Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Graphics, Usability, and Visualization Lab

School of Computing Science
Simon Fraser University
Burnaby BC
Canada V5A 1S6

Diplomarbeit Nr. 2510

GPU-basierte Vektorfeldvisualisierung mittels 3D LIC

Martin Falk

Studiengang:	Informatik
Prüfer:	Prof. Dr. Thomas Ertl
Betreuer:	Prof. Dr. Daniel Weiskopf
begonnen am:	15. Juni 2006
beendet am:	15. Dezember 2006
CR-Klassifikation:	I.3.3, I.3.7

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Motivation 1

1.2 Aufgabenstellung 2

2 Volumenvisualisierung **3**

2.1 Isoflächen 3

2.2 Schnittebene 4

2.3 DVR – Direct Volume Rendering 5

2.3.1 Slicing 5

2.3.2 Ray-Casting 7

2.3.3 Beschleunigung der GPU-Verfahren 8

3 LIC – Line Integral Convolution **11**

3.1 2D LIC 12

3.2 3D LIC 14

3.3 GPU-basierter 3D LIC 14

3.4 Beleuchtung von Linien 15

3.4.1 Stromlinien nach Zöckler 16

3.4.2 Stromlinien nach Mallo 17

3.4.3 Beleuchtung des LIC anhand von Gradienten 19

3.4.4 Wahl der Transferfunktion 20

3.5 Features 20

3.5.1 Erkennung von Wirbeln 22

4 Modellierung des Rauschens **23**

4.1 Weißes Rauschen 24

4.2 Dünnes Rauschen 24

4.3 Filterung 26

4.4 Konstante Raumfrequenz 27

5 GPU-basierter 3D LIC **31**
5.1 Berechnung des 3D LIC . . . 32
5.1.1 LIC-Berechnung im Fragmentprogramm . . . 32
5.1.2 Ray-Casting . . . 33
5.1.3 Ray-Casting mit Depth-Peeling . . . 34
5.1.4 Slicing . . . 35
5.2 Beleuchtung . . . 36
5.3 Features . . . 37
5.4 Visualisierung zeitabhängiger Datensätze . . . 37

6 Leistungsbetrachtung **41**
6.1 Einfluss der Viewportgröße . . . 42
6.2 Optimierungen . . . 42
6.3 Abhängigkeit von der Kameraposition . . . 44
6.4 Beleuchtungsmodelle . . . 47
6.5 Samplingdistanz . . . 48
6.6 Anzahl der LIC-Schritte pro Abtastpunkt . . . 49
6.7 Anzahl der Abtastungen pro Depth-Peel-Schicht . . . 49

7 Qualitative Ergebnisse **51**
7.1 Visuelle Auswertung . . . 51
7.2 Beispiele . . . 57

8 Zusammenfassung und Ausblick **63**

Literaturverzeichnis **65**

Abbildungsverzeichnis

2.1 Object-Aligned Slicing 5
2.2 View-Aligned Slicing 6
2.3 Ray-Casting 7
3.1 Vergleich LIC und Spot Noise 13
3.2 VolumeLIC ohne und mit Halo 14
3.3 Beleuchtungstextur von Zöckler 16
3.4 Diffuse Beleuchtung eines Zylinders 17
3.5 Beleuchtungstexturen von Mallo 18
3.6 Velocity-Masking 21
3.7 Clipping des Vektorfelds 21
4.1 LIC bei verschiedenen Raumfrequenzen der Rauschtextur 23
4.2 Weißes Rauschen 24
4.3 Gleichverteilung von dünnem Rauschen 25
4.4 Filterung mit einem Tiefpassfilter 28
4.5 Konstante Raumfrequenz des Rauschens im Bildraum 29
5.1 Architektur zur Berechnung des 3D LIC auf der GPU 31
5.2 Schema des Fragmentprogramms für 3D LIC mittels Ray-Casting 33
5.3 Ablauf des Ray-Castings mit Depth-Peeling 34
5.4 Ablauf des Slicings 35
5.5 Empty-Space-Leaping durch Ray-Casting mit Depth-Peeling 38
6.1 Testszenario: Viewportgröße 42
6.2 Diagramm: Einfluss Viewportgröße 43
6.3 Testszenario: Framebufferobjekt und Early-Z-Test 43
6.4 Diagramm: FBO und Early-Z-Test, ohne Optimierungen 45
6.5 Diagramm: FBO und Early-Z-Test, vorzeitiger Strahlabbruch 45
6.6 Diagramm: FBO und Early-Z-Test, mit Optimierungen 45
6.7 Diagramm: FBO und Early-Z-Test, Vergleich 46
6.8 Testszenario: Abhängigkeit von der Kameraposition 46
6.9 Diagramm: Abhängigkeit von der Kameraposition 46
6.10 Testszenario und Diagramm: Beleuchtungsmodelle 47
6.11 Testszenario und Diagramm: Abstand der Samplingpositionen 48

6.12 Testszenario und Diagramm: LIC-Schritte 49
6.13 Testszenario und Diagramm: Abtastungen pro Depth-Peel-Schicht 50
7.1 Vergleich zwischen hoch und niedrig aufgelöster Visualisierung 52
7.2 Maximales Instruktionslimit im Fragmentprogramm 53
7.3 Skalierung der Raumfrequenz des Rauschens 53
7.4 Einfluss des Rauschens 54
7.5 Normalen der Beleuchtungsmodelle 55
7.6 Vergleich der Beleuchtungsmodelle 55
7.7 Hervorhebung mittels Transferfunktion 56
7.8 Verbesserung der Tiefenwahrnehmung durch die Transferfunktion 56
7.9 Benard-Strömung von Daniel Weiskopf 57
7.10 Datensatz des IEEE Visualization 2004 Contest 58
7.11 Tornado von Roger Crawfis 59
7.12 Large-Eddy-Simulation von Octavian Frederich 60
7.13 Datensatz 6603-small 61

1 Einleitung

1.1 Motivation

Die Strömungsvisualisierung ist ein großes Teilgebiet der Visualisierung. In diesem Gebiet müssen oft Vektorfelder aus Messungen oder Simulationen dargestellt werden. Handelt es sich dabei um zweidimensionale Daten, wird oft die Line Integral Convolution, kurz LIC, dazu verwendet. Die Line Integral Convolution gehört zu den globalen Techniken der Vektorfeldvisualisierung.

Es existieren bereits einige Ansätze, die die Berechnung des LIC auch auf dreidimensionalen Daten durchführen können. Durch den Wechsel der Dimension ergeben sich jedoch Probleme. Zum einen steigt die Berechnungskomplexität an, zum anderen erhöht sich die visuelle Komplexität und auch die Verdeckung nimmt zu. Da der LIC lokal berechnet werden kann, würde sich für die Berechnung die GPU aufgrund ihrer SIMD-Architektur anbieten.

Zur Verringerung der visuellen Komplexität können verschiedene Ansätze verfolgt werden. So können bei Vektorfeldern Bereiche mit einer bestimmten Geschwindigkeit ausgeblendet werden. Ebenso kann ein Beleuchtungsmodell für die Schattierung des LIC eingesetzt werden.

In dieser Arbeit wird die erste GPU-basierte Implementierung eines volumetrischen LIC vorgestellt. Die Berechnung erfolgt dabei vollständig auf der Grafikkarte. Es werden dazu Ray-Casting als bildraumbasiertes Verfahren und Slicing als ein Objektraumverfahren eingesetzt. Für beide Techniken werden verschiedene Beschleunigungsverfahren vorgestellt um die Berechnungskomplexität weiter zu senken.

Die visuelle Komplexität soll durch die Beleuchtung des LIC verringert werden. Da es sich bei Ergebnis des LIC jedoch um Linien handelt müssen spezielle Verfahren eingesetzt werden. Darüber hinaus soll ein neuer Ansatz zur Beleuchtung von Linien, die sich aus der Berechnung des LIC ergeben, vorgestellt werden.

1.2 Aufgabenstellung

Die Visualisierung von 3D Vektorfeldern spielt eine wichtige Rolle für die Darstellung von Simulationsergebnissen aus CFD-Berechnungen (Computational Fluid Dynamics). Ein für 2D Strömungen beliebter Ansatz ist die LIC-Technik (LIC = Line Integral Convolution). Im 3D Fall ist LIC-Technik jedoch mit sehr hohen Rechenkosten verbunden. Deshalb sollte in dieser Arbeit eine effiziente Realisierung von 3D LIC mittels GPUs entwickelt werden. Die Darstellung der LIC-Berechnungen sollte durch GPU-Ray-Casting erfolgen. Ziele der Arbeit waren im Einzelnen:

- Implementierung der 3D LIC-Berechnung in einem Fragment-Programm
- Darstellung der LIC-Berechnung durch GPU-Ray-Casting
- Beschleunigung des Ray-Casting durch Standardmethoden (z.B. Early-Ray-Termination)
- Anti-Aliasing und Multi-Resolution-Methoden für die Modellierung der injizierten Rauschtexturen
- Anpassung der Granularität und Opazität der Volumendarstellung, um eine gute Wahrnehmung der Tiefenstrukturen zu erreichen (z.B. durch Anpassung der Dichte der Rauschtexturen und durch Wahl angemessener Transferfunktionen)

Die Implementierung sollte in C/C++, OpenGL und ARB-Shadern erfolgen. Für das GPU-Ray-Casting sollte auf der existierenden Implementierung von Stegmaier et al. [54] aufgebaut werden.

2 Volumenvisualisierung

Die Volumenvisualisierung wird vorrangig im Bereich der medizinischen Visualisierung angewandt, wenn es darum geht, Daten aus einem Computertomographen (CT) oder einem Magnetresonanztomographen (MRT) darzustellen [55]. Sie wird aber auch bei der Visualisierung von Daten aus den Bereichen der CFD (Computational Fluid Dynamics), der Seismik und der Geologie verwendet. Die Arbeit von Kajiya und von Herzen [22] bildet dabei die Grundlage für die Darstellung von Volumendaten.

Ist nur eine zweidimensionale Visualisierung der Daten erforderlich, können Ebenen eingesetzt werden, die das Volumen schneiden. Für die dreimdimensionale Darstellung muss zwischen den Techniken der direkten und der indirekten Volumenvisualisierung unterschieden werden. Bei der indirekten Volumendarstellung wird in einem Vorbereitungsschritt eine Zwischenrepräsentation erstellt, die anschließend mit klassischen Techniken dargestellt werden kann. Dadurch ist es möglich, Isoflächen aus dem Volumen zu extrahieren und mittels Geometrie zu rendern. Dagegen wird bei der direkten Volumenvisualisierung, kurz DVR (Direct Volume Rendering), das Volumen ohne Zwischenrepräsentation gerendert. Zu den Techniken des DVR zählen das Ray-Casting, das Slicing, das Splatting [65] und die Shear-Warp-Faktorisierung [31]. Beim Ray-Casting werden Strahlen von der Kamera aus durch das Volumen gesendet. Durch regelmäßiges Abtasten entlang des Strahls kann so das Volumenintegral gelöst werden. Ray-Casting ist ein Bildraumverfahren, bei dem für jedes Pixel ein Strahl berechnet werden muss. Dagegen wird beim Slicing das Volumen von mehreren Ebenen geschnitten, die texturiert werden und anschließend das Ergebnis mittels Compositing berechnet. Splatting ist, ebenso wie das Slicing und die Shear-Warp-Faktorisierung, ein Verfahren, das im Objektraum berechnet wird. Beim Splatting werden alle Voxel auf die Bildebene projiziert. Dabei wird jedes Voxel als Scheibe, deren Dichte zum Rand abfällt, gerendert. Die Shear-Warp-Faktorisierung verwendet im ersten Schritt eine Scherung des Volumens (Shear), um eine konstante Samplingdistanz beizubehalten. Im zweiten Schritt wird das Ergebnis auf die Bildebene projiziert (Warp). Sowohl das Splatting als auch die Shear-Warp-Faktorisierung sind bei der Darstellung von Volumen sehr schnell. Dies hat zur Folge, dass die Ergebnisse qualitativ nicht so hochwertig wie beim Ray-Casting oder Slicing sind.

Einige der im folgenden näher vorgestellten Techniken werden in leicht abgewandelter Form auch für die Visualisierung von Vektorfeldern verwendet. In dieser Arbeit werden das Ray-Casting und das Slicing, einschließlich deren Beschleunigungen zur Berechnung des LIC eingesetzt.

2.1 Isoflächen

Die Visualisierung von Volumen anhand von Isoflächen zählt zur indirekten Volumendarstellung, da zuerst die Isoflächen aus dem Volumen extrahiert und diese anschließend gerendert werden. Eine zum Isowert c zugehörige Isofläche wird aus der Menge von Punkten, die gleich c sind, gebildet. Die Bestimmung und Berechnung erfolgt auf verschiedene Arten.

So wird beim Verfolgen von Konturen (Contour Tracing) das Volumen durch mehrere parallele Ebenen unterteilt. Innerhalb jeder Ebene wird nun versucht, eine geschlossene Konturlinie mit dem Isowert c zu finden. Danach werden die Konturlinien von zwei benachbarten Ebenen mit einem Streifen aus Dreiecken verbunden. In einem Zwischenschritt können die Konturlinien klassifiziert werden. Dies ermöglicht die Zuordnung der Linien über die Ebenen hinweg anhand der Klassifikation. Die so entstandenen Dreiecke bilden die Isofläche zu c. Von Nachteil ist, dass eine hohe Varianz zwischen den Ebenen das Finden von Konturen erheblich erschweren kann und innerhalb einer Ebenen keine Geometrie erzeugt wird.

Zur Bestimmung von Isoflächen kann auch der Cuberille-Ansatz von Herman und Liu [18] verwendet werden. Dabei wird der Datensatz in ein uniformes Gitter aufgeteilt. Die Werte der Gitterpunkte werden mit dem Isowert verglichen und die Punkte entsprechend positiv oder negativ markiert. Danach werden die Flächen der Gitterzellen gerendert, deren Normale aus der Zelle und in Richtung Kamera zeigt. Da innerhalb der Zellen nicht interpoliert wird, ist die so berechnete Isofläche sehr blockartig. Um diese Artefakte zu verringern, kann eine adaptive Unterteilung der Würfel eingesetzt werden. Dies führt zu dem Ansatz der *Dividing Cubes* von Cline et al. [6].

Der Marching-Cube-Algorithmus von Lorensen und Cline [37] ist wohl das bekannteste Verfahren zur Extraktion von Isoflächen. Das Volumen wird dabei ebenfalls in ein uniformes Gitter unterteilt. Da in diesem Algorithmus jeweils nur eine Zelle des Gitters betrachtet und anschließend zur nächsten übergegangen wird und die Zellen einem Würfel gleichen, bekam er den Namen Marching-Cube (marschierender Würfel). Die Eckpunkte des Würfels werden wie beim Cuberille-Ansatz [18] entsprechend dem Isowert markiert. Diese Markierung erfolgt in einer Bitmaske, mit welcher anschließend in einer Lookuptabelle nachgeschlagen werden kann. Diese Lookuptabelle enthält alle möglichen Kombination von Isoflächen, die sich durch trilineare Interpolation innerhalb eines Würfels ergeben können. Zusammen mit den Schnittpunkten, die entlang der Würfelkanten durch lineare Interpolation berechnet werden, ergibt sich so eine effiziente Triangulierung der Isofläche.

Durch die heutige Funktionalität der Grafikhardware ergeben sich aber auch neue Möglichkeiten zur Berechnung der Isoflächen. So lassen sich beispielsweise Isoflächen direkt mittels Ray-Casting aus dem Volumendatensatz extrahieren. Stegmaier et al. [54] führen dazu die Suche nach dem Isowert während des Ray-Castings im Fragmentprogramm durch. Jedoch können diese Isoflächen nur direkt visualisiert werden, da es sich um Fragmente, nicht aber um Geometrie im eigentlichen Sinne handelt.

2.2 Schnittebene

Ist nur eine zweidimensionale Darstellung des Volumens erwünscht, kann dies mit Hilfe von Ebenen realisiert werden. Diese Technik ist auch unter dem Namen *Multiplanar Reformatting* (MPR) bekannt [17, 46, 29, 40, 55]. Das Volumen wird dabei mit der Ebene geschnitten. Erfolgt das Schneiden orthogonal zu den Hauptachsen x, y oder z, kann die Ebene durch Resampling mit bilinearer Interpolation aufgefüllt werden. Bei beliebigen Ebenen muss dagegen trilineare Filterung verwendet werden. Das Resampling kann durch die CPU oder auch durch die Grafikhardware erfolgen.

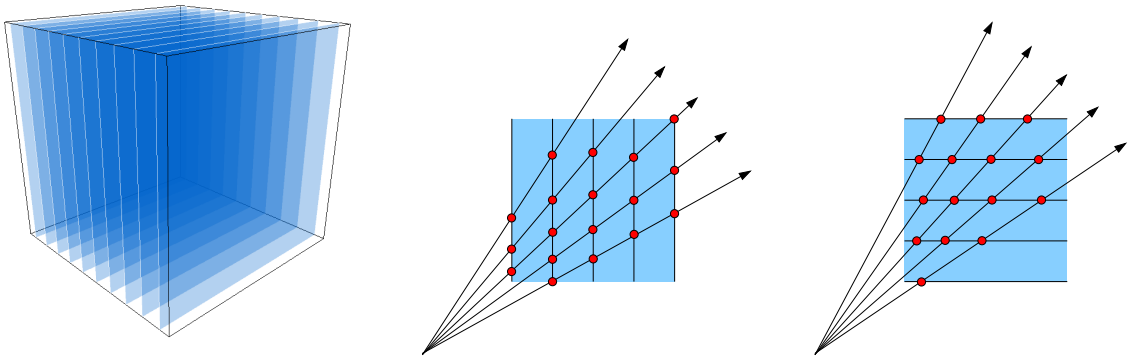


Abbildung 2.1: Slicing mit an den Hauptachsen des Volumens ausgerichteten Ebenen (Object-Aligned). Links die Anordnung der Ebenen, in der Mitte und rechts die Positionen der Abtastpunkte bei verschiedenen Kamerapositionen.

2.3 DVR – Direct Volume Rendering

Bei der direkten Volumenvisualisierung wird nicht wie bei der indirekten zuerst eine Zwischenrepräsentation erzeugt, sondern das Volumen als Ganzes direkt visualisiert. Dabei können die Techniken in Objektraum- und Bildraumverfahren eingeteilt werden. Slicing wird den Objektraumverfahren zugeordnet, da der Datensatz dabei im Objektraum traversiert wird. Bildraumverfahren, zu denen auch das Ray-Casting gehört, arbeiten auf Pixelebene.

2.3.1 Slicing

Slicing gehört zu den texturbasierten Visualisierungstechniken [47, 10, 67]. Da die Grafikhardware bislang keine volumetrischen Primitive unterstützt, wird eine Proxygeometrie erzeugt. Diese Proxygeometrie besteht aus einer Menge von Primitiven, die das Volumen repräsentieren. Meist wird dazu das Volumen mit mehreren Ebenen geschnitten. Diese Ebenen werden mit Daten des Volumendatensatzes texturiert und anschließend gerendert.

Der Ablauf dieser Technik lässt sich in drei grobe Schritte unterteilen. Zuerst wird die Proxygeometrie erzeugt und anschließend rasterisiert. Nach der Rasterisierung erfolgt die Texturierung durch 2D- oder 3D-Texturen abhängig von der verwendeten Proxygeometrie. Im letzten Schritt müssen die texturierten Ebenen zusammengefügt werden. Dies geschieht durch das Compositing. Werden die Ebenen der Reihe nach von hinten nach vorne gerendert, kann das Back-To-Front-Compositing, auch als Over-Operator [43] bekannt, eingesetzt werden. Die Farbe C'_i für die i. Ebene, $i \in \{0, \ldots, n-1\}$, lässt sich mit

$$C'_i = (1 - \alpha_i)\, C'_{i+1} + \alpha_i C_i \tag{2.1}$$

berechnen, wobei C'_n mit null initialisiert wird und C_i und α_i den Farbwert beziehungsweise die Opazität der Ebene i beschreiben. Mit dieser Traversierung der Ebenen ist es aber nicht möglich, den vorzeitigen Strahlabbruch zur Beschleunigung zu implementieren (siehe Abschnitt 2.3.3). Findet das Zeichnen der Proxygeometrie von vorne noch hinten statt, wird die Gleichung für Front-To-Back-Compositing verwendet.

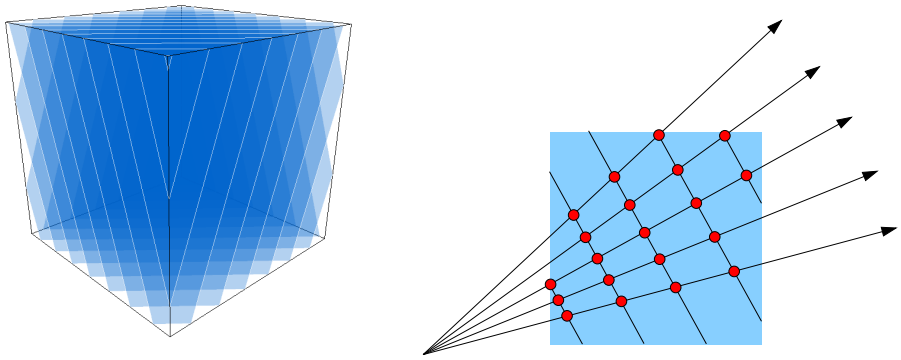


Abbildung 2.2: Slicing mit Ebenen, deren Normale gleich der Blickrichtung ist (View-Aligned). Links die Anordnung der Ebenen, rechts die Positionen der Abtastpunkte.

Mit i aus $\{1, \ldots, n\}$, $C'_0 = 0$ und $\alpha'_0 = 0$ ergibt sich

$$C'_i = C'_{i-1} + \left(1 - \alpha'_{i-1}\right) \alpha_i C_i \tag{2.2}$$

$$\alpha'_i = \alpha'_{i-1} + \left(1 - \alpha'_{i-1}\right) \alpha_i \tag{2.3}$$

Zu beachten ist, dass bei dieser Art des Compositing der berechnete Opazitätswert explizit im Framebuffer gespeichert werden muss.

Die Erzeugung der Proxygeometrie kann auf mehrere Arten erfolgen. Die einfachste Möglichkeit bietet die Ausrichtung der Ebenen entlang der Hauptachsen des Volumens (Object-Aligned Slicing). Dazu werden getrennt für jede Achse 2D-Texturen angelegt, die zur Texturierung der Proxygeometrie verwendet werden (Abbildung 2.1 links). Die Anzahl der Ebenen ist auf die Auflösung des Volumendatensatzes in der jeweiligen Dimension beschränkt. Für die Darstellung werden die Ebenen und Texturen der Hauptachse verwendet, die der Blickrichtung der Kamera am nächsten sind. Da 2D-Texturen verwendet werden, kann bei der Rasterisierung der Proxygeometrie nur bilineare Filterung eingesetzt werden. Jedoch lässt sich diese Technik auch auf älterer Grafikhardware ohne Unterstützung für 3D-Texturen realisieren. Durch die Ausrichtung entlang der Achsen ergeben sich aber Probleme: so wird das Dreifache des Speicherbedarfs des Volumens für die Texturen der drei Achsen benötigt. Hinzu kommt, dass der Abstand der Abtastpunkte nicht konstant ist, wenn sich die Kameraposition verändert. Wird die Kamera so verändert, dass eine andere Achsenrichtung als bisher angezeigt werden muss, kommt es zu Helligkeitsschwankungen, da sich die Positionen der Abtastpunkte verändern (Abbildung 2.1 Mitte und rechts).

Werden 3D-Texturen von der Grafikhardware bereitgestellt, lässt sich die Proxygeometrie auch direkt an der Blickrichtung der Kamera ausrichten (View-Aligned Slicing). Zur Bestimmung der Texturkoordinaten müssen die Ebenen mit der Boundingbox des Volumens geschnitten werden. Das Volumen wird als 3D-Textur gespeichert und diese mit trilinearer Filterung für jede Ebene abgetastet. Da wegen der trilinearen Filterung acht anstelle von vier Texturnachschlägen durchgeführt werden müssen und auf 3D-Texturen nicht so effizient zugegriffen werden kann wie auf 2D-Texturen, steigt die Renderzeit an. Jedoch lässt sich die Samplingdichte einfach erhöhen, indem mehr Ebenen verwendet werden. In Abbildung 2.2 sind die Ausrichtung der Ebenen zur Kamera und die entsprechenden Abtastpunkte dargestellt. Durch Verwendung der Preintegration kann die Qualität des Slicings weiter verbessert werden [48].

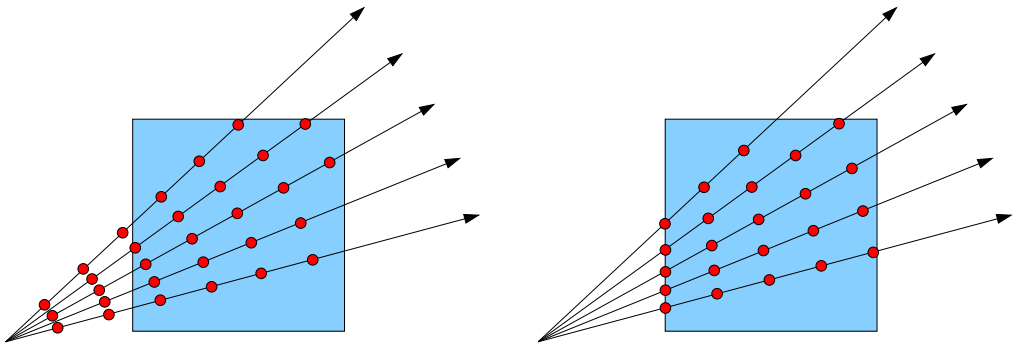


Abbildung 2.3: Strahlenverlauf und Position der Abtastpunkte beim Ray-Casting. Links liegt die Startposition im Auge, rechts auf der Oberfläche des Volumens

2.3.2 Ray-Casting

Ray-Casting besitzt große Ähnlichkeit mit Ray-Tracing, jedoch werden nur die Primärstrahlen betrachtet und die Sekundärstrahlen vernachlässigt. Bei beiden Verfahren, werden vom Auge beziehungsweise der Kamera aus Strahlen durch jedes Pixel der Bildebene geschickt. Für jeden Strahl wird überprüft, ob ein Objekt vom Strahl getroffen wird. Da im Volumen keine explizite Geometrie gegeben ist, kann keine Schnittberechnung mit den Strahlen durchgeführt werden. Stattdessen werden die Strahlen mit konstantem Abstand innerhalb des Volumens abgetastet [34, 7]. Als Startposition kann für die Strahlen die Augposition oder ein Punkt auf der Volumenoberfläche verwendet werden. Dadurch können sich aber unterschiedliche Positionen der Abtastpunkte ergeben, wie in Abbildung 2.3 zu sehen ist.

Da beim Ray-Casting die Berechnung für jeden Strahl unabhängig von den anderen erfolgt, eignet sich dieses Verfahren für die parallele SIMD-Architektur heutiger Grafikkarten. Purcell et al. stellten dazu im Jahr 2002 ein Konzept für Ray-Tracing auf der GPU vor [44]. Ein Jahr darauf stellten Krüger und Westerman [30] und Röttger et al. [47] jeweils einen Algorithmus für GPU-basiertes Ray-Casting zur Volumenvisualisierung vor.

Für die Durchführung des Ray-Casting auf der GPU wird der Volumendatensatz in einer 3D-Textur gespeichert. Zur Bestimmung der Anfangspositionen wird nur die Boundingbox des Volumens mit angepassten Texturkoordinaten gerendert. Durch die Interpolation während der Rasterisierung ergibt sich so für jedes Fragment ein Startpunkt. Zusammen mit der Kameraposition kann nun die Strahlrichtung bestimmt werden. Anschließend wird das Volumen entlang jeden Strahls traversiert. Texturzugriffe auf den Datensatz erfolgen dabei mit trilinearer Filterung. Die einzelnen Samples werden mittels Front-To-Back-Compositing (Gleichungen (2.2) und (2.3)) geblendet.

Da 2003 die maximale Anzahl der Instruktionen pro Fragmentprogramm sehr gering war, waren Krüger und Röttger gezwungen, die Traversierung des Volumens in mehrere Teilschritte zu zerlegen. Erst seit der Einführung von Schleifen und Verzweigungen mit Shader Model 3.0 und einer größeren maximalen Anzahl an Instruktionen [50] ist es möglich, das Ray-Casting in nur einem Renderpass durchzuführen [28, 54]. Stegmaier et al. führen dazu die Strahlintegration im Inneren von zwei ineinander geschachtelten Schleifen durch [54].

2.3.3 Beschleunigung der GPU-Verfahren

Die Volumenvisualisierung auf der GPU lässt sich durch angepasste Beschleunigungsverfahren aus der direkten Volumenvisualisierung ebenfalls beschleunigen [30]. So kann zum Beispiel wertvolle Zeit eingespart werden, wenn große Bereiche des Volumens leer sind. Diese Technik ist unter dem Namen *Empty Space Leaping*, dem Überspringen von leerem Raum, bekannt [64, 47, 36, 32, 28]. Westermann und Sevenich [64] verwenden dazu in Ihrer GPU-Implementierung einen zusätzlichen Renderpass, der das Volumen traversiert und dabei die Position bestimmt, an der zum ersten Mal ein Datenwert ungleich null ist. Diese so gefundenen Positionen werden beim nachfolgenden Ray-Casting als Startpositionen verwendet. Um weitere Zeit zu sparen kann auch die Berechnung des Volumenintegrals abgebrochen werden. Dieser vorzeitige Strahlabbruch (*Early Ray Termination*) tritt ein, wenn die Opazität einen bestimmten Schwellwert überschritten hat oder keine nennenswerten Beiträge mehr zu erwarten sind [1, 34, 7]. Das Volumen muss dazu von vorne nach hinten traversiert werden. Auf der GPU lässt sich dieser Strahlabbruch auch einsetzen, wenn Fragmentprogramme verwendet werden können. Dabei wird im Shader die Abbruchbedingung überprüft und die weitere Berechnung gegebenenfalls abgebrochen [47, 54].

Auf heutiger Grafikhardware bietet sich zusätzlich zu den eben angeführten Techniken eine weitere Möglichkeit zur Beschleunigung an, den *Early-Z-Test* [67]. Dabei handelt es sich um einen Test, der hardwareseitig in die Renderingpipeline integriert ist. Der Name des Early-Z-Tests rührt daher, dass ein zusätzlicher Tiefentest direkt nach der Rasterisierung durchgeführt wird, bevor für das Fragment das Fragmentprogramm und die nachfolgenden Tests durchgeführt werden. Ist dieser Tiefentest erfolgreich, wird das Fragment weiter in der Renderingpipeline verarbeitet, ansonsten verworfen. Voraussetzung für das Funktionieren des Early-Z-Test ist, dass weder das nachfolgende Fragmentprogramm den Tiefenwert des Fragments verändert noch Zustände von OpenGL, die den Tiefenpuffer betreffen, beeinflusst werden. Um von diesem Test zu profitieren, wird in einem zusätzlichen Renderpass der Tiefenpuffer mit einem weiteren Fragmentprogramm so präpariert, dass nicht zu berechnende Fragmente blockiert werden. Wird für die Vergleichsfunktion des Tiefentests der Vergleich auf kleiner, GL_LESS in OpenGL, gesetzt, werden mit einem Tiefenwert gleich null weitere Fragmente an dieser Position verworfen. Dagegen werden bei einem Wert gleich eins die Fragmente der nachfolgenden Fragmentprogramme wie gewöhnlich verarbeitet. Nach dem Vorbereiten des Tiefenpuffers wird das Schreiben des Puffers ab- und der Tiefentest angeschaltet. Anschließend kann mit der darzustellenden Geometrie wie bisher verfahren werden.

Es muss jedoch davon ausgegangen werden, dass die Beschleunigung durch den Early-Z-Test nicht so ist wie durch das Auslassen von Geometrie vor der Rasterisierung. Da die Grafikhardware den Viewport in mehrere Blöcke unterteilt, sollten diese möglichst homogen aufgebaut sein. So genügt beispielsweise die Berechnung eines Fragments des Blocks, während die verbleibenden Fragmente vom Early-Z-Test verworfen werden, um die Beschleunigung dieser Überprüfung zunichte zu machen. Allerdings ergeben sich durch den Early-Z-Test auch neue Möglichkeiten. Beim Slicing kann, vorausgesetzt die Proxygeometrie wird von vorn nach hinten gerendert, mit diesem zusätzlichen Test die Erkennung von leeren Voxeln und auch der vorzeitige Strahlabbruch durchgeführt werden. Dazu wird in dem Fragmentprogramm, das den Tiefenpuffer präpariert, geprüft, ob die Opazität des Zwischenergebnisses hoch genug oder das nächste Voxel leer ist. Bei einfachen Fragmentprogrammen, wie beispielsweise bei

der reinen Volumenvisualisierung, kann dieser Ansatz aber zur Verschlechterung der Renderzeit führen, da pro Ebene das Fragmentprogramm zweimal gewechselt werden muss. Ist die Berechnung pro Abtastpunkt aber komplexer, kann sich die Renderzeit insgesamt verkürzen.

3 LIC – Line Integral Convolution

Die Line Integral Convolution, kurz LIC, ist eine von vielen Techniken zur Visualisierung von Vektorfeldern. Der Schwerpunkt der Vektorfeldvisualisierung liegt im Bereich der Strömungssimulation. Diese Art der Visualisierung ist unter dem Begriff *Flow Visualization* bekannt. Dabei handelt es sich um die Darstellung der Bewegung von Flüssigkeiten und Gasen. Aber auch bei der Repräsentation von elektrischen oder magnetischen Feldern findet die Vektorfeldvisualisierung Anwendung.

Die zu visualisierenden Daten stammen aus Simulationen, physikalischen Messungen oder aus analytischen Modellen. Für die Simulation von Strömungen werden Methoden aus dem Bereich der numerischen Strömungsmechanik, kurz CFD (*Computational Fluid Dynamics*), verwendet. Meist wird dabei versucht, die Navier-Stokes-Gleichungen zu lösen. Die Klassifizierung der Daten erfolgt anhand der verwendeten Dimensionen, der Unterscheidung zwischen stationärer oder instationärer Strömung, dem verwendeten Gittertyp (kartesisch, regulär, curvilinear, unstrukturiert) und ob das Medium kompressibel ist.

Ein Vektorfeld ist durch Geschwindigkeit und Richtung $\mathbf{v}(\mathbf{x})$ definiert. Dabei entspricht $\mathbf{x}$ einer Gitterposition der Abtastpunkte des Datensatzes. Handelt es sich um eine nicht stationäre Strömung, ist die Geschwindigkeit $\mathbf{v}(\mathbf{x}, t)$ auch abhängig von der Zeit t. Zusätzlich kann das Vektorfeld weitere externe Größen wie beispielsweise den Druck p, die Temperatur T, die Dichte ρ und auch die Wirbelstärke (Vorticity) $\nabla \times \mathbf{v}$ enthalten.

Zur direkten Visualisierung von Vektorfeldern können Pfeile und Bildzeichen (Glyphen) [57] verwendet werden. Diese werden zur Darstellung von lokalen Merkmalen eingesetzt. So kann mit Pfeilen die Richtung der Strömung repräsentiert werden. Die Geschwindigkeit lässt sich beispielsweise durch die Länge des Pfeils oder durch eine Farbkodierung darstellen. Durch die Verwendung von Bildzeichen können mehrere Größen gleichzeitig visualisiert werden. Bei der Verwendung von Datensätzen höherer Auflösung treten jedoch mit diesen beiden Techniken sehr schnell Verdeckungsprobleme auf.

Eine weitere Möglichkeit zur Vektorfeldvisualisierung bietet sich durch das Verfolgen von individuellen Partikeln an. Dabei wird ein einzelnes masseloses Partikel oder auch mehrere in das Vektorfeld eingestreut und verfolgt. Dadurch ergeben sich die *charakteristischen Linien* des Vektorfelds. Es kann zwischen vier Arten unterschieden werden: *Pfadlinien* (*Pathlines*), *Strombahnen* (*Streaklines*), *Zeitlinien* (*Timelines*) und *Stromlinien* (*Streamlines*). Für die Erzeugung von Pfadlinien wird der Weg eines Partikels über die Zeit beobachtet und die Positionen werden mit einer Linie verbunden. Bei den Strombahnen werden die Partikel beziehungsweise ein Farbstoff an einer festen Position in das Feld eingestreut. Zeitlinien entstehen durch die Ausbreitung von Linien oder Flächen, die sich aus Partikeln zusammensetzen. Wird das Vektorfeld zu einem vorher festgesetzten Zeitpunkt t betrachtet und ein Partikel entlang der Tangenten des Felds bewegt ergeben sich Stromlinien. Pfadlinien, Strombahnen und Stromlinien sind im Fall von stationären Strömungen identisch.

Formal gesehen ist eine Stromlinie eine mögliche Lösung des Eigenwertproblems einer gewöhnlichen Differentialgleichung zum Zeitpunkt t.

$$\boldsymbol{\phi}(0) = \mathbf{x}_0 \qquad \frac{\mathrm{d}\,\boldsymbol{\phi}(s)}{\mathrm{d}\,s} = \mathbf{v}(\boldsymbol{\phi}(s), t) \tag{3.1}$$

Damit ergibt sich für eine Stromlinie durch $\mathbf{x}_0$ eine Kurve $\boldsymbol{\phi}(s)$ mit dem Parameter s.

Globale Verfahren zur Visualisierung von Vektorfeldern sind im Bereich der dichten und texturbasierten Strömungsvisualisierung zu finden. Ein Überblick der hierbei eingesetzten Techniken ist in *The State of the Art in Flow Visualization: Dense and Texture-Based Techniques* von Laramee et al. [33] zu finden. Van Wijk stellte 1991 einen Algorithmus vor, der Daten mit Hilfe von Texturen visualisiert [59]. Dazu verwendet er ein Rauschen an zufällig verteilten Positionen, welches er *Spot Noise* nennt. Wird dieses Rauschen zur Visualisierung der Geschwindigkeit eines Vektorfelds eingesetzt, so wird die Textur entlang der Bewegungsrichtung skaliert. Bei der Texturadvektion [39] werden dagegen die Texel einer Textur entlang des zugrunde liegenden Vektorfelds verschoben.

Line Integral Convolution gehört ebenfalls zu den texturbasierten Verfahren. Dabei wird die Domäne mit einer Rauschtextur bedeckt und diese anschließend entlang von Stromlinien verwischt. Das Verwischen erfolgt durch die Faltung der Rauschtextur mit einem Filter. Sei $T(\mathbf{x}')$ die Intensität der Rauschtextur an der Stelle $\mathbf{x}'$ und sei k ein Filter, so ergibt sich für die Faltung

$$\varrho(\mathbf{x}) = \int k(\mathbf{s}) \cdot T(\mathbf{s} - \mathbf{x})\,\mathrm{d}\,\mathbf{s} \tag{3.2}$$

Wird die Faltung entlang der Stromlinie $s \to \boldsymbol{\phi}_x(s)$ durch den Punkt $\mathbf{x}$ durchgeführt, folgt für die Intensität ϱ an dieser Stelle

$$\varrho(\mathbf{x}) = \int_{-L}^{L} k(s) \cdot T(\boldsymbol{\phi}_x(s + s_0))\,\mathrm{d}\,s \tag{3.3}$$

Der Filter besitzt hier nur lokalen Einfluss im Intervall $[-L, L]$. Wird dieses Intervall zu groß gewählt, nähert sich die berechnete Intensität dem Mittelwert der Rauschtextur an. Ist es dagegen zu klein, ist der Effekt des Verwischens minimal. Als Filter wird meist ein einfacher Boxfilter oder ein isotroper Filter verwendet. Durch die Filterung ergibt sich entlang einer Stromlinie eine hohe Korrelation, jedoch nicht zwischen benachbarten Stromlinien.

3.1 2D LIC

Die Erfinder der Line Integral Convolution, Cabral und Leedom, stellten diese Technik zur Visualisierung von zweidimensionalen Strömungen im Jahr 1993 vor [4]. Dabei wird eine Rauschtextur mittels Faltung entlang von Stromlinien verschmiert. Die Integration erfolgt dabei aber weder kontinuierlich noch diskret mit konstantem Abstand, sondern pro Gitterzelle des Vektorfelds, die von der Stromlinie traversiert wird. Dadurch ist es möglich, der Stromlinie durch jede Zelle zu folgen. Die Rauschtextur besitzt dieselbe Auflösung wie das Vektorfeld

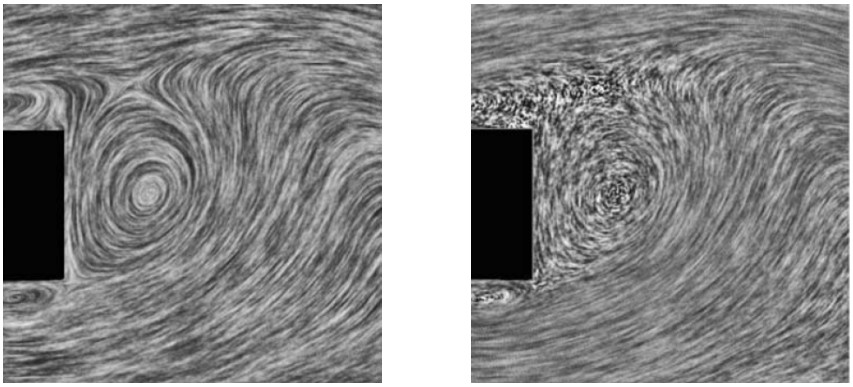


Abbildung 3.1: Vergleich zwischen LIC (links) und Spot Noise (rechts).[33]

und besteht aus weißem Rauschen. Shen et al. [51] kombinierten 1996 den LIC mit der Advektion von Farbstoff. Dabei handelt es sich jedoch nicht um eine physikalisch korrekte Advektion, da die Diffusion vernachlässigt wurde.

Stalling und Hege [53] stellten 1995 einen beschleunigten, auflösungsunabhängigen LIC unter dem Namen FastLIC vor. Sie beschleunigten die Berechnung, indem die LIC-Integration nicht mehr für jedes einzelne Pixel sondern entlang einer Stromlinie durchgeführt wird und Zwischenergebnisse wiederverwendet werden. Dazu muss jedoch ein Boxfilter eingesetzt werden. 1998 erweiterten sie den FastLIC um Filter auf Polynombasis [15].

Die Berechnung des LIC auf curvilinearen Gittern wurde durch Forssell [13] möglich. Forssell führte in dieser Arbeit auch eine Technik ein, mit der die Geschwindigkeit des Vektorfelds visualisiert und der LIC animiert werden kann. Zusammen mit Cohen hat Forssell den LIC zur Darstellung nicht stationärer Strömungen 1995 erweitert [14]. Shen und Kao erweiterten das Konzept für nicht stationäre Strömungen und gaben ihm den Namen UFLIC (Unsteady Flow LIC) [52]. 2006 stellten Li et al. eine GPU-basierte Version des UFLIC, genannt GPUFLIC, vor [35].

Beim Multifrequency LIC von Kiu und Banks [27] wird die Raumfrequenz des verwendeten Rauschens abhängig von der Geschwindigkeit verändert. Urness et al. [58] verwenden für ihren Multivariate LIC das Multifrequenz-Rauschen von Kiu und Banks nicht für die Visualisierung der Geschwindigkeit sondern für Features. Zusätzlich erlaubt ihre *Color-Weaving*-Technik (Verweben von Farben) die gleichzeitige Darstellung mehrerer Skalardaten zusammen mit den Vektorattributen. Mit dem HyperLIC von Zheng et al. [68] lassen sich symmetrische zwei- oder dreidimensionale Tensorfelder visualisieren.

Die Richtung der Strömung ist beim klassischen LIC nicht erkennbar. Um diese visualisieren zu können nutzen Wegenkittl et al. [60] für ihren OLIC (Oriented LIC) einen asymmetrischen Filter. Aufgrund der hohen Dichte der Stromlinien reicht dies jedoch nicht allein aus. Sie verwenden deshalb eine Rauschtextur, die nur dünn besetzt ist.

De Leeuw und van Liere haben in Ihrer Arbeit Spot Noise und LIC verglichen [8]. Sie kamen zu dem Schluss, dass der LIC die Richtung des Flusses mehr hervorhebt, jedoch keine Geschwindigkeitsinformation darstellt. In Abbildung 3.1 sind die Ergebnisse einer Berechnung durch LIC und Spot Noise gegenübergestellt.

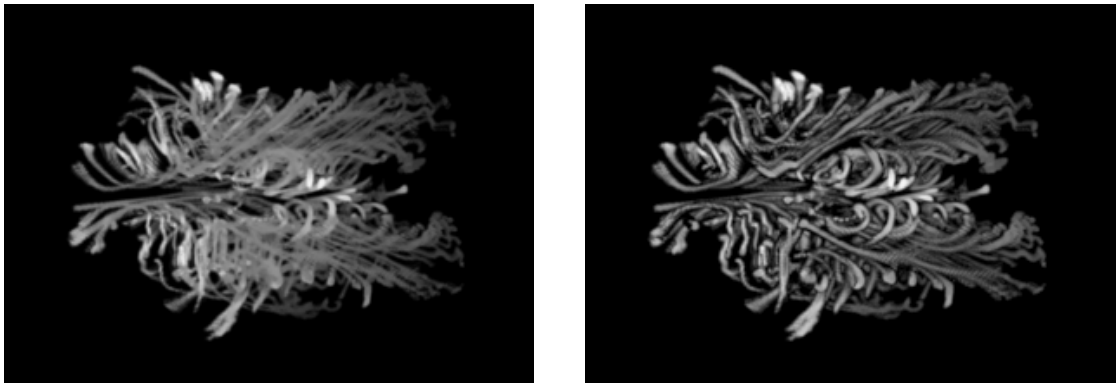


Abbildung 3.2: VolumeLIC mit Abbildung der Geschwindigkeit auf den Alphawert. Links ohne Halo, rechts mit Halo zur Hervorhebung der Tiefeninformationen. (Eigentum von Victoria Interrante) [http://www-users.cs.umn.edu/∼interran/3Dflow.html]

3.2 3D LIC

Der 3D LIC ist eine Erweiterung des 2D LIC. Die Berechnung des LIC in drei anstatt in zwei Dimensionen erfolgt auf dieselbe Weise mit Gleichung (3.3), da die Definition der Stromlinien (Gleichung (3.1)) auch im dreidimensionalen Raum gültig ist. Durch den Dimensionswechsel ergeben sich jedoch Probleme. So nimmt die visuelle Komplexität und die Verdeckung zu. Aber auch die Komplexität der Berechnung nimmt zu, da mehr Stromlinien berechnet werden.

Interrante und Grosch [19, 20] stellten 1997 mit VolumeLIC erstmals einen volumetrischen LIC vor. Zur Reduktion der visuellen Komplexität wird die Geschwindigkeit auf den Alphawert abgebildet und Halos eingesetzt. Dadurch ergeben sich Bilder mit klarer Tiefenstruktur (Abbildung 3.2). Dies sehen Laramee et al. [33] als „Schritt in Richtung geometrischer Strömungsvisualisierung, bei der diskrete Objekte wie Stromlinien unterschieden werden können“.

Um Benutzerinteraktionen trotz langer Rechenzeiten des 3D LIC zu ermöglichen, verwenden Rezk-Salama et al. [45] einen Vorberechnungsschritt. Während dieser Vorbereitung wird der 3D LIC berechnet und in einer 3D-Textur gespeichert. Diese Textur wird anschließend mittels texturbasierter Volumenvisualisierung (vgl. Abschnitt 2.3.1) angezeigt. Mittels geeigneter Transferfunktionen und Clipping (siehe Abschnitt 3.5) ist es möglich, interessante Teile der Strömung zu extrahieren. Suzuki et al. [56] setzen ebenfalls eine 3D-Textur ein, jedoch findet die Visualisierung auf der VolumePro-Hardware statt.

Zur Verringerung der für die LIC-Berechnung benötigten Zeit kann auch der sogenannte Seed LIC eingesetzt werden [16]. Dabei handelt es sich um einen LIC, dessen Stromlinien nur von bestimmten Punkten aus, definiert durch eine dünnbesetzte Seed-Textur, berechnet werden.

3.3 GPU-basierter 3D LIC

Grundlage dieser Arbeit ist der erstmalige Einsatz eines GPU-basierten Ansatzes zur Berechnung des dreidimensionalen LIC. Die Berechnung der Stromlinien erfolgt für den LIC, auch im dreidimensionalen Fall, getrennt pro Pixel, sieht man von der Verwendung des FastLIC

ab. Dies spricht aufgrund der hohen Parallelisierbarkeit für die Verwendung der GPU als *General Purpose GPU* (GPGPU). Dazu wird das Vektorfeld in einer 3D-Textur repräsentiert und die Berechnung des LIC erfolgt in Fragmentprogrammen. Zur Beschleunigung eignen sich die Techniken, die auch bei der Volumenvisualisierung auf der GPU zum Einsatz kommen (siehe Abschnitt 2.3.3). Dazu gehören der Early-Z-Test, das Empty-Space-Leaping und der vorzeitige Strahlabbruch. Die visuelle Komplexität wird verringert durch die Verwendung von dünnem Rauschen anstelle von weißem Rauschen, dem Einsatz geeigneter Transferfunktionen und der Beleuchtung der Stromlinien. Um bei der Visualisierung und speziell bei der Interaktion Aliasingeffekten vorzubegeugen, sollte die Raumfrequenz des verwendeten Rauschens im Bildraum konstant sein.

3.4 Beleuchtung von Linien

Wird ein $k$dimensionales Objekt in einen n-dimensionalen Euklidischen Raum $n > k$ eingebettet, ist die Kodimension gegeben durch $n - k$. Dadurch ist die Kodimension von Flächen im dreidimensionalen Raum gleich eins. In diesem Fall erfolgt die Beleuchtungsberechnung anhand der eindeutigen Normale der Fläche [12]. Lediglich die Orientierung der Fläche muss dabei berücksichtigt werden. Für Kodimensionen größer eins existieren viele mögliche Normalen. So liegen beispielsweise bei dreidimensionalen Linien mit Kodimension zwei die möglichen Normalen in einer Ebene, die orthogonal zur Tangente der Linie liegt. Banks stellte zu diesem Problem die Arbeit *Beleuchtung in diversen Kodimensionen* vor [2]. Für die Beleuchtung von Linien mit Kodimension zwei verwendet er unendlich dünne Zylinder und maximiert getrennt für den diffusen und spekularen Anteil das reflektierte Licht entlang des Umfangs. In seiner Arbeit geht Banks auch auf die zunehmende Helligkeit mit zunehmender Kodimension ein.

Zöckler et al. [69] stellten 1996 eine Methode zur Beleuchtung von Stromlinien vor. Darin setzten sie das Modell von Banks ein. Mallo et al. [38] verwenden ebenfalls unendlich dünne Zylinder, gebrauchen aber nicht die Maxima für den spekularen und diffusen Anteil. Stattdessen integrieren sie die Intensität über den sichtbaren und beleuchteten Bereich des Zylinders.

Eine weitere Möglichkeit bietet die Verwendung des Gradienten als Normale [62, 63]. Bei der Volumenvisualisierung kann der Gradient aus dem gegebenen Skalarfeld berechnet werden. Weiskopf et al. berechnen den Gradienten nach jedem Zeitschritt ihrer Texturadvektion [62, 63]. Werden die Stromlinien durch einen LIC erzeugt, können die Gradienten der Rauschtextur verwendet werden (Abschnitt 3.4.3).

Die Berechnung der Beleuchtung erfolgt mit dem Modell von Phong [42] beziehungsweise mit dem Blinn-Phong-Modell von Blinn [3]. Beim Phong-Modell ergibt sich die Intensität mit

$$I = k_a + k_d \mathbf{L} \cdot \mathbf{N} + k_s \left(\mathbf{V} \cdot \mathbf{R}\right)^n \tag{3.4}$$

Dabei sind k_a, k_d und k_s jeweils die Koeffizienten für den ambienten, den diffusen und den spekularen Anteil des Lichts. $\mathbf{V}$ steht für die Blickrichtung, $\mathbf{N}$ für die Normale, $\mathbf{L}$ für die Lichtrichtung und $\mathbf{R}$ für den Reflektionsvektor von $\mathbf{L}$ an der Normale $\mathbf{N}$. Die Größe des spekularen Glanzpunkts wird vom Exponenten n beeinflusst.

Abbildung 3.3: Textur für die Beleuchtung von Stromlinien nach Zöckler. Die Abszisse gibt das Skalarprodukt $\mathbf{L} \cdot \mathbf{T}$, die Ordinate das Skalarprodukt $\mathbf{V} \cdot \mathbf{T}$ wieder.

Das Blinn-Phong-Modell approximiert den Winkel zwischen Reflektionsvektor und Blickrichtung mit dem Winkel zwischen der Normalen und dem Halfwayvektor. Der Halfwayvektor $\mathbf{H}$ ist definiert durch $\mathbf{H} = (\mathbf{V} + \mathbf{L}) / \|\mathbf{V} + \mathbf{L}\|$. Daraus folgt für die Intensität nach Blinn-Phong

$$I = k_a + k_d \mathbf{L} \cdot \mathbf{N} + k_s \left(\mathbf{H} \cdot \mathbf{N}\right)^n \tag{3.5}$$

3.4.1 Stromlinien nach Zöckler

Zöckler et al. [69] wählen für die Beleuchtung von Stromlinien die Normale aus, mit der die berechnete Intensität am größten ist. Dazu verwenden sie die Normale, die koplanar zu dem Tangentenvektor $\mathbf{T}$ der Linie und der Lichtrichtung $\mathbf{L}$ ist. Damit ist es möglich, die Beleuchtung mittels Gleichung (3.4) zu berechnen. Um die Berechnung von der Normalen unabhängig zu machen, führen Zöckler et al. mehrere Umformungen durch.

Wird die Lichtrichtung $\mathbf{L}$ im Tangentenraum der Linie zerlegt, ergibt sich

$$\mathbf{L} = \mathbf{L}_N + \mathbf{L}_T \tag{3.6}$$

Hieraus folgt mit dem Satz von Pythagoras für den diffusen Teil der Gleichung (3.4)

$$\mathbf{L} \cdot \mathbf{N} = \|\mathbf{L}_N\| = \sqrt{1 - \|\mathbf{L}_T\|^2} = \sqrt{1 - \left(\mathbf{V} \cdot \mathbf{T}\right)^2} \tag{3.7}$$

Der spekulare Anteil lässt sich durch eine ähnliche Umformung ohne die Normale N ausdrücken

$$\begin{aligned} \mathbf{V} \cdot \mathbf{R} &= \mathbf{V} \cdot \left(\mathbf{L}_T - \mathbf{L}_N\right) \\ &= \left(\mathbf{L} \cdot \mathbf{T}\right)\left(\mathbf{V} \cdot \mathbf{T}\right) - \sqrt{\left(\mathbf{L} \cdot \mathbf{T}\right)^2}\sqrt{\left(\mathbf{V} \cdot \mathbf{T}\right)^2} \end{aligned} \tag{3.8}$$

Mit diesen beiden Gleichungen ist es nun möglich, die Gleichung (3.4) des Phong-Modells nur abhängig von den Skalarprodukten $\mathbf{L} \cdot \mathbf{T}$ und $\mathbf{V} \cdot \mathbf{T}$ auszudrücken und in einer 2D-Textur abzuspeichern (Abbildung 3.3). Das Beleuchten der Linien erfolgt durch Texturierung mit dieser 2D-Textur. Dazu wird für die Texturkoordinaten eines Liniensegments die Tangente verwendet. Wird von gerichtetem Licht und einer orthographischen Kamera ausgegangen, können

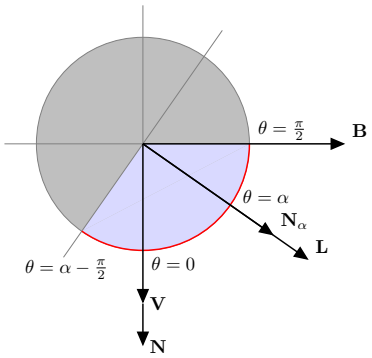


Abbildung 3.4: Querschnitt eines Zylinders zur Beleuchtungsberechnung. Der vom Auge aus sichtbare und diffus beleuchtete Teil der Oberfläche ist rot eingefärbt.

die zwei für den Nachschlag benötigten Skalarprodukte durch die Texturtransformationsmatrix ausgerechnet werden. Dazu wird die Matrix mit der Lichtrichtung **L** und der Blickrichtung **V** folgendermaßen gefüllt:

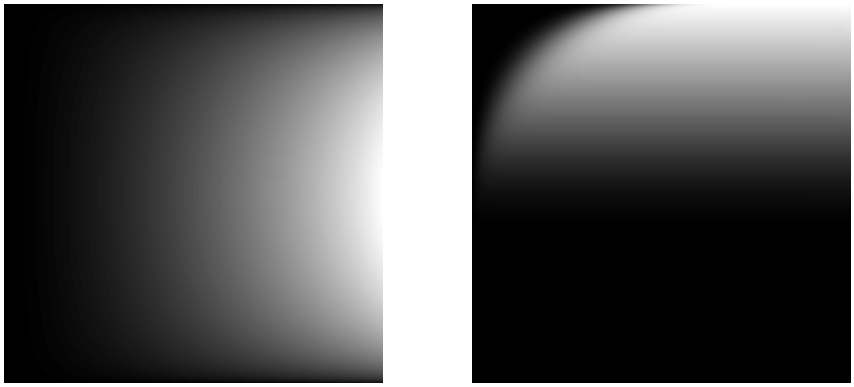
$$M = \frac{1}{2} \begin{pmatrix} L_x & V_x & 0 & 0 \\ L_y & V_y & 0 & 0 \\ L_z & V_z & 0 & 0 \\ 1 & 1 & 0 & 2 \end{pmatrix} \tag{3.9}$$

Wird diese Matrix mit den Texturkoordinaten der Liniensegmente multipliziert, ergeben sich neue Texturkoordinaten zwischen 0 und 1, mit denen der Texturnachschlag durchgeführt werden kann.

Da die Normale immer in Richtung Lichtquelle ausgerichtet ist, existiert keine definierte „Rückseite“ der Linie. Dies führt dazu, dass die Linie dieselbe diffuse Intensität besitzt, wenn sich das Vorzeichen der Lichtrichtung ändert. In diesem Zusammenhang wird von bidirektionaler Beleuchtung gesprochen. Der Effekt lässt sich minimieren, wenn die Lichtrichtung gleich der Blickrichtung ist. Zöckler et al. verwenden für den diffusen Term zusätzlich einen Exponenten, um den Überschuss der Helligkeit aufgrund der höheren Kodimension auszugleichen. Banks schlägt dafür einen konstanten Wert von 4,8 vor.

3.4.2 Stromlinien nach Mallo

In dem Ansatz von Mallo et al. [38] erfolgt die Beleuchtung nicht durch die Transformation von Texturkoordinaten und anschließendem Texturnachschlag. Vielmehr wird dazu die programmierbare Pipeline der Grafikkarte mit Vertex- und Fragmentprogramm verwendet. Ihr Verfahren verbessert die diffusen Reflektionen von Zöckler et al. [69]. Dazu wird der diffuse und der spekulare Anteil nicht über die maximale Intensität bestimmt, sondern wie bei Schussmann und Ma [49] berechnet. Dies geschieht durch Integration über den beleuchteten und sichtbaren Teil der Oberfläche eines unendlich dünnen Zylinders. In Abbildung 3.4 ist der sichtbare Teil des Zylinders rot eingefärbt. Der Tangentenraum entlang der Linie ist definiert durch die Tangente, die Normale und die Binormale $(\mathbf{T}, \mathbf{N}, \mathbf{B})$. Die Binormale ist gegeben mit $\mathbf{B} = \mathbf{T} \times \mathbf{V} / \|\mathbf{T} \times \mathbf{V}\|$ und die Normale mit $\mathbf{N} = \mathbf{B} \times \mathbf{T}$.



(a) (b)

Abbildung 3.5: Texturen für die Beleuchtung von Stromlinien nach Mallo. (a) Diffuser Anteil: Auf der Abszisse ist $\cos(\alpha)$ und auf der Ordinate $\mathbf{L} \cdot \mathbf{T}$ dargestellt. (b) Spekularer Anteil: Auf der Abszisse ist $\cos(\alpha)$ und auf der Ordinate $\cos(\beta)$ dargestellt.

Der sichtbare Bereich des Zylinders hängt von dem Winkel α zwischen den Projektionen von $\mathbf{V}$ und $\mathbf{L}$ auf die Ebene $\mathbf{N}, \mathbf{B}$ ab. Dieser Winkel lässt sich mit Hilfe von $\mathbf{V}$ und $\mathbf{L}$ ausdrücken:

$$\alpha = \arccos \frac{\mathbf{V} \cdot \mathbf{L} - \mathbf{V}_T \cdot \mathbf{L}_T}{\sqrt{1 - \|\mathbf{V}_T\|^2}\sqrt{1 - \|\mathbf{L}_T\|^2}} \tag{3.10}$$

Dabei bezeichnet $\mathbf{V}_T$ und $\mathbf{L}_T$ die Blickrichtung beziehungsweise die Lichtrichtung im Tangentenraum.

Für den diffusen Teil ergibt sich derselbe Term wie bei Zöckler (Gleichung (3.7)), jedoch skaliert

$$\mathbf{L} \cdot \mathbf{N} = \sqrt{1 - \|\mathbf{L}_T\|^2} \cdot \frac{\sin\alpha + (\pi - \alpha)\cos\alpha}{4} \tag{3.11}$$

Für den spekularen Anteil folgt für das Modell nach Blinn-Phong

$$\mathbf{H} \cdot \mathbf{N} = \sqrt{1 - \|\mathbf{H}_T\|}^n \int_{\alpha - \frac{\pi}{2}}^{\frac{\pi}{2}} \cos^n(\theta - \beta) \frac{\cos\theta}{2} \, \mathrm{d}\,\theta \tag{3.12}$$

mit β gleich dem Winkel zwischen der Projektion von $\mathbf{V}$ und $\mathbf{H}$ auf die Ebene $\mathbf{N}, \mathbf{B}$. Das Integral über θ muss numerisch gelöst werden (siehe [38]). Mallo et al. legen für den diffusen und den spekularen Anteil je eine vorberechnet Textur an. Die diffuse Textur T_{diff} wird dabei in Abhängigkeit von $\cos\alpha$ und $\mathbf{L} \cdot \mathbf{T}$ und die spekulare Textur T_{spec} von $\cos\alpha$ und $\cos\beta$ berechnet (Abbildung 3.5).

Beim Rendern werden der Lichtvektor $\mathbf{L}$, die Blickrichtung $\mathbf{V}$ und die Tangente $\mathbf{T}$ im Vertexprogramm berechnet. Im Fragmentprogramm werden die Binormale, die Normale und der Halfwayvektor ermittelt. Zusammen mit

$$\cos\alpha = \frac{\mathbf{L} \cdot \mathbf{N}}{\sqrt{1 - (\mathbf{L} \cdot \mathbf{T})^2}} \qquad \text{und} \qquad \cos\beta = \frac{\mathbf{H} \cdot \mathbf{N}}{\sqrt{1 - (\mathbf{H} \cdot \mathbf{T})^2}} \tag{3.13}$$

ergibt sich dann für das beleuchtete Fragment mit der Farbe C_{in}

$$C_{out} = C_{in}\left(k_a + k_d \cdot T_{diff}(\cos\alpha, \mathbf{L}\cdot\mathbf{T})\right) + k_s \cdot \sqrt{1-(\mathbf{H}\cdot\mathbf{T})^2} \cdot T_{spec}(\cos\alpha, \cos\beta) \tag{3.14}$$

3.4.3 Beleuchtung des LIC anhand von Gradienten

Der Gradient wird in der Volumenvisualisierung und auch bei der Texturadvektion zur Beleuchtungsberechnung verwendet [63]. Wird der Gradient des LIC benötigt, so könnte das Ergebnis des LIC zwischengespeichert und die Gradienten von diesem Skalarfeld berechnet werden. Anschließend könnte die Beleuchtungsberechnung für das Skalarfeld zusammen mit den Gradienten durchgeführt werden. Bei dem hier vorgestellten Ansatz erfolgt die Beleuchtungsberechnung gleichzeitig mit der Berechnung des LIC, um beispielsweise vom vorzeitigen Strahlabbruch profitieren zu können. Dazu wird der Gradient gleichzeitig mit der Faltung entlang der Stromlinien berechnet.

Die Dichte beziehungsweise Intensität $\varrho(\mathbf{x})$ des LIC entlang einer Stromlinie $s \rightarrow \boldsymbol{\phi}_x(s)$ ist gegeben durch Gleichung (3.3)

$$\varrho(\mathbf{x}) = \int_{-L}^{L} k(s) \cdot T(\boldsymbol{\phi}_x(s+s_0))\,\mathrm{d}\,s$$

Der Gradient der Dichte $\varrho(\mathbf{x})$ ist definiert durch

$$\nabla\varrho(\mathbf{x}) = \nabla \int_{-L}^{L} k(s) \cdot T(\boldsymbol{\phi}_x(s+s_0))\,\mathrm{d}\,s \tag{3.15}$$

Wird nun davon ausgegangen, dass der Gradient des Filters k annähernd konstant ist und es sich um gerade Stromlinien handelt, ergibt sich

$$\nabla\varrho(\mathbf{x}) = \int_{-L}^{L} k(s) \cdot \nabla T(\boldsymbol{\phi}_x(s+s_0))\,\mathrm{d}\,s \tag{3.16}$$

Damit ist der Gradient nur noch vom eingesetzten Rauschen abhängig. Das Rauschen darf aber kein reines weißes Rauschen sein, da benachbarte Gradienten durch die hohen Raumfrequenzen sonst keine Korrelation besitzen. Eine Möglichkeit bietet die Filterung des weißen Rauschens mit einem Tiefpassfilter, wodurch sich glattere Übergänge zwischen den Gradienten ergeben.

Wird anstelle des weißen Rauschen ein dünnes Rauschen eingesetzt und dieses ebenfalls mit einem Tiefpassfilter gefiltert, ergeben sich Stromlinienbündel. Diese besitzen im Inneren eine höhere Dichte als außen. Die berechneten Gradienten entsprechen den Normalen einer Isofläche um dieses Strahlenbündel. Dadurch entsteht durch die Beleuchtung der Eindruck von Geometrie.

3.4.4 Wahl der Transferfunktion

Eine Transferfunktion dient zur Abbildung eines Werts auf einen oder mehrere Werte. Dies ermöglicht, Skalarwerte auf Farbwerte und auch auf die Opazität abzubilden. Da der Skalarwert meist auf Farben abgebildet wird, werden die Abbildungen auf die einzelnen Farbkomponenten in einer Transferfunktion zusammengefasst und als Kanäle der Transferfunktion bezeichnet.

Die Wahl der richtigen Transferfunktion ist bei der Visualisierung von 3D LIC ebenso wichtig wie bei der Darstellung von Volumen. In der klassischen Volumenvisualisierung wird die Dichte des Volumens auf vier Kanäle der Transferfunktion abgebildet. Diese vier Kanäle lassen sich dabei in drei Abbildungen auf den Farbwert im RGB-Farbraum und einer Abbildung auf den Alphawert aufteilen. Dadurch ist es möglich, Bereiche mit einer bestimmten Dichte innerhalb des Volumens transparent andere opak darzustellen und gleichzeitig Farben für jeden Dichtewert zu definieren. In dieser Arbeit wurde eine Transferfunktion bestehend aus fünf Kanälen gewählt. Die ersten drei Kanäle bilden einen Skalarwert auf einen Farbwert ab. Bei den verbleibenden zwei Kanälen handelt es sich um Abbildungen auf den Alphawert. Als Eingabewert für den ersten Alphakanal kann die Geschwindigkeit des Vektors aber auch andere Größen, wie beispielsweise das Ergebnis der Lambda2-Berechnung, eingesetzt werden. Dadurch können Bereiche des Vektorfelds ausgeblendet oder hervorgehoben werden. Der zweite Alphakanal der Transferfunktion wird dazu verwendet, die Intensität ϱ, die sich aus der LIC-Berechnung ergibt, auf den Alphawert abzubilden. Dies ermöglicht zum Beispiel die Freistellung der Bereiche mit hoher Intensität. So können bei Verwendung eines dünnen Rauschens Stromlinienbündel freigelegt werden. Die zwei Alphakanäle können im Fragmentprogramm mittels Tensorprodukt miteinander kombiniert werden.

Wird die Transferfunktion entsprechend gewählt, kann sich ein Effekt namens *Limb Darkening* [16] einstellen. Es handelt sich dabei um ein Phänomen aus der Astrophysik. Dabei erscheint die Atmosphäre eines Sterns am Rand dunkler als in der Mitte. Mit einer Transferfunktion kann ein ähnlicher Effekt erzeugt werden, indem die Intensität der Farbe für die Skalarwerte am Rand geringer als im Inneren des Objekts ist. Gleichzeitig sollte der Alphawert am Rand des Objekts sehr gering sein und nach innen zunehmen.

3.5 Features

Features oder auch Merkmale sind in der Visualisierung von besonderem Interesse. Es handelt sich dabei um spezielle Bereiche, die sich meist im Inneren des Datensatzes befinden. In der medizinischen Visualierung könnten dies beispielsweise Knochen, Arterien oder auch Tumore sein. Im Bereich der Strömungsmechanik sind dies Wirbel und Turbulenzen.

Diese Features können auf verschieden Weisen extrahiert werden. So lässen sich mit einer geeigneten Transferfunktion Bereiche des Volumens ausblenden. Je nach verwendeten Skalardaten kann so eine Maskierung, zum Beispiel anhand der Geschwindigkeit (Velocity-Masking), durchgeführt werden. In Abbildung 3.6 wurden die langsameren Schichten um den Tornado ausgeblendet, so dass der Schlauch des Tornados besser zu erkennen ist. Besteht das Volumen jedoch überwiegend aus denselben Dichte- beziehungsweise Skalarwerten, wird das Feature mit ausgeblendet.

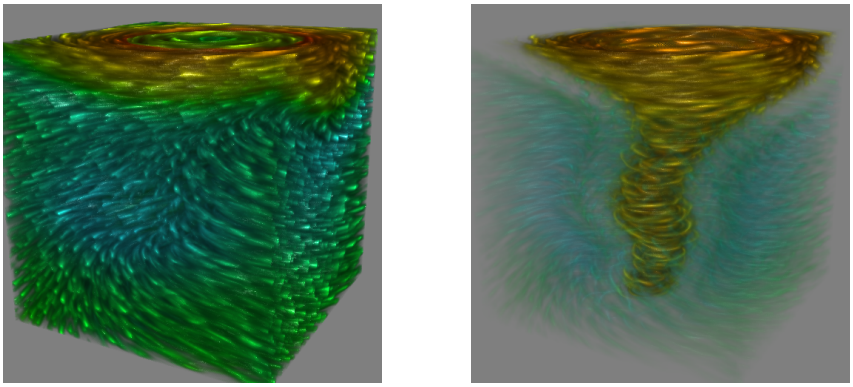


Abbildung 3.6: Ausblenden von Bereichen des Vektorfelds mittels Velocity-Masking. Links der opake Datensatz, rechts derselbe Datensatz mit Velocity-Masking. Als Datensatz wurde der Tornado von Roger Crawfis verwendet.

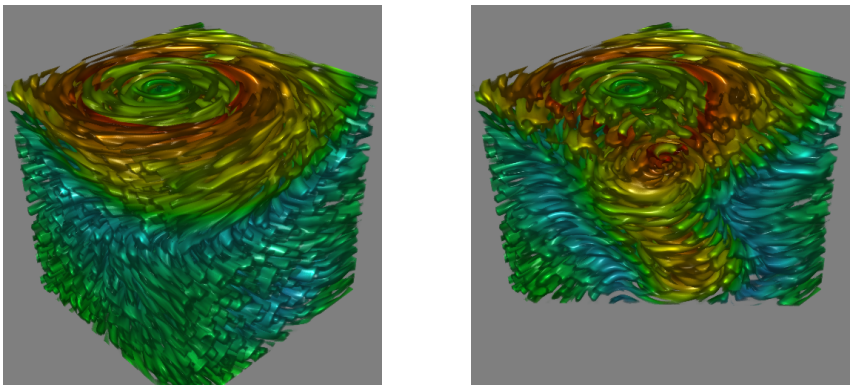


Abbildung 3.7: Links ist der vollständige Datensatz abgebildet, rechts wurde mit einer Clipebene der vordere Teil weggeschnitten. Als Datensatz wurde der Tornado von Roger Crawfis verwendet.

Zur Extraktion kann aber auch ein sogenanntes Importance-Volumen eingesetzt werden. Dabei handelt es sich um ein weiteres Volumen, dass jedoch einzig zur Maskierung der eigentlichen Daten dient. Diese Maskierung kann kontinuierlich oder diskret erfolgen. In den Daten von Strömungssimulationen können so mit Hilfe des Lambda2-Verfahrens Wirbel hervorgehoben werden. Dazu wird das Ergebnis aus der Lambda2-Berechnung in diesem Importance-Volumen gespeichert. Bei dem Seed LIC von Helgeland et al. [16] können mit der entsprechenden Gestaltung der Seed-Textur Features extrahiert werden.

Beim Clipping werden Teile des Volumens mit Hilfe von Primitiven weggeschnitten [41, 61]. Meist werden dazu Ebenen eingesetzt, die frei im Raum platziert werden können. Es können aber auch beliebig komplexe Formen verwendet werden [61]. Eine weitere Möglichkeit bietet die Schnittebene (Abschnitt 2.2). Mit ihr lassen sich einzelne Schichten aus dem Volumen extrahieren und analysieren.

3.5.1 Erkennung von Wirbeln

Jeong und Hussain stellten 1995 eine Methode vor, mit der es möglich ist, Wirbel innerhalb von Strömungen zu identifizieren [21]. Sie verwenden dazu die Eigenwerte der Jacobimatrix. Die Jacobimatrix, auch Geschwindigkeitsgradient-Tensor genannt, ist definiert durch

$$\underline{\mathbf{J}} = \nabla \mathbf{v}\left(\mathbf{x}, t\right) = \begin{pmatrix} \frac{\partial}{\partial x}\mathbf{v}_x & \frac{\partial}{\partial y}\mathbf{v}_x & \frac{\partial}{\partial z}\mathbf{v}_x \\ \frac{\partial}{\partial x}\mathbf{v}_y & \frac{\partial}{\partial y}\mathbf{v}_y & \frac{\partial}{\partial z}\mathbf{v}_y \\ \frac{\partial}{\partial x}\mathbf{v}_z & \frac{\partial}{\partial y}\mathbf{v}_z & \frac{\partial}{\partial z}\mathbf{v}_z \end{pmatrix} \tag{3.17}$$

Die Eigenwerte der Jacobimatrix erlauben eine Klassifizierung der Strömung [26]. So können Verwirbelungen, Anziehung und auch ein Abbremsen der Strömung festgestellt werden. Zur einfacheren Bestimmung der Eigenwerte wird die Jacobimatrix in einen symmetrischen Teil, den Diffusionstensor, und einen antisymmetrischen Teil, den Rotationstensor, unterteilt. Der Diffusionstensor beschreibt den Fluss in und aus einer Region und ist gegeben durch

$$\underline{\mathbf{S}} = \frac{1}{2}\left(\underline{\mathbf{J}} + \underline{\mathbf{J}}^T\right) \tag{3.18}$$

Durch den Rotationstensor wird die Rotation eines Differenzvektors beschrieben. Er ist definiert durch

$$\underline{\mathbf{\Omega}} = \frac{1}{2}\left(\underline{\mathbf{J}} - \underline{\mathbf{J}}^T\right) \tag{3.19}$$

Nach Bestimmung der Eigenwerte werden diese der Größe nach geordnet

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \tag{3.20}$$

Das Lambda2-Kriterium definiert einen Wirbel (Vortex) als eine zusammenhängende Region, in der zwei Eigenwerte negativ sind. Oder anders ausgedrückt λ_2 kleiner null ist. Dadurch ergibt sich mehr ein verschwommener Bereich für einen Wirbel anstatt einer binären Entscheidung. Wird die Bestimmung des Lambda2-Werts für das gesamte Vektorfeld durchgeführt und positive Werte nicht berücksichtigt ergibt sich ein Skalarfeld. Die Wahl eines geeigneten Isowerts für die Extraktion der Wirbel ist jedoch sehr intuitiv.

4 Modellierung des Rauschens

Das Ergebnis der LIC-Berechnung hängt wesentlich vom verwendeten Rauschen ab. So beeinflussen beispielsweise die Raumfrequenzen des Rauschens die Anzahl der Stromlinien. Höhere Frequenzen führen zu mehr und feineren, niedrige Frequenzen zu weniger und breiteren Stromlinien. Abbildung 4.1 verdeutlicht dies. Cabral und Leedom [4] verwenden in Ihrer Arbeit weißes Rauschen, das dieselbe Dimension wie das Vektorfeld besitzt. Erst seit dem Ansatz von Stalling und Hege [53] ist es möglich, für das Rauschen eine Auflösung zu wählen, die sich von der des Vektorfelds unterscheidet. Dies erlaubt eine Anpassung der Raumfrequenzen ohne vorher einen Filter auf das Rauschen anwenden zu müssen. Obwohl der Schwerpunkt dieser Arbeit auf dreidimensionalem LIC liegt und ein Rauschvolumen eingesetzt wird, wird im folgenden nur auf 2D-Rauschtexturen eingegangen. Der dreidimensionale Fall ergibt sich, wenn nicht anders vermerkt, einfach durch Erweiterung der Dimensionalität.

Weißes Rauschen ist jedoch für die gleichzeitige Darstellung von Richtung und Orientierung des Vektorfelds mittels LIC weniger geeignet. Wegenkittel et al. [60] verwenden für den orientierten LIC mit asymmetrischem Filterkernel statt des weißen Rauschens ein dünnes Rauschen. Das von ihnen verwendete dünne Rauschen ist regelmäßig aufgebaut, wobei die Abtastpunkte auch minimal verschoben sein können (Jitter). Bei dem 3D LIC von Interrante und Grosch [19] werden die besten Ergebnisse mit in einem Volumen zufällig verteilten Punkten erzielt. Diese Punkte sollten mit einer Poissonverteilung oder einer Approximation der Poissonverteilung im Volumen platziert werden, um eine Gleichverteilung zu erhalten.

Um bei Verwendung von dünnem Rauschen die Breite der Stromlinien zu beeinflussen, gibt es zwei Möglichkeiten. So genügt es, die Auflösung der Rauschtextur und gleichzeitig auch die Zahl der Samples zu reduzieren. Dadurch vergrößert sich im Bildraum der Querschnitt jedes Samples, wodurch mehr Stromlinien durch jedes Sample verlaufen. Jedoch können sich trotz linearer Texturfilterung an den Rändern der breiteren Stromlinien harte Kanten bilden. Wird nun die Intensität der Stromlinien mittels einer Transferfunktion auf die Opazität abgebildet, so ist es kaum möglich, das Innere eines solchen Stromlinienbündels opak darzustellen, während der Rand transparent ist. Dies rührt daher, dass aufgrund der Skalierung des Samples

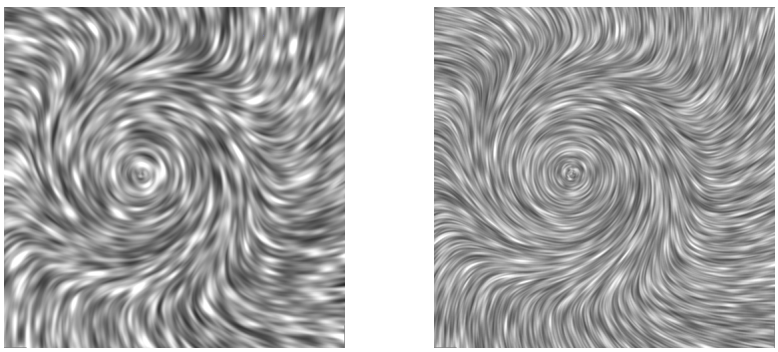


Abbildung 4.1: Einfluss der Raumfrequenzen der Rauschtextur auf das Resultat der LIC-Berechnung. Links niedrige, rechts hohe Raumfrequenzen. Die Parameter des LIC wurden nicht verändert.

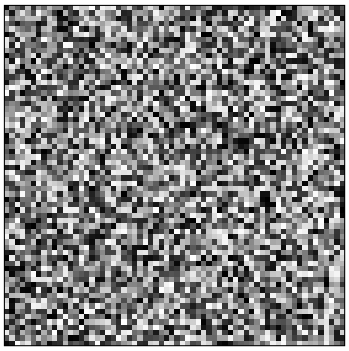


Abbildung 4.2: Weißes Rauschen erzeugt durch Zufallszahlen zwischen null und eins.

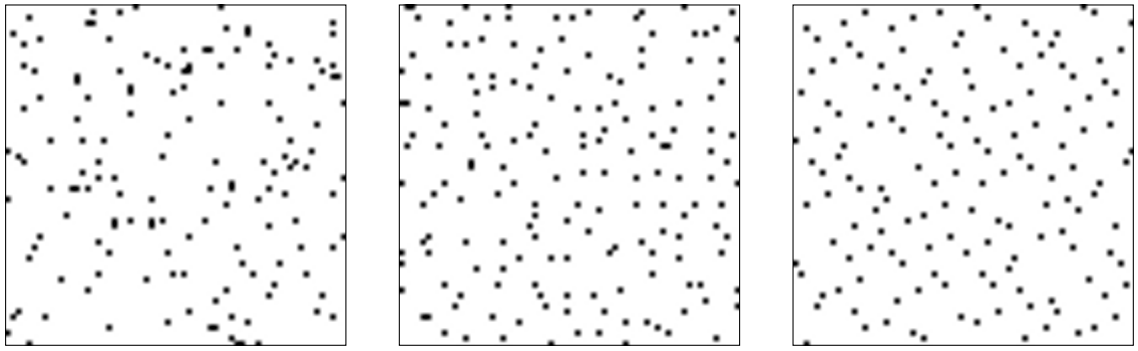
sämtliche Stromlinien eine ähnliche Intensität besitzen. Dieses Problem lässt sich umgehen, indem nicht die Auflösung der Rauschtextur verändert wird, sondern das Rauschen mit einem Tiefpassfilter gefiltert wird. Dadurch vergrößert sich ebenfalls der Querschnitt der Samples. Gleichzeitig wird aber die Intensität eines einzelnen Samples mit zunehmendem Abstand zur ursprünglichen Abtastposition geringer. Wird der LIC auf dieses gefilterte Rauschen angewendet, so ergeben sie Stromlinienbündel, die am Rand eine recht geringe Intensität besitzen, die sich zum Mittelpunkt hin erhöht. Da die Gradienten dieses Rauschens bei der Berechnung der Beleuchtung als Normalen verwendet werden können (siehe Abschnitt 3.4.3), wirkt sich so die Filterung indirekt auch auf die Beleuchtung aus. Dies hat zur Folge, dass die Gradienten einen wesentlich glatteren Verlauf aufweisen. Hiervon profitiert die Beleuchtung mittels Gradienten.

4.1 Weißes Rauschen

Weißes Rauschen lässt sich sehr einfach mit Zufallszahlen erzeugen. Dazu werden für jedes Texel der Rauschtextur Zufallszahlen zwischen null und eins erzeugt. Im Mittel ergibt sich so ein Wert von 0,5. Abbildung 4.2 zeigt ein auf diese Weise erzeugtes Rauschen. Weißes Rauschen kann beim zweidimensionalen LIC als Ausgangstextur verwendet werden [4][53]. Beim OLIC [60] und beim 3D LIC [19] wird zur besseren Visualisierung jedoch dünnes Rauschen eingesetzt.

4.2 Dünnes Rauschen

Für die Erzeugung von dünnem Rauschen wird die gesamte Rauschtextur mit null initialisiert. Anschließend werden zufällige Positionen auf eins gesetzt. Die Anzahl der Abtastpunkte sollte klein gegenüber der Gesamtzahl aller möglichen Positionen sein. Der Erwartungswert befindet sich dementsprechend im Intervall von null bis eins. Durch die Verwendung von Pseudozufallszahlen können Häufungen mehrerer Samples ebenso wie größere leere Bereiche auftreten (vgl. Abbildung 4.3(a)). Um die Uniformität beziehungsweise die Diskrepanz der Abtastungen zu erhöhen, müsste eine Poissonverteilung verwendet werden, die jedoch sehr



(a) (b) (c)

Abbildung 4.3: Gleichverteilung von 150 Abtastpunkten zur Erzeugung von dünnem Rauschen. Verwendung von (a) Pseudozufallszahlen, (b) Stratified Sampling, (c) Haltonsequenz.

aufwendig zu berechnen ist. Stattdessen können verschiedene Approximationen der Poissonverteilung zur Verbesserung dieser Uniformität eingesetzt werden.

Die Diskrepanz ist optimal, wenn ein reguläres Gitter für die Ausgangspositionen der Abtastpunkte verwendet wird. Die Abtastpunkte können anschließend relativ zu ihrer ursprünglichen Position mit Zufallszahlen minimal verschoben (Jitter). Jedoch sind dann die Samples sehr regelmäßig verteilt. Weitere Möglichkeiten zur Erhöhung der Diskrepanz der Abtastungen bieten die Stratifikation und das Latin-Hypercube-Sampling [25]. Bei der Stratifikation wird die Rauschtextur in mehrere Strata unterteilt und diese jeweils mit n Punkten gefüllt (Stratified Sampling). Die Rauschtextur wird beim Latin-Hypercube-Sampling ebenfalls in Strata unterteilt. Jedoch darf höchstens ein Sample pro Spalte und Zeile auftreten. Realisiert wird diese Auswahl der Strata durch zufällige Permutationen.

Die Uniformität kann auch durch die Verwendung von Sequenzen geringer Diskrepanz (Low Discrepancy Sequences), wie zum Beispiel der Haltonsequenz, verbessert werden [23][24]. Zur Erzeugung der Haltonsequenz wird die Radixinvertierung verwendet [25]. Die Radixinvertierung zur Basis b der Zahl i ist definiert durch

$$\varphi_b(i) = \sum_{j=0}^{\infty} a_j(i) b^{-j-1} \tag{4.1}$$

Dabei bezeichnet $a_j(i)$ die Stelle j von i. Für i aus $\mathbf{N}_0$ ergibt sich daraus die van der Corputsequenz der Basis b. Bei einer n-dimensionalen Haltonsequenz wird für jede Dimension eine van der Corputsequenz verwendet. Die Position eines Samples $\mathbf{p}$ ergibt sich dann aus

$$\mathbf{p}(i) = (\varphi_{b1}(i), \varphi_{b2}(i), \ldots, \varphi_{bn}(i)) \tag{4.2}$$

Um eine Gleichverteilung zu erhalten, sollten die Basen $b_1, \ldots, b_n$ aus aufsteigenden Primzahlen bestehen. In Abbildung 4.3 sind einige Beispiele für eine Gleichverteilung dargestellt.

4.3 Filterung

Die Filterung eines eindimensionalen Signals entspricht der Faltung des Signals mit dem Filter. Sei $h(t)$ das Signal und $g(t)$ der Filter, dann ist die Faltung definiert durch

$$(g * h)(t) = \int_{-\infty}^{\infty} g(\tau) \cdot h(t - \tau) \mathrm{d}\tau \tag{4.3}$$

Das Faltungstheorem besagt, dass die Faltung einer Multiplikation im Fourierraum entspricht:

$$(g * h)(t) \Longleftrightarrow G(\nu) \cdot H(\nu) \tag{4.4}$$

$G(\nu)$ und $H(\nu)$ sind die entsprechenden Funktionen im Fourierraum. Damit lässt sich für Faltungen der Aufwand reduzieren. Dazu muss eine Fouriertransformation in den Frequenzraum und anschließend die inverse Fouriertransformation zurück in den Bildraum durchgeführt werden. Diese Methode ist schneller als die Filterung im Bildraum, wenn der Filterkern eine große Ausdehnung oder sogar unendlichen Einfluss besitzt.

Sei $h(t)$ eine Funktion im Bildraum, dann ist $H(\nu)$ die analoge Repräsentation im Frequenzraum. Für die Fouriertransformation gilt

$$H(\nu) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} h(t) \mathrm{e}^{-i\nu t} \,\mathrm{d}\, t \tag{4.5}$$

und für die Rücktransformation

$$h(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} H(\nu) \mathrm{e}^{i\nu t} \,\mathrm{d}\, \nu \tag{4.6}$$

Ist das Signal mehrdimensional, so erfolgt die Faltung nacheinander entlang jeder Dimension. Ebenso wird mit der Fouriertransformation verfahren.

Um bei einem Signal die hohen Frequenzen zu eliminieren, wird ein Tiefpassfilter eingesetzt. Ein idealer Filter dafür ist der sinc-Filter mit

$$h(t) = \Delta\nu \cdot \mathrm{sinc}(\pi \Delta\nu t) \quad \text{mit} \, \mathrm{sinc}(x) = \frac{\sin(x)}{x} \tag{4.7}$$

Dabei ist $\Delta\nu/2$ die maximale Frequenz des Filters. Wird der sinc-Filter in den Fourierraum überführt ergibt sich ein Boxfilter

$$H(\nu) = \begin{cases} 1 & -\frac{\Delta\nu}{2} \leq \nu \leq \frac{\Delta\nu}{2} \\ 0 & \text{sonst} \end{cases} \tag{4.8}$$

Soll nun ein Signal gefiltert werden, wird es mittels Fouriertransformation in den Frequenzraum überführt. Dort werden alle Frequenzen größer $\Delta\nu/2$ auf null gesetzt. Anschließend wird das Signal mit der inversen Fouriertransformation zurücktransformiert.

Dieser Tiefpassfilter eignet sich für weißes, nicht jedoch für dünnes Rauschen. Da die hohen Frequenzen komplett abgeschnitten werden, kommt es zu Oberschwingungen, die nicht mehr

ausgeglichen werden. Diese Oberschwingungen sind aufgrund der geringen Intensität im weißen Rauschen kaum bemerkbar, jedoch im dünnen Rauschen deutlich sichtbar (Abbildung 4.4 mittlere Spalte). Um diese Problematik zu umgehen, kann ein Filter verwendet werden, der im Frequenzraum mit zunehmender Frequenz an Einfluss verliert. Hierfür bietet sich der Gaußfilter an:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \mathrm{e}^{-x^2/\left(2\sigma^2\right)} \tag{4.9}$$

Wird der Gaußfilter in den Frequenzraum transformiert, so ergibt sich wieder eine Gaußkurve mit

$$F(\nu) = \frac{1}{\sqrt{2\pi}} \cdot \mathrm{e}^{-\sigma^2\nu^2/2} \tag{4.10}$$

Bei der Filterung mit diesem Filter wird das Signal entsprechend dem Faltungstheorem (Gleichung (4.4)) im Frequenzraum nur mit dem Gaußfilter multipliziert. Da höhere Frequenzen immer geringer gewichtet werden, können keine Oberschwingungen mehr auftreten. Abbildung 4.4 zeigt die Auswirkungen der Filterung mit einem Box- und einem Gaußfilter auf dünnes Rauschen und auf weißes Rauschen. Die Zwischenergebnisse der Filterung im Frequenzraum sind in der letzten Zeile abgebildet.

4.4 Konstante Raumfrequenz von Rauschen im Bildraum

Sind die Raumfrequenzen im Bildraum zu hoch, kann es zu Aliasingeffekten kommen. Dies kann sich in Flimmern oder harten Übergängen äußern. Wird bei einer perspektivischen Kamera der Abstand zwischen dem Vektorfeld und der Augposition verringert, so nehmen die Raumfrequenzen ab, da das Vektorfeld insgesamt mehr Raum im Viewport einnimmt. Dagegen nehmen die Raumfrequenzen zu, wenn sich das Vektorfeld vom Auge entfernt. Um zu verhindern, dass sich die Raumfrequenzen in diesem Umfang ändern, muss die Distanz zwischen Vektorfeld und Auge mit in Betracht gezogen werden. Ein möglicher Ansatz soll hier vorgestellt werden.

Die Idee besteht darin, dass der Texturnachschlag in Texturen verschiedener Auflösung durchgeführt wird. Die Auflösung der Texturen soll dabei abhängig von der Entfernung zum Auge sein. Da eine perspektivische Kamera verwendet wird, muss diese Entfernung in Gerätekoordinaten oder in Bildkoordinaten berechnet werden. In Bildkoordinaten entspricht die Entfernung zwischen Auge und der Position genau dem z-Wert der Position (nach der Homogenisierung). Da der Verlauf der Tiefenwerte nicht linear ist, wird zur Approximation die Potenz zur Basis zwei verwendet. Um die nächstkleinere Texturstufe m für einen Tiefenwert d zu berechnen, wird nur der ganzzahlige Teil der Potenz verwendet, der sich aus

$$m = \lfloor \mathrm{ld}\,(d+1) \rfloor \tag{4.11}$$

ergibt. Wird nun aus den Texturen für Stufe m und $m+1$ jeweils die Intensität des Rauschens ermittelt, kann mit linearer Interpolation die Intensität eines neuen Rauschens berechnet werden. Dieses Rauschen sollte, unabhängig vom Augabstand, eine näherungsweise konstante Raumfrequenz im Bildraum besitzen.

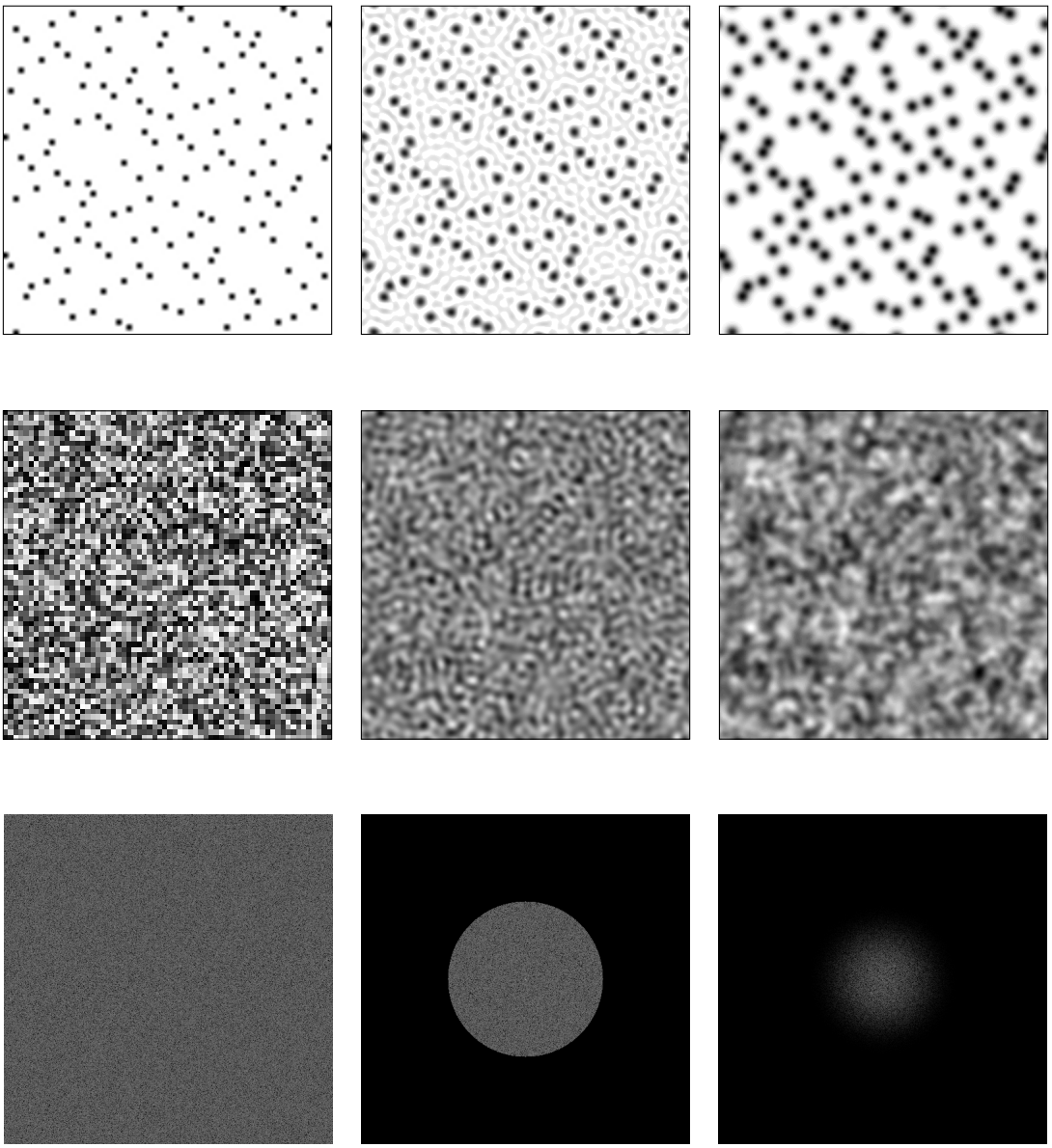


Abbildung 4.4: Verwendung eines Box- und eines Gaußfilters zur Filterung der niedrigen Frequenzen. In den ersten beiden Zeilen ist das ursprüngliche Rauschen, das Ergebnis nach Anwendung des Boxfilters und nach Anwendung des Gaußfilters von links nach rechts abgebildet. In der dritten Zeile ist das Resultat der Filterung im Frequenzraum dargestellt.

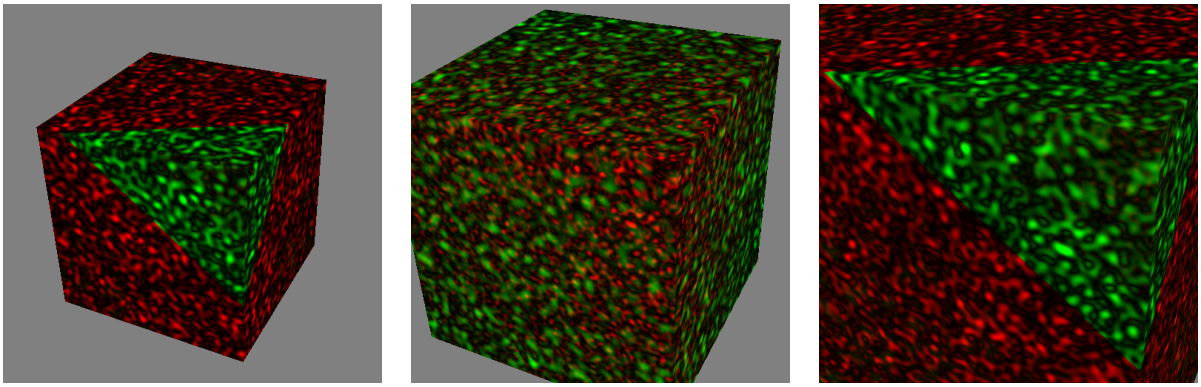


Abbildung 4.5: Konstante Raumfrequenz des Rauschens im Bildraum. Das Rauschen der Stufe m ist rot und der Stufe $m+1$ grün eingefärbt. Der harte Übergang im linken und rechten Bild tritt auf, da der Interpolationskoeffizient von eins auf null zurückspringt. Im mittleren Bild ist deutlich die Mischung von niederfrequentem (grün) und hochfrequentem Rauschen (rot) zu erkennen.Der Abstand der Kamera wird von links nach rechts verringert.

Anstatt viele Rauschtexturen unterschiedlicher Auflösung, vergleichbar mit Mip-Mapping, anzulegen, werden nur die Texturkoordinaten im Fragmentprogramm entsprechend skaliert und eine einzige Rauschtextur verwendet. So muss beispielsweise, um Rauschen mit höherer Frequenz zu erhalten, die Texturkoordinate mit einem Faktor größer eins multipliziert werden, was zu Undersampling führt. Eine Skalierung kleiner eins führt dagegen bei Verwendung von trilinearer Texturfilterung zu Oversampling und somit zu niedrigeren Frequenzen. Voraussetzung dafür ist jedoch, dass beim Texturzugriff der ganzzahlige Anteil der Texturkoordinaten ignoriert wird, was in OpenGL dem Verhalten von GL_REPEAT entspricht (siehe [50]). Die Berechnung dieses Skalierungsfaktors f_m ergibt sich aus

$$\begin{aligned} f_m &= s \cdot 2^{-m} \\ &= s \cdot 2^{-\lfloor \mathrm{ld}(d+1) \rfloor} \end{aligned} \tag{4.12}$$

Dabei lässt sich mit s die Raumfrequenz des Bildraums justieren. Mit f_m und f_{m+1} lassen sich nun die Texturkoordinaten für die Texturstufen m und $m+1$ berechnen, indem die ursprüngliche Koordinate mit f_m beziehungsweise f_{m+1} multipliziert wird. Anschließend wird mit diesen Texturkoordinaten jeweils in derselben Rauschtextur nachgeschlagen. Sei n_m das Ergebnis dieses Nachschlags für Auflösung m und n_{m+1} für $m+1$, dann ergibt sich für das Rauschen mit konstanter Raumfrequenz n_{const} mittels linearer Interpolation

$$\begin{aligned} t &= \mathrm{ld}\,(d+1) - \lfloor \mathrm{ld}\,(d+1) \rfloor \\ n_{const} &= t n_{m+1} + (1-t) n_m \end{aligned} \tag{4.13}$$

In Abbildung 4.5 ist eine solche Textur mit konstanter Raumfrequenz dargestellt. Der Abstand der Kamera nimmt dabei von links nach rechts ab. Zur Hervorhebung der unterschiedlichen Texturstufen wurde die Stufe m rot und Stufe $m+1$ grün eingefärbt. Die harte Kante zwischen dem roten und dem grünen Rauschen im ersten und dritten Bild tritt auf, weil hier aufgrund der Entfernung zur Kamera $m = m+1$ für verschiedene d gilt.

5 GPU-basierter 3D LIC

Im Folgenden wird das Ziel dieser Arbeit, die Berechnung des dreidimensionalen LIC auf der GPU, erläutert. Obwohl der LIC zu den globalen Verfahren der Vektorfeldvisualisierungen zählt, findet die Auswertung lokal beschränkt entlang der Stromlinien statt. Diese Auswertung wird für jedes Pixel durchgeführt, wodurch sich ein globaler Effekt einstellt. Die einzige Ausnahme bildet der FastLIC von Stalling und Hege [53]. Da die Berechnungen der einzelnen Pixel nur abhängig vom Vektorfeld und der eingesetzten Rauschtextur sind, lässt sich die Auswertung des LIC sehr gut parallelisieren. Durch die SIMD-Archtitektur (*Single Instruction Multiple Data* ist heutige Grafikhardware hierfür geradezu prädestiniert. Werden dazu das Vektorfeld und die Rauschtextur auf der GPU durch 3D-Texturen repräsentiert und die Berechnung des LIC im Fragmentshader durchgeführt, können abhängig von der Anzahl der Pixelpipelines der GPU bis zu 128 Pixel beziehungsweise Fragmente (nVidia GeForce 8800GTX) gleichzeitig ausgewertet werden. Die Traversierung des Volumens kann dabei auf unterschiedliche Weisen erfolgen. Entsprechend der Aufgabenstellung wird hierfür das Ray-Casting verwendet. Darüber hinaus wurde in dieser Arbeit auch ein vielversprechender Ansatz mittels Slicing verfolgt. Beide Verfahren werden in der Volumenvisualisierung eingesetzt (siehe Kapitel 2). Grundlage für das Ray-Casting auf der GPU bilden die Arbeiten von Carr et al. [5] und von Purcell et al. [44] aus dem Jahr 2002. In diesen Publikationen werden GPU-basierte Implementierungen von Ray-Tracing vorgestellt. Als Ausgangsbasis für diese Arbeit diente dabei das Grundgerüst des Volumenrenderers für GPU-basiertes Ray-Casting von Stegmaier et al. [54].

In Abbildung 5.1 ist das Flussdiagramm des Algorithmus dargestellt. Im ersten Schritt wird das zu visualisierende Vektorfeld aus dem Datensatz geladen. Damit auf das Vektorfeld auf der GPU zugegriffen werden kann, muss dieses in einer 3D-Textur gespeichert werden. Dazu wird eine Textur mit vier Kanälen verwendet. Als Datentyp der Textur werden vorzeichenlose Bytes eingesetzt. Auf die Verwendung von 16 Bit-Gleitkommazahlen wurde bewusst wegen des größeren Speicherbedarfs verzichtet. Die Richtung des Vektorfelds wird in den ersten drei Kanälen gespeichert. Besitzt der Datensatz in einer Richtung nicht die Größe einer Zweierpotenz, so werden die Daten in dieser Richtung nicht interpoliert und die nicht mit Daten

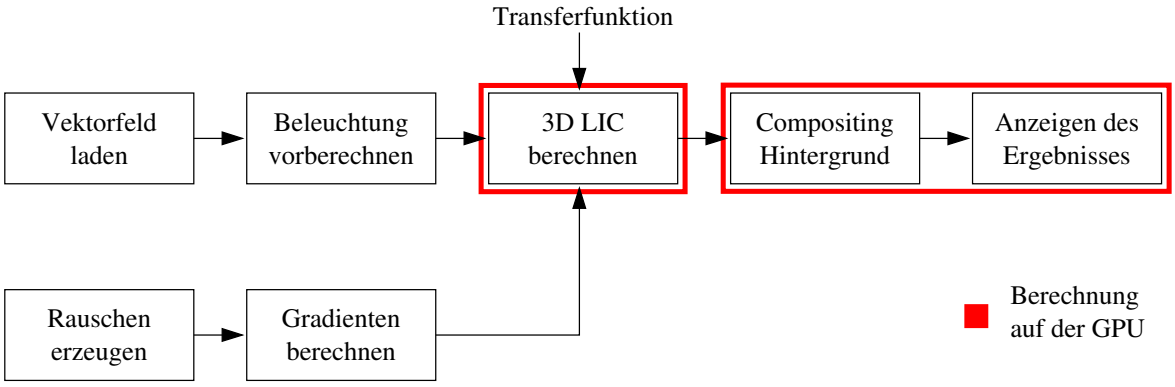


Abbildung 5.1: Architektur zur Berechnung des 3D LIC auf der GPU.

gefüllten Texel mit Null aufgefüllt. Im Fragmentprogramm werden bei Zugriffen auf diese Textur die Texturkoordinaten so skaliert, dass nur auf den Datenbereich zugegriffen wird. Da nur acht Bit pro Kanal zu Verfügung stehen, müssen die Vektoren quantisiert und verschoben werden. Durch die Quantisierung können sich beim Texturzugriff und trilinearer Filterung auf der GPU Artefakte ergeben. Um dies zu umgehen, wird die Richtung der Vektoren normalisiert und der Betrag jeden Vektors in der vierten Komponente des Vektorfelds gespeichert. Anschließend werden die für die Beleuchtung erforderlichen Vorberechnungen durchgeführt. Dazu zählt die Erstellung der Texturen für die Modelle von Zöckler et al. [69] und Mallo et al. [38] (Abschnitt 3.4).

Für die Berechnung des LIC wird eine Rauschtextur benötigt, die zu Beginn erzeugt wird. Dabei kann es sich um reines weißes Rauschen oder auch dünnes Rauschen handeln (siehe Kapitel 4). Dieses Rauschen kann anschließend zur Verringerung der hohen Raumfrequenzen noch mit einem Tiefpass gefitert werden. Im nachfolgenden Schritt werden die Gradienten des Rauschens berechnet, da diese für die Beleuchtung mittels Gradienten benötigt werden (Abschnitt 3.4.3). Gradienten und die Intensität des Rauschens werden wie auch das Vektorfeld in einer 3D-Textur mit vier Komponenten gespeichert. Das aufwendige Erzeugen des Rauschens und auch die Berechnung der Gradienten kann in eine Anwendung ausgelagert werden. Dadurch muss das Rauschen für den LIC nur geladen und in die 3D-Textur geschrieben werden.

Nach diesen Vorbereitungsschritten kann nun die Berechnung des LIC auf der GPU erfolgen. Dabei hat die verwendete Transferfunktion wesentlichen Einfluss auf das Ergebnis (vgl. Abschnitt 3.4.4). Im nachfolgenden Abschnitt wird diese Berechnung, die im Fragmentprogramm stattfindet, näher erläutert. Das Ergebnis wird dann mittels Compositing mit dem Hintergrund kombiniert und kann anschließend angezeigt werden.

5.1 Berechnung des 3D LIC

Für die Implementierung des 3D LIC auf der GPU wurden das Ray-Casting und das Slicing als separate, austauschbare Techniken implementiert. Um bei Verwendung des Ray-Castings vom Effekt des Empty-Space-Leaping zu profitieren wurde eine Technik namens *Depth Peeling* eingesetzt. Diese Technik wird das zum korrekten Rendern transparenter Szenen eingesetzt [11]. Dabei wird die Szene durch mehrfaches Rendern in mehrere Schichten unterschiedlicher Tiefe unterteilt. Beim nachfolgenden Rendern der Szene werden diese Tiefenschichten dazu verwendet, die Szene mit korrekter Tiefensortierung, die bei transparenten Objekten sehr wichtig ist, darzustellen.

5.1.1 LIC-Berechnung im Fragmentprogramm

Die Berechnung des LIC erfolgt auf der GPU entsprechend Gleichung (3.3). Dazu werden zwei nacheinander ausgeführte Schleifen mit je L Durchläufen verwendet. In der ersten Schleife wird der positive Teil des LIC-Integrals durch eine Diskretisierung gelöst, in der zweiten der negative Teil. Das Verfolgen der Stromlinie erfolgt in Richtung des aus dem Vektorfelds ermittelten Vektors. Entlang der Stromlinie wird für jeden Abtastpunkt ein Texturnachschlag in der Rauschtextur durchgeführt und die Intensität des Rauschen mit dem Filter entsprechend gewichtet. Die ermittelten Intensitäten werden aufsummiert und anschließend normalisiert. Dies

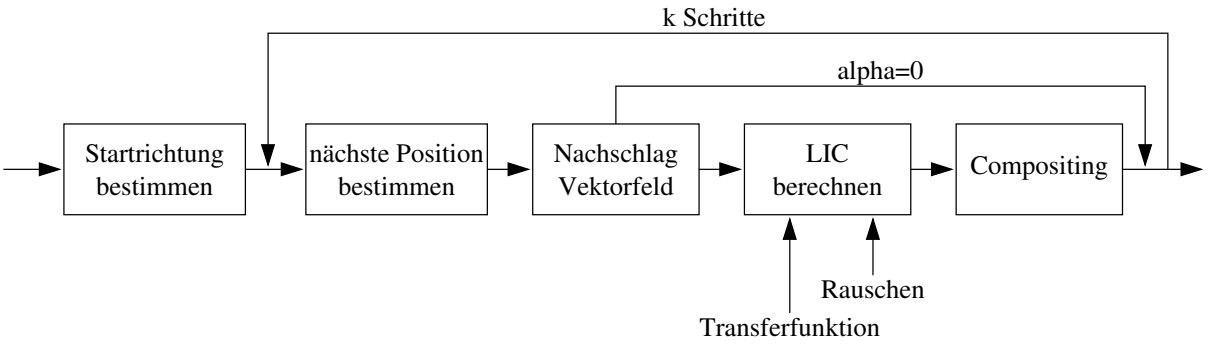


Abbildung 5.2: Schema des Fragmentprogramms für die Berechnung des 3D LIC mittels Ray-Casting

ergibt den Dichtewert des LIC an der aktuellen Samplingposition auf dem Strahl. Die Dichte kann anschließend in der Beleuchtungsberechnung weiter verwendet oder direkt visualisiert werden.

5.1.2 Ray-Casting

Beim Ray-Casting werden die Traversierung der Strahlen und die Berechnung des LIC in demselben Fragmentprogramm durchgeführt. Abbildung 5.2 verdeutlicht dazu den Aufbau des Fragmentprogramms. Zu Beginn wird für jedes Fragment mit Hilfe der Startposition des Strahls und der Kameraposition die Richtung des Strahls bestimmt. Danach kann mit dem Abtasten entlang des Strahls begonnen werden. Dazu wird die Position in Strahlrichtung um die Samplingdistanz verschoben. Aus der 3D-Textur des Vektorfelds ergibt sich an dieser neuen Position mit trilinearer Filterung die Richtung und der Betrag des Vektors. Soll die Geschwindigkeit mit der Transferfunktion auf eine Farbe und einen Alphawert abgebildet werden, wird die Transferfunktion auf den Betrag des Vektors angewendet. Ist der berechnete Alphawert gleich null, kann zum nächsten Abtastpunkt weitergegangen werden. Anderfalls wird die Richtung des Vektors und die aktuelle Position im Volumen für die LIC-Berechnung verwendet (Abschnitt 5.1.1). Das Ergebnis der LIC-Berechnung wird danach mit Front-To-Back-Compositing mit den Ergebnissen der vorherigen Strahlpositionen geblendet. Wurden k Schritte durchgeführt, das Volumen verlassen oder überschreitet die Opazität einen Schwellwert (vorzeitiger Strahlabbruch), kann die weitere Berechnung abgebrochen werden. Die Anzahl der Schritte ist dabei beschränkt durch die maximale Anzahl der Instruktionen pro Fragmentprogramm.

Damit dieses Fragmentprogramm ausschließlich für das Volumen ausgeführt wird, werden nur die sichtbaren Flächen der Boundingbox des Volumens gezeichnet. Da die Abtastung entlang der Strahlen erst auf der Oberfläche des Volumens beginnen soll, wird an den Vertizes der Boundingbox die Objektraumposition in eine Texturkoordinate geschrieben. Durch die Interpolation durch die Rasterisierung ergibt sich so für jedes Fragment eine Position auf der Oberfläche der Boundingbox in Objektkoordinaten, auf die im Fragmentprogramm zurückgegriffen werden kann.

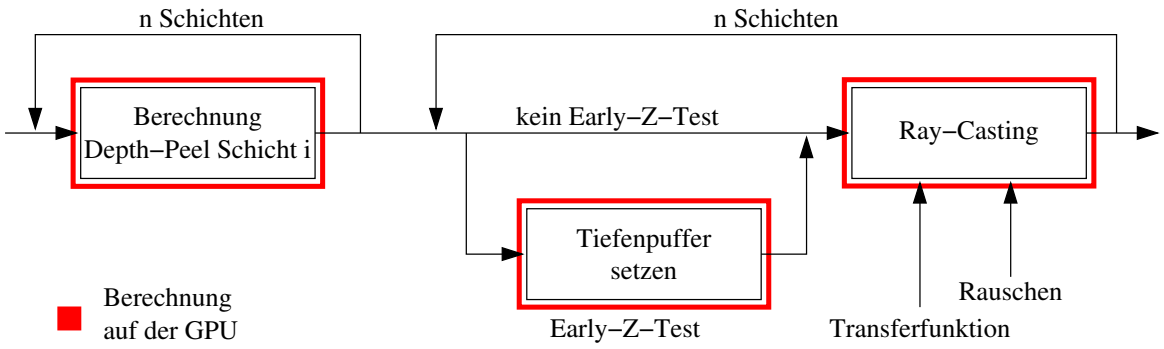


Abbildung 5.3: Ablauf des Ray-Castings mit gleichzeitigem Einsatz von Depth-Peeling.

5.1.3 Ray-Casting mit Depth-Peeling

Das Konzept des Depth-Peeling [11] wurde mit dem Ray-Casting kombiniert, um das Überspringen von leeren Voxeln zu realisieren und die maximale Anzahl der Abtastungen pro Strahl gegenüber dem Ray-Casting ohne Depth-Peeling zu erhöhen. Letzteres gelingt dadurch, weil es sich beim Ray-Casting mit Depth-Peeling um ein Multi-Pass-Verfahren handelt. Dabei wird das Ray-Casting abgesehen von einer kleinen Änderung wie bisher eingesetzt.

Der Ablauf des Algorithmus ist in Abbildung 5.3 dargestellt. In einem ersten Schritt wird das Volumen in mehrere Depth-Peel-Schichten unterteilt. Dazu wird ein Fragmentprogramm verwendet, welches denselben Aufbau besitzt wie das Programm für das Ray-Casting. Die maximale Anzahl an Schritten n pro Depth-Peel-Schicht wird vorab auf festgelegt und beeinflusst zusammen mit der Samplingdistanz die Anzahl der Schichten.

Zur Berechnung der einzelnen Depth-Peel-Schichten wird in diesem Fragmentprogramm nur die Traversierung des Strahls und keine LIC-Berechnung durchgeführt. Während dieser Traversierung wird in der 3D-Textur des Vektorfelds an jedem Abtastpunkt ein Texturnachschlag ausgeführt und die Transferfunktion auf die Geschwindigkeit, die sich in der vierten Komponte befindet, angewandt. Ist der abgebildete Alphawert gleich null, wird mit der nächsten Position auf dem Strahl fortgefahren. Bei einem Wert größer null wird die Ausführung des Ray-Casting abgebrochen und die Anzahl der im leeren Raum zurückgelegten Schritte als Ausgabe des Fragmentprogramms verwendet. Zur Bestimmung der ersten Schicht wird wie beim Ray-Casting an der Oberfläche des Volumens begonnen. Bei den darauffolgenden Schichten wird die Startposition um $i \cot n$ Schritte entlang des Strahls verschoben, wenn i der Index der aktuell zu berechnenden Schicht ist. Die Ergebnisse werden für jede Schicht getrennt in eine 2D-Textur geschrieben, die den gesamten Viewport füllt.

Für jede dieser Schichten wird anschließend ein Ray-Casting mit n Schritten durchgeführt. Dabei werden die Startpositionen wie schon bei der Berechnung der Depth-Peel-Schichten an den Beginn der jeweiligen Schicht verschoben. Im Fragmentprogramm wird zusätzlich an der Fragmentposition in der 2D-Textur der aktuellen Depth-Peel-Schicht die Anzahl der Schritte im leeren Raum nachgeschlagen und die Startposition entsprechend angepasst. Dabei werden für die Fragmentposition Texturkoordinaten im Bildraum verwendet. Anschließend wird das Ray-Casting für die verbleibenden Schritte durchgeführt.

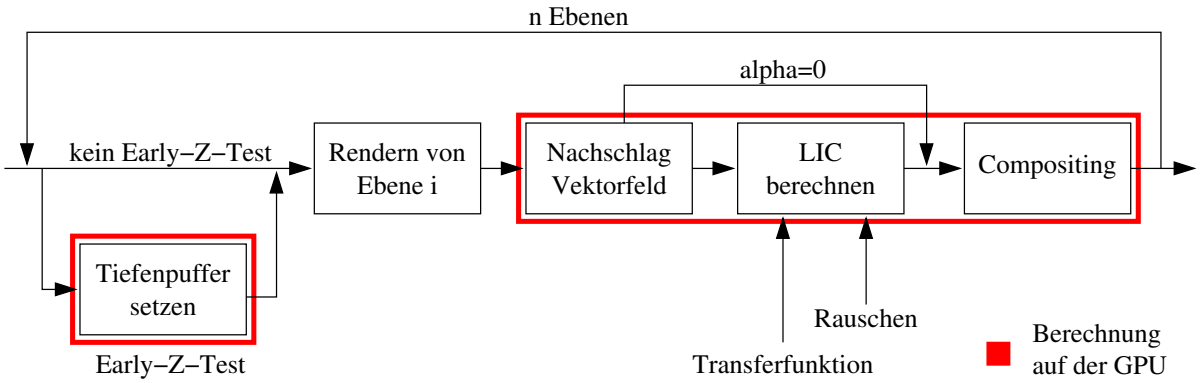


Abbildung 5.4: Ablauf des Slicings für n Ebenen.

Zur Verwendung des Early-Z-Tests wird vor dem Durchführen des Ray-Castings für jede Schicht der Tiefenpuffer entsprechend vorbereitet. Wird der Early-Z-Test eingesetzt, kann die Textur der jeweiligen Depth-Peel-Schicht dazu verwendet werden, komplett leere Bereiche von der LIC-Berechnung auszusparen. Da mehrere Renderdurchläufe benötigt werden, wird während des Ray-Castings und der Vorbereitung des Tiefenpuffers der vorzeitige Strahlabbruch durchgeführt. Dadurch wird verhindert, dass die nachfolgenden Schichten eines bereits abgebrochenen Strahls weiterberechnet werden.

In Abbildung 5.5 sind auf der linken Seite die Texturen für die ersten vier Depth-Peel-Schichten und auf der rechten Seite das jeweilige Ergebnis des LIC, bestehend aus den vorangegangenen Schichten, dargestellt. Die Rotschattierungen geben die Anzahl der Schritte im leeren Raum wieder. Je heller das Rot ist, desto mehr Schritte können übersprungen werden. Grün eingefärbte Bereiche deuten darauf hin, dass die Boundingbox verlassen wurde ohne auf ein opakes Voxel gestoßen zu sein.

5.1.4 Slicing

Es wurde für das Slicing die Variante gewählt, bei der die Ebenen an der Kamera ausgerichtet werden (View-Aligned-Slicing). Die Anzahl der Ebenen n ist abhängig von der Samplingdistanz. Deshalb wird zu Beginn die Distanz entlang der Blickrichtung zwischen der Vorderseite und der Rückseite des Volumens berechnet. n ergibt sich aus dem Verhältnis dieser Distanz zur Samplingdistanz. In Abbildung 5.4 ist die LIC-Berechnung für n Ebenen dargestellt.

Wird der Early-Z-Test zur Beschleunigung verwendet, muss der Tiefenpuffer entsprechend modifiziert werden. Dazu wird dieser aufgrund des Framebufferinhalts des letzten Durchlaufs vorbereitet. Ist die Opazität eines Fragments größer als der Schwellenwert des vorzeitigen Strahlabbruchs, wird die Tiefe auf null gesetzt. Dadurch schlägt beim Rendern der nächsten Ebene der Tiefentest für diese Fragmente fehl und das Fragmentprogramm wird nicht ausgeführt. Der Einsatz des Early-Z-Test ist dringend zu empfehlen, da im Optimalfall einer größtenteils opaken Transferfunktion die LIC-Berechnung nur für wenige Fragemente ausgeführt werden muss (vgl. 6.2).

Die Rendern der aktuellen Ebene i erfolgt nach deren Schnittberechnung mit dem Volumen. Diese Schnittpunkte bilden nach einer Sortierung einen Fächer aus Dreiecken. Beim Rendern dieses Fächers werden die Positionen der Vertizes als Texturkoordinaten verwendet. Mit dieser Position aus den Texturkoordinaten kann im Fragmentprogramm in der 3D-Textur des Vektorfelds nachgeschlagen werden. Die Geschwindigkeit des Vektors kann wie beim Ray-Casting als Eingabeparameter der Transferfunktion verwendet werden. Die Berechnung des LIC erfolgt wie in Abschnitt 5.1.1 beschrieben im Fragmentprogramm. Das anschließende Compositing kann entweder durch das Hardware-Blending von OpenGL oder durch Front-To-Back-Blending im selben Fragmentprogramm erfolgen. Da der Einsatz des Early-Z-Tests den Inhalt des Framebuffers für den vorzeitigen Strahlabbruch benötigt, kann dieser für das Compositing im Fragmentprogramm wieder verwendet werden.

5.2 Beleuchtung

Für die Beleuchtung der Stromlinien werden nach der Berechnung des LIC im Fragmentprogramm für jeden Abtastpunkt die Lichtrichtung und die Blickrichtung berechnet. Bei Verwendung des Ray-Castings entspricht die Blickrichtung der umgekehrten Strahlrichtung, beim Slicing muss sie aus der Kameraposition und der gegenwärtigen Position ausgerechnet werden. Je nach verwendetem Beleuchtungsmodell erfolgen weitere Berechnungen (siehe Abschnitt 3.4). Wird die Beleuchtung mittels Gradienten gleichzeitig mit der Methode zur Erhaltung einer konstanten Raumfrequenz des Rauschens (Abschnitt 4.4) eingesetzt, können die Gradienten ebenfalls mit Gleichung (4.13) interpoliert werden. Jedoch muss der Gradient aus der Stufe m zuvor mit Faktor zwei multipliziert werden, um ihn an die höheren Frequenzen anzupassen.

Werden verschiedene Schrittweiten für die Traversierung des Volumens verwendet, ergeben sich für transparente Bereiche unterschiedliche Ergebnisse. Dies wird durch die unterschiedliche Anzahl der Abtastungen verursacht. Durch einen geringeren Abstand ergeben sich mehr Abtastungen und dies führt zu einem dunkleren Ergebnis. Um dies zu korrigieren kann eine Opazitätskorrektur (Opacity Correction) durchgeführt werden [31, 48]. Wurde die Opazität für die Samplingdistanz d_{old} festgelegt und ist d_{new} die neue Samplingdistanz, ergibt sich nach Lacroute und Levoy [31] der korrigierte Alphawert aus

$$\alpha_{korrigiert} = 1 - (1 - \alpha)^{d_{new}/d_{old}} \tag{5.1}$$

Diese Korrektur der Opazität lässt sich im Fragmentprogramm mit den folgenden drei Zeilen durchführen:

```
SUB tmp.a, 1, dst.a;
POW tmp.a, tmp.a, ratio.r;
SUB dst.a, 1, tmp.a;
```

`ratio.r` enthält dabei das Verhältnis d_{new}/d_{old}.

5.3 Features

Zur Extraktion von Features lassen sich verschiedene Techniken einsetzen (siehe Abschnitt 3.5). Das Velocity-Masking lässt sich sehr einfach umsetzen. Im Fragmentprogramm muss dazu nur die Geschwindigkeit des Vektors, für den der LIC berechnet wird, über die Transferfunktion auf den Alphawert abgebildet werden. Ist dieser Alphawert gleich null, kann die LIC-Berechnung für diesen Vektor übersprungen werden. Zur Hervorhebung von Wirbeln mit dem Lambda2-Verfahren wird anstelle der Geschwindigkeit der Skalarwert aus einer 3D-Textur, die das Lambda2-Volumen enthält, verwendet. Auf diesen Skalarwert kann anschließend die Transferfunktion angewandt werden, um die uninteressanten Bereiche auszublenden.

Für das Clipping werden die von OpenGL zur Verfügungen gestellten Hardware-Clip-Ebenen verwendet. Jedoch erfordert die Verwendung der programmierbaren Pipeline der GPU, dass die Entfernung von einem Vertex zu jeder Clipebene im Vertexprogramm explizit ausgerechnet werden muss. Beim Slicing kann anschließend die LIC-Berechnung wie bisher durchgeführt werden. Wird dagegen Ray-Casting eingesetzt, werden durch das Clipping Teile der Boundingbox entfernt. Da das Fragmentprogramm für das Ray-Casting nur für gerenderte Primitive durchgeführt wird, müssen die durch das Clipping entstandenen Flächen ebenfalls mit Primitiven gefüllt werden. Zur Erzeugung dieser Primitive wird der beim Slicing verwendete Algorithmus eingesetzt. Dieser wird so modifiziert, dass nur eine Ebene erzeugt wird, die identisch mit der Clipebene ist. Anschließend wird das Ray-Casting für diese zusätzliche Ebene durchgeführt.

5.4 Visualisierung zeitabhängiger Datensätze

In der Strömungsvisualisierung sollen oft Daten, die aus mehreren Zeitschritten bestehen, visualisiert werden. Da die Messungen beziehungsweise die Simulationen mit diskreten Zeitschritten erfolgen, können sich bei der Visualisierung Diskontinuitäten ergeben. Um diese zu verringern können Zwischenschritte berechnet werden. Dies kann beispielsweise durch Interpolation zwischen den einzelnen Zeitschritten erfolgen. Dazu werden die Datensätze von zwei Zeitschritten jeweils in einer 3D-Textur abgelegt. Im Fragmentprogramm der LIC-Berechnung findet in diesem Fall der Texturnachschlag mit der aktuellen Abtastposition in beiden Vektorfeldern statt. Die Berechnung des LIC wird anschließend mit dem Wert durchgeführt, der sich durch lineare Interpolation aus den Werten beider Texturen ergibt. Der Interpolationskoeffizient wird dazu von der Anwendung entsprechend gesetzt.

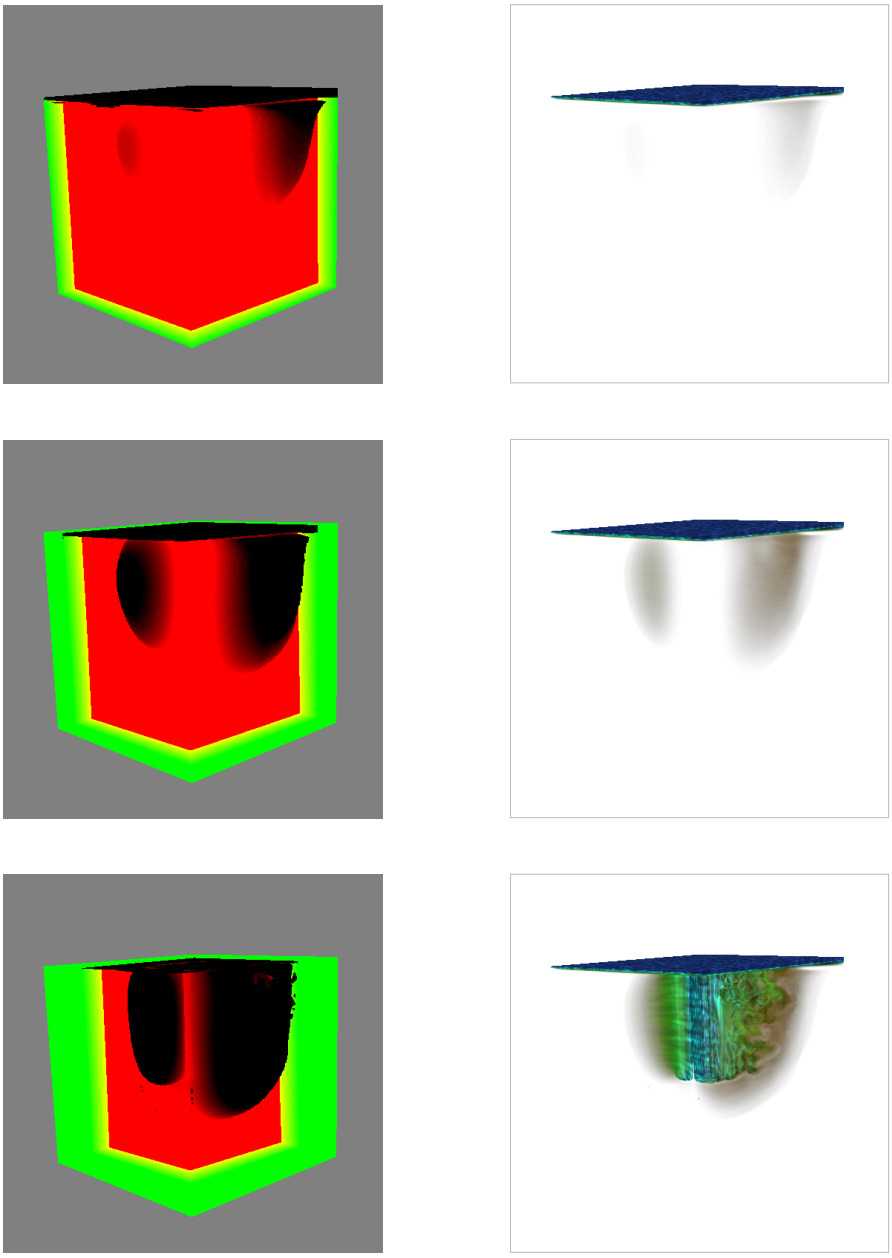


Abbildung 5.5: Hier sind die ersten drei Schritte einer LIC-Berechnung mittels Ray-Casting in Kombination mit Depth-Peeling dargestellt. Links sind die Texturen der jeweiligen Depth-Peel-Schicht dargestellt. Die Rotfärbung gibt an, wie viele Schritte im leeren Raum übersprungen werden können. In grünen Bereichen wurde das Volumen verlassen. Rechts ist das Ergebnis des LIC für jeden Schritt abgebildet. Als Datensatz wurde die Large Eddy Simulation von Octavian Frederich verwendet.

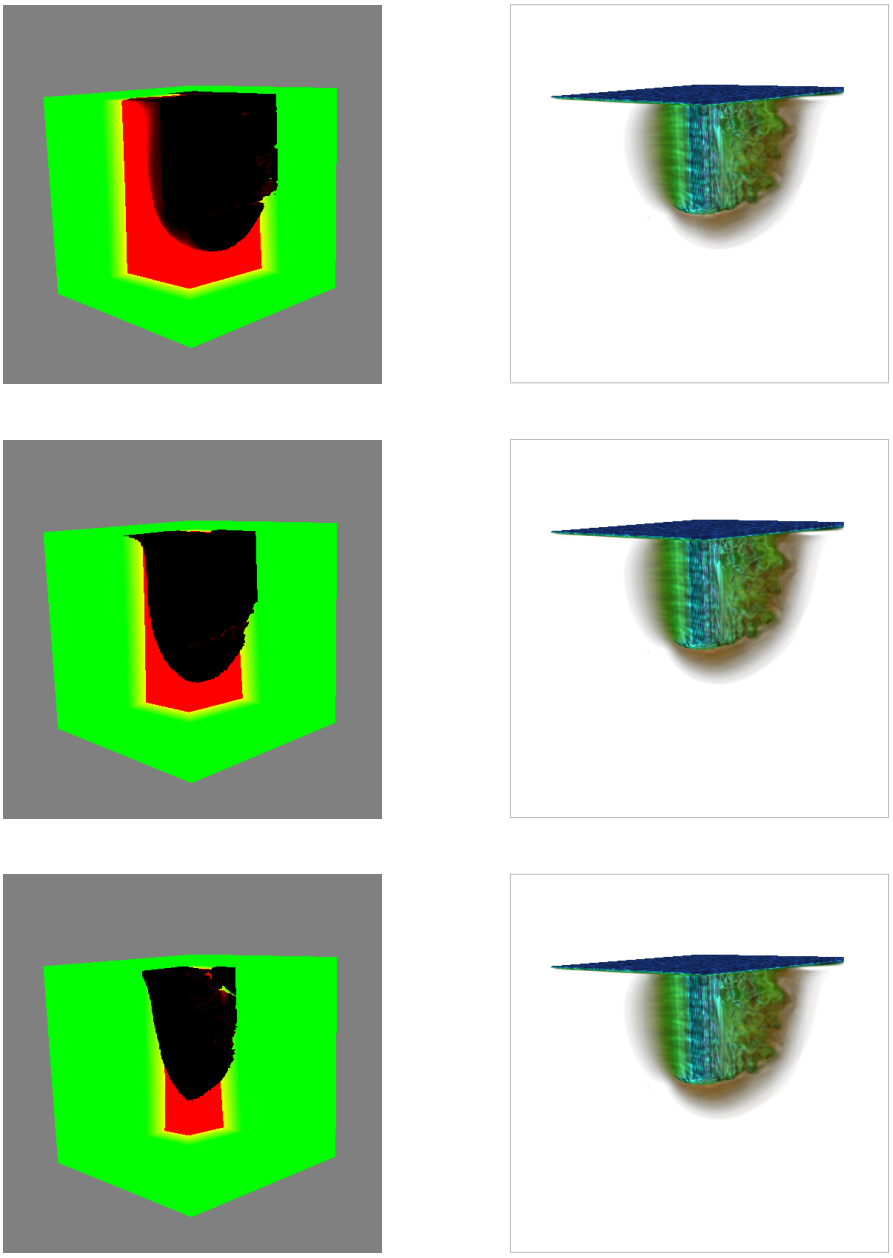


Abbildung 5.5: Hier sind die Schritte vier bis sechs einer LIC-Berechnung mittels Ray-Casting in Kombination mit Depth-Peeling dargestellt. Links sind die Texturen der jeweiligen Depth-Peel-Schicht dargestellt. Die Rotfärbung gibt an, wie viele Schritte im leeren Raum übersprungen werden können. In grünen Bereichen wurde das Volumen verlassen. Rechts ist das Ergebnis des LIC für jeden Schritt abgebildet. Als Datensatz wurde die Large Eddy Simulation von Octavian Frederich verwendet.

6 Leistungsbetrachtung

Die Messungen wurden auf einem AMD Opteron 2800 mit Dual Core und 8 GB RAM durchgeführt. Als Grafikkarte wurde eine Quadro FX 3450 mit 256 MB Grafikspeicher von nVidia eingesetzt. Die Viewportgröße wurde auf 512×512 Pixel gesetzt. Wenn in den einzelnen Messungen nicht anders vermerkt, wurde eine Samplingdistanz von $1/128$ verwendet. Die Anzahl der Abtastpunkte pro Depth-Peel-Schicht beim Ray-Casting mit Depth-Peeling wurde auf 50 festgelegt. Bei der Verwendung weniger Abtastpunkte steigt die Zahl der Schichten, die vorberechnet und in Texturen gespeichert werden müssen. Speziell bei der Messung des Einflusses verschiedener Viewportgrößen reicht deswegen bei höheren Auflösungen der Grafikspeicher nicht mehr aus. Wird stattdessen die Zahl der Abtastpunkte pro Schicht erhöht, so übersteigt die Zahl der Instruktionen im Fragmentprogramm das maximale Limit von 65536.

Gemessen wird die Zeit, die für das Rendern eines Bild benötigt wird, die Zahl der berechneten Pixel und die Anzahl der Abtastpunkte. Um die Zeit zu messen, wird die Szene zwischen zehn und 60 Mal, abhängig von der erwarteten Zeit, gerendert und die gemessenen Zeiten gemittelt. Zur Bestimmung der Anzahl der berechneten Pixel wird dieselbe Geometrie verwendet wie bei der eigentlichen LIC-Berechnung, d.h. ein Würfel beim Ray-Casting und sämtliche Slices beim Slicing. Für das Ray-Casting wurden die Fragmentprogramme so abgeändert, dass nur ein Schritt durchgeführt wird, aber keine LIC-Berechnung stattfindet. Beim Slicing wird die Geometrie ohne Fragmentprogramm gerendert. Anschließend wird der Inhalt des Framebuffers zurückgelesen und die Pixel, deren Alphawert größer null ist, gezählt. Für die Ermittlung der Anzahl der Abtastpunkte wurden die Fragmentprogramme für die LIC-Berechnung ebenfalls etwas abgeändert. Da jedoch das Ergebnis der LIC-Berechnung Einfluss auf die Anzahl der Abtastpunkte hat, kann diese Messung nur gleichzeitig mit einer LIC-Berechnung durchgeführt werden. Ein Beispiel hierfür wäre der vorzeitige Strahlabbruch. Zu diesem Zweck wurden die Fragmentprogramme mit einem zusätzlichen Zähler versehen, der inkrementiert wird, wenn an der aktuellen Samplingposition eine LIC-Berechnung stattfindet. Um nicht das Ergebnis des LIC zu beeinflussen, wird dieser Zähler in ein weiteres Rendertarget geschrieben. Durch Zurücklesen dieses weiteren Rendertargets kann sodann die Zahl der Abtastpunkte ermittelt werden. Wird Ray-Casting mit Depth-Peeling oder Slicing verwendet, so muss dieses Rendertarget nach jedem Depth-Peel-Schicht beziehungsweise nach jedem Slice zurückgelesen und das Ergebnis akkumuliert werden. In den Diagrammen wird das Ray-Casting mit Depth-Peeling abgekürzt mit Ray-Casting+DP.

Der für die Messungen verwendete Datensatz stammt aus einer Large-Eddy-Simulation und ist Eigentum von Octavian Frederich, TU Berlin. Der Datensatz besitzt eine Auflösung von $128 \times 128 \times 128$ Voxeln. Es handelt sich dabei um den Fluss einer Flüssigkeit oder eines Gases um einen massiven Zylinder.

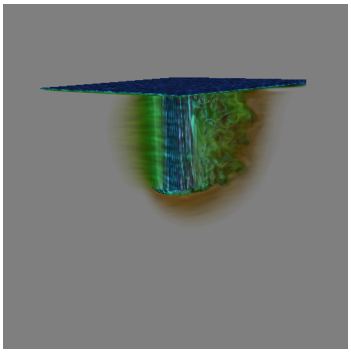


Abbildung 6.1: Testszenario zur Messung des Einflusses der Viewportgröße. Als Datensatz wurde die Large Eddy Simulation von Octavian Frederich verwendet.

6.1 Einfluss der Viewportgröße

Wenn die durchschnittlich benötigte Zeit pro Pixel konstant ist, sollte sich bei einer Vergrößerung des Viewports die Gesamtzeit für das Rendern linear dazu verhalten. Dazu wird, beginned bei einer Viewportgröße von 64×64 Pixeln, die Auflösung schrittweise um 64 Pixel jeweils in beide Dimensionen vergrößert. Die verwendete Szene ist in Abbildung 6.1 dargestellt.

Gemessen wurde die benötigte Dauer pro Bild und die Zahl der berechneten Pixel. Da eine Transferfunktion gewählt wurde, mit der große Teile des Vektorfelds ausgeblendet werden, musste die Schrittweite für die Messung mit einfachem Ray-Casting auf $1/64$ reduziert werden (Abbildung 6.2(a)). Ansonsten wäre das Instruktionslimit des Fragmentprogramms überschritten und die Messung verfälscht worden. In Abbildung 6.2(b) beträgt die Schrittweite $1/128$. Es wurde ein logarithmischer Maßstab gewählt, da sich die Anzahl der Pixel exponentiell zur Breite und Höhe des Viewports verhält. Das lineare Verhalten ist deutlich zu erkennen. Vervierfacht sich jedoch die Zahl der zu berechnenden Pixel, beispielsweise von 512×512 auf 1024×1024, so ist der Faktor bei der Renderdauer nicht ebenfalls vier sondern 2,5. Dies deutet auf einen Faktor hin, der mit zunehmender Viewportgröße beziehungsweise Renderdauer an Gewicht verliert. Möglicherweise ist das bedingt durch das Vorbereiten des Framebuffers und den Wechsel der Fragmentprogramme während eines Renderpasses.

6.2 Framebufferobjekt, Early-Z-Test und weitere Optimierungen

In dieser Messung werden verschiedene Möglichkeiten verglichen, die das Rendern beschleunigen oder das Zurücklesen von Texturen überflüssig machen. Ersteres wird unterstützt durch den Early-Z-Test (siehe Abschnitt 2.3.3), letzteres durch Framebufferobjekte, kurz FBO. Diese Framebufferobjekte erlauben es, direkt in Texturen zu rendern anstatt den Inhalt des Framebuffers nach dem Rendern in eine Textur zurückzulesen. Hinzu kommen weitere Optimierungen wie vorzeitiger Strahlabbruch (Abschnitt 2.3.3) oder Verzweigungen im Fragmentprogramm. Um diese Effekte zu messen, wird der Datensatz aus zwei unterschiedlichen Positionen, jedoch mit derselben Transferfunktion gerendert. Die Transferfunktion wurde so gewählt, dass nur der Fluss und die Wirbel in unmittelbarer Nähe des Zylinders zu sehen sind. Von der Seite muss so ein Großteil des Vektorfelds traversiert werden. Werden die LIC-Berechnungen

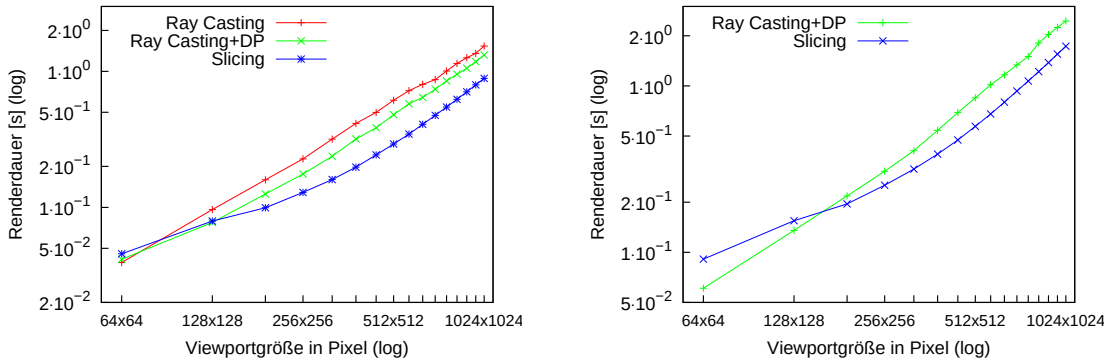
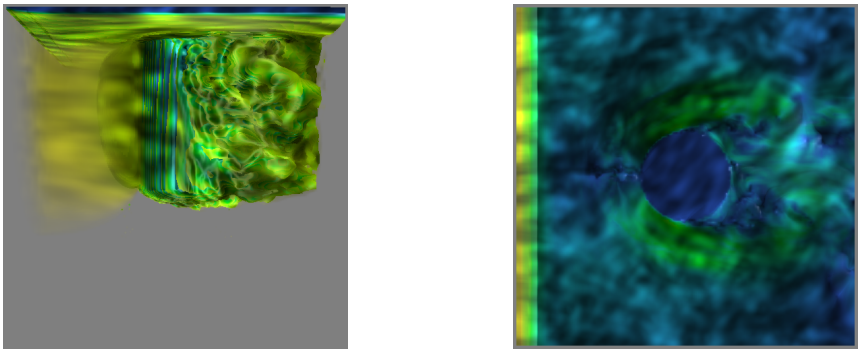


Abbildung 6.2: Einfluss der Größe des Viewports auf die Renderdauer. Der Abstand der Samplingpositionen beträgt 1/64 (a) und 1/128 (b).



(a) (b)

Abbildung 6.3: Testszenario zur Messung von Optimierungen und der Verwendung von Framebufferobjekten. Als Datensatz wurde die Large Eddy Simulation von Octavian Frederich verwendet. Die Messungen wurden mit Blick von der Seite (a) und von oben (b) durchgeführt.

innerhalb von diesem leeren Bereich nicht durchgeführt, sollte sich die Renderdauer verkürzen (Abbildung 6.3(a)). Da der Datensatz im oberen Bereich eine schmale Schicht mit sehr geringer Geschwindigkeit enthält, lässt sich hier der Effekt des Early-Z-Tests und der vorzeitige Strahlabbruch aufgrund hoher Opazitäten messen (Abbildung 6.3(b)).

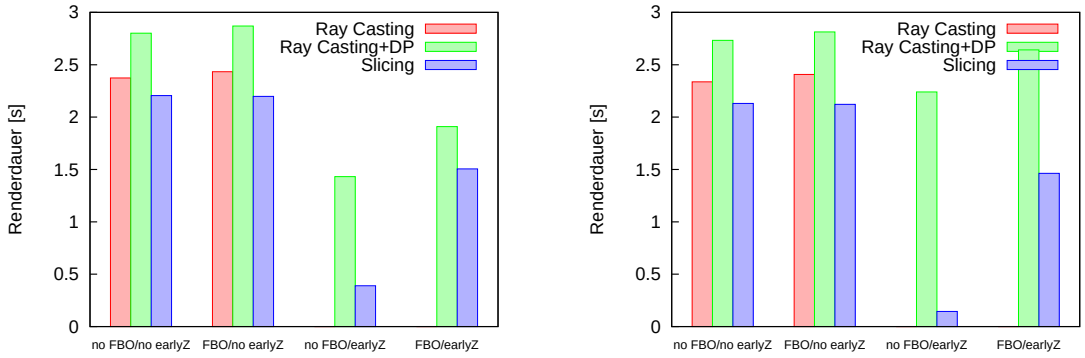
Bei dieser Messung wurden die Renderdauer und die Anzahl der Abtastpunkte ermittelt. Dafür wurden für beide Positionen drei Messreihen, jeweils mit allen Kombinationen von Early-Z-Test und Framebufferobjekten, durchgeführt. In der erste Messreihe wurden sämtliche Optimierungen in den Fragmentprogrammen einschließlich dem vorzeitigen Strahlabbruch deaktiviert. In der zweiten Testreihe wurde nur der vorzeitige Strahlabbruch und in der letzten Testreihe wurden sämtliche Optimierungen angeschaltet. Ist das Framebufferobjekt abgeschaltet, wird der Inhalt des Framebuffers mittels Zurücklesen in die entsprechenden Texturen kopiert. Für das Ray-Casting ohne Depth-Peeling wurden keine Messungen für den aktivierten Early-Z-Test durchgeführt, da dieses Verfahren für genau einen Pass konzipiert wurde. Abbildung 6.4 zeigt die Renderdauer bei gleichzeitiger Verwendung der nicht

optimierten Fragmentprogramme. Der positive Effekt des Early-Z-Tests ist hier deutlich zu erkennen. Wird nun der vorzeitige Strahlabbruch angeschaltet, verkürzt sich die Renderdauer (Abbildung 6.5) auch ohne den Early-Z-Test. Beim Blick von oben sorgt dieser Test dafür, dass die Berechnung schon nach ein oder zwei Ray-Casting-Schritten abgebrochen wird, da hier die gewünschte Opazität erreicht wurde. Das Slicing profitiert von diesem Strahlabbruch nicht, da dieser schon implizit im Early-Z-Test enthalten ist und weiterhin alle Slices gerendert werden müssen. Erst weitere Optimierungen in den Fragmentprogrammen sorgen dafür, dass die Renderzeiten beim Slicing sich denen der Ray-Casting-Verfahren angleichen (Abbildung 6.6). Zu diesen Optimierungen gehört das Auslassen der LIC-Berechnung für Abtastpunkte, an denen der Alphawert aus der Transferfunktion gleich null ist. Auffallend ist bei allen Messungen, dass das Zurücklesen aus dem Framebuffer schneller als die Verwendung von Framebufferobjekten und dem gleichzeitigen Rendern in Texturen ist. In den Diagrammen von Abbildung 6.7 sind die tatsächlich durchgeführten LIC-Berechnungen mit Optimierungen im Vergleich zu den insgesamt möglichen dargestellt. In Abbildung 6.7(b) ergibt sich der hohe Anteil an gesparten Berechnungen aus der vollständig opaken Schicht gleich zu Beginn des Vektorfelds.

6.3 Abhängigkeit von der Kameraposition

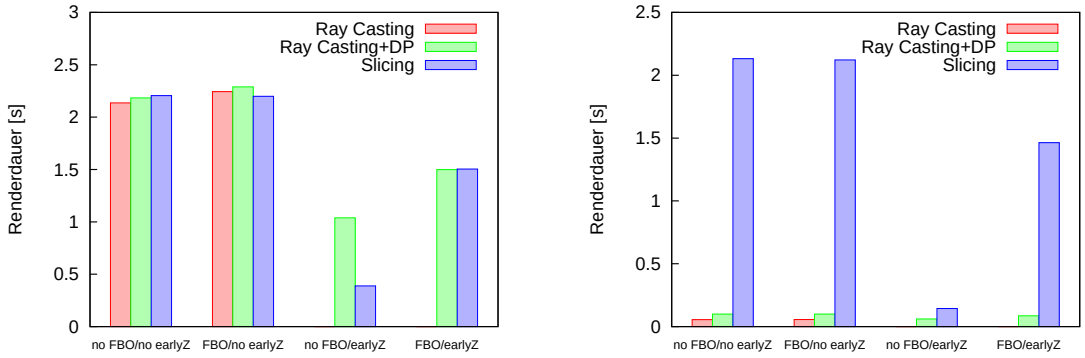
Bei dieser Messung werden die Zeiten von verschiedenen Kamerapositionen verglichen. Für die verschiedenen Kamerapositionen wird eine Haltonsequenz mit drei Komponenten verwendet. Die Elemente der Haltonsequenz werden zur Erzeugung von Positionen in einem Einheitswürfel verwendet. Dabei werden Elemente, deren Position sich außerhalb der Einheitskugel befinden, verworfen. Damit soll verhindert werden, dass sich in der Nähe der Diagonalen des Würfels mehr mögliche Positionen befinden als im übrigen Raum. Die so gewonnenen Positionen werden nach einer Normierung als neue Kamerarichtung verwendet. Die Kamera wird anschließend ausgehend von der positiven z-Achse mit Hilfe von Quaternionen in die neue Richtung gedreht. Dadurch ist sichergestellt, dass die Kamera immer auf den Ursprung ausgerichtet ist. Gleichzeitig soll auch der Einfluss der Transferfunktion gemessen werden. Dazu wurden zwei Transferfunktionen erstellt. Die erste Transferfunktion enthält nur Alphawerte von null (transparent) und eins (opak). Bei der zweiten wurden weichere Übergänge zwischen transparenten und opaken Werten gewählt (Abbildung 6.8).

Die Zeitmessungen für die erste Transferfunktion wurde für alle drei Techniken durchgeführt (Abbildung 6.9(a)). Bei der zweiten Transferfunktion wurde auf Messungen mit Ray-Casting ohne Depth-Peeling verzichtet, da hier öfter das Instruktionslimit des Fragmentprogramms überschritten worden wäre (Abbildung 6.9(b)). Auffallend ist, dass Slicing bei den hier gemessenen Kamerapositionen schneller ist als die Techniken, die auf Ray-Casting basieren. Die einzige Ausnahme bildet Position 10. Allerdings gilt zu beachten, dass hier nur die Zeit pro Bild gemessen und keine Aussagen über die Qualität gemacht wurden.



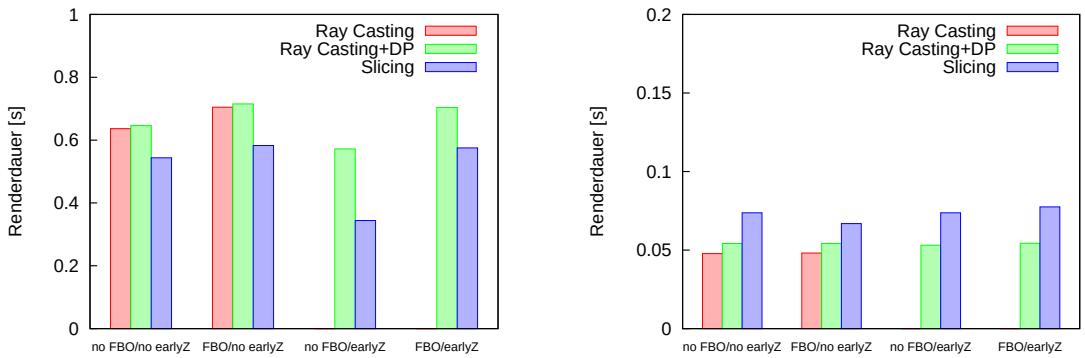
(a) (b)

Abbildung 6.4: Renderdauer ohne vorzeitigen Strahlabbruch und zusätzlichen Optimierungen im Fragmentprogramm. Vektorfeld von der Seite (a) und von oben (b).



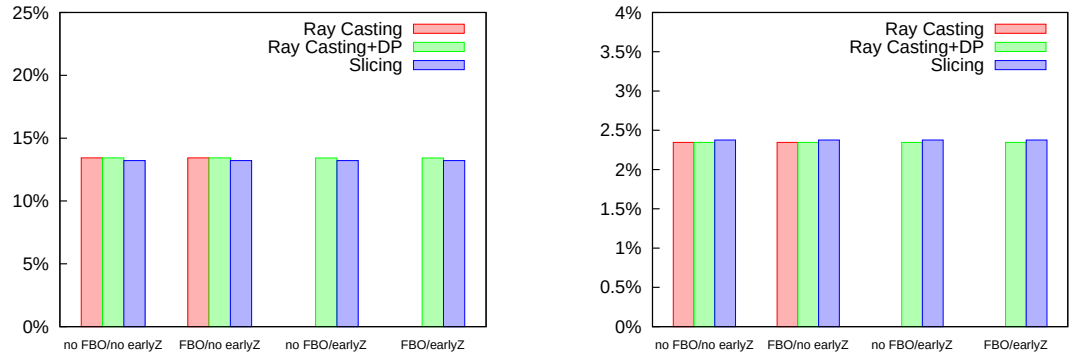
(a) (b)

Abbildung 6.5: Renderdauer mit aktiviertem vorzeitigen Strahlabbruch ohne zusätzliche Optimierungen im Fragmentprogramm. Vektorfeld von der Seite (a) und von oben (b).



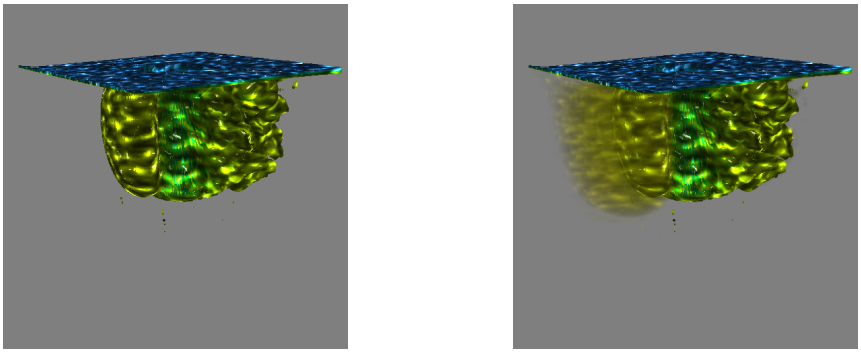
(a) (b)

Abbildung 6.6: Renderdauer mit aktiviertem vorzeitigem Strahlabbruch und zusätzlichen Optimierungen im Fragmentprogramm. Vektorfeld von der Seite (a) und von oben (b).



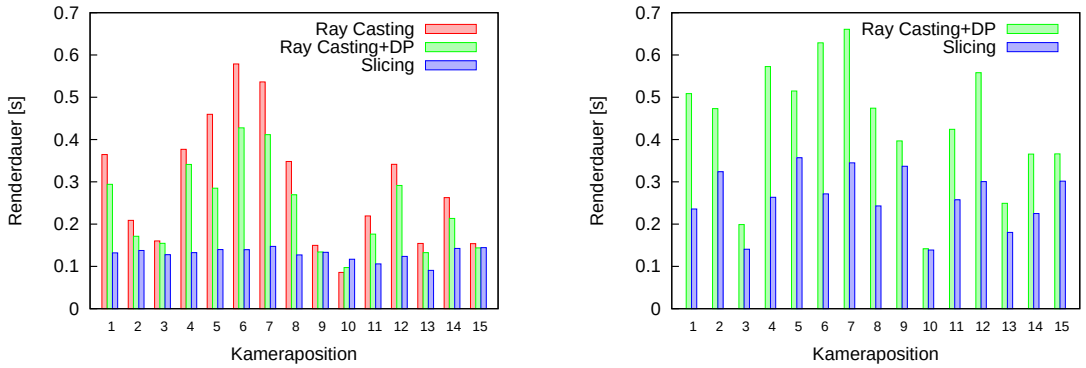
(a) (b)

Abbildung 6.7: Verhältnis zwischen durchgeführten LIC-Berechnungen mit optimiertem Fragmentprogramm und der Anzahl der Berechnungen im nichtoptimierten Fall. Vektorfeld von der Seite (a) und von oben (b).



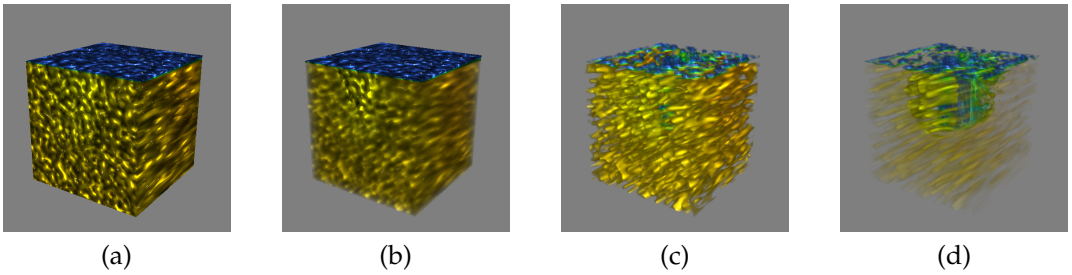
(a) (b)

Abbildung 6.8: Testszenario, um die Auswirkung verschiedener Kamerapositionen auf die Renderdauer zu messen. (a) harter Übergang der Opazität. (b) weicher Übergang der Opazität.

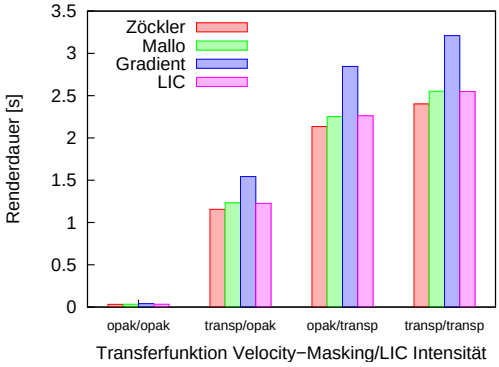


(a) (b)

Abbildung 6.9: Auswirkung verschiedener Kamerapositionen auf die Renderdauer. (a) harter Übergang der Opazität. (b) weicher Übergang der Opazität.



(a) (b) (c) (d)



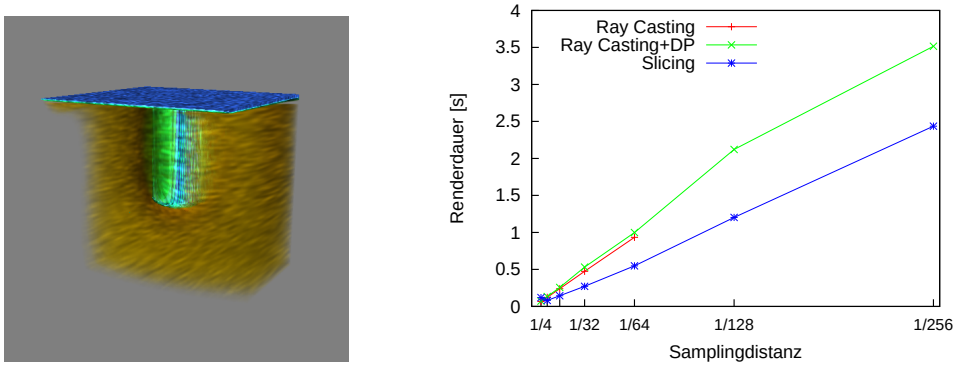
(e)

Abbildung 6.10: Gegenüberstellung der Beleuchtungsmodelle von Zöckler und Mallo, der Beleuchtung mittels Gradienten und des unbeleuchteten LICs bei Veränderung der Transferfunktionen für Velocity-Masking und LIC-Intensität. Testszenario mit Gradientenbeleuchtung: (a) beide Transferfunktionen vollständig opak. (b) Velocity-Masking mit Transparenz. (c) niedrige LIC-Intensität mit Transparenz. (d) Kombination aus (b) und (c).
(e) Messung der benötigten Zeit pro Bild für diese Transferfunktionen.

6.4 Beleuchtungsmodelle

Für den Vergleich der verschiedenen Beleuchtungsmodelle von Zöckler, Mallo, Beleuchtung mittels Gradienten und des unbeleuchteten LIC (siehe Abschnitt 5.2) wird Ray-Casting mit Depth-Peeling zusammen mit mehreren Transferfunktionen eingesetzt. Eine Transferfunktion wird für das Velocity-Masking verwendet, um Teile des Vektorfelds auszublenden (siehe Abschnitt 3.5), eine weitere wird zur Abbildung der LIC-Intensität verwendet. Damit ist es beispielsweise möglich, Bereiche, in denen die Intensität des LIC sehr gering ist, auszublenden. Für die Messung werden diese zwei Transferfunktionen miteinander kombiniert. In Abbildung 6.10(a)-(d) sind die Effekte dieser Transferfunktionen exemplarisch dargestellt.

Die benötigte Zeit pro Bild ist in Abbildung 6.10(e) dargestellt. Auf der x-Achse sind vier Kombinationen der Transferfunktionen aufgetragen. Die erste Komponente jeder Kombination bezieht sich auf die Transferfunktion des Velocity-Maskings, die zweite bezieht sich auf die Abbildung der LIC-Intensität. Sind beide Transferfunktionen opak, so wird nur eine LIC-Berechnung direkt an der Oberfläche des Würfels durchgeführt. Für alle Beleuchtungsmodelle ergeben sich deshalb sehr geringe Renderzeiten. Erkennbar ist, dass die Beleuchtung



(a) (b)

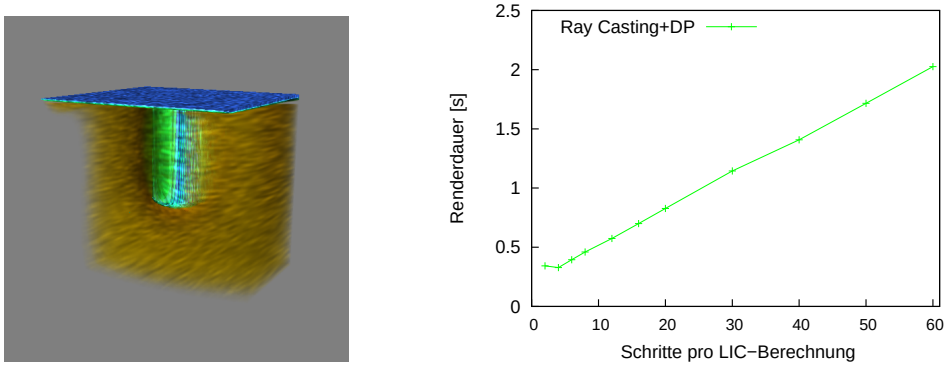
Abbildung 6.11: (a) Testszenario zur Messung des Einflusses der Samplingdistanz. Als Datensatz wurde die Large Eddy Simulation von Octavian Frederich verwendet. (b) Dauer pro Bild in Abhängigkeit von der Samplingdistanz. Die Unterteilung der x-Achse entspricht der Anzahl der Abtastpunkte.

anhand von Gradienten die meiste Zeit beansprucht. Dies dürfte auf die zusätzlichen Komponenten des Rauschens zurückzuführen sein. Denn anstatt nur die Intensität des Rauschens aus einer Textur zu lesen, müssen zusätzlich die Gradienten ausgelesen und verarbeitet werden. Erstaunlich ist, dass der unbeleuchtete LIC mehr Zeit zur Berechnung benötigt als das Modell von Zöckler, wobei nach der Berechnung des LIC nur die Abbildung auf die zweite Transferfunktion und die Korrektur der Opazität durchgeführt wird. Bei Zöckler hingegen ist ein Texturnachschlag erforderlich. Möglicherweise kann durch diesen Texturnachschlag das Fragmentprogramm besser optimiert werden oder mögliche Latenzen werden dadurch aufgefangen. Für eine nähere Untersuchung müssen genauere Messungen durchgeführt werden. Es muss auch mehr über den internen Aufbau der Grafikhardware bekannt sein. Dagegen erscheint es logisch, dass die Beleuchtung nach Mallo langsamer ist als bei Zöckler, da die Berechnung der Texturkoordinaten komplexer ist und zwei Texturnachschläge benötigt werden.

6.5 Samplingdistanz

Die pro Bild benötigte Zeit sollte linear zur Anzahl der Samplingpositionen sein, da jede LIC-Berechnung pro Samplingposition exakt dieselbe Zeit beanspruchen sollte. Da sich die Zahl der Abtastungen schwer einstellen lässt, wird bei dieser Messung die Samplingdistanz verwendet, die sich reziprok dazu verhält. Die Transferfunktion wurde so gewählt, dass der Fluss um den Zylinder halb transparent ist. Dadurch müssen mehrere LIC-Berechnungen durchgeführt werden, um ein opakes Fragment zu erhalten. Abbildung 6.11(a) zeigt das Vektorfeld mit der entsprechenden Transferfunktion.

In Abbildung 6.11(b) ist das Ergebnis der Messung dargestellt. Die x-Achse wurde so skaliert, dass sie der tatsächlichen Anzahl von Abtastpunkten entspricht. So sollte sich die Anzahl der Abtastungen um Faktor zwei verdoppeln, wenn die Samplingdistanz halbiert wird. Zu erkennen ist der erwartete lineare Verlauf. Für das Ray-Casting ohne Depth-Peeling wurden nur



(a) (b)

Abbildung 6.12: (a) Testszenario um den Einfluss der Anzahl der LIC-Schritte zu bestimmen. Als Datensatz wurde die Large Eddy Simulation von Octavian Frederich verwendet. (b) Dauer pro Bild in Abhängigkeit der LIC-Schritte.

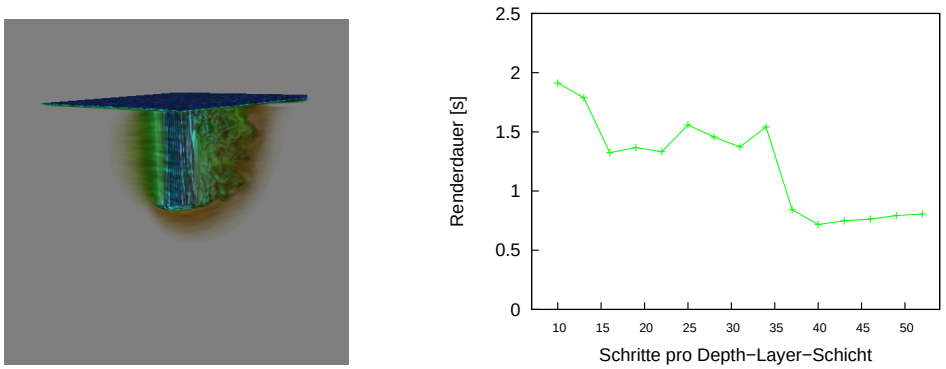
die Messdaten bis zu einer Samplingdistanz von $1/64$ abgebildet, da das Instruktionslimit bei kleineren Schrittweiten überschritten wurde.

6.6 Anzahl der LIC-Schritte pro Abtastpunkt

Die Breite des LIC-Kernels wird durch zwei Faktoren bestimmt: Zum Einen durch die Schrittweite zwischen den einzelnen Abtastungen und zum Anderen durch die Anzahl der Schritte in beide Richtungen. Die Änderung der Schrittweite hat nur Einfluss auf das Ergebnis der Berechnung, aber nicht auf die Komplexität, da das Vektorfeld nur an anderen Punkten abgetastet wird. Nimmt dagegen die Anzahl der LIC-Schritte zu, so müssen die zwei Schleifen für die Vorwärts- beziehungsweise die Rückwärtsrichtung mehrmals ausgeführt werden. Da der Schleifenrumpf in beiden Fällen nur aus Instruktionen ohne Verzweigung oder Sprüngen besteht, liegt die Komplexität des Rumpfs in $\mathcal{O}(1)$. Dadurch sollte sich ein linearer Zuwachs bei der benötigten Zeit pro Bild ergeben. Bei dieser Messung wurde als Technik Ray-Casting mit Depth-Peeling eingesetzt. In Abbildung 6.12 ist die Testszene und das Ergebnis der Messung dargestellt. Mit Ausnahme des ersten Messwerts ergibt sich der erwartete lineare Verlauf.

6.7 Anzahl der Abtastungen pro Depth-Peel-Schicht

Wird Ray-Casting mit Depth-Peeling verwendet, so werden abhängig von der Anzahl der Abtastungen pro Depth-Peel-Schicht mehr oder weniger Schichten benötigt. Da mit jeder zusätzlichen Depth-Peel-Schicht beim Rendern ein gewisser Overhead verbunden ist, sollte die Zahl der Schichten möglichst gering sein und damit die Anzahl der Abtastungen pro Schicht groß. Allerdings ist die maximale Zahl der Abtastungen aufgrund des Instruktionslimits für Fragmentprogramme auf 50 beschränkt. Für diese Messung wurde die Transferfunktion so gewählt, dass große Bereiche entstehen, die vollständig transparent sind, Teile des Inneren jedoch gleichzeitig opak sind (Abbildung 6.13(a)). Bei dieser Transferfunktion sollte die Zahl



(a) (b)

Abbildung 6.13: (a) Zur Messung der Auswirkungen von unterschiedlich vielen Abtastungen pro Depth-Peel-Schicht verwendetes Testszenario. Als Datensatz wurde die Large Eddy Simulation von Octavian Frederich verwendet. (b) Dauer pro Bild in Abhängigkeit zur Anzahl der Abtastungen pro Depth-Peel-Schicht.

der Abtastungen pro Depth-Peel-Schicht möglichst groß sein, um den leeren Raum mit möglichst wenigen Schichten überspringen zu können. Im Inneren des Vektorfelds sollte jedoch die Zahl der Schichten etwas höher sein, um vom Early-Z-Test zwischen den einzelnen Schichten profitieren zu können.

Das Ergebnis der Messung für diese Transferfunktion ist in Abbildung 6.13(b) zu sehen. Auffällig sind die starke Schwankungen besonders im mittleren Bereich. Eine Erklärung dafür könnte das interne Caching der Grafikkarte sein. Das notwendige Wechseln der Shader während des Rendervorgangs könnte sich ebenfalls auswirken. Da über die interne Architektur der heutigen Grafikkarten wenig bekannt ist, lassen sich in beiden Fällen keine konkreten Aussagen treffen. Anzumerken bleibt, dass hier nur eine spezielle Transferfunktion untersucht wurde. Da die Anzahl der Abtastungen pro Depth-Peel-Schicht abhängig von der verwendeten Transferfunktion ist, ist es nicht möglich, eine generelle Aussage dazu zu treffen.

7 Qualitative Ergebnisse

7.1 Visuelle Auswertung

Verbesserung der Interaktion

Da die Berechnung des 3D LIC nicht interaktiv erfolgt, wird während der Benutzerinteraktionen der Datensatz mit reduzierter Qualität dargestellt (Abbildung 7.1). Dazu wird die Auflösung des Viewports halbiert und die Samplingdistanz vergrößert. Gleichzeitig wird die Raumfrequenz des Rauschens verringert, um den Gesamteindruck ansatzweise zu erhalten.

Maximales Instruktionslimit des Fragmentprogramms

Aufgrund der hohen Komplexität der LIC-Berechnung kann das Instruktionslimit der Fragmentprogramme überschritten werden. Da beim Ray-Casting ohne Depth-Peeling die komplette Berechnung des 3D LIC innerhalb eines Fragmentprogramms erfolgt, wird dieses Instruktionslimit sehr schnell erreicht. Sobald das Instruktionslimit überschritten wird, tauchen im Framebuffer beziehungsweise in der Textur des Rendertargets meist grüne Pixel auf. In Abbildung 7.2 ist dieser Fall abgebildet.

Skalierung der Raumfrequenzen des Rauschens

Um den Einfluss der Raumfrequenz des Rauschens auf das Ergebnis des 3D LIC zu demonstrieren, wurde ein dünnes Rauschen verwendet und die Raumfrequenzen des Rauschen durch eine Skalierung verändert. Abbildung 7.3 zeigt, dass die Granularität des 3D LIC von der Raumfrequenz des Rauschens beeinflusst wird. So ergeben sich für niedrige Frequenzen größere Anhäufungen und für höhere Frequenzen feine Stromlinien.

Einfluss des verwendeten Rauschens

Der Typ des verwendeten Rauschen wirkt sich auf das Ergebnis des 3D LIC auswirken. Besonders bei der Verwendung der Beleuchtung mittels Gradienten sollte das Rauschen keine hohen Frequenzen enthalten, damit durch Beleuchtung der Eindruck glatter Flächen erweckt wirkt. Abbildung 7.4 veranschaulicht dazu auf der linken Seite verschiedene Typen von Rauschen. Auf der rechten Seite sind Ausschnitte derselben Szene zu sehen. Das weiße Rauschen erweckt einen guten Eindruck durch die feine Struktur. In der Vergrößerung ist jedoch zu erkennen, dass die Oberfläche der Stromlinien nicht glatt ist. Dies ist auf die hohen Raumfrequenzen der Textur zurückzuführen. Ist das Rauschen zu dünn, ergeben sich nur Bruchstücke der Stromlinien (Abbildung 7.4, vierte Reihe). Durch die Erhöhung der Schrittweite und der Schrittanzahl innerhalb des LIC sollten sich längere Stromlinien ergeben.

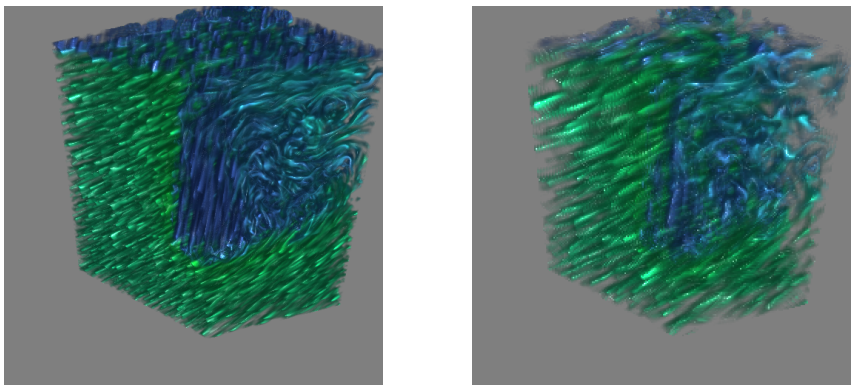


Abbildung 7.1: Zur Verbesserung der Interaktion wird die Qualität verringert. Links ist die hochaufgelöste Version dargestellt, rechts dieselbe Szene mit reduzierter Auflösung, größerer Samplingdistanz und kleinerem Skalierungsfaktor für das Rauschen. Der verwendete Datensatz stammt aus einer Large-Eddy-Simulation von Octavian Frederich.

Vergleich der Beleuchtungsmodelle

In Abbildung 7.5 sind die Normalen der drei verwendeten Modelle für Stromlinienbeleuchtung dargestellt. Bei den Normalen von Zöckler ist deutlich zu erkennen, dass diese koplanar zu der Ebene sind, die von der Blickrichtung und der Lichtrichtung aufgespannt wird. Die Normalen aus dem Modell von Mallo sind zwar abhängig von der Richtung des Vektorfelds, aber für jedes Stromlinienbündel wird nur eine einzige Normale verwendet. Die Verwendung des Gradienten aus der Rauschtextur führt zu Normalen, die einer Oberfläche entsprechen. Die drei Beleuchtungsmodelle und der unbeleuchtete LIC sind in Abbildung 7.6 veranschaulicht.

Einfluss der Transferfunktion

Die Transferfunktion kann wie in der Volumenvisualisierung dazu verwendet werden, bestimmte Bereiche des Datensatzes einzufärben und hervorzuheben. Gleichzeitig können andere Teile des Volumens ausgeblendet werden. Dies wird am Beispiel eines Wirbel in Abbildung 7.7 illustriert. Durch die Verwendung der speziellen Transferfunktion für den 3D LIC (Abschnitt 3.4.4) kann der Tiefeneindruck der Visualisierung deutlich verbessert werden. In Abbildung 7.8 werden links niedrige Intensitätswerte des LIC auf null abgebildet, wodurch sich leere Bereiche ergeben. Der Tiefeneindruck wird aber wesentlich verstärkt, wenn die Intensität auf Werte, die minimal über null liegen, abgebildet wird.

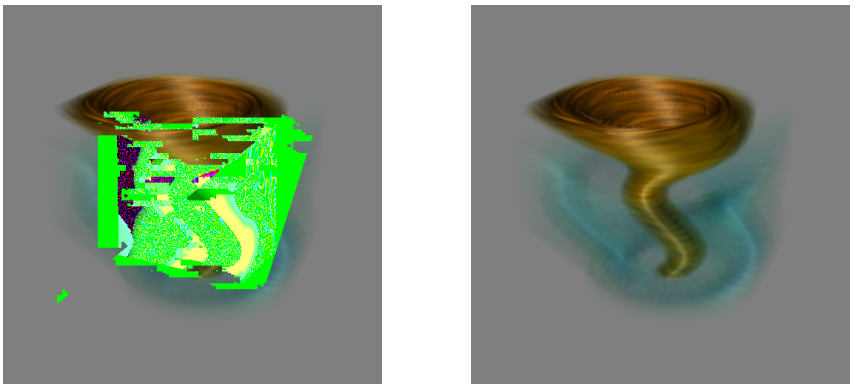


Abbildung 7.2: Maximale Anzahl der Instruktionen im Fragmentprogramm: Links wurde beim Ray-Casting die maximale Anzahl überschritten. Rechts wurde dieselbe Szene mit Ray-Casting und Depth-Peeling gerendert. Der verwendete Tornadodatensatz stammt von Roger Crawfis.

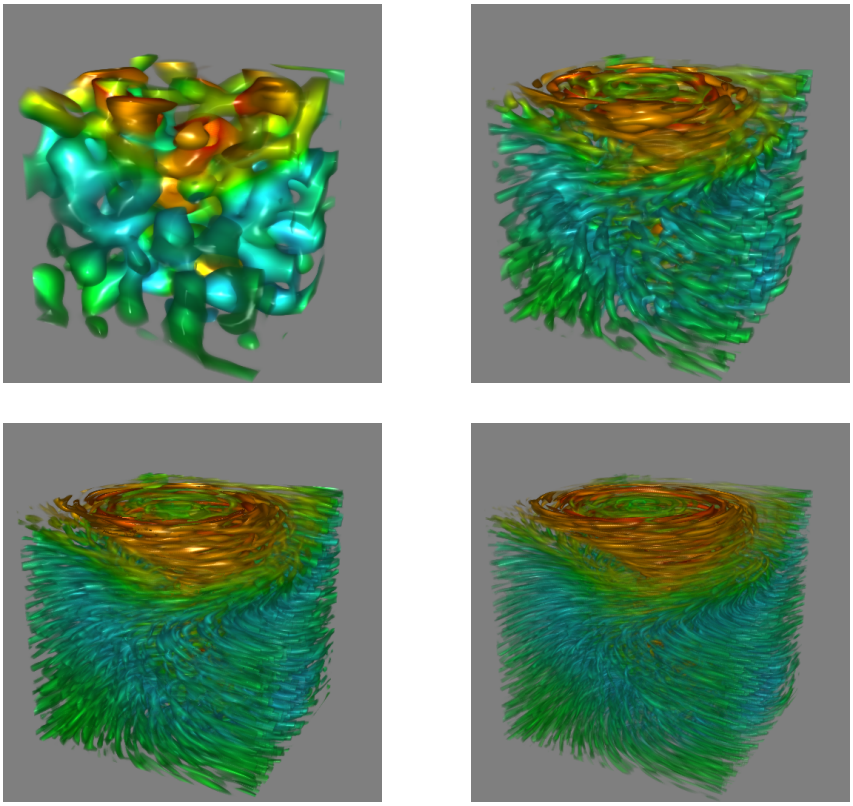


Abbildung 7.3: Die Skalierung der Raumfrequenz des Rauschens hat großen Einfluss auf das Ergebnis des 3D LIC. Für diese Abbildungen wurde ein dünnes Rauschen mit einer Auflösung von $256 \times 256 \times 256$ verwendet. Der Skalierungsfaktor beträgt 0,5, 1,5, 2,5 und 3,5 (von links nach rechts, von oben nach unten). Der verwendete Tornadodatensatz stammt von Roger Crawfis.

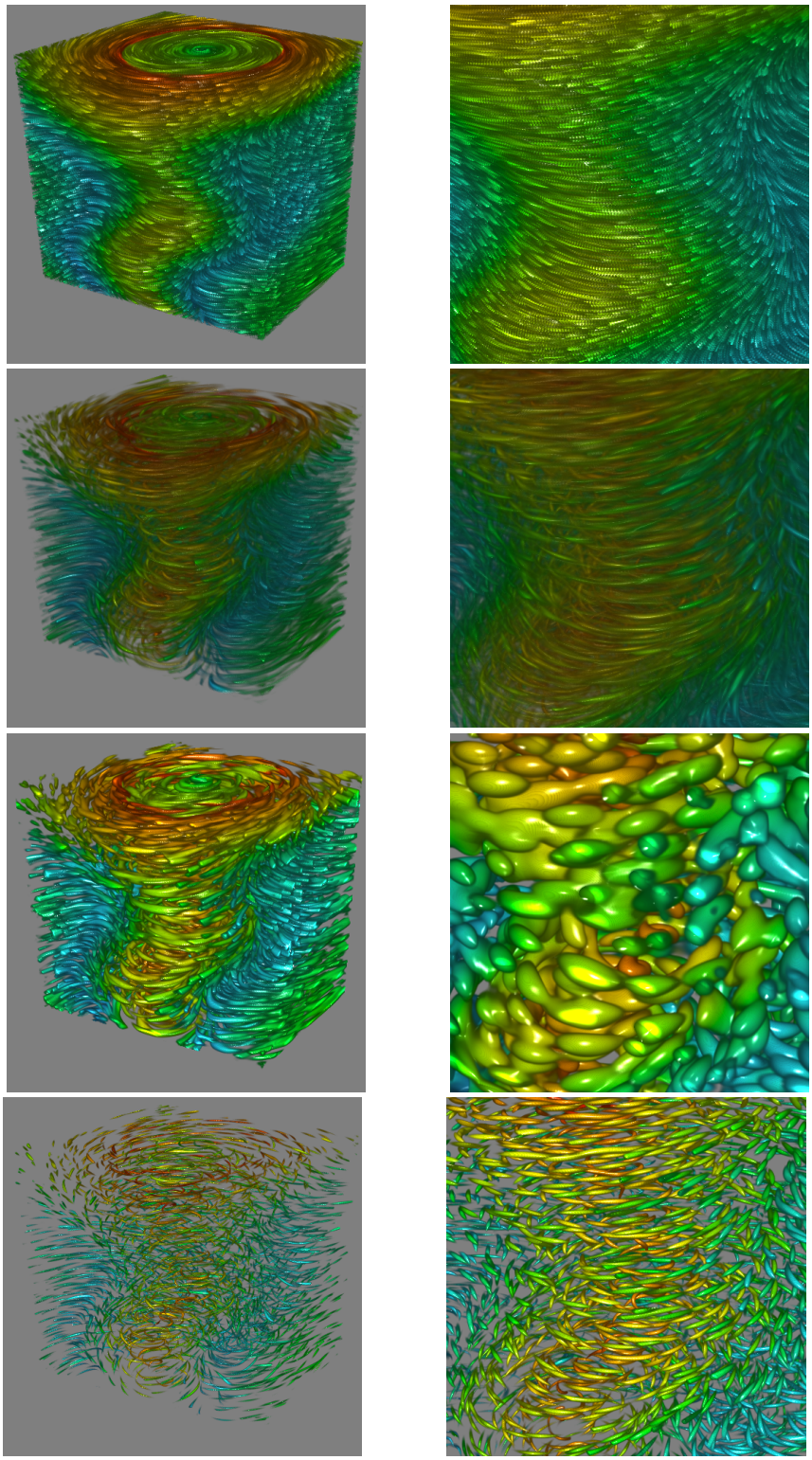


Abbildung 7.4: Der Datensatz ist links vollständig und rechts als Ausschnitt dargestellt. In der ersten Reihe wurde weißes Rauschen, in der zweiten, dritten und vierten Reihe wurde dünnes Rauschen eingesetzt. Dabei wurde für das Rauschen in Reihe zwei das Stratified Sampling eingesetzt. Reihe drei und vier zeigen jeweils ein Rauschen aus einer Haltonsequence, für Reihe vier wurde ein Tiefpassfilter mit sehr geringer Grenzfrequenz gewählt. Der Tornadodatensatz stammt von Roger Crawfis.

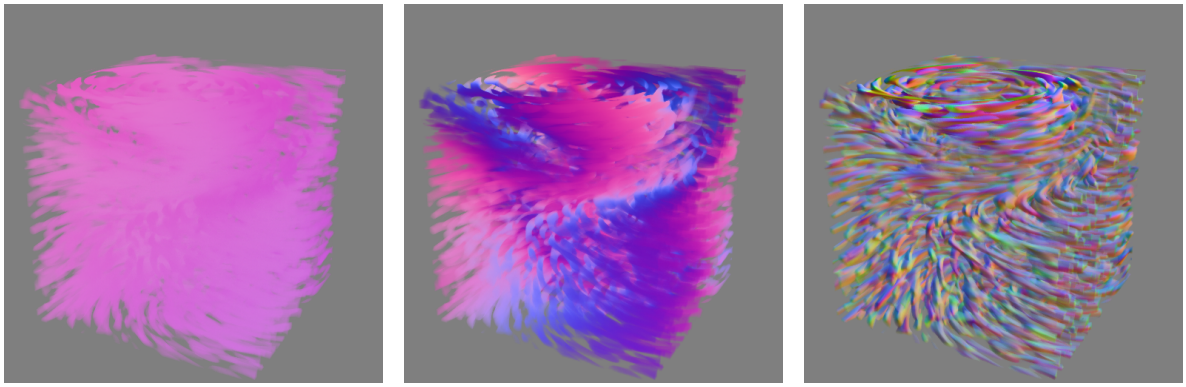


Abbildung 7.5: Hier sind die Normalen der verschiedenen Beleuchtungsmodelle dargestellt. Ganz links ist das Normalenfeld von Zöckler, in der Mitte das von Mallo und rechts das der Gradienten abgebildet. Als Datensatz wurde der Tornado von Roger Crawfis eingesetzt.

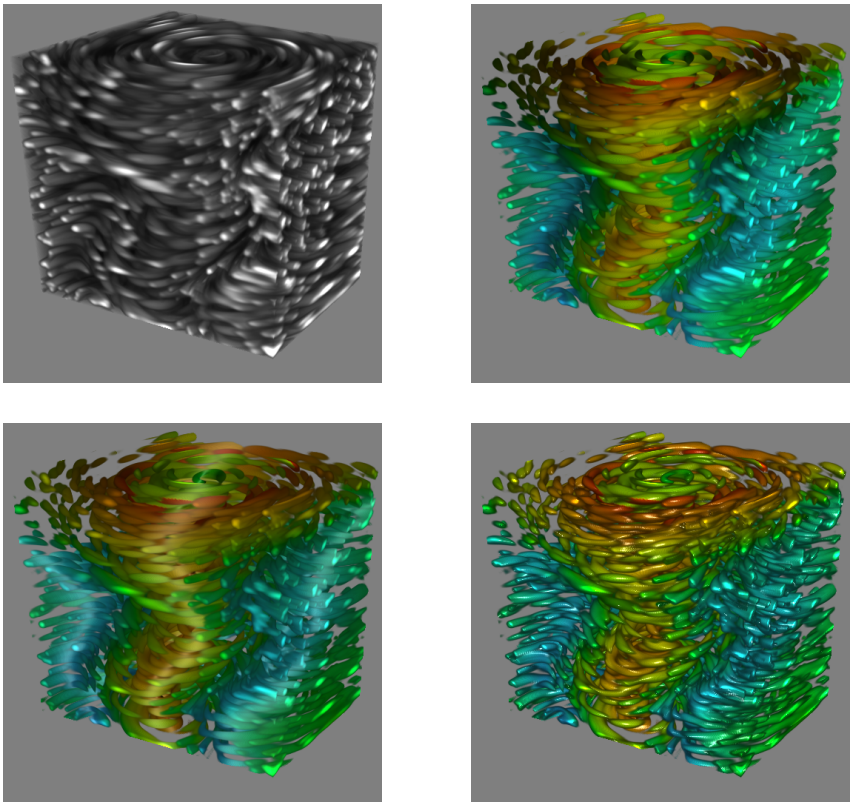


Abbildung 7.6: Links oben ist der unbeleuchtete LIC, rechts davon die Beleuchtung nach Zöckler et al. dargestellt. In der unteren Reihe befindet sich links das Modell von Mallo et al. und rechts die neu vorgestellte Beleuchtung mittels Gradienten des eingesetzten Rauschens. Als Datensatz wurde der Tornado von Roger Crawfis eingesetzt.

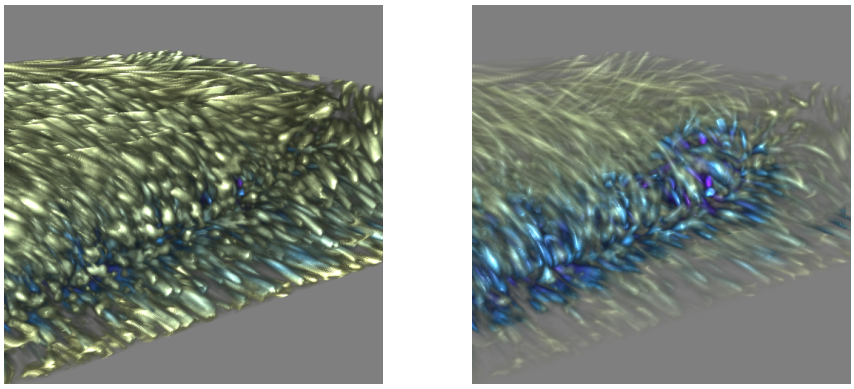


Abbildung 7.7: Mit einer geeigneten Transferfunktion lässt sich aus dem Volumen links ein Wirbel extrahieren (rechts). Als Datensatz wurde 6603-small eingesetzt.

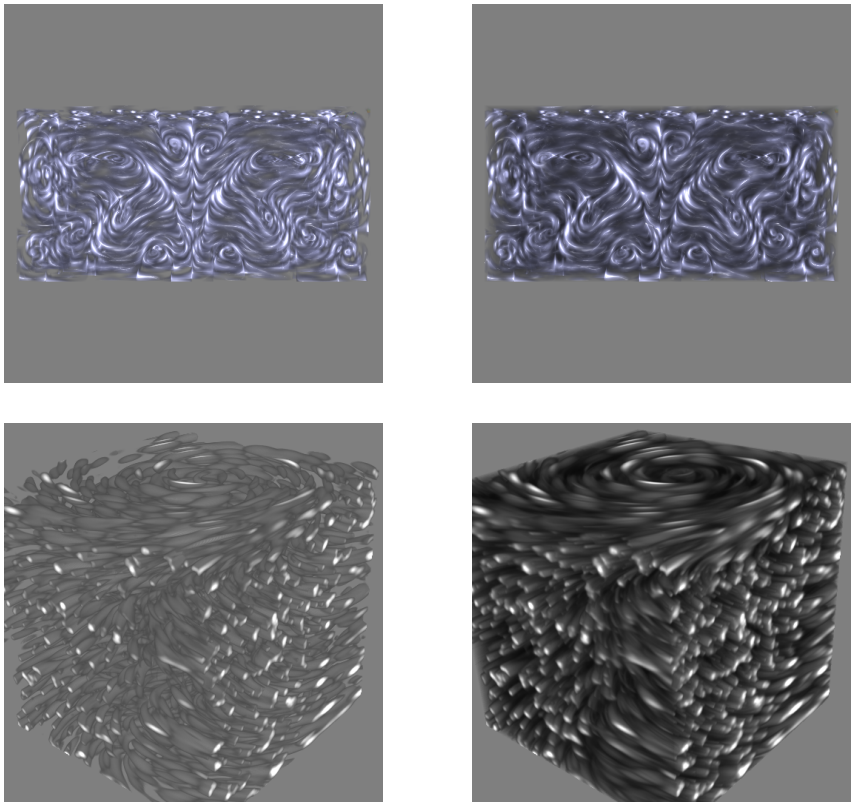


Abbildung 7.8: Zur Verbesserung der Tiefenwahrnehmung kann die Transferfunktion verwendet werden. In der linken Spalte werden niedrige Intensitäten des 3D LIC auf null abgebildet. Wird stattdessen ein Wert größer null verwendet ergibt sich ein besseres Tiefenbild (rechts). Als Datensätze wurden die Benard-Strömung von Daniel Weiskopf (oben) und der Tornado von Roger Crawfis (unten) verwendet.

7.2 Beispiele

Benard-Strömung

Die Benard-Strömung von Daniel Weiskopf besteht aus zahlreichen Wirbeln. Der Datensatz besitzt eine Auflösung von $128 \times 32 \times 64$. Leider stand der Datensatz nur als Vektorfeld mit normierten Richtungen ohne Geschwindigkeitsinformation zur Verfügung. In Abbildung 7.9 ist links der Datensatz mit einem dünnen Rauschen und dem 3D LIC dargestellt. Rechts werden mit Hilfe des Lambda2-Verfahrens die Wirbel identifiziert und rot hervorgehoben.

IEEE Visualization 2004 Contest

Dieser Datensatz stammt aus dem IEEE Visualization Contest im Jahr 2004. Es handelt sich dabei um einen Datensatz einer Hurrikansimulation des National Center for Atmospheric Research in den Vereinigten Staaten. Die Strömungsdaten wurden aus dem multivariten Originaldatensatz gewonnen. Der Datensatz besitzt eine Auflösung von $500 \times 500 \times 100$ und wird in Abbildung 7.10 veranschaulicht.

Tornado

Dieser Tornado stammt aus einer Simulation von Roger Crawfis. Der Datensatz wurde mit einer Auflösung von $256 \times 256 \times 256$ erzeugt. Für die Darstellung in Abbildung 7.11 werden Velocity-Masking, Clipping und das Lambda2-Verfahren eingesetzt.

Large-Eddy-Simulation

Für diese Simulation wurde eine Auflösung von $256 \times 128 \times 128$ zugrunde gelegt. Hierbei handelt es sich um einen Datensatz von Octavian Frederich, in dem ein Zylinder von einer

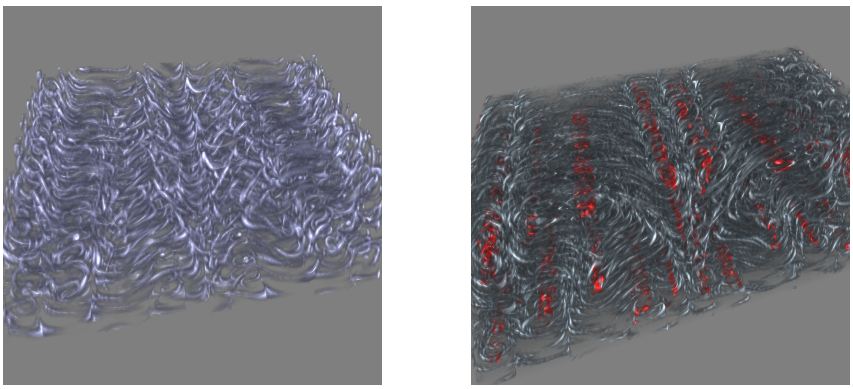


Abbildung 7.9: Benard-Strömung von Daniel Weiskopf: Da in diesem Datensatz leider nur die normierten Vektoren zur Verfügung standen, kann die Strömung nur in einem Farbton dargestellt werden (links). Rechts sind die Wirbel basierend auf dem Lambda2-Wert mit rot hervorgehoben.

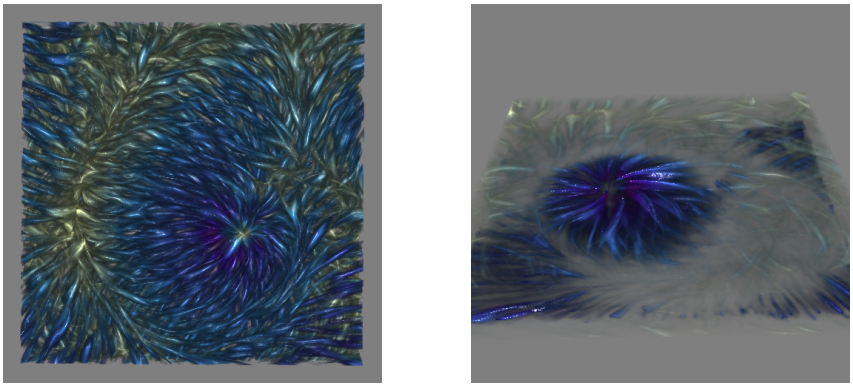


Abbildung 7.10: IEEE Visualization 2004 Contest: Links wurde die Kamera direkt über dem Datensatz platziert. Der Hurrikan befindet sich etwas rechts unterhalb der Bildmitte. Im rechten Bild wurde der Kern des Hurrikan mittels Lambda2-Verfahren extrahiert. Die Farben repräsentieren die Geschwindigkeit. Weiße Bereiche deuten auf eine langsame und blau-violette Bereiche auf eine schnelle Strömung hin.

Strömung umflossen wird. In Abbildung 7.12 ist der Datensatz von der Seite und von oben dargestellt. Für die Farbkodierung wurde die Geschwindigkeit verwendet. Langsame Bereiche sind von blau bis grün, schnellere Bereiche von grün über gelb bis hin zu rot eingefärbt.

6603-small

Bei diesem Datensatz handelt es sich um zwei Wirbel mit hoher Geschwindigkeit. Diese Wirbel sind umgeben von einer langsamen Strömung, welche in Abbildung 7.13 zu sehen sind. Die Auflösung des Datensatzes beträgt $135 \times 225 \times 129$.

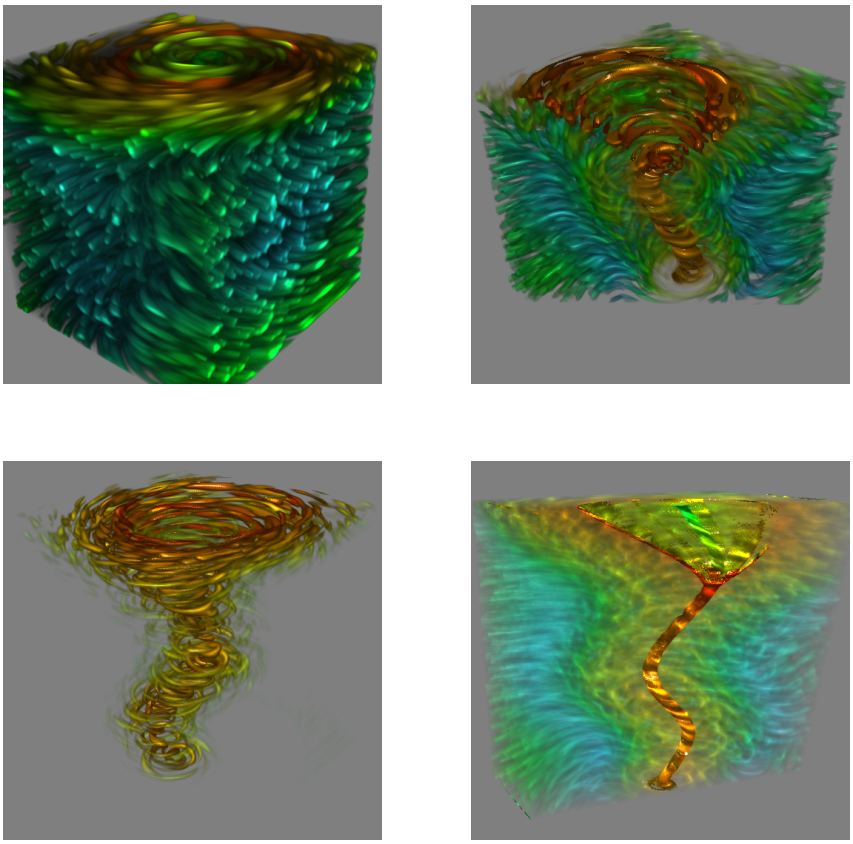


Abbildung 7.11: Tornado von Roger Crawfis: Links ist der Tornadodatensatz von aussen zu sehen. Die Transferfunktion ist so gewählt, dass die Stromlinienbündel deutlich zu sehen sind. Im Bild rechts wird das Innere des Tornados durch Clipping sichtbar gemacht. In der unteren Reihe wird der Tornado links durch Velocity-Masking hervorgehoben. Rechts wurde der Wirbelkern des Tornados durch das Lambda2-Kriterium bestimmt und opak gerendert.

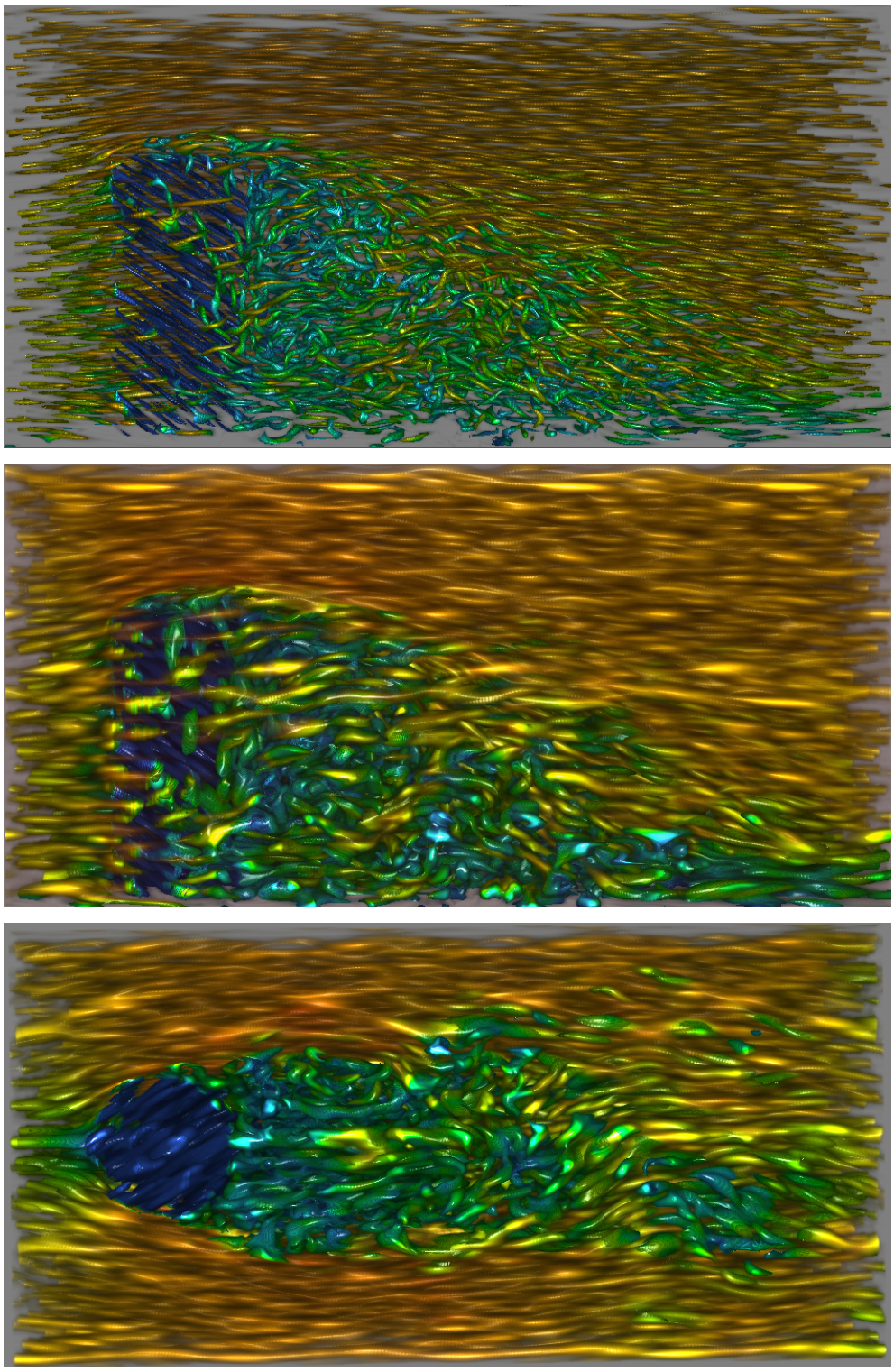


Abbildung 7.12: Large-Eddy-Simulation von Octavian Frederich: Im ersten Bild wird ein sehr dünnes Rauschen verwendet. Der Zylinder ist dunkelblau eingefärbt. Im nächsten Bild wird dieselbe Kameraposition aber ein dichteres Rauschen verwendet. Für das letzte Bild wird mit einer Clipping-Ebene der obere Teil des Volumens entfernt und von oben auf die Strömung geblickt. In allen Bildern sind deutlich die Verwirbelungen hinter dem Zylinder zu erkennen. (von oben nach unten)

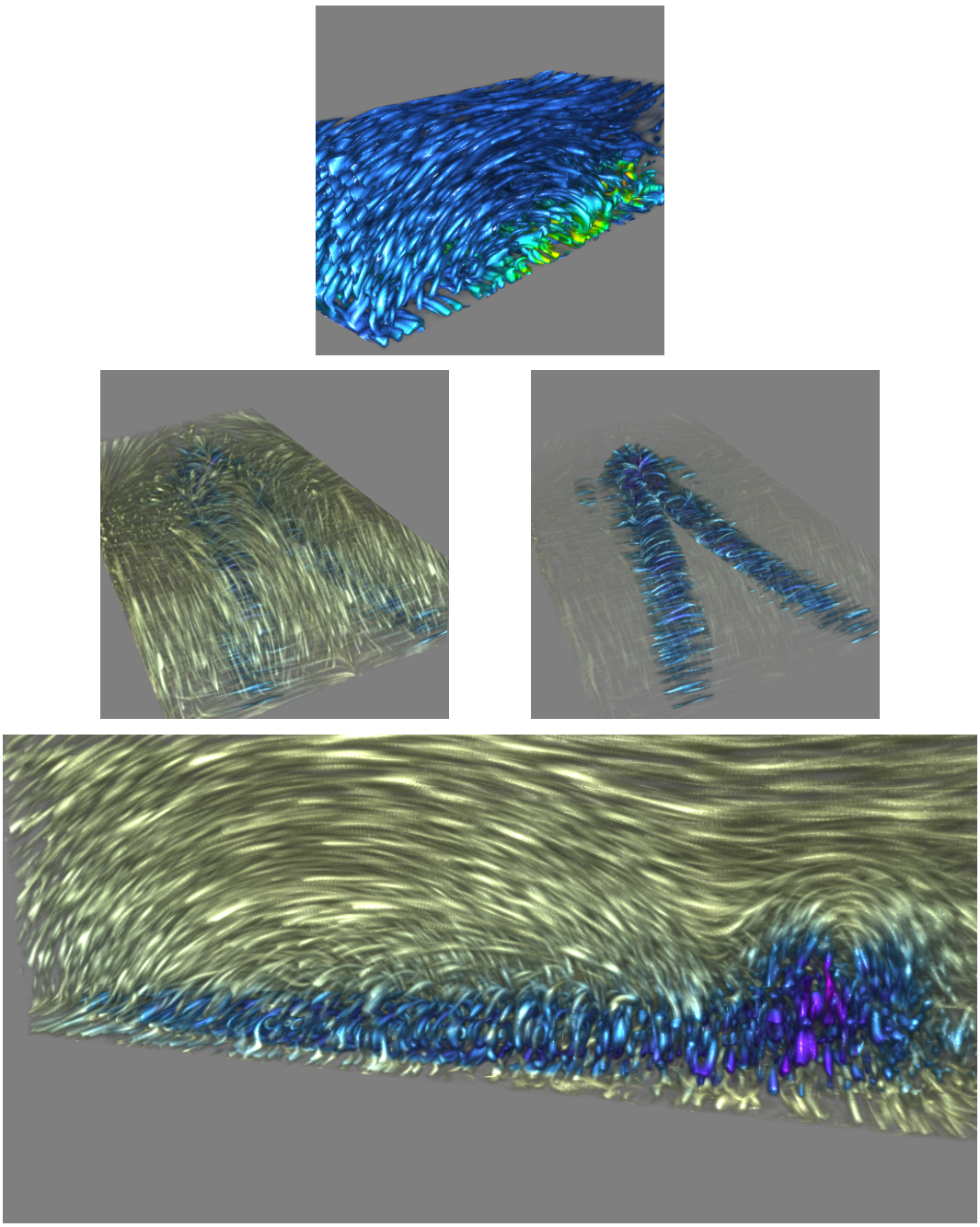


Abbildung 7.13: 6603-small: In der ersten Reihe wird eine Clipping-Ebene verwendet, um das Innere des Datensatzes zu visualisieren. Am unteren Rand ist einer der zwei Wirbel zu erkennen. Bei den Bildern der mittleren Reihe wird eine andere Transferfunktion eingesetzt und beim rechten Bild zusätzlich Velocity-Masking eingesetzt. Für das Bild in der untersten Reihe wurde eine Clipping-Ebene entlang eines Wirbels platziert.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein neuer Ansatz zur Berechnung des volumetrischen LIC vorgestellt. Dabei handelt es sich um ein GPU-basiertes Verfahren. Der 3D LIC wurde dazu mittels verschiedener Techniken berechnet. Das Ray-Casting und das Slicing aus dem Bereich der Volumenvisualisierung wurden an die Berechnung des LIC angepasst. Die Integration des LIC erfolgt dabei vollständig im Fragmentprogramm. Zur Beschleunigung wurden Verfahren wie der vorzeitige Strahlabbruch, das Überspringen von leerem Raum und der Early-Z-Test verwendet.

Zur Bewältigung der visuellen Komplexität, die sich durch den Wechsel der Dimension von 2D auf 3D ergibt, wurden verschiedene Lösungsansätze vorgestellt. Dazu wurde die Transferfunktion um einen weiteren Kanal erweitert. Dieser Kanal dient dazu, die berechnete Intensität des LIC auf einen Opazitätswert abzubilden. Die ermöglicht es, Bereiche in denen die LIC-Intensität sehr gering ist auszublenden. Wird für diese Opazitätswerte ein Wert verwendet, der geringfügig größer als null ist, führt dies zu einem verstärkten Tiefeneindruck.

Erstmalig wurden für die Beleuchtung der Stromlinien die Gradienten des Rauschens verwendet. Aus diesen Gradienten wird im Integrationsschritt des LIC ein neuer Gradient berechnet. Dieser wird in der nachfolgenden Beleuchtung als Normale verwendet. Durch die Verwendung geeigneter Rauschtexturen ergeben sich so Stromlinienbündel mit glatter Oberfläche und den entsprechenden Normalen. Zum Vergleich dazu wurden auch die Modelle zur Beleuchtung von Stromlinien von Zöckler et al. und Mallo et al. implementiert.

Bei der Berechnung des Gradienten während der Integration des LIC wurden die Annahmen getroffen, dass der Gradient des Filters konstant ist und es sich um gerade Stromlinien handelt. Die Annahme, dass Stromlinien gerade sind, ist in turbulenten Strömungen nicht gültig. Wenn sich die Stromlinie krümmt, müsste der Gradient entsprechend angepasst werden.

Das Clipping mit den Hardware-Clip-Ebenen ist für das Ray-Casting nicht immer korrekt. Wird eine Clipebene entgegen der Blickrichtung verwendet, werden die Strahlen des Ray-Castings nicht an dieser Clip-Ebene sondern erst an der Boundingbox des Volumens abgebrochen. Eine mögliche Lösung wäre, nicht die Boundingbox für den Strahlabbruch zu verwenden sondern eine beliebige Geometrie.

Zur Verbesserung der Wahrnehmung wäre die Animation des 3D LIC denkbar, um die Strömungsrichtung besser darstellen zu können. Interessant wäre auch, die Berechnung des 3D LIC auf der GeForce 8800 von nVidia durchzuführen. Da diese Grafikkarte über 128 Pipelines verfügt dürfte sich dies positiv auf die Gesamtperfomance auswirken.

Literaturverzeichnis

[1] Arvo, J. R.; Kirk, D. B.: *Particle Transport and Image Synthesis*. In: Computer Graphics (ACM SIGGRAPH), Seiten 63–66. 1990

[2] Banks, D. C.: *Illumination in diverse codimensions*. In: Proc. ACM SIGGRAPH, Seiten 327–334. 1994

[3] Blinn, J. F.: *Models of light reflection for computer synthesized pictures*. In: Proc. ACM SIGGRAPH, Seiten 192–198. 1977

[4] Cabral, B.; Leedom, L. C.: *Imaging Vector Fields Using Line Integral Convolution*. In: Proc. ACM SIGGRAPH, Seiten 263–270. 1993

[5] Carr, N. A.; Hall, J. D.; Hart, J. C.: *The ray engine*. In: Proc. ACM SIGGRAPH/EUROGRAPHICS, Seiten 37–46. 2002

[6] Cline, H. E.; Ludke, S.; Lorensen, W. E.: *Dividing cubes system and method for the display of surface structures contained within the interior region of a solid body*. United States Patent No. 4,719,585, 1988

[7] Danskin, J.; Hanrahan, P.: *Fast algorithms for volume ray tracing*. Workshop on Volume Visualization, Seiten 91–98. 1992

[8] de Leeuw, W.; van Liere, R.: *Comparing LIC and spot noise*. In: Proc. IEEE Visualization, Seiten 359–365. 1998

[9] Diepstraten, J.; Weiskopf, D.; Ertl, T.: *Transparency in interactive technical illustrations*. Computer Graphics Forum, Vol. 21.3, Seiten 317–317. 2002

[10] Engel, K.; Hadwiger, M.; Kniss, J. M.; Lefohn, A. E.; Salama, C. R.; Weiskopf, D.: *Real-time volume graphics*. In: ACM SIGGRAPH 2004 Course Notes. 2004

[11] Everitt, C.: *Interactive order-independent transparency*. nVidia White Paper. 2001. URL `http://developer.nvidia.com/object/Interactive_Order_Transparency.html`

[12] Foley, J. D.; van Dam, A.; Feiner, S. K.; Hughes, J. F.: *Computer Graphics — Principles and Practice*. 2. Auflage Addision-Wesley 1990

[13] Forssell, L.: *Visualizing flow over curvilinear grid surfaces using line integralconvolution*. In: Proc. IEEE Visualization, Seiten 240–247. 1994

[14] Forssell, L.; Cohen, S.: *Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows*. In: IEEE Transactions on Visualization and Computer Graphics, Seiten 133–141. 1995

[15] Hege, H.; Stalling, D.: *Fast LIC with Piecewise Polynomial Filter Kernels*. Mathematical Visualization, Seiten 295–314. 1998

[16] Helgeland, A.; Andreassen, O.: *Visualization of vector fields using seed LIC and volume rendering*. IEEE Transactions on Visualization and Computer Graphics, Vol. 10.6, Seiten 673–682. 2004

[17] Herman, G.; Liu, H.: *Display of three-dimensional information in computed tomography.* J Comput. Assisted Tomograph, Vol. 1.1, Seiten 155–160. 1977

[18] Herman, G. T.; Liu, H. K.: *3D display of human organs from computed tomograms*. Computer Graphics Image Processing, Vol. 9.1, Seiten 1–21. 1979

[19] Interrante, V.; Grosch, C.: *Strategies for Effectively Visualizing 3D Flow with Volume LIC*. In: Proc. IEEE Visualization, Seiten 421–424. 1997

[20] Interrante, V.; Grosch, C.: *Visualizing 3D flow*. IEEE Computer Graphics & Applications, Vol. 18.4, Seiten 49–53. 1998

[21] Jeong, J.; Hussain, F.: *On the identification of a vortex*. Journal of Fluid Mechanics Digital Archive, Vol. 285, Seiten 69–94. 1995

[22] Kajiya, J. T.; Herzen, B. P. V.: *Ray tracing volume densities*. In: ACM SIGGRAPH, Seiten 165–174. 1984

[23] Keller, A.: *The Fast Calculation of Form Factors Using Low Discrepancy Sequences*. In: Proc. Spring Conference on Computer Graphics (SCCG 96), Seiten 195–204. 1996

[24] Keller, A.: *Instant radiosity*. Computer Graphics, Vol. 31.Annual Conference Series, Seiten 49–56. 1997

[25] Keller, A. Strictly deterministic sampling methods in computer graphics. Technical report, Mental Images, 2001

[26] Kenwright, D. N.; Mallinson, G. D.: *A 3-D streamline tracking algorithm using dual stream functions*. In: Proc. IEEE Visualization, Seiten 62–68. 1992

[27] Kiu, M.-H.; Banks, D. C.: *Multi-frequency noise for LIC*. In: Proc. IEEE Visualization, Seiten 121–126. 1996

[28] Klein, T.; Strengert, M.; Stegmaier, S.; Ertl, T.: *Exploiting Frame-to-Frame Coherence for Accelerating High-Quality Volume Raycasting on Graphics Hardware*. In: Proc. IEEE Visualization, Seite 29. 2005

[29] Kramer, D.; Kaufman, L.; Guzman, R.; Hawryszko, C.: *A general algorithm for oblique image reconstruction*. IEEE Computer Graphics and Applications, Vol. 10.2, Seiten 62–65. 1990

[30] Krüger, J.; Westermann, R.: *Acceleration Techniques for GPU-based Volume Rendering*. In: Proc. IEEE Visualization, Seiten 287–292. 2003

[31] Lacroute, P.; Levoy, M.: *Fast Volume Rendering Using a Shear–Warp Factorization of the Viewing Transformation*. In: Proc. ACM SIGGRAPH, Seiten 451–458. 1994

[32] Lakare, S.; Kaufman, A.: *Light Weight Space Leaping using Ray Coherence*. In: Proc. IEEE Visualization, Seiten 19–26. 2004

[33] Laramee, R. S.; Hauser, H.; Doleisch, H.; Vrolijk, B.; Post, F. H.; Weiskopf, D.: *The state of the art in flow visualization: Dense and texture-based techniques*. Computer Graphics Forum, Vol. 23.2, Seiten 143–161. 2004

[34] Levoy, M.: *Efficient ray tracing of volume data*. ACM Transactions on Graphics, Vol. 9.3, Seiten 245–261. 1990

[35] Li, G.; Tricoche1, X.; Hansen, C.: *GPUFLIC: Interactive and Accurate Dense Visualization of*

Unsteady Flows. In: Proc. Eurovis 2006 (EG / IEEE VGTC Symposium on Visualization). 2006

[36] Li, W.; Mueller, K.; Kaufman, A.: *Empty Space Skipping and Occlusion Clipping for Texture-based Volume Rendering*. In: Proc. IEEE Visualization, Seiten 317–324. 2003

[37] Lorensen, W. E.; Cline, H. E.: *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. In: Computer Graphics (ACM SIGGRAPH), Seiten 163–169. 1987

[38] Mallo, O.; Peikert, R.; Sigg, C.; Sadlo, F.: *Illuminated Lines Revisited*. In: Proc. IEEE Visualization, Seite 3. 2005

[39] Max, N.; Crawfis, R.; Williams, D.: *Visualizing wind velocities by advecting cloud textures*. In: Proc. IEEE Visualization, Seiten 179–184. 1992

[40] Mosher, C.; van Hook, T.: *A geometric approach for rendering volume data*. In: Proc. National Computer Graphics Association (NCGA), Seiten 184–193. 1990

[41] Pfister, H.; Hardenbergh, J.; Knittel, J.; Lauer, H.; Seiler, L.: *The VolumePro real-time ray-casting system*. In: Proc. ACM SIGGRAPH, Seiten 251–260. 1999

[42] Phong, B. T.: *Illumination for computer generated pictures*. Communications of the ACM, Vol. 18.6, Seiten 311–317. 1975

[43] Porter, T.; Duff, T.: *Compositing digital images*. In: Proc. ACM SIGGRAPH, Seiten 253–259. 1984

[44] Purcell, T. J.; Buck, I.; Mark, W. R.; Hanrahan, P.: *Ray tracing on programmable graphics hardware*. In: Proc. ACM SIGGRAPH, Seiten 703–712. 2002

[45] Rezk-Salama, C.; Hastreiter, P.; Teitzel, C.; Ertl, T.: *Interactive exploration of volume line integral convolution based on 3D-texture mapping*. In: Proc. IEEE Visualization, Seiten 233–240. 1999

[46] Rhodes, M.; Glenn, W.; Azaawi, Y.: *Extracting oblique planes from serial ct sections.* J Comput. Assisted Tomograph, Vol. 4.5, Seiten 649–657. 1980

[47] Röttger, S.; Guthe, S.; Weiskopf, D.; Ertl, T.; Strasser, W.: *Smart hardware-accelerated volume rendering*. In: Proc. EG / IEEE TCVG Symposium on Visualisation, Seiten 231–238. 2003

[48] Schulze, J. P.; Kraus, M.; Lang, U.; Ertl, T.: *Integrating pre-integration into the shear-warp algorithm*. In: Proc. EG / IEEE TVCG Workshop on Volume graphics, Seiten 109–118. 2003

[49] Schussman, G.; Ma, K.-L.: *Anisotropic Volume Rendering for Extremely Dense, Thin Line Data*. In: Proc. IEEE Visualization, Seiten 107–114. 2004

[50] Segal, M.; Akeley, K.: *The OpenGL graphics system: A specification (version 2.0)*, 2004. URL `http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf`

[51] Shen, H.-W.; Johnson, C. R.; Ma, K.-L.: *Visualizing Vector Fields Using Line Integral Convolution and Dye advection*. In: Proc. Volume Visualization Symposium, Seiten 63–70. 1996

[52] Shen, H.-W.; Kao, D. L.: *UFLIC: a line integral convolution algorithm for visualizing unsteady flows*. In: Proc. IEEE Visualization, Seiten 317–322. 1997

[53] Stalling, D.; Hege, H.-C.: *Fast and resolution independent line integral convolution*. In: Proc. ACM SIGGRAPH, Seiten 249–256. 1995

[54] Stegmaier, S.; Strengert, M.; Klein, T.; Ertl, T.: *A Simple and Flexible Volume Rendering Framework for Graphics-Hardware-based Raycasting*. In: Proc. EG / IEEE VGTC Workshop on Volume Graphics 2005, Seiten 187–195. 2005

[55] Stytz, M. R.; Frieder, G.; Frieder, O.: *Three-dimensional medical imaging: Algorithms and computer systems*. ACM Computing Surveys, Vol. 23.4, Seiten 421–499. 1991

[56] Suzuki, Y.; Fujishiro, I.; Chen, L.; Nakamura, H.: *Case Study: Hardware-Accelerated Selective LIC Volume Rendering*. In: Proc. IEEE Visualization, Seiten 485–488. 2002

[57] Tufte, E. R.: *The Visual Display of Quantitative Information*. Graphics Press 1983

[58] Urness, T.; Interrante, V.; Marusic, I.; Longmire, E.; Ganapathisubramani, B.: *Effectively Visualizing Multi-Valued Flow Data using Color and Texture*. In: Proc. IEEE Visualization, Seiten 115–121. 2003

[59] van Wijk, J.: *Spot noise texture synthesis for data visualization*. In: Computer Graphics (Proc. ACM SIGGRAPH), Seiten 309–318. 1991

[60] Wegenkittl, R.; Gröller, E.; Purgathofer, W.: *Animating Flow Fields: Rendering of Oriented Line Integral Convolution*. In: Proc. Computer Animation, Seiten 15–21. 1997

[61] Weiskopf, D.; Engel, K.; Ertl, T.: *Interactive clipping techniques for texture-based volume visualization and volume shading*. IEEE Transactions on Visualization and Computer Graphics, Vol. 9.3, Seiten 298–312. 2003

[62] Weiskopf, D.; Schafhitzel, T.; Ertl, T.: *Real-Time Advection and Volumetric Illumination for the Visualization of 3D Unsteady Flow*. In: Proc. Eurovis 2005 (EG / IEEE VGTC Symposium on Visualization), Seiten 13–20. 2005

[63] Weiskopf, D.; Schafhitzel, T.; Ertl, T.: *Texture-based visualization of 3d unsteady flow by real-time advection and volumetric illumination*. Angenommen bei IEEE Transactions on Visualization and Computer Graphics. 2006

[64] Westermann, R.; Sevenich, B.: *Accelerated volume ray-casting using texture mapping*. In: Proc. IEEE Visualization, Seiten 271–278. 2001

[65] Westover, L.: *Footprint evaluation for volume rendering*. In: Proc. ACM SIGGRAPH, Seiten 367–376. 1990

[66] Woo, M.; Neider, J.; Davis, T.; OpenGL Architecture Review Board,: *The Official Guide to Learning OpenGL, Version 1.1*. Addison-Wesley 1997

[67] Xue, D.; Zhang, C.; Crawfis, R.: *iSBVR: Isosurface-aided Hardware Acceleration Techniques for Slice-Based Volume Rendering*. In: Proc. EG / IEEE VGTC Workshop on Volume Graphics 2005, Seiten 207–215. 2005

[68] Zheng, X.; Pang, A.: *HyperLIC*. In: Proc. IEEE Visualization, Seiten 249–256. 2003

[69] Zöckler, M.; Stalling, D.; Hege, H.-C.: *Interactive visualization of 3D-vector fields using illuminated stream lines*. In: Proc. IEEE Visualization, Seiten 107–113. 1996

Erklärung

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

_____

(Martin Falk)