Masterarbeit

Schematische 2D Visualisierung von RNA-Protein Interaktionen

Niklas Teichmann

13. April 2018

Zusammenfassung: Die Visualisierung von Tensorfeldern ist trotz vielen Jahrzehnten aktiver Forschung noch immer ein aktives Thema. Komplexe Strukturen innerhalb von Tensorfeldern stellen hohe Anforderungen an Visualisierungssoftware, um die Interpretation durch Benutzer zu erleichtern. Innerhalb dieser Arbeit wurde eine Erweiterung für die Visualisierungssoftware 'FAnToM' entwickelt, die Techniken aus dem Direct Volume Rendering und dem Continuous Scatterplotting verwendet, um Invarianten von symmetrischen Tensoren 2. Grades und ihre Verteilung innerhalb eines Datensatzes darzustellen. Dabei wurde besonderer Wert auf Interaktivität gelegt, um Nutzern die explorative Analyse der Daten zu ermöglichen.

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

Leipzig, d. _____ _____

Ort, Datum Matrikelnummer: 2878372 Unterschrift

Inhaltsverzeichnis

1 Einleitung

Tensorfelder sind eine sowohl in der Forschung als auch in der Praxis häufig vorkommende Art von Datensäetzen. Beispiele dafür sind die Diffusions-Tensor-Bildgebung[4], welche die Diffusion von Wasser in Gewebe wie z.B. dem Hirn untersucht, oder Verformungs-[13, S. 122] und Spannungstensoren[13, S. 154] in der Mechanik. Ein häufig gewählter Ansatz, um die Interpretation von Tensordaten zu erleichtern, ist die Tensorfeldvisualisierung, ein Teilgebiet der wissenschaftlichen Visualisierung, das sich mit der Erzeugung von für Menschen verständlichen visuellen Repräsentationen von Tensorfeldern beschäftigt. Auch diese muss sich jedoch den genannten Herausforderungen stellen. Dabei treten jedoch eine Reihe von Problem auf, von denen einige im Folgenden genannt werden[14][12]:

Menge an Daten pro Tensor Ein einzelner Tensor kann, abhängig von Grad und Dimension, beliebig viele Datenwerte umfassen. Aber selbst ein Tensor 2. Grades und 3. Dimension besteht bereits aus 9 Werten, für die eine passende Darstellung gefunden werden muss.

Menge an Daten pro Datensatz Häufig enthalten Datensätze Tausende oder Millionen von Tensoren, was gut skalierende Werzeuge zur Analyse notwendig macht. Zudem bestehen relevante Merkmale der Datensätze oft nur aus wenigen Datenpunkten, weshalb effektives Filtern wichtig wird.

Fehlende Intuition Tensoren beschreiben im Allgemeinen lineare Abhängigkeiten zwischen Skalaren, Vektoren und anderen Tensoren. Während bei niedrigen Rängen (Skalare/Vektoren) noch intuitive Interpretation existieren (Zahlenwert/Punkt im dreidimensionalen Raum), fällt es Menschen erheblich schwieriger, 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 besonere Bedeutung sind [14, S. 4]. Eine Anwendung zu entwickeln, die über Domänen hinweg verwendbar ist, ist daher schwer.

Im Zuge der vorliegenden Arbeit wurde ein neues Verfahren zur Visualisierung von symmetrischen Tensoren 2. Grades im dreidimensionalen Raum entwickelt und als Erweiterung der Visualisierungssoftware ‘FAnToM’ implementiert. Im Speziellen wurden Invariantenfelder aus den der 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 Definitionen und Grundlagen aus der Mathematik und Kontinuumsmechanik vorgestellt, die in dieser Arbeit verwendet wurden. Als nächstes werden in 4 Verwendetete Verfahren und Technologien Programmschnittstellen und Visualisierungstechniken erläutert, die innerhalb dieser Arbeit verwendet wurden. Insbesondere wird dort auch auf FAnToM als Softwaregrundlage eingegangen. Kapitel 5 Umsetzung beschreibt die konkrete Umsetzung der FAnToM-Erweiterung mit allen implementierten Funktionen. Nachfolgend werden in Kapitel 6 Ergebnisse wird die entwickelte Erweiterung exemplarisch auf einige Datensätze angewendet, und die Ergebnisse diskutiert. Zum Abschluss werden in Kapitel 7 Ausblick im Verlauf dieser Arbeit neu aufgetretene Problemstellungen sowie weitere Verbesserungsmöglichkeiten erörtert.

2 Verwandte Arbeiten

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

Eine relative einfache Darstellung eines Vektorfelds besteht darin, die einzelnen Komponenten eines Tensor 2. Grades als Skalarfelder aufzufassen und als Grauwertbild zu zeichnen. Dabei ist der Grauwert an einem Datenpunkt abhängig vom Verhältniss des Wertes der Komponente des Vektors zu dem höchsten Wert dieser Komponente im Datensatz. Die früheste gefundene Erwähnung dieses Verfahrens ist in einem Paper von Kindlmann und Weinstein aus dem Jahre 1999 [17], in dem es jedoch als weder neu noch besonders intuitiv beschrieben wird.

Die wohl verbreiteteste Art von Tensorvisualisierungen sind die glyphenbasierten Verfahren. Diese Verfahren beschränkt 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 odern oder Superquadrics[16], 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 Benutzer ein Bild von der Verteilung und Struktur der Tensoren im Datensatz. Glyphenbasierte Verfahren sind besonders

in der Medizin beliebt, da sie leicht Rückschlüsse auf die Richtung von Nerven- und Muskelfasern zulassen.

Hyperstreamlines[8] bilden ein Analogon zu den Stromlinien bei Vektorfeldern. Als Eingabedaten sind nur Felder von reellen, symmetrischen, dreidimensionalen Tensoren 2. Grades mit nichtnegativen Eigenwerten zugelassen, da so sichergestellt wird, dass die Eigenwerte ganzzahlig und größer 0 sind sowie die Eigenvektoren paarweise orthogonal zueinander. Das Verfahren zeichnet ausgehend von festgelegten Punkten Schläuche durch das Vektorfeld. Die Mittellinien dieser Schläuche entsprechen dabei Stromlinien, deren Richtung vom Eigenvektor mit dem höchsten Eigenwert abhängt. Der Durchschnitt durch den Schlauch, orthogonal zur Mitellinie ist stets eine Ellipse, deren Halbachsen den zwei kleineren Eigenwerten entsprechen. Da so aber Informationen über den Wert des größten Eigenwertes verloren geht, werden einzelne Stücken des Schlauchs abhängig von diesem Wert eingefärbt.

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

3 Grundlagen

3.1 Mathematische Grundlagen

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

Eine erheblicher Teil der verwendeten Formeln und Definitionen stammen 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 $f : A \to B$, die Objekten aus der Menge A Objekte aus B zuordnet, lässt sich das Bild einer Menge $A' \subset A$ als

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

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

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

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

3.1.2 Einschränkung einer Funktion

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

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

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

3.1.3 Dirac Delta

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

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

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

Da diese beiden Eigenschaften für die vorliegende Arbeit ausreichen wird auf eine genaue Definition von δ verzichtet. Diese kann jedoch in [19, S. 100 ff.] nachgelesen werden.

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

Das n-dimensionale Volumen einer Menge $Vol(A), A \subset \mathbb{R}^n$ wird über das Lebesgue-Stieltjes Maß definiert[18]. Intuitiv entspricht es in $\mathbb{R}$ der Länge, in $\mathbb{R}^2$ dem Flächeninhalt und in $\mathbb{R}^3$ dem Volumen.

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

3.1.5 Jacobi Matrix

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

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

Sie entspricht damit der ersten Ableitung in der mehrdimensionalen Analysis. Die Determinante der Jacobi-Matrix $det(J_f)$ wird auch als Funktionaldeterminante bezeichnet, und beschreibt einige Eigenschaften der Funktion f. Die für diese Arbeit wichtigste Rolle spielt der der absolute Wert $|det(J_f)|$ an einem Punkt p, die die Expansion bzw. das Schrumpfen der Funktion in der Nähe von p beschreibt. Für eine lineare Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$, deren Funktionaldeterminante in jedem Punkt $p_n \in \mathbb{R}$ gleich ist, bedeutet das mit dem n-dimensionalen euklidischen Abstand $||p_1, p_2||_n$

$$||f(p_1), f(p_2)||_n = det(J_f) \cdot ||p_1, p_2||_n \tag{7}$$

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

3.1.6 Lineare Abbildungen

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

$$\begin{aligned} \varphi(u + v) &= \varphi(u) + \varphi(v) \\ \varphi(\lambda v) &= \lambda \cdot \varphi(v) \end{aligned} \tag{8}$$

also ϕ ein homogener Homomorphismus ist.

Die Menge aller linearen Abbildungen von V nach U wird mit $L(V, U)$ bezeichnet. [5, S. 97].

3.1.7 Dualraum

Sei V ein Vektorraum und K sein zugrundeliegender Körper. Dann bildet die Menge der linearen Abbildungen der Form $\varphi : V \to K$ wiederum einen Vektorraum V^*. Dieser 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 $\varphi : V_1 \times \cdots \times V_n \to K$, wobei jedes V_i ein Vektorraum über K ist, und zusätzlich

$$\varphi(\lambda \cdot v_1 + \mu \cdot v_1', \ldots, v_n) = \lambda \cdot \varphi(v_1, \ldots, v_n) + \mu \cdot \varphi(v_1', v_2, \ldots, v_n) \tag{9}$$

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

3.1.9 Tensor

Multilineare Funktionen der Form $T : V^* \times \cdots \times V^* \times V \times \cdots \times V \to K$, 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 bildet wiederum einen Vektorraum [5, S. 220]. Durch diese Defintion ist ein Tensor immer invariant zur Basis des Vektorraums seiner Variablen. Egal in welche Basis er umgerechnet wird, er drückt steht dasselbe aus.

In der vorliegenden Arbeit werden ausschließlich Tensoren 2. Grades verwendet, die in kartesischen, dreidimensionalen Koordinatensystemen erzeugt wurden. Dabei ist zu beachten, dass in kartesischen Koordinaten die Basis eines Vektorraumes V und seines Dualraumes V^* die gleiche Darstellung hat, also V und V^* austauschbar sind. Wenn in einen Tensor 2. Grades T die Basisvektoren $e_{1,\ldots,d}$ 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:

$$c_{i,j} = \sum_{i=1}^{d} \sum_{j=1}^{d} T(e_i, e_j), \tag{10}$$

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

$$m \cdot v = u \tag{11}$$

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

3.1.10 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 der linear unabhängigen 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.

3.1.11 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

$$tr(A) = \sum_{i=1}^{n} aii \tag{12}$$

also die Summe aller Elemente in der Hauptdiagonale. Eine wichtige Eigenschaft der Spur ist, dass bei der Überführung einer Matrix in eine andere Basis gleich bleibt (siehe auch 3.1.17).

3.1.12 Norm einer Matrix

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

1. $f(kv) = |k|f(v)$ (Absolute Homogenität)
2. $f(u + v) \leq f(u) + f(v)$ (Erfüllung der Dreiecksungleichung)
3. $f(v) = 0 \iff v = 0$ ist der Nullvektor (Definitheit)

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

$$norm(A) = \sqrt{tr(AA^T)} \tag{13}$$

Die euklidische Norm wird auch als 'Frobenius Norm' bezeichnet.

3.1.13 Deviator einer Matrix

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

$$\tilde{A} = A - \bar{A} \tag{14}$$

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

$$\bar{A} = \frac{1}{3} tr(A) I \tag{15}$$

wobei I die Matrixdarstellung des Einheitstensor ist.

3.1.14 Modus einer Matrix

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

$$mode(A) = 3\sqrt{6}\ det(A \backslash norm(A)) \tag{16}$$

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

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

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

Um die Intuition hinter dem Modus zu verdeutlichen, sind in Abb. 1 Superquadrics von Matrizen unterschiedlicher Modi dargestellt. Insbesondere wird dadurch betont, dass der Modus nicht von der Größe der Eigenwerte abhängt, sondern von deren Verteilung.

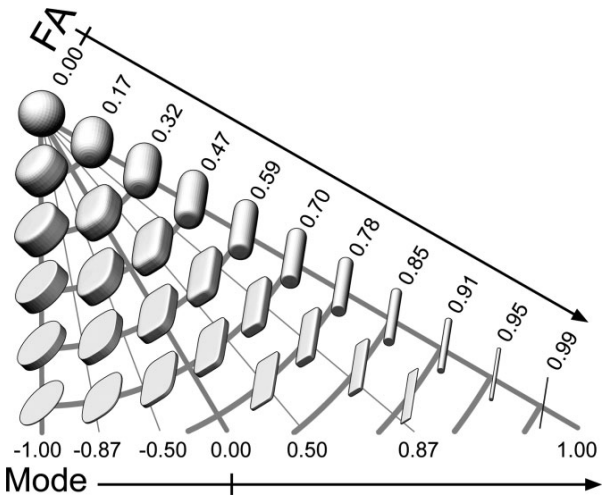


Abb. 1: Darstellung der fraktionalen Anisotropie und des Modus von Matrizen in Form von Superquadrics[16]. Die Fraktionale Anisotropie nimmt mit größerer Entfernung zum obersten linken Superquadric zu. Der Modus des Deviators wird abhängig vom Winkel dargestellt, wobei er links -1 beträgt und rechts 1. Entnommen aus [10, S. 140].

3.1.15 Gradient

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

$$grad(f) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \tag{17}$$

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

$$grad(g) = \begin{pmatrix} \frac{\partial f}{\partial a_{11}} & \cdots & \frac{\partial f}{\partial a_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial a_{m1}} & \cdots & \frac{\partial f}{\partial a_{mn}} \end{pmatrix} \tag{18}$$

wobei a_{ij} die Komponenten der Matrizen sind.

3.1.16 Orthogonalität von Matrizen

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

$$tr(U, V^T) = 0 \tag{19}$$

3.1.17 Matrixinvarianten

Als Invarianten werden zu mathematischen Objekten zugeordnete Größen bezeichnet, die invariant gegenüber der Anwendung bestimmten Transformationen auf die Objekte sind. Invarianten eines Tensors in Matrixdarstellung sind beispielsweise Größen, die sich unabängig von der Wahl der Basis der Matrix nicht verändern[10]. Eine Invariante ist somit eine Funktion $\Psi : M \to A$ 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 das vorliegende Paper auf symmetrische, dreidimensionale Tensoren 2. Grades und ausschließlich orthogonale Transformationen bezieht, werden die Invarianten eines Tensors T durch seine Eigenwerte vollständig charakterisiert. Invarianten sind in diesem Spezialfall also auch als Funktion $\Psi : \mathbb{R}^d \to \mathbb{R}$ der Form

$$\Psi : \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \to \mathbb{R} \tag{20}$$

darstellbar, wobei $\lambda_1, \lambda_2, \lambda_3$ die Eigenwerte von T sind. Die stimmt mit der Betrachtunsweise von Zobel und Scheuermann [22] überein.

3.1.18 Invariantensätze

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

Für 3×3 Matrizen enthalten alle orthogonalen Invariantensätze höechstens 3 Invarianten. In der Praxis spielen jedoch eine Vielzahl von Invariantensätzen eine Rolle, von denen im Folgendenden einige erläutert werden:

Die Eigenwerte Eigenvektoren v_i, $1 \leq i \leq 3$ einer 3×3 Matrix M sind vom Nullvektor verschiedene Vektoren, für die gilt

$$M \cdot v_i = \lambda_i v_i \tag{21}$$

Intuitiv bedeutet das, dass sich durch Multiplikation mit M ihre Richtung nicht verändert. Die zugehörigen λ_i werden als Eigenwerte bezeichnet und bilden einen orthogonalen Invariantensatz. Sie sind insbesondere in der Medizin sehr beliebt, da hohe Eigenwerte Hinweise auf Diffusionsbewegung in Gewebe liefern können.

Der I-Invariantensatz Das charakteristische Polynom χ einer 3×3 Matrix A hat die Form

$$\begin{aligned} \chi_A(\lambda) &= det(\lambda I - A) \\ \chi_A(\lambda) &= -\lambda^3 + I_1\lambda^2 - I_2\lambda + I_3 \end{aligned} \tag{22}$$

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

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

I bildet jedoch keinen orthogonalen Invariantensatz.

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:

- $J_1(A) = tr(\tilde{A})$ (Spur des Deviators von A)
- $I_2(A) = \frac{1}{2}(tr(\tilde{A})^2 - tr(\tilde{A}^2))$ (Summe der Hauptminoren des Deviators von A)
- $I_3(A) = det(\tilde{A})$ (Determinante des Deviators von A)

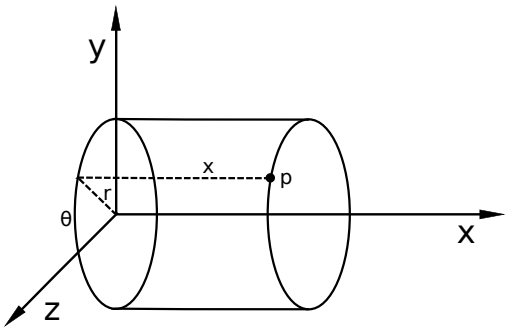


Abb. 2: In dieser Abbildung werden zylindrische Koordinaten innerhalb eines kartesischen Koordinatensystems dargestellt. x, y und z entsprechen dabei den kartesischen Achsen. Die Koordinaten des Punktes p sind angegeben als x, r, θ. x entspricht der kartesischen Koordinate, r ist die Entfernung zwischen der Projektion von p auf die von y und z Achse aufgespannte Fläche und dem Koordinatenursprung und θ entspricht dem Winkel zwischen Projektion von p, dem Koordinatenursprung und der x-Achse.

Dabei ist jedoch zu beachten, dass

$$\begin{aligned} tr(\tilde{A}) &= tr(A - \frac{1}{3}tr(A)I) \\ &= a_{11} - \frac{1}{3}tr(A) + a_{22} - \frac{1}{3}tr(A) + a_{33} - \frac{1}{3}tr(A) \\ &= a_{11} + a_{22} + a33 - tr(A) \\ &= tr(A) - tr(A) \\ &= 0 \end{aligned} \tag{23}$$

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

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

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

Da $K_1 \in [-\infty, \infty]$, $K_2 \in [0, \infty]$, $K_3 \in [-1, 1]$ bietet sich fr den K-Invariantensatz eine Darstellung in einem zylindrischen Koordinatensystem an, wobei K_1 eine Position auf

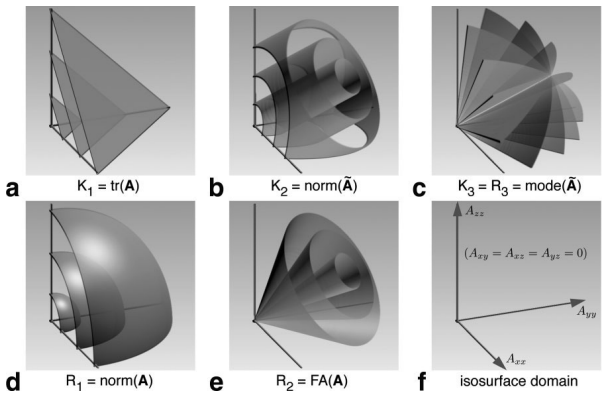


Abb. 3: Dargestellt sind Isoflächen der K und R Invarianten von diagonalisierten 3×3 Matrizen (alle Werte ausserhalb der Hauptdiagonale sind 0). Die Koordinatenachsen entsprechen den drei Eigenwerten der Matrizen. Entnommen aus [10, S. 139]

einer zentralen Achse beschreibt, K_2 die orthogonale Entfernung zu diesem Punkt und K_3 den Winkel zu einer festgelegten, zur zentralen Achse orthogonalen, zweiten Achse. Sowohl ein zylindrisches als auch ein kartesisches Koordinatensystem sind in Abbildung 2 dargestellt. K_1 entspricht dabei der zylindrischen x-Koordinate, K_2 dem Radius r und K_3 dem Winkel θ

Der R-Invariantensatz Der orthogonale R-Invariantensatz verwendet die Invarianten R_1, R_2 und R_3. Für eine Matrix A sind sie definiert als

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

Dabei fällt auf, dass $R_3(A) = K_3(A) = mode(A)$. Ähnlich wie der K-Invariantensatz kann auch R in einem zylindrischen Koordinatensystem dargestellt werden.

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

3.2 Mechanische Grundlagen

3.2.1 Physikalisches Feld

Ein physikalisches Feld ist eine physikalische Größe, die an verschiedenen Positionen im Raum unterschiedliche Werte annimmt[11, 1–2 Electric and magnetic fields]. Abstrahiert kann es als Funktion $\mathbb{R}^n \to V$ 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 $\mathbb{R} \to \mathbb{R}^{m\times n}$ 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 3 zueinander orthogonalen Ebenen durchgeführt werden, erhält man so drei Vektoren, die zusammen die Matrixdarstellung des Cauchy Tensors σ bilden:

$$\sigma = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} \tag{24}$$

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

3.2.3 Mechanische Verformung

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

$$\epsilon = \begin{bmatrix} \epsilon_x & \frac{1}{2}\gamma_{xy} & \frac{1}{2}\gamma_{xz} \\ \frac{1}{2}\gamma_{xy} & \epsilon_y & \frac{1}{2}\gamma_{yz} \\ \frac{1}{2}\gamma_{xz} & \frac{1}{2}\gamma_{yz} & \epsilon_z \end{bmatrix} \tag{25}$$

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

4 Verwendete Verfahren und Technologien

4.1 FAnToM

FAnToM[1][21] ('Field Analysis using Topological Methods') ist ein Programm, dessen Entwicklung im Jahre 1999 begann. Obwohl es zu Anfang noch als Hilfsmittel für die Analyse von Vektorfeldtopologien konzipiert war, entwickelte es sich über die Jahre hinweg zu einer Plattform für diverse Visualisierungen.

Erweiterungen von FAnToM werden in Form von sogenannten Algorithmen entwickelt. Dabei unterscheidet man 2 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 einzelnen Feld mit Vektorwerten kombiniert, deren Komponenten den Skalaren entsprechen. Visualization Algorithms dagegen dienen dazu, Visualisierungen zu erzeugen. Interaktionen mit der den Visualisierungen können entweder über die Maus oder über Optionen stattfinden.

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

Die aktuelle FAnToM-Session, also die momentan benutzten Algorithmen 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 hoch entwickelte (ToDo: anderes Wort) Datenmodell, das viele Funktionen zum Verarbeiten von Feldern und Tensoren bietet, die bestehenden Schnittstellen für Qt und OpenGL sowie die große Bibliothek von wiederverwendbaren Algorithmen.

4.2 OpenGL

OpenGL[15] ist eine Spezifikation die das Verhalten eines rasterbasierten Renderingsystems beschreibt. Sie definiert eine Schnittstelle, gegen die von zwei Seiten entwickelt werden kann: Zum einen von Seiten der Hardwarehersteller, die Funktionen von OpenGL auf ihrer Hardware implementieren. Zum anderen von Programmierern, die durch OpenGL unabhängig von der Hardware, auf der das Programm am Ende laufen soll, entwickeln können.

Die Verarbeitungsschritte von der Übergabe von Daten zu OpenGL bis zum fertig gerenderten Bild wird als Rendepipeline bezeichnet. Seit Version 2.0 unterstützt OpenGL sogenannte 'Shader', kleine Programmstücke, die es Benutzern von OpenGL ermöglicht, 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.

In der vorliegenden Arbeit wurden Vertex, Geometry und Fragment Shader entwickelt. Diese drei Typen sind folgend 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 beispielsweise 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 dazugehörenden Punkte, z.B. eine Liste von 3 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 bespielsweise aus einer Linie ein Dreieck erzeugen. Zusätzlich kann er pro Punkt Uniforms definieren, die an später ausgeführte Shader (Vertex Shader) weitergegeben werden. Pro Primitv wird der Geometry Shader einmal ausgeführt.

Fragment Shader Fragment Shader bestimmen die Farben der Fragmente. Fragmente sind Stücke der projizierten Primitve, deren Größe von der Art der Rasterisierung abhängen (z.B. Supersampling). Die Eingaben sind der Mittelpunkt des Fragments und die für diesen Punkt interpolierten Attribute der Eckpunkte des Primitivs. Neben der Fragmentfarbe können auch Werte ausgegeben werden, z.B. ein Tiefenattribut, das korrekte Überlappung von Primitiven ermöglicht.

Weitere im Folgenden verwendete Funktionen von OpenGL sind 'Blending', eine Funktion durch die Fragmente übereinander gezeichnet und ihre Farbwerte gewichtet kombininert werden können, was z.B. für transparente Objekte wichtig ist sowie die Möglichkeit, gerenderte Bilder 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, plattforübergreifende Bibliothek für die Entwicklung von Programmoberflächen. Die Popularität von Qt beruht auf der Plattformunabhängigkeit und der großen Vielfalt leicht zu verwendender Funktionen.

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

4.4 Direct Volume Rendering

In der Computergrafik werden dreidimensionale Objekte üblicherweise durch Dreiecke dargestellt, durch die die Oberfläche der Objekte approximiert werden. Dies entfernt jedoch alle Informationen über Strukturen im Inneren der Objekte, so wie Bereiche unterschiedlicher Materialien oder Dichte. Durch das Berechnen und Zeichnen von Isoflächen kann dieses Problem behoben werden, dies erfordert jedoch hohe Rechenzeiten und gute Kenntnis der Datensätze, um geeignete Isowerte festzulegen.

Eine Alternative bietet das Direct Volume Rendering[9] (im Folgenden DVR). Als Eingabe wird ein Feld $f : \mathbb{R}^3 \to \mathbb{R}$ verwendet, das Punkten im dreidimensionalen Raum skalare Werte zuordnet. Die Interpretation dieser Werte kann, abhängig vom Einsatzgebiet, unterschiedlich sein. Beispiele dafür sind die abgegebene Energie von Regionen in der

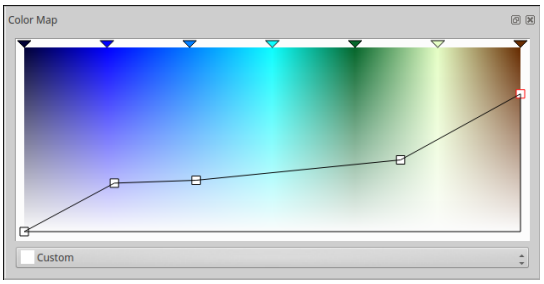


Abb. 4: Eine Implementierung einer interaktiven Transferfunktion in FAnToM.

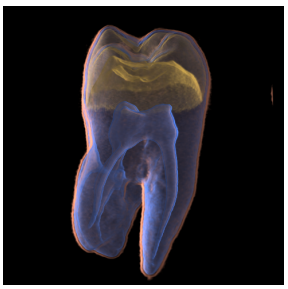


Abb. 5: Ein Direct Volume Rendering eines Zahns. Entnommen aus [9, S. 6]

Magnetresonanztomographie oder die Menge von absorbierter Röntgenstrahlung in der Computertomographie. Den skalaren Werte werden mittels einer 'Transferfunktion' Farbwerte und Transparenz zugeordnet.

Eine Implementierung einer Interaktiven Transferfunktion in FAnToM ist in Abb. 4 zu sehen. Die horizontale Achse entspricht den skalaren Werten zwischen Minimum und Maximum im Datensatz sowie den zugeordneten Farben. Die vertikale Achse entspricht der Transparenz, von vollständig transparent am unteren Rand bis zu komplett intransparent am oberen Rand. Durch Position und Farbe der Dreiecke am oberen Rand der Transferfunktion kann die Farbzuordnung festgelegt werden. Die Quadrate stellen interaktiv ausgewählte Punkte dar, die Zuordnungen von Werten im Skalarfeld zu Farb- und Transparenzwerten entsprechen, zwischen denen entlang der verbindenden Linien interpoliert wird. So werden allen Werten im Feld eine Farbe und eine Transparenz zugeordnet.

Ziel des DVR ist es, mithilfe der Transferfunktion eine dreidimensionale Darstellung des Feldes zu erzeugen, in der Bereiche entsprechend der Transferfunktion Licht ausstrahlen, das von anderen Bereichen entsprechend ihrer Transparenz durchgelassen oder absorbiert wird. Dadurch ist es möglich, Strukturen im Inneren des Feldes sichtbar zu machen. Ein Beispiel für eine so erzeugtes Bild ist in Abb. 5 dargestellt.

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

4.4.1 Raycasting

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

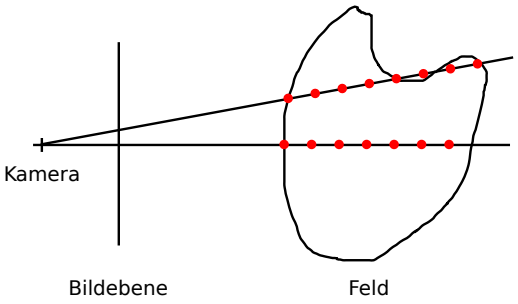


Abb. 6: Eine vereinfachte Darstellung eines optimierten Raycastings. Die Bildebene entspricht der Ebene, die durch die durch die imaginäre Position der Pixel gebildet wird. Die Roten Punkte entlang der Strahlen entsprechen den Samplepunkten.

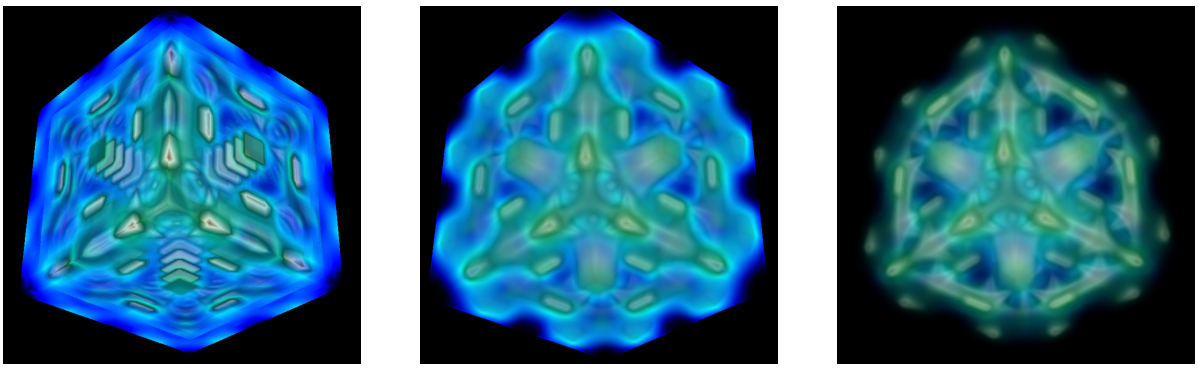
Dabei sei die Transparenz ein Wert im Intervall $[0, 1]$ und Farbe ein Vektorraum über $\mathbb{R}$, wie z.B. im RGBA Farbmodell, für den die Multiplikation mit einem Skalar definiert ist. Die Farbe des Pixels p wird dann für die Samplepunkte entlang des entsprechenden Strahls berechnet als

$$p = \sum_{i=1}^{n} \left(f(s_i) \cdot \prod_{j=1}^{i} \frac{(1 - t(s_j))}{n} \right) \tag{26}$$

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

Eine weitere Optimierung besteht darin, die Samplepunkte erst nach dem Eintreten des Strahls in das Feld beginnen zu lassen und die Berechnung zu beenden, sobald alle verbleibenden Samplepunkte ausserhalb des Feldes liegen. Eine vereinfachte Darstellung dieses optimierten Verfahrens ist in 6 dargestellt.

Die Qualität eines Raycastings ist stark abhängig von der Anzahl an Samplepunkten pro Strahl und somit vom Abstand zwischen den Samplepunkten. Ein geringerer Abstand führt generell zu weniger Artefakten und macht kleine Strukturen besser erkennbar, erhöht jedoch den Rechenaufwand. Indem der Benutzer die Anzahl von Samplepunkten selbst auswählt, kann er seinen optimalen Tradeoff zwischen Rechenzeit und Bildqualität finden. In Abb. 7a und 7b sind zwei Darstellungen desselben Skalarfeldes zu sehen, die durch Raytracing mit einer unterschiedlichen Anzahl von Samplepunkten erzeugt wurden. 7a wurde mit 20 Samplepunkten erstellt, wodurch visuelle, scheibenartige Artefakte



(a) (b) (c)

Abb. 7: Drei Raycastings des Felds $f : \mathbb{R}^3 \to \mathbb{R}, f \begin{pmatrix} x \\ y \\ z \end{pmatrix} = cos(x \cdot y \cdot z)$.

entstanden. Bei einer höheren Menge von 200 Samplepunkten in 7b verschwinden diese Artefakte.

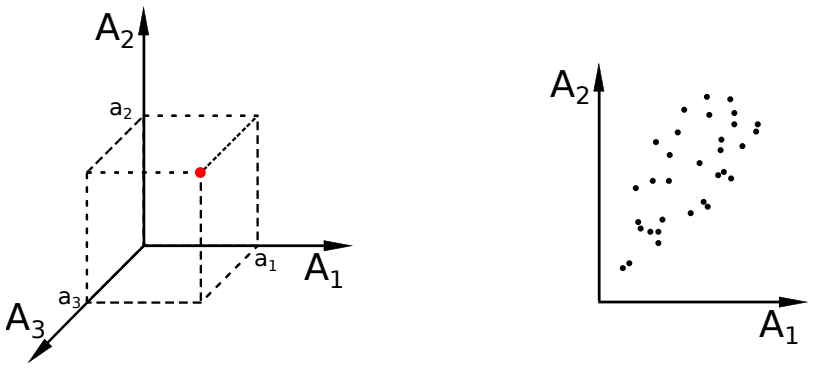
Eine weitere wichtige Einstellung ist, welche Zahl in Formel 26 für Variable n verwendet wird. Wenn n die Anzahl der Samplepunkte pro Strahl ist, kann dies, wenn die Blickrichtung der Kamera nicht senkrecht auf einer der Seitenflächen steht, dazu führen, dass Bereiche am Rand des Feldes transparenter dargestellt werden als in der Mitte. Das liegt daran, dass der Strahl am Rand weniger Samplepunkte produziert die innerhalb des Feldes liegen. Ein Beispiel dafür ist in 7c mit 200 Samplepunkten dargestellt.

Alternativ kann für n die Anzahl der Samplepunkte, die tatsächlich im Feld liegen, gewählt werden. Dies wurde in Abb. 7b getan. Wenn man nun 7b und 7c miteinander vergleicht, fällt auf, dass die Pixel in der mitte der Darstellung etwa die gleichen Farben erhalten haben, 7c in Richtung Rand jedoch erheblich transparenter wird. Welche von beiden Varianten bevorzugt wird, hängt von der Art des Datensatzes ab sowie davon, wie wichtig es ist, korrekte Werte ablesen und vergleichen zu können.

4.5 Contiuous Scatterplotting

4.5.1 Scatterplotting

Scatterplotting ist ein weiter verbreitetes Verfahren, um Attribute von Objekten relativ zueinander darzustellen. Es bildet eine Funktion auf einer Menge von Objekten M als $\tau : M \to \mathbb{R}^n, n \in \{1, 2, 3\}$, wobei das Bild eine Projektion auf einzelne oder zusammengesetzte Attribute der Objekte darstellt. Dazu wird pro Objekt ein Punkt in ein in zwei- oder dreidimensionales Koordinatensystem eingetragen, dessen Koordinatenachsen Attributwerten entsprechen. Beispiele für Scatterplottings sind in Abb. 8 abgebildet. Das



(a) (b)

Abb. 8: 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.

Bild von τ ist auf maximal 3 Dimensionen beschränkt, da höherdimensionale Daten visuell nur schwer zu interpretieren sind.

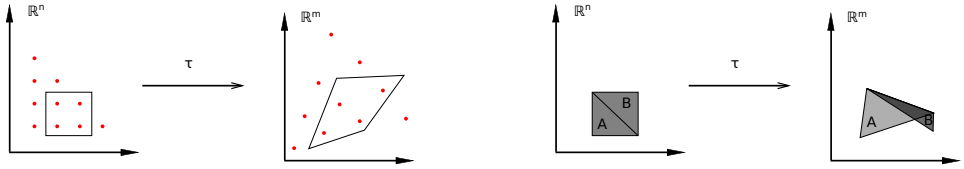
Ein Vorteil von Scatterplottings ist die leichte Erkennbarkeit von Strukturen in den Datensätzen: Cluster, Outlier und funktionale Zusammenhänge zwischen den verwendeten Attributen sind leicht erkennbar.

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

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

4.5.2 Erklärung des Continuous Scatterplottings

Das Continuous Scatterplotting stellt eine Erweiterung des Scatterplottings dar, so dass statt nur einer endlichen Anzahl von Objekten auch kontinuierliche Bereiche verarbeitet werden können. Die von Bachthaler und Weiskopf[3] vorgestellte Definition wurde von Fritzsch[12] um einige wichtige Punkte erweitert. Dabei verwendet es einen Ansatz, der dem Verfahren bei der Herleitung der Kontinuumsmechanik aus Systemem mit diskreten Massepunkten ähnelt [3, S. 1429]. Das Ziel des Continuous Scatterplottings ist es eine



(a) (b)

Abb. 9: Darstellung der Dichteverteilung abhängig von τ. (a) Die roten Punkte deuten die Dichte in den beiden Scatterplots an, die zwei Vierecke sind ein Bereich des Urbilds, der durch τ verformt wird. Zu erkennen ist, dass eine Streckung eines Bereichs zur Verringerung der Dichte und eine Stauchung zur Erhöhung der Dichte führt. (b) Der Continuous Scatterplot zweier Dreiecke. Der Grauwert gibt die Dichte in den jeweiligen Bereichen an.

Dichtefunktion $\sigma : \mathbb{R}^m \to \mathbb{R}$ zu finden, die den Punkten des Bildraums, abhängig von τ, Dichtewerte zuordnet. σ kann als Funktion aufgefasst werden, die jedem Pixel des Scatterplots einen skalaren Wert zuordnet, der z.B. als Farbe oder Opazität dargestellt werden kann. In Abb. 9 ist der Effekt, den τ auf die Dichtefunktion hat, dargestellt. Um σ berechnen zu können, werden 2 Annahmen getroffen:

Die erste Annahme ist, dass eine Dichtefunktion $s : \mathbb{R}^n \to \mathbb{R}$ existiert, die jedem Punkt des Urbilds von τ einen skalaren Wert, die Dichte an diesem Punkt, zuordnet. Der Einfachheit halber wird 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:

$$M = \int_V s(x) d^n x \tag{27}$$

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

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

$$\int_\phi \sigma(\xi) d^m \xi = \int_{\tau^{-1}(\phi)=V} s(x) d^n x \tag{28}$$

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

$$\int_V s(x) d^n x = \int_{\phi=\tau(V)} \sigma(\xi) d^m \xi \tag{29}$$

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

$$\int_{\phi} \sigma_v(\xi) d^m\xi = \int_{\tau|_V^{-1}(\phi)} s(x) d^n x \tag{30}$$

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

$$\begin{aligned}
\int_{\phi} \sigma(\xi) d^m\xi &= \int_{\bigcup\limits_{V_i} \tau|_{V_i}^{-1}(\phi)} s(x) d^n x \\
&= \sum_{V_i} \int_{\tau|_{V_i}^{-1}(\phi)} s(x) d^n x \\
&= \sum_{V_i} \int_{\phi} \sigma_{V_i}(\xi) d^m\xi \\
&= \int_{\phi} \sum_{V_i} \sigma_{V_i}(\xi) d^m\xi
\end{aligned} \tag{31}$$

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

$$\sigma(\xi) = \sum_{V_i} \sigma_{V_i}(\xi) \tag{32}$$

Eine intuitive Interpretation von Formel 32 ist, dass gesamte Dichtefunktion von τ als Summe der Dichtefunktionen der Einschränkungen von τ auf die V_i gebildet werden kann.

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

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

$$\int_{\tau(V)} \sigma(\xi) d^{m=n}\xi = \int_V \sigma(\tau(x)) |det(J_\tau)(x)| d^n x = \int_V s(x) d^n x \tag{33}$$

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

$$\sigma(\xi) = \frac{s(\tau^{-1}(\xi))}{|\det(J_\tau)(\tau^{-}1(\xi))|} \tag{34}$$

Wie schon in 3 Grundlagen erwähnt entspricht der Betrag der Determinanten der Jacobimatrix der Expandieren oder Schrumpfen der Funktion in der Nähe des jeweiligen Punktes. Wenn die Funktion expandiert nimmt also die Dichte ab, wenn sie schrumpft nimmt die Dichte zu.

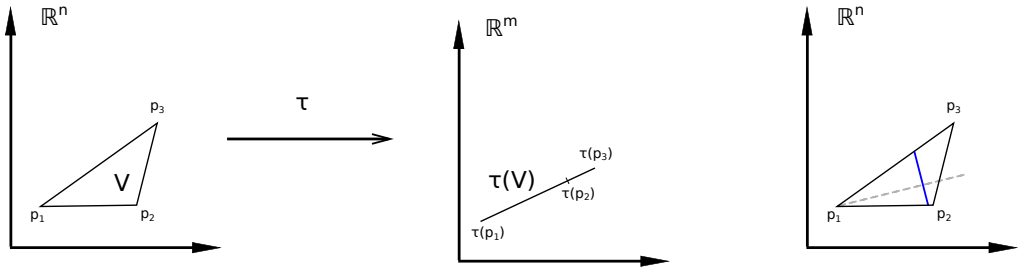
Fall 2: τ ist differenzierbar und konstant über V, V ist keine Nullmenge Wenn $det(J_\tau) = 0$, dann ist τ kein Diffeomorphismus und σ kann nicht so wie in Fall 1 definiert werden. Das ist z.B. der Fall, wenn τ Bereiche mit konstanten Werten enthält und führt dazu, dass ein Bereich aus $\mathbb{R}^n$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

$$\begin{aligned} \sigma(\xi) &= \sum_{V_i \in V} \delta(\xi - \tau(V_i)) \cdot \int_{V_i} s(x) d^n x \\ &= \sum_{V_i \in V} \delta(\xi - \tau(V_i)) \cdot Vol(V_i) \qquad (da\ s(x) = 1) \end{aligned} \tag{35}$$

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

$$\begin{aligned} \int_\phi \sigma(\xi) d^n \xi &= \sum_{V_i} \int_\phi \delta(\xi - \tau(V_i)) \cdot Vol(V_i) d^n \xi \\ &= \sum_{\forall V_i \ :\ \tau(V_i) \in \phi} Vol(V_i) \\ &= \int_{\tau^{-}1(\phi)} 1 \ d^n x \end{aligned} \tag{36}$$

Da $s(x) = 1$ festgelegt ist, ist die Bedingung erüllt. In 9b ist ein Beispielfall dafür angegeben. Die Dichte im überlappenden Bereich der beiden Dreiecke ist gleich der Summe der Dichte der einzelnen Dreiecke. Bei anderen Dichtefunktionen muss Formel 35 entsprechend angepasst werden.



(a) (b)

Abb. 10: Eine Darstellung den 5. 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 Fritzsch vorgeschlagenen Ansatzes. Die graue, gestrichelte Linie entspricht dem Gradienten, die Länge der blauen Linie ist proportional zur Dichte.

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

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

Fall 5: τ ist differenzierbar und kein Diffeomorphismus, V ist keine Nullmenge Von Bachthaler und Weiskopf wurde noch ein fünfter Fall genannt, den sie jedoch als nicht in der Praxis vorkommend ansahen. Fritzsch konnte jedoch Beispiele für diesen Fall angeben[12, S. 23 f.], die in der Praxis eine Rolle spielen. Dies tritt ein, wenn V keine Nullmenge ist, jedoch auf eine Nullmenge $\tau(V)$ abgebildet wird (siehe Abb. 10a), also wenn entweder mindestens eine Komponente des Bildes konstant ist, oder eine funktionale Abhängigkeit zwischen mindestens zwei Komponenten existiert.

Die Jacobi Matrix enthält in diesem Fall mindestens eine Nullzeile, wodurch für ihren Rang gilt $0 < rang(M) < m$.

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 $rang(M) = 1$ 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 zunächt 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 10b zu sehen. Die Dichte an den Punkten von $\tau(V)$ wird dann proportional zur Masse gewählt.

5 Umsetzung

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

Die Grundidee der entwickelten Visualisierung besteht darin, zwei DVRs zu erzeugen. Das erste stellt die Zellen des urspünglichen Feldes dar, über die eine konstante Dichte angenommen wird. 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. Die Dichte in den Zellen der neuen Zellstruktur werden durch Methoden des Contiuous Scatterplottings berechnet.

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

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

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

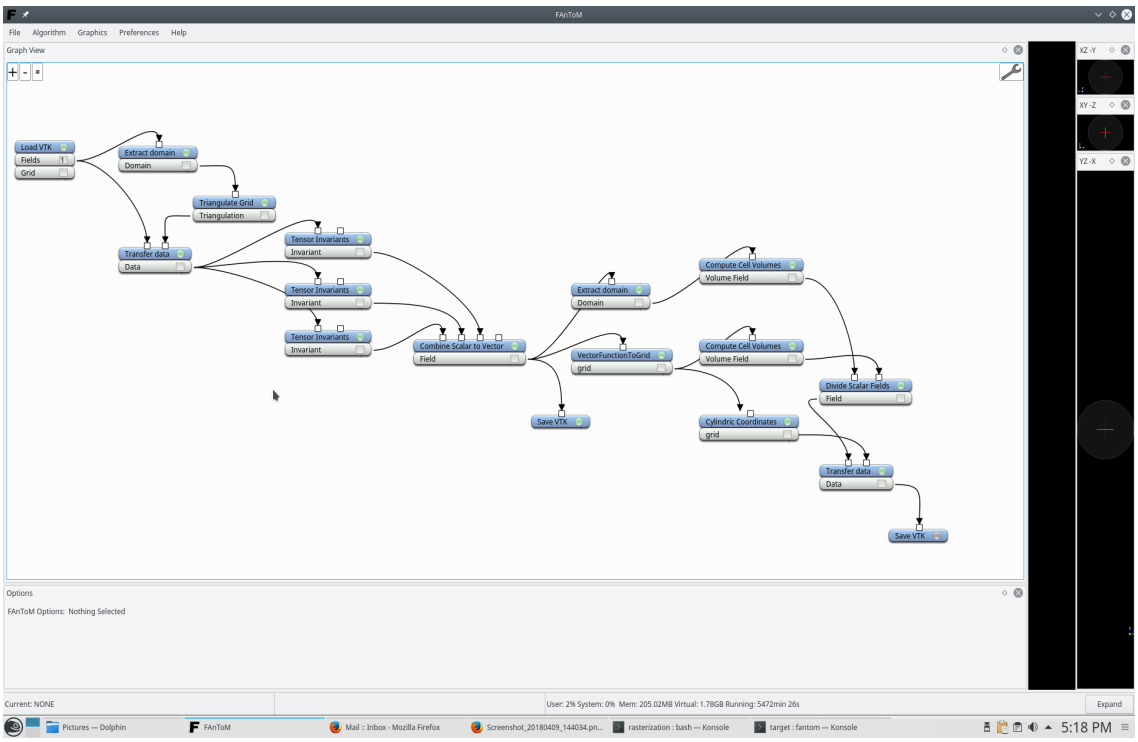


Abb. 11: Der Flowgraph der FAnToM Session, die zur Vorbereitung der Daten verwendet wird.

5.1 Datenaufbereitung

Bevor die Visualisierung beginnen kann, müssen die Daten aufbereitet werden. Um Zeit zu sparen, wird dieser Schritt für jeden Datensatz nur einmal durchgeführt, und die Ergebnisse abgespeichert. Die Aufbereitung geschieht durch eine Reihe von FAnToM Algorithmen, die im Folgenden beschrieben werden. Ein Screenshot der für die Aufbereitung verwendeten Session ist in Abb. 11 zu sehen. Die Algorithmen der Session werden im Folgenden 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 urspr̈unglichen Feldes werden auf diese neue Zellstruktur durch den Algorithmus ‘Transfer Data’ kopiert.

Als nächstes werden die Invarianten der Tensoren berechnet. Da die Datensätze aus 3×3 Matrizen bestehen, enthalten die zugehörigen Invariantensätze jeweils 3 Invarianten. Dazu werden 3 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. 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, die als Eingabe der eigentlichen Visualisierung verwendet wird.

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

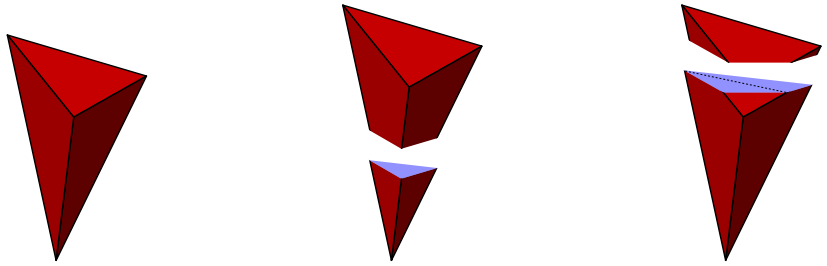
Im nächsten Schritt wird das Volumen der Zellen des ursprünglichen Feldes und der neu erzeugten Zellstruktur berechnet (‘Compute Cell Volumes’). Es entstehen zwei Skalarfelder, die den Zellen ein Volumen zuordnen. Die Skalarfelder werden elementweise durcheinander geteilt (Original geteilt durch verschobenes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

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

5.2 Voxelisierung

Da das in dieser Arbeit verwendete DVR Verfahren (siehe Abschnitt 5.5) eine Repräsentation der Dichtefeldes als 3D Texturen vorraussetzt, 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 4 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



(a) (b) (c)

Abb. 12: 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.

und die restlichen drei Farbwerte den interpolierten Invarianten der Zelle, in der das Voxel liegt. Die Invarianten spielen allerdings nur bei der Visualisierung des ursprünglichen Feldes 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 selbstdurchdringengenden 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.[20]. Dabei werden Schichten der 3D Textur berechnet, indem die Schnittflächen zwischen den Tetraedern und Ebenen berechnet. 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 es als Folge von einem oder zwei Dreiecken ('Triangle Strip') dargestellt werden (siehe Abb. 12). 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 berechneteten 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 zeite und dritte Punkt entsprechen dem aktuellen Liniensegment und der vierte Punkt ist der Endpunkt des nächsten Liniensegments. Da es immer genau 4 Punkte pro Primitiv sind, eignet es sich jedoch auch, um Tetraeder an die Pipeline zu übergeben.

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

Wie schon im Abschnitt 4.2 angesprochen, ist es möglich, die Ausgabe der Renderpipeline in eine 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 Benutzer 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 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 zunächt 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
$schnittZahl \leftarrow 0$;
$schnittPunkte \leftarrow []$;
for $\forall(p_m, p_n), n < m;$ `// Für jede Kante des Tetraeders`
do
  $p \leftarrow schnittpunktBerechnen(z, p_m, p_n)$;
  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 $[0, \ldots, 1]$, 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 Benutzer 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 mehr als 10 Grössenordnungen. Durch das Ziehen der dritten Wurzel nähern sich die Werte weiter an die 1 an. Da die Dichtewerte nicht direkt abgelesen werden müssen, stellt das auch kein Problem dar.

Durch den Fragment Shader werden im Invariantenraum mehrere der erzeugten Dreiecke auf dieselben Voxel abgebildet. Die Kombination der Werte pro Voxel erfolgt durch eine von OpenGL bereitgestellten Funktion, dem 'Blending'. Im ursprünglichen Feld kann dies nicht vorkommen, 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 Continuos Scatterplottings implementiert.

5.3 Umsetzung des Continuous Scatterplottings

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

Der erste Teil, das Berechnen des Dichtebeitrags σ_{V_i} der Tetraeder V_i, efolgt bereits in der Vorverarbeitung der Daten. Dabei wird die für Fall 1 angegebene Formel 34 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 32 anwenden. Die σ_V entsprechen den berechneten Dichtewerten pro Tetraeder, die für jedes Voxel durch das Blending aufaddiert werden.

5.4 Berechnung der maximalen Dichte

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

Zunächst wird die höchste, überhaupt mögliche Dichte in einem Voxel berechnet. Diese kommt genau dann vor, wenn alle Tetraeder $V_i, i = 1, \ldots, n$ sich in einem Voxel überschneiden. Somit entspricht sie der Summe der Dichten aller Tetraeder. Danach wird die Voxelisierung einmal ausgeführt, wobei jeder Dichtewert durch die größte mögliche Dichte geteilt und somit auf das Intervall $[0, \ldots, 1]$ normalisiert wird.

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

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

Die Eingabe der Shaderpipeline ist die 3D Textur sowie eine Linie zwischen den Punkten (-1.0, 0.0, 0.0) und (1.0, 0.0, 0.0). Berechnet wird eine Darstellung der Linie aus Z Pixeln, wobei Z die Anzahl an Schichten der 3D Textur ist. Der Wert jedes Pixels entspricht

dem Maximum der jeweiligen Schicht. Durch Verwendung eines Framebuffers wird das Ergebnis in eine Textur gerendert, die erheblich wenige Speicherplatz verbraucht als die 3D Textur. Diese wird zurück in den Arbeitsspeicher geladen und das Maximum durch linearen Durchlauf gefunden.

Vertex Shader Der Vertex Shader ordnet (-1.0, 0.0, 0.0) den Wert 0 und (1.0, 0.0, 0.0) den Wert 1 zu und gibt diese Werte an den Fragment Shader weiter.

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

Data: T 3D Textur, z interpolierte z-Koordinate der Schicht,
x_{max}, y_{max} Ausdehung der Schicht in x und y Richtung
Result: Maximum der Schicht
$max \leftarrow 0$;
for $x \leftarrow 0$ **to** x_{max} **do**
  for $y \leftarrow 0$ **to** y_{max} **do**
    $p \leftarrow punktAnKoordinaten(\frac{x}{x_{max}}, \frac{y}{y_{max}}, z)$
    $voxelDichte = texturwertAnPunkt(T, p)$
    $max = maximuim(max, voxelDichte)$
  end
end
return max;

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

5.5 Umsetzung des Raycastings

Für die beiden DVR Darstellungen wird eine Variante des schon in 4.4.1 beschriebenen Verfahrens verwendet. Dies geschieht wiederum in Form einer Shaderpipeline.

Die Eingabe der Pipeline ist die vorberechnete 3D Textur und ein Rechteck bestehend aus zwei Dreiecken. Das Rechteck entspricht Bildeben in Abb. 6. Die Bildebene befindet sich so vor der Kamera, dass sie das gesamte Blickfeld einnimmt. Das Raycasting wird durchgeführt, indem die Shader die Farben auf der Bildebene berechnen.

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

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

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

Danach wird der Schnitt zwischen den die 3D Textur begrenzenden, achsenparallelen Ebenen (der 'Bounding Box', kurz BB) und diesem Strahl berechnet. Abhängig davon, welche Ebenen der Strahl in welcher Reihenfolge geschnitten, kann effizient bestimmt werden, ob die 3D Textur vom Strahl durchquert wird oder nicht. Falls nicht, kann sofort schwarz als Farbe des Fragments zurückgegeben werden.

Falls der Strahl die 3D Textur schneidet, wird der Abstand der Samplepunkte berechnet. Dem Benutzer steht dabei frei, welche der beiden in Abschnitt 6 beschriebenen Varianten er bevorzugt.

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

Die eigentliche Berechnung der Fragmentfarbe geschieht in einer Schleife. Für jeden Samplepunkt entlang des Strahls wird die Dichte an diesem Punkt gemessen, durch die Transferfunktion in Farbe und Transparenz übersetzt und nach Formel 26 zu einem Farbwert kombiniert.

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

Wenn die Invarianten (nur beim ursprünglichen Feld) oder der Punkt selbst (nur im Invariantenraum) ausserhalb des ausgewählten Bereichs liegen, wird Farbe und Dichte am Samplepunkt als 0 angenommen.

Der Pseudocode in Algorithmus 3 beschreibt die Funktionsweise der Schleife.

Data: s Strahlvektor der Länge 1, t Schrittweite,
T 3D Textur, p_0 erster Samplepunkt
Result: akkumulierte Farbe entlang des Strahls
$transparenz \leftarrow 1.0$;
for $i \leftarrow 0$ **to** n **do**
    $texturwert \leftarrow texturwertAnPosition(T, p)$;
    $invarianten \leftarrow invarianten(texturwert)$;
    $dichte \leftarrow dichte(texturwert)$; `// Der`
    ;
    $color \leftarrow transferfunktion(dichte)$;
end

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

5.6 Funktionen

5.7 Handbuch

5.8 Muss noch überarbeitet werden

6 Ergebnisse

6.1 title

7 Ausblick

Fehler reduzieren durch Berechnung der Größe des tatsächlich belegten Volumens pro Tetraeder und Voxel -> Supersampling???

Abschlussarbeiten

[12] Clemens Fritzsch. “Visuelle Analyse kontinuumsmechanischer Simulationen durch kontinuierliche Streudiagramme”. Diploma Thesis, University of Leipzig. Magisterarb. Leipzig, Germany: Universität Leipzig, 2016.

Bücher

[2] Lisa S Avila u. a. *The VTK User's Guide*. Kitware New York, 2010.

[5] Ray M Bowen und Chao-Cheng Wang. *Introduction to vectors and tensors*. Bd. 1. Courier Corporation, 2008.

[11] 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] Keith D. Hjelmstad. *Fundamentals of Structural Mechanics*. 10. Aufl. Springer US Verlag KG, 2005. ISBN: 978-3-13-477010-0.

[18] Norbert Kusolitsch. *Maß-und Wahrscheinlichkeitstheorie: Eine Einführung*. Springer-Verlag, 2014.

[19] Bruce R Kusse und Erik A Westwig. *Mathematical physics: applied mathematics for scientists and engineers*. John Wiley & Sons, 2010.

Artikel

[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 Basser, James Mattiello und Denis LeBihan. “MR diffusion tensor spectroscopy and imaging”. In: *Biophysical journal* 66.1 (1994), S. 259–267.

[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] Thierry Delmarcelle und Lambertus Hesselink. “Visualizing second-order tensor fields with hyperstreamlines”. In: *IEEE Computer Graphics and Applications* 13.4 (1993), S. 25–33.

[9] Robert A Drebin, Loren Carpenter und Pat Hanrahan. “Volume rendering”. In: *ACM Siggraph Computer Graphics*. Bd. 22. 4. ACM. 1988, S. 65–74.

[10] 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.

[14] Mario Hlawitschka u. a. “Top Challenges in the Visualization of Engineering Tensor Fields”. In: *Visualization and Processing of Tensors and Higher Order Descriptors for Multi-Valued Data*. Springer, 2014, S. 3–15.

[16] Gordon Kindlmann. “Superquadric tensor glyphs”. In: *Proceedings of the Sixth Joint Eurographics-IEEE TCVG conference on Visualization*. Eurographics Association. 2004, S. 147–154.

[17] 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.

[20] Antonio J Rueda u. a. “Voxelization of solids using simplicial coverings”. In: (2004).

[21] Alexander Wiebel u. a. “Fantom-lessons learned from design, implementation, administration, and use of a visualization system for over 10 years”. In: (2009).

[22] Valentin Zobel und Gerik Scheuermann. “Extremal curves and surfaces in symmetric tensor fields”. In: *The Visual Computer* (2017), S. 1–16.

Internetressourcen

[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).

[6] The Qt Company. *Qt Website*. URL: https://www.qt.io/ (besucht am 28. 03. 2018).

[15] The Khronos™ Group Inc. *OpenGL Website*. URL: https://www.khronos.org/opengl/ (besucht am 28. 03. 2018).