

Visualisierungsinstitut der Universität Stuttgart
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Studienarbeit Nr. 2261

Progressives Rendering Multivariater Daten

Bernhard Maier

Studiengang: Informatik
Prüfer: Prof. Dr. Weiskopf
Betreuer: Dipl.-Inf. Julian Heinrich

begonnen am: 28. Januar 2010

beendet am: 20. Juli 2010

CR-Klassifikation: I.3.8

Inhaltsverzeichnis

1	Einleitung	6
1.1	Aufgabenstellung	6
1.2	Gliederung	7
2	Grundlagen	8
2.1	Megamol	8
2.2	Shader	8
2.3	Frame Buffer Objects (FBO's)	8
2.4	Streudiagramme	9
2.5	Parallele Koordinaten	10
2.6	Multivariate Datensätze mit kontinuierlichem Definitionsbereich	10
2.7	Visualisierung Multivariater Daten mit kontinuierlichem Definitionsbereich . . .	11
2.7.1	Kontinuierliche Scatterplots	11
2.7.2	Kontinuierliche Parallele Koordinaten	12
2.8	Splatting	13
3	Lösungsansatz	15
3.1	Ansatz für die Beschleunigung des Renderns	15
3.2	Aufbau des Plug-Ins	16
3.2.1	Das bestehende Plug-In für diskrete Plots	17
3.2.2	Kommunikation zwischen den Renderern des Plug-Ins	17
3.2.3	Das Plug-In für kontinuierliche Scatterplots und Parallele Koordinaten	17
4	Zufällige Auswahl von Samples, Berechnung der Skalarwerte und Gradienten	20
4.1	Quasi-Random Sequences	20
4.1.1	Sobol Sequences	21
4.1.2	Hammersley Sequences [Hal60]	22
4.1.3	Halton Sequences [Hal60]	22
4.1.4	Vergleich der Sequenzen	23
4.1.5	Implementation der Halton Sequences	24
4.2	Sampling und Ermittlung der Gradienten	24
5	Splatting von kontinuierlichen Scatterplots und kontinuierlichen Parallelen Koordinaten	26
5.1	Kontinuierliche Scatterplots	26
5.1.1	Beschreibung der Abbilder und der Berechnung der Dichte	26
5.1.2	Implementation des Splatting	28
5.2	Kontinuierliche Parallele Koordinaten	29
5.2.1	Beschreibung der Abbilder und der Berechnung der Dichte	29
5.2.2	Implementation des Splatting	31

6 Colormapping	35
6.1 Einfaches Colormapping	35
6.2 Probleme beim einfachen Colormapping	35
6.3 Verbesserter Ansatz: Logarithmisches Colormapping	36
6.4 Automatische Wahl der Faktoren a und b	36
7 Qualitativer und Quantitativer Vergleich mit herkömmlichen Methoden	38
7.1 Beschleunigung	38
7.2 Qualität	39
8 Zusammenfassung und Ausblick	41
8.1 Zusammenfassung	41
8.2 Ausblick	41
Literaturverzeichnis	42

Abbildungsverzeichnis

2.1	Die Werte der ersten beiden Formanten von 5 deutschen Vokalen, von 2 verschiedenen Sprechern im Vergleich. (+ steht für Sprecher 1, x steht für Sprecher 2. Die Y-Achse zeigt den 1. Formanten, die X-Achse zeigt den 2. Formanten.	9
2.2	Dieser Parallele Koordinaten Plot zeigt die Merkmale verschiedener Prozessoren.	10
2.3	Kontinuierlicher Scatterplot. Gerendert mit dem Ansatz von [BWo8]	12
2.4	Kontinuierliche Parallele Koordinaten. Gerendert mit dem Ansatz von [HWo9]	13
3.1	Die Abbildung zeigt den Zusammenhang zwischen Größe der Splats in der Data- ($\tau(V_s)$) und der Spatial-Domain (V_s) im 1D-Fall.	16
3.2	UML-Klassendiagramm der implementierten Klassen	18
4.1	Zwei Zufalls-Sequenzen mit $N = 10$ Punkten. Links ist die Verteilung schlecht, rechts ist sie besser.	20
4.2	Die ersten 100 Paare einer 2D-Halton-Sequenz. Als Basen für die Berechnung wurden 2 und 3 gewählt.	23
4.3	Bei Verwendung von Central Differences wird um ein Sample herum im Abstand $\Delta (= \Delta_{sampling})$ nochmals gesamplet, und daraus der Gradient berechnet.	25
5.1	Die Abbildung zeigt einen rechteckigen Splat für Scatterplots. Die Seitenlängen l_1 und l_2 sind proportional zu den Gradientenbeträgen $ \nabla \zeta_1 $ und $ \nabla \zeta_2 $	26
5.2	Links: Intensitätscodierte Dichteverteilung eines Splats. Rechts: Farbcodierte Dichteverteilung eines Splats.	28
5.3	Ein einfacher Trapez-Splat in Parallelen Koordinaten.	30
5.4	Ein Splat als Körper. Die in Parallelen Koordinaten vertikal verlaufenden 1D Gaussverteilungen sind Rot dargestellt. Die Varianzen der Gausskurven wurden in diesem Beispiel von $x = 0$ bis $x = 1$ zwischen $\sigma_0 = 1$ und $\sigma_1 = 2$ linear interpoliert.	30
5.5	Ein Gauss-Splat in Parallelen Koordinaten. Links: Intensitätscodiert. Rechts: Farbcodiert.	31
5.6	Ein Splat mit seiner rechteckigen Hilfsgeometrie. Links: Texturkoordinaten. Rechts: Globale Koordinaten.	33
5.7	Beispiele für alternative Hilfsgeometrien eines Splats in Parallelen Koordinaten (im Bild grün markiert). In Rot sind die Umrisse der eigentlichen Splats zu sehen.	34
6.1	Farbskala zur Farbcodierung der Dichtewerte. (Standard Temperaturskala) Die Farbe Blau hat in einer entsprechenden 1D Textur die Texturkoordinate $s = \frac{1}{6}$	35
7.1	Das Diagramm zeigt, wie sich die benötigte Dauer für das Splatting erhöht, wenn man die Anzahl der Splats vergrößert.	38
7.2	Das Diagramm zeigt, wie sich die L2 Distanz der Ergebnisse des Splatting zum Ergebnis der Ansätze [BWo8] bzw. [HWo9] entwickelt, wenn man die Splat-Anzahl erhöht.	39

Tabellenverzeichnis

3.1	Im Rahmen der Studienarbeit implementierte Renderer.	19
4.1	Aufwandsvergleich Sobol- und Halton Sequenzen	23
7.1	Ergebnisse des Splatting für Scatterplots und Parallele Koordinaten. Das Color-mapping wurde ohne konstante Parameter a, b automatisch durchgeführt. . . .	40

Verzeichnis der Listings

Verzeichnis der Algorithmen

3.1	Allgemeiner Splatting-Ansatz für kontinuierliche Plots. Kann direkt auf Scatterplots angewendet werden. Bei Parallelen Koordinaten wird das Innere der Schleife für alle Achsenpaare je einmal ausgeführt.	16
4.1	Umwandlung eines Bitstrings der Länge m in einen float-Wert.	22

1 Einleitung

Visualisierung beschäftigt sich damit, wie wissenschaftliche Daten dem Menschen so zugänglich gemacht werden können, dass er sie am besten verstehen und auswerten kann. Visualisierungen nutzen dabei oft graphische Repräsentationen von Daten, um dies zu ermöglichen, sind aber nicht darauf beschränkt. Wissenschaftliche Daten sind oft räumlicher Natur, (also Volumendaten) multivariat und interpolierbar. Zum Beispiel Datensätze, die mit Hilfe von Fluid Dynamik Simulationen ermittelt wurden, haben in der Regel diese Eigenschaften. Die Daten, die dann ausgelesen werden können, sind z.B. Temperatur- und Druckangaben für beliebige Positionen im Simulationsbereich. Ein mögliches Mittel zur Visualisierung von statistischen Zusammenhängen zwischen 2 Komponenten solcher multivariater Daten sind Streudiagramme. Parallele Koordinaten sowie Streudiagramm-Matrizen leisten dasselbe für $n > 2$ Komponenten. In [BW08] und [HW09] wurden analog dazu kontinuierliche Plots eingeführt: Kontinuierliche Scatterplots und kontinuierliche Parallele Koordinaten. Diese Visualisierungstechniken sind darauf ausgelegt, Zusammenhänge zwischen interpolierbaren Volumendaten korrekt zu visualisieren. Das bedeutet, dass nicht nur Datenwerte an diskreten Positionen in das Rendering des Plots eingehen, sondern dass auch alle Zwischenwerte berücksichtigt werden. Die Plots werden aber auch aufgrund ihrer Gestalt kontinuierlich genannt. Während normale Streudiagramme einzelne Datenwerte als "Punkte" in einem 2D-Diagramm zeigen, zeigt ein kontinuierlicher Plot stattdessen für jede Wertekombination im Diagramm einen farbcodierten Dichtewert an. Anhand der Farbe kann die relative Häufigkeit einer bestimmten Wertekombination abgelesen werden. In der Praxis entsteht so ein Farbverlauf.

1.1 Aufgabenstellung

Da das Rendern kontinuierlicher Scatterplots und kontinuierlicher Paralleler Koordinaten bisher relativ zeitaufwändig ist, sollte es in dieser Studienarbeit durch einen Splatting-Ansatz beschleunigt werden. Dazu sollten Methoden entwickelt und implementiert werden. Es sollten dabei folgende Punkte bearbeitet werden:

- Es sollte eine geeignete Interpolationsfunktion und Abtastung verwendet werden.
- Das Abbild und die Berechnung der Dichte der Splats unter Berücksichtigung eines Korrektheitskriteriums, der Masseerhaltung [BW08] (Siehe Abschnitt 2.7.1 in der vorliegenden Arbeit.) sollte beschrieben werden.
- Zudem sollte das Rendering progressiv ablaufen. D.h. wieder unter Berücksichtigung der Masseerhaltung sollte das Rendering in mehreren Schritten ablaufen und am Bildschirm mitverfolgt werden können.
- Die Methoden sollten als MegaMol-Plugin in C++ implementiert werden.
- Die entwickelten Methoden sollten abschließend mit herkömmlichen Methoden qualitativ und quantitativ verglichen werden.

1.2 Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: Bespricht die Grundlagen, die für das Verständnis der Studienarbeit notwendig sind, und geht auf die Arbeiten [BWo8] und [HWo9] ein, die der Ausgangspunkt für diese Studienarbeit sind.

Kapitel 3 – Lösungsansatz: Behandelt den Lösungsansatz, der implementiert wurde, und erklärt, aus welchen Klassen sich das Plug-In zusammensetzt und wie diese interagieren.

Kapitel 4 – Zufällige Auswahl von Samples, Berechnung der Skalarwerte und Gradienten: Stellt verschiedene Low-Discrepancy Sequences vor und beschreibt die Implementierung einer geeigneten Sequenz. Geht auf die Berechnung von Skalarwerten und Gradienten ein.

Kapitel 5 – Splatting von kontinuierlichen Scatterplots und kontinuierlichen Parallelen Koordinaten: Behandelt alles Wichtige über die Implementation der kontinuierlichen Scatterplots und Parallelen Koordinaten.

Kapitel 6 – Colormapping: Hier wird die Abbildung der normalisierten Dichtewerte auf ihre Farbcodierung beschrieben.

Kapitel 7 – Qualitativer und Quantitativer Vergleich mit herkömmlichen Methoden: Stellt die Ergebnisse des Splatting anhand eines verbreiteten Datensatzes vor. Dazu werden mit dem Splatting Ansatz gerenderte Plots mit den korrekten Ergebnissen verglichen.

Kapitel 8 – Zusammenfassung und Ausblick: Fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

2 Grundlagen

2.1 Megamol

Die Renderer für die kontinuierlichen Streudiagramme und Parallelen Koordinaten sollten als Plug-In in das Visualisierungs-Framework MegaMol eingefügt werden. Megamol setzt auf der hardwarenahen Grafikkbibliothek OpenGL, sowie auf der, ebenfalls an der Universität Stuttgart entwickelten, VisLib auf. Das ist eine Bibliothek die viele Basisklassen für die Entwicklung von Anwendungen aus dem Bereich der wissenschaftlichen Visualisierung bereithält, um z.B. Vektorrechnung zu unterstützen, oder die Verwendung von Framebuffer-Objekten zu erleichtern, die später noch vorgestellt werden. Näheres zu MegaMol folgt in Kapitel 3.2.

2.2 Shader

Auch der Einsatz von sogenannten Shadern wird von den MegaMol Klassen vereinfacht. Shader sind Programme die auf einem Vertex-, Geometry- oder Fragment-Prozessor einer Grafikkarte ausgeführt werden können [GLS]. Je nachdem auf welchem dieser Prozessoren sie ausgeführt werden, nennt man sie Vertex-, Geometry- oder Fragment-Shader. Für diese Studienarbeit wurden Vertex- und Fragment-Shader implementiert. Vertex Shader bekommen als Eingabe pro Ausführung die Position eines Vertex, die entsprechende Farbe, eine Normale und die Texturkoordinaten des Vertex. Der Vertex-Shader hat dann die Aufgabe, die Welt-Koordinaten des Vertex zu berechnen, und kann dessen zugeordnete Farbe anpassen [BC09]. Ein Fragment Shader wird immer erst nach dem entsprechenden Vertex-Shader ausgeführt. Für jeden Pixel auf dem Bildschirm, den eine mit OpenGL gezeichnete Geometrie abdeckt, wird er einmal ausgeführt. Er bekommt als Eingabe die unveränderliche Position des abgedeckten Pixels sowie dessen Texturkoordinaten und kann dann, z.B. durch Texturierung, seine Farbe festlegen.

2.3 Frame Buffer Objects (FBO's)

Die OpenGL Erweiterung EXT-framebuffer-object erlaubt das Rendern in "Framebuffer-Attachable Images" [FBO]. Das heißt, dass damit anstatt in den Framebuffer, in ein oder mehrere Ziele gerendert werden kann, die nicht direkt auf dem Bildschirm angezeigt werden. Wird dieses Offscreen-Rendern aktiviert, so dient das entsprechende "Framebuffer-Attachable Image" als Quelle und Ziel für die Fragment-Operationen [FBO]. Das heißt: Wenn man in einem Fragment-Shader die Farbe eines Fragments auf Farbe X setzt, so wird entsprechend der Blending-Funktion, diese Farbe X über die Farbe Y des "Framebuffer-Attachable Image" an der Stelle geblendet.

2.4 Streudiagramme

Streudiagramme sind Diagramme, die statistische Zusammenhänge in einer Grundgesamtheit visualisieren können. Sie bestehen aus einem Koordinatensystem und darin eingezeichneten Punkten oder Icons. Die Achsen X und Y des Koordinatensystems, können ordinalen oder nominalen Typs sein. Ein an Position (x, y) eingezeichneter Punkt zeigt an, dass es in der entsprechenden Grundgesamtheit eine Stichprobe mit den Merkmalen $X = x$ und $Y = y$ gibt. Solche bivariaten Stichproben können zum Beispiel als eine Liste von Paaren $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ repräsentiert werden. Ein Anwendungsbereich der Streudiagramme ist die Phonetik. Wenn man Spektrogramme von (deutschen) Sprachsignalen (also Sprachsignale im Frequenzbereich) untersucht, kann man, besonders bei Vokalen, waagrecht verlaufende Schwärzungen erkennen. Die entsprechenden Frequenzen werden Formanten genannt. Da die Vokale mehrere Formanten haben, kann man den Zusammenhang zweier Formanten bei verschiedenen Realisationen eines Vokals in einem Streudiagramm darstellen. In diesem Fall entspricht die Grundgesamtheit der Menge der Exemplare der gemessenen Vokale. Ein Streudiagramm kann erstellt werden, indem auf den 2 Achsen je ein Formant aufgetragen wird. Mit einem Streudiagramm wie in Abbildung 2.1 kann beispielsweise schnell überprüft werden, wie ähnlich sich zwei konkrete Sprecher sind, was ihre Formanten betrifft.

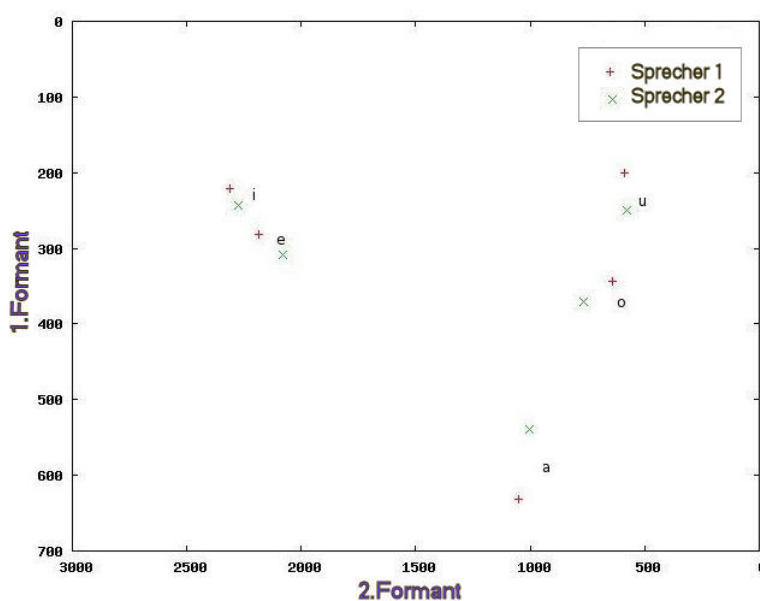


Abbildung 2.1: Die Werte der ersten beiden Formanten von 5 deutschen Vokalen, von 2 verschiedenen Sprechern im Vergleich. (+ steht für Sprecher 1, x steht für Sprecher 2. Die Y-Achse zeigt den 1. Formanten, die X-Achse zeigt den 2. Formanten.

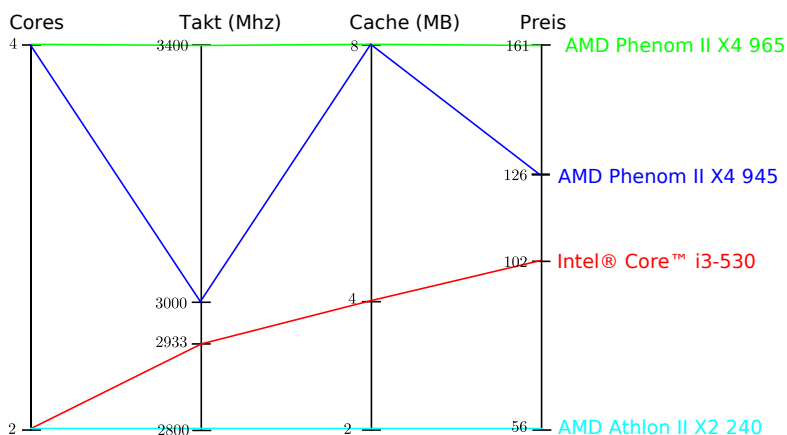


Abbildung 2.2: Dieser Parallele Koordinaten Plot zeigt die Merkmale verschiedener Prozessoren.

2.5 Parallele Koordinaten

Parallele Koordinaten wurden von Alfred Inselberg als Visualisierungstechnik für n -dimensionale Geometrie [ID87] eingeführt. Inselberg entwickelte in seiner Arbeit von 1987 auf der Basis von Parallelen Koordinaten einen Algorithmus zur Kollisionserkennung von Flugzeugen. Der Ansatz lässt sich aber auch dazu verwenden, multivariate Datensätze statistisch zu untersuchen. Multivariate Datensätze können z.B. einfache Mengen von n -Tupeln (a_1, a_2, \dots, a_n) sein. Für jedes Attribut a_i des Tupels ist dessen Typ bekannt, der wieder ordinal oder nominal sein kann. Darüberhinaus ist dessen maximaler Wertebereich bekannt. Jedes n -Tupel stellt eine Stichprobe der Grundgesamtheit dar, die mit Hilfe von Parallele Koordinaten Plots visualisiert werden kann. Dazu werden n zueinander parallele Achsen gezeichnet. Jeder der Achsen wird ein Attributtyp zugewiesen, und dementsprechend wird die Achse mit dem Wertebereich des Typs beschriftet. Ein Tupel wird visualisiert indem je $n - 1$ Liniensegmente zwischen den Achsen eingezeichnet werden, die die Achsen an den Attributwerten a_1, a_2, \dots, a_n schneiden. Abbildung 2.2 zeigt Parallele Koordinaten, die einen Vergleich verschiedener Prozessoren anhand der Merkmale Kernanzahl (Cores), Takt, Cachegröße und Preis ermöglichen. Wenn sehr viele Tupel im Plot repräsentiert sind, kann in dem Plot unter der Voraussetzung, dass die Achsen "richtig" geordnet sind und dass ein statistischer Zusammenhang wie z.B. lineare Abhängigkeit in den Daten vorliegt, dieser im Plot erkannt werden. (Lineare Abhängigkeit wäre an vielen parallelen Liniensegmenten zu erkennen.)

2.6 Multivariate Datensätze mit kontinuierlichem Definitionsbereich

Mit Hilfe von kontinuierlichen Streudiagrammen und Parallelen Koordinaten können räumliche, multivariate Datensätze untersucht werden. Im Rahmen dieser Studienarbeit werden nur solche multivariate Datensätze betrachtet, die mathematisch als Abbildungen vom 3-dimensionalen Raum (in [BW08] "Spatial Domain" genannt) auf die m -dimensionale "Data Domain" angesehen werden können: $\tau : \mathbb{R}^3 \rightarrow \mathbb{R}^m$. Alternativ kann man diese Datensätze als eine Menge von m Skalarfeldern auffassen: $\{\zeta_1, \zeta_2, \dots, \zeta_m\}$ mit $\zeta_i : \mathbb{R}^3 \rightarrow \mathbb{R}$. Dadurch sind theoretisch jedem Raumpunkt m Skalarwerte zugeordnet. Ein Computer kann jedoch nur endlich viele Werte speichern, deshalb repräsentieren Datensätze immer nur endlich viele

Funktionswerte solcher Abbildungen direkt. Diese Werte werden analog zu Pixeln, also den Farbinformationen eines computergespeicherten Bildes, Voxel genannt. Jedoch können, z.B. wenn der Datensatz ein uniformes Gitter nutzt, mit trilinearer Interpolation die Skalarwerte eines beliebigen Orts im Definitionsbereich von τ ermittelt werden. Das erfolgt, wenn man trilinear interpoliert, auf Basis der 8 nächsten Nachbar-Voxel des Orts. Dieser Vorgang wird Sampling oder zu Deutsch Abtastung genannt. In dieser Arbeit wurde zunächst der "Hurricane Isabel Dataset" verwendet. Das ist ein Datensatz, der in einer Auflösung von $500 \times 500 \times 100$ Voxeln zur Verfügung steht, und pro Voxel z.B. Druck und Temperaturangaben liefert. Druck und Temperatur sind also zwei Skalarfunktionen, deren Zusammenhang mit den hier besprochenen Visualisierungstechniken untersucht werden kann. Außerdem wurde der "Bucky Ball" Datensatz ($32 \times 32 \times 32$ Voxel) verwendet, ein synthetischer Datensatz, der pro Voxel einen Dichtewert und den Gradientenbetrag liefert. Ein (kontinuierlicher) Scatterplot, der den Zusammenhang von Dichte und Gradient der Dichte darstellt, kann dazu verwendet werden, auf einfache Weise homogene Bereiche des Volumendatensatzes auszuwählen, und als Volumen anzuzeigen.

2.7 Visualisierung Multivariater Daten mit kontinuierlichem Definitionsbereich

Multivariate Daten, wie der "Hurricane Isabel Dataset", lassen sich aufgrund von folgenden Problemen nur schlecht mit normalen Diagrammtechniken wie Streudiagrammen untersuchen. Ein Grund dafür ist das "Overplotting" Problem: zeichnet man einfach sehr viele Punkte in ein Streudiagramm ein, können wichtige Strukturen in den Daten nicht mehr erkannt werden. Entschärfen lässt sich dieses Problem indem man statt einem normalen Streudiagramm einen Dichteplot anfertigt. Dazu könnte man im Scatterplot übereinanderliegende Punkte/Samples in einem Renderziel eines FBO's aufaddieren und nach dem Sampeln normalisieren. Wenn ein normales uniformes Samplingschema verwendet wird tritt jedoch das Problem auf, dass Strukturen wahrzunehmen sind, die eigentlich in den Daten so nicht existieren, und somit nur dem gleichmäßigen Sampling geschuldet sind. Ein Beispiel für uniformes Sampling wäre, einfach jeden Voxel direkt in den Scatterplot abzubilden. Jedoch würde auch ein Ansatz mit zufälligem, also non-uniformem Sampling nicht die Datenwerte zwischen den einzelnen Samples berücksichtigen. Diese 3 Probleme werden durch die Ansätze [BWo8] und [HW09] behoben.

2.7.1 Kontinuierliche Scatterplots

Kontinuierliche Streudiagramme [BWo8] sind Dichteplots, die die kontinuierliche Eigenschaft multivariater Daten berücksichtigen, genau wie mit normalen Streudiagrammen lassen sich damit Zusammenhänge zwischen 2 Größen aufdecken. Abbildung 2.3 zeigt einen solchen kontinuierlichen Scatterplot, der aus dem "Blunt Fin"-Datensatz berechnet wurde. Auf der X-Achse ist der Skalarwert und auf der Y-Achse der Betrag des Gradienten aufgetragen. Das Rendern von kontinuierlichen Streudiagrammen wird in [BWo8] beschrieben. Es basiert auf einem mathematischen Modell, dessen grundlegende Annahme die Masseerhaltung ist. Sie besagt, dass die Masse in der "Spatial Domain", dem Definitionsbereich, auf dem die Skalarfunktionen definiert sind, gleich der Masse in der "Data Domain", also gleich der

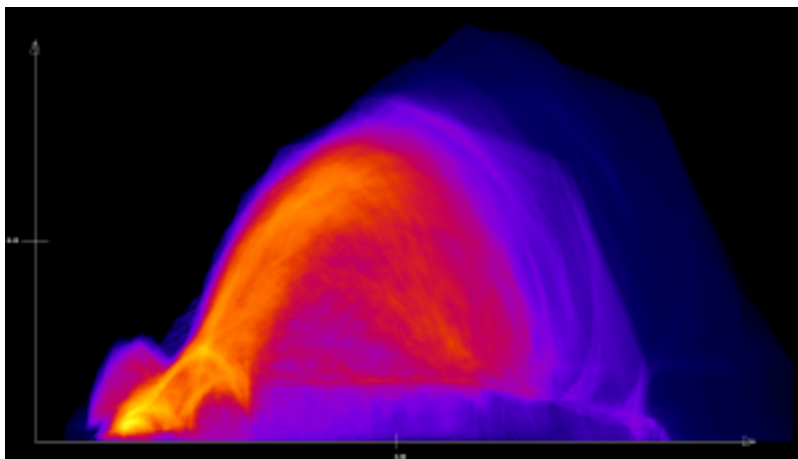


Abbildung 2.3: Kontinuierlicher Scatterplot. Gerendert mit dem Ansatz von [BWo8]

Masse im Scatterplot sein muss. Die Autoren leiten daraus eine Formel für Dichte $\sigma(\xi_0)$ einer beliebigen Wertekombination ξ_0 im Scatterplot her.

$$(2.1) \quad \sigma(\xi_0) = \int_{\tau^{-1}((\xi_1, \xi_2))} \frac{s(\hat{x})}{|Vol(D\tau)(\hat{x})|} d\hat{x}$$

Dabei ist $\tau^{-1}((\xi_1, \xi_2))$ die Schnittkurve der Isoflächen mit den Isowerten ξ_1 und ξ_2 . $s(\hat{x})$ ist die Dichte in der Spatial Domain, die in der Regel konstant 1 ist. Und $|Vol(D\tau)(\hat{x})|$ stellt die Fläche des von den beiden Gradienten $\frac{\partial \xi_1}{\partial x}$ und $\frac{\partial \xi_2}{\partial x}$ aufgespannten Parallelogramms dar. Dieser Wert kann somit an einem gegebenen Punkt im Volumen als Betrag des Kreuzprodukts der beiden Gradienten berechnet werden:

$$(2.2) \quad |Vol(D\tau)(\hat{x})| = \left| \frac{\partial \xi_1}{\partial x} \times \frac{\partial \xi_2}{\partial x} \right|$$

Um das Integral (2.1) für jede Wertekombination (ξ_1, ξ_2) zu bestimmen, wird in [BWo8] dasselbe Muster wie beim Projected Tetrahedra Algorithmus [ST90] verwendet. Genau wie beim Volumenrendern mit Projected Tetrahedra, werden die einzelnen Tetraeder des gegebenen Grids auf maximal 4 Dreiecke abgebildet. Anstatt diese auf die Bildebene zu projizieren, werden sie jedoch mit entsprechenden Geometrie- und Texturkoordinaten in den Scatterplot abgebildet. Die Formel (2.2) muss pro Tetraeder nur einmal berechnet werden, weil die Gradienten in Tetraedern aufgrund des baryzentrischen Interpolationsschemas immer konstant sind. Durch Überlagern (additives Blending) aller Dreiecke werden pro Wertekombination (ξ_1, ξ_2) automatisch die vollen Längen der Isolinien berücksichtigt.

2.7.2 Kontinuierliche Parallele Koordinaten

Kontinuierliche Parallele Koordinaten [HWo9] sind Dichteplots, die für multivariate Daten mit kontinuierlichem Definitionsbereich geeignet sind. Ihr Aufbau ist derselbe wie bei normalen Parallelen Koordinaten, jedoch wird anstatt vieler sich überlagernder Linien wieder eine übersichtliche Dichteverteilung eingezeichnet. In [HWo9] wird ein Algorithmus beschrieben, mit dem man auf der Grundlage eines zuvor gerenderten kontinuierlichen Scatterplots mit relativ geringem Aufwand einen Parallele Koordinaten Plot ermitteln kann. Der Algorithmus

kann entweder so realisiert werden, dass er die vom Scatterplot Algorithmus (siehe 2.7.1) projizierten Dreiecke wiederverwendet, oder sie neu aus den Volumendaten ermittelt. Ein Dreieck in einem Scatterplot entspricht in Parallelen Koordinaten einer Footprint-Geometrie dessen Gestalt mit 3 charakteristische Linien, oder mit einer kleinen Zahl von Vierecken beschrieben werden kann. Der Algorithmus zeichnet die Vierecke mit OpenGL Quads unter Verwendung eines Fragment-Shaders in die Parallelen Koordinaten ein. Der Shader berechnet dann für jeden Pixel einen Wert, der sich aus einer ebenfalls in [HW09] hergeleiteten Formel für die Dichte in Parallelen Koordinaten ergibt. Um diese Formel auszuwerten, benötigt man lediglich die Dichten der drei Ecken des aktuellen Dreiecks, und die Position in Parallelen Koordinaten deren Dichte berechnet werden soll.

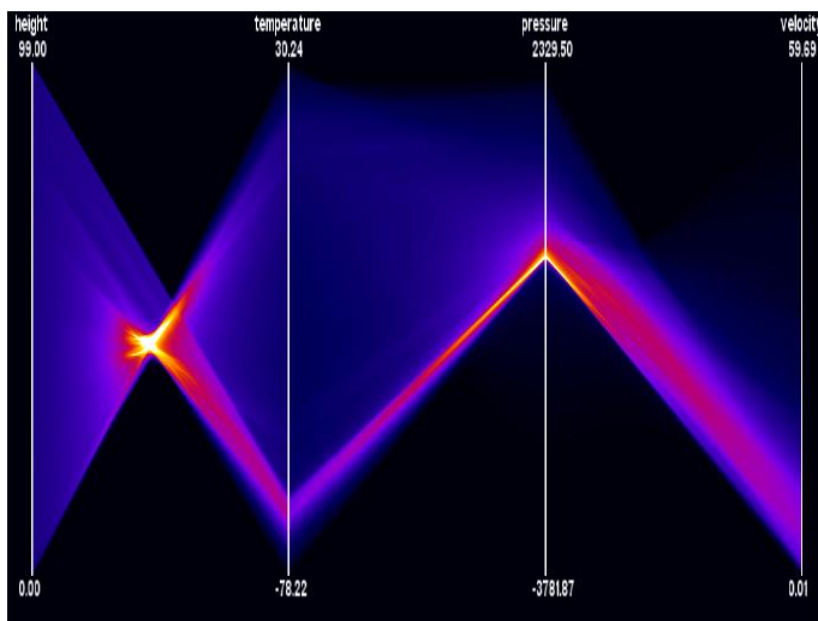


Abbildung 2.4: Kontinuierliche Parallele Koordinaten. Gerendert mit dem Ansatz von [HW09]

2.8 Splatting

In dieser Studienarbeit sollte für das Rendern kontinuierlicher Streudiagramme und Paralleler Koordinaten ein "Splatting"-Ansatz verwendet werden. Splatting ist ein Verfahren, das ursprünglich von Lee Westover für Volumenrendering entwickelt wurde (siehe [Wes90]). Es handelt sich dabei um einen forward-mapping Ansatz für direktes Volumenrendering. Durch seine Parallelisierbarkeit und durch Verwendung von Lookup-Tabellen erlaubt er eine interaktive Visualisierung von Volumendaten, die auf einem rektilinearen Grid definiert sind. Darüberhinaus ermöglicht er es, die Darstellung adaptiv zu verfeinern. Der Algorithmus rendert einen Volumendatensatz, indem er ihn in, zum Viewport parallele, Scheiben aufgeteilt sampelt. Jeder der Samples durchläuft 4 Schritte [Wes90]:

- Transformieren: Der Sample wird auf Basis seiner Grid-Position auf seine Viewport-Position abgebildet.
- Shading: Dem Sample wird eine Farbe und ein alpha-Wert zugewiesen.

- **Rekonstruktion:** Es wird die Region des Viewport bestimmt, die der Sample beeinflussen kann. Der Beitrag des Sample wird im sogenannten "sheet accumulator" aufaddiert.
- **Visibility:** Sobald eine Scheibe vollständig gesampelt wurde, wird ihr "Sheet Accumulator" über das bereits vorhandene Bild geblendet. Dabei wird ein Standard-Compositing Operator verwendet.

Der Rekonstruktionsschritt wird effizient durch das Sampeln einer "Footprint Function" unterstützt. Westover [Wes90] nennt den Vorgang, bei dem die Footprint Function erzeugt wird, dann gesampelt wird und ihr Beitrag im Sheet Accumulator verteilt wird, "Splatting". Dem Splatting liegt folgende Idee zugrunde: Ein Signal, also z.B. die Dichtewerte eines Volumendatensatzes, kann durch Faltung mit einem Kernel rekonstruiert werden. Der Ansatz nutzt nun die Tatsache, dass man Faltung als die Verteilung der "Energie" von Abtastpunkten in den Raum ansehen kann. Genau das passiert beim Aufaddieren eines Splat in den Sheet Accumulator. Das Volumenrendern kann dadurch deshalb beschleunigt werden, weil die Footprint Function für alle Samples bei gleichbleibendem Augpunkt und Blickrichtung dieselbe ist. Für eine Ansicht wird also nur einmal eine Footprint Function erzeugt. Dies geschieht mit Hilfe einer "Generic Footprint Function", die für alle Ansichten gleich ist.

3 Lösungsansatz

3.1 Ansatz für die Beschleunigung des Renderns

Analog zu dem in 2.8 genannten Verfahren sollten kontinuierliche Streudiagramme und Parallele Koordinaten mit einem Splatting-Ansatz so implementiert werden, dass das Rendering progressiv abläuft. Dazu wird im i -ten Rendschritt der Datensatz $i * N$ mal an möglichst zufällig gewählten Orten abgetastet. Auf der Grundlage dieser $i * N$ Abtastwerte wird ein Bild der Masse $M_{spatial} = s * Volumen_{Datensatz}$ (mit $s = 1$) erzeugt, dass dann über das vorherige Bild mit $\alpha = 0.5$ geblendet wird. Die Masse in dem neu entstandenen Bild ist wieder genau $M_{spatial}$:

$$M_{neu} = \alpha * M_1 + (1 - \alpha) * M_2 = \alpha * M_{spatial} + M_{spatial} - \alpha * M_{spatial} = M_{spatial}$$

Die Abtastung sollte möglichst gut auf das Volumen verteilt sein, was durch Verwendung einer "Low-Discrepancy Sequence" sichergestellt wird (siehe Kapitel 4.1). Für jeden Abtastort werden die Datenwerte und die Gradienten erhoben. Mit Hilfe dieser Daten wird der Sample in den kontinuierlichen Scatterplot bzw. in die Parallelen Koordinaten abgebildet. Dieser Vorgang ist in diesem Fall das Splatting. Dabei wird jeweils eine einfache Repräsentation des Samples über den bereits gerenderten Plot überlagert, sie wird "Splat" genannt. Die Gradientenbeträge bestimmen dabei das Seitenverhältnis der Splats. Die Datenwerte legen direkt deren Position im Plot fest. Die Dichtewerte, die im Plot aufgetragen werden, ergeben sich bei Scatterplots aus der in Kapitel 2.7.1 beschriebenen Formel 2.1 für die Dichte einer Datenwertkombination. Jedoch erfolgt keine explizite Integration, sondern es wird diese Formel für jeden Sample berechnet:

$$(3.1) \sigma(\xi_0)_{Scatterplot} = \frac{1}{\left| \frac{\partial \xi_1}{\partial x} \times \frac{\partial \xi_2}{\partial x} \right|} = \frac{1}{Vol_{Splat}}$$

Die Fläche der Splats ϕ wird analog zum 1D Fall festgelegt. Liegt eine 1D Spatial Domain und eine 1D Data Domain vor, so ergibt sich ϕ als Produkt aus Gradientenbetrag und der Größe des Splats in der Spatial Domain (siehe Abbildung 3.1). Im Fall einer 3D Spatial Domain und einer 2D Data Domain ersetzt man nun den Gradientenbetrag durch das Volume Measure: $\phi = V_s * Vol_{Splat}$ Dabei ist V_s das Volumen eines Splats in der Spatial Domain, also das Volumen des Datensatzes durch die Anzahl der Splats.

$$(3.2) V_s = \frac{Volumen_{Datensatz}}{i * N}$$

Die Masse ist dann in jedem Schritt i des Renderns für alle Splats dieselbe:

$$M = \phi * \sigma(\xi_0)_{Scatterplot} = V_s * Vol_{Splat} * \frac{1}{Vol_{Splat}} = V_s$$

Beim Splatting wird also immer die Masse V_s mit einem entsprechenden Kernel über einen Splat verteilt. Da $V_s = \frac{Volumen_{Datensatz}}{i * N}$ gilt und $i * N$ Splats gerendert werden, hat der Plot der

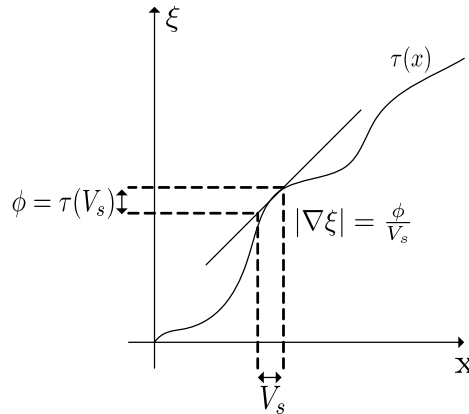


Abbildung 3.1: Die Abbildung zeigt den Zusammenhang zwischen Größe der Splats in der Data- ($\tau(V_s)$) und der Spatial-Domäne (V_s) im 1D-Fall.

Algorithmus 3.1 Allgemeiner Splatting-Ansatz für kontinuierliche Plots. Kann direkt auf Scatterplots angewendet werden. Bei Parallelen Koordinaten wird das Innere der Schleife für alle Achsenpaare je einmal ausgeführt.

```
procedure DOSPLATTING( $\Delta N$ )
```

```
  for  $c = 0$  to  $(\Delta N - 1)$  do
```

```
     $p \leftarrow$  NEXTHALTONPOINT
```

```
     $\xi_1 \leftarrow$  GRID.INTERPOLATE( $p, Dim = 1$ )
```

```
     $\xi_2 \leftarrow$  GRID.INTERPOLATE( $p, Dim = 2$ )
```

```
     $\nabla \xi_1 \leftarrow$  GETGRADIENT( $p, Dim = 1$ )
```

```
     $\nabla \xi_2 \leftarrow$  GETGRADIENT( $p, Dim = 2$ )
```

```
     $(l_1, l_2) \leftarrow$  GETSPLATEXTENTS( $\nabla \xi_1, \nabla \xi_2$ )
```

```
    RENDERSPLAT( $\xi_1, \xi_2, l_1, l_2, 1$ ) // Zeichne einen Splat mit Masse  $V_s$  mit Ausdehnung  $l_1$  entlang der  $\xi_1$  Achse und Ausdehnung  $l_2$  entlang der  $\xi_2$  Achse.
```

```
  end for
```

```
end procedure
```

entsteht immer genau die Masse $Volumen_{Datensatz}$. Der Kernel ist bei kontinuierlichen Scatterplots z.B. eine einfache (3D) Box-Funktion. Dabei gewichtet man genau die Dichte (3.1) mit der Box-Funktion, für jeden Pixel der zum Splat gehört. Oder man verwendet alternativ einen Gauss-Kernel. Ausgehend von dem einfachen Box/Rechteck-Splat in Scatterplots, ergibt sich die Dimensionierung aller anderen Splat-Typen sowohl in Scatterplots als auch in Parallelen Koordinaten. Da die Masse pro Splat immer dieselbe ist, kann auf Grundlage der Dimensionierung die passende Dichteverteilung gewählt werden. An dieser Stelle sei noch angemerkt dass der Kernel der Footprint Function beim Volumenrendern entspricht, wie es in 2.8 besprochen wurde. Die Integration über den Dichten (2.1) fällt nicht weg, sondern sie entspricht genau der Überlagerung der Splats. Der Vorgang ist in Algorithmus 3.1 zusammengefasst.

3.2 Aufbau des Plug-Ins

Kontinuierliche Scatterplots und Parallele Koordinaten gehören nicht zur Kernfunktionalität von MegaMol, deshalb sollten sie in Form eines Plug-Ins realisiert werden. Ein Plug-In

besteht aus Modulen und sonstigen Klassen, die untereinander zum Zweck des Renderings kommunizieren können (näheres dazu siehe in Abschnitt 3.2.2).

3.2.1 Das bestehende Plug-In für diskrete Plots

Das Plug-In wurde auf der Basis eines bereits vorhandenen Plug-Ins für diskrete Plots programmiert. Dieses stellt Module für das Rendern von Scatterplots (ScatterplotRenderer.h) und Parallelen Koordinaten (PCRenderer.h) für multivariate Datensätze bereit. Jedoch berücksichtigen diese nicht das Interpolationsschema der Daten. Die verwendeten Algorithmen verwenden eine Binning-Technik, bei der jeder Pixel eines Plots einen Behälter (Bin) darstellt. Jeder Voxel wird auf seine entsprechende Position im Scatterplot abgebildet und sein Footprint (ein Pixel) wird über den bestehenden Plot additiv geblendet. Analog wird bei Parallelen Koordinaten eine Kette von Liniensegmenten über den vorhandenen Plot geblendet. Vor diese Renderer wird ein Normalisierungsmodul geschaltet (DensityRenderer.h). Es normalisiert die Pixel auf den Intervall $[0, 1]$: $gl_FragColor.rgb = \frac{Pixel.r - min_r}{max_r - min_r}$

3.2.2 Kommunikation zwischen den Renderern des Plug-Ins

Die Renderer des bestehenden Plug-Ins und die neu implementierten Renderer kommunizieren untereinander ausschliesslich über MegaMol Calls, die wie die Renderer selbst, auch als Klassen implementiert wurden. Einzelne Instanzen dieser Klassen können zwischen Mega-Mol Modulen ausgetauscht werden. Das sendende Modul wird dabei "Caller" genannt. Der Empfänger ist der "Callee". Damit ein solcher Austausch stattfindet, werden in einer XML-Konfigurationsdatei Caller und Callee miteinander verknüpft. Zudem muss zur Entwicklungszeit seitens des Callee ein entsprechender Slot deklariert werden. Sobald er aktiviert ist, kann der Caller den Call starten. Dessen Empfänger kann zur Ausführungszeit über die Konfigurationsdatei frei gewählt werden. In der XML-Konfigurationsdatei deklariert man eine Ansicht ("View"), indem man einzelne Module über Calls zusammenschaltet. Man deklariert die notwendigen Module, also die verschiedenen Renderer, sowie die Calls die dazwischen ausgetauscht werden sollen.

3.2.3 Das Plug-In für kontinuierliche Scatterplots und Parallele Koordinaten

Abbildung 3.2 zeigt die Klassenhierarchie, die dem Plug-In zugrundeliegt. Dabei wurden im Rahmen der Studienarbeit neu implementiert: SplattingRendererBlend, SplattingSPRendererBlend, SPRectangles, SplattingPCRendererBlend, PCTrapezoids, ColorMapRendererRGB, HaltonGenerator, fbotools. Darüber hinaus wurde der DensityRenderer leicht umgewandelt und heißt jetzt DensityRendererRGB. Die Klassen für kontinuierliche Plots erben von SplattingRendererBlend und von Renderer2DModule, wovon letztendlich jeder Megamol Renderer erbt. SplattingRendererBlend implementiert alle Gemeinsamkeiten der Renderer, insbesondere das progressive Rendering wird hier ausgeführt und verwaltet. Dazu benutzt die Klasse 2 FBO's, auf die mit den Zeigern "fbo" und "next_fbo" verwiesen wird. "fbo" zeigt immer auf das FBO, das das letzte Zwischenergebnis des Renderns beinhaltet. In das FBO "next_fbo" erfolgt in jeder Runde zunächst das Splatting, dann wird das Ergebnis davon mit dem letzten Zwischenergebnis aus "fbo" überblendet, sofern progressives Rendern erwünscht ist (Wenn das der Fall ist, setzt man die Variable mode in der Create Methode von SplattingRendererBlend auf PROGRESSIVE, sonst auf NON_PROGRESSIVE.) Nach dem Blenden, zu Beginn des nächsten

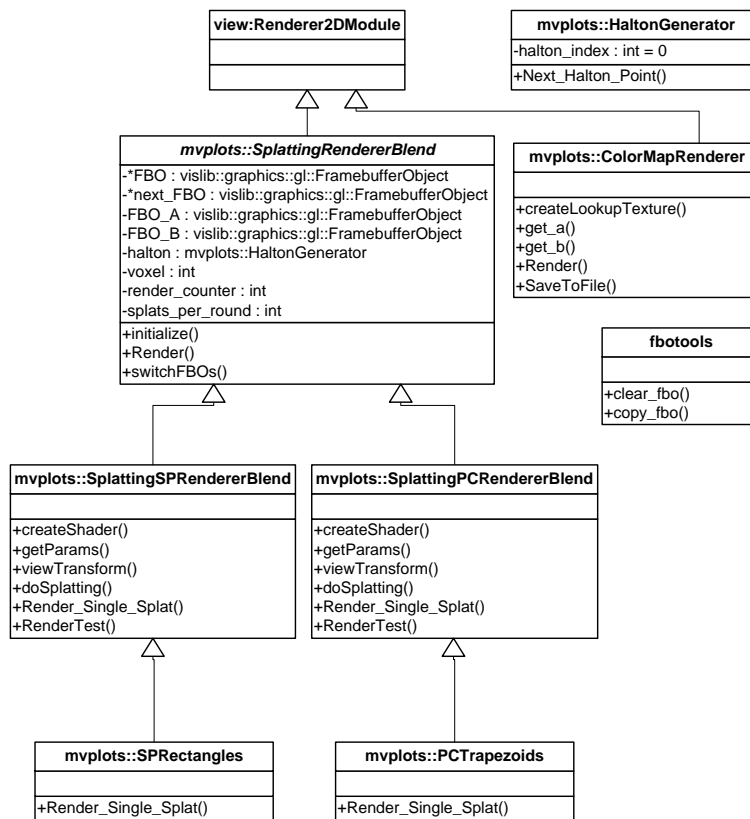


Abbildung 3.2: UML-Klassendiagramm der implementierten Klassen

Renderer	Plot-Typ	Splat-Typ
SplattingSPRendererBlend	kontinuierliche Scatterplots	Gauss-Splats
SPRectangles	kontinuierliche Scatterplots	Rechteck-Splats
SplattingPCRendererBlend	kontinuierliche Parallele Koordinaten	Gauss-Splats
PCTrapezoids	kontinuierliche Parallele Koordinaten	Trapez-Splats

Tabelle 3.1: Im Rahmen der Studienarbeit implementierte Renderer.

Rendererschrittes, werden die Ziel-FBO's der beiden Zeiger vertauscht.

SplattingSPRendererBlend und SplattingPCRendererBlend implementieren u.a. die abstrakten Methoden `viewTransform()` und `doSplatting()` der Klasse `SplattingRendererBlend`. `viewTransform()` ist dafür zuständig, die Model-View Matrix von OpenGL so zu konfigurieren, dass hinterher auf direkte Art und Weise in den entsprechenden Plot gezeichnet werden kann, ohne Plot-Koordinaten in OpenGL-Fensterkoordinaten umrechnen zu müssen. Dabei wurde sowohl für die Scatterplots als auch für die Parallelen Koordinaten eine lokale Skalierung verwendet, das heißt dass jede Dimension entsprechend ihrem (lokalen) Minimal- und Maximalwert skaliert wird, sodass der Plot den gesamten zur Verfügung stehenden Bildschirmbereich verwendet. `doSplatting()` ist dafür zuständig, pro Runde $i * N$ Splats zu zeichnen. Die Arbeitsweise von `doSplatting()` wird in den Kapiteln 4.2 und 5 behandelt, und ist in Algorithmus 3.1 kurz zusammengefasst.

4 Zufällige Auswahl von Samples, Berechnung der Skalarwerte und Gradienten

Die Orte, an denen ein Sampling erfolgt, müssen möglichst zufällig gewählt werden und gut über das Volumen verteilt sein. Dadurch soll die Entstehung von Mustern vermieden werden, die sonst nur aufgrund von uniformen oder regelmäßigen Abtastschemata in Streudiagrammen oder Parallelen Koordinaten auftreten. Zudem sollte ein und derselbe Ort höchstens einmal abgetastet werden.

4.1 Quasi-Random Sequences

Quasi-Random Sequenzen sind Zahlenreihen, die eine bessere Verteilung der Punkte im Raum erreichen, als einfache Zufallszahlen. Ein Maß dafür wie gut die Punkte einer Sequenz im Raum verteilt sind ist die Diskrepanz. Eine niedrige Diskrepanz bei einer Sequenz besagt, dass eine gute Verteilung erreicht wird. Deshalb werden Quasi-Random Sequenzen auch "Low-Discrepancy Sequences" genannt. Bei der Ermittlung der Diskrepanz betrachtet man für jeden von der Sequenz ausgegebenen Punkt $x = (x_1, x_2, \dots)$ den Wert einer Funktion $d(x)$. Sie berücksichtigt die Lage aller Punkte und deren Gesamtzahl. Das Supremum dieser Funktionswerte ist die Diskrepanz. Eine formale Definition ist in [BF88] zu finden. Abbildung 4.1 zeigt als 2D-Beispiel die Diskrepanz bei einer schlechten Verteilung von $N = 10$ Punkten (links) und die Diskrepanz bei einer guten Verteilung (rechts). $d(x)$ ergibt sich im 2D-Beispiel für den grün markierten Punkt als die Anzahl der vom roten Rechteck eingeschlossenen Punkte minus der Gesamtzahl der Punkte mal der Fläche des roten Rechtecks.

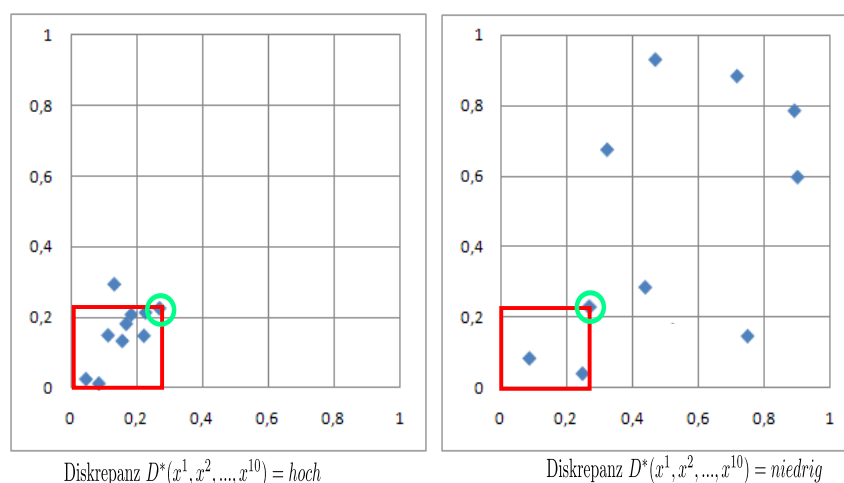


Abbildung 4.1: Zwei Zufalls-Sequenzen mit $N = 10$ Punkten. Links ist die Verteilung schlecht, rechts ist sie besser.

Die nachfolgenden Sequenzen x_n haben eine gute Diskrepanz und erfüllen die "self-avoiding"-Anforderung. Das heißt, dass für alle n bekannt ist, welche Zahlen bereits für $n' < n$ aufgetreten sind und dass diese automatisch vermieden werden. Somit gilt immer $\forall n' < n : x_n \neq x_{n'}$.

4.1.1 Sobol Sequences

Ein Vorschlag für eine solche Sequenz kommt von Sobol [Sob67]. Die Implementation davon ist in [BF88] beschrieben. Auf dieser Grundlage soll hier die Berechnung einer 3D-Sobol Sequenz diskutiert werden. Eine 3D-Sobol Sequenz erhält man, indem man zunächst drei primitive Polynome P,Q,R auf dem Körper Z_2 aussucht. Primitive Polynome [Knu97] sind Polynome, für deren Koeffizienten a_1, a_2, \dots gilt, dass ihr größter gemeinsamer Teiler 1 ist. Das ist auf dem Körper Z_2 für beliebige Koeffizienten $a_1, a_2, \dots \in \{1, 0\}$ der Fall. Die Autoren von [BF88] schlagen für solche Polynome diese Form vor: $P \equiv x^d + a_1 x^{d-1} + \dots + a_{d-1} x^{d-1} + 1$. Z.B. könnte man wie in [PFT⁺86] wählen:

$$P \equiv x^3 + x + 1 \text{ mit } a_1 = 0, a_2 = 1$$

$$Q \equiv x^3 + x^2 + 1 \text{ mit } a_1 = 1, a_2 = 0$$

$$R \equiv x^4 + x + 1 \text{ mit } a_1 = 0, a_2 = 0, a_3 = 1$$

Anschließend werden nur noch die Koeffizienten benötigt, um pro Polynom eine rekursive Gleichung [BF88] aufzustellen.

$$m_i = 2a_1 m_{i-1} \oplus 2^2 a_2 m_{i-2} \oplus \dots \oplus 2^{d-1} a_{d-1} m_{i-d} \oplus 2^d m_{i-d} \oplus m_{i-d}$$

\oplus ist dabei die XOR Verknüpfung zweier binär-dargestellter Zahlen. Für P,Q,R ergibt sich also

$$m_{P,i} = 2^2 m_{P,i-2} \oplus 2^3 m_{P,i-3} \oplus m_{P,i-3}$$

$$m_{Q,i} = 2m_{Q,i-1} \oplus 2^3 m_{Q,i-3} \oplus m_{Q,i-3}$$

$$m_{R,i} = 2^3 m_{R,i-3} \oplus 2^4 m_{R,i-4} \oplus m_{R,i-4}$$

Die Berechnung kann mit folgender Initialisierung starten:

$$P : m_{P,1} = 1, m_{P,2} = 3, m_{P,3} = 7$$

$$Q : m_{Q,1} = 1, m_{Q,2} = 3, m_{Q,3} = 7$$

$$R : m_{R,1} = 1, m_{R,2} = 3, m_{R,3} = 7, m_{R,4} = 9$$

Es müssen dabei immer ungerade Zahlen m_i kleiner 2^i gewählt werden. Mit der Vorschrift

$$v_{X,i} = \frac{m_{X,i}}{2^i}$$

lassen sich dann die sogenannten Direction Numbers für $X = P$, $X = Q$, und $X = R$ ermitteln. Das sind Binärbrüche kleiner 1: $v_{X,i} = 0.v_{X,i1}v_{X,i2}v_{X,i3}...$ Die 3D-Sobol Sequenz ergibt sich dann als:

$$x_n = \begin{pmatrix} b_1 v_{P,1} \oplus b_2 v_{P,2} \dots \\ b_1 v_{Q,1} \oplus b_2 v_{Q,2} \dots \\ b_1 v_{R,1} \oplus b_2 v_{R,2} \dots \end{pmatrix}$$

Algorithmus 4.1 Umwandlung eines Bitstrings der Länge m in einen float-Wert.

```

function GETFLOAT( $b = 0.b_1b_2b_3\dots b_m$ )
  for  $c = 1$  to  $m$  do
     $result \leftarrow result + b_c * 2^{-c}$ 
  end for
   $getFloat \leftarrow result$ 
end function

```

Dabei sind die b_i die Binärziffern von n : $n = (...b_3b_2b_1)_2$. Desto größer n wird, desto mehr Binärziffern hat n und somit werden entsprechend mehr Zahlen $v_{x,i}$ benötigt. Die Komponenten von x_n liegen zunächst nur als Bitsequenz vor und müssen somit abschließend noch in einen Float oder Double Vektor umgewandelt werden. Dazu muss jede der 3 Komponenten von x_n wie in Algorithmus 4.1 verarbeitet werden.

4.1.2 Hammersley Sequences [Hal60]

Ein weiterer Vorschlag für eine Low-Discrepancy Sequence kommt von Hammersley [Hal60]. Jede ganze Zahl n lässt sich mit jeder ganzen Zahl R als Basis darstellen:

$$n \equiv (n_M n_{M-1} \dots n_1 n_0)_R = n_0 + n_1 * R^1 + n_2 * R^2 + \dots + n_M * R^M$$

Wenn man die Ziffern von n bezüglich der Basis R anstatt mit R^0 bis R^M mit R^{-1} bis R^{-M-1} multipliziert, bekommt man eine Zahl zwischen 0 und 1:

$$(4.1) \quad \varphi_R(n) = n_0 * R^{-1} + n_1 * R^{-2} + n_2 * R^{-3} + \dots + n_M * R^{-M-1}$$

Eine 3D-Hammersley Sequenz der Länge N erhält man, wenn man φ für 2 verschiedene Basen $R_1 \neq R_2$ mit aufsteigendem n berechnet:

$$x_n = \begin{pmatrix} \frac{n}{N} \\ \varphi_{R_1}(n) \\ \varphi_{R_2}(n) \end{pmatrix}$$

Dabei müssen als Basen Primzahlen gewählt werden. Eine solche Sequenz hat jedoch den Nachteil, dass man von Anfang an auf N Samples festgelegt ist.

4.1.3 Halton Sequences [Hal60]

Halton hat diese Sequenz leicht modifiziert und erfand damit eine von N unabhängige Quasi-Random Sequenz. Eine 3D-Halton-Sequenz erhält man, wenn man φ für 3 verschiedene Basen $R_1 \neq R_2 \neq R_3$ mit aufsteigendem n berechnet:

$$x_n = \begin{pmatrix} \varphi_{R_1}(n) \\ \varphi_{R_2}(n) \\ \varphi_{R_3}(n) \end{pmatrix}$$

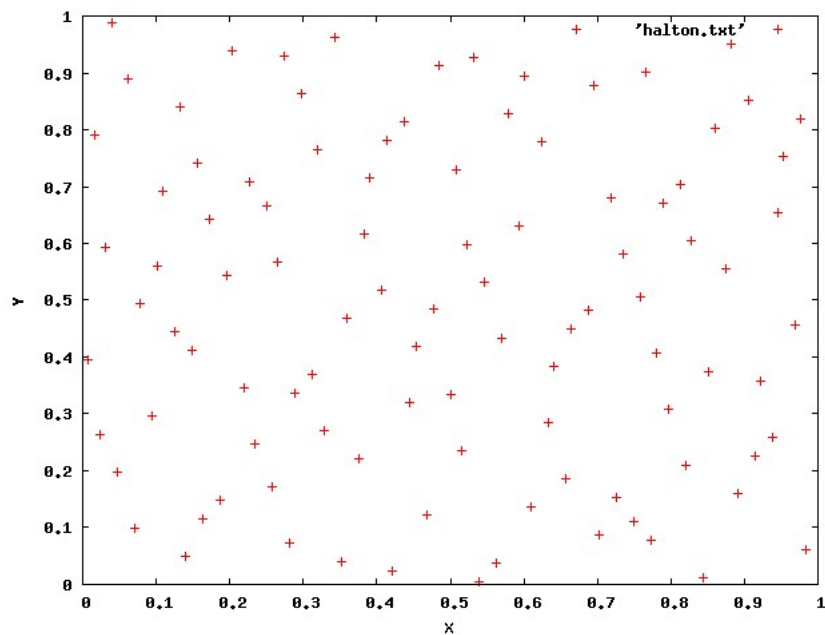


Abbildung 4.2: Die ersten 100 Paare einer 2D-Halton-Sequenz. Als Basen für die Berechnung wurden 2 und 3 gewählt.

4.1.4 Vergleich der Sequenzen

Die vorgestellten Sequenzen haben, mit Ausnahme der Hammersley-Sequenzen, eine niedrige Diskrepanz [BF88]. Sie ist bei den Halton Sequenzen etwas niedriger als bei den Sobol Sequenzen. Sie wird jedoch laut [BF88] von den Niederreiter-Sequenzen noch unterboten [Nie87]. Die Sequenzen erfüllen die self-avoiding Anforderung. Sie sind im Gegensatz zu den Niederreiter-Sequenzen relativ leicht zu implementieren. Tabelle 4.1 zeigt die Anzahl elementarer Operationen, die zur Berechnung des n -ten Elements der Sobol- bzw. Halton Sequenz pro Dimension ausgeführt werden müssen. Dabei wurde davon ausgegangen dass die Direction Numbers für die Sobol Sequenz bereits vorberechnet wurden. Der Aufwand ist also bei den Halton Sequenzen etwas geringer. Aufgründessen und wegen der niedrigeren Diskrepanz wurden in dieser Studienarbeit Halton Sequenzen implementiert (siehe 4.1.5).

Algorithmus	\oplus	+	*
Sobol Sequences	$m-1$		$2*m$
Halton Sequences		$m-1$	m

Tabelle 4.1: Anzahl Operationen pro Dimension, bei der Berechnung eines Elements x_n der Sobol-Sequenz bzw. Halton Sequenz. m ist die Anzahl der Binärstellen von n . Hinzu kommt bei den Sobol-Sequenzen noch die Vorberechnung der $v_{X,i}$ und bei den Halton-Sequenzen noch Potenz-Berechnungen, die aber tabelliert und bei Bedarf ausgelesen werden können.

4.1.5 Implementation der Halton Sequences

Die Methode `NextHaltonPoint()` der Klasse `HaltonGenerator` liefert pro Aufruf ein Element einer 3D-Halton-Sequenz. Dabei werden die Basen 2,3 und 5 verwendet. Die Klasse kapselt 3 Arrays und speichert darin je eine Zahl zu den 3 Basen. `NextHaltonPoint()` inkrementiert die Zahlen entsprechend ihrer Basis (die Zahlen sind anfangs 0) und liefert die entsprechenden Ergebnisse von Gleichung (4.1) als Vektor v zurück ($v \in [0, 1] \times [0, 1] \times [0, 1]$). Wenn ein uniformes Gitter vorliegt, können dessen Abmessungen mit diesem Vektor multipliziert werden, was den Ortsvektor eines Samples ergibt.

4.2 Sampling und Ermittlung der Gradienten

Nachdem ein Abtastpunkt mit Hilfe der Sequenz gewählt wurde, wird dieser Punkt in den Volumendaten gesampelt. Das heißt, es werden im Fall eines Scatterplots die beiden Skalarwerte ξ_1 und ξ_2 , der zu plottenden Skalarfunktionen, ausgelesen. Außerdem wird der Gradient ermittelt. Dies geschieht durch Bildung zentraler Differenzen. (siehe Gleichung (4.2) und (4.3)). Dazu wird im Abstand $\Delta_{sampling} = 1$ um den Abtastpunkt herum entlang jede der 3 Achsen je 2 mal gesampelt. Hat der Abtastpunkt einen zu geringen Abstand zum Rand des Datensatzes (Abstand kleiner 1), so wird er nicht gesplattet. Abbildung 4.3 zeigt ein Beispiel für einen 2D-Beobachtungsraum. Für Parallele Koordinaten wird dasselbe gemacht, nur dass hier für jedes Liniensegment je zwei Gradienten ermittelt werden.

$$(4.2) \quad g_1 = \nabla \xi_1 = \frac{\begin{pmatrix} \xi_{1,x+\Delta_{sampling}} - \xi_{1,x-\Delta_{sampling}} \\ \xi_{1,y+\Delta_{sampling}} - \xi_{1,y-\Delta_{sampling}} \\ \xi_{1,z+\Delta_{sampling}} - \xi_{1,z-\Delta_{sampling}} \end{pmatrix}}{2 * \Delta_{sampling}}$$

$$(4.3) \quad g_2 = \nabla \xi_2 = \frac{\begin{pmatrix} \xi_{2,x+\Delta_{sampling}} - \xi_{2,x-\Delta_{sampling}} \\ \xi_{2,y+\Delta_{sampling}} - \xi_{2,y-\Delta_{sampling}} \\ \xi_{2,z+\Delta_{sampling}} - \xi_{2,z-\Delta_{sampling}} \end{pmatrix}}{2 * \Delta_{sampling}}$$

Alternativ können die Gradienten auch auf Grundlage der 8 einen Sample umgebenden Voxel berechnet werden: Dazu leitet man die (trilineare) Interpolationsfunktion ab und setzt die 8 Voxelwerte ein. Oder man bestimmt an jedem Voxel einen Gradienten und interpoliert dann zwischen den Gradienten der 8 umgebenden Voxel.

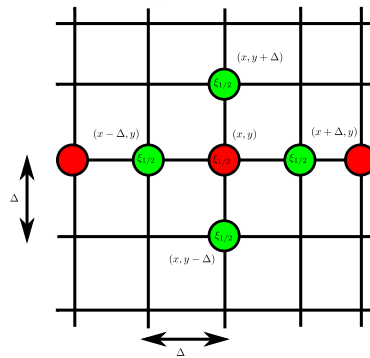


Abbildung 4.3: Bei Verwendung von Central Differences wird um ein Sample herum im Abstand Δ ($= \Delta_{\text{sampling}}$) nochmals gesampelt, und daraus der Gradient berechnet.

5 Splatting von kontinuierlichen Scatterplots und kontinuierlichen Parallelen Koordinaten

Jedes Sample wird entsprechend seiner Skalarwerte und seiner Gradienten in den Scatterplot bzw. den Parallele Koordinaten Plot abgebildet. In den folgenden beiden Abschnitten soll diese Abbildung und ihre Implementation beschrieben werden.

5.1 Kontinuierliche Scatterplots

Nachdem der Datenwert $\zeta = \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix}$ eines Abtastpunktes, sowie die Gradienten $\nabla\zeta_1$ und $\nabla\zeta_2$ erhoben sind, kann der entsprechende Dichtewert unter Verwendung eines geeigneten Abbildes, und eines geeigneten Kerns, in ein aktives Renderziel des Framebufferobjekts gerendert werden. Die Dichte ergibt sich aus Formel (3.1).

5.1.1 Beschreibung der Abbilder und der Berechnung der Dichte

”Ebene” Splats

Das Abbild eines Samples ist bei kontinuierlichen Streudiagrammen ein Rechteck, das mit seinem Zentrum genau an der Stelle $\zeta = \begin{pmatrix} \zeta_1 \\ \zeta_2 \end{pmatrix}$ platziert wird. Seine Seitenlängen l_1 und l_2 ergeben sich aus dem Verhältnis der Gradientenbeträge zueinander, sowie aus dem Volume Measure $Vol_{Splat} = |(\nabla\zeta_1 \times \nabla\zeta_2)|$ und der Größe V_s der Splats in der Spatial Domain. Zunächst

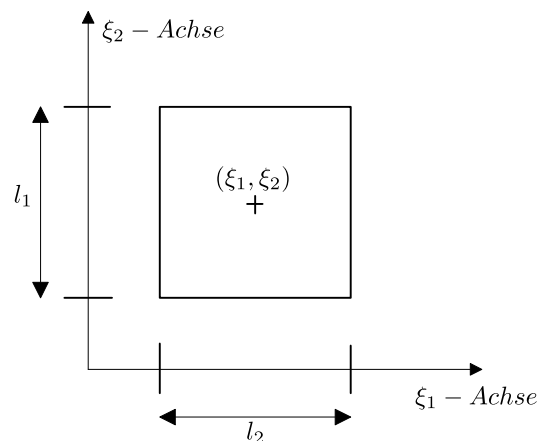


Abbildung 5.1: Die Abbildung zeigt einen rechteckigen Splat für Scatterplots. Die Seitenlängen l_1 und l_2 sind proportional zu den Gradientenbeträgen $|\nabla\zeta_1|$ und $|\nabla\zeta_2|$.

wird die Fläche ϕ der Splats in der Data Domain analog zum 1D-Fall ermittelt. (Wie bereits in 3.1 erläutert.)

$$(5.1) \quad \phi = V_s * Vol_{Splat} = l_1 * l_2$$

Außerdem gilt:

$$(5.2) \quad \frac{l_1}{l_2} = \frac{|\nabla \xi_1|}{|\nabla \xi_2|}$$

Wenn man Gleichung (5.1) nach l_1 auflöst erhält man:

$$(5.3) \quad l_1 = \frac{V_s * Vol_{Splat}}{l_2}$$

Auflösen von (5.2) nach l_2 und einsetzen in (5.3) ergibt für l_1 :

$$(5.4) \quad l_1 = \sqrt[2]{\frac{V_s * Vol_{Splat} * |\nabla \xi_1|}{|\nabla \xi_2|}}$$

Dann ist l_2 gegeben durch:

$$(5.5) \quad l_2 = \frac{\phi}{l_1} = \frac{V_s * Vol_{Splat}}{l_1}$$

Da sich die Wertebereiche der einzelnen Dimensionen der Datensätze voneinander unterscheiden können, wäre eigentlich zunächst eine Normalisierung der Gradientenbeträge und des entsprechenden Volume Measures auf den Intervall $[0, 1]$ notwendig. Jedoch muss am Ende dann wieder mit den Wertebereichen der beiden Dimensionen multipliziert werden. Da diese Faktoren sich aber wieder wegkürzen, kann direkt mit den Gleichungen (5.4) und (5.5) gearbeitet werden.

Gauss-Splats

Anstelle von "ebenen" Splats können für kontinuierliche Scatterplots auch Splats gezeichnet werden, die eine vom Zentrum nach außen hin abfallende Dichte aufweisen. Das kann man realisieren, indem man die Splats mit einer Gauss-Dichtefunktion shadet, wobei die Seitenlängen der Splats wie bei normalen Splats dimensioniert werden. Das Doppelintegral über der Gauss-Dichtefunktion [Bey99] geht mit entsprechend groß gewähltem Integrationsbereich gegen 1.

$$f(x, y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 - 2\rho\frac{x-\mu_x}{\sigma_x}\frac{y-\mu_y}{\sigma_y} + \left(\frac{y-\mu_y}{\sigma_y}\right)^2\right]\right)$$

Mit den Erwartungswerten: $(\mu_x, \mu_y) = (0, 0)$ und Korrelation $\rho = 0$ ergibt sich:

$$(5.6) \quad f(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{1}{2}\left[\left(\frac{x}{\sigma_x}\right)^2 + \left(\frac{y}{\sigma_y}\right)^2\right]\right)$$

Es gilt:

$$V = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (f(x, y)) dy dx = 1$$

Somit kann man einen Splat mit Masse V_s realisieren, indem man auf jeden Pixel, den der Splat beeinflusst, den (Dichte-)Wert $f(x, y) * V_s$ aufaddiert. In der Abbildung 5.2 kann man die Gaussverteilung sehen.

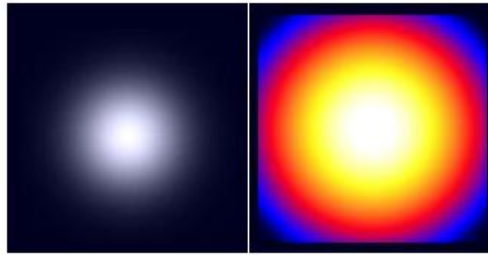


Abbildung 5.2: Links: Intensitätscodierte Dichteverteilung eines Splats. Rechts: Farbcodierte Dichteverteilung eines Splats.

5.1.2 Implementation des Splattings

”Ebene” Splats

Der Box-Kernel für Scatterplots wurde als einfaches OpenGL-Rechteck (GL_QUAD) realisiert. Es hat das Zentrum (ξ_1, ξ_2) und die Eckpunkte:

$$\begin{pmatrix} \xi_1 + \frac{l_1}{2} \\ \xi_2 + \frac{l_2}{2} \end{pmatrix} \begin{pmatrix} \xi_1 - \frac{l_1}{2} \\ \xi_2 + \frac{l_2}{2} \end{pmatrix} \begin{pmatrix} \xi_1 + \frac{l_1}{2} \\ \xi_2 - \frac{l_2}{2} \end{pmatrix} \begin{pmatrix} \xi_1 - \frac{l_1}{2} \\ \xi_2 - \frac{l_2}{2} \end{pmatrix}$$

Sollen normale, also ”ebene” Rechteck-Splats gezeichnet werden, so genügt es, vor dem Zeichnen des Quad, die aufzuaddierende Dichte als Farbe an OpenGL zu übergeben: $glColor(\frac{1}{Vol_{Splat}}, \frac{1}{Vol_{Splat}}, \frac{1}{Vol_{Splat}}, 0)$ Der Alphawert 0 hat keine besondere Bedeutung, da beim Blenden in der Methode `doSplatting()` ein konstanter Wert (1) verwendet wird:

```
glEnable(GL_BLEND);
glBlendColor(0.0f, 0.0f, 0.0f, 1.0f);
glBlendEquation(GL_FUNC_ADD);
glBlendFuncSeparate(GL_CONSTANT_ALPHA, GL_CONSTANT_ALPHA, GL_ONE, GL_ZERO);
```

Folglich werden die Dichtewerte jedes Splat auf das gerade aktuelle Zwischenergebnis aufaddiert.

Gauss Splats

Sollen Gauss-Splats gezeichnet werden, sind stattdessen folgende Schritte notwendig: Man normalisiert zunächst die Seitenlängen l_1 und l_2 der Splats mit dieser Funktion:

$$Normalize_{Extent}(l, dim) = \frac{(l)}{range_{\xi_{dim}}}$$

Mit

$$(5.7) \quad range_{\xi_{dim}} = \xi_{dim,max} - \xi_{dim,min}$$

(Größe des Wertebereiches der Dimension dim $range_{\xi_{dim}}$ ist gleich dem Maximalwert der Dimension dim minus ihrem Minimalwert.) Dann halbiert man diese Splatmaße und übergibt

sie als Texturkoordinaten für das OpenGL Quadrilateral, das nun mit einem aktivierten Fragmentshader gezeichnet wird:

$$\begin{pmatrix} \text{Normalize}_{Extent}\left(\left|\frac{l_1}{2}\right|,1\right) \\ \text{Normalize}_{Extent}\left(\left|\frac{l_2}{2}\right|,2\right) \end{pmatrix} \begin{pmatrix} -\text{Normalize}_{Extent}\left(\left|\frac{l_1}{2}\right|,1\right) \\ \text{Normalize}_{Extent}\left(\left|\frac{l_2}{2}\right|,2\right) \end{pmatrix} \begin{pmatrix} \text{Normalize}_{Extent}\left(\left|\frac{l_1}{2}\right|,1\right) \\ -\text{Normalize}_{Extent}\left(\left|\frac{l_2}{2}\right|,2\right) \end{pmatrix} \begin{pmatrix} -\text{Normalize}_{Extent}\left(\left|\frac{l_1}{2}\right|,1\right) \\ -\text{Normalize}_{Extent}\left(\left|\frac{l_2}{2}\right|,2\right) \end{pmatrix}$$

Und als Varianzen übergibt man als uniform Parameter: $\frac{1}{d}$ mal die Länge dieser halbierten Seitenlängen.

$$\sigma_1 = \frac{\text{Normalize}_{Extent}\left(\frac{l_1}{2}, dim\right)}{d}$$

$$\sigma_2 = \frac{\text{Normalize}_{Extent}\left(\frac{l_2}{2}, dim\right)}{d}$$

d bestimmt dabei, wie "schnell" die Gausskurve auf 0 abfällt, es wurde $d = 8$ gewählt. Außerdem übergibt man die gewünschte Gesamtmasse des Splat, also V_s ebenfalls als uniform Parameter. Dadurch wird erreicht, das in etwa die Masse V_s in den Scatterplot übertragen wird. Wenn man die Varianzen noch kleiner wählt kommt die Masse noch näher an V_s heran. In der aktuellen Version wird im Fragment Shader die Wahrscheinlichkeitsdichte wie in (5.6) der Gaussverteilung ausgerechnet und dann mit V_s multipliziert. Dadurch entsteht pro Fragment ein relativ hoher Rechenaufwand, dieser kann jedoch durch ein einmaliges Pre-processing, nämlich dem Tabellieren der 2D-Standardnormalverteilung verringert werden. Dazu berechnet man Werte der Dichte der 2D-Standardnormalverteilung für n^2 Paare $(x, y) \in [-8, 8] \times [-8, 8]$. Diese speichert man in einer Datei. Bei Programmstart füllt man einen 2D Array mit diesen Werten. Diesen Array übergibt man als Textur an den Fragmentshader. Dieser substituiert dann das Paar (x, y) durch das Paar $\left(\frac{x}{\sigma_x}, \frac{y}{\sigma_y}\right)$ und interpoliert mit diesen Werten einen Wert aus der Textur der Standardnormalverteilung. Anschließend teilt der Shader noch durch $\sigma_x * \sigma_y$ und multipliziert mit V_s . Dann wird das Ergebnis im RGB-Kanal des entsprechenden Pixels aufaddiert.

5.2 Kontinuierliche Parallele Koordinaten

5.2.1 Beschreibung der Abbilder und der Berechnung der Dichte

"Ebene" Splats

In Parallelen Koordinaten sind die Abbilder keine Rechtecke sondern Trapeze. Die Splats werden mit denselben Ausdehnungen entlang der ξ_1 und der ξ_2 -Achse dimensioniert (l_1 und l_2), wie in den Scatterplots: Es gelten also die Gleichungen (5.4) und (5.5) auch für Parallele Koordinaten. Allerdings soll die Masse der Splats aufgrund der Masseerhaltung auch in Parallelen Koordinaten V_s sein. Deshalb wird die pro Pixel aufzuaddierende Dichte σ auf den Kehrwert der Trapezfläche mal V_s gesetzt. Dann gilt wieder $m_{\text{Splat}} = G * h = G_{\text{Trapez}} * \sigma = \frac{l_1+l_2}{2} * \frac{2}{l_1+l_2} * V_s = V_s$

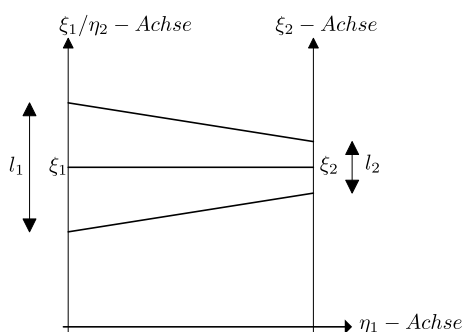


Abbildung 5.3: Ein einfacher Trapez-Splat in Parallelen Koordinaten.

Gauss-Splats

Auch in Parallelen Koordinaten kann man die Dichte bis zum Rand der Splats langsam abfallen lassen, indem man auf die Splats, die wieder dieselben Ausdehnungen auf der ξ_1 - und der ξ_2 -Achse haben, eine Gaussverteilung aufprägt. Jedoch handelt es sich dabei um eine 1D Gaussverteilung, die entlang der Mittellinie der Splats extrudiert wird (siehe Abbildung 5.4). Die Körper, die dabei entstehen, haben aufgrund der Anwendung der 1D-Gaussverteilung ohne

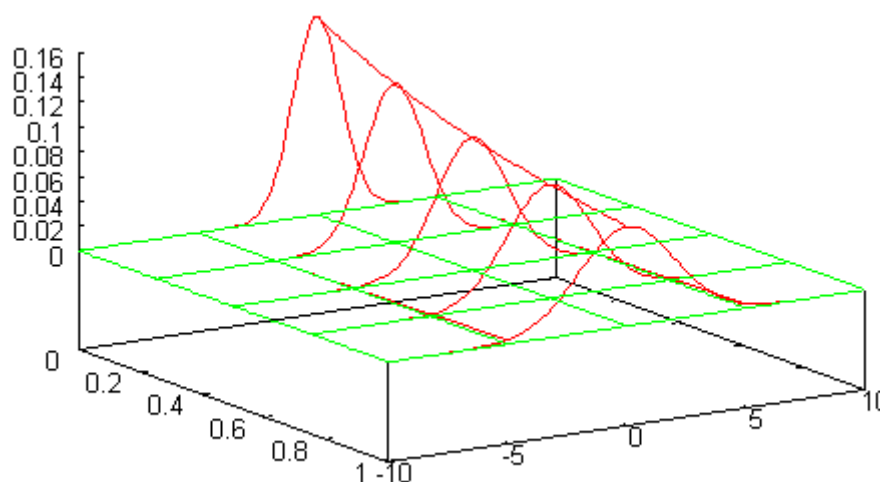


Abbildung 5.4: Ein Splat als Körper. Die in Parallelen Koordinaten vertikal verlaufenden 1D Gaussverteilungen sind Rot dargestellt. Die Varianzen der Gausskurven wurden in diesem Beispiel von $x = 0$ bis $x = 1$ zwischen $\sigma_0 = 1$ und $\sigma_1 = 2$ linear interpoliert.

Skalierung automatisch die Masse 1. Sei $f(x)$ die 1D-Gaussfunktion, die vertikal entlang des Splats verläuft. Ihr Integral über den Splat sei annähernd 1. Wenn man den Splat in Abständen von Δx entlang η_0 zerteilt, und Δx gegen 0 gehen lässt, so erhält man (unabhängig von der Scherung in η_2 -Richtung) einfache Körper mit definierter Grundfläche 1 und Höhe Δx . Davon entstehen genau $\frac{1}{\Delta x}$ Stück. Also gilt:

$$m_{\text{Splat}} = V = G \cdot h \lim_{\Delta x \rightarrow 0} \sum_{i=1}^{\frac{1}{\Delta x}} 1 * \Delta x = \frac{1}{\Delta x} * 1 * \Delta x = 1$$

Um die Gesamtmasse V_s für einen Splat zu erreichen, genügt also wieder eine Multiplikation der Höhe $f(x)$ mit V_s .

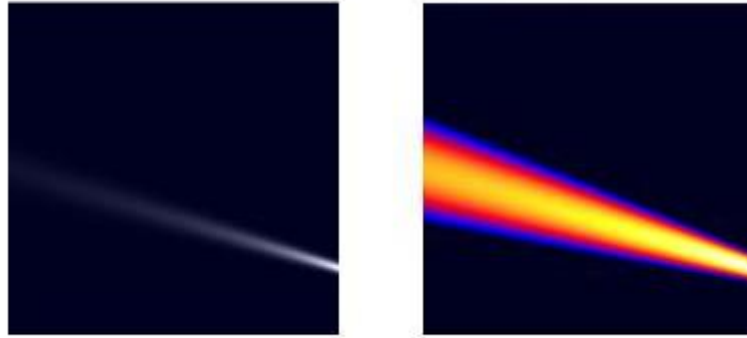


Abbildung 5.5: Ein Gauss-Splat in Parallelen Koordinaten. Links: Intensitätscodiert. Rechts: Farbcodiert.

5.2.2 Implementation des Splattings

”Ebene” Splats

Um Splats mit lokaler Skalierung in Parallelen Koordinaten zu zeichnen, wird zunächst ein passender horizontaler Splat gezeichnet. Die Datenwerte ξ_1 und ξ_2 und ihre Gradienten werden zunächst auf den Intervall $[0, 1]$ umgerechnet. Dabei gilt:

$$\text{Normalize}(\xi, \text{dim}) = \frac{(\xi - \xi_{\text{dim}, \text{min}})}{\text{range}_{\xi_{\text{dim}}}}$$

mit $\text{range}_{\xi_{\text{dim}}} = \xi_{\text{dim}, \text{max}} - \xi_{\text{dim}, \text{min}}$ (Erläuterung siehe auch Gleichung (5.7))

$$\text{Normalize}_{\text{Extent}}(l, \text{dim}) = \frac{(l)}{\text{range}_{\xi_{\text{dim}}}}$$

Soll der Splat ”eben” sein, so zeichnet man ein Trapez und übergibt mit $\text{glColor}(\frac{2}{l_1+l_2} * V_s, \frac{2}{l_1+l_2} * V_s, \frac{2}{l_1+l_2} * V_s, 0)$ den Kehrwert seiner Fläche im Produkt mit V_s als aufzuaddierende Dichte an OpenGL. Das Trapez hat die Eckkoordinaten

$$\begin{pmatrix} 0 \\ \eta_{2, \text{high}, \text{left}} \end{pmatrix}, \begin{pmatrix} 0 \\ \eta_{2, \text{low}, \text{left}} \end{pmatrix}, \begin{pmatrix} 1 \\ \eta_{2, \text{high}, \text{right}} \end{pmatrix}, \begin{pmatrix} 1 \\ \eta_{2, \text{low}, \text{right}} \end{pmatrix}$$

mit:

$$\eta_{2, \text{high}, \text{left}} = \text{range}_{\text{global}} * \left(\text{Normalize}(\xi_1, \text{dim}) + \text{Normalize}_{\text{Extent}}\left(\frac{l_1}{2}\right) \right) + \text{min}_{\text{global}}$$

$$\eta_{2, \text{low}, \text{left}} = \text{range}_{\text{global}} * \left(\text{Normalize}(\xi_1, \text{dim}) - \text{Normalize}_{\text{Extent}}\left(\frac{l_1}{2}\right) \right) + \text{min}_{\text{global}}$$

$$\eta_{2, \text{high}, \text{right}} = \text{range}_{\text{global}} * \left(\text{Normalize}(\xi_2, \text{dim}) + \text{Normalize}_{\text{Extent}}\left(\frac{l_2}{2}\right) \right) + \text{min}_{\text{global}}$$

$$\eta_{2,low,right} = range_{global} * \left(Normalize(\xi_1, dim) - Normalize_{Extent}\left(\frac{l_2}{2}\right) \right) + min_{global}$$

Anschliessend erfolgt eine Scherung mit einer passenden Matrix M_{shear} des Trapezes in η_2 -Richtung, sodass die Mittellinie des Splat genau die Werte ξ_1 und ξ_2 miteinander verbindet.

$$(5.8) \quad M_{shear} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ s & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dabei wird der Scherungsfaktor s so berechnet:

$$s = range_{global} * (Normalize(\xi_2, dim) - Normalize(\xi_1, dim))$$

Es wird also zunächst ξ_1 und ξ_2 auf den Bereich $[0, 1]$ umgerechnet, danach wird ihre Differenz mit dem maximalen Wertebereich $range_{global}$ aller Achsen multipliziert.

Gauss Splats

Im Fall der Gauss-Splats wird ein Rechteck gezeichnet, das als Hilfsgeometrie für die Gaussverteilung dient. Dazu wird das Maximum der beiden Ausdehnungen l_1 und l_2 bestimmt:

$$l_{max} = \max(Normalize_{Extent}\left(\frac{l_1}{2}\right), Normalize_{Extent}\left(\frac{l_2}{2}\right))$$

und es erfolgt eine Umrechnung von $Normalize(\xi_1, dim) \pm l_{max}$ auf den größten Wertebereich $range_{global}$ aller Achsen. Das ergibt die "globalen" Koordinaten des den Splat begrenzenden Rechtecks: die Werte $\eta_{2,high}$ und $\eta_{2,low}$

$$\eta_{2,high} = range_{global} * (Normalize(\xi_1, dim) + l_{max}) + min_{global}$$

$$\eta_{2,low} = range_{global} * (Normalize(\xi_1, dim) - l_{max}) + min_{global}$$

Das begrenzende Rechteck hat die folgenden Eckpunkt-Koordinaten, und ist in Abbildung 5.6 (rechts) zu sehen.

$$(5.9) \quad \begin{pmatrix} 0 \\ \eta_{2,high} \end{pmatrix}, \begin{pmatrix} 0 \\ \eta_{2,low} \end{pmatrix}, \begin{pmatrix} 1 \\ \eta_{2,high} \end{pmatrix}, \begin{pmatrix} 1 \\ \eta_{2,low} \end{pmatrix}$$

Mit diesen Koordinaten wird das OpenGL-Quad gezeichnet (die Hilfsgeometrie für den Splat). Vor dem Zeichnen wird ein passender Shader aktiviert, der den Dichtebeitrag des Splats auf den bisherigen Plot aufaddiert.

Der Shader bekommt die Masse V_s des Splats (siehe Gleichung (3.2)), die Gradienten und die Standardabweichungen

$$\sigma_{y,1} = \frac{Normalize_{Extent}\left(\frac{l_1}{2}, dim\right)}{d}$$

$$\sigma_{y,2} = \frac{Normalize_{Extent}\left(\frac{l_2}{2}, dim\right)}{d}$$

als uniform-Parameter übergeben. d bestimmt dabei, wie "schnell" die Gausskurve auf 0 abfällt, es wurde $d = 8$ gewählt. Der Shader erhält zudem die interpolierten Texturkoordinaten. Dazu werden die Ecken des Quads mittels

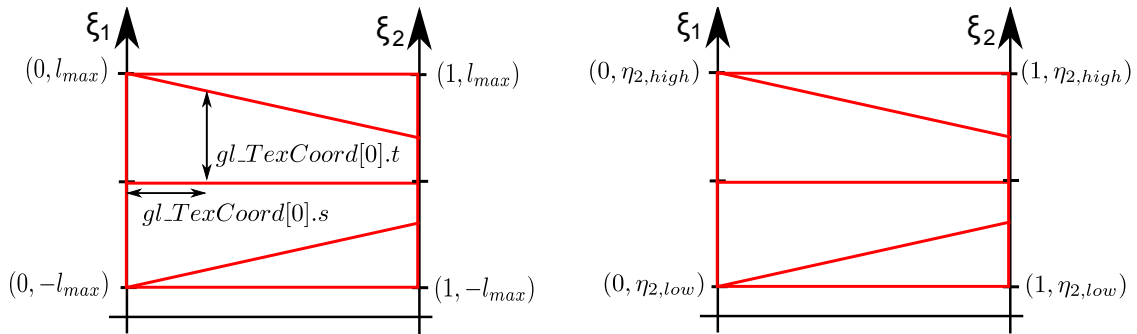


Abbildung 5.6: Ein Splat mit seiner rechteckigen Hilfsgeometrie. Links: Texturkoordinaten. Rechts: Globale Koordinaten.

`glTexCoord2f(x, y);`

mit den Texturkoordinaten $(\begin{smallmatrix} 0 \\ l_{max} \end{smallmatrix})$, $(\begin{smallmatrix} 0 \\ -l_{max} \end{smallmatrix})$, $(\begin{smallmatrix} 1 \\ -l_{max} \end{smallmatrix})$, $(\begin{smallmatrix} 1 \\ l_{max} \end{smallmatrix})$ versehen. Beim Rendern interpoliert der Shader auf der Grundlage der ersten Texturkoordinate $gl_TexCoord[0].s$, die aufgrund der oben angegebenen Texturkoordinaten sich zwischen 0 und 1 bewegt, linear zwischen den beiden Varianzen $\sigma_{y,1}$ und $\sigma_{y,2}$, und berechnet dann den entsprechenden 1D-Gausswert mit dieser Varianz für $x = gl_TexCoord[0].t$. Dabei gilt wieder wegen den Texturkoordinaten dass: $x \in [-l_{max}, l_{max}]$. Danach wird auch der Gauss-Splat mit einer passenden Matrix M_{shear} (siehe Gleichung (5.8)) in y Richtung gescheert, sodass er die Eckpunkte aus (5.9) hat.

Anstatt erst einen horizontalen Splat zu zeichnen, und dann zu scheren, bestehen auch andere Lösungsmöglichkeiten, die im Rahmen der Studienarbeit untersucht wurden. Diese basieren auf aufwändigeren Fragment-Shader Programmen. So könnte man zum Beispiel als Hilfsgeometrie ein großes, achsenparalleles Rechteck zeichnen, in das der Splat gerade hineinpasst (siehe 5.7 links). Man müsste dann aber für jedes Fragment des Rechtecks prüfen, ob es zu dem Splat gehört oder nicht. Wenn ja, müsste man den Abstand zur Mittelachse des Splats bestimmen, und den Wert der Gaussverteilung ausrechnen. Desto größer die Gradienten und der Steigungswinkel der Mittelachse, desto mehr Fragmente müsste man behandeln.

Weniger Aufrufe des Fragment-Shader Programms könnte man durch eine Verkleinerung der Hilfsgeometrie erreichen. Diese ist in Abbildung 5.7 (rechts) zu sehen. Ihre Maße müssen dabei für jeden Splat mit geringem Zusatzaufwand berechnet werden, sie ergeben sich allerdings erst indirekt aus den Datenwerten und ihren Gradienten. Die Geometriekoordinaten für die Eckpunkte sind schon deutlich aufwändiger zu berechnen. Hinzu kommen noch Aufrufe trigonometrischer Funktionen im Fragment-Shader.

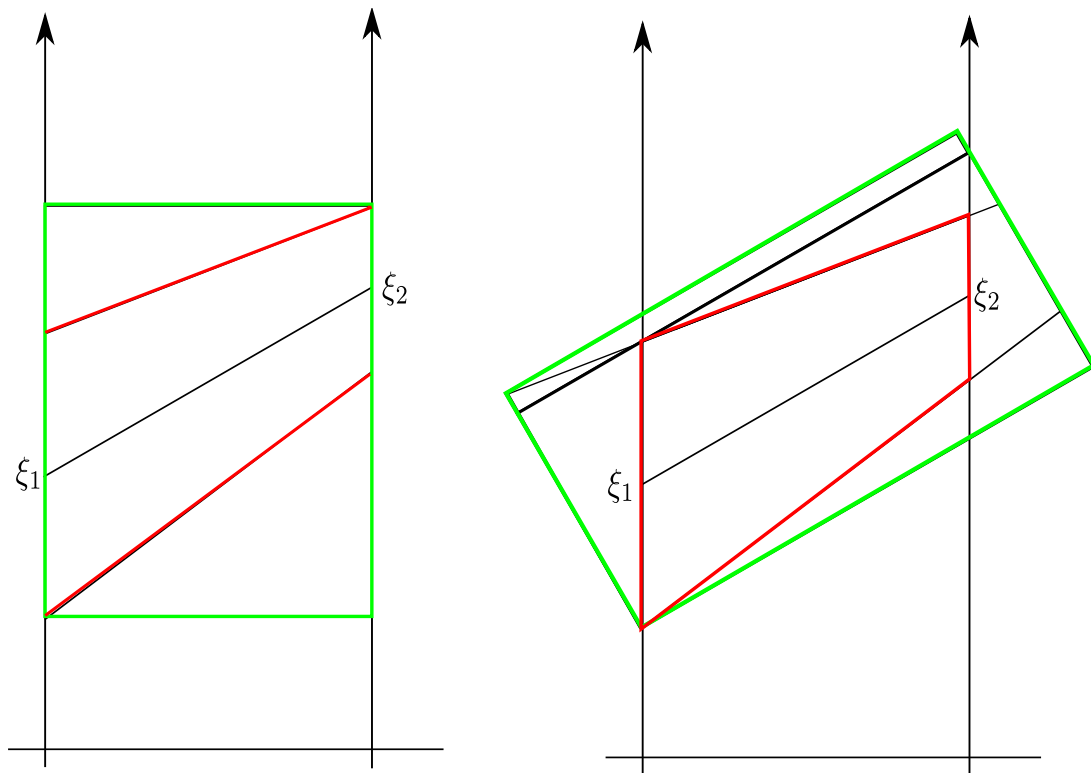


Abbildung 5.7: Beispiele für alternative Hilfsgeometrien eines Splats in Parallelen Koordinaten (im Bild grün markiert). In Rot sind die Umrisse der eigentlichen Splats zu sehen.

6 Colormapping

Wenn die Splatting-Renderer für Scatterplots und Parallele Koordinaten mit dem Rendern einer Runde fertig sind, soll möglichst der gesamte Wertebereich der ermittelten Dichten mit Hilfe einer Farbskala angezeigt werden. Bevor die Abbildung der Dichten auf Farben erfolgen kann, werden sie durch den DensityRenderer auf den Intervall $[0, 1]$ normalisiert. Ein sehr niedriger Dichtewert soll dann Schwarz dargestellt werden, höhere Werte in aufsteigender Reihenfolge mit Blau, Violett, Rot, Orange, Gelb, Weiß.

6.1 Einfaches Colormapping

Zunächst wurde ein Colormapping implementiert, das Dichtewerte ohne vorherige Transformation auf eine Farbe abbildet. Die entsprechende Farbskala wurde mit Hilfe eines Array implementiert, der die RGB Werte der Farben der Standard Temperaturskala enthält. Ein solcher Array kann durch lineare Interpolation zwischen den Farben der Temperatur-Skala gewonnen werden. Der für das Colormapping zuständige Shader kann mittels einem Index vom Typ float, mit der Funktion `texture1D`, interpolierte Farben aus diesem Array abrufen. Der Index darf sich dabei nur im Bereich $[0.0, 1.0]$ bewegen (0.0 entspricht Schwarz, 1.0 entspricht Weiß).

6.2 Probleme beim einfachen Colormapping

Die Renderer liefern jedoch Dichtewerte, die stark voneinander abweichen können. Sie werden vor dem Colormapping zwar an den DensityRenderer weitergegeben und dieser normalisiert die Dichtewerte auf den Intervall $[0, 1]$, trotzdem bleibt das Problem der Abweichungen, da es nicht auf die Differenz zwischen den kleinstem und größtem Wert ankommt ($max_{\sigma} - min_{\sigma} \leq 1$), ob der volle Wertebereich sichtbar wird, sondern auf deren Verhältnis $\frac{max_{\sigma}}{min_{\sigma}}$. Dieses kann trotz vorheriger Normalisierung beliebig groß werden. Z.B. dann wenn $max_{\sigma} = 1$ und $min_{\sigma} =$



Abbildung 6.1: Farbskala zur Farbcodierung der Dichtewerte. (Standard Temperaturskala) Die Farbe Blau hat in einer entsprechenden 1D Textur die Texturkoordinate $s = \frac{1}{6}$.

$1 * 10^{-6}$. Das kann z.B. dann bei Scatterplots der Fall sein, wenn die Skalarfelder an einem Ort mit parallelen Gradienten abgetastet werden. Denn dann geht das Volume Measure gegen 0 und die Dichte des Splats wird unendlich groß. $\lim_{Vol_i \rightarrow 0} \frac{1}{Vol_i} = \infty$

6.3 Verbesserter Ansatz: Logarithmisches Colormapping

Nach der Normalisierung durch den DensityRender berechnet der ColorMapRenderer für jeden Dichtewert $\sigma_{(x,y)}$ folgenden Wert:

$$\sigma_{(x,y),neu} = a * \log(1 + b * \sigma_{(x,y)})$$

Man vervielfacht so zunächst die Dichtewerte auf Werte größer 1 und nutzt dann eine langsam steigende Funktion, nämlich den 10er-Logarithmus, um den Wertebereich $max(\sigma_{(x,y),neu}) - min(\sigma_{(x,y),neu})$ stark zu verkleinern. Anschließend korrigiert man die logarithmierten Werte so, dass alle davon wieder im Intervall $[0, 1]$ liegen. Die Parameter a und b kann der Anwender selbst wählen oder einen Algorithmus verwenden, der versucht, Werte für a und b zu finden, die den gesamten Wertebereich der Dichten sichtbar machen. Damit bei anwenderdefinierten Werten a, b keine Werte kleiner 0 herauskommen wird im Logarithmus noch 1 addiert. $\sigma_{(x,y),neu}$ wird schließlich zur Interpolation einer Farbe aus der 1D Temperaturskala verwendet.

6.4 Automatische Wahl der Faktoren a und b

Damit alle Dichtewerte gut sichtbar sind muss folgendes gelten:

$$(6.1) \quad a * \log(1 + Min(\sigma) * b) = \frac{1}{6}$$

$$(6.2) \quad a * \log(1 + Max(\sigma) * b) = 1$$

Mit $Max(\sigma) = 1$ gilt dann:

$$a * \log(1 + Min(\sigma) * b) = \frac{1}{6}$$

$$a * \log(1 + b) = 1$$

Würde man a und b so wählen, dass diese Gleichungen erfüllt sind, dann würde der kleinste gerade noch über 0 liegende Wert gerade auf die Farbe Blau abgebildet werden. Das Maximum wäre Weiß. Jedoch möchte man in der Regel erreichen, dass die kleinsten Dichtewerte schwarz dargestellt werden. Das kann man ausdrücken, indem man den Divisor 6 in (6.1) durch eine Variable γ ersetzt. :

$$(6.3) \quad a * \log(1 + Min(\sigma) * b) = \frac{1}{\gamma}$$

Wählt man γ hoch (z.B. $\gamma = 512$), so erreicht man eine Abbildung sehr kleiner Werte auf Schwarz. Die Gleichung (6.3) lässt sich nach b auflösen.

$$\gamma * a * \log(1 + Min(\sigma) * b) = 1$$

$$\log((1 + Min(\sigma) * b)^{\gamma * a}) = 1$$

$$(1 + \text{Min}(\sigma) * b)^{\gamma * a} = 10$$

$$1 + \text{Min}(\sigma) * b = \sqrt[\gamma * a]{10}$$

$$b = \frac{\sqrt[\gamma * a]{10} - 1}{\text{Min}(\sigma)}$$

Die Formel für b kann in (6.2) eingesetzt werden, dadurch erhält man ein Polynom $f(a)$, dessen Nullstellen zu bestimmen sind:

$$a * \log\left(1 + \frac{\sqrt[\gamma * a]{10} - 1}{\text{Min}(\sigma)}\right) = 1$$

$$f(a) = a * \log\left(1 + \frac{\sqrt[\gamma * a]{10} - 1}{\text{Min}(\sigma)}\right) - 1 = 0$$

Die Berechnung der Nullstellen kann mittels Newton-Verfahren [BSMM01] erfolgen. Dabei wird eine Zahlenfolge berechnet, die gegen eine Nullstelle des Polynoms konvergiert. Die Folge ist definiert durch:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

bzw.

$$a_{n+1} = a_n - \frac{f(a_n)}{f'(a_n)}$$

Für das Verfahren wird also die Funktion $f(a)$ und ihre Ableitung benötigt. Die Ableitung ergibt sich zu:

$$f'(a) = \log\left(1 + \frac{(\sqrt[\gamma * a]{10}) - 1}{\text{Min}(\sigma)}\right) + a * \frac{1}{\log(10) * \left(1 + \frac{(\sqrt[\gamma * a]{10}) - 1}{\text{Min}(\sigma)}\right)} * \frac{1}{\gamma * a} * 10^{\frac{1}{\gamma * a} - 1} * \text{Min}(\sigma)$$

Als Startwert a_0 wurde in der Implementation 10 gewählt, was zu guten Ergebnissen geführt hat. Die Berechnung der Folge endet sobald $|a_{n+1} - a_n| < \epsilon$ gilt, also sobald die Änderung kleiner einem Schwellwert ϵ (z.B. $\epsilon = 0,00001$) wird.

7 Qualitativer und Quantitativer Vergleich mit herkömmlichen Methoden

Die in Kapitel 5 vorgestellten Methoden sollten mit den herkömmlichen Methoden verglichen werden:

- Wie stark wird das Rendern beschleunigt?
- Welche Qualität haben die Plots?

Um diese Fragen zu klären, wurden insgesamt 10 Testläufe durchgeführt. 5 je für Scatterplots mit ebenen Splats (SPRectangles), und Parallele Koordinaten mit ebenen Splats (PCTrapezoids). Dabei wurde im n -ten Testlauf $n * 5000$ mal gesampelt. Da sich bei Verwendung von Gauss-Splats keine eindeutige Verbesserung der Ergebnisse ergab, wurde hier auf eine entsprechende Auswertung verzichtet.

7.1 Beschleunigung

Der herkömmliche Renderer für kontinuierliche Scatterplots benötigt 3906.5 ms. Diagramm 7.2 zeigt, welche Laufzeiten beim Splatting gemessen wurden:

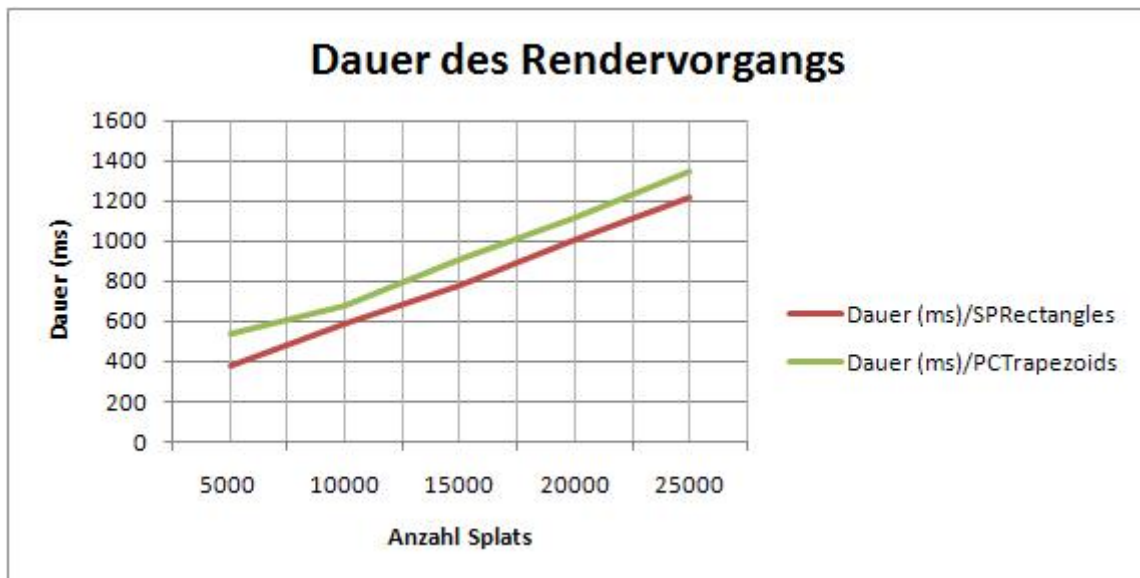


Abbildung 7.1: Das Diagramm zeigt, wie sich die benötigte Dauer für das Splatting erhöht, wenn man die Anzahl der Splats vergrößert.

7.2 Qualität

Als Qualitätsmaß wurde die L2 Distanz der Dichtewerte der Plots zu den herkömmlich gerenderten Plots gewählt. Die Dichtewerte erhält man, indem man die einzelnen Pixel eines Plots ausliest (Intensitätswerte ohne Farbcodierung). Bevor ein Dichtewert in die Formel für die L2 Distanz eingeht, wird der Mittelwert aller Dichtewerte abgezogen. Dadurch haben die Plots auch dann eine niedrige L2 Distanz, wenn ihre Dichtewerte zwar unterschiedlich, aber Vielfache voneinander sind.

$$d = \sqrt{\sum_{i=1}^{768 \cdot 1024} (\sigma_{i,Plot1} - \sigma_{Mittelwert,Plot1} - (\sigma_{i,Plot2} - \sigma_{Mittelwert,Plot2}))^2}$$

Die Qualität der Plots verbesserte sich bei den Messungen zunächst mit der Erhöhung der Anzahl der Splats, jedoch zeigte sich bei weiterem Erhöhen der Splatanzahl wieder eine Verschlechterung der L2 Distanzen. Auch beim progressiven Rendern wurde kein eindeutiger Trend hin zu niedrigeren L2 Distanzen festgestellt. Da die Samples jedoch beim Splatting trilinear interpoliert werden, und bei den herkömmlichen Ansätzen [BWo8] und [HWo9] ein lineares Interpolationsschema verwendet wird, wird man bei einem Vergleich immer entsprechend hohe Distanzwerte (größer 0) bekommen.

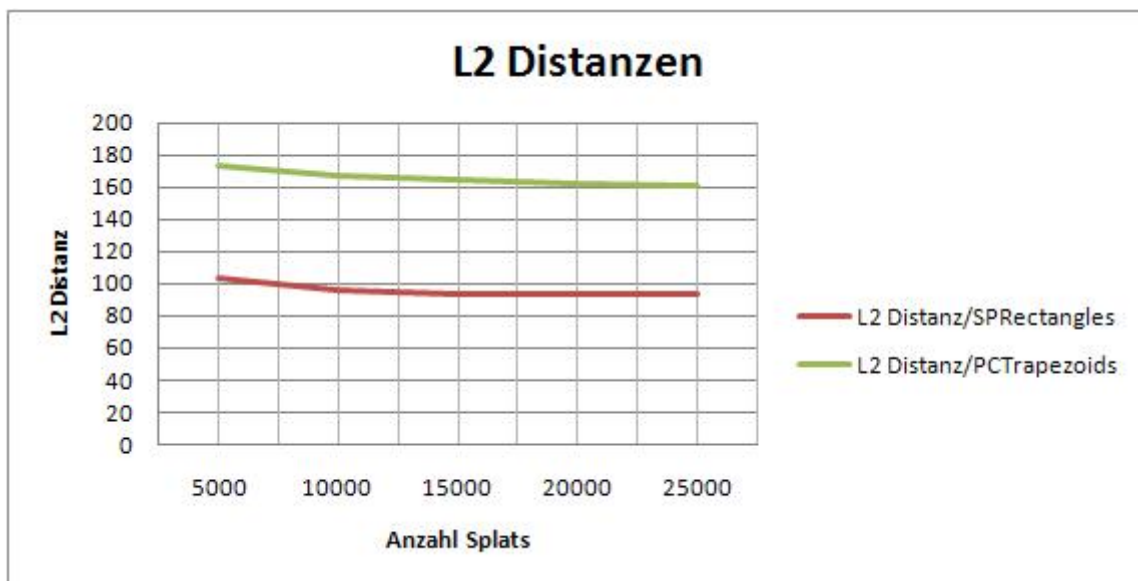


Abbildung 7.2: Das Diagramm zeigt, wie sich die L2 Distanz der Ergebnisse des Splatting zum Ergebnis der Ansätze [BWo8] bzw. [HWo9] entwickelt, wenn man die Splat-Anzahl erhöht.

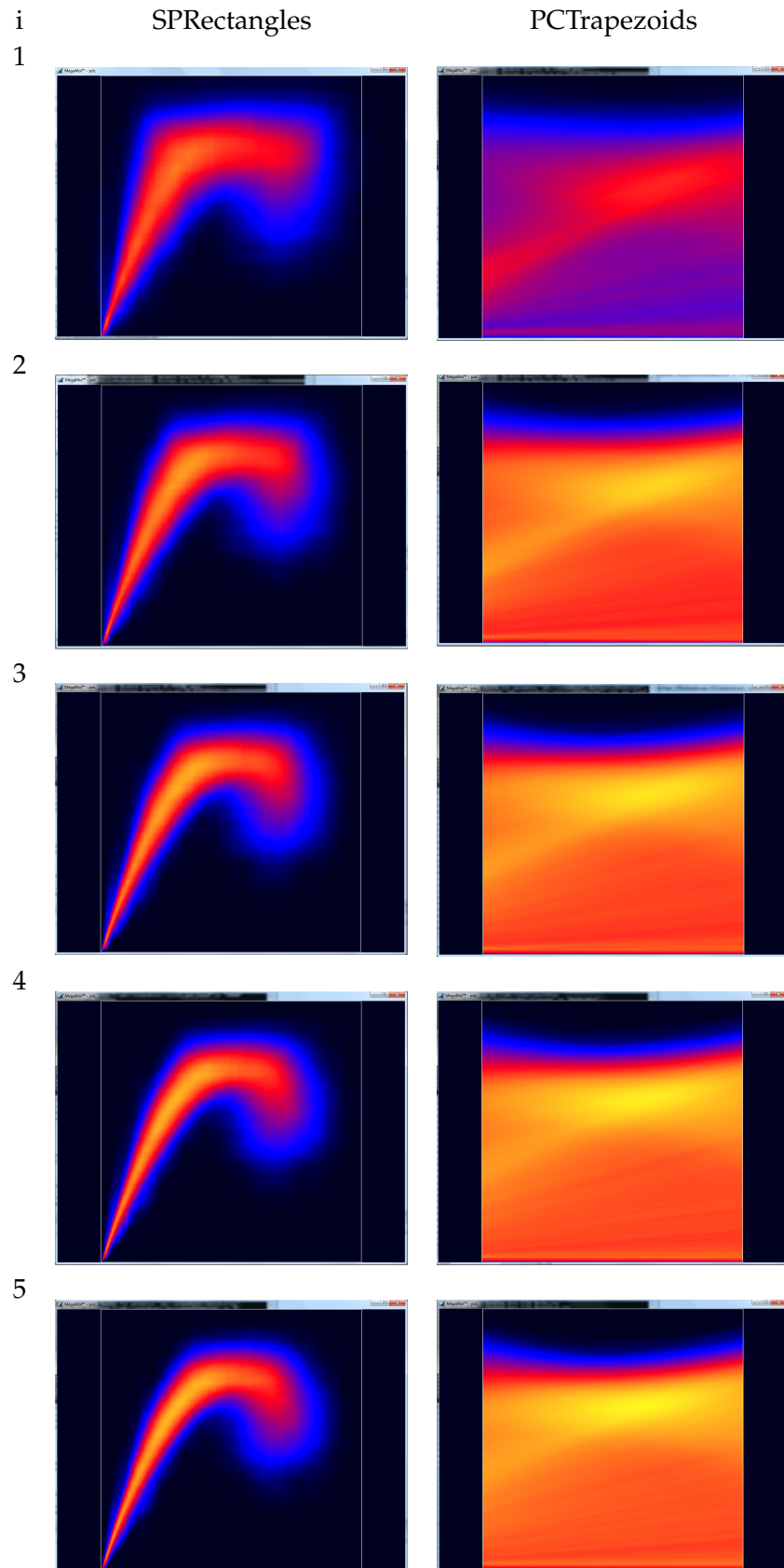


Tabelle 7.1: Ergebnisse des Splatting für Scatterplots und Parallele Koordinaten. Das Color-mapping wurde ohne konstante Parameter a, b automatisch durchgeführt.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Splatting ist ein geeigneter Ansatz, um das Rendern kontinuierlicher Scatterplots und von kontinuierlichen Parallelen Koordinaten zu beschleunigen. Im Rahmen dieser Studienarbeit wurden für Scatterplots einfache Rechteck-Splats und für Parallele Koordinaten Trapez-Splats, sowie für beide Visualisierungstechniken entsprechende Gauss-Splats, untersucht. Die normalisierten Dichten der Plots haben eine deutliche L2 Distanz zu den Ergebnissen herkömmlicher Verfahren, und das Verfahren für progressives Rendern liefert keinen eindeutigen Trend hin zu niedrigeren Distanzen. Aufgrund des (im Gegensatz zu herkömmlichen Methoden) trilinearen Interpolationsschemas, spricht das aber nicht gegen die Qualität der Plots. Trotz den deutlichen L2 Distanzen, sind die Plots den Dichteverteilungen der Skalarkomponenten von τ , wie sie sich bei herkömmlichen Methoden ergeben, sehr ähnlich. Um die Dichten gut ablesbar zu machen, wurde ein Renderer entwickelt, der normalisierte Dichten so auf eine Farbskala abbildet, dass alle Dichtewerte ablesbar und gut unterscheidbar sind.

8.2 Ausblick

Im Rahmen weiterführender Arbeiten könnte untersucht werden inwiefern Low-Discrepancy Sequences mit niedrigerer Diskrepanz, wie z.B. Niederreiter-Sequenzen [Nie87], dazu in der Lage sind, die Plots noch zu verbessern. Eine weitere Möglichkeit zur Verbesserung ist die Verwendung einer besseren Randbehandlung, die es erlaubt, auch sehr nah am Rand des Grid liegende Samples in die Plots abzubilden. Die Schlüsselfrage dabei wäre dann, wie man die Gradienten solcher Samples ermittelt. Das Rendering könnte durch Reduktion der Anzahl der OpenGL-Aufrufe optimiert werden. Dazu müssten in jeder Runde zunächst auf der Grundlage der Skalarwerte und Gradienten für alle Splats der Runde die Eckpunkte und Dichten vorberechnet und in Arrays gespeichert werden. Diese Arrays könnte man mit dem Befehl `glVertexArray()` OpenGL zugänglich machen. Dann könnten mit einem entsprechenden Shader $i * N$ Splats auf einmal mit dem `glDrawElements()`-Befehl gezeichnet werden. In Bezug auf den `ColorMapRenderer` wäre noch zu klären, wann genau die automatische Wahl der Parameter a, b funktioniert und wann nicht.

Literaturverzeichnis

- [BC09] BAILEY, Mike ; CUNNINGHAM, Steve: *Graphics Shaders: Theory and Practice*. Natick, MA, USA : A. K. Peters, Ltd., 2009. – ISBN 1568813341, 9781568813349 (Zitiert auf Seite 8)
- [Bey99] BEYER, O.: *Wahrscheinlichkeitsrechnung und mathematische Statistik*. Teubner, 1999 (Zitiert auf Seite 27)
- [BF88] BRATLEY, Paul ; FOX, Bennett L.: Algorithm 659: Implementing Sobol’s quasirandom sequence generator. In: *ACM Trans. Math. Softw.* 14 (1988), Nr. 1, 88–100. <http://doi.acm.org/10.1145/42288.214372>. – ISSN 0098–3500 (Zitiert auf den Seiten 20, 21 und 23)
- [BSMM01] BRONSTEIN, I. N. ; SEMENDJAJEW, K. A. ; MUSIOL, G. ; MÜHLIG, H.: *Taschenbuch der Mathematik*. 5. Frankfurt/Main : Verlag Harri Deutsch, 2001 (Zitiert auf Seite 37)
- [BW08] BACHTHALER, Sven ; WEISKOPF, Daniel: Continuous Scatterplots. In: *IEEE Transactions on Visualization and Computer Graphics* 14 (2008), Nr. 6, 1428–1435. <http://dx.doi.org/10.1109/TVCG.2008.119>. – ISSN 1077–2626 (Zitiert auf den Seiten 4, 6, 7, 10, 11, 12 und 39)
- [FBO] *EXT-framebuffer-object Dokumentation*. http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt (Zitiert auf Seite 8)
- [GLS] *The OpenGL Shading Language*. <http://www.opengl.org/registry/doc/GLSLangSpec.1.50.09.pdf> (Zitiert auf Seite 8)
- [Hal60] HALTON, J H.: On the efficiency of certain quasi-random sequences of points in evaluating multidimensional integrals. In: *Numer. Math* (1960), Nr. 2, S. 84–90 (Zitiert auf den Seiten 2 und 22)
- [HW09] HEINRICH, Julian ; WEISKOPF, Daniel: Continuous Parallel Coordinates. In: *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), Nr. 6, 1531–1538. <http://dx.doi.org/10.1109/TVCG.2009.131>. – ISSN 1077–2626 (Zitiert auf den Seiten 4, 6, 7, 11, 12, 13 und 39)
- [ID87] INSELBERG, A. ; DIMSDALE, B.: Parallel coordinates for visualizing multi-dimensional geometry. In: *CG International ’87 on Computer graphics 1987*. New York, NY, USA : Springer-Verlag New York, Inc., 1987. – ISBN 4-431-70022-6, S. 25–44 (Zitiert auf Seite 10)
- [Knu97] KNUTH, Donald E.: *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1997 <http://portal.acm.org/citation.cfm?id=270146>. – ISBN 0201896842 (Zitiert auf Seite 21)

-
- [Nie87] NIEDERREITER, Harald: Point sets and sequences with small discrepancy. In: *Monatshefte für Mathematik* Volume 104 (1987), S. 273–337 (Zitiert auf den Seiten 23 und 41)
- [PFT⁺86] PRESS, W. H. ; FLANNERY, B. P. ; TEUKOLSKY, S. A. ; ; VETTERLING, W. T.: *Numerical Recipes – The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1986 (Zitiert auf Seite 21)
- [Sob67] SOBOL, I. M.: The distribution of points in a cube and the approximate evaluation of integrals. In: *UDSSR Comput. Math. Math. Phys.* 7 4 (1967), S. 86–112 (Zitiert auf Seite 21)
- [ST90] SHIRLEY, Peter ; TUCHMAN, Allan: A Polygonal Approximation to Direct Scalar Volume Rendering. In: *Computer Graphics*, 1990, S. 63–70 (Zitiert auf Seite 12)
- [Wes90] WESTOVER, Lee: Footprint evaluation for volume rendering. In: *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1990. – ISBN 0-89791-344-2, 367–376 (Zitiert auf den Seiten 13 und 14)

Alle URLs wurden zuletzt am 6.07.2010 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Bernhard Maier)