Homogenität}) \quad (23)$$

$$f(u + v) \leq f(u) + f(v) \quad (\text{Erfüllung der Dreiecksungleichung}) \quad (24)$$

$$f(v) = 0 \iff v = 0 \text{ ist der Nullvektor} \quad (\text{Definitheit}) \quad (25)$$

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

$$\text{norm}(A) = \sqrt{\text{tr}(AA^T)} \quad (26)$$

Die euklidische Norm wird auch als 'Frobeniusnorm' bezeichnet.

3.1.14 Deviator einer Matrix

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

$$\tilde{A} = A - \bar{A} \quad (27)$$

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

$$\bar{A} = \frac{1}{3} \text{tr}(A) I \quad (28)$$

wobei I die Matrixdarstellung des Einheitstensors ist, also

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (29)$$

3.1.15 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

$$M \cdot v_i = \lambda_i v_i \quad (30)$$

Intuitiv bedeutet das, dass sich durch Multiplikation mit M ihre Richtung nicht verändert. Die zugehörigen λ_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.

3.1.16 Fraktionale Anisotropie einer Matrix

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

$$FA(M) = \sqrt{\frac{3((\lambda_1 - \bar{\lambda})^2 + (\lambda_2 - \bar{\lambda})^2 + (\lambda_3 - \bar{\lambda})^2)}{2(\lambda_1^2 + \lambda_2^2 + \lambda_3^2)}} \quad (31)$$

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

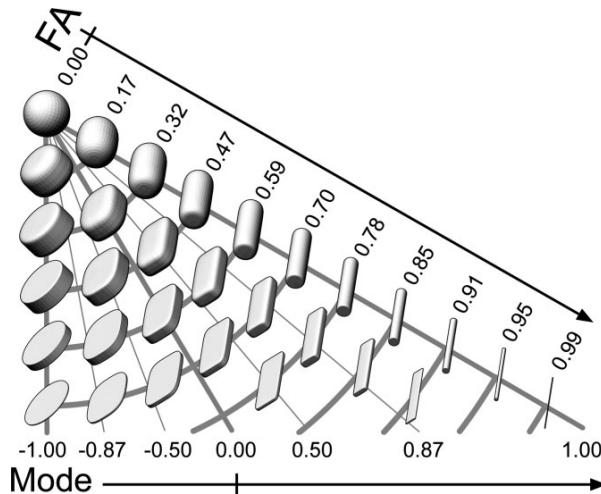


Figure 1: Darstellung der fraktionalen Anisotropie und des Modus von Matrizen in Form von Superquadrics[17]. Die fraktionale Anisotropie nimmt mit größerer werdender Entfernung zum obersten linken Superquadric zu. Der Modus des Deviators wird abhängig vom Winkel dargestellt, wobei er links -1 beträgt und rechts 1. Entnommen aus [10, S. 140].

größer als die anderen beiden. Dies kann beispielsweise in Aufnahmen der Diffusions-MRT ein Hinweis auf die Position und Richtung von Nervenbahnen sein. Niedrige FA dagegen drückt aus, dass die Eigenwerte etwa die gleichen Werte annehmen. In der Kontinuumsmechanik ist dies beispielsweise in Bereichen isotroper Verformung (also gleichmäßiger Ausdehnung / gleichmäßigem Schrumpfen in alle Richtungen) der Fall.

3.1.17 Modus einer Matrix

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

$$mode(A) = 3\sqrt{6} \det(A \setminus norm(A)) \quad (32)$$

wobei $\det()$ die Determinante ist. Im Folgenden wird meistens der Modus des Deviators von A verwendet.

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

- $mode(A) = 1$: ein hoher, zwei gleiche niedrige Eigenwerte; lineare Anisotropie
- $mode(A) = 0$: ein hoher, ein niedriger und ein mittlerer Eigenwert; Orthotropie
- $mode(A) = -1$: 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.

3.1.18 Gradient

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

$$\text{grad}(f) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \quad (33)$$

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

$$\text{grad}(g) = \begin{pmatrix} \frac{\partial f}{\partial a_{11}} & \cdots & \frac{\partial f}{\partial a_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial a_{m1}} & \cdots & \frac{\partial f}{\partial a_{mn}} \end{pmatrix} \quad (34)$$

wobei a_{ij} die Komponenten der Matrizen sind.

3.1.19 Orthogonalität von Matrizen

Zwei Matrizen U, V werden als orthogonal zueinander bezeichnet, wenn [10]

$$\text{tr}(U, V^T) = 0 \quad (35)$$

3.1.20 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 sind beispielsweise Größen, die sich unabhängig von der Wahl der Basis der Matrix nicht verändern[10]. 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 2. Grades und ausschließlich orthogonale Transformationen beschäftigt, beschränkt sich auch die Betrachtung der Invarianten auf diese. 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

$$\Psi : \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \rightarrow \mathbb{R} \quad (36)$$

darstellbar, wobei $\lambda_1, \lambda_2, \lambda_3$ die Eigenwerte von T sind. Dies stimmt mit der Betrachtungsweise von Zobel und Scheuermann [28] überein.

3.1.21 Invariantensätze

Mengen von Invarianten werden als Invariantensätze bezeichnet. Matrixinvarianten Ψ_1 und Ψ_2 werden als orthogonal zueinander bezeichnet, wenn ihre Gradienten für jede mögliche Eingabematrix orthogonal sind. Die genauen Definitionen und Berechnungen dazu sind in ‘Orthogonal tensor invariants and the analysis of diffusion tensor magnetic resonance images’ von Ennis, Kindlmann et al. [10] nachzulesen. Da Gradienten von skalarwertigen Funktionen auf Matrizen wiederum Matrizen sind [10, S. 137], genügt zu zeigen, dass diese orthogonal zueinander sind. Invariantensätze, deren Elemente paarweise orthogonal sind, werden als orthogonale Invariantensätze bezeichnet.

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

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 χ einer 3×3 Matrix A hat die Form

$$\begin{aligned} \chi_A(\lambda) &= \det(\lambda I - A) \\ \chi_A(\lambda) &= -\lambda^3 + I_1 \lambda^2 - I_2 \lambda + I_3 \end{aligned} \quad (37)$$

wobei λ 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.

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)

Dabei ist jedoch zu beachten, dass

$$\begin{aligned}
 \text{tr}(\tilde{A}) &= \text{tr}(A - \frac{1}{3}\text{tr}(A)I) \\
 &= a_{11} - \frac{1}{3}\text{tr}(A) + a_{22} - \frac{1}{3}\text{tr}(A) + a_{33} - \frac{1}{3}\text{tr}(A) \\
 &= a_{11} + a_{22} + a_{33} - \text{tr}(A) \\
 &= \text{tr}(A) - \text{tr}(A) \\
 &= 0
 \end{aligned} \tag{38}$$

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

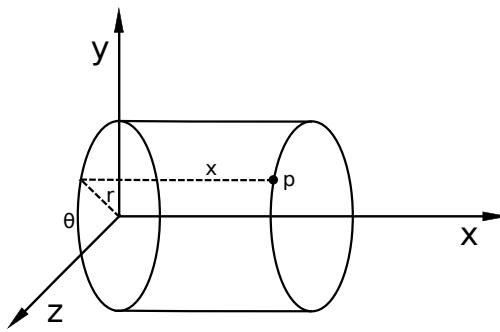


Figure 2: Darstellung eines zylindrischen Koordinatensystems. x, y und z entsprechen dabei den kartesischen Achsen. Die Koordinaten des Punktes p sind angegeben als x, r, θ . x entspricht der kartesischen Koordinate, r ist die Entfernung zwischen der Projektion von p auf die von y und z Achse aufgespannte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

Der K-Invariantensatz Der K-Invariantensatz ist orthogonal und besteht aus den Invarianten K_1, K_2 und K_3 . Diese sind für eine Matrix A definiert als

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

Da $K_1 \in [-\infty, \infty]$, $K_2 \in [0, \infty]$, $K_3 \in [-1, 1]$ 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 und K_3 den Winkel zu einer festgelegten, zur zentralen Achse orthogonalen, zweiten Achse. Sowohl ein zylindrisches als auch ein kartesisches Koordinatensystem sind in Abbildung 2 dargestellt. K_1 entspricht dabei der zylindrischen x -Koordinate, K_2 dem Radius r und K_3 dem Winkel θ .

Ein Vorteil des K-Invariantensatzes ist die relativ leichte Interpretierbarkeit in der Kontinuumsmechanik. K_1 kann als Maß der absoluten ‘Größe’ der Matrix, K_2 als Maß des anisotropen Anteils angesehen werden [19]. 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 Der orthogonale R-Invariantensatz verwendet die Invarianten R_1, R_2 und R_3 . Für eine Matrix A sind sie definiert als

- $R_1(A) = \text{norm}(A)$ (Norm von A)

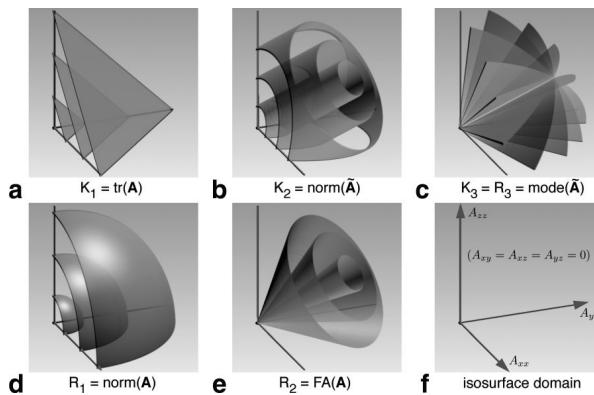


Figure 3: Dargestellt sind Isoflächen der K und R Invarianten von diagonalisierten 3×3 Matrizen (alle Werte ausserhalb der Hauptdiagonale sind 0). Die Koordinatenachsen entsprechen den drei Eigenwerten der Matrizen. Entnommen aus [10, S. 139]

- $R_2(A) = \sqrt{\frac{3}{2} \frac{\text{norm}(\tilde{A})}{\text{norm}(A)}}$ (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 kann auch R in einem zylindrischen Koordinatensystem dargestellt werden.

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

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

3.2 Mechanische Grundlagen

3.2.1 Physikalisches Feld

Ein physikalisches Feld ist eine physikalische Größe, die an verschiedenen Positionen im Raum unterschiedliche Werte annimmt [11, 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, z.B. weil 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.

3.2.2 Mechanische Spannung

In der Mechanik beschreibt Spannung (engl. stress) die Kraft, 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 σ bilden:

$$\sigma = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} \quad (39)$$

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 σ entlang der Hauptdiagonale, die Kraft in Richtung der Normalen der jeweiligen Ebene angeben, und den τ , die Scherspannungen angeben, die parallel zu den Schnittebenen verlaufen.

3.2.3 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, engl. strain tensor) lässt sich die Verformung an einem Punkt durch Kraftanwendung angeben:

$$= \begin{bmatrix} \epsilon_x & \frac{1}{2}\gamma_{xy} & \frac{1}{2}\gamma_{xz} \\ \frac{1}{2}\gamma_{xy} & \epsilon_y & \frac{1}{2}\gamma_{yz} \\ \frac{1}{2}\gamma_{xz} & \frac{1}{2}\gamma_{yz} & \epsilon_z \end{bmatrix} \quad (40)$$

Ähnlich wie beim Spannungstensor drücken die ϵ in der Hauptdiagonale Dehnungen oder Stauchungen des Objektes entlang der Hauptachsen aus, die γ 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][27] ('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 Algorithms entwickelt. Dabei unterscheidet man zwei Arten: den 'Data Algorithm' und den 'Visualization Algorithm'. Data Algorithms 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 Algorithms dagegen dienen dazu, Visualisierungen zu erzeugen. Interaktionen mit den Visualisierungen können entweder über die Maus oder über Optionen stattfinden.

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

Die aktuelle FAnToM-Session, also die momentan genutzten Algorithms mit ihren Optionen sowie jeweiligen Ein- und Ausgaben, können als 'Session' gespeichert und zu einem späteren Zeitpunkt wieder geladen werden.

Erweiterungen können auf eine Vielzahl von mächtigen Werkzeugen zugreifen, die von FAnToM zur Verfügung gestellt werden. Besonders wichtig für die vorliegende Arbeit waren dabei das effiziente und weit entwickelte Datenmodell, das viele Funktionen zum Verarbeiten von Feldern und Tensoren bietet, die bestehenden Schnittstellen für Qt und OpenGL sowie die große Bibliothek von wiederverwendbaren Algorithms.

4.2 OpenGL

OpenGL[15] 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 Programmierern, die durch

OpenGL unabhängig von der Hardware, auf der das Programm laufen soll, entwickeln können.

Die Verarbeitungsschritte von der Übergabe von Daten zu OpenGL bis zum fertig gerenderten Bild wird 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 Grafikprimitivs, beispielsweise 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 zum Beispiel üblich, um die Sicht auf die Szene von der Position der Kamera aus zu berechnen. Vertex Shader werden einmal pro Punkt ausgeführt. Die Ausgabe des Vertex Shaders sind die neuen Koordinaten der Punkte der Primitive.

Geometry Shader Die Renderpipeline übergibt dem Geometry Shader für jedes Primitiv 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 Primitive 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 Primitiv wird der Geometry Shader einmal ausgeführt.

Fragment Shader Fragment Shader bestimmen die Farben der Fragmente. Fragmente sind Stücke der projizierten Primitve, deren Größe von der Art der Rasterisierung abhängen. In der Regel wird pro Pixel mindestens ein Fragment berechnet. Wenn mehrere Fragmente pro Pixel berechnet, und deren Farbwerte 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 Primitivs. 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 Primitiven ermöglicht.

Eine weitere im Folgenden verwendete Funktion von OpenGL ist ‘Blending’, durch das Fragmente übereinander gezeichnet und ihre Farbwerte gewichtet kombiniert werden können, was z.B. für transparente Objekte wichtig ist. Zudem existiert die Möglichkeit, gerenderte Bilder mithilfe von ‘Framebuffern’ nicht anzuzeigen, sondern stattdessen in einer Textur zu speichern.

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 auch für andere rechenintensive, jedoch 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 Algorithms 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 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.

Bei sehr einfachen Feldern kann es ausreichen, die Stringrepräsentation einer mathematischen Funktion, durch die das Feld hinreichend 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 (Mittelpunkt, Eckpunkte, Seitenflächen, ...) die Werte zugewiesen werden, ist vom Anwendungsfall abhängig. Im einfachsten Fall geben diese Werte binär an, ob das Voxel ein Objekt Voxel 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.

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.

4.5 Volume Visualization

In der Computergrafik werden dreidimensionale Objekte üblicherweise durch Dreiecke dargestellt, durch die die Oberflächen der Objekte approximiert werden. Solches Vorgehen entfernt jedoch alle Informationen über Strukturen im Inneren der Objekte, wie beispielsweise Bereiche gleicher Materialien oder Dichte. Es werden besondere Verfahren benötigt, um volumetrische Daten darstellen zu können. Diese werden allgemein unter dem Begriff “Volume Visualization” zusammengefasst. Genauer definiert bezeichnet Volume Visualization Methoden der Extraktion von bedeutungsvollen Informationen aus volumetrischen Daten durch interaktive Grafiken und Bildgebung [16, S. 127]. Nachfolgend werden einige Volume Visualizations für skalare Felder mit Vor- und Nachteilen vorgestellt. Dabei beschränken sich die Erläuterungen auf Datensätze, die auf (meist quaderförmigen) Voxeln basieren, und der skalare Wert dem Volumen des Voxels zugeordnet ist. Für die meisten Algorithmen existieren jedoch auch Varianten, die mit anderen Formen volumetrischer Daten umgehen können.

4.5.1 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 Volume Visualization dar. Eine Möglichkeit diese Informationen zu vermitteln stellt die Berechnung und Darstellung von Isosurfaces dar. Die Isosurface 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 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 $f(r_1) > i$ [25]. Die Isosurface muss dabei nicht zusammenhängend sein. Auf der Isosurface selbst ist f gleich dem Isowert.

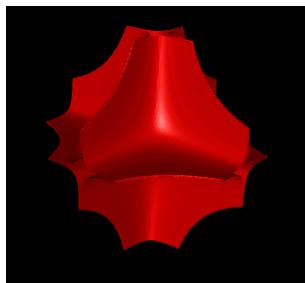


Figure 4: Darstellung einer Isofläche in einem automatisch generierten Datensatz. Erzeugt mithilfe von FAnToM [1].

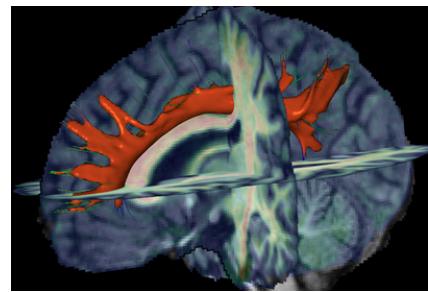


Figure 5: Darstellung von drei Slicings durch einen volumetrischen MRI Datensatz. In Orange ist zusätzlich eine Isofläche eingezeichnet. Entnommen aus [24].

In Abb. 4 ist ein Beispiel für eine Isofläche abgebildet. 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 einige Herausforderungen, besonders bezüglich der Korrektheit des Verlaufs der Isoflächen innerhalb eines Voxels und der benötigten Rechenzeit. In der Vergangenheit wurden einige Verfahren entwickelt, die sich in Hinblick auf Korrektheit und Geschwindigkeit stark unterscheiden. Der wohl einflussreichste davon ist "Marching Cubes". 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 Isofläche nicht neu berechnet werden, solang sich der Isowert nicht ändert. Die resultierenden Dreiecke können von Standard 3D Grafikbibliotheken und -hardware sehr effizient 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 des dreidimensionalen Form zu vermitteln.

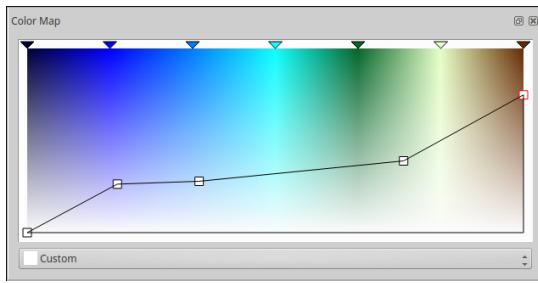


Figure 6: Eine Implementierung einer interaktiven Transferfunktion in FAnToM.

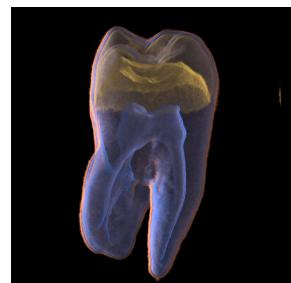


Figure 7: Ein Direct Volume Rendering eines Zahns. Entnommen aus [9, S. 6]

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.

4.5.2 Slicing

Ein einfacher Ansatz um einen volumetrischen Datensatz zu visualisieren ist das sogenannte “Slicing”[22]. Dabei wird der Schnitt zwischen dem Datensatz und einer Ebene berechnet, und dieser Schnitt durch eine Menge von verbundenen, texturierten Dreiecken, einem sogenannten “Dreieckszug” (engl. “Triangle Strip”) 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, bezeichnet als Samplepunkte, berechnet. Um aus den Samplepunkten die Färbungen der Dreiecke zu berechnen, gibt es zwei Verfahren, bezeichnet als Präklassifizierung und Postklassifizierung.

Bei der Präklassifizierung wird zunächst die Farbe an den Samplepunkten 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. 6 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 Farbzuzuordnung 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 Samplepunkte interpoliert. Ein Nachteil der Päklassifizierung 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 $\text{pin}\mathbb{R}^3$ einen skalaren Wert $s \in \mathbb{R}$ zuordnet, also $f(p) = s$. Sei 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

$$f(x) = \begin{cases} (1.0, 0.0, 0.0, 1.0) \text{ (Rot), wenn } x \leq 0 \\ (0.0, 1.0, 0.0, 1.0) \text{ (Grün), sonst} \end{cases} \quad (41)$$

Wenn nun drei Punkte p_0, p_1, p_2 existieren, von denen p_0 und p_2 Samplepunkte sind und p_1 genau in der Mitte zwischen p_0 und p_2 liegt, so wird, abhängig von der Wahl der Interpolation, p_1 ein Wert zugeordnet, der zwischen dem von p_0 und dem von p_2 liegt. Bei linearer, komponentenweiser Interpolation beispielsweise entsteht als Farbe von p_1 ein dunkles Gelb. Die Interpretation dieser Farbe durch den Benutzer ist schwierig, da die Werte nicht an der Transferfunktion ablesen kann, sondern zunächst herausfinden muss, welche Farben gemischt wurden. 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, also zunächst die Datenwerte interpoliert und danach die Transferfunktion auf die interpolierten Werte angewendet wird. Dadurch werden Farbverläufe so wie in der Transferfunktion dargestellt. Ein neuer Nachteil entsteht jedoch: Durch die Interpolation können neue Datenwerte entstehen, die so im Datensatz nicht vorkommen. In dem meisten Fällen wird dies jedoch als das geringere Übel angesehen, weshalb Postklassifizierung fast immer bevorzugt wird.

In Abbildung 5 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 sogar ablesen zu können. Der Rechenaufwand ist sogar noch geringer als bei der Berechnung der Isoflächen und macht es möglich, die Slicings ohne spürbare Verzögerung zu repositionieren. Slicings eignen sich auch, um andere Volume Visualizations zu ergänzen (wie z.B. in Abb. 5).

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.

4.5.3 Direct Volume Rendering

Die bisher vorgestellten Volume Visualization 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, fast unmöglich sein.

Der Begriff “Direct Volume Rendering” bezeichnet eine Reihe von Verfahren der Volume Visualization, die den gesamten Datensatz gleichzeitig darstellen können, wenngleich die Darstellung auch auf Teile beschränkt werden kann. 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.

Dazu wird der Datensatz zunächst als Wolke von Partikeln aufgefasst, wobei die Partikel Licht emmitieren und absorbieren [16, s. 134f]. Wie viel Licht Partikel in einem Voxel absorbieren und emittieren sowie die Farbe des emittierten Lichts wird über eine Transferfunktion bestimmt. Diese ähnelt der, die auch beim Slicing verwendet wird, ordnet einem skalaren Wert jedoch neben einer Farbe auch noch eine Opazität zu (siehe dazu Abb. 6 und die Beschreibung im Abschnitt 4.5.2).

advantages: insensitivity to scene complexity (after preprocessing), object complexity, no texturing after preprocessing, viewpoint independence (viewpoint independent attributes [color, texture color, normal vector, antialiasing, visibility of light sources, viewpoint independent parts of illumination]) - unlike surface graphics, datasets can come directly from sampled datasets - inner structures

weaknesses: memory size, processing power (special architecture), discrete form -> artifacts, limits to accuracy of volume measurements, rotation by not 90 degree, no information about the geometry of the original object (surface and normal computation)

Texture Slicing

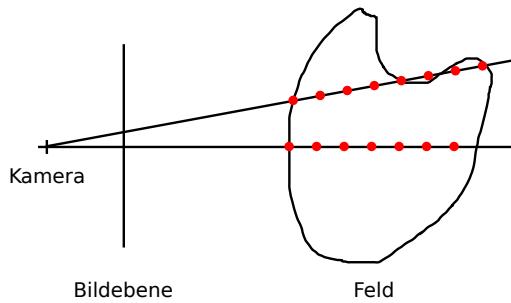


Figure 8: Eine vereinfachte Darstellung eines optimierten Raycastings. Die Bildebene entspricht der Ebene, die durch die imaginäre Position der Pixel gebildet wird. Die roten Punkte entlang der Strahlen entsprechen den Samplepoints.

Splatting

Raycasting

Shear Warp

Cell Projection Eine Alternative bietet das Direct Volume Rendering[9] (im Folgenden DVR). Als Eingabe wird ein Feld $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ verwendet, das Punkten im dreidimensionalen Raum skalare Werte zuordnet. Die Interpretation dieser Werte kann, abhängig vom Einsatzgebiet, unterschiedlich sein. Beispiele dafür sind die abgegebene Energie von Regionen in der Magnetresonanztomographie oder die Menge von absorbiert Röntgenstrahlung in der Computertomographie. Den skalaren Werten werden mittels einer ‘Transferfunktion’ Farbwerte und Transparenz zugeordnet.

Eine Implementierung einer interaktiven Transferfunktion in FAnToM ist in Abb. 6 zu sehen. Ziel des DVR ist es, mithilfe der Transferfunktion eine dreidimensionale Darstellung des Feldes zu erzeugen, in der Bereiche entsprechend der Transferfunktion Licht ausstrahlen. Abhängig von ihrer Transparenz absorbieren andere Bereiche dieses Licht oder lassen es durch. Dadurch ist es möglich, Strukturen im Inneren des Feldes sichtbar zu machen. Ein Beispiel für eine so erzeugtes Bild ist in Abb. 7 dargestellt.

Es existieren viele DVR Verfahren die sich in Laufzeit, Bildqualität und Anfälligkeit für Artefakte unterscheiden. In dieser Arbeit wurde Raycasting verwendet, was folgend erläutert wird.

4.5.4 Raycasting

Beim Raycasting wird für jedes Pixel ein Strahl durch den Mittelpunkt des Pixels und die Position der Kamera berechnet. Entlang dieses Strahls wird, an nach ihrer Entfernung von der Kamera geordneten Punkten s_1, \dots, s_n ('Samplepoints') der interpolierte Wert des Feldes $v(s_i), 1 \leq i \leq n$ berechnet. Der paarweise Abstand der Samplepunkte ist dabei konstant. Aus dem ermittelten Wert wird mithilfe der Transferfunktion die Farbe $f(v(s_i))$ und die Transparenz $t(v(s_i))$ für diesen Wert bestimmt.

Dabei sei die Transparenz ein Wert im Intervall $[0, 1]$ und Farbe ein Vektorraum über \mathbb{R} , wie z.B. im RGBA Farbmodell, für den die Multiplikation mit einem Skalar definiert ist. Die Farbe des Pixels p wird dann für die Samplepunkte entlang des entsprechenden Strahls berechnet als

$$p = \sum_{i=1}^n \left(f(s_i) \cdot \prod_{j=1}^i \frac{(1 - t(s_j))}{n} \right) \quad (42)$$

Das hat den Effekt, dass weiter von der Kamera entfernte Samplepunkte von näheren Punkten entsprechend ihrer Transparenz verdeckt werden. Insbesondere folgt daraus die Konsequenz, dass, sobald genügend Punkte verarbeitet wurden, so dass die 'verbleibende Transparenz' einen Grenzwert unterschreitet, sich die Farbe des Pixels nicht mehr stark verändert, und die Berechnung frühzeitig beendet werden kann. Die verbleibende Transparenz wird durch die Gesamtzahl der Samplepunkte n geteilt, damit eine höhere Anzahl an Samplepoints nicht zu einem schnelleren Verlust von Transparenz führt.

Eine weitere Optimierung besteht darin, den ersten Samplepunkt erst nach dem Eintreten des Strahls in das Feld zu setzen und die Berechnung zu beenden, sobald alle verbleibenden Samplepunkte ausserhalb des Feldes liegen. Eine vereinfachte Darstellung dieses optimierten Verfahrens ist in 8 dargestellt.

Die Qualität eines Raycastings ist stark abhängig von der Anzahl an Samplepoints pro Strahl und somit vom Abstand zwischen den Samplepoints. Ein geringerer Abstand führt generell zu weniger Artefakten und macht kleine Strukturen besser erkennbar, erhöht jedoch den Rechenaufwand. Indem der Nutzer die Anzahl von Samplepoints selbst auswählt, kann er seinen optimalen Tradeoff zwischen Rechenzeit und Bildqualität finden. In Abb. 9a und 9b sind zwei Darstellungen desselben Skalarfeldes zu sehen, die durch Raytracing mit einer unterschiedlichen Anzahl von Samplepoints erzeugt wurden. 9a wurde mit 20 Samplepoints erstellt, wodurch visuelle, scheibenartige Artefakte entstanden. Bei einer höheren Menge von 200 Samplepoints in 9b verschwinden diese Artefakte.

Eine weitere wichtige Einstellung ist, welche Zahl in Formel 42 für Variable n verwendet wird. Wenn n die Anzahl der Samplepunkte pro Strahl ist, kann dies, wenn die Blickrichtung der Kamera nicht senkrecht auf einer der Seitenflächen steht, dazu führen, dass Bereiche am Rand des Feldes transparenter dargestellt werden als in der Mitte. Das liegt

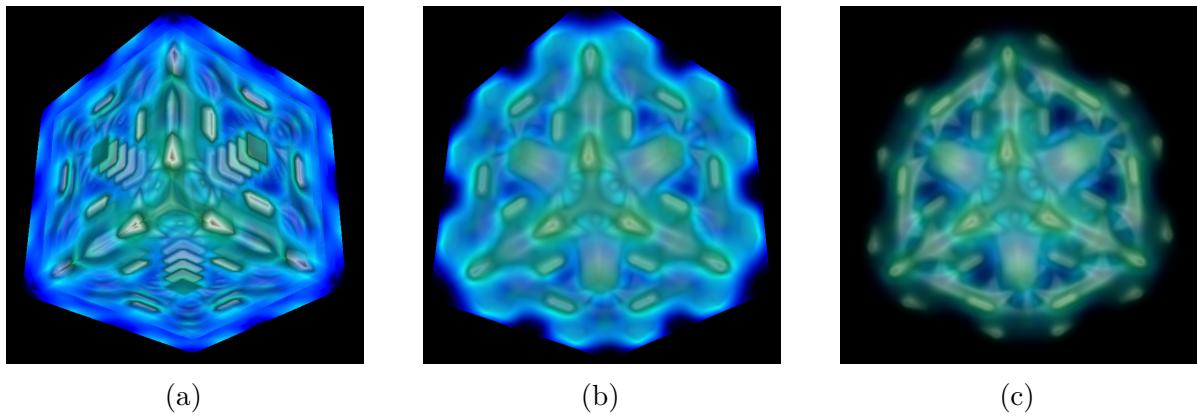


Figure 9: Drei Raycastings des Felds $f : \mathbb{R}^3 \rightarrow \mathbb{R}$, $f \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \cos(x \cdot y \cdot z)$.

daran, dass der Strahl am Rand weniger Samplepunkte produziert, die innerhalb des Feldes liegen. Ein Beispiel dafür ist in 9c mit 200 Samplepoints dargestellt.

Alternativ kann für n die Anzahl der Samplepunkte, die tatsächlich im Feld liegen, gewählt werden. Dies wurde in Abb. 9b getan. Wenn man nun 9b und 9c miteinander vergleicht, fällt auf, dass die Pixel in der Mitte der Darstellung etwa die gleichen Farben erhalten haben, 9c in Richtung Rand jedoch erheblich transparenter wird. Welche von beiden Varianten bevorzugt wird, hängt von der Art des Datensatzes ab sowie davon, wie wichtig es ist, korrekte Werte ablesen und vergleichen zu können.

4.6 Continuous Scatterplotting

4.6.1 Scatterplotting

Scatterplotting ist ein weit verbreitetes Verfahren, um Attribute von Objekten relativ 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-, zwei- oder dreidimensionales Koordinatensystem eingetragen, dessen Koordinatenachsen Attributwerten entsprechen. Beispiele für Scatterplottings sind in Abb. 10 abgebildet. Das Bild von τ 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: Cluster, Outlier und funktionale Zusammenhänge zwischen den verwendeten Attributen lassen sich gut identifizieren.

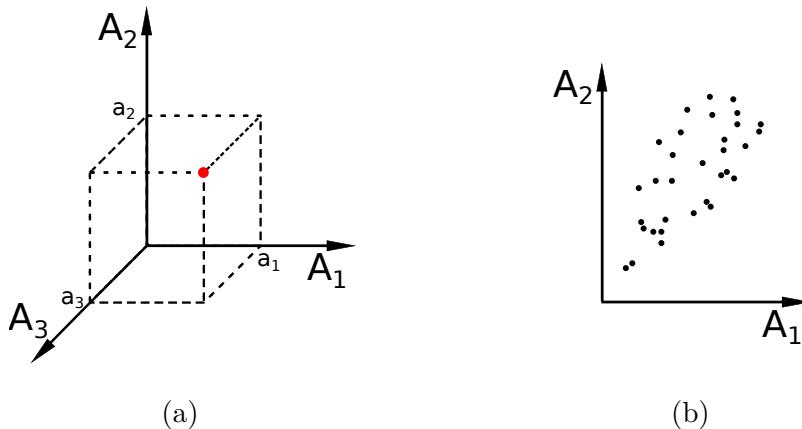


Figure 10: Zwei Beispiele für Scatterplottings. In (a) wird ein Objekt mit den Attributwerten a_1, a_2, a_3 als roter Punkt in einem dreidimensionalen Scatterplot dargestellt. Um die Attribute besser ablesen zu können, wurden gestrichelte Linien eingezeichnet. In (b) wurden mehrere Objekte mit unterschiedlichen Attributen im selben, zweidimensionalen Scatterplot als schwarze Punkte eingezeichnet.

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

Ein Beispiel für ein Urbild mit unendlich vielen Elementen ist \mathbb{R}^n bei physikalischen Feldern. Dort sind an jedem der unendlich vielen Punkte Werte definiert. Ein Scatterplot dieser unendlich vielen Elemente ist nun weder in endlicher Laufzeit möglich, noch wären einzelne Punkte in der entstehenden Darstellung erkennbar. Eine Alternative besteht darin, nur den Scatterplot der Eckpunkte der Zellen zu erzeugen, doch damit gehen Informationen über die Punkte innerhalb der Zellen verloren.

4.6.2 Erklärung des Continuous Scatterplottings

Das Continuous Scatterplotting stellt eine Erweiterung des Scatterplottings dar, sodass statt nur einer endlichen Anzahl von Objekten auch kontinuierliche Bereiche verarbeitet werden können. Die von Bachthaler und Weiskopf[3] vorgestellte Definition wurde von Fritzsche[12] um einige wichtige Punkte erweitert. Beim Continuous Scatterplotting wird ein Ansatz verwendet, der dem Verfahren bei der Herleitung der Kontinuumsmechanik aus Systemem mit diskreten Massenpunkten ähnelt [3, S. 1429]. Das Ziel des Continuous Scatterplottings ist es, eine Dichtefunktion $\sigma : \mathbb{R}^m \rightarrow \mathbb{R}$ zu finden, die den Punkten des Bildraums, abhängig von τ , Dichtewerte zuordnet. σ 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. 11 ist der Effekt, den τ auf die

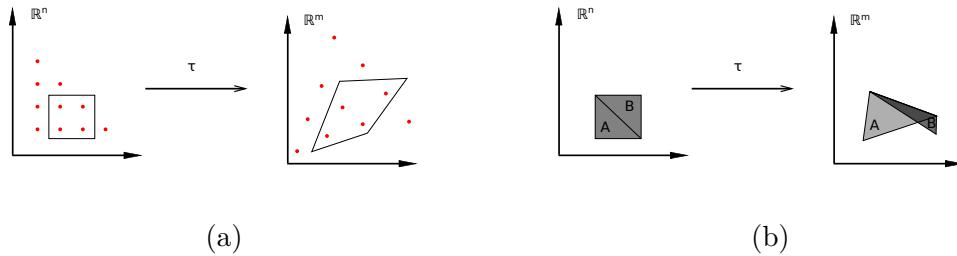


Figure 11: Darstellung der Dichteverteilung abhängig von τ . (a) Die roten Punkte deuten die Dichte in den beiden Scatterplots an, die zwei Vierecke sind ein Bereich des Urbilds, der durch τ verformt wird. Zu erkennen ist, dass eine Streckung eines Bereichs zur Verringerung der Dichte und eine Stauchung zur Erhöhung der Dichte führt. (b) Der Continuous Scatterplot zweier Dreiecke. Der Grauwert gibt die Dichte in den jeweiligen Bereichen an.

Dichtefunktion hat, dargestellt. Um σ 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 τ einen skalaren Wert, die Dichte an diesem Punkt, zuordnet. Der Einfachheit halber wird im Folgenden eine uniforme Dichte von $s(x) = 1$ angenommen. Aus der Dichte an den Punkten lässt sich die Gesamtdichte M , auch bezeichnet als Masse, einer Teilmenge $V \subset \mathbb{R}^n$ berechnen:

$$M = \int_V s(x) d^n x \quad (43)$$

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

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

$$\int_{\phi} \sigma(\xi) d^m \xi = \int_{\tau^{-1}(\phi)=V} s(x) d^n x \quad (44)$$

Da τ weder injektiv noch surjektiv ist, gilt der umgekehrte Fall nur, wenn zusätzlich $\forall x \in V : f^{-1}(f(x)) \subset V$ gilt [12, S. 20].

$$\int_V s(x) d^n x = \int_{\phi=\tau(V)} \sigma(\xi) d^m \xi \quad (45)$$

Fritzs [12, S. 20 f.] definiert zusätzlich ein σ_v , das die Dichtefunktion der Einschränkung von τ auf V beschreibt, also den ‘Beitrag’ von V zu σ .

$$\int_{\phi} \sigma_v(\xi) d^m \xi = \int_{\tau|_V^{-1}(\phi)} s(x) d^n x \quad (46)$$

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

$$\begin{aligned} \int_{\phi} \sigma(\xi) d^m \xi &= \int_{\bigcup_{V_i} \tau|_{V_i}^{-1}(\phi)} s(x) d^n x \\ &= \sum_{V_i} \int_{\tau|_{V_i}^{-1}(\phi)} s(x) d^n x \\ &= \sum_{V_i} \int_{\phi} \sigma_{V_i}(\xi) d^m \xi \\ &= \int_{\phi} \sum_{V_i} \sigma_{V_i}(\xi) d^m \xi \end{aligned} \quad (47)$$

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

$$\sigma(\xi) = \sum_{V_i} \sigma_{V_i}(\xi) \quad (48)$$

Eine intuitive Interpretation von Formel 48 ist, dass die gesamte Dichtefunktion von τ als Summe der Dichtefunktionen der Einschränkungen von τ auf die V_i gebildet werden kann.

Die Berechnung von σ 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 Eräuterung auf diesen Fall. Für $m = n$ unterscheiden Bachthaler und Weiskopf folgende Fälle für $V \subseteq \mathbb{R}^n$ [3, S. 1430]:

Fall 1: τ ist differenzierbar und ein Diffeomorphismus Ein Diffeomorphismus ist eine bijektive, stetig differenzierbare Abbildung, deren Umkehrabbildung ebenfalls stetig differenzierbar ist. Wenn τ ein Diffeomorphismus ist, lässt sich durch die Anwendung des Transformationssatzes von Integralen die Gleichung

$$\int_{\tau(V)} \sigma(\xi) d^{m=n} \xi = \int_V \sigma(\tau(x)) |\det(J_\tau)(x)| d^n x = \int_V s(x) d^n x \quad (49)$$

bilden, wobei J_τ die Jacobimatrix von τ 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

$$\sigma(\xi) = \frac{s(\tau^{-1}(\xi))}{|\det(J_\tau)(\tau^{-1}(\xi))|} \quad (50)$$

Wie schon in [3 Grundlagen](#) erwähnt, entspricht der Betrag der Determinanten der Jacobimatrix dem Betrag Expandierung oder Schrumpfung der Funktion in der Nähe des jeweiligen Punktes. Wenn die Funktion expandiert nimmt die Dichte ab, wenn sie schrumpft nimmt die Dichte zu.

Fall 2: τ ist differenzierbar und konstant über V , V ist keine Nullmenge Wenn $\det(J_\tau) = 0$, dann ist τ kein Diffeomorphismus und σ kann nicht so wie in Fall 1 definiert werden. Das ist z.B. der Fall, wenn τ Bereiche mit konstanten Werten enthält und führt dazu, dass ein Bereich aus \mathbb{R}^n auf einen einzelnen Punkt ξ abgebildet wird. Eine Möglichkeit die unendlich hohe Dichte an ξ darzustellen und dennoch die Gesamtmasse nicht zu verändern ist es, bei ξ ein Dirac-Delta einzufügen, das mit dem Volumen aller konstanten Bereiche $V_i \in V$, die nach ξ abgebildet werden, skaliert wird:

$$\begin{aligned} \sigma(\xi) &= \sum_{V_i \in V} \delta(\xi - \tau(V_i)) \cdot \int_{V_i} s(x) d^n x \\ &= \sum_{V_i \in V} \delta(\xi - \tau(V_i)) \cdot \text{Vol}(V_i) \quad (\text{da } s(x) = 1) \end{aligned} \quad (51)$$

Aus den Eigenschaften des Dirac Deltas ergibt sich, dass die Masseerhaltung erfüllt ist:

$$\begin{aligned} \int_{\phi} \sigma(\xi) d^n \xi &= \sum_{V_i} \int_{\phi} \delta(\xi - \tau(V_i)) \cdot \text{Vol}(V_i) d^n \xi \\ &= \sum_{\forall V_i : \tau(V_i) \in \phi} \text{Vol}(V_i) \\ &= \int_{\tau^{-1}(\phi)} 1 d^n x \end{aligned} \quad (52)$$

Da $s(x) = 1$ festgelegt ist, ist die Bedingung erfüllt. In [11b](#) 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 Formel [51](#) entsprechend angepasst werden.

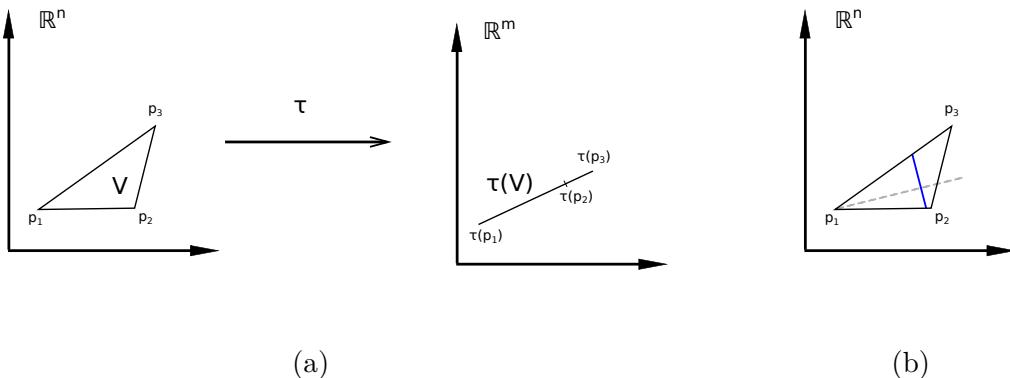


Figure 12: Eine Darstellung des fünften Falls. (a) Alle drei Punkte des Dreiecks V im linken Koordinatensystem werden auf eine Gerade abgebildet. Dadurch ist $\tau(V)$ eine Nullmenge. (b) Eine Darstellung des von Fritzs vorgeschlagenen Ansatzes. Die graue, gestrichelte Linie entspricht dem Gradienten, die Länge der blauen Linie ist proportional zur Dichte.

Fall 3: τ 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 τ einfach aus dem Scatterplotting entfernt werden, ohne die Masseerhaltung zu verletzen.

Fall 4: τ ist nicht differenzierbar Wenn im Datensatz Unstetigkeiten auftreten, zum Beispiel zwischen Zellen, ist τ an diesen nicht differenzierbar. Da die unstetigen Bereiche ebenfalls Nullmengen bilden, kann die gleiche Lösung wie in Fall 3 gewählt werden.

Fall 5: τ 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. Fritzschi konnte jedoch Beispiele für diesen Fall angeben[12, 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. 12a), 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 für ihren Rang gilt $0 < \text{rang}(M) < m$.

Fritzschi 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 $m = 2, n = 2$ 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 12b zu sehen. Die Dichte an den Punkten von $\tau(V)$ wird dann proportional zur Masse gewählt.

5 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 DVRs 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 Folgenden als ‘Objekt’ bezeichnet. Für das zweite DVR werden die 3 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 zusätzlich in ein zylindrisches Koordinatensystem überführt. Dieses Feld wird deshalb im Folgenden als ‘Feld im Invariantenraum’ bezeichnet. Die Dichten in den Zellen der neuen Zellstruktur werden durch Methoden des Continuous Scatterplottings berechnet.

In die Darstellung des zylindrischen Invarianten DVRs werden außerdem noch Interaktionsflächen in Form von Kreissektoren eingezeichnet. Durch Interaktion mit der Maus können die Flächen verschoben sowie ihr Radius und die Winkel der beiden Kreisradien zur XY Ebene verändert werden. Mithilfe der Interaktionsflächen kann der Nutzer Bereiche von Invarianten auswählen. Alles, was außerhalb des ausgewählten Bereichs liegt, wird im zylindrischen DVR ausgeblendet. Auch in der Darstellung des Objekts werden alle Bereiche ausgeblendet, deren Invarianten außerhalb des gewählten Bereichs liegen.

Zusätzlich existieren weitere Optionen, durch die der Nutzer die Darstellungen an seine Problemstellung anpassen kann.

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

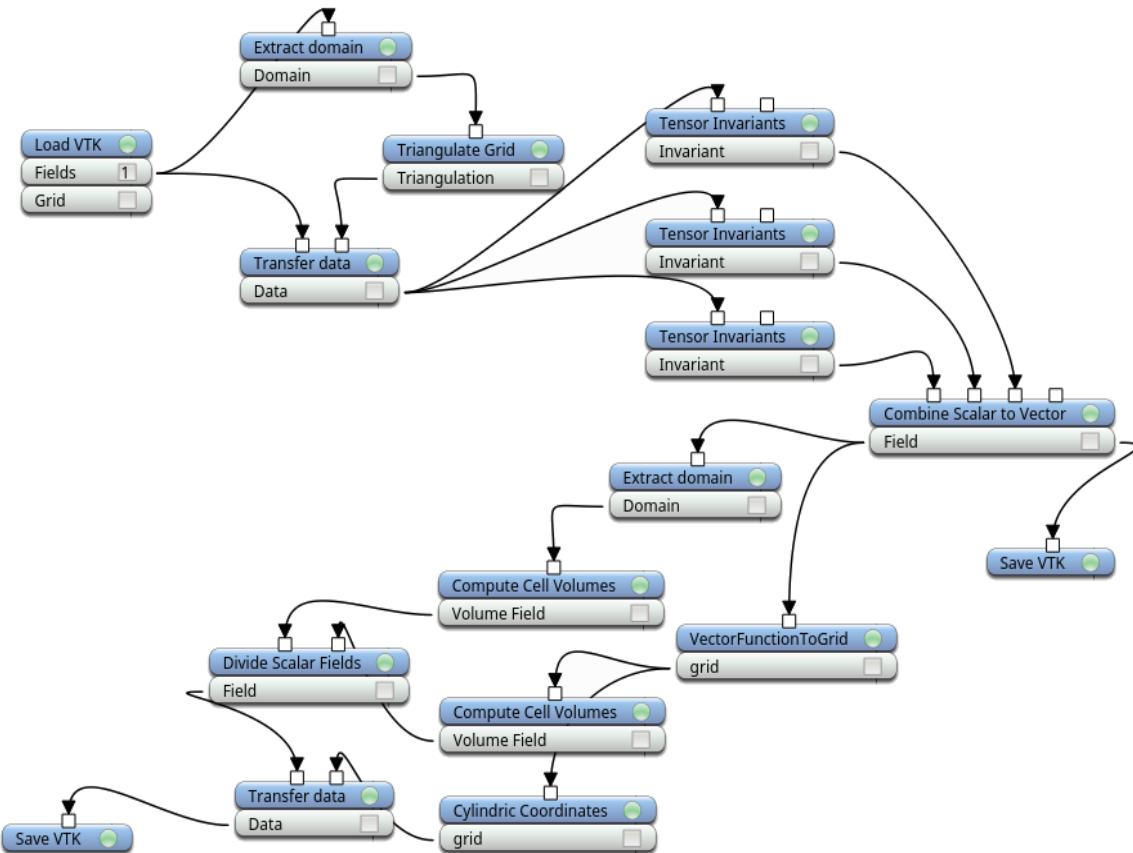


Figure 13: Der Flowgraph der FAnToM Session, die zur Vorbereitung der Daten verwendet wird.

5.1 Datenvorbereitung

5.1.1 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. Die Vorbereitung geschieht durch eine Reihe von FAnToM Algorithms, die im Folgenden beschrieben werden. Ein Screenshot der für die Datenvorbereitung verwendeten Session ist in Abb. 13 zu sehen. Die Algorithms der Session werden im weiteren Verlauf erklärt.

Die verwendeten Eingabedaten liegen im VTK Format[2] vor. Daher lädt der erste Algorithmus ‘Load/VTK’ die VTK Dateien. Der Algorithmus ‘Extract Domain’ extrahiert die Zellstruktur, auf der das Tensorfeld definiert ist. Falls die Zellen keine Tetraeder sind, werden sie vom Algorithmus ‘Triangulate Grid’ in Tetraeder zerlegt. Dabei wird die Anzahl und die Position der Eckpunkte der Zellen nicht verändert. Die Tensoren

an den Punkten des Objekts werden auf diese neue Zellstruktur durch den Algorithmus ‘Transfer Data’ kopiert.

Als Nächstes werden die Invarianten der Tensoren berechnet. Da die Datensätze aus 3×3 Matrizen bestehen, enthalten die zugehörigen Invariantensätze jeweils drei Invarianten. Dazu werden drei Instanzen des Algorithmus ‘Tensor Invariants’ verwendet, von denen jeder ein Skalarfeld auf derselben Tetraederzellstruktur berechnet, dessen Werte die Invarianten der Tensoren an den Eckpunkten sind. Für jeden Algorithmus kann eine eigene Invariante aus den genannten Invariantensätzen ausgewählt werden. Die drei Skalarfelder werden von ‘Combine Scalar to Vector’ zu einem Vektorfeld kombiniert, dessen Komponenten den Werten der Skalare entspricht. Eine Kopie des Vektorfeldes wird direkt als VTK Datei abgespeichert. Dies ist das erste von zwei Feldern, das als Eingabe der eigentlichen Visualisierung verwendet wird.

Durch den Algorithmus ‘Cylindric Coordinates’ werden die Werte des Vektorfeldes als Koordinaten von Punkten in einem zylindrischen Koordinatensystems aufgefasst. Ein Punkt ist durch die drei Komponenten des Vektors dann wie folgt beschrieben: Die erste Komponente wird als X-Koordinate interpretiert, die zweite Komponente als minimaler Abstand zwischen dem Punkt und der X-Achse und die dritte als Winkel zwischen dem kürzesten Liniensegment, das die X-Achse mit dem Punkt verbindet, und der XY-Ebene. Die letzte Komponente wird dabei auf das Intervall $[-1, \dots, 1]$ normiert, wobei -1 ein Liniensegment in negative Y Richtung beschreibt, 0 ein Liniensegment in positive Z Richtung und 1 ein Liniensegment in positive Y Richtung. Aus diesen neuen Punkten werden Tetraederzellen äquivalent zum Eingabefeld definiert. Die Ausgabe entspricht also der Zellstruktur der Eingabe, in der alle Punkte auf die neuen Koordinaten ‘verschoben’ wurden.

Im nächsten Schritt wird das Volumen der Zellen des Objekts und der neu erzeugten Zellstruktur berechnet (‘Compute Cell Volumes’). Es entstehen zwei Skalarfelder, die den Zellen ein Volumen zuordnen. Die Skalarfelder werden elementweise durcheinander geteilt (Original geteilt durch verschobenes Feld, ‘Divide Scalar Fields’). Das Ergebnis ist wiederum ein Skalarfeld, das σ_V der Zellen für das Continuous Scatterplotting approximiert. Die Herangehensweise wird in Abschnitt 5.3 genauer erklärt und begründet.

Zum Schluss wird das Quotientenfeld auf die verschobene Zellstruktur zellenweise übertragen und das Ergebnis abgespeichert, um als zweite Eingabe der Visualisierung zu dienen.

5.1.2 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. 14 zu sehen.

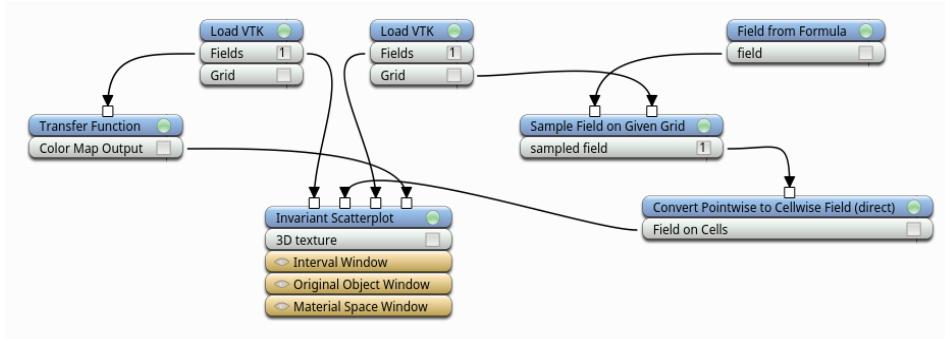


Figure 14: Der Flowgraph der Visualisierungssession.

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. 6 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.

5.2 Voxelisierung

Da das in dieser Arbeit verwendete DVR Verfahren (siehe Abschnitt 5.5) eine Repräsentation der Dichtefelder als 3D Texturen voraussetzt, müssen diese Texturen zunächst erzeugt werden. 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.

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 des Objekts eine Rolle, weshalb diese Werte ansonsten leer gelassen werden.

Um die Textur zu erzeugen, ist es notwendig, die korrekten Werte für die Voxel zu berechnen und das Ergebnis an die Grafikkarte zu übergeben. Dies wird als Voxelisierung bezeichnet. FAnToM besitzt zwar eine effiziente Funktion, um Werte innerhalb eines Feldes zu berechnen, diese ist jedoch nicht in der Lage, mit selbstdurchdringenden Feldern,

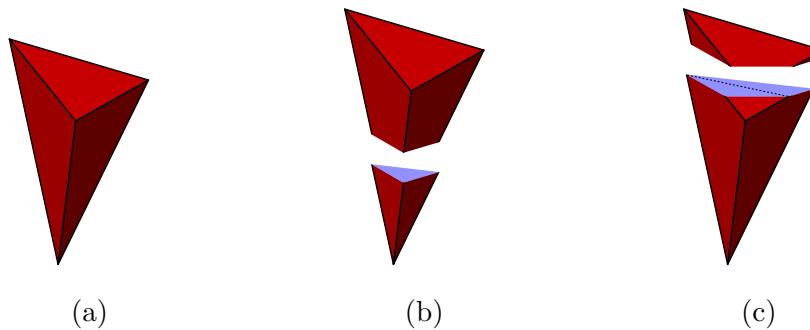


Figure 15: Darstellungen von möglichen Schnitten durch einen Tetraeder. (a) Der Tetraeder ohne Schnitte. (b) Ein Schnitt, der durch ein einzelnes Dreieck dargestellt werden kann. (c) Ein Schnitt, der ein Viereck bildet. Die gestrichelte Linie deutet eine mögliche Unterteilung in Dreiecke an.

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.

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.[23]. 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 ('Triangle Strip') dargestellt werden (siehe Abb. 15). 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 Pseudocode 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 Primitiv: 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 Primitiv gibt, eignet das Format sich jedoch auch, um Tetraeder an die Pipeline zu übergeben.

Da die zylindrische Darstellung in Richtung der X-Achse sehr groß werden kann, ist ein Verfahren 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 Ausdehung einer 3D Textur in jede Richtung.

Vertex Shader Der Vertex Shader bekommt 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 standardmäßig keine Geometry Shader. Mithilfe einer Erweiterung kann das jedoch umgangen werden.

Der Geometry Shader bekommt als Eingabe alle Punkte eines Primitivs 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 leicht zwei überlappungsfreie Dreiecke konstruieren zu können. In Pseudocode 1 wird das Verfahren dargestellt.

Fragment Shader Der Fragment Shader normalisiert die Invarianten auf das Intervall $[0, \dots, 1]$, 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. Zum einen wird sie mit einem vom Nutzer gewählten Faktor (standardmäßig 1) multipliziert. Zum anderen wird, wenn vom Nutzer ausgewählt, die dritte Wurzel der Dichte berechnet. Dies hat sich als sinnvoll erwiesen, da in den verwendeten Datensätzen große Unterschiede in der Dichte der Voxel auftreten, teilweise um mehr als 10 Größenordnungen. Durch das Ziehen der dritten Wurzel nähern

```

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

```

Algorithmus 1: Die Berechnung der Tetraederschnittflächen im Geometry Shader.

sich die Werte weiter der 1 an. Da die Dichtewerte nicht direkt abgelesen werden müssen, stellt auch das kein Problem dar.

Durch den Fragment Shader werden im Invariantenraum mehrere der erzeugten Dreiecke auf dieselben Voxel abgebildet. Die Kombination der Werte pro Voxel erfolgt durch eine von OpenGL bereitgestellte Funktion, dem ‘Blending’. Im Objekt kann dies nicht vorkommen, weshalb die Invariantenwerte in der Textur davon nicht betroffen sind. Durch das Blending wird σ_{V_i} für jeden Tetraeder V_i berechnet und korrekt pro Voxel aufaddiert. Damit ist die zweite Hälfte des Continuous Scatterplottings implementiert.

5.3 Umsetzung des Continuous Scatterplottings

Die beiden beschriebenen Verfahren implementieren gemeinsam ein vollständiges Continuous Scatterplotting. Dies wird in diesem Abschnitt genauer begründet.

Der erste Teil, das Berechnen des Dichtebeitrags σ_{V_i} der Tetraeder V_i , erfolgt bereits in der Vorverarbeitung der Daten. Dabei wird die für Fall 1 angegebene Formel 50 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 Teilmengen 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 solcher extremen Formen von Fall 1 zu den anderen Fällen ist nicht

ohne Weiteres möglich. Fall 4 kommt nicht vor, da die verwendeten Felder überall differenzierbar sind.

Da die V_i eine vollständige Zerlegung des Feldes darstellen, lässt sich Formel 48 anwenden. Die σ_V entsprechen den berechneten Dichtewerten pro Tetraeder, die für jedes Voxel durch das Blending aufaddiert werden.

Die Dichte innerhalb eines Voxels entspricht im Moment stets 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 Scatterplottings, die Masseerhaltung, verstößt. Eine Möglichkeit diesen Fehler zu reduzieren oder sogar komplett zu vermeiden wäre es, pro Voxel die Werte an mehreren Samplepoints zu berechnen und die Dichte auf den Mittelwert der Samplepunkte zu setzen. OpenGL bietet standardmäßig eine Option dieses Supersampling zu implementieren. Da aber Supersampling die Rechenzeit der Voxelisierung stark erhöhen würde und die Voxelisierung schon einen Großteil der Rechenzeit der Visualisierung ausmacht, wurde dies noch nicht implementiert.

5.4 Berechnung der maximalen Dichte

Da Texturen nur Werte im Intervall $[0, \dots, 1]$ enthalten können, aber vor der fertiggestellten Voxelisierung die maximale Dichte in der 3D Textur nicht bestimmt werden kann, muss ein Verfahren implementiert werden, um das korrekte Maximum zu berechnen.

Zunächst wird die höchste überhaupt mögliche Dichte in einem Voxel berechnet. Diese kommt genau dann vor, wenn alle Tetraeder $V_i, i = 1, \dots, n$ sich in einem Voxel überschneiden. 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 auf das Intervall $[0, \dots, 1]$ normalisiert wird.

Als Nächstes wird das Maximum der entstandenen 3D Textur gesucht. Da die 3D Textur relativ groß werden kann (bis zu mehreren Gigabyte), die Übertragung von Daten von der Grafikkarte zum Arbeitsspeicher relativ langsam ist und das Finden des Maximums gut parallelisiert werden kann, wurde auch dieses Problem durch eine Shaderpipeline gelöst. Indem das Maximum (ein Wert zwischen 0 und 1) mit dem größten möglichen Wert multipliziert wird, erhält man das korrekte Maximum. Dies kann verwendet werden, um die Werte einer weiteren Voxelisierung zu normalisieren. Dadurch wird die Präzision in der 3D Textur erhöht.

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 an 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.

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 interpolierten Werte aus dem Vertex Shader berechnet. Dieser Wert entspricht der Schicht, deren Maximum berechnet werden soll. Danach wird über alle Voxel dieser Schicht iteriert und das Maximum der Schicht zurückgegeben. Der Algorithmus ist in 2 dargestellt.

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.

5.5 Umsetzung des Raycastings

Für die beiden DVR Darstellungen wird eine Variante des schon in 4.5.4 beschriebenen Verfahrens 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 Abb. 8. Die Bildebene befindet sich so vor der Kamera, dass sie das gesamte Blickfeld einnimmt. Das Raycasting wird durchgeführt, indem die Shader die Farben auf der Bildebene berechnen.

Der Code basiert auf dem Algorithmus ‘Volume Rendererer GLSL’, erweitert diesen aber deutlich.

Vertex Shader Im Vertex Shader werden lediglich die X- und Y-Koordinaten der Punkte in einer Variable gespeichert und an den Fragment Shader übergeben.

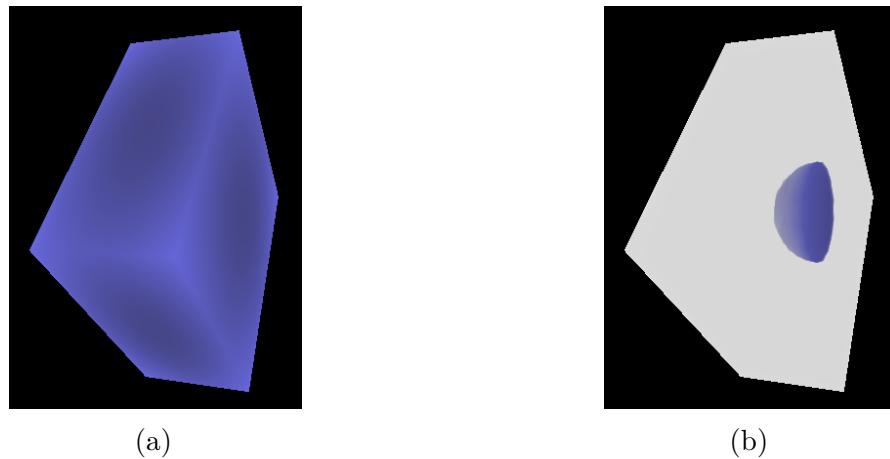


Figure 16: Ergebnisse des Raycastings auf einem quaderförmigen Feld von 3×3 Matrizen. In (a) ist der gesamte Invariantenbereich ausgewählt, in (b) ist er auf hohe Werte begrenzt. Zu beachten ist, dass in (a) die Farben von Punkten näher am Zentrum des Feldes dunkler und in (b) die ausgeblendeten Bereiche als stark transparent und weiß dargestellt werden.

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 (der ‘Bounding Box’, kurz BB) und diesem Strahl berechnet. Abhängig davon, welche Ebenen der Strahl in welcher Reihenfolge geschnitten hat, kann effizient bestimmt werden, ob die 3D Textur vom Strahl durchquert wird oder nicht. Falls nicht, kann sofort Schwarz als Farbe des Fragments zurückgegeben werden.

Falls der Strahl die 3D Textur schneidet, wird der Abstand der Samplepunkte berechnet. Dem Nutzer steht dabei frei, welche der beiden in Abschnitt 8 beschriebenen Varianten er bevorzugt.

Um Artefakte zu verhindern, kann ebenfalls optional ein pseudo-zufälliger Offset erzeugt werden, der die Position des ersten Samplepunkts beeinflusst.

Die eigentliche Berechnung der Fragmentfarbe geschieht in einer Schleife. Für jeden Samplepunkt entlang des Strahls wird die Dichte an diesem Punkt gemessen, durch die Transferfunktion in Farbe und Transparenz übersetzt und nach Formel 42 zu einem Farbwert kombiniert.

Auch die bereits angesprochenen Optimierungen (Samplepunkte beginnen erst am Rand der 3D Textur, Abbruch nachdem die verbleibende Transparenz einen Grenzwert unterschreitet) sind implementiert.

Data: s Strahlvektor der Länge 1, t Schrittweite,
 T 3D Textur, p_0 erster Samplepunkt, B ausgewählter Invariantenbereich

Result: akkumulierte Farbe entlang des Strahls

```

transparenz  $\leftarrow$  1.0;
fragmentfarbe  $\leftarrow$  (0, 0, 0, 0);
for  $i \leftarrow 0$  to  $n$  do
    if transparenz  $<$  grenzwert then
        | break;
    end
    invarianten  $\leftarrow$  invarianten( $T, p$ );
    dichte  $\leftarrow$  dichte( $T, p$ );
    // Nur im Objekt
    if invariantennichtinB then
        | fragmentFarbe  $\leftarrow$  fragmentFarbe + (1.0, 1.0, 1.0, 0.05);
        | transparenz = transparenz - 0.05;
        | continue;
    end
    // Nur im Invariantenraum
    if pnichtinB then
        | continue;
    end
    voxelFarbe  $\leftarrow$  transferfkt(dichte);
    fragmentFarbe  $\leftarrow$  fragmentFarbe + voxelFarbe  $\cdot$  transparenz;
    transparenz = transparenz - voxelFarbe.transparenz;
end
```

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

Wenn beim Rendering des Feldes im Invariantenraumes ein Samplepunkt ausserhalb des ausgewählten Bereichs liegt, wird dort die Farbe und Dichte als 0 angenommen. Wenn dagegen beim Rendern des Objekts ein Samplepunkt an einer Stelle liegt, an der die Invarianten ausserhalb des Bereichs liegen, wird ein stark transparentes Weiß als Farbe festgelegt. Dadurch entsteht ein Schemen des restlichen Objektes, wodurch die Position und Form der noch sichtbaren Bereiche besser einschätzbar wird. Dieser Effekt ist in Abb. 16 zu sehen. Der Pseudocode in Algorithmus 3 beschreibt die Funktionsweise der Schleife.

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. 16a). Dadurch sind Löcher im Volumen leichter zu erkennen, ohne Lichtquellen und Schattenbildung berechnen zu müssen.

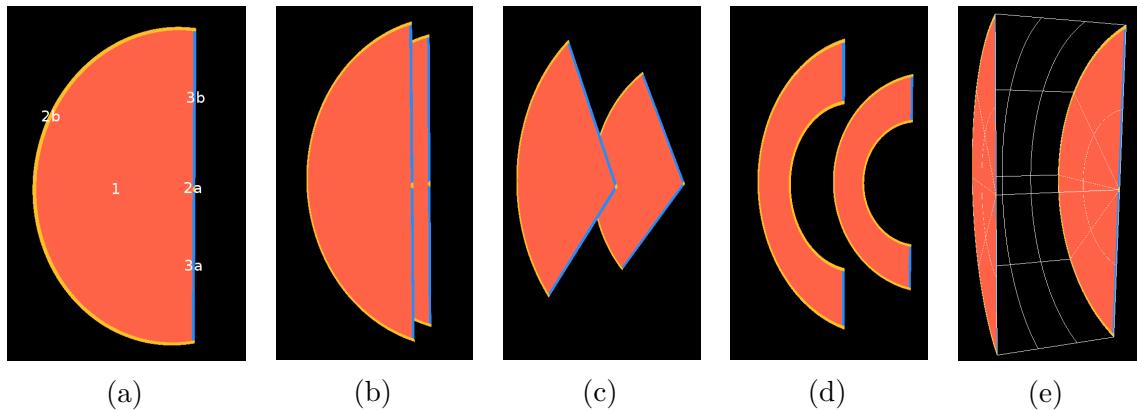


Figure 17: Die Darstellungen der Interaktionsflächen. (a) Eine Interaktionsfläche mit den einzelnen Bereichen. Ein Klick auf die Bereiche 1,2 und 3 wählen die jeweiligen Invarianten aus. Die Bereiche 2a und 3a sind dabei die Minima der Invariante, 2b und 3b die Maxima. (b) bis (d) zeigen veränderte Positionen und Formen der Interaktionsflächen durch gewählte Invariantenbereiche. In (e) ist der Wireframe dargestellt.

5.6 Die Interaktionsflächen

Um in der Darstellung des Feldes im Invariantenraum Bereiche auswählen zu können, wurden Interaktionflächen implementiert. Diese haben die Form von Halbkreisen und können durch die Maus manipuliert werden. Durch Klicken und Halten der mittleren Maustaste können untere und obere Schranken der Invariantenbereiche ausgewählt und durch Bewegung der Maus ihre Werte verändert werden. Eine Interaktionsfläche ist in Abb. 17a dargestellt.

Die Berechnung, ob ein Mausklick eine Interaktionsfläche trifft oder nicht, geschieht mittels ‘Ray-Picking’. Dabei wird, ähnlich wie beim Raycasting, ein Strahl ausgehend von der Kamera durch die Szene berechnet. Wenn der Strahl eine Interaktionsfläche trifft, wird der getroffene Bereich ausgewählt. Wenn beide Interaktionsflächen getroffen werden, wird die nähere von beiden bevorzugt.

Wenn, nachdem ein Bereich ausgewählt wurde, die Maus bewegt wird, passen die Interaktionsflächen ihre Position entsprechend an. Wenn die 1. Invariante gewählt wurde, wird die Interaktionsfläche entlang der X-Achse verschoben. Abhängig davon welche Interaktionsfläche ausgewählt wurde, wird dadurch das Minimum oder Maximum der ersten Invariante gewählt. Die Bereiche können auf einer einzelnen Interaktionsfläche eingestellt werden. Die Form der zweiten wird automatisch angepasst. Die Veränderungen der Formen und Positionen sind in Abb. 17b, 17c und 17d dargestellt.

5.7 Weitere Interaktionen

Es existieren noch eine Reihe weiterer Interaktionen, die auf der Visualisierung durchgeführt werden können:

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.

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.

Setzen des Rotationszentrums Das Rotationszentrum des Invariantenfeldes kann mittels doppeltem Linksklick an einen Punkt entlang der X-Achse gesetzt 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.

5.8 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.

5.8.1 Gemeinsame Optionen

Anzahl von Samplingpunkten Die Anzahl der Samplepunkte 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 Offset vor dem ersten Samplepunkt gesetzt werden.

Helligkeit Ein Slider ermöglicht die Einstellung der Helligkeit des Volume Renderings unabhängig von der Transferfunktion.

Dichte Durch einen weiteren Slider 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 Sliders benötigt werden.

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

Lineare Interpolation Standardmäßig misst das Sampling die Dichtewerte der Textur an einem vorgegebenen Punkt. Da die Texur 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 für den Punkt interpoliert.

Skalierung der Transparenz mit der Anzahl der Samplepunkte im Feld Diese Option macht es möglich, die im Abschnitt [4.5.4](#) und Abb. [5.5](#) erläuterte Skalierung der Transparenz mit der Anzahl der Samplepunkte, die tatsächlich im Feld liegen, an- und auszuschalten.

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.

5.8.2 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 eine ‘Bounding Box’ dargestellt werden.

5.9 Optionen des Invariantenraumrenderings

Wireframe Optional kann ein ‘Wireframe’ angezeigt werden, ein Liniengitter das das Feld begrenzt. Die Position der Linien macht es einfacher, die ausgewählten Bereiche einzuschätzen. Form und Position des Wireframes sind nicht abhängig vom ausgewählten Invariantenbereich. Der Wireframe ist in Abb. 17e zu sehen.

Koordinatenlabel Um die Invariantenbereiche mit der Maus besser einstellen zu können, ist es optional möglich, die Werte der Schranken an den Interaktionsflächen anzuzeigen.

Interaktionsflächen ausblenden Da die Interaktionsflächen 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 mit der Maus verändert werden.

Wurzelskalierung Die im Fragment Shader der Voxelisierung durchgeföhre Skalierung durch Ziehen der dritten Wurzel kann an- und ausgeschaltet werden.

Normalisierung des Feldes Im Invariantenraum kann das Feld beliebige Formen annehmen. Wenn beispielsweise eine Dimension sehr viel kleiner ist als die beiden anderen, kann dies die Betrachtung des Datensatzes erschwererden. Deshalb ist möglich, das Feld durch eine Option zu skalieren. Dabei werden die Begrenzungsflächen des Feldes in allen drei zylindrischen Dimensionen berechnet und die Darstellung des Feldes anschließend so skaliert, dass die Begrenzungsflächen einen Halbzylinder mit fester Höhe und Radius von 1 bilden. Zusätzlich die Skalierung die Vorteile dass leerer Raum in der Darstellung vermieden und kleine Tetraeder besser sichtbar gemacht werden.

5.10 Das Intervallfenster

Um die Invariantenbereiche auf exakte Werte einstellen zu können, existiert ein weiteres Fenster (siehe Abb. 18). Darin sind die sechs Invariantenschranken tabellarisch angegeben. Das linke Textfeld gibt den absoluten, das rechte den relativen Wert im Intervall $[0, \dots, 1]$ an. Alle Textfelder sind editierbar und Änderungen werden direkt auf die DVRs angewendet.

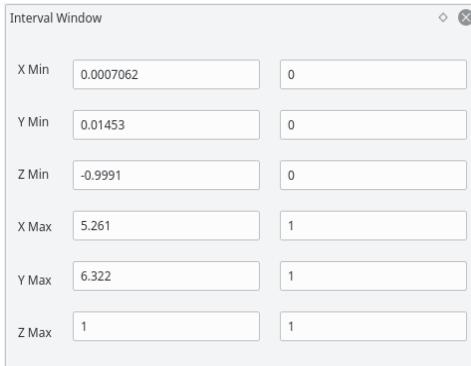


Figure 18: Das Intervall-Fenster.

6 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 mögliche Interpretationen vorgestellt.

Als Vergleichsdarstellung, und um eine Vorstellung über die Struktur der Datensätze zu vermitteln, wurde für jeden Datensatz eine Visualisierung der Tensoren als Ellipsoidglyphen erzeugt. 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.

6.1 Single Point Load

Der erste Testdatensatz ist ein von FAnToM erzeugtes Tensorfeld. Dabei wird simuliert, dass auf einen unendlichen Halbraum an einem Punkt eine Kraft einwirkt. Die entstehende mechanische Spannung wird an Punkten innerhalb des Objekts gemessen und die Tensoren in Form von 3×3 Matrizen gespeichert. Darstellungen des Datensatzes sind in 19 zu sehen. Der Punkt, an dem die Kraft einwirkt ist im der linken Hälfte der vorderen, quadratischen Seite.

Abb. 19a zeigt eine Darstellung der Tensoren als Ellipsoidglyphen. Die Längen der Halbachsen entsprechen den Beträgen der Eigenwerte. 19c zeigt die Darstellung des Datensatzes im Invariantenraum in grün mit Interaktionsflächen links und rechts davon. Das DVR des Objekts ist in Abb. 19b zu sehen.

In Abb. 20 wurden verschiedene Invariantenbereiche ausgewählt. Verwendet wurde der K-Invariantensatz. Bei 20a und 20b sind die Bereiche mit niedrigem K_1 ausgeblendet, in 20c und 20d die Bereiche mit niedrigem K_2 und in 20e und 20f sind nur noch Bereiche mit sehr hohem K_3 zu sehen.

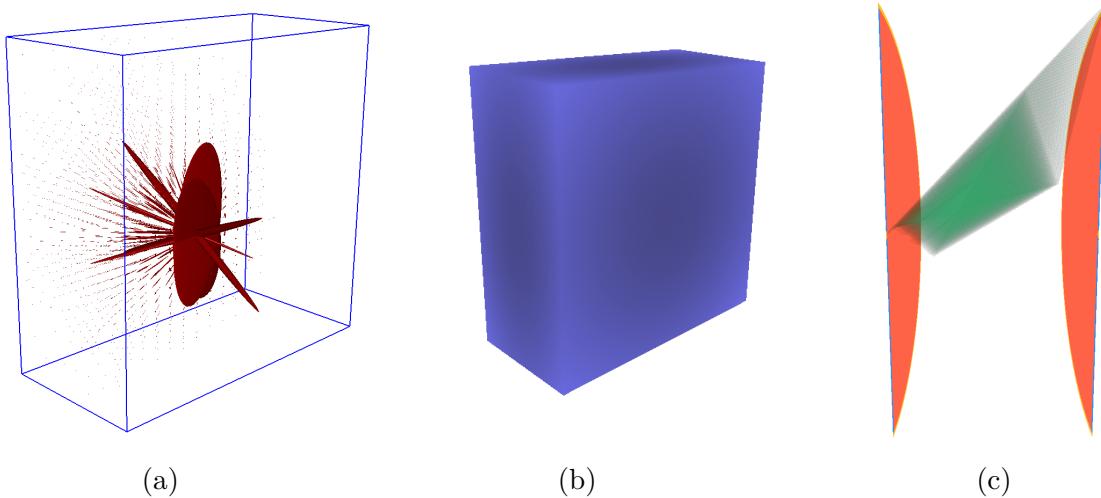


Figure 19: Darstellung des Single Point Load Datensatzes: (a) als Ellipsoide, (b) als DVR des Objekts und (c) im Invariantenraum, berechnet mit den K-Invarianten.

Aus diesen Darstellungen lässt sich ein starker Zusammenhang zwischen der ersten (Spur) und zweiten (Norm des Deviators) Invariante im Datensatz herleiten. Das Ausblenden niedriger Bereiche für die erste und zweite Invariante führt zu sehr ähnlichen Effekten im DVR des Objekts.

Aus der Darstellung von Bereichen mit hohem Modus (20e und 20f) lässt sich ableiten, dass die einwirkende Kraft Tensoren mit einem einzelnen Eigenwert mit hohem Betrag erzeugt, die sich in einem kegelförmigen Bereich ausgehend vom Auftrittspunkt der Kraft befinden.

Insgesamt lassen sich aus der Darstellung der Invariantenbereiche eine Reihe von Informationen ziehen, die aus klassischen Visualisierungen nicht oder nur mit großem Aufwand abgelesen werden können. Der funktionale Zusammenhang zwischen K_1 und K_2 ist mithilfe der Interaktionen leicht zu sehen, eine Eigenschaft des Datensatzes, die beispielsweise in der Glyphendarstellung in 19a nur schwer erkennbar ist. Auch Bereiche, in keine oder sehr kleine Glyphen gezeichnet wurden, lassen sich leichter analysieren, wie z.B. die Bereiche mit hohem Modus zeigen.

6.2 Metallscheibe

Als zweites Beispiel wurde das Ergebnis einer thermo-mechanischen Simulation, durchgeführt von Dr. Thomas Nagel, verwendet, die auch in der Arbeit von Fritzsch benutzt wurde [12, S.15, 45 ff.]. Fritzsch beschreibt den Datensatz 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.”[12, S. 15]

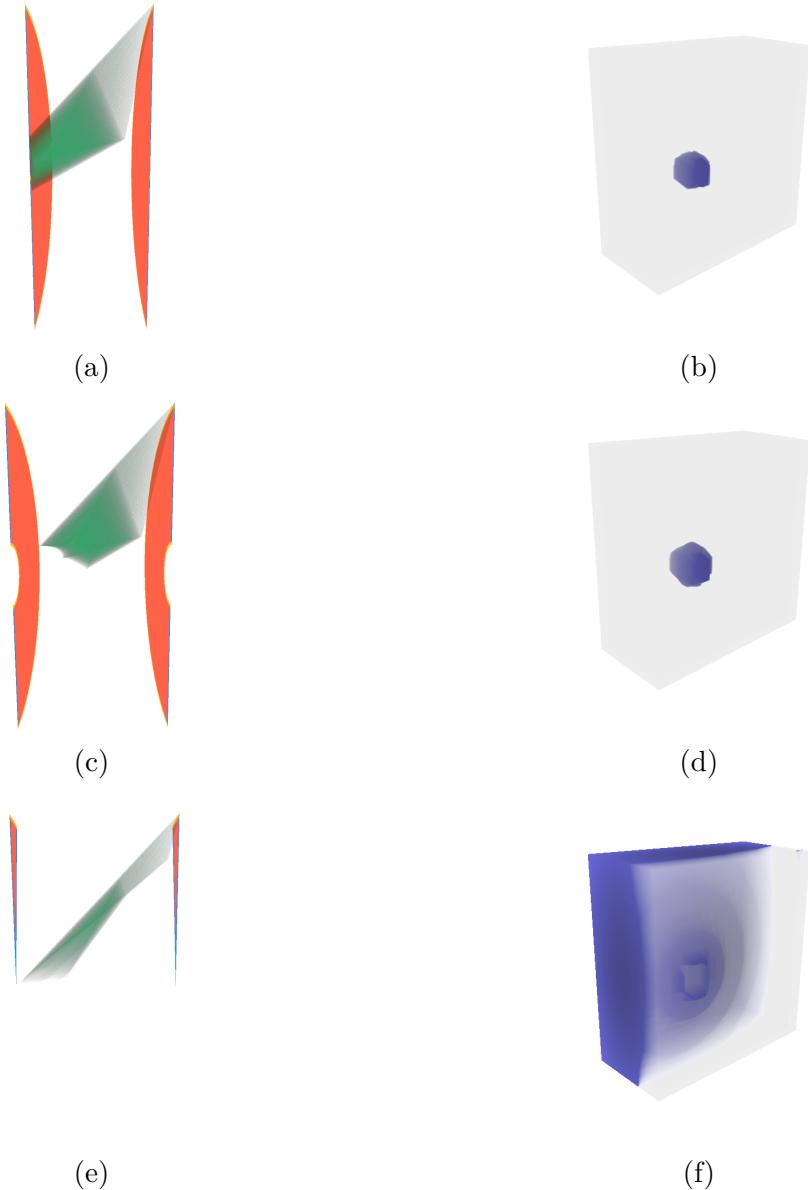


Figure 20: Die mechanische Spannung des Single Point Load-Datensatzes mit ausgewählten Invariantenbereichen. (a) und (b) zeigen Bereiche mit hohem K_1 , (c) und (d) Bereiche mit hohem K_2 , (e) und (f) Bereiche mit hohem K_3 .

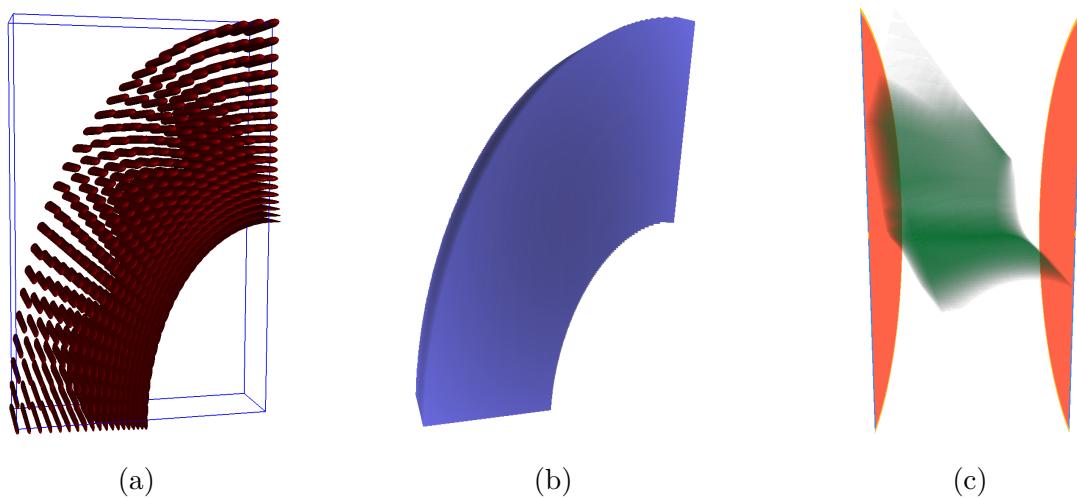


Figure 21: Darstellung des Metallscheiben-Datensatzes: (a) als Ellipsoide, (b) als DVR des Objekts und (c) im Invariantenraum, berechnet mit den K-Invarianten.

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.

Da die Metallscheibe spiegelsymmetrisch zur 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. Für beide Datensätze wurden getrennt voneinander Visualisierungen mit dem K-Invariantensatz erzeugt, die im Folgenden vorgestellt werden.

6.2.1 Spannung

Abb. 21a zeigt die Spannungstensoren der Metallscheibe in Form von Ellipsoidglyphen. Pro Punkt wird eine Glyphe gezeichnet, die sich aber überschneiden können, was die Interpretation erschwert. Erkennbar ist jedoch, dass die Glyphen am äußeren Rand der Scheibe eher länglich und die am inneren Rand eher scheibenförmig sind. Das bedeutet, dass die Tensoren am äusseren Rand einen einzelnen Eigenwert mit hohem und zwei Eigenwerten mit kleinem Betrag besitzen, wogegen am inneren Rand zwei Eigenwerte hohe und nur einer einen kleinen Betrag hat. In beiden Bereichen ist hohe Anisotropie zu erwarten. Gleichzeitig nimmt die Größe der Glyphen im Bild von unten links nach oben rechts zu. Dies ist darauf zurückzuführen, dass nach oben rechts hin negative Eigenwerte mit immer größerem Betrag vorkommen. In der Glyphendarstellung gehen Informationen über die Vorzeichen der Eigenwerte verloren.

Die Darstellungen der Invariantenbereiche (Abb. 22) lassen weitere Interpretationen zu. In Abb. 22b ist zu sehen, dass die Spur im oberen rechten Teil des Scheibenausschnitts am kleinsten ist, da die negativen Eigenwerte an dieser Stelle den größten Betrag haben. Abb. 22d zeigt, dass die Norm des Deviators am inneren und äußeren Rand der Scheibe

groß und im Bereich dazwischen eher klein ist, was auf hohe Anisotropie hinweist. Der Modus ist am inneren Rand der Scheibe groß und am äußeren klein (Abb. [22e](#) und [22f](#)). Das bedeutet, dass die Anisotropie am inneren Rand eher planar und am äusseren Rand eher linear ist.

Eine weitere Information, die aus der Visualisierung der Invariantenbereiche gewonnen werden kann, ist, dass sowohl der isotrope als auch der anisotrope Anteil der Spannung auf der Rückseite der Scheibe höher ist als auf der Vorderseite. Das ist darauf zurückzuführen, dass die Rückseite erhitzt wird, sich also das Material ausdehnt. Aus der Darstellung der Glyphen in Abb. [21a](#) ist dies nur schwer abzulesen, da weder die Form noch die Größe der Glyphen auf Vorder- und Rückseite stark voneinander abweicht. Zusätzlich erlauben die DVRs auch die Analyse von Tensoren im Inneren der Metallscheibe, was in der Glyphendarstellung durch Überlagerung der Glyphen nicht möglich ist.

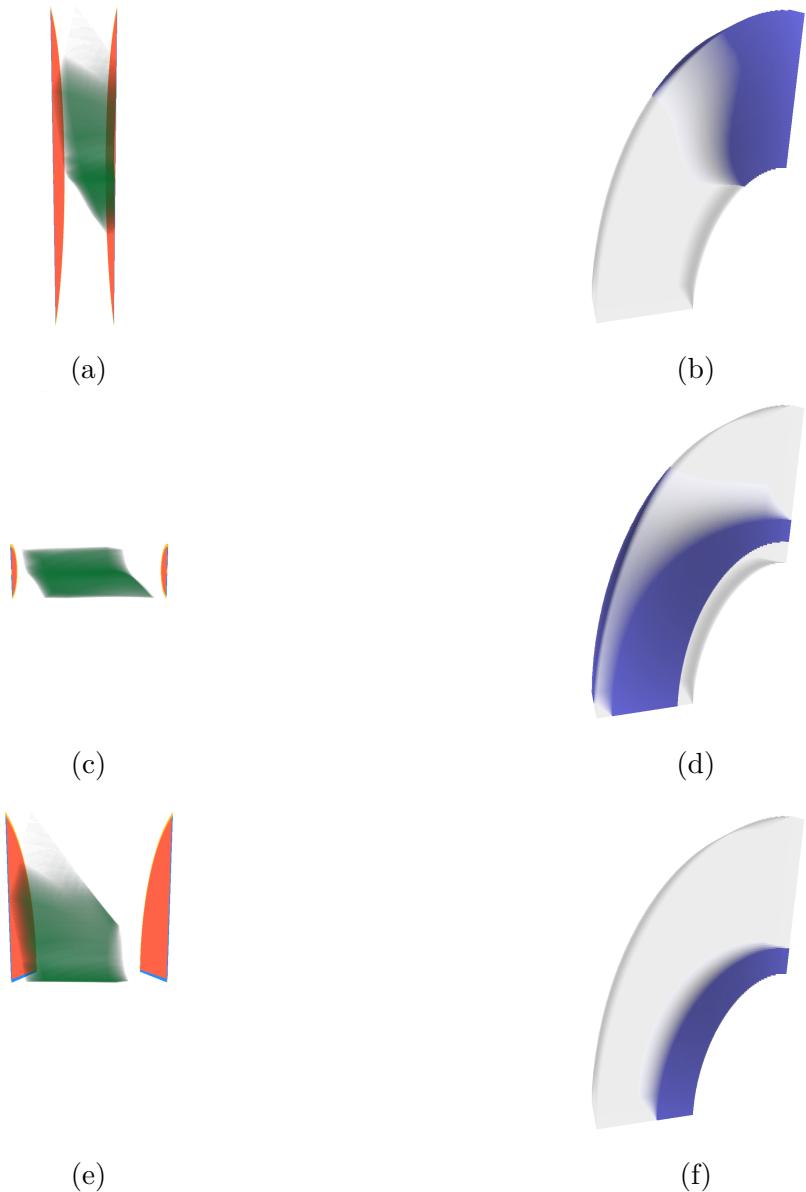


Figure 22: Die mechanische Spannung des Metallscheiben-Datensatzes mit ausgewählten Invariantenbereichen. (a) und (b) zeigen Bereiche mit niedrigem K_1 , (c) und (d) Bereiche mit niedrigem K_2 , (e) und (f) Bereiche mit hohem K_3 .

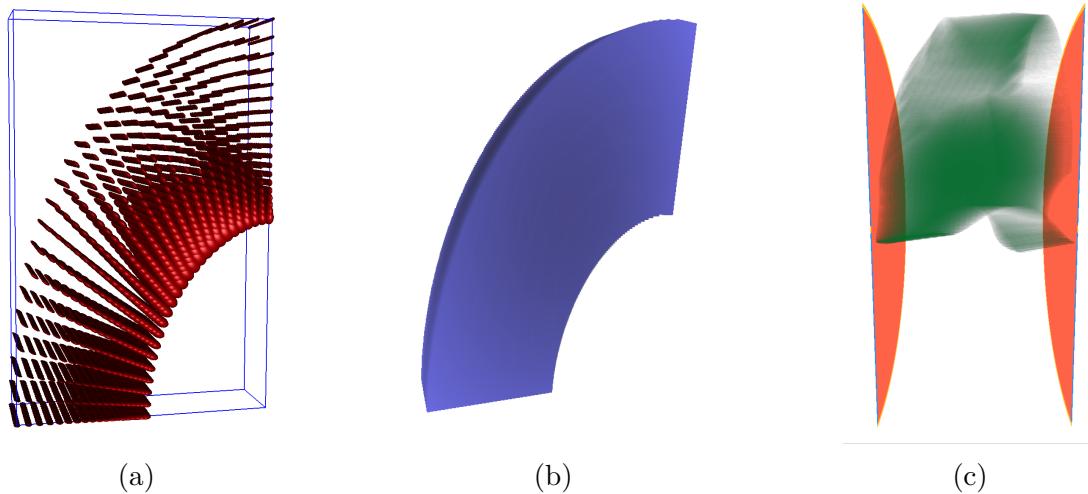


Figure 23: Darstellung des Metallscheiben-Datensatzes: (a) als Ellipsoide, (b) als DVR des Objekts und (c) im Invariantenraum, berechnet mit den K-Invarianten.

6.2.2 Verformung

Die Verformungstensoren der Metallscheibensimulation sind in Abb. 23a dargestellt. Erkennbar ist, dass die Glyphen am inneren Rand erheblich größer sind als alle anderen. Die Eigenwerte dieser Tensoren haben tatsächlich einen relativ hohen Betrag, sind jedoch teilweise negativ, was dazu führt, dass auch die Spur dort niedrig oder sogar negativ ist. Dies fällt auch in Abb. 24a auf, wo Bereiche niedriger Spur ausgeblendet werden. Die Spur ist, ähnlich wie beim Spannungsdatensatz, unten links größer als oben rechts. K_2 , und somit auch die Anisotropie, ist in einem halbkreisförmigen Bereich an der Innenseite der Scheibe am größten (siehe Abb. 24c). Im gleichen Bereich ist auch K_3 hoch (siehe Abb. 24e). Das Material wird also am inneren Rand der Scheibe in die Richtung zweier Eigenvektoren zusammengepresst und dehnt sich in entlang des dritten aus. Aus der Glyphendarstellung wird auch die Richtung dieses Eigenvektors ersichtlich, nämlich zum Mittelpunkt der Scheibe hin. Die Scheibe dehnt sich also in diesem Bereich zur Mitte hin aus.

Die Vorteile der Darstellung der Invarianten sind identisch zum Spannungs-Datensatz: Unterschiede, die sonst nur zu geringen Formänderungen der Ellipsoide führen, sind leichter zu erkennen und ohne Überdeckungen sind Strukturen im Inneren der Metallscheibe sichtbar. Insbesondere die Unterschiede zwischen den Tensoren des oberen rechten und unteren linken Teils der Scheibe werden deutlich, wogegen die Ellipsoidglyphen in diesen Bereichen sehr ähnlich sind (Abb. 23a).

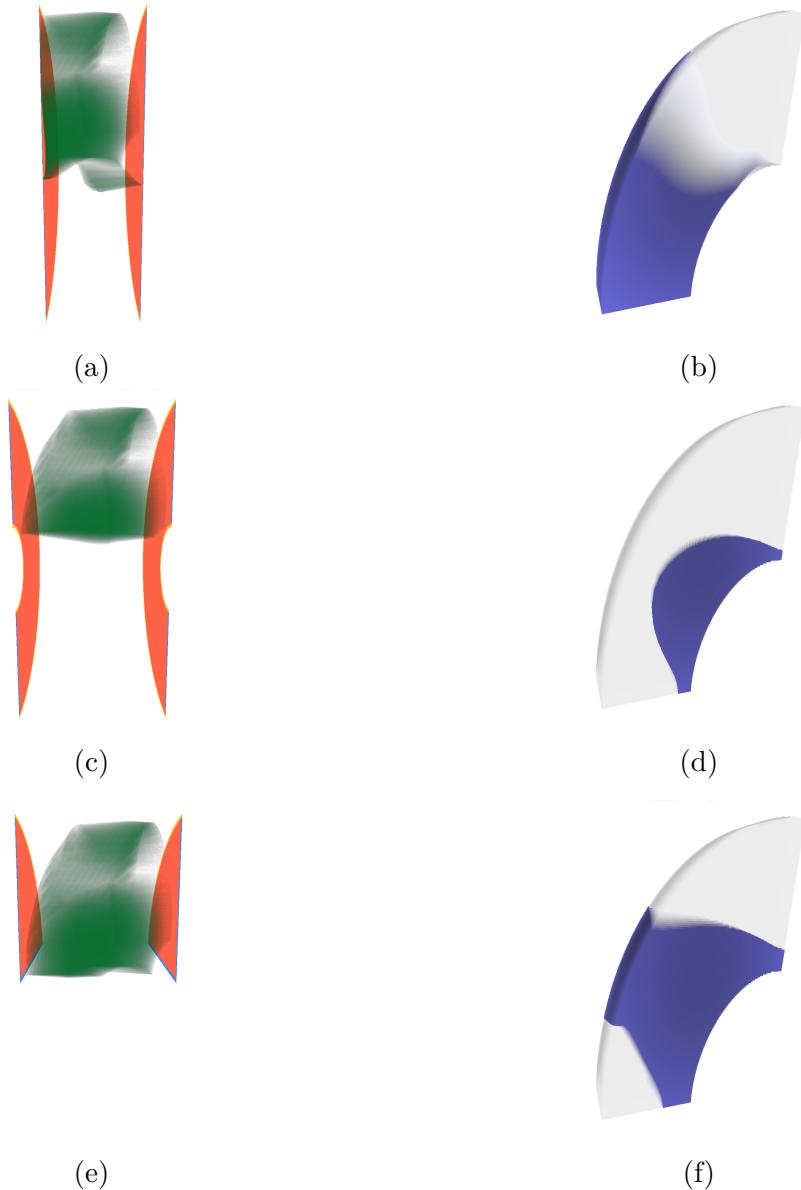


Figure 24: Die Verformung des Metallscheiben-Datensatzes mit ausgewählten Invariantenbereichen. (a) und (b) zeigen Bereiche mit hohem K_1 , (c) und (d) Bereiche mit niedrigem K_2 , (e) und (f) Bereiche mit hohem K_3 .

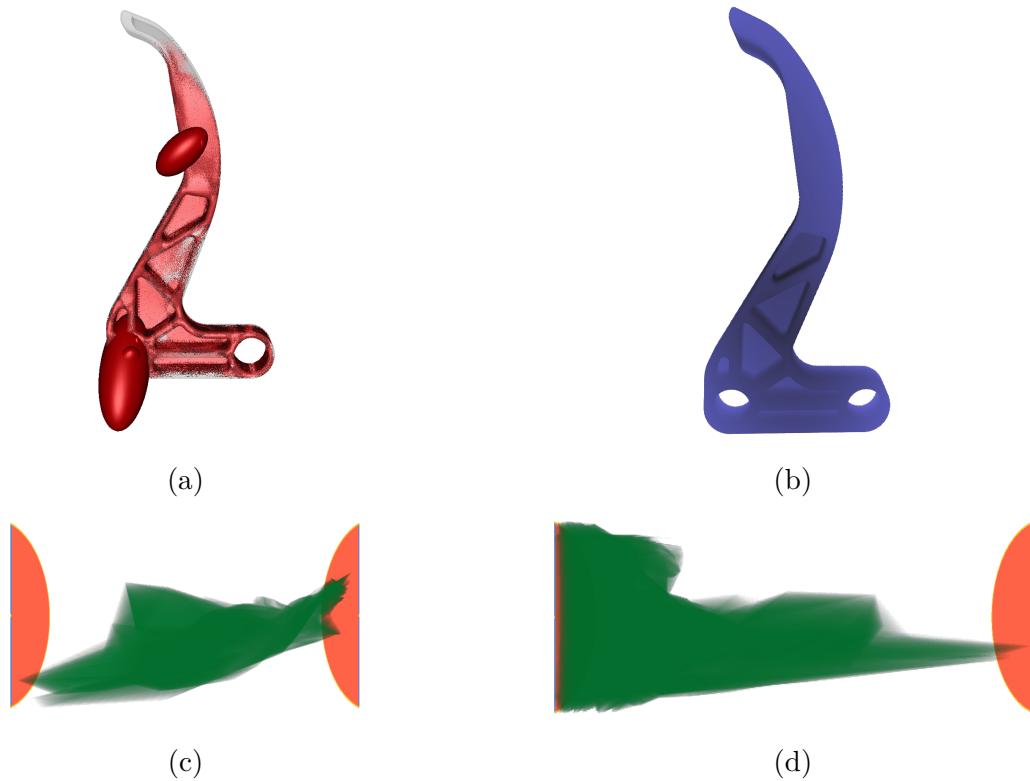


Figure 25: Darstellung des Bremshebel-Datensatzes: (a) als Ellipsoide, (b) als DVR des Objekts, (c) im K-Invariantenraum und (d) im R-Invariantenraum.

6.3 Bremshebel

Die bisherigen Datensätze behandelten ausschließlich einfache geometrische Objekte, auf die Kraft einwirkte. Der Bremshebel-Datensatz wurde ausgewählt, um die Ergebnisse der Visualisierung auch auf einem komplexeren Objekt zeigen zu können.

Beim Bremshebel-Datensatz handelt es sich wie zuvor um Simulationsdaten. Dabei wurden die mechanischen Spannungen des Hebels einer Fahrradbremse simuliert, wenn auf den Hebelarm eine Kraft einwirkt. Der Hebel ist dabei eingespannt, um Bewegung zu verhindern. Die einwirkende Kraft sowie die Stelle, an der der Hebel eingespannt ist, sind gut in Abb. 25a zu erkennen. Die scheinbare Rotfärbung des Hebels stammt dabei aus der großen Menge kleiner Glyphen, die an den Punkten des Hebels gemessen wurden.

Die Simulation wurde mit Abaqus [Standardversion 6.13-4][26] erzeugt.

Die Glyphendarstellung zeigt, dass an zwei Stellen des Objekts besonders große Spannungen wirken: Am Hebelarm, an dem die Kraft einwirkt und unten links, wo der

Hebel fixiert ist. Die große Anzahl und deshalb notwendige geringe Größe der restlichen Glyphen machen eine Interpretation jedoch praktisch unmöglich.

Der Bremshebel wurde jeweils einmal mit dem K- und einmal mit dem R-Invariantensatz visualisiert.

6.3.1 K-Invarianten

Die Darstellung durch Glyphen wird durch die große Anzahl der Datenpunkte erschwert. Dieses Problem tritt im Rendering des Invariantenraumes nicht auf, da mehr Datenpunkte nur zu exaktere Verläufen der Isoflächen führen. In Abb. 27a und 27b ist erkennbar, dass die Spannungstensoren am Auftrittspunkt der Kraft, entlang der linken Seite des Hebels und einigen Streben im Inneren des Hebels am größten sind. Der anisotrope Anteil ist am Auftrittspunkt der Kraft, an der Beuge des Hebels und an den Zusammentreffpunkten einiger Streben am stärksten (siehe Abb. 26c und 26d).

Bereiche mit niedrigem Modus, also mit planarer Anisotropie, sind in Abb. 26e ausgewählt. Der am deutlichsten erkennbare Bereich liegt an der Beuge des Hebels, wo das Material zusammengepresst wird. Aber auch am Auftrittspunkt der Kraft ist der Modus gering, da Kraft dort entlang der Oberfläche weitergeleitet wird.

Tensoren mit hohen Werten für K_2 deuten häufig auf Bereiche hin, innerhalb derer starke anisotrope Kräfte wirken. Solche Kräfte führen oft zu starken Verformungen oder sogar zu Brüchen des Materials. Gerade deshalb ist eine Untersuchung des Tensorfeldes auf Bereiche mit hohem K_2 ein wichtiger Anwendungsfall. Vom Bremshebedatensatz existieren einige Varianten, die sich in der Anzahl und Form der inneren Verstrebungen unterscheiden. Die Untersuchung der Höhe und Verteilung der Werte für K_2 kann dabei wichtige Hinweise darauf liefern, welche Verstrebungen die Kraft besser verteilen als andere.

6.3.2 R-Invarianten

Trotz der sehr unterschiedlichen Form des Feldes im Invariantenraum (Abb. 25d), ähneln sich die Bereiche, in denen die die erste Invariante hohe Werte annimmt. Da R_1 und K_1 beide Maße für die Isotropie Tensoren sind, ist dieses Verhalten zu erwarten. Die Bereiche für hohe R_2 und K_2 , beide Maße für die Anisotropie, unterscheiden sich jedoch stark, R_2 scheint überall im Inneren des Objekts hoch zu sein, darunter Bereiche, in denen es K_2 eher niedrig ist.

Die Bereiche mit niedrigem Modus (Abb. 27e) sind, da die Ausgangsdaten die gleichen sind, natürlich identisch.

Gerade in Abb. 27d zeigt sich ein weitere Vorteil der implementierten Visualisierung: Die Bereiche mit hohem R_2 sind überall im Objekt verteilt. Diese Bereiche klar darzustellen

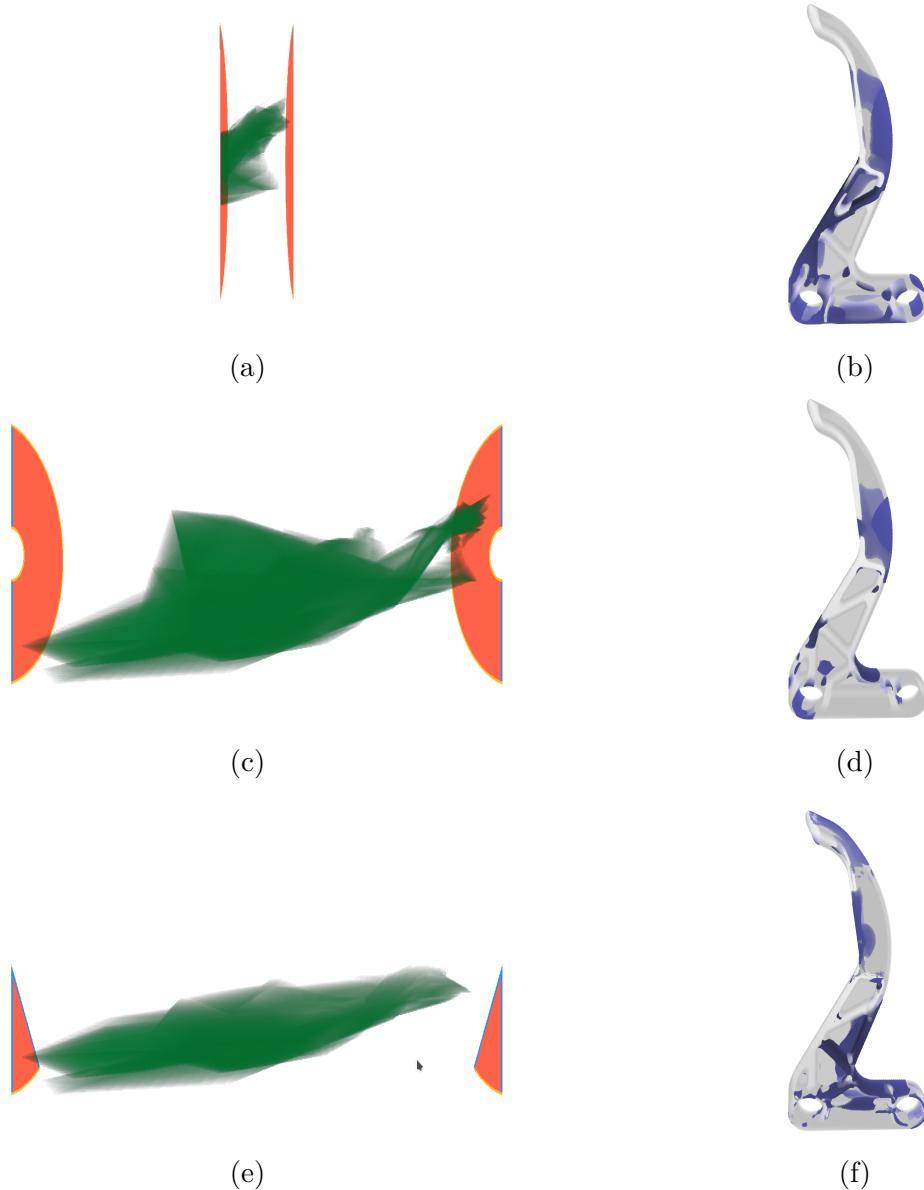


Figure 26: Die mechanische Spannung des Bremshebel-Datensatzes mit ausgewählten Invariantenbereichen. (a) und (b) zeigen Bereiche mit hohem K_1 , (c) und (d) Bereiche mit hohem K_2 , (e) und (f) Bereiche mit niedrigem K_3 .

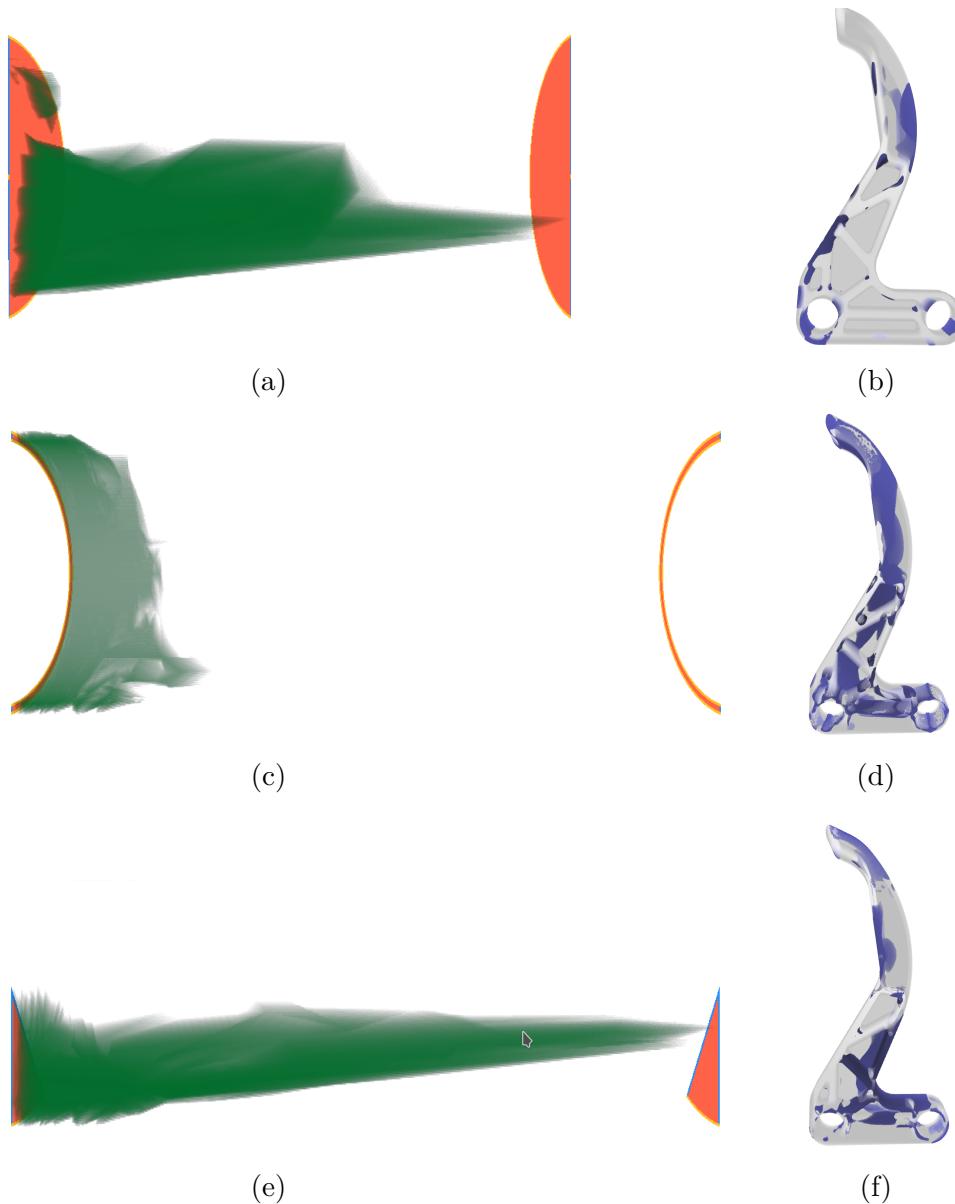


Figure 27: Die mechanische Spannung des Bremshebel-Datensatzes mit ausgewählten Invariantenbereichen. (a) und (b) zeigen Bereiche mit hohem R_1 , (c) und (d) Bereiche mit hohem R_2 , (e) und (f) Bereiche mit niedrigem R_3 .

wäre bei den meisten klassischen Visualisierungen nur schwer möglich, hier dagegen ist es ein Leichtes den passenden Bereich auszuwählen.

6.4 Fazit

Aus den genannten Beispieldatensätzen hat sich ergeben, dass das implementierte Verfahren folgende Vorteile gegenüber den klassischen Visualisierungen bietet:

Durch die Voxelisierung ist es dem Benutzer möglich, die Auflösung der Darstellung selbst festzulegen. Das hat den Vorteil, dass ein Kompromiss zwischen Geschwindigkeit und Qualität der Visualisierung gefunden werden kann, unabhängig von der Menge an Datenpunkten im Feld.

Im Gegensatz zu z.B. der Darstellung von Tensoren als Ellipsoide oder Superquadrics kommt es nicht zu Überdeckungen. Durch das DVR kann das Innere der Datensätze leicht betrachtet werden. Außerdem können die Werte der Tensoren sich um mehrere Größenordnungen unterscheiden, ohne dass die Qualität der Darstellung sich verschlechtert.

Die meisten in wählten klassischen Tensorfeldvisualisierungen verwenden die Eigenwerte und -vektoren um Tensoren darzustellen. Der Betrag eines einzelnen Eigenwertes liefert jedoch oft nicht sehr aussagekräftig, viel wichtiger ist das Verhältnis der Eigenwerte zueinander. Invarianten, wie sie vorgestellt wurden, liefern diese Informationen, weshalb sie in der Kontinuumsmechanik und der Untersuchung von Materialien verwendet werden.

Durch die in Echtzeit vornehmbaren Interaktionen ist es zudem möglich, Bereiche mit bestimmten Invariantenkombinationen hervorzuheben. Diese explorative Analyse unterscheidet die Visualisierung von den klassischen statischen Ansätzen und bietet die Möglichkeit, uninteressante Bereiche auszublenden. Auch Zusammenhänge zwischen den Verteilungen der Invarianten im Objekt können durch die Interaktionen leicht gefunden werden. Selbst kleine Veränderungen der Invarianten über den Datensatz hinweg lassen sich verfolgen.

7 Ausblick

Innerhalb der vorliegenden Arbeit wurde ein System entwickelt und evaluiert, das die Untersuchung von Tensorfeldern mithilfe von Invariantensätzen ermöglicht. Die Umsetzung liefert Ergebnisse, die mit den Erwartungen übereinstimmen. Dadurch, dass die Darstellung durch Interaktionen in Echtzeit angepasst werden kann ist eine explorative Analyse der Daten möglich.

Die Ergebnisse wurden Experten für Kontinuumsmechanik von der Technischen Universität Dortmund und dem Helmholtz-Zentrum für Umweltforschung fürvorgeführt,

wobei die Reaktionen positiv waren. Es wurde jedoch kritisiert, dass die explorative Analyse intensive Kenntnisse der Datensätze voraussetzt. In der Praxis werden so genannte Materialmodelle eingesetzt, die das Verhalten von Materialien beschreiben. Ausgehend davon lassen sich Invariantenbereiche definieren. Diese sind jedoch nicht immer zylinderachsenparallel wie die bereits implementierten Bereiche. Dies wäre z.B. bei einem Material der Fall, das bei höheren isotropen Spannungen auch höhere anisotrope Spannungen aushält. Ein solches Materialmodell würde im 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.

Ein weiterer Ansatzpunkt für Verbesserungen wäre das Voxelisierungsverfahren. Da die Voxelisierung für einen Großteil der Rechenzeit der Visualisierung verantwortlich ist, wäre sie ein guter Ansatzpunkt für Optimierungen. Dadurch wäre es auch möglich, die Masseerhaltung durch Supersampling zu implementieren, ohne die benötigte Zeit zu stark zu erhöhen.

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

Alles in allem stellt das implementierte Verfahren eine neue Variante der Tensorfeldvisualisierung dar. Die Interaktionen ermöglichen es Benutzern den Datensatz explorativ zu analysieren, die DVRs und die aus dem Continuous Scatterplotting übernommenen Verfahren legen innere Strukturen der Datensätze offen.

References

- [1] Universität Leipzig Abteilung für Bild- und Signalverarbeitung. *FAnToM Website*. URL: <http://www.informatik.uni-leipzig.de/fantom/content/about-fantom> (visited on 03/25/2018).
- [2] Lisa S Avila et al. *The VTK User's Guide*. Kitware New York, 2010.
- [3] Sven Bachthaler and Daniel Weiskopf. “Continuous scatterplots”. In: *IEEE transactions on visualization and computer graphics* 14.6 (2008), pp. 1428–1435.
- [4] Peter J Basser, James Mattiello, and Denis LeBihan. “MR diffusion tensor spectroscopy and imaging”. In: *Biophysical journal* 66.1 (1994), pp. 259–267.
- [5] Ray M Bowen and Chao-Cheng Wang. *Introduction to vectors and tensors*. Vol. 1. Courier Corporation, 2008.
- [6] The Qt Company. *Qt Website*. URL: <https://www.qt.io/> (visited on 03/28/2018).
- [7] John C Criscione et al. “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), pp. 2445–2465.
- [8] Thierry Delmarcelle and Lambertus Hesselink. “Visualizing second-order tensor fields with hyperstreamlines”. In: *IEEE Computer Graphics and Applications* 13.4 (1993), pp. 25–33.
- [9] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. “Volume rendering”. In: *ACM Siggraph Computer Graphics*. Vol. 22. 4. ACM. 1988, pp. 65–74.
- [10] Daniel B Ennis and Gordon Kindlmann. “Orthogonal tensor invariants and the analysis of diffusion tensor magnetic resonance images”. In: *Magnetic resonance in medicine* 55.1 (2006), pp. 136–146.
- [11] Richard P Feynman, Robert B Leighton, and Matthew Sands. *The Feynman lectures on physics, Vol. I: The new millennium edition: mainly mechanics, radiation, and heat*. Vol. 1. Basic books, 2011.
- [12] Clemens Fritzsch. “Visuelle Analyse kontinuumsmechanischer Simulationen durch kontinuierliche Streudiagramme”. Diploma Thesis, University of Leipzig. MA thesis. Leipzig, Germany: Universität Leipzig, 2016.
- [13] Keith D. Hjelmstad. *Fundamentals of Structural Mechanics*. 10th ed. Springer US Verlag KG, 2005. ISBN: 978-3-13-477010-0.
- [14] Mario Hlawitschka et al. “Top Challenges in the Visualization of Engineering Tensor Fields”. In: *Visualization and Processing of Tensors and Higher Order Descriptors for Multi-Valued Data*. Springer, 2014, pp. 3–15.
- [15] The Khronos™ Group Inc. *OpenGL Website*. URL: <https://www.khronos.org/opengl/> (visited on 03/28/2018).
- [16] Arie Kaufman and Klaus Mueller. “Overview of volume rendering”. In: *The visualization handbook* 7 (2005), pp. 127–174.

- [17] Gordon Kindlmann. “Superquadric tensor glyphs”. In: *Proceedings of the Sixth Joint Eurographics-IEEE TCVG conference on Visualization*. Eurographics Association. 2004, pp. 147–154.
- [18] Gordon Kindlmann and 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, pp. 183–189.
- [19] Gordon Kindlmann et al. “Diffusion tensor analysis with invariant gradients and rotation tangents”. In: *IEEE Transactions on Medical Imaging* 26.11 (2007), pp. 1483–1499.
- [20] Norbert Kusolitsch. *Maß-und Wahrscheinlichkeitstheorie: Eine Einführung*. Springer-Verlag, 2014.
- [21] Bruce R Kusse and Erik A Westwig. *Mathematical physics: applied mathematics for scientists and engineers*. John Wiley & Sons, 2010.
- [22] Tamara Munzner. *Visualization analysis and design*. AK Peters/CRC Press, 2014.
- [23] Antonio J Rueda et al. “Voxelization of solids using simplicial coverings”. In: (2004).
- [24] Gerik Scheuermann and Matthias Goldau. *Vorlesung Visualisierung in Naturwissenschaft und Technik*. 2015.
- [25] William J Schroeder and Kenneth M Martin. “Overview of visualization”. In: *The visualization handbook* (2005), pp. 3–35.
- [26] Dassault Systèmes. *Abaqus Website*. URL: <https://www.3ds.com/products-services/simulia/products/abaqus/> (visited on 04/26/2018).
- [27] Alexander Wiebel et al. “Fantom-lessons learned from design, implementation, administration, and use of a visualization system for over 10 years”. In: (2009).
- [28] Valentin Zobel and Gerik Scheuermann. “Extremal curves and surfaces in symmetric tensor fields”. In: *The Visual Computer* (2017), pp. 1–16.