

## Masterarbeit

# Continuous Scatterplotting von Tensorfeldern in 3D

Niklas Teichmann

31. Januar 2019

**Zusammenfassung:** Die Visualisierung von Tensorfeldern ist trotz vieler Jahrzehnte der Forschung noch immer ein aktives Thema. Komplexe Strukturen innerhalb von Tensorfeldern stellen hohe Anforderungen an Visualisierungssoftware, um 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 zweiten 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

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Verwandte Arbeiten</b>	<b>2</b>
<b>3 Grundlagen</b>	<b>3</b>
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 Lineare Abbildungen . . . . .	5
3.1.6 Vektorraum . . . . .	5
3.1.7 Dualraum . . . . .	6
3.1.8 Multilineare Funktionen . . . . .	6
3.1.9 Tensor . . . . .	6
3.1.10 Jacobi Matrix . . . . .	7
3.1.11 Determinante einer Matrix . . . . .	7
3.1.12 Determinante der Jacobimatrix . . . . .	8
3.1.13 Rang einer Matrix . . . . .	8
3.1.14 Spur einer Matrix . . . . .	8
3.1.15 Norm einer Matrix . . . . .	9
3.1.16 Deviator einer Matrix . . . . .	9
3.1.17 Eigenwerte und Eigenvektoren einer Matrix . . . . .	10
3.1.18 Fraktionale Anisotropie einer Matrix . . . . .	10
3.1.19 Modus einer Matrix . . . . .	10
3.1.20 Gradient . . . . .	11
3.1.21 Orthogonalität von Matrizen . . . . .	12
3.1.22 Matrixinvarianten . . . . .	12
3.1.23 Invariantensätze . . . . .	13
3.1.24 Faltung . . . . .	16
3.1.25 Gaußsche Glockenkurve . . . . .	16
3.2 Mechanische Grundlagen . . . . .	17
3.2.1 Physikalisches Feld . . . . .	17
3.2.2 Mechanische Spannung . . . . .	18
3.2.3 Mechanische Verformung . . . . .	18
<b>4 Verwendete Verfahren und Technologien</b>	<b>19</b>
4.1 FAnToM . . . . .	19
4.2 OpenGL . . . . .	19
4.3 Qt . . . . .	21
4.4 Continuous Scatterplotting . . . . .	21
4.4.1 Scatterplotting . . . . .	21

4.4.2	Erklärung des Continuous Scatterplottings . . . . .	22
4.5	Volumetrische Daten . . . . .	27
4.6	Volume Visualization . . . . .	28
4.6.1	Isoflächen . . . . .	29
4.6.2	Slicing . . . . .	30
4.6.3	Direct Volume Rendering . . . . .	31
<b>5</b>	<b>Direct Volume Rendering Verfahren</b>	<b>36</b>
5.1	Texture Slicing . . . . .	36
5.2	Raycasting . . . . .	38
5.3	Cell Projection . . . . .	41
5.4	Splatting . . . . .	43
5.5	Shear Warp . . . . .	44
<b>6</b>	<b>Umsetzung</b>	<b>46</b>
6.1	Datenvorbereitung . . . . .	46
6.1.1	Die Vorbereitungssession . . . . .	46
6.1.2	Die Visualisierungssession . . . . .	49
6.2	Wahl des DVR Verfahrens . . . . .	49
6.3	Voxelisierung . . . . .	51
6.4	Umsetzung des Continuous Scatterplottings . . . . .	54
6.5	Berechnung der maximalen Dichte . . . . .	55
6.6	Umsetzung des Raycastings . . . . .	56
6.7	Die Interaktionswidgets . . . . .	58
6.7.1	Quadratisches Interaktionswidget . . . . .	60
6.7.2	Zylindrisches Interaktionswidget . . . . .	62
6.8	Labeling . . . . .	63
6.9	Weitere Interaktionen . . . . .	64
6.10	Optionen des Algorithmus . . . . .	65
6.10.1	Gemeinsame Optionen . . . . .	65
6.10.2	Optionen des Objektrenderings . . . . .	66
6.10.3	Optionen des Invariantenraumrenderings . . . . .	66
6.11	Das Intervallfenster . . . . .	67
<b>7</b>	<b>Ergebnisse</b>	<b>67</b>
7.1	Single Point Load . . . . .	68
7.2	Metallscheibe . . . . .	71
7.3	Bremshebel . . . . .	75
7.3.1	K-Invarianten . . . . .	76
7.3.2	R-Invarianten . . . . .	76
7.4	Fazit . . . . .	79
<b>8</b>	<b>Ausblick</b>	<b>79</b>

**Literatur**

|

## 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 3 Grundlagen 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-[16, S. 122] und Spannungstensoren[16, 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[17][14]:

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

**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 Skalen, 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 [17, 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 zweiten Grades im dreidimensionalen Raum entwickelt, das versucht, die genannten Herausforderungen 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. Auf eine Reihe von Direct Volume Rendering Verfahren wird in Kapitel [5 Direct Volume Rendering Verfahren](#) näher eingegangen. Kapitel [6 Umsetzung](#) beschreibt die konkrete Umsetzung der FAnToM-Erweiterung mit allen implementierten Funktionen. Nachfolgend wird in Kapitel [7 Ergebnisse](#) die entwickelte Erweiterung exemplarisch auf einige Datensätze angewendet, und die Ergebnisse diskutiert. Zum Abschluss werden in Kapitel [8 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 Tensorfelds besteht darin, die einzelnen Komponenten eines Tensor zweiten Grades als Skalarfelder aufzufassen und als Grauwertbild zu zeichnen. Dabei ist der Grauwert an einem Datenpunkt abhängig vom Verhältnis des Wertes der Komponente des Tensors zu dem höchsten Wert dieser Komponente im Datensatz. Die früheste von mir gefundene Erwähnung dieses Verfahrens stammt aus einem Paper von Kindlmann und Weinstein aus dem Jahre 1999 [21], 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[20], 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[9] bilden ein Analogon zu den Stromlinien bei Vektorfeldern. Als Eingabedaten sind nur Felder von reellen, symmetrischen, dreidimensionalen Tensoren zweiten Grades mit nichtnegativen Eigenwerten zugelassen, da so sichergestellt wird, dass die Eigenwerte ganzzahlig und größer 0 sind, sowie dass die Eigenvektoren paarweise orthogonal zueinander sind. Das Verfahren zeichnet ausgehend von festgelegten Punkten Tuben durch das Tensorfeld. Die Mittellinien dieser Tuben entsprechen dabei Stromlinien, deren Richtung vom Eigenvektor mit dem höchsten Eigenwert abhängt. Der Durchschnitt durch den Schlauch orthogonal zur 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 Stücken des Schlauchs abhängig von diesem Wert eingefärbt.

Weiterhin muss die Diplomarbeit von Clemens Fritzsch, ‘Visuelle Analyse kontinuumsmechanischer Simulationen durch kontinuierliche Streudiagramme’[14], erwähnt werden, auf der die vorliegende Arbeit direkt aufbaut. Fritzsch verwendet Methoden des Continuous Scatterplotting, um Invarianten von zweidimensionalen, symmetrischen Tensoren zweiten Grades darzustellen. Dabei beschränkt er sich jedoch auf zwei der drei Invarianten in jedem Invariantensatz, um einen zweidimensionalen Scatterplot zu erzeugen. 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 der Menge  $B$  zuordnet, lässt sich das Bild einer Menge  $A' \subset A$  als

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

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

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

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

### 3.1.2 Einschränkung einer Funktion

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

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

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

### 3.1.3 Dirac Delta

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

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

Das Dirac Delta ist formal gesehen keine Funktion. Es kann jedoch mithilfe des Lebesgue Integrals definiert werden. Dies würde hier jedoch zu weit führen. Genaueres kann in der Arbeit von Kusse et al. [24, S. 100 ff.] nachgelesen werden.

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

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

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

### 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[23]. Intuitiv entspricht das Lebesgue-Stieltjes Maß in  $\mathbb{R}$  der Länge, in  $\mathbb{R}^2$  dem Flächeninhalt und in  $\mathbb{R}^3$  dem Volumen.

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

### 3.1.5 Lineare Abbildungen

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

$$\varphi(u + v) = \varphi(u) + \varphi(v) \quad (6)$$

$$\varphi(\lambda v) = \lambda \cdot \varphi(v) \quad (7)$$

### 3.1.6 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:

$$\forall u, v, w \in V : u \oplus (v \oplus w) = (u \oplus v) \oplus w \quad (\text{Assoziativität}) \quad (8)$$

$$\exists 0_V \in V. \forall v \in V : v \oplus 0_V = 0_V \oplus v = 0_V \quad (\text{neutrales Element}) \quad (9)$$

$$\forall v_+ \in V. \exists v_- \in V : v_+ \oplus v_- = v_- \oplus v_+ = 0_V \quad (\text{inverse Elemente}) \quad (10)$$

$$\forall u, v \in V : u \oplus v \quad (\text{Kommutativität}) \quad (11)$$

und für die Skalarmultiplikation folgende Eigenschaften:

$$\forall k \in K. \forall u, v \in V : k \odot (u \oplus v) = (k \odot u) \oplus (k \odot v) \quad (\text{Distributivität}) \quad (12)$$

$$\forall k, l \in K. \forall v \in V : (k + l) \odot v = (k \odot v) \oplus (l \odot v) \quad (\text{Distributivität}) \quad (13)$$

$$\forall k, l \in K. \forall v \in V : (k \cdot l) \odot v = k \odot (l \odot v) \quad (\text{Assoziativität}) \quad (14)$$

$$\exists k_1 \in K. \forall v \in V : k_1 \odot v = v \quad (\text{Einselement}) \quad (15)$$

### 3.1.7 Dualraum

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

$$\varphi(v_1, \dots, v_n) = \varphi_1 \cdot v_1 + \dots + \varphi_n \cdot v_n = k. \quad (16)$$

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

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

### 3.1.8 Multilineare Funktionen

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

$$\varphi(\lambda \cdot v_1 + \mu \cdot v'_1, \dots, v_n) = \lambda \cdot \varphi(v_1, \dots, v_n) + \mu \cdot \varphi(v'_1, v_2, \dots, v_n) \quad (17)$$

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

### 3.1.9 Tensor

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

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

$d$  in jeder möglichen Kombination eingesetzt werden, ergeben sich für  $1 \leq i, j \leq d$  folgende Komponenten:

$$c_{i,j} = \sum_{i=1}^d \sum_{j=1}^d T(e_i, e_j), \quad (18)$$

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  $v$

$$m \cdot v = u \quad (19)$$

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.

### 3.1.10 Jacobi Matrix

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

$$J_f(a) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(a) & \dots & \frac{\partial f_1}{\partial x_n}(a) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(a) & \dots & \frac{\partial f_m}{\partial x_n}(a) \end{pmatrix} \quad (20)$$

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

### 3.1.11 Determinante einer Matrix

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

$$\det(m) = \sum_{\sigma \in S_n} \left( sgn(\sigma) \prod_{i=1}^n m_{i,\sigma(i)} \right) \quad (21)$$

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

### 3.1.12 Determinante der Jacobimatrix

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

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

$$\|f(p_1), f(p_2)\|_n = det(J_f) \cdot \|p_1, p_2\|_n \quad (22)$$

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

### 3.1.13 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  $rang(M)$  der Matrix  $M$  bezeichnet werden.

### 3.1.14 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

$$tr(A) = \sum_{i=1}^n a_{ii} \quad (23)$$

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 [3.1.22](#)).

### 3.1.15 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:

$$f(kv) = |k|f(v) \quad (\text{Absolute Homogenität}) \quad (24)$$

$$f(u + v) \leq f(u) + f(v) \quad (\text{Erfüllung der Dreiecksungleichung}) \quad (25)$$

$$f(v) = 0 \iff v = 0 \text{ ist der Nullvektor} \quad (\text{Definitheit}) \quad (26)$$

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

$$\text{norm}(A) = \sqrt{\text{tr}(AA^T)} \quad (27)$$

Die euklidische Norm wird auch als ‘Frobeniusnorm’ bezeichnet.

### 3.1.16 Deviator einer Matrix

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

$$\tilde{A} = A - \bar{A} \quad (28)$$

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

$$\bar{A} = \frac{1}{3}\text{tr}(A)I \quad (29)$$

wobei  $I$  die Matrixdarstellung des Einheitstensors ist, also

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (30)$$

### 3.1.17 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

$$M \cdot v_i = \lambda_i v_i. \quad (31)$$

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

### 3.1.18 Fraktionale Anisotropie einer Matrix

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

$$\text{FA}(M) = \sqrt{\frac{3((\lambda_1 - \bar{\lambda})^2 + (\lambda_2 - \bar{\lambda})^2 + (\lambda_3 - \bar{\lambda})^2)}{2(\lambda_1^2 + \lambda_2^2 + \lambda_3^2)}}. \quad (32)$$

Sie entspricht also der Standardabweichung der Eigenwerte, dividiert durch den Mittelwert ihrer Quadrate. Dadurch wird die Standardabweichung auf das Intervall  $[0; 1]$  normiert. Bei Matrizen mit hoher FA (nahe 1) ist ein Eigenwert um ein Vielfaches größer als die anderen beiden. Hohe FA tritt in der Kontinuumsmechanik zum Beispiel in Bereichen auf, in denen das Material in die Richtung eines Eigenvektors auseinandergespannt wird, während es in Richtung der anderen Eigenvektoren gleich bleibt oder sogar schrumpft. Eine niedrige FA dagegen drückt aus, dass die Eigenwerte etwa die gleichen Werte annehmen. In der Kontinuumsmechanik ist dies beispielsweise in Bereichen isotroper Verformung, also gleichmäßiger Ausdehnung / gleichmäßigem Schrumpfen in alle Richtungen, der Fall.

### 3.1.19 Modus einer Matrix

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

$$\text{mode}(A) = 3\sqrt{6} \det(A \setminus \text{norm}(A)). \quad (33)$$

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

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

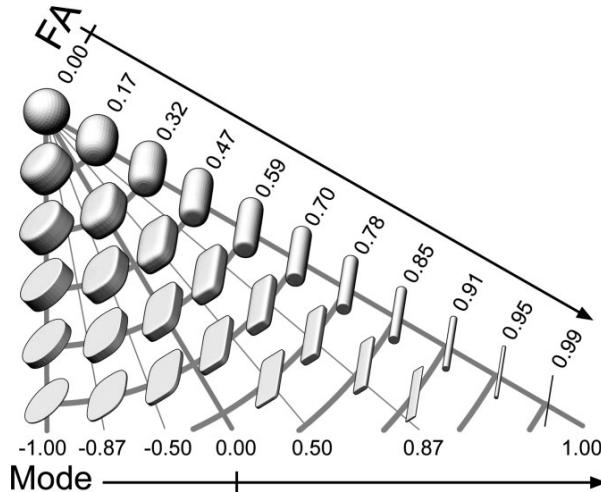


Abbildung 1: Darstellung der fraktionalen Anisotropie und des Modus von Matrizen in Form von Superquadrics[20]. Die fraktionale Anisotropie nimmt mit größerer werdender Entfernung zum obersten linken Superquadric zu. Der Modus des Deviators wird abhängig vom Winkel dargestellt, wobei er links -1 beträgt und rechts 1. Entnommen aus [11, S. 140].

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

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

### 3.1.20 Gradient

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

$$\text{grad}(f) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \quad (34)$$

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

$$grad(g) = \begin{pmatrix} \frac{\partial f}{\partial a_{11}} & \cdots & \frac{\partial f}{\partial a_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial a_{m1}} & \cdots & \frac{\partial f}{\partial a_{mn}} \end{pmatrix} \quad (35)$$

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

### 3.1.21 Orthogonalität von Matrizen

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

$$tr(U, V^T) = 0 \quad (36)$$

### 3.1.22 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[11]. Eine Invariante ist somit eine Funktion  $\Psi : M \rightarrow A$  die Objekten aus dem Vektorraum aller Matrizen  $M$  Objekte aus der Menge  $A$  zuordnet. In der Praxis wird für  $A$  meistens  $\mathbb{R}$  gewählt.

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

$$\Psi : \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \rightarrow \mathbb{R} \quad (37)$$

darstellbar, wobei  $\lambda_1, \lambda_2, \lambda_3$  die Eigenwerte von  $T$  sind. Dies stimmt mit der Betrachtungsweise von Zobel und Scheuermann [38] überein.

### 3.1.23 Invariantensätze

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

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

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

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

$$\begin{aligned}\chi_A(a) &= \det(a \cdot I - A) \\ \chi_A(a) &= -a^3 + I_1 \cdot a^2 - I_2 \cdot a + I_3,\end{aligned}\tag{38}$$

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

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

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

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

Während für  $I_1$  in der Mechanik eine Interpretation als Maß für den isotropen Anteil des Tensors existiert, fehlen eindeutige Interpretationen für  $I_2$  und  $I_3$ . Es gibt zwar einen

Zusammenhang zwischen  $I_2$  und dem deviatorischen Anteil des Tensors, dieser ist jedoch nicht eindeutig genug um aus hohem  $I_2$  auf hohe Anisotropie schließen zu können. Für  $I_3$  existiert in der Mechanik keine verbreitete Interpretation.

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

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

Dabei ist jedoch zu beachten, dass

$$\begin{aligned}
 \text{tr}(\tilde{A}) &= \text{tr}(A - \frac{1}{3}\text{tr}(A)I) \\
 &= a_{11} - \frac{1}{3}\text{tr}(A) + a_{22} - \frac{1}{3}\text{tr}(A) + a_{33} - \frac{1}{3}\text{tr}(A) \\
 &= a_{11} + a_{22} + a_{33} - \text{tr}(A) \\
 &= \text{tr}(A) - \text{tr}(A) \\
 &= 0
 \end{aligned} \tag{39}$$

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

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

- $K_1(A) = \text{tr}(A)$  (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 Abb. 2 dargestellt.  $K_1$  entspricht dabei der zylindrischen  $x$ -Koordinate,  $K_2$  dem Radius  $r$  und  $K_3$  dem Winkel  $\theta$ .

Ein Vorteil des K-Invariantensatzes ist die relativ leichte Interpretierbarkeit in der Kontinuumsmechanik.  $K_1$  kann als Maß der absoluten ‘Größe’ der Matrix ,  $K_2$  als

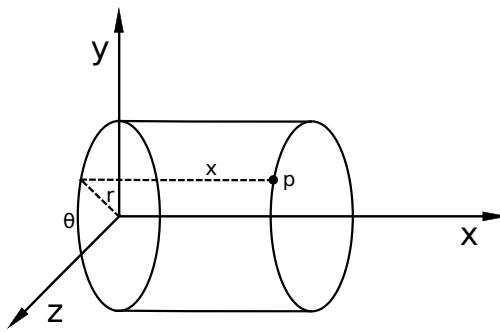


Abbildung 2: Darstellung eines zylindrischen Koordinatensystems.  $x, y$  und  $z$  entsprechen dabei den kartesischen Achsen. Die Koordinaten des Punktes  $p$  sind angegeben als  $x, r, \theta$ .  $x$  entspricht der kartesischen Koordinate,  $r$  ist die Entfernung zwischen der Projektion von  $p$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

Maß des anisotropen Anteils angesehen werden [22]. Dagegen kann aus dem Wert von  $K_3$  die Verteilung der Eigenwerte und damit die Form der Anisotropie des Tensors geschlussfolgert werden.

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

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

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

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

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

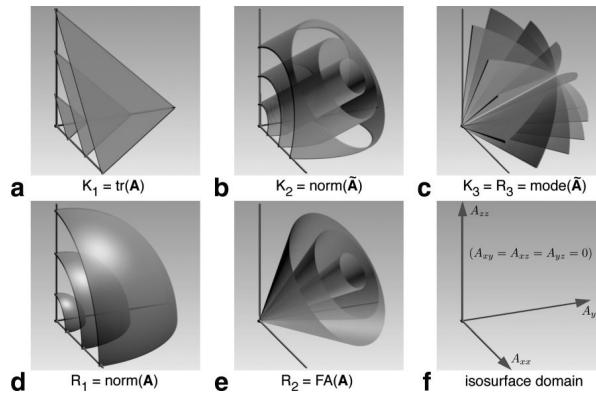


Abbildung 3: Dargestellt sind Isoflächen der K- und R-Invarianten von diagonalisierten  $3 \times 3$  Matrizen (alle Werte ausserhalb der Hauptdiagonale sind 0). Die Koordinatenachsen entsprechen den drei Eigenwerten der Matrizen. Entnommen aus [11], S. 139]

### 3.1.24 Faltung

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

$$(f *_c g)(x) = \int_{\mathbb{R}^n} f(\tau)g(x - \tau)d\tau \quad (40)$$

Um den Rechen- und Speicheraufwand zu reduzieren, wird oft die diskrete Faltung  $*_d$  verwendet. Diese ist für den diskreten Definitionsbereich  $\mathbb{D} \subset \mathbb{Z}$  und Funktionen  $f, g : \mathbb{D} \rightarrow \mathbb{C}$  definiert als

$$(f *_d g)(n) = \sum_{k \in \mathbb{D}} f(k)g(n - k) \quad (41)$$

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

### 3.1.25 Gaußsche Glockenkurve

Glockenkurven sind Funktionen der Form

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (42)$$

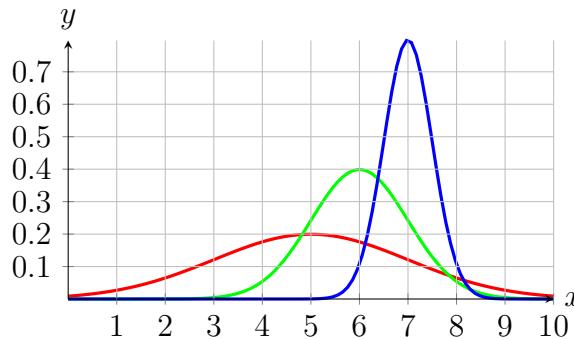


Abbildung 4: Darstellung von drei Gaußschen Glockenkurven mit unterschiedlichen  $\mu$  und  $\sigma$ : Rot: $\mu = 5, \sigma = 2$ , Grün: $\mu = 6, \sigma = 1$ , Blau: $\mu = 7, \sigma = 0.5$

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

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

Im zweidimensionalen Fall existiert analog eine Glockenkurve mit der Formel

$$f(x, y) = \frac{1}{\sqrt{2\pi}\sigma_x^2} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}} \cdot \frac{1}{\sqrt{2\pi}\sigma_y^2} e^{-\frac{(y-\mu_y)^2}{2\sigma_y^2}} \quad (43)$$

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

## 3.2 Mechanische Grundlagen

### 3.2.1 Physikalisches Feld

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

Wenn Felder nicht in Form einer kontinuierlichen Funktion beschrieben werden können, 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  $\sigma$  bilden:

$$\sigma = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} \quad (44)$$

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

### 3.2.3 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:

$$\epsilon = \begin{bmatrix} \epsilon_x & \frac{1}{2}\gamma_{xy} & \frac{1}{2}\gamma_{xz} \\ \frac{1}{2}\gamma_{xy} & \epsilon_y & \frac{1}{2}\gamma_{yz} \\ \frac{1}{2}\gamma_{xz} & \frac{1}{2}\gamma_{yz} & \epsilon_z \end{bmatrix} \quad (45)$$

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

## 4 Verwendete Verfahren und Technologien

### 4.1 FAnToM

FAnToM[1][37] ('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[18] 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 Primitive, 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 Continuous Scatterplotting

### 4.4.1 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 ein-, zwei- oder dreidimensionales Koordinatensystem eingetragen, dessen Koordinatenachsen Attributwerten entsprechen. Beispiele für Scatterplottings sind in Abb. 5 abgebildet. Das Bild von  $\tau$  ist auf maximal drei Dimensionen beschränkt, da höherdimensionale Daten visuell nur schwer zu interpretieren sind.

Ein Vorteil von Scatterplottings ist die leichte Erkennbarkeit von Strukturen in den Datensätzen: Cluster, Outlier und funktionale Zusammenhänge zwischen den verwendeten Attributen lassen sich gut identifizieren.

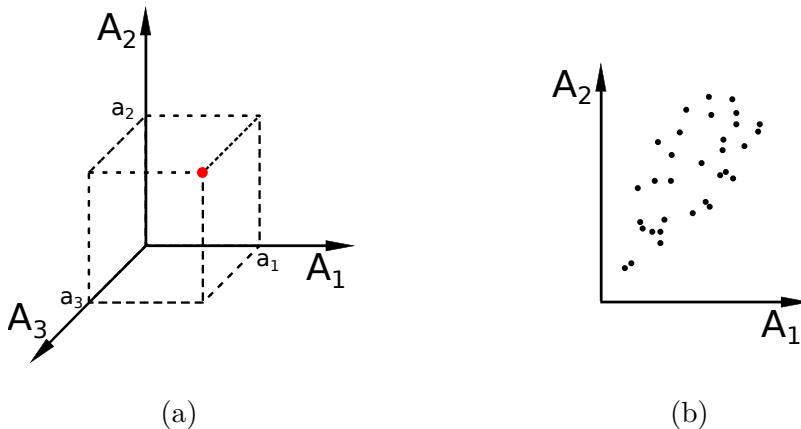


Abbildung 5: Zwei Beispiele für Scatterplottings. In (a) wird ein Objekt mit den Attributwerten  $a_1, a_2, a_3$  als roter Punkt in einem dreidimensionalen Scatterplot dargestellt. Um die Attribute besser ablesen zu können, wurden gestrichelte Linien eingezeichnet. In (b) wurden mehrere Objekte mit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

#### 4.4.2 Erklärung des Continuous Scatterplottings

Das Continuous Scatterplotting stellt eine Erweiterung des Scatterplottings dar, sodass statt nur einer endlichen Anzahl von Objekten auch kontinuierliche Mengen verarbeitet werden können. Die von Bachthaler und Weiskopf[3] vorgestellte Definition wurde von Fritzsche[14] 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  $\tau$ , Dichtewerte zuordnet.  $\sigma$  kann als Funktion aufgefasst werden, die jedem Pixel des Scatterplots einen skalaren Wert zuordnet, der z.B. als Farbe oder Opazität dargestellt werden kann. In Abb. 6 ist der Effekt, den  $\tau$  auf die Dichtefunktion hat, dargestellt. Um  $\sigma$  berechnen zu können, werden zwei Annahmen getroffen:

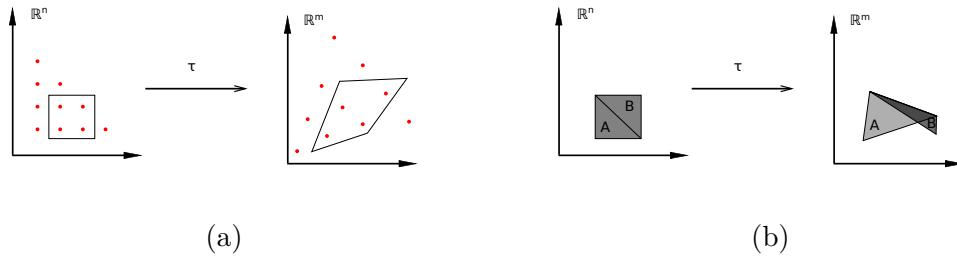


Abbildung 6: Darstellung der DichteVerteilung abhängig von  $\tau$ . (a) Die roten Punkte deuten die Dichte in den beiden Scatterplots an, die zwei Vierecke sind Teilmengen des Urbilds, die durch  $\tau$  verformt werden. Zu erkennen ist, dass die Streckung des Vierecks zur Erhöhung der Entfernung zwischen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

Die erste Annahme ist, dass eine Dichtefunktion  $s : \mathbb{R}^n \rightarrow \mathbb{R}$  existiert, die jedem Punkt des Urbilds von  $\tau$  einen skalaren Wert, die Dichte an diesem Punkt, zuordnet. Der Einfachheit halber wird im Folgenden eine uniforme Dichte von  $s(x) = 1$  angenommen. Aus der Dichte an den Punkten lässt sich die Gesamtdichte  $M$ , auch bezeichnet als Masse, einer Teilmenge  $V \subset \mathbb{R}^n$  berechnen:

$$M = \int_V s(x) d^n x \quad (46)$$

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

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

$$\int_{\phi} \sigma(\xi) d^m \xi = \int_{\tau^{-1}(\phi)} s(x) d^n x \quad (47)$$

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

$$\int_V s(x) d^n x = \int_{\phi=\tau(V)} \sigma(\xi) d^m \xi \quad (48)$$

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

$$\int_{\phi} \sigma_v(\xi) d^m \xi = \int_{\tau|_V^{-1}(\phi)} s(x) d^n x \quad (49)$$

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

$$\begin{aligned}
 \int_{\phi} \sigma(\xi) d^m \xi &= \int_{\bigcup_{V_i} \tau|_{V_i}^{-1}(\phi)} s(x) d^n x \\
 &= \sum_{V_i} \int_{\tau|_{V_i}^{-1}(\phi)} s(x) d^n x \\
 &= \sum_{V_i} \int_{\phi} \sigma_{V_i}(\xi) d^m \xi \\
 &= \int_{\phi} \sum_{V_i} \sigma_{V_i}(\xi) d^m \xi
 \end{aligned} \tag{50}$$

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

$$\sigma(\xi) = \sum_{V_i} \sigma_{V_i}(\xi) \tag{51}$$

Eine intuitive Interpretation von Formel 51 ist, dass die gesamte Dichtefunktion von  $\tau$  als Summe der Dichtefunktionen der Einschränkungen von  $\tau$  auf die  $V_i$  gebildet werden kann.

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

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

$$\int_{\tau(V)} \sigma(\xi) d^{m=n} \xi = \int_V \sigma(\tau(x)) |\det(J_\tau)(x)| d^n x = \int_V s(x) d^n x \tag{52}$$

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

$$\sigma(\xi) = \frac{s(\tau^{-1}(\xi))}{|\det(J_\tau)(\tau^{-1}(\xi))|} \tag{53}$$

Wie schon in Kapitel 3 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:  $\tau$  ist differenzierbar und konstant über  $V$ ,  $V$  ist keine Nullmenge** Wenn  $\det(J_\tau) = 0$ , dann ist  $\tau$  kein Diffeomorphismus und  $\sigma$  kann nicht so wie in Fall 1 definiert werden. Das ist z.B. der Fall, wenn  $\tau$  Bereiche mit konstanten Werten enthält und führt dazu, dass eine nicht leere Teilmenge von  $\mathbb{R}^n$  auf einen einzelnen Punkt  $\xi$  abgebildet wird. Eine Möglichkeit die unendlich hohe Dichte an  $\xi$  darzustellen und dennoch die Gesamtmasse nicht zu verändern ist es, bei  $\xi$  ein Dirac-Delta einzufügen, das mit dem Volumen aller konstanten Bereiche  $V_i \in V$ , die nach  $\xi$  abgebildet werden, skaliert wird:

$$\begin{aligned}\sigma(\xi) &= \sum_{V_i \in V} \delta(\xi - \tau(V_i)) \cdot \int_{V_i} s(x) d^n x \\ &= \sum_{V_i \in V} \delta(\xi - \tau(V_i)) \cdot \text{Vol}(V_i) \quad (\text{da } s(x) = 1)\end{aligned}\tag{54}$$

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

$$\begin{aligned}\int_{\phi} \sigma(\xi) d^n \xi &= \sum_{V_i} \int_{\phi} \delta(\xi - \tau(V_i)) \cdot \text{Vol}(V_i) d^n \xi \\ &= \sum_{\forall V_i : \tau(V_i) \in \phi} \text{Vol}(V_i) \\ &= \int_{\tau^{-1}(\phi)} 1 d^n x\end{aligned}\tag{55}$$

Da  $s(x) = 1$  festgelegt ist, ist die Bedingung erfüllt. In 6b ist ein Beispieldfall dafür angegeben. Die Dichte im überlappenden Bereich der beiden Dreiecke ist gleich der Summe der Dichten der einzelnen Dreiecke. Bei anderen Dichtefunktionen muss Formel 54 entsprechend angepasst werden.

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

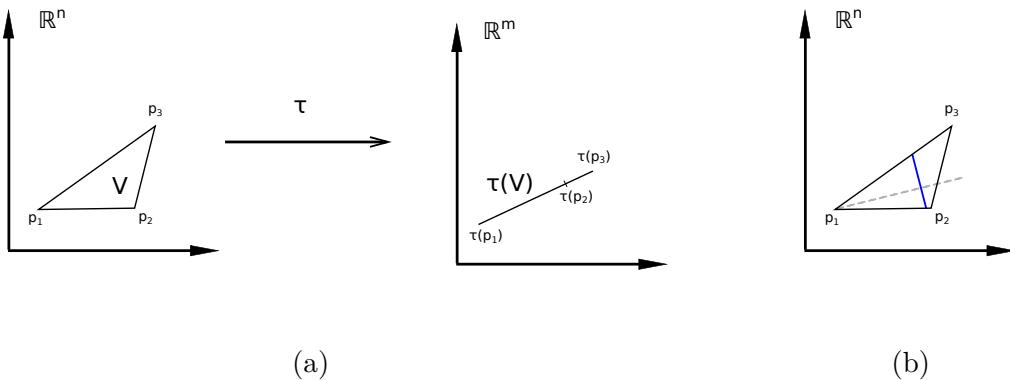


Abbildung 7: Eine Darstellung des fünften Falls. (a) Alle drei Punkte des Dreiecks  $V$  im linken Koordinatensystem werden auf eine Gerade abgebildet. Dadurch ist  $\tau(V)$  eine Nullmenge. (b) Eine Darstellung des von Fritzs vorgeschlagenen Ansatzes. Die graue, gestrichelte Linie entspricht dem Gradienten,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

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

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

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

Die Jacobimatrix enthält in diesem Fall mindestens eine Nullzeile, wodurch für ihren Rang gilt  $0 < \text{rang}(M) < m$ .

Fritzsch schlägt zur Lösung dieses Problems zwei Ansätze vor.

Der erste orientiert sich an Fall 2 und ordnet dem Bildbereich sehr hohe Dichten zu, um die Masseerhaltung zu gewährleisten.

Der zweite Ansatz ist nur für den Fall  $m = 2, n = 2$  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 7b zu sehen. Die Dichte an den Punkten von  $\tau(V)$  wird dann proportional zur Masse gewählt.

## 4.5 Volumetrische Daten

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

Das Ergebnis eines Continuos Scatterplottings der Form  $m = n = 3$  beispielsweise stellt einen volumetrischen Datensatz dar, in dem Punkten ein Dichtewert zugeordnet wird. Da die Visualisierung genau diese Art von Datensätzen die zentrale Aufgabe der vorliegenden Arbeit ist, ist die Abspeicherung und das Auslesen von volumetrischen Daten ein wichtiges Teilproblem.

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

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

Voxel haben einen entscheidenden Vorteil gegenüber anderen volumetrischen Datensätzen: Die Komplexität der Daten, und daraus folgend die Rechenzeit der meisten Algorithmen, die Voxel als Eingabe verwenden, ist nicht abhängig von der Anzahl und Komplexität der in der ursprünglichen Szene dargestellten Objekte, sondern nur noch von der Anzahl der Voxel, auf die die Szene abgebildet wurde. Durch Anpassung der Anzahl der Voxel ist es möglich, Rechenzeit und Qualität der Ergebnisse von Algorithmen gegeneinander abzuwägen.

In der Praxis werden Voxel in unterschiedlichen Formen verwendet. Die populärste Variante sind Datensätze, die ein quaderförmiges Volumen in gleichgroße quaderförmige

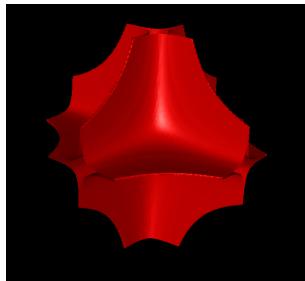


Abbildung 8: Darstellung einer Isosfläche in einem automatisch generierten Datensatz. Erzeugt mithilfe von FAnToM [1].

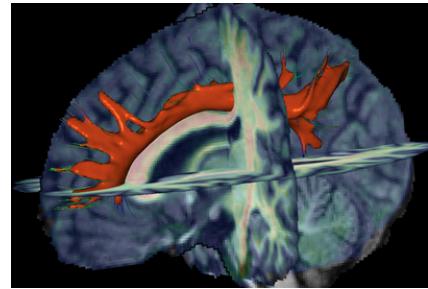


Abbildung 9: Darstellung von drei Slicings durch einen volumetrischen MRI Datensatz. In Orange ist zusätzlich eine Isosfläche eingezeichnet. Entnommen aus [32].

Voxel zerlegen. Aber auch Datensätze mit tetraedrischen Voxeln oder solche, deren Voxel unterschiedliche Formen annehmen, existieren.

## 4.6 Volume Visualization

In der Computergrafik werden dreidimensionale Objekte üblicherweise durch Dreiecke dargestellt, durch die die Oberflächen der Objekte approximiert werden. Dieses Vorgehen entfernt jedoch alle Informationen über Strukturen im Inneren der Objekte, wie beispielsweise Bereiche gleicher Materialien oder Dichte. Um Continuous Scatterplots der Form  $m = n = 3$  visualisieren zu können ohne Informationen über innere Strukturen zu verlieren werden deshalb besondere Verfahren zur Darstellung volumetrischer Daten benötigt. 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 [15, 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.

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

#### 4.6.1 Isoflächen

Volumetrische Datensätze beinhalten oft komplexe Strukturen. Die Position und Form sowie die Datenwerte innerhalb dieser Strukturen gut zu vermitteln stellt oft ein zentrales Ziel der Volume Visualization dar. Eine Möglichkeit diese Informationen zu vermitteln stellt die Berechnung und Darstellung von Isoflächen dar. Die Isofläche einer Funktion  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$  und eines Isowerts  $i \in \mathbb{R}$  ist definiert als eine Oberfläche, die den  $\mathbb{R}^3$  in zwei Regionen  $R_0, R_1$  teilt, wobei für jeden Punkt  $r_0 \in R_0$  gilt  $f(r_0) < i$  und für jeden Punkt  $r_1 \in R_1$  gilt  $f(r_1) > i$  [15, S. 7]. Die Isofläche muss dabei nicht zusammenhängend sein. Auf der Isofläche selbst ist  $f$  gleich dem Isowert.

In Abb. 8 ist ein Beispiel für eine Isofläche 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 dieses 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.

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

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

#### 4.6.2 Slicing

Ein einfacher Ansatz um einen volumetrischen Datensatz zu visualisieren ist das sogenannte “Slicing”[28]. 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 Abtastpunkte, berechnet. Das Festlegen und Auswerten von Abtastpunkten wird im Folgenden als “Abtastung” bezeichnet. Um aus den Abtastpunkten die Färbungen der Dreiecke zu berechnen, gibt es zwei Verfahren, bezeichnet als Präklassifizierung und Postklassifizierung.

Bei der Präklassifizierung wird zunächst die Farbe an den Abtastpunkten bestimmt. Dazu wird eine sogenannte “Transferfunktion”  $T$ , verwendet die einem Punkt anhand seiner Datenwerte eine Farbe zuordnet. Ein Beispiel für eine Transferfunktion ist in Abb. 10 dargestellt. Die horizontale Achse entspricht den skalaren Werten zwischen Minimum und Maximum im Datensatz sowie den zugeordneten Farben. Die vertikale Achse entspricht der Opazität, von vollständig transparent am unteren Rand bis zu opak am oberen Rand. Durch Position und Farbe der Dreiecke am oberen Rand der Transferfunktion kann die Farbzordnung festgelegt werden. Die Quadrate stellen interaktiv ausgewählte Punkte dar, die Zuordnungen von Werten im Skalarfeld zu Farb- und Opazitätswerten entsprechen, zwischen denen entlang der verbindenden Linien interpoliert wird. So wird allen Werten im Feld eine Farbe und eine Opazität zugeordnet. Dabei ist zu beachten, dass Opazität im allgemeinen Fall bei Slicing nicht verwendet wird. Um die Farben an allen anderen Punkten der Schnittfläche zu bestimmen, werden die Farben der Abtastpunkte interpoliert. Ein Nachteil der Präklassifizierung liegt darin, dass durch die Interpolation Farben entstehen können, die nicht im Bild der Transferfunktion enthalten sind. Ein Beispiel: Sei  $f$  eine Funktion, die Punkten  $p \in \mathbb{R}^3$  einen skalaren Wert  $s \in \mathbb{R}$  zuordnet, also  $f(p) = s$ . Sei  $V$  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

$$f(x) = \begin{cases} (1.0, 0.0, 0.0, 1.0) \text{ (Rot), wenn } x \leq 0 \\ (0.0, 1.0, 0.0, 1.0) \text{ (Grün), sonst} \end{cases} \quad (56)$$

Wenn nun drei Punkte  $p_0, p_1, p_2$  existieren, von denen  $p_0$  und  $p_2$  Abtastpunkte 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 Abb. 9 sind drei Slicings zusammen mit einer orangenen Isofläche abgebildet.

Indem Interaktionen angeboten werden, durch die der Benutzer Position und Orientierung der Schnittfläche interaktiv anpassen kann, bietet Slicing eine gute Möglichkeit um die Verteilung von Werten innerhalb eines volumetrischen Datensatzes zu analysieren und Werte 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. 9).

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

#### **4.6.3 Direct Volume Rendering**

Die bisher vorgestellten 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

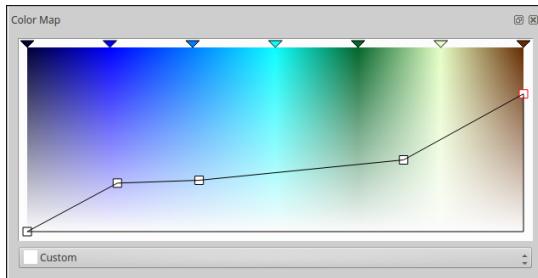


Abbildung 10: Eine Implementierung einer interaktiven Transferfunktion in FAnToM.

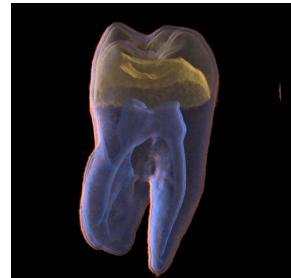


Abbildung 11: Ein Direct Volume Rendering eines Zahns. Entnommen aus [10, S. 6]

Verständnis des Datensatzes entwickeln kann. Dies kann, wiederum abhängig von Größe und Komplexität, fast unmöglich sein.

Der Begriff “Direct Volume Rendering” (kurz DVR) 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. Ein großer Teil der Informationen in diesem Abschnitt stammt aus Buch “The Visualization Handbook”[15], speziell aus den Kapiteln “Overview of Volume Rendering” (S. 127-164) von Kaufman und Mueller, “Volume Rendering Using Splatting” (S. 175-188) von Crawfis, Xue und Sang, “Pre-Integrated Volume Rendering” (S. 211-228) von Kraus und Ertl sowie “Hardware Accelerated Volume Rendering” (S. 229-258) von Pfister.

Ähnlich wie beim Slicing wird eine Transferfunktion verwendet, um den Voxeln eine Farbe zuzuordnen. Zusätzlich legt die Transferfunktion auch noch einen Wert im Intervall  $[0; 1]$  als Opazität fest, die ausdrückt, wie stark ein Voxel hinter ihm gerenderte Voxel überdeckt (siehe Abb. 10 und die Beschreibung im Abschnitt 4.6.2). Eine Opazität von 1 bedeutet dabei, dass das Voxel komplett opak ist, eine Opazität von 0 dagegen dass das Voxel vollständig durchsichtig und damit in der Visualisierung unsichtbar ist. Ein Beispiel für ein durch Direct Volume Rendering erzeugtes Bild zeigt Abb. 11.

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

Allen image-order Verfahren ist gemein, dass sie eine Position im Raum als “Kamera-position” festlegen, die Position, von der aus der Benutzer den Datensatz betrachtet. Diese Position befindet sich dabei im gleichen Koordinatensystem wie die Voxel selbst. Vor der Kamera, orthogonal zur Blickrichtung, wird eine imaginäre Ebene platziert, die sogenannte Bildebene (engl. “image plane”). Ihre Größe und Position ist so gewählt, dass die das gesamte Blickfeld der Kamera einnimmt (siehe Abb. 12). Indem Positionen der Bildebene auf Pixel des Bildschirms abgebildet werden, ist es möglich, durch Einfärbung

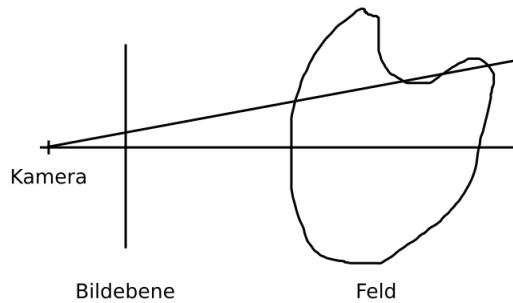


Abbildung 12: Schematische Darstellung der image-order Verfahren. Farbwerte an Positionen auf der Bildebene werden mittels Abtastung entlang von Strahlen, die an der Kameraposition beginnen und die Position auf der Bildebene sowie den Datensatz schneiden, bestimmt.

der Bildebene eine Visualisierung zu erzeugen. Die Farbe eines Pixels ergibt sich dabei, indem ein Strahl berechnet wird, der sowohl durch die Position des Mittelpunkts des Pixels auf der Bildebene als auch durch die Kameraposition verläuft. Entlang des Strahls werden Abtastpunkte gewählt, an denen die Transferfunktion ausgewertet wird. Genau wie beim Slicing können dabei Prä- und Postklassifizierung verwendet werden. Abhängig davon, wie die Farb- und Opazitätswerte der Abtastpunkte kombiniert werden, können unterschiedliche Darstellungen entstehen. Eine Reihe davon wird später im Text vorgestellt. Da die Kombination von Farb- und Opazitätswerten in OpenGL fast immer durch die Funktion des Blendings implementiert wird, verwenden ich den Begriff ‘Blending’ im Folgenden synonym dazu. Bei manchen Varianten ist es möglich, die Berechnung der Farbe eines Pixels zu frühzeitig zu beenden, wenn die berechneten Farb- und Opazitätswerte bestimmte Werte annehmen. In diesem Fall wird davon ausgegangen, dass der Einfluss der noch zu berechnenden Abtastpunkte zu klein ist, um vom Benutzer noch wahrgenommen zu werden. Diese frühzeitige Terminierung ist einer der wichtigsten Vorteile der image-order Verfahren.

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

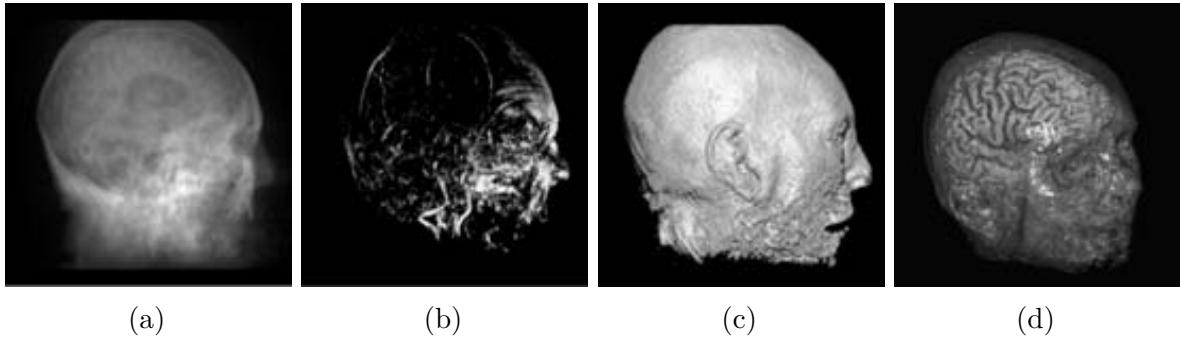


Abbildung 13: Arten von DVR Visualisierungen: (a) Röntgenbild, (b) MIP,  
(c) Isofläche, (d) Transluzent. Entnommen aus [15, S. 134]

Sortierung muss jedes mal aktualisiert werden, wenn die Kamera bewegt wird.

Hybride Verfahren versuchen Vorteile von image-order (frühzeitige Terminierung) und object-order Verfahren (geringer Speicherverbrauch, Rechenzeit abhängig von Anzahl Voxeln) zu kombinieren. Die Details der Implementierung unterscheiden sich jedoch stark zwischen den einzelnen Verfahren.

Zuletzt gibt es noch die Domain-Verfahren. Dabei wird der komplette Datensatz so transformiert, dass die DVR Visualisierung mittels mathematischer Eigenschaften des Wertebereichs (englisch “Domain”) erzeugt werden kann. Ein Beispiel dafür ist, den Datensatz zunächst mittels der Fourier Transformation in den Frequenzraum zu überführen. Indem im Frequenzraum ein bestimmter 2D Ausschnitt gewählt und zurück in den Ortsraum transformiert wird, kann eine Projektion des ursprünglichen Datensatzes erzeugt werden. Ein großer Vorteil von Domain-Verfahren liegt häufig in der sehr viel kürzeren Rechenzeit. Bei der bereits erwähnten Variante im Frequenzraum beispielsweise hat eine Komplexität von  $O(N^2 \log(N))$  statt den bei DVR Verfahren üblichen. Leider schränkt dieses Verfahren die möglichen erzeugbaren Visualisierungen auf ein simples Integral entlang der Sichtstrahlen ein  $O(N^3)$  [15, S. 143]. Es existieren Ergebnisse, die versuchen andere Visualisierungen zu approximieren, jedoch stellt dies eine deutliche Einschränkung des DVR dar, weshalb im Folgenden Domain-Verfahren nicht weiter beleuchtet werden.

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

**Röntgenbilder** Die Bezeichnung “Röntgenbilder” stammt aus der Ähnlichkeit der entstehenden Bilder zu denen, die beim klassischen Röntgen in der bildgebenden Medizin erzeugt werden. Die Färbung eines Pixels ergibt sich in diesem Verfahren dadurch, dass die ermittelten Farb- und Transparenzwerte aller zur Färbung des Pixels beitragenden Voxel aufaddiert werden. Damit sind die Voxel gemeint, die zum Beispiel vom selben

Strahl durch den Mittelpunkt des Pixels geschnitten (image-order) werden oder deren Basisfunktionen sich in diesem Pixel überschneiden (object-order). Das Blending entspricht in diesem Fall einer Addition, die die Farbe eines Voxels auf die bisher bestimmte Farbe eines Pixels aufaddiert. Ein Beispiel für ein so erzeugtes Bild ist in Abb. 13a zu sehen.

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

**Isofläche** Wenn  $T_O$ , der Teil der Transferfunktion  $T$  der skalare Werte  $s$  auf eine Opazität abbildet, definiert ist als

$$T_O(s) = \begin{cases} 1 & \text{wenn } s = x \text{ für genau ein } x \in \mathbb{R} \\ 0 & \text{sonst} \end{cases} \quad (57)$$

so entsprechen die erzeugten Bilder Isoflächen. Da Isoflächen komplett opak sind, wird die Berechnung abgebrochen, sobald für einen Pixel der zur Kamera nähste Voxel, der Teil einer Isofläche ist, gefunden wurde. Blending entspricht hier einer Minimumsfunktion, die anstelle der Farbwerte die Distanz zur Kamera zur Entscheidung verwendet. Abhängig vom Verfahren kann auch sofort terminiert werden wenn ein opakes Voxel gefunden wurde, dies erschwert jedoch das spätere Hinzufügen von z.B. Schatten. Durch geschickte Wahl der Reihenfolge, in der die Voxel verarbeitet werden (z.B. mit zunehmender Entfernung zur Kamera), kann weitere Beschleunigung erreicht werden. In Abb. 13c ist eine so erzeugte Isofläche dargestellt.

**Transluzent** Transluzente Bilder entstehen, wenn der für die Opazität zuständige Teil der Transferfunktion anstelle der Extremwerte 0 und 1 auch dazwischenliegende Werte annehmen kann. Voxel, für die die Transferfunktion dies tut, werden als teilweise transparent dargestellt. Sie stellen somit eine Verallgemeinerung des Verfahrens zur Erzeugung von Isoflächen dar. Dazu muss jedoch ein Verfahren verwendet werden, das Farben und Opazitäten korrekt kombiniert, sodass transparente Voxel die Sicht auf dahinterliegende Strukturen freigeben. Die Formulierung des Blendings hängt hierbei von der Verarbeitungsreihenfolge der Voxel ab. Im Paragraphen zu Raycasting wird dies genauer erläutert. Transluzente Visualisierungen sind besonders populär, da sie sowohl hohe Flexibilität (durch Wahl der Transferfunktion) aufweisen als auch den Grundgedanken des DVR, das gleichzeitige und vollständige Darstellen des Datensatzes,

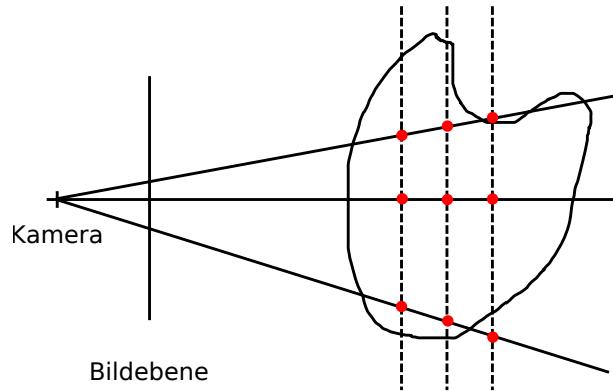


Abbildung 14: Schematische Darstellung des Texture Slicings. Die gestrichelten Linien entsprechen Schnittbildern durch den Datensatz. Die roten Punkte stellen die Positionen auf den Schnittbildern dar, die auf das Pixel des jeweiligen Strahls projiziert werden.

am besten umsetzen. Der Rest der Arbeit beschäftigt sich deshalb ausschließlich mit Verfahren zur Erzeugung von transluzenten Visualisierungen. Ein Beispiel für ein durch Transluzentes DVR erzeugtes Bild ist in 13d zu sehen.

## 5 Direct Volume Rendering Verfahren

Es existieren eine Reihe von DVR Verfahren, die die in Abschnitt 4.6.3 erläuterten Konzepte umsetzen. Diese Verfahren bieten dabei individuelle Vor- und Nachteile, die einen Vergleich auf Grundlage der von der Anwendung gestellten Anforderungen nötig macht. Dazu werden in diesem Kapitel einige der wichtigsten DVR Verfahren vorgestellt.

### 5.1 Texture Slicing

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

Beim Texture Slicing werden mehrere imaginäre Ebenen parallel zur Bildebene durch den Datensatz gelegt. Die Schnittmenge dieser Ebenen mit dem Datensatz bilden Polygone. Indem Abtastpunkte auf diesen Polygonen ausgewertet werden, lassen sich durch Anwendung der Transferfunktion die Polygone einfärben. Das Verfahren dazu ist praktisch identisch zu dem in Abschnitt 4.6.2 beschriebenen Slicing. Die entstandenen Schnittbilder

werden danach auf die Bildebene projiziert und die Farbwerte der Pixel durch Blending der berechnet. Die für das Blending verwendete Formel hängt von der Reihenfolge ab, in der die Schnitte miteinander kombiniert werden. Wenn die Schnitte mit zunehmender Entfernung zur Kamera (front-to-back) verarbeitet werden, so entspricht das Blending den Formeln

$$c = c_{alt} + c_{neu} \cdot (1 - \alpha_{alt}) \cdot \alpha_{neu} \quad (58)$$

$$\alpha = \alpha_{alt} + \alpha_{neu} \cdot (1 - \alpha_{alt}) \quad (59)$$

Wobei  $c$  die akkumulierte Farbe und  $\alpha$  die akkumulierte Opazität ist. Der Index  $alt$  gibt an, dass dieser Wert das Ergebnis des vorherigen Blendings ist (oder 0, falls dies das erste ist), der Index  $neu$  dass dies der Wert der neu verarbeitenden Schnittfläche ist. Wenn sich der Wert von  $\alpha_{alt}$  in Pixeln nah genug an 1 annähert, so können diese in späteren Blending Schritten übersprungen werden, da der Einfluss dieser Blendings auf die Farbe der Pixel sehr klein ist. Dies setzt jedoch voraus, dass das Blending dahingehend konfigurierbar ist - dies ist nicht in allen Versionen von OpenGL oder anderen Renderingsystemen der Fall.

Wenn die Schnitte dagegen mit abnehmender Entfernung zur Kamera verarbeitet werden, so ergeben sich für das Blending die Gleichungen

$$c = (1 - \alpha_{neu}) \cdot c_{alt} + c_{neu} \cdot \alpha_{neu} \quad (60)$$

$$\alpha = (1 - \alpha_{neu}) \cdot \alpha_{alt} + \alpha_{neu} \quad (61)$$

so muss die akkumulierte Opazität der Pixel nicht gespeichert werden, was bei geringer Speicherkapazität ein Vorteil sein kann. Dafür ist frühzeitiges Terminieren nicht möglich.

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

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

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

## 5.2 Raycasting

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

Image-order Verfahren können mathematisch als die approxmierte Berechnung von Linienintegralen beschrieben werden. Im Folgenden sei ein Beispiel für ein solches Integral am Beispiel der Erzeugung von transluzenten Bildern gegeben. Der Datensatz wird zunächst als Wolke von Partikeln aufgefasst, wobei die Partikel Licht emmitieren und absorbieren [15, s. 134 f.]. Wie viel Licht Partikel in einem Voxel absorbieren und emittieren sowie die Farbe des emitierten Lichts wird über die Transferfunktion beschrieben. Das aus Richtung der Strahlen zur Kameraposition hin auf der Bildebene eintreffende Licht entspricht dann dem Integral

$$I(x, r) = \int_0^L C(s)\mu(s)e^{-\int_0^s \mu(t)dt} ds \quad (62)$$

Dabei entspricht  $x$  der Position auf der Bildebene,  $r$  der Richtung der Strahlen,  $L$  der Länge der Strahlen, bis sie zum letzten Mal einen Voxel schneidet,  $C(s)$  dem an Position  $s$  emittierten Licht und  $\mu(s)$  dem Okklusionskoeffizienten an  $s$ , also dem Anteil von Licht, das an  $s$  absorbiert wird ( $\mu(s) \in [0; 1]$ ). Einfacher ausgedrückt emittiert jeder Punkt Licht und absorbiert gleichzeitig einen Teil des von Punkten emittierten Lichts, die auf demselben Strahl liegen und weiter von der Kamera entfernt sind. Das absorbierte Licht entspricht der berechneten Opazität des Voxels an diesem Punkt.

Dieses Integral lässt sich im allgemeinen Fall nur mit viel Rechenaufwand, wenn überhaupt, automatisch berechnen [15, S. 136]. Um dennoch Visualisierungen mit Interaktionen, die keine spürbare Verzögerung hervorrufen, umsetzen zu können, wird das Integral

vereinfacht. Durch Vereinfachungsschritte (siehe [15, S. 136] für Details) ergibt sich die Formel

$$I(x, r) = \sum_{i=0}^{L/\Delta s - 1} C(i\Delta s) \alpha(i\Delta s) \prod_{j=0}^{i-1} (1 - \alpha(j\Delta s)) \quad (63)$$

Hier entspricht  $\Delta s$  einem Bruchteil von  $L$ . Desto kleiner  $\Delta s$  wird, desto besser wird das Integral approximiert. Die Funktion  $\alpha(i\Delta s)$  ermittelt die Opazität an dem Punkt des Strahls, der  $i\Delta s$  von der Kamera entfernt ist. Um die schon beim Texture Slicing beschriebenen Probleme zu vermeiden, wird die Opazität direkt mit der Entfernung zwischen den Abtastpunkten auf einem Strahl skaliert, die proportional zum Inversen der Anzahl der Abtastpunkte pro Strahl ist. Die Gleichung kann in zwei äquivalente rekursive Folgen zerlegt werden (für  $i > 0$ ):

$$\alpha_0 = c_0 = 0 \quad (64)$$

$$c_i = C(i\Delta s) \cdot \alpha(i\Delta s) \cdot (1 - \alpha_{i-1}) + c_{i-1} \quad (65)$$

$$\alpha_i = \alpha \cdot (i\Delta s) \cdot (1 - \alpha_{i-1}) + \alpha_{i-1} \quad (66)$$

Diese rekursiven Funktionen entsprechen der Abtastung entlang des Strahls ausgehend von der Kamera (front-to-back). Wenn  $\alpha_i$  sich nah genug an 1 annähert kann die Berechnung frühzeitig terminieren. Für die umgekehrte Richtung (zur Kamera hin, back-to-front) ergibt sich

$$\alpha_0 = c_0 = 0 \quad (67)$$

$$c_i = c_{i-1} \cdot (1 - \alpha(i\Delta s)) + C(i\Delta s) \cdot \alpha(i\Delta s) \quad (68)$$

$$\alpha_i = \alpha_{i-1} \cdot (1 - \alpha(i\Delta s)) + \alpha(i\Delta s) \quad (69)$$

Dies hat jedoch den Nachteil, dass frühzeitiges Terminieren nicht möglich ist, weshalb diese Variante nur sehr selten benutzt wird.

Durch Verwendung der rekursiven Gleichungen wird der Farbwert eines Pixels bestimmt ohne Zwischenergebnisse in eine Textur schreiben zu müssen. Daher wird beim Raycasting kein Blending benötigt.

Eine direkte Umsetzung der front-to-back Funktionen traversiert den Datensatz entlang der Strahlen und berechnet das emittierte und absorbierte Licht an Abtastpunkten, die in regelmäßigen Abständen auf den Strahlen liegen, was der üblichen Herangehensweise bei image-order Verfahren entspricht. Es ist dabei sinnvoll, Abtastpunkte erst innerhalb des Datensatzes, also im oder am Rand des ersten geschnittenen Voxels zu setzen. In Abb. 15 wird Raycasting mit dieser Optimierung schematisch dargestellt. Da für

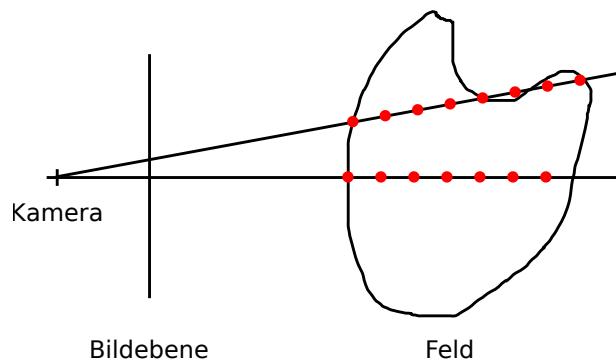


Abbildung 15: Schematische Darstellung des Raycastings. Die roten Punkte stellen Abtastpunkte dar, die in regelmässigen Abständen auf den Strahlen liegen.

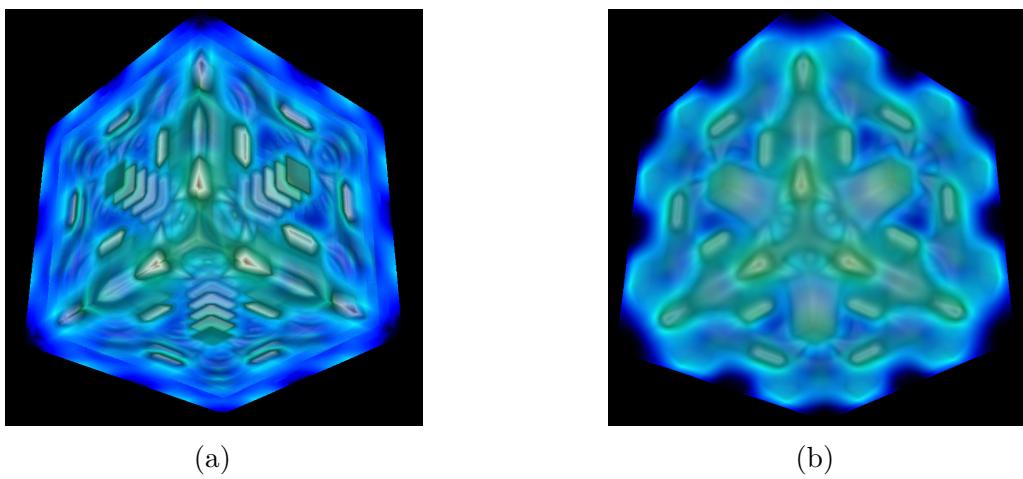


Abbildung 16: Zwei Raycastings desselben Skalarfelds mit unterschiedlichen Abtastraten.

dieses Verfahren Strahlen durch das Volumen berechnet werden, entlang derer abgetastet wird, bezeichnet man es als Raycasting. Die Vorteile des Raycastings liegen der schon angesprochenen Möglichkeit zur frürzeitigen Terminierung und der Möglichkeit, die Anzahl der Abtastpunkte, auch bezeichnet als “Abtrastrate”, pro Strahl festzulegen.

Die Qualität eines Raycastings ist stark abhängig von der Abtastrate auf den Strahlen und somit vom Abstand zwischen den Abtastpunkten. Ein geringerer Abstand führt generell zu weniger Artefakten und macht kleine Strukturen besser erkennbar, erhöht jedoch den Rechenaufwand. Indem der Nutzer die Abtastrate selbst auswählt, kann er den Tradeoff zwischen Rechenzeit und Bildqualität an seine Bedürfnisse anpassen, genau wie schon beim Texture Slicing. In Abb. 16a und 16b sind zwei Darstellungen desselben Skalarfeldes  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ ,  $f(x, y, z) = \cos(x \cdot y \cdot z)$  zu sehen, die durch Raytracing mit einer unterschiedlichen Anzahl von Abtastpunkten erzeugt wurden. 16a wurde mit 20 Abtastpunkten erstellt, wodurch visuelle, scheibenartige Artefakte entstanden. Bei einer

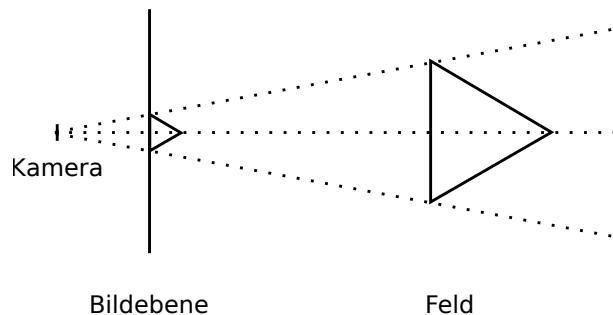


Abbildung 17: Schematische Darstellung von Cell Projection. Das große Dreieck rechts wird auf die Bildebene links projiziert. Die Projektion wird durch das kleinere Dreieck direkt an der Bildebene dargestellt.

höheren Menge von 200 Abtastpunkten in 16b verschwinden diese Artefakte.

### 5.3 Cell Projection

Cell Projection zählt zu den object-order Verfahren. Es basiert auf der Arbeit von Lenz, Gudmundsson, Lindskog und Danielsson [27] sowie der von Frieder, Gordon und Reynolds [13]. Die grundlegende Idee der Cell Projection besteht darin, für jedes Voxel ein zweidimensionales Polygon zu berechnen und dieses auf die Bildebene zu projizieren. Form, Farbe und Transparenz des Polygons werden dabei durch die Form des Voxels sowie die relative Position der Kamera zum Voxel bestimmt. Punkte des Polygons entsprechen Geraden durch das Voxel, deren Richtung von der Kameraposition abhängt. Die Opazität an diesen Punkten entspricht dem Integral der Opazität entlang der Linien. Dieses Vorgehen ist vergleichbar mit dem schon vorgestellten Continuous Scatterplotting, speziell dem Fall 5. In Abb. 17 ist das Vorgehen schematisch abgebildet. Auf jeden Punkt der Bildebene wird ein eindimensionaler Ausschnitt des Voxels abgebildet. Die Länge dieses Ausschnitts gibt die Dicke des Voxels in Richtung des Punkts auf der Bildebene an, betrachtet von der Kameraposition aus. Diese Dicke wird in Abb. 17 durch die Entfernung der Punkte auf den Schenkeln des Dreiecks zur Bildebene ausgedrückt.

Wenn sich mehrere Projektionen der Voxel auf der Bildebene überschneiden, werden sie mittels Blending kombiniert. Die für das Blending verwendeten Formeln sind abhängig von der Reihenfolge, in der die Voxel verarbeitet werden: Die Formeln 58 und 59 im front-to-back Fall und die Formeln 60 und 61 im back-to-front Fall. Bei object-order Verfahren ist in der Regel keine frühzeitige Terminierung möglich. Aus diesem Grund und weil back-to-front die akkumulierte Opazität nicht abspeichern muss wird es meistens bevorzugt.

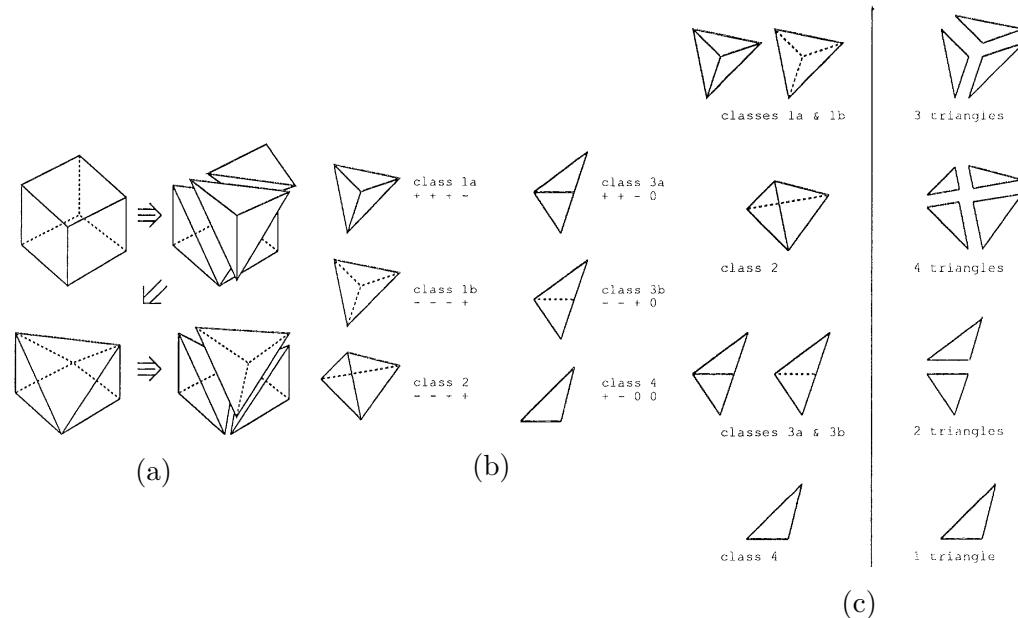


Abbildung 18: Teilaufgaben einer beispielhaften Cell Projection: (a) Zerlegung eines quadratischen Voxels in fünf tetraedrische Voxel. (b) Klassifizierung von Tetraeder abhängig von ihrer Ausrichtung zur Kamera. (c) Zerlegung der Projektion des Tetraeders in Dreiecke.

Cell Projection Verfahren unterscheiden sich stark darin, welche Arten von Voxel als Eingabe dienen können und wie die Umwandlung in zweidimensionale Voxel ausgeführt wird. Diese beiden Teilprobleme werden im Folgenden anhand des von Shirley und Tuchman [33] entwickelten Algorithmus erläutert.

Im ersten Schritt zerlegt der Algorithmus die Voxel in tetraedrische Teile (siehe Abb. 18a). Die daraus entstehenden Tetraeder werden anhand der Ausrichtung ihrer Flächen zur Blickrichtung der Kamera klassifiziert. Dazu wird das Skalarprodukt zwischen den Normalen der Flächen (die hier immer als nach außen zeigend angenommen werden) und der Blickrichtung berechnet. Die Klassen sind in Abb. 18b dargestellt. Die +, - und 0 neben den Klassen geben dabei an, welche Werte die Skalarprodukte jeweils annehmen können: Ein + steht für ein positives, ein - für ein negatives Skalarprodukt. 0 steht dafür dass das Skalarprodukt 0 ist, also die Normale orthogonal zur Blickrichtung.

Nachdem die Tetraeder klassifiziert sind, werden die von ihnen durch Projektion auf die Bildebene erzeugten Flächen bestimmt. Dazu werden die Eckpunkte projiziert und die entstehenden Polygone in Dreiecke zerlegt (siehe Abb. 18c). Werte für neu entstandene Eckpunkte der Dreiecke werden durch Interpolation berechnet. Die Dicke des Tetraeders an Punkten im Inneren des Polygons wird durch die Euklidische Distanzformel bestimmt. Die Opazität an diesen Punkten wird abhängig von der Dicke des Tetraeders dort und den skalaren Werten entlang der Geraden durch den Tetraeder, die auf den Punkt abgebildet wird, bestimmt.

$\frac{1}{273}$	1	4	7	4	1
	4	16	26	16	4
	7	26	41	26	7
	4	16	26	16	4
	1	4	7	4	1

Abbildung 19: Ein diskreter  $5 \times 5$  Gauß Kern. Die Werte in den Zellen werden mit dem Faktor links multipliziert, sodass die Summe aller Zellen 1 entspricht.

Schlussendlich werden die Dreiecke auf die Bildebene gerendert. Die Farb- und Opazitätswerte im Inneren der Dreiecke ergeben sich durch Interpolation der Werte an ihren Eckpunkten.

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

## 5.4 Splatting

Splatting basiert auf einem von Westover entwickelten Algorithmus[36][35]. Wie auch bei der Cell Projection werden für die Voxel in back-to-front Reihenfolge zweidimensionale Repräsentationen gebildet, die auf die Bildebene projiziert und dort mittels Blending kombiniert werden. Im Gegensatz zur Cell Projection wird jedoch nicht versucht den Datensatz so exakt wie möglich darzustellen, sondern das ursprüngliche Feld aus den diskreten Werten der Voxel zu so gut wie möglich zu rekonstruieren und darzustellen.

In der Signalverarbeitung ist die Rekonstruktion von kontinuierlichen Funktionen aus diskreten Funktionen ein zentrales Problem. Dazu wird meistens Faltung verwendet.

Die Erzeugung einer diskreten Annäherung an das ursprüngliche kontinuierliche Feld kann durch Faltung der diskreten Abtastung mit einem diskreten Kern geschehen. Dafür werden eine Reihe unterschiedlicher Kerne benutzt. Einer der beliebtesten davon ist die diskrete Abtastung von Gaußschen Glockenkurven. Splatting bedient sich dieser Technik, indem es die Projektion der Mittelpunkte der Voxel auf der Bildebene berechnet und diskrete Abtastungen von zweidimensionalen Glockenkurven an diesen platziert. Die Abtastungen werden abhängig von Eigenschaften der Voxel skaliert: Ihre Größe, Form und Drehung werden so angepasst, dass sie die Form des Voxels so gut wie möglich wiedergeben. Zusätzlich wird die Höhe des Maximums so skaliert, dass sie dem Wert des Voxels entspricht. Damit nicht für jeden Voxel ein neuer Kern berechnet werden

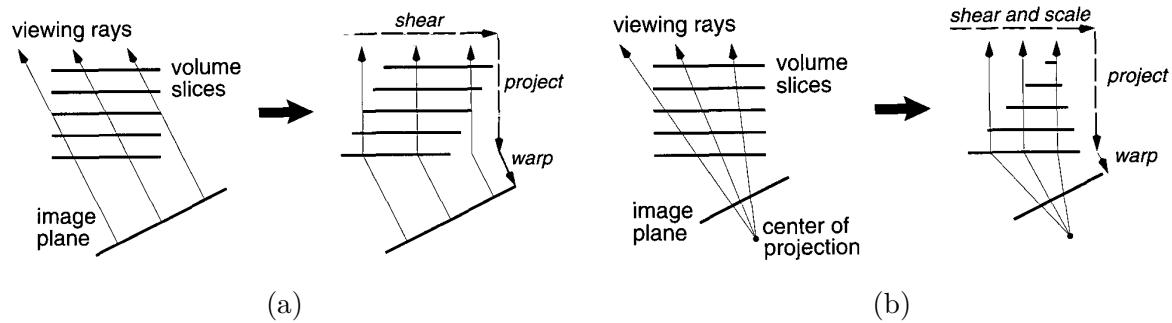


Abbildung 20: Darstellungen der im Shear Warp verwendeten Scherungs- und Verzerrungsoperationen auf dem Datensatz und den Sichtstrahlen in (a) paralleler Projektion und (b) perspektivischer Projektion. Entnommen aus [25].

muss, wird zu Beginn des Rendering ein einzelner Filterkern berechnet und in einer Lookup-Tabelle abgespeichert. Eine solche Tabelle ist in Abb. 19 dargestellt.

Splatting bietet gegenüber der Cell Projection und ähnlichen Verfahren den Vorteil, dass im entstehenden Bild keine harten Voxelkanten zu sehen sind. Die aus der Signalverarbeitung übernommenen Methoden garantieren eine hohe Bildqualität mit relativ kurzen Rechenzeiten. Splatting ist daher eines der beliebtesten object-order Verfahren. Das Fehlen von harten Kanten stellt jedoch auch einen Nachteil von Splatting dar: Die Grenzen des Datensatzes werden weich gezeichnet. Ein zusätzliches Problem tritt auf, wenn die Größe der Voxel stark variiert. Wenn beispielsweise einige wenige Voxel sehr viel größer sind als der Rest, können die für diese Voxel erzeugten Kerne deutlich erkennbar sein. Da einzelne Kerne die Form ihrer Voxel nur schlecht wiedergeben können, kann dies zu deutlichen visuellen Artefakten im erzeugten Bild führen.

## 5.5 Shear Warp

Shear Warp zählt zu den hybriden Verfahren. Es kombiniert Techniken aus object- und image-order Verfahren, wodurch es typische Vorteile beider Arten kombiniert. Entwickelt wurde Shear Warp von Philipp Lacroute und Marc Levoy, die es 1994 zuerst vorstellten [25].

Der Grundansatz des Shear Warp besteht darin, den volumetrischen Datensatz mittels einer Scherung (engl. ‘shear’) so zu transformieren, dass von der Kameraposition ausgehende und die Bildebene schneidende Strahlen parallel zu einer der Koordinatenachsen sind. Welche Koordinatenachse dafür gewählt wird hängt von Blickrichtung der Kamera ab, in der Regel wird die Koordinatenachse mit dem geringsten Winkel zur Blickrichtung bevorzugt. Im Folgenden wird diese Achse als ‘Sichtachse’ bezeichnet, die zugehörige Koordinate ‘Sichtkoordinate’.

Abhängig von der Art der für die spätere Bilderzeugung gewählten Projektion muss die Scherung unterschiedlich durchgeführt werden. Bei paralleler Projektion genügt eine

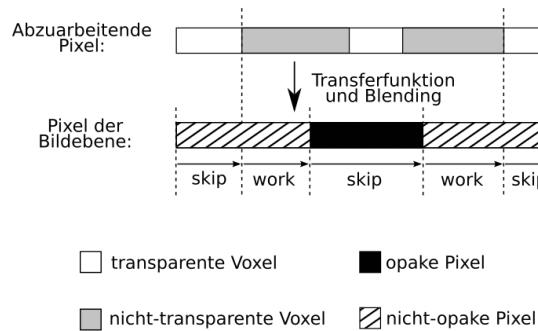


Abbildung 21: Darstellung der optimierten Abbildung von Voxeln auf Pixel. Wenn entweder die Voxel vollständig transparent oder die Zielpixel vollständig opak sind, werden sie übersprungen ('skip'), ansonsten werden die Voxelwerte auf die Pixel abgebildet ('work'). Erstellt nach einem Bild aus [25].

Verschiebung der Voxelschichten untereinander (siehe Abb. 20a). Dies setzt voraus, dass die Voxel eine einheitliche Größe und Form haben, in der Regel quaderförmig. Bei perspektivischer Projektion muss zusätzlich die Größe der Schichten so skaliert werden, dass sie mit zunehmender Entfernung zur Kamera kleiner werden (siehe Abb. 20b). Wenn durch Verschiebung und Skalierung mehrere Eingabevoxel zum Teil oder ganz auf einen Ausgabevoxel abgebildet werden, müssen die Werte der Voxel gewichtet nach der Größe des Anteils interpoliert werden, um den Datensatz so gut wie möglich darstellen zu können.

Nachdem die Scherung durchgeführt wurde, wird die Projektion der Voxel auf die Bildebene begonnen. Der Datensatz wird dabei schichtweise verarbeitet. Die Pixel, auf die ein Voxel abgebildet wird, sind durch die Scherung leicht zu bestimmen. Es sind genau die, deren zwei Koordinaten innerhalb der entsprechenden nicht-Sichtkoordinaten des jeweiligen Voxels liegen. Dies erspart teure Projektionsoperatoren wie bei object-order oder schrittweise Abtastung wie bei image-order Verfahren üblich. Diese Eigenschaft erlaubt eine Optimierung. Wenn ein Pixel der Bildebene bereits eine hohe Opazität akkumuliert hat (ähnlich wie die frühzeitige Terminierung bei image-order Verfahren), kann dieser übersprungen werden. Implementiert wird dies meistens, indem für jeden opaken Pixel der Offset zum nächsten nicht-opaken Pixel gespeichert wird. Das gleiche Verfahren kann auch bei Voxeln verwendet werden, denen die Transferfunktion eine Opazität von 0, also vollständige Transparenz, zuordnet. Diese enthalten den Offset zum nächsten Voxel mit Opazität größer 0. Abb. 21 zeigt diese Optimierung. Die notwendigen Offsets der Voxel werden für jede achsenparallele Blickrichtung vorberechnet, also insgesamt dreimal. Optional können komplett transparente Voxel auch direkt entfernt und die Offsets in

den nicht-transparenten Voxeln gespeichert werden, wodurch sich die bei object-order Verfahren üblichen Speicherersparnisse ergeben.

Im Gegensatz zu object-order Verfahren werden Voxel beim Shear Warp nicht in Basisfunktionen übersetzt, sondern pro Pixel die jeweiligen Werte durch Interpolation bestimmt. Da die Dicke der Voxel von der Blickrichtung abhängt, wird die Opazität mittels einer vorberechneten Tabelle skaliert. Im perspektivischen Fall kann es sein, dass auf einen Pixel zwei oder mehr Voxel abgebildet werden. Dies muss bei der Optimierung berücksichtigt werden.

Zum Abschluss wird eine zur Scherung inverse Verzerrung auf die Bildebene angewendet (engl. ‘warp’), um das endgültige Bild zu erzeugen. Diese Operation ist, wie in Abb. 20a und 20b dargestellt, identisch für parallele und perspektivische Projektion.

## 6 Umsetzung

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

Die Grundidee der entwickelten Visualisierung besteht darin, zwei DVR zu erzeugen. Das erste stellt die Zellen des ursprünglichen Feldes dar, über die eine konstante Dichte angenommen wird. Da das Zielgebiet der Anwendung die Materialforschung ist, wird das ursprüngliche Feld im 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.

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

### 6.1 Datenvorbereitung

#### 6.1.1 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

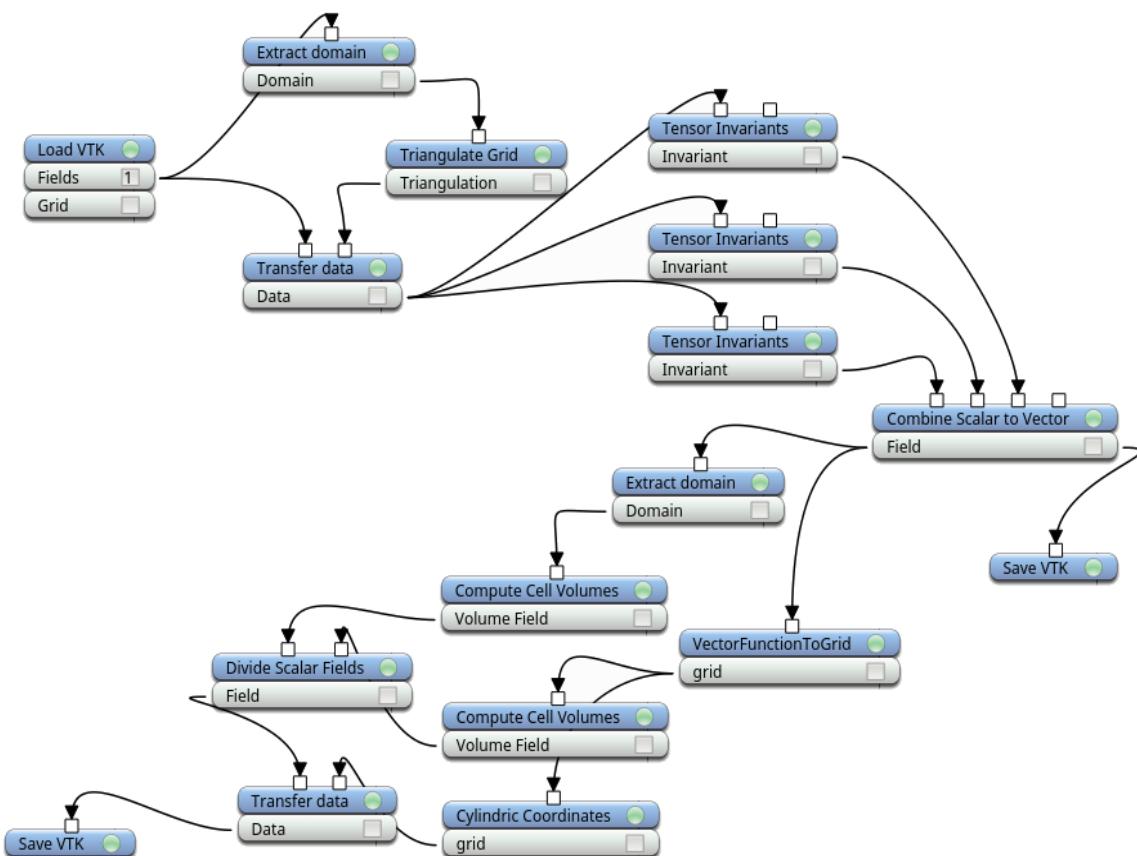


Abbildung 22: Der Flowgraph der FAnToM Session, die zur Vorbereitung der Daten verwendet wird.

verwendeten Session ist in Abb. 22 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  $3 \times 3$  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.

Der Algorithmus ‘Cylindric Coordinates’ ist optional. Er interpretiert die Komponenten der Vektoren als zylindrische Koordinaten und berechnet die kartesischen Koordinaten der Punkte. Ein Punkt der Eingabe 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  $[-1; 1]$  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. Die Zellstruktur der Eingabe wird beibehalten, lediglich die Positionen der Eckpunkte der Zellen werden verändert.

Ziel des Cylindric Coordinate Algorithmus ist es, eine einheitliche Eingabe in kartesischen Koordinaten für die eigentliche Visualisierung zu erzeugen. Er ist für die Anwendung bei Invariantensätzen gedacht, die sich für die Darstellung in zylindrischen Koordinaten eignen, beispielsweise die I- und J-Invariantensätze. Bei anderen Invariantensätzen sollte er nicht verwendet werden.

Der Vorteil dieser Vereinheitlichung liegt darin, dass das Format der erzeugten Datensätze optimiert werden kann, dass die Laufzeit des Visualisierungsalgorithmus minimiert wird.

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

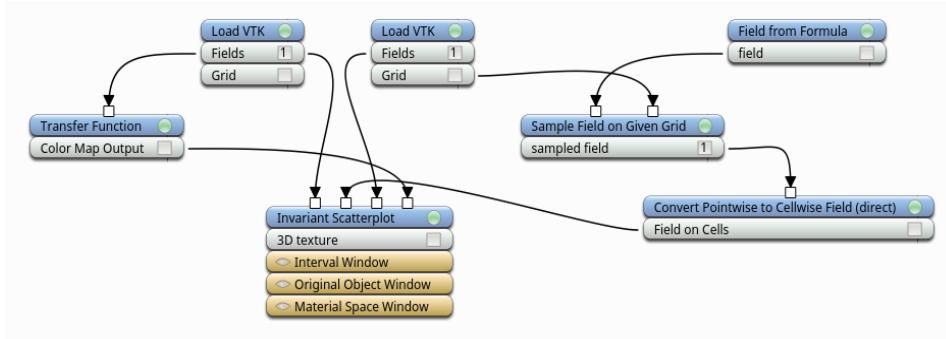


Abbildung 23: Der Flowgraph der Visualisierungssession.

(Original geteilt durch verschobenes Feld, ‘Divide Scalar Fields’). Das Ergebnis ist wiederum ein Skalarfeld, das  $\sigma_V$  der Zellen für das Continuous Scatterplottings approximiert. Die Herangehensweise wird in Abschnitt 6.4 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.

### 6.1.2 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 Dichten innerhalb der Voxel im Objekt. Die Session ist in Abb. 23 zu sehen.

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

## 6.2 Wahl des DVR Verfahrens

Die vorgestellten Verfahren unterscheiden sich in Hinsicht auf Laufzeit, Speicherverbrauch, Adaptierbarkeit und Qualität der erzeugten Visualisierung. Zum Abschluss des

Kapitels werden sie deshalb anhand dieser Eigenschaften und den aus der Aufgabenstellung resultierenden Anforderungen verglichen und eins der Verfahren zur Umsetzung ausgewählt.

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

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

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

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

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

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

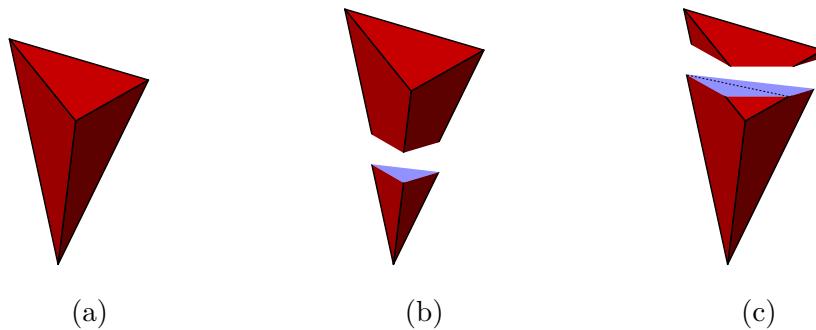


Abbildung 24: Darstellungen von möglichen Schnitten durch einen Tetraeder. (a) Der Tetraeder ohne Schnitte. (b) Ein Schnitt, der durch ein einzelnes Dreieck dargestellt werden kann. (b) Ein Schnitt, der ein Viereck bildet. Die gestrichelte Linie deutet eine mögliche Unterteilung in Dreiecke an.

### 6.3 Voxelisierung

Da das in dieser Arbeit verwendete DVR Verfahren (siehe Abschnitt 6.6) 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, 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.[31]. 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. 24). 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, wurde ein Verfahren entwickelt und implementiert, das das Feld entlang der x-Achse unterteilt und in mehrere einzelne Texturen voxelisiert. Dasselbe Verfahren kann theoretisch auch für die anderen Richtungen implementiert werden, das war jedoch für keinen der verwendeten Datensätze nötig.

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

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

**Data:**  $p_i$  Eckpunkte des Tetraeders,  $z$  Parameter der Schnittebene  
**Result:** Schnittfläche zwischen Tetraeder und Schnittebene

```

schnittZahl ← 0;
schnittPunkte ← [];
for ∀( $p_m, p_n$ ),  $n < m$ ;                                // 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] ←  $p$ ;
    |   | schnittZahl++;
    | end
end
if schnittZahl == 4 then
    | sortieren(schnittPunkte);
end
erzeugeDreiecke(schnittPunkte);
```

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

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

Zudem wird die Dichte der Voxel modifiziert. 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 zehn Größenordnungen. Durch das Ziehen der dritten Wurzel nähern 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 die schon mehrmals erwähnte von OpenGL bereitgestellte Funktion des Blendings. Im Objekt können Tetraeder nicht überlappen, weshalb die Invariantenwerte in der Textur davon nicht betroffen sind. Durch das Blending wird  $\sigma_{V_i}$  für jeden Tetraeder  $V_i$  berechnet und korrekt pro Voxel aufaddiert. Damit ist die zweite Hälfte des Continuous Scatterplottings implementiert.

## 6.4 Umsetzung des Continuous Scatterplottings

Durch die Kombination der beiden beschriebenen Verfahren wird in dieser Arbeit ein vollständiges Continuous Scatterplotting Verfahren implementiert. Dies wird in diesem Abschnitt genauer begründet.

Der erste Teil, das Berechnen des Dichtebeitrags  $\sigma_{V_i}$  der Tetraeder  $V_i$ , erfolgt bereits in der Vorverarbeitung der Daten. Dabei wird die für Fall 1 angegebene Formel 53 verwendet. Die anderen Fälle können zwar theoretisch vorkommen, sind jedoch extrem unwahrscheinlich. Durch unvermeidbare Rundungsfehler werden Bereiche konstanter Invarianten (Fall 2), Nullmengen (Fall 3) und Abbildungen auf Teilräume mit geringerer Dimension (Fall 5) zerstört. Das Erkennen von diesen Rundungsfehlern wird dadurch erschwert, dass in den verwendeten Datensätzen tatsächlich Bereiche mit sehr geringen Veränderungen der Invarianten oder Abbildungen auf sehr flache Tetraeder vorkommen. Die Unterscheidung 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 51 anwenden. Die  $\sigma_V$  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.

## 6.5 Berechnung der maximalen Dichte

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

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

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

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

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

Die Daten in der 3D Textur sind nicht geordnet und es existiert auch keine andere Struktur, durch die die Berechnung des Maximums vereinfacht werden könnte. Deshalb musste sie als lineare Suche implementiert werden. Um die potentiell hohe Laufzeit von  $O(n)$  zu verringern, wurde die Suche mithilfe eines ‘Divide-and-Conquer’ Ansatzes parallelisiert. Divide-and-Conquer beschreibt ein häufig angewendetes Verfahren zur Lösung von Problemen. Dabei werden komplexe Probleme in einfachere Teilprobleme aufgeteilt, deren Lösungen so kombiniert werden, dass sie eine Lösung des ursprünglichen komplexen Problems darstellen [19].

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

Die Eingabe der Shaderpipeline ist die 3D Textur, sowie eine Linie zwischen den Punkten  $(-1.0, 0.0, 0.0)$  und  $(1.0, 0.0, 0.0)$ . Berechnet wird eine Darstellung der Linie aus  $Z$  Pixeln, wobei  $Z$  die Anzahl 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.

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

**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.

## 6.6 Umsetzung des Raycastings

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

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

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

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

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

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.

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 Formeln 65 und 66 zu einem Farbwert kombiniert.

Auch die bereits angesprochenen Optimierungen (Samplepunkte beginnen erst am Rand des Volumendatensatzes, frühzeitige Terminierung) sind implementiert.

Wenn beim Rendering des Feldes im Invariantenraumes ein Samplepunkt ausserhalb der ausgewählten Intervalle liegt, werden 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 Intervalls 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. 25 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. 25a). Dadurch sind Löcher im Volumen leichter zu erkennen, ohne Lichtquellen und Schattenbildung berechnen zu müssen.

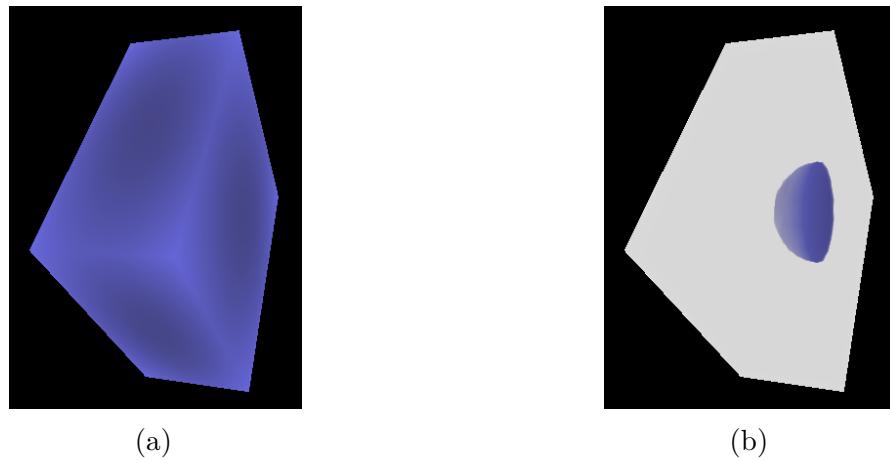


Abbildung 25: Ergebnisse des Raycastings auf einem quaderförmigen Feld von  $3 \times 3$  Matrizen. In (a) ist der gesamte Invariantenbereich ausgewählt, in (b) ist er auf hohe Werte begrenzt. Zu beachten ist, dass in (a) die Farben von Punkten näher am Zentrum des Feldes dunkler und in (b) die ausgeblendeten Bereiche als stark transparent und weiß dargestellt werden.

Die Implementierung basiert auf dem FAnToM Algorithmus ‘Volume Rendererer GLSL’, verändert diesen jedoch deutlich um mehrere 3D Texturen in x-Richtung als Eingabe zuzulassen, Bereiche des Volume Renderings ausblenden zu können, korrekte Überlappung mit dahinter gezeichneten Objekten sicherzustellen und die in Abschnitt 6.10 beschriebenen Optionen zu ermöglichen.

## 6.7 Die Interaktionswidgets

Um in der Darstellung des Feldes im Invariantenraum Intervalle von Invarianten auswählen zu können, wurden Interaktionflächen implementiert. Leiner et al. [26] stellen an solche Elemente, die sie als “3D Widgets” bezeichnen, eine Reihe von Kriterien:

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

**2. Zerlegung in Handles** Da Tastatur und Maus oft nicht genügend Freiheitsgrade bieten um dreidimensionale Interaktionen direkt durch sie umsetzen zu können, wird die Interaktion in Teile zerlegt. Für jeden Teil sollte eine Komponente des Widgets reserviert werden, der durch Maus und Tastatur manipuliert werden kann. Diese Komponenten werden von Leiner et al. als “Handles” bezeichnet. Die Interaktion mit einem Widget gliedert

**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 invarianten  $\notin B$  then
        | fragmentFarbe  $\leftarrow$  fragmentFarbe = (1.0, 1.0, 1.0, 0.05);
        | transparenz  $\leftarrow$  transparenz - 0.05;
        | continue;
    end
    // Nur im Invariantenraum
    if  $p \notin B$  then
        | continue;
    end
    voxelFarbe  $\leftarrow$  transferfkt(dichte);
    fragmentFarbe  $\leftarrow$  fragmentFarbe + voxelFarbe  $\cdot$  transparenz;
    transparenz  $\leftarrow$  transparenz - voxelFarbe.transparenz;
end

```

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

sich damit in zwei Teile: Die Selektion des gewünschten Handles und die Manipulation des Handles.

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

Preim et al. [30, S. 340] fügen noch ein weiteres Kriterium hinzu:

**4. Affordances der Handles** Mit dem Begriff “Affordances” beschreiben Preim et al. [29, S. 137] Eigenschaften von Objekten, die auf die Interaktionsmöglichkeiten dieser Objekte hinweisen. Beispielsweise sollte ein Stuhl Affordances besitzen, die darauf schließen lassen, dass es möglich ist sich auf diesen zu setzen. Welche Eigenschaften als Affordances erkannt

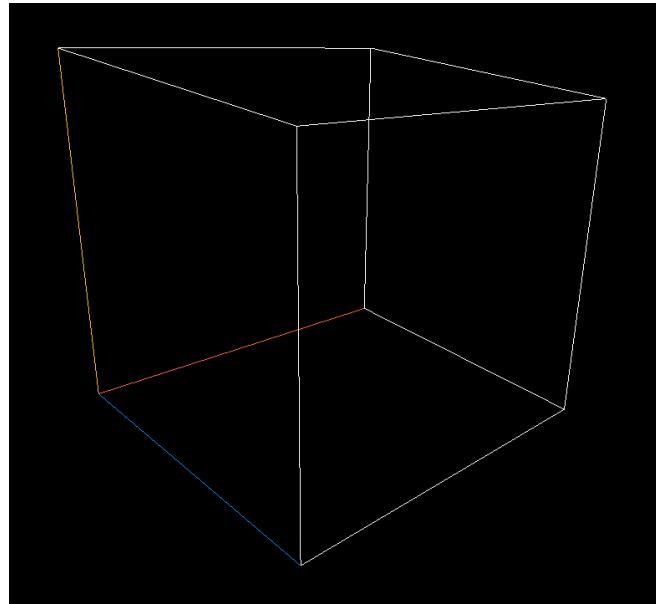


Abbildung 26: Eine Darstellung des quadratischen Interaktionswidgets. Die drei eingefärbten Kanten schneiden sich in dem Punkt dessen Koordinaten die Minima aller drei Invarianten sind. Die Farben dienen der Orientierung im Raum und weisen darauf hin, zu welcher Achse die jeweilige Kante parallel ist: Rot zur x-, Gelb zur y- und Blau zur z-Achse.

werden, hängt von einer Vielzahl von physikalischen und logischen Faktoren ab, sowie davon, welche anderen Interaktionen der Benutzer bereits kennt.

Da in der Literatur kein für die gewünschten Interaktionen geeignetes Widget beschrieben ist, wurden im Rahmen dieser Arbeit eigene Widgets entwickelt, durch die Intervalle von Invarianten im Invariantenraum ausgewählt werden können. Da die Invariantenräume meistens in einem von zwei Koordinatensystemen dargestellt werden, den kartesischen und den zylindrischen Koordinaten, wurden für diese beiden Koordinatensysteme jeweils ein eigenes Widget entwickelt. Die Auswahl des angezeigten Widgets muss dabei der Benutzer treffen, da VTK keine Möglichkeit unterstützt, das verwendete Koordinatensystem mit abzuspeichern.

Beide Widgets kombinieren die Konzepte des Cutaways [30, S. 406 f.], einer Interaktion bei der Teile des Datensatzes ausgeblendet werden um das Innere des Datensatzes zu zeigen, mit den Interaktionen zur Positionierung von Schnittebenen, wie sie auch in Abschnitt 4.6.2 beschrieben werden.

### **6.7.1 Quadratisches Interaktionswidget**

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

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

Die Geraden unterscheiden sich deutlich von der Darstellung des DVR, wodurch Kriterium 1 erfüllt wird. Ein Problem kann jedoch auftreten, wenn die Farben der Transferfunktion sich mit denen der Kanten überschneiden. Abb. 26 zeigt das quadratische Interaktionswidget.

Die Interaktion findet statt, indem eine Seitenfläche durch Klicken und Halten der mittleren Maustasten ausgewählt wird und durch Bewegung der Maus entlang der zur Fläche orthogonalen Achse verschoben wird. Jede Seitenfläche stellt einen Handle dar, durch den das Minimum oder Maximum des angezeigten Intervalls einer einzelnen Invariante eingestellt wird. Dadurch ist Kriterium 2 erfüllt. Die Position der Seitenflächen folgt dabei der Maus, sodass immer die aktuellen Intervallgrenzen gezeigt werden. Dies erfüllt Kriterium 3 zum Teil: Das aktuelle Handle liegt immer unter dem Mauszeiger und bewegt sich mit ihm, wodurch das ausgewählte Handle meist erkennbar ist.

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

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

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

Die Berechnung, ob ein Mausklick eine Handle trifft, geschieht mittels ‘Ray-Picking’. Dabei wird, ähnlich wie beim Raycasting, ein Strahl ausgehend von der Kamera durch die Szene berechnet. Wenn der Strahl ein Handle trifft, wird die jeweilige Intervallschranke ausgewählt. Wenn mehr als ein Handle getroffen wird, wird das nächste bevorzugt. Preim et al. vergleichen diese Art der Auswahl von Widgets als Metapher zum Zeigen auf

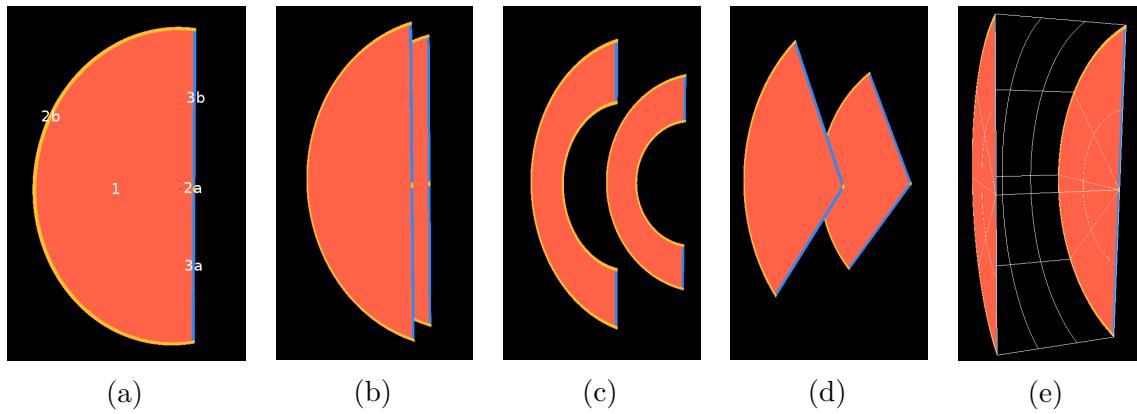


Abbildung 27: Die Darstellungen der zylindrischen Interaktionswidgets. (a) Ein Widget mit den einzelnen Handles. Ein Klick auf die Komponenten 1,2 und 3 des Interaktionswidgets wählt das Handle der jeweils ersten, zweiten oder dritten Invariante aus. Die Komponenten 2a und 3a sind dabei die Minima der Invariante, 2b und 3b die Maxima. (b) bis (d) zeigen veränderte Positionen und Formen der Interaktionswidgets durch gewählte Invariantenbereiche. In (e) ist der Wireframe dargestellt.

ein Objekt mit einem Laserpointer oder dem Greifen eines Objekts mit der Hand [30, S. 344].

### 6.7.2 Zylindrisches Interaktionswidget

Wie im Abschnitt 3.1.23 beschrieben, kann es für manche Variantensätze (z.B. die K- und R-Variantensätze) sinnvoll sein, sie in einem zylindrischen Koordinatensystem darzustellen. Dafür müssen eigene Interaktionswidgets entwickelt werden, durch die, äquivalent zum quadratischen Widget, die Intervallgrenzen der Varianten manipuliert werden können.

Die implementierten Widgets haben die Form von Kreissegmenten mit unterschiedlich eingefärbten Komponenten, die parallel zur yz-Ebene gezeichnet werden. Ein zylindrisches Interaktionswidget ist in Abb. 27a dargestellt. Jede farbige Komponente markiert einen Teil des Widgets, durch den eine andere Variante manipuliert werden kann. Diese Teile gliedern sich wiederum in Handles für das Minimum und Maximum der jeweiligen Variante auf. Die Wahl der Farben basiert folgt der gleichen Logik wie beim quadratischen Widget. Dadurch wird Kriterium 1 erfüllt.

Die Interaktion selbst ist analog zum quadratischen Widget: Durch Klicken der Maustaste wird ein Handle ausgewählt, dessen Position der Bewegung der Maus folgt. Für die erste Variante entspricht dies der x-Koordinate des jeweiligen Widgets (siehe Abb. 27b). Bei der zweiten und dritten Variante wird die Form des Widgets angepasst: Da die zweite Variante der Entfernung zur x-Achse entspricht, werden innerer und äußerer

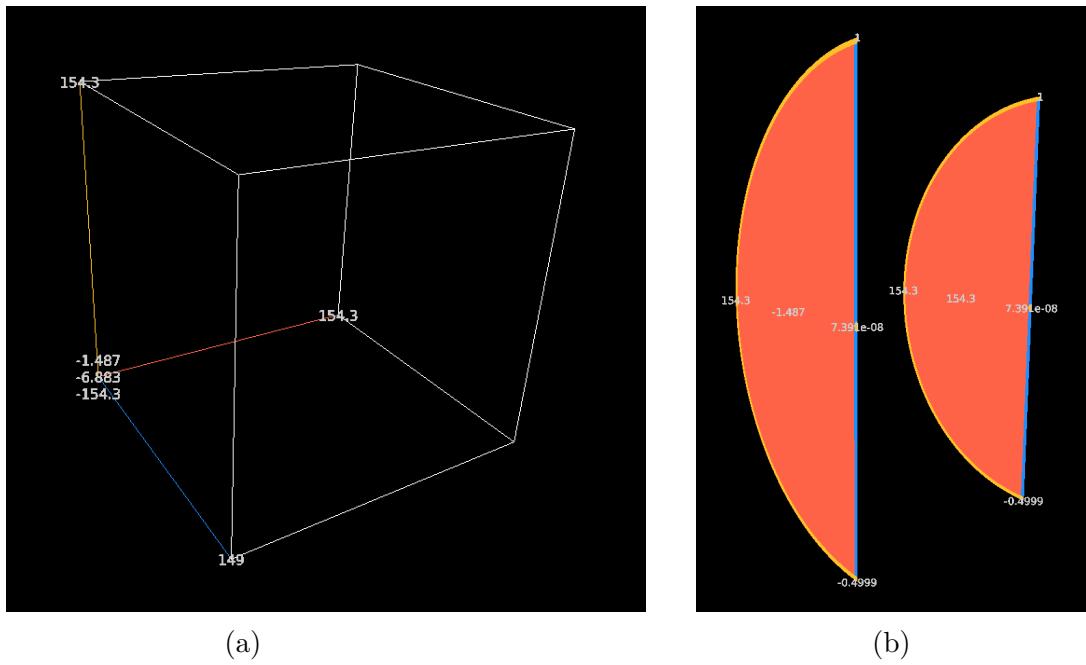


Abbildung 28: Labels

Radius des Kreissegments angepasst. Wenn der innere Radius größer als null ist, wird somit aus dem Kreissegment ein Kreisringsektor (siehe Abb. 27c). Die dritte Invariante entspricht dem Skalarprodukt zwischen der Projektion des Punktes auf die yz-Ebene und der y-Achse. Die zugehörigen Handles sind die beiden geraden Seiten des Kreissegments, deren Winkel zur y-Achse verändert wird (siehe Abb. 27d). Dies erfüllt Kriterium 2.

Kriterien 3 und 4 werden genau wie beim quadrischen Widget nur teilweise erfüllt und werden Thema von weiterer Forschung.

Die zylindrischen Widgets stellen somit die Seitenflächen eines Ausschnitts aus einem Zylinder dar, der den dargestellten Teil des Invariantenraums umfasst. Um diese Eigenschaft zu verdeutlichen, kann ein zusätzliches Gitter aus Linien, bezeichnet als ‘Wireframe’, gezeichnet werden, die sich auf der Oberfläche dieses Zylinders befinden (siehe Abb. 27e).

## 6.8 Labeling

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

Beim quadrischen Interaktionswidget werden die Labels an den Endpunkten der farbigen Kanten angezeigt. An dem Punkt, an dem sich die drei Kanten schneiden, werden die

Labels untereinander angeordnet, mit dem Label der ersten Invariante oben und dem der dritten unten. Abb. 28a zeigt die Positionierung der Labels.

Die Labels am zylindrischen Interaktionswidget sind an den Handles eingezeichnet. Für die erste Invariante befinden sich die Labels im Zentrum der zugehörigen Handles, der großen roten Flächen. Ebenso sind Labels der zweiten Invariante im Mittelpunkt ihrer Handles eingezeichnet, der gelben Bögen an der Innen- und Aussenseite der Kreisringausschnitte. Für die Position der Labels der dritten Invariante wurden die äußeren Eckpunkte der geraden Seiten der Kreisringausschnitte gewählt. Die Positionen sind in Abb. 28b zu sehen.

## 6.9 Weitere Interaktionen

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

**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.

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

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

**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.

## 6.10 Optionen des Algorithmus

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

### 6.10.1 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. Voxel, 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 beim Raycasting pro Pixel akkumulierte Transparenz durch die Anzahl der Abtastpunkte des Strahls zu teilen. Dies hat den Vorteil, dass stark transparente Voxel am Rand des Datensatzes deutlicher dargestellt werden, was die Erkennung von Strukturen in diesen Bereichen erleichtert, jedoch das Ablesen konkreter Werte durch umgekehrtes Anwenden der Transferfunktion erschwert.

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

### 6.10.2 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.

### 6.10.3 Optionen des Invariantenraumrenderings

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

**Wireframe** Wenn die zylindrischen Interaktionswidgets angezeigt werden, kann optional ein Wireframe angezeigt werden, ein Liniengitter an den Grenzen des maximalen Invariantenintervalls. 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. 27e zu sehen.

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

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

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

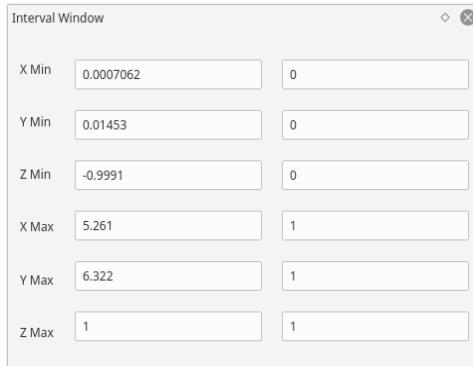


Abbildung 29: Das Intervall-Fenster.

**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 erschweren. 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.

**Minimale Dichte** Die Dichtewerte der Voxel im Invariantenraum können sich extrem unterscheiden. Abhängig von der Größe der jeweiligen Voxel traten in Testdatensätzen Werte zwischen  $10^{-4}$  und  $10^2$  auf. Damit auch Bereiche mit sehr geringer Dichte sichtbar sind, ist es möglich, eine minimale Dichte anzugeben. Wenn die Dichte an einem Abtastpunkt geringer als dieser Wert ist, so wird stattdessen die minimale Dichte verwendet.

## 6.11 Das Intervallfenster

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

## 7 Ergebnisse

Die Visualisierung wurde auf eine Reihe von Testdatensätzen angewendet, um die Korrektheit und Qualität der Darstellung zu bewerten. In diesem Abschnitt werden die

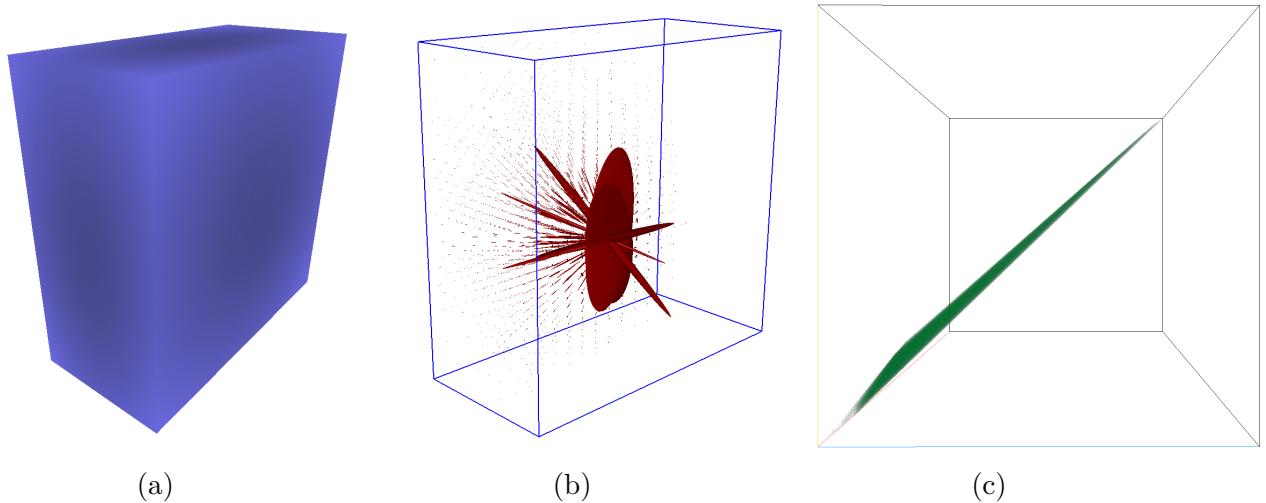


Abbildung 30: Darstellung des Single Point Load Datensatzes: (a) als Ellipsoide, (b) als DVR des Objekts und (c) im Invariantenraum, berechnet mit den K-Invarianten.

Datensätze vorgestellt, die Ergebnisse der Visualisierungen gezeigt und mit bestehenden Visualisierungen verglichen sowie mögliche Interpretationen erläutert.

Als Vergleichsdarstellung, und um eine Vorstellung über die Struktur der Datensätze zu vermitteln, wurde für die ersten beiden Datensätze 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.

## 7.1 Single Point Load

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

Ein DVR eines quaderförmigen Ausschnitts des Halbraums in der Nähe des Einwirkungspunktes der Kraft ist in Abb. 30a zu sehen. Diese Darstellung soll vor allem als Vergleichspunkt für die später gezeigten Visualisierungen des Halbraums dienen, bei denen Teile abhängig von den gewählten Invarianten ausgeblendet wurden.

Abb. 30b zeigt eine Darstellung der Tensoren als Ellipsoidglyphen. Deutlich erkennbar ist der Punkt an dem die Kraft einwirkt, dort sind die Glyphen am größten. Durch die Überdeckung der Glyphen ist es schwer zu erkennen an welchen Punkten die Glyphen genau eingezeichnet sind, es scheint jedoch so als ob die Matrizen nahe der Oberfläche stark planar anistrop sind, was durch flache runde Glyphen dargestellt ist. Weiter im Inneren ist die Form der Glyphen eher länglich, was auf lineare Anisotropie hinweist.

Für die Visualisierung wurde der I-Invariantensatz gewählt. Abb. 30c zeigt die Form des Datensatzes im I-Invariantenraum. Die Transferfunktion ordnet Bereichen hoher Dichte die Farbe grün und hohe Opazitätswerte zu, Bereichen geringer Dichte die Farbe rot und geringe Opazitätswerte. Da der I-Invariantensatz keiner der Invariantensätze ist, die sich für die Darstellung in zylindrischen Koordinaten eignen, wurden ein kartesisches Koordinatensystem und das quadratische Interaktionswidget verwendet.

Durch die Normalisierungsfunktion wurden die Dimensionen des Invariantenraumes so skaliert, dass die Intervalle der drei Invarianten gleich groß dargestellt werden. Tatsächlich sind die Intervalle jedoch wie folgt:

- $I_1$ : Spur, im Intervall [0, 00916; 44, 035]
- $I_2$ : Summe der Hauptminoren, im Intervall [-3, 195; 345, 12]
- $I_3$ : Determinante, im Intervall [-3, 301; 751, 25]

Die Form des DVR im Invariantenraum ist länglich und befindet sich entlang einer imaginären Gerade. Scheinbar besteht im Datensatz eine starke lineare Abhängigkeit zwischen den drei Invarianten. Dies ist teilweise durch die fehlende Orthogonalität des I-Invariantensatzes zu erklären: Desto höher die Eigenwerte einer Matrix, desto höher sind auch die Werte der I-Invarianten.

Da nur für  $I_1$  eine eindeutige Interpretation in der Mechanik existiert, beschränkt sich die Demonstration der Funktion des Interaktionswidgets in diesem Datensatz auf  $I_1$ .

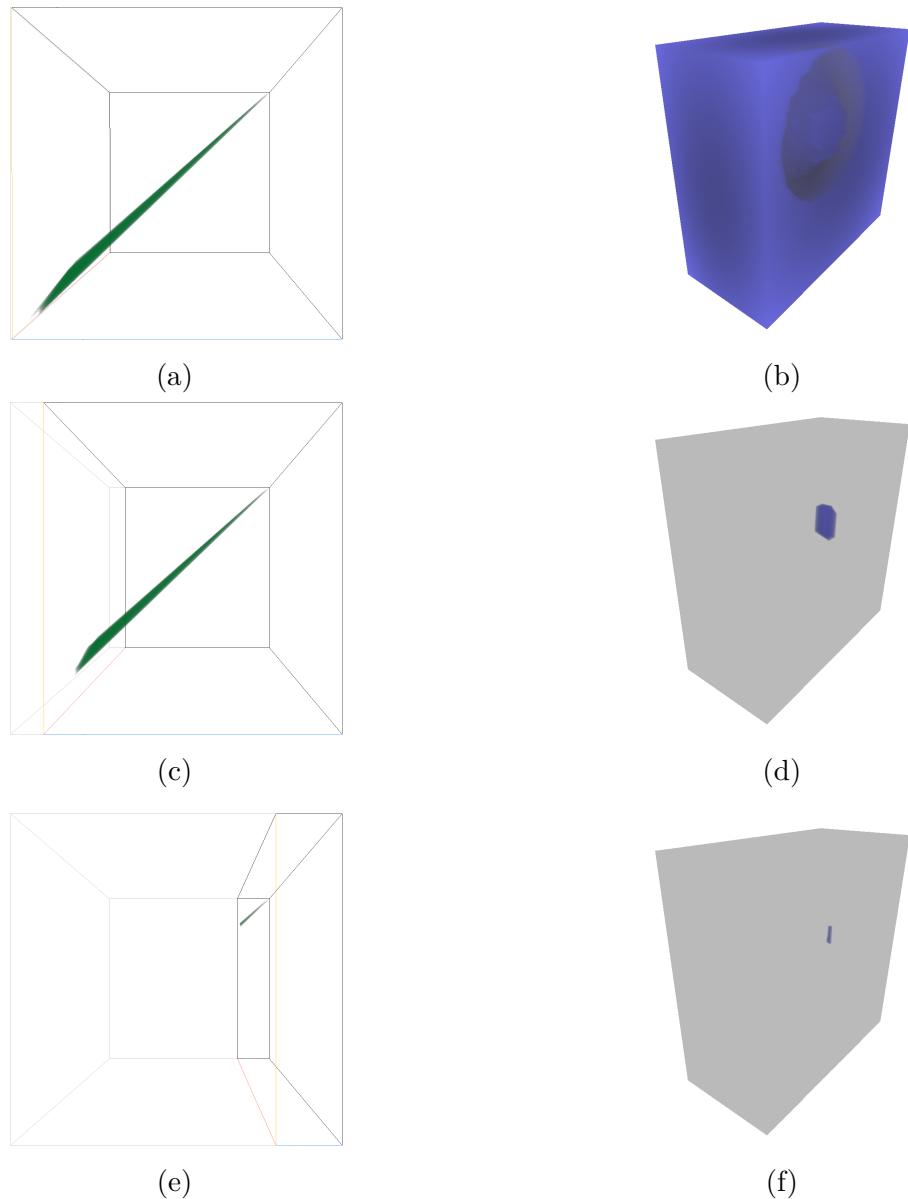


Abbildung 31: Die mechanische Spannung des Single Point Load-Datensatzes mit ausgewählten Invariantenbereichen. (a) und (b) zeigen Bereiche mit hohem  $K_1$ , (c) und (d) Bereiche mit hohem  $K_2$ , (e) und (f) Bereiche mit hohem  $K_3$ .

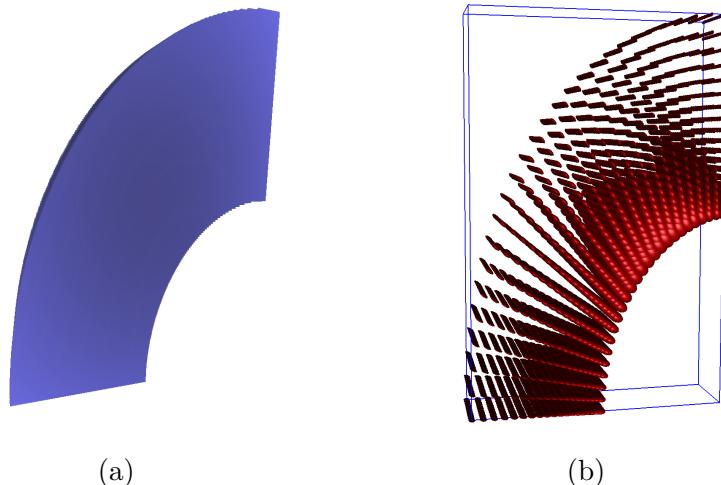


Abbildung 32: Der Metallscheiben-Datensatz: (a) DVR des Objektes, (b) Tensorglyphen der mechanischen Verformung.

## 7.2 Metallscheibe

Als zweites Beispiel wurden die Ergebnisse einer thermo-mechanischen Simulation einer Metallscheibe, durchgeführt von Dr. Thomas Nagel, verwendet. Dieselben Daten bilden auch die Grundlage für ein Anwendungsbeispiel in der Arbeit von Fritzsch [14, S.15, 45 ff.]. Dort werden die Datensätze von Fritzsch wie folgt beschrieben:

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

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

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. Die Metallscheibe sowie Tensorglyphen der beiden Datensätze sind in Abb. 32 dargestellt.

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

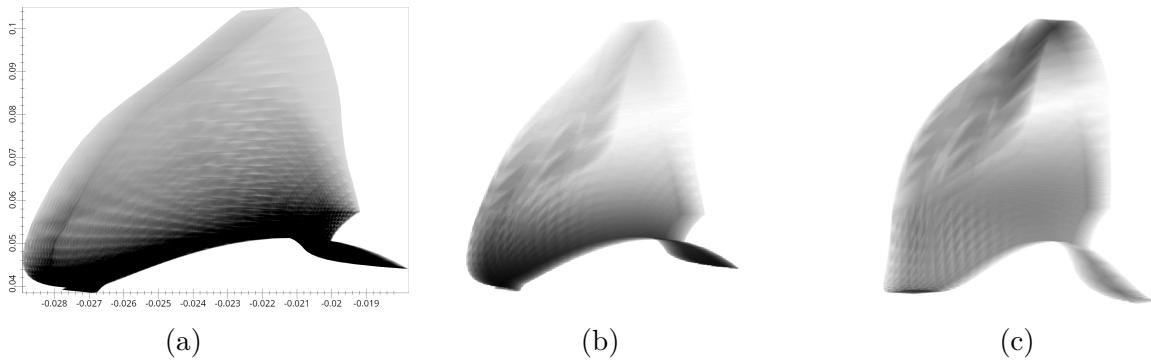


Abbildung 33: Darstellungen des Invariantenraums des K-Invariantensatzes für die mechanische Verformung in der Simulation der Metallscheibe: (a) Die von Fritzsch vorgestellte Visualisierung [14, S. 45], (b) ein DVR mit orthogonaler Projektion in kartesischen Koordinaten, (c) ein DVR mit perspektivischer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

Abb. 32b zeigt eine Darstellung der Verformungsmatrizen als Ellipsoidglyphen. Am inneren Rand der Scheibe, insbesondere im oberen Bereich, sind die Ellipsoide am größten. Dies lässt auf einen hohen Betrag der Eigenwerte der Matrizen dort schließen. Das Vorzeichen der Eigenwerte lässt sich jedoch nicht daraus ableiten. Zum äusseren Rand hin nimmt die Größe der Ellipsoide ab. Eine weitere Eigenschaft lässt sich aus der Form der Ellipsoide ablesen. Diese scheint am unteren Ende des Scheibenausschnitts eher länglich, am oberen Ende eher rund zu sein. Scheinbar ist die Verformung am oberen Ende stärker isotrop als am unteren.

Abb. 33a zeigt die von Fritzsch erzeugte Visualisierung der mechanischen Verformung. Die x-Achse entspricht dem Wert von  $K_1$ , die y-Achse dem von  $K_2$ .

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

Die Abbildungen 33a und 33b ähneln sich stark, weisen jedoch eine Reihe deutlicher Unterschiede auf. Zunächst ist die Abb. 33a deutlich breiter, was an einer unterschiedlichen Skalierung in Richtung der x-Achse liegt. Des Weiteren ist der rechte obere Teil von Abb. 33b deutlich heller als der von 33a. Das ist darauf zurückzuführen, dass die Tetraeder in diesem Bereich sehr flach und fast parallel zur xy-Ebene sind. Dadurch werden sie nur von wenigen oder gar keiner der bei der Rasterisierung verwendeten Schnittebenen getroffen, wodurch nur wenige Voxel Dichtewerte erhalten. Durch die Skalierung der Dichte mit

der Anzahl der Abtastpunkte wird der Bereich deutlich heller als in der Darstellung von Fritzsch. Ein weiterer wichtiger Faktor ist, dass das von Fritzsch implementierte Verfahren die Dichtewerte der projizierten Tetraeder einfach aufaddiert. Im DVR würde dies dem Röntgenbildverfahren entsprechen. Da stattdessen ein transluzentes Verfahren implementiert wurde, erscheinen dichte Bereiche wie z.B. an der unteren Kante des DVR heller als dieselben Stellen in der Darstellung von Fritzsch.

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

Für alle vom Invariantenraum der Metallscheibe erzeugten DVR wurden die Intervalle der Invarianten normalisiert. Die tatsächlichen Werte sind:

- $K_1$ : Spur, im Intervall  $[0, 00916; 44, 035]$
- $K_2$ : Norm des Deviators, im Intervall  $[-3, 195; 345, 12]$
- $K_3$ : Determinante, im Intervall  $[-3, 301; 751, 25]$

Abb. 34 demonstriert die Funktionsweise des zylindrischen Interaktionswidgets. Abb. 34a, 34c und 34e zeigen Darstellungen der Metallscheibe im Invariantenraum. Links und rechts des DVR sind die Interaktionswidgets eingeblendet. Durch Interaktion wurden die Intervalle der anzuseigenden Invarianten begrenzt, was durch Position und Form der Interaktionswidgets angezeigt wird. Teile der DVR, die ausserhalb dieser Intervalle liegen, werden ausgeblendet. Abb. 34b, 34d und 34f zeigen die dazugehörenden Darstellungen der Metallscheibe, bei der die Bereiche mit Invarianten ausserhalb des ausgewählten Bereichs ebenfalls ausgeblendet sind.

! 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. 32b). !

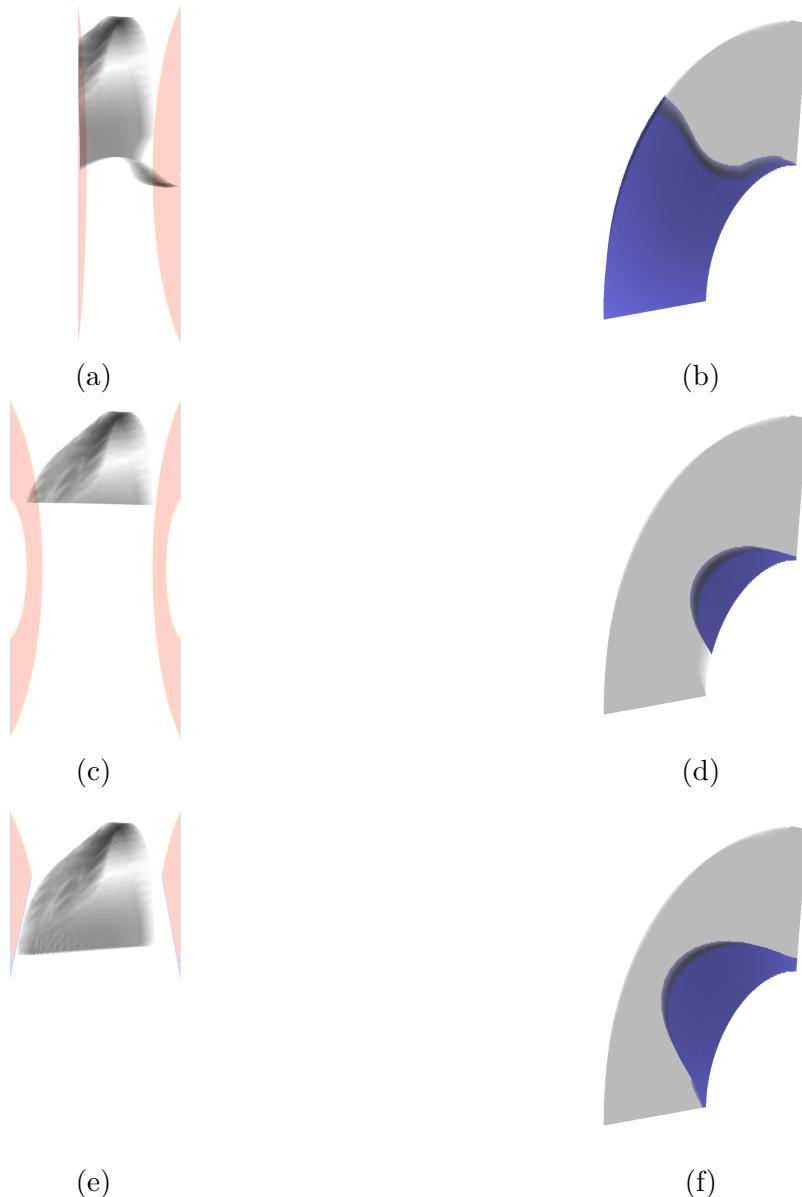


Abbildung 34: Die Verformung des Metallscheiben-Datensatzes mit ausgewählten Invariantenbereichen. (a) und (b) zeigen Bereiche mit hohem  $K_1$ , (c) und (d) Bereiche mit niedrigem  $K_2$ , (e) und (f) Bereiche mit hohem  $K_3$ .

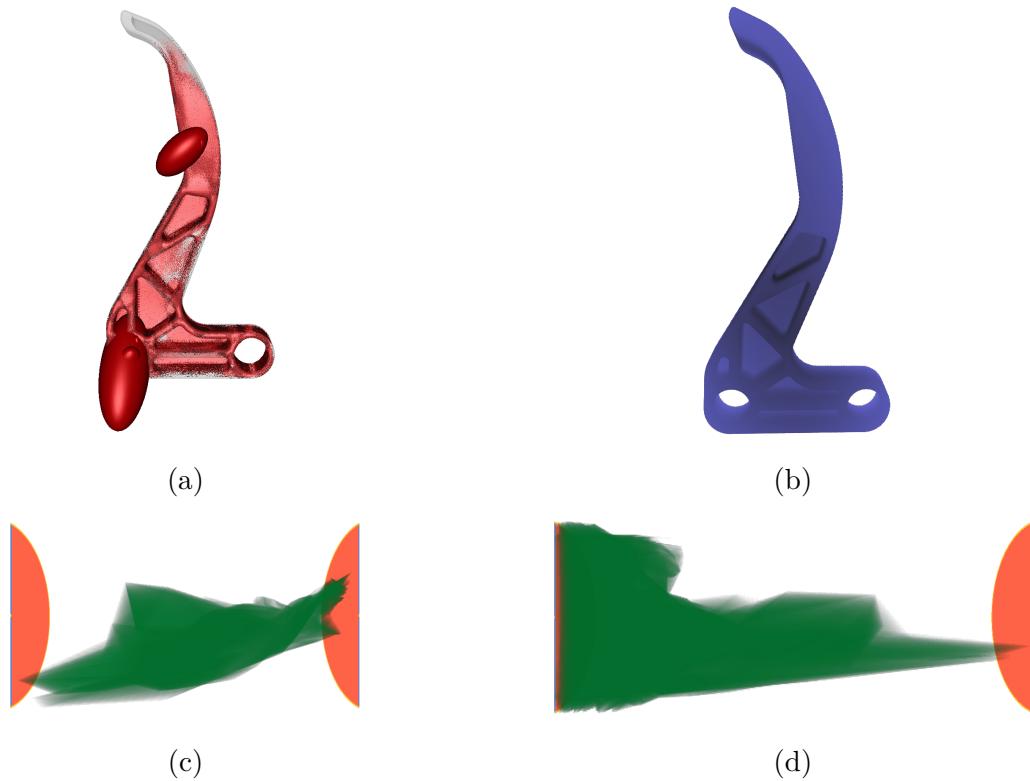


Abbildung 35: Darstellung des Bremshebel-Datensatzes: (a) als Ellipsoide, (b) als DVR des Objekts, (c) im K-Invariantenraum und (d) im R-Invariantenraum.

### 7.3 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. 35a 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][34] 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.

### 7.3.1 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. 37a und 37b 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. 36c und 36d).

Bereiche mit niedrigem Modus, also mit planarer Anisotropie, sind in Abb. 36e 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.

### 7.3.2 R-Invarianten

Trotz der sehr unterschiedlichen Form des Feldes im Invariantenraum (Abb. 35d), 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. 37e) sind, da die Ausgangsdaten die gleichen sind, natürlich identisch.

Gerade in Abb. 37d zeigt sich ein weitere Vorteil der implementierten Visualisierung: Die Bereiche mit hohem  $R_2$  sind überall im Objekt verteilt. Diese Bereiche klar darzustellen

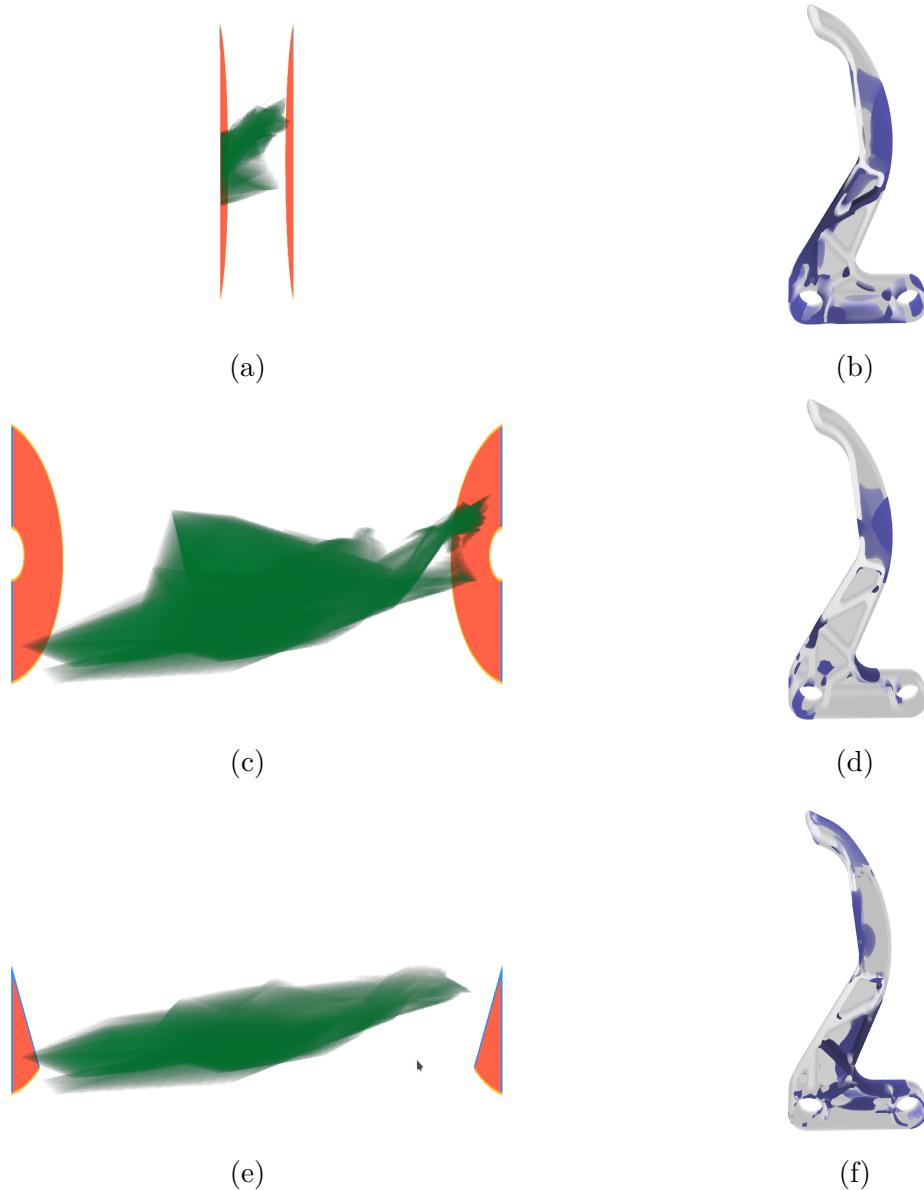


Abbildung 36: Die mechanische Spannung des Bremshebel-Datensatzes mit ausgewählten Invariantenbereichen. (a) und (b) zeigen Bereiche mit hohem  $K_1$ , (c) und (d) Bereiche mit hohem  $K_2$ , (e) und (f) Bereiche mit niedrigem  $K_3$ .

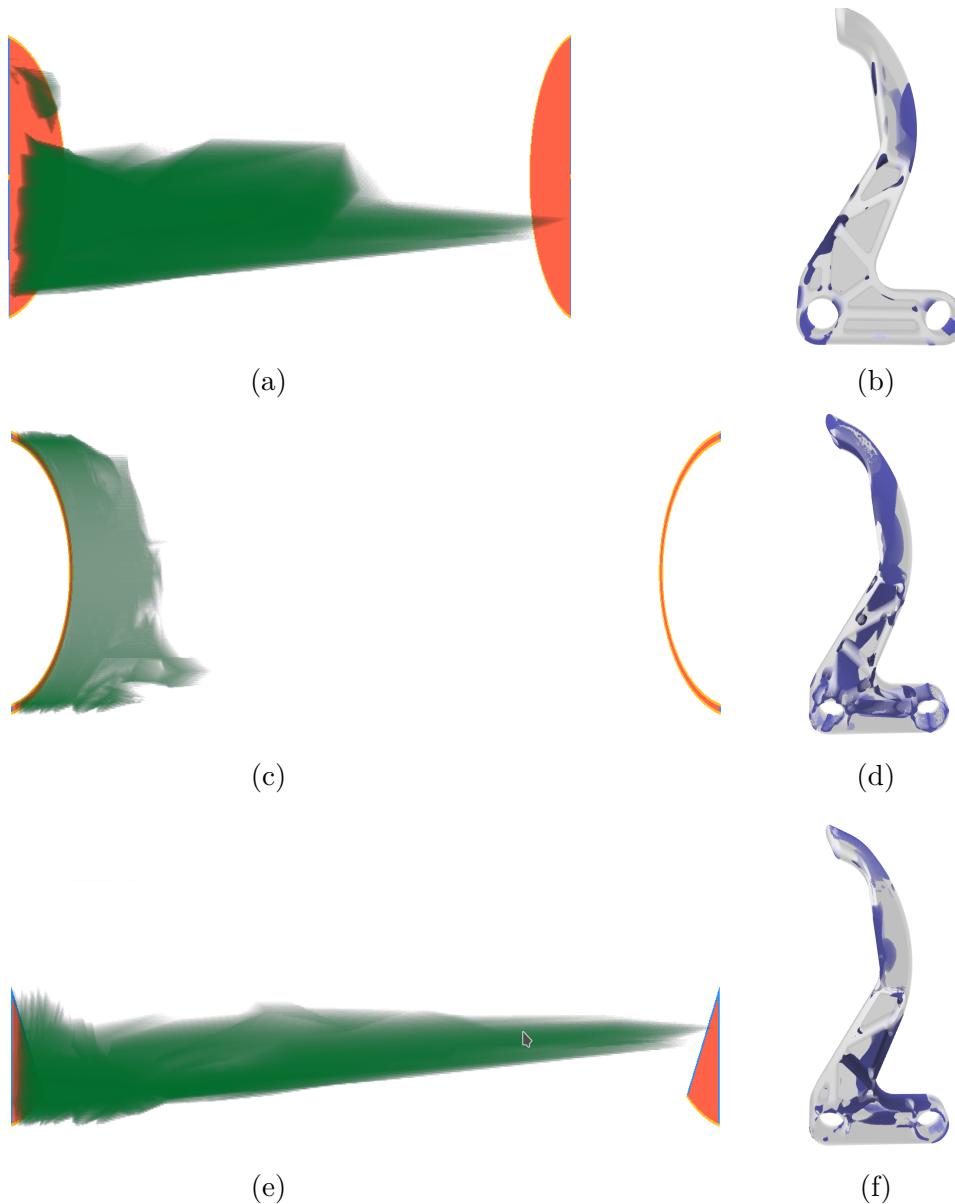


Abbildung 37: Die mechanische Spannung des Bremshebel-Datensatzes mit ausgewählten Invariantenbereichen. (a) und (b) zeigen Bereiche mit hohem  $R_1$ , (c) und (d) Bereiche mit hohem  $R_2$ , (e) und (f) Bereiche mit niedrigem  $R_3$ .

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

## 7.4 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 im Kapitel [2 Verwandte Arbeiten](#) erwähnten klassischen Tensorfeldvisualisierungen verwenden die Eigenwerte und -vektoren um Tensoren darzustellen. Der Betrag eines einzelnen Eigenwertes liefert ist 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.

## 8 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 sogenannte 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. Auch Probleme wie dünne, ebenenparallele Bereiche könnten durch Verbesserungen der Rasterisierung behoben werden.

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

Zuletzt bieten auch die Interaktionswidgets Raum für Weiterentwicklung, besonders in Hinblick auf Affordances, Hervorhebung der Handles bei Selektion und der Darstellung der Widgets in abhängig von den Farben der Transferfunktion.

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 DVR und die aus dem Continuous Scatterplotting übernommenen Verfahren legen innere Strukturen der Datensätze offen.

## Literatur

- [1] Universität Leipzig Abteilung für Bild- und Signalverarbeitung. *FAnToM Website*. URL: <http://www.informatik.uni-leipzig.de/fantom/content/about-fantom> (besucht am 25.03.2018).
- [2] Lisa S Avila u. a. *The VTK User's Guide*. Kitware New York, 2010.
- [3] Sven Bachthaler und Daniel Weiskopf. "Continuous scatterplots". In: *IEEE transactions on visualization and computer graphics* 14.6 (2008), S. 1428–1435.
- [4] Peter J Bassler, James Mattiello und Denis LeBihan. "MR diffusion tensor spectroscopy and imaging". In: *Biophysical journal* 66.1 (1994), S. 259–267.
- [5] Ray M Bowen und Chao-Cheng Wang. *Introduction to vectors and tensors*. Bd. 1. Courier Corporation, 2008.
- [6] The Qt Company. *Qt Website*. URL: <https://www.qt.io/> (besucht am 28.03.2018).
- [7] John C Criscione u. a. "An invariant basis for natural strain which yields orthogonal stress response terms in isotropic hyperelasticity". In: *Journal of the Mechanics and Physics of Solids* 48.12 (2000), S. 2445–2465.
- [8] Timothy J Cullip und Ulrich Neumann. "Accelerating volume reconstruction with 3D texture hardware". In: (1993).
- [9] Thierry Delmarcelle und Lambertus Hesselink. "Visualizing second-order tensor fields with hyperstreamlines". In: *IEEE Computer Graphics and Applications* 13.4 (1993), S. 25–33.
- [10] Robert A Drebin, Loren Carpenter und Pat Hanrahan. "Volume rendering". In: *ACM Siggraph Computer Graphics*. Bd. 22. 4. ACM. 1988, S. 65–74.
- [11] Daniel B Ennis und Gordon Kindlmann. "Orthogonal tensor invariants and the analysis of diffusion tensor magnetic resonance images". In: *Magnetic resonance in medicine* 55.1 (2006), S. 136–146.
- [12] Richard P Feynman, Robert B Leighton und Matthew Sands. *The Feynman lectures on physics, Vol. I: The new millennium edition: mainly mechanics, radiation, and heat*. Bd. 1. Basic books, 2011.
- [13] Gideon Frieder, Dan Gordon und R Anthony Reynolds. "Back-to-front display of voxel based objects". In: *IEEE Computer Graphics and Applications* 5.1 (1985), S. 52–60.
- [14] Clemens Fritzsch. "Visuelle Analyse kontinuumsmechanischer Simulationen durch kontinuierliche Streudiagramme". Diplomarbeit. Universität Leipzig, 2016.
- [15] Charles D Hansen und Chris R Johnson. *The Visualization handbook*. Elsevier, 2005.
- [16] Keith D. Hjelmstad. *Fundamentals of Structural Mechanics*. 10. Aufl. Springer US Verlag KG, 2005. ISBN: 978-3-13-477010-0.

- [17] Mario Hlawitschka u. a. “Top Challenges in alization of Engineering Tensor Fields”. In: *Visualization and Processing of Tensors and Higher Order Descriptors for Multi-Valued Data*. Springer, 2014, S. 3–15.
- [18] The Khronos™ Group Inc. *OpenGL Website*. URL: <https://www.khronos.org/opengl/> (besucht am 28.03.2018).
- [19] Michael I Jordan und Robert A Jacobs. “Hierarchical mixtures of experts and the EM algorithm”. In: *Neural computation* 6.2 (1994), S. 181–214.
- [20] Gordon Kindlmann. “Superquadric tensor glyphs”. In: *Proceedings of the Sixth Joint Eurographics-IEEE TCVG conference on Visualization*. Eurographics Association. 2004, S. 147–154.
- [21] Gordon Kindlmann und David Weinstein. “Hue-balls and lit-tensors for direct volume rendering of diffusion tensor fields”. In: *Proceedings of the conference on Visualization'99: celebrating ten years*. IEEE Computer Society Press. 1999, S. 183–189.
- [22] Gordon Kindlmann u. a. “Diffusion tensor analysis with invariant gradients and rotation tangents”. In: *IEEE Transactions on Medical Imaging* 26.11 (2007), S. 1483–1499.
- [23] Norbert Kusolitsch. *Maß-und Wahrscheinlichkeitstheorie: Eine Einführung*. Springer-Verlag, 2014.
- [24] Bruce R Kusse und Erik A Westwig. *Mathematical physics: applied mathematics for scientists and engineers*. John Wiley & Sons, 2010.
- [25] Philippe Lacroute und Marc Levoy. “Fast volume rendering using a shear-warp factorization of the viewing transformation”. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM. 1994, S. 451–458.
- [26] Ulrich Leiner, Bernhard Preim und Stephan Ressel. “Entwicklung von 3D-Widgets-Überblicksvortrag”. In: *Proceedings of Simulation und Animation: SCS Europe, Erlangen, S* (1997), S. 170–188.
- [27] Reiner Lenz u. a. “Display of Density Volumes”. In: *IEEE Computer Graphics and Applications* 6.7 (1986), S. 20–29. DOI: [10.1109/MCG.1986.276813](https://doi.org/10.1109/MCG.1986.276813). URL: <https://doi.org/10.1109/MCG.1986.276813>.
- [28] Tamara Munzner. *Visualization analysis and design*. AK Peters/CRC Press, 2014.
- [29] Bernhard Preim und Raimund Dachselt. *Interaktive Systeme: Band 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung*. Springer-Verlag, 2010.
- [30] Bernhard Preim und Raimund Dachselt. *Interaktive Systeme: Band 2: User Interface Engineering, 3D-Interaktion, Natural User Interfaces*. Springer-Verlag, 2015.
- [31] Antonio J Rueda u. a. “Voxelization of solids using simplicial coverings”. In: (2004).
- [32] Gerik Scheuermann und Matthias Goldau. *Vorlesung Visualisierung in Naturwissenschaft und Technik*. 2015.

- [33] Peter Shirley und Allan Tuchman. *A polygonal approximation to direct scalar volume rendering*. Bd. 24. 5. ACM, 1990.
- [34] Dassault Systèmes. *Abaqus Website*. URL: <https://www.3ds.com/products-services/simulia/products/abaqus/> (besucht am 26.04.2018).
- [35] Lee Westover. “Footprint evaluation for volume rendering”. In: *ACM Siggraph Computer Graphics* 24.4 (1990), S. 367–376.
- [36] Lee Westover. “Interactive volume rendering”. In: *Proceedings of the 1989 Chapel Hill workshop on Volume visualization*. ACM. 1989, S. 9–16.
- [37] Alexander Wiebel u. a. “Fantom-lessons learned from design, implementation, administration, and use of a visualization system for over 10 years”. In: (2009).
- [38] Valentin Zobel und Gerik Scheuermann. “Extremal curves and surfaces in symmetric tensor fields”. In: *The Visual Computer* (2017), S. 1–16.