Institut für Bildinformatik
Computergraphik und Multimediasysteme
Universität Siegen

Interaktive Flow-Visualisierung anhand geometrischer Primitive mit adaptiven Zeitschrittverfahren

Diplomarbeit im Fach Informatik

vorgelegt von

Albert Pritzkau

Geboren am 16. Januar 1978 in Paderborn, Deutschland

Angefertigt am

Institut für Bildinformatik
Computergraphik und Multimediasysteme
Fachbereich 12
Universität Siegen

Betreuer: Dipl.-Inf. Nicolas Cuntz (Universität Siegen, Computergraphik und Multimediasysteme)
Erstgutachter: Prof. Dr. A. Kolb (Universität Siegen, Computergraphik und Multimediasysteme)
Zweitgutachter: Dr. C. Rezk-Salama (Universität Siegen, Computergraphik und Multimediasysteme)

Beginn der Arbeit: 05. November 2007

Abgabe der Arbeit: 03. April 2008

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Siegen, den 03. April 2008

Übersicht

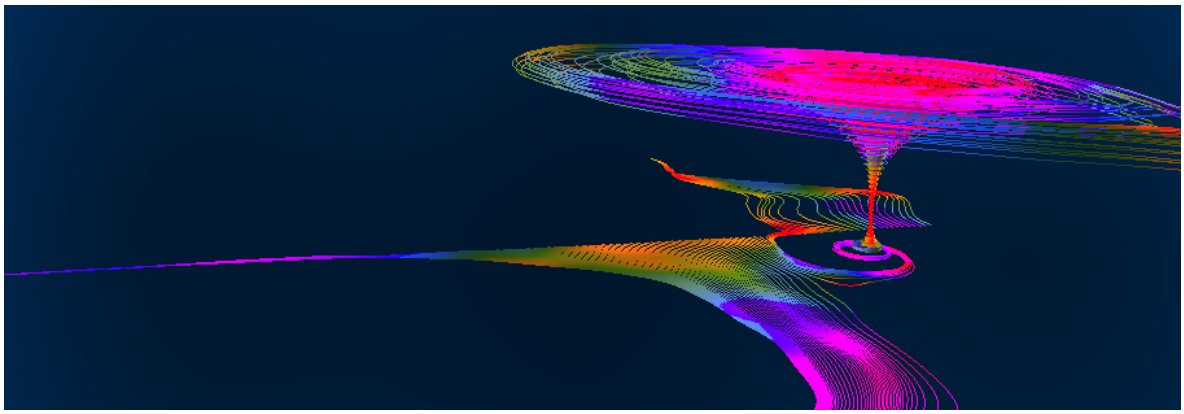


Bild 1: Geometry Shader-basierte Streamlines mit adaptiver Integrationsschrittweite und einer einfachen Beleuchtungsberechnung

Das Gebiet Strömungsvisualisierung hat sich als fester Bestandteil der Forschung im Bereich der Visualisierung etabliert. Bei der Umsetzung lässt sich ein Trend zunehmender Nutzung moderner Graphikhardware über eine bloße Szenendarstellung hinaus verzeichnen. Der Erfolg einer derartigen Strategie setzt sowohl eine ganz bestimmte Struktur des zugrunde liegenden Berechnungsgegenstandes, als auch ein entsprechendes Programmiermodell voraus.

Im Kontext der Strömungsvisualisierung kann man dabei schon auf eine relativ lange Tradition zurückblicken. Die GPU-basierte Visualisierung von Strömungsdaten durch Strömungslinien bildet das Zentrum dieser Arbeit. Der Einsatz komplexer Integrationsverfahren zur Ermittlung diskreter Linienpunkte als Grundlage der Visualisierung führt zunächst zu einer entscheidenden Rechenbelastung. Durch die Parallelisierung einzelner Teilprozesse, die optimiert auf der Graphikhardware ausgeführt werden, wird dieser Ablauf wesentlich beschleunigt. Dabei bildet die Generierung von Stream- und Streaklines den Ansatzpunkt der Betrachtung. Unter der Prämisse der allgemeinen Anforderungen an Visualisierungssysteme, können bisherige Ansätze hinsichtlich der Effizienz sowie Präzision klassifiziert werden.

Generell ergeben sich Strömungslinien aus der Integration einzelner Partikelpositionen, die in das Strömungsfeld injiziert werden. Für eine GPU-basierte Realisierung werden Texturen zur Speicherung der ermittelten Partikelpositionen eingesetzt. Zur Darstellung werden die einzelnen Partikelpositionen dann in entsprechender Reihenfolge zu einer Linie zusammengefasst. Die unmittelbare Umsetzung einer derartigen Konfiguration für Stream- und Streakline ermöglicht hinsichtlich der Effizienz beeindruckende Ergebnisse.

Unter Berücksichtigung des Leistungsspektrums moderner Graphikhardware können verschiedene Aspekte der Anpassung bisheriger Vorgehensweisen identifiziert werden. So ermöglicht die Verwendung des Geometry Shaders (GS) als relativ neue Stufe innerhalb der programmierbaren

Rendering-Pipeline die dynamische Erzeugung einer variablen Anzahl von Ausgabeprimitiven. Diese Eigenschaft kann zur Flexibilisierung sowohl der Konstruktion als auch der Darstellung von Strömungslinien eingesetzt werden.

Bezüglich der Stabilität eingesetzter Integrationsverfahren erweist sich der Einsatz einer adaptiven Integrationsschrittweite als entscheidendes Kriterium. Die Umsetzung einer entsprechenden Fehlerkontrolle stellt den Hauptbeitrag dieser Arbeit dar. Insbesondere bei Streamlines wird die Implementierung durch den Einsatz des GS begünstigt.

Darüber hinaus werden anhand einer mehrdimensionalen Transferfunktion Ansätze eines merkmalsbasierten Visualisierungsverfahrens integriert, welche die Selektion bestimmter Strömungsbereiche ermöglicht. Zuletzt werden die ermittelten Linienprimitive um weitere visuelle Attribute ergänzt und erlauben so auch die Berücksichtigung lokaler Vektorfeldeigenschaften.

Danksagungen

An dieser Stelle möchte ich mich bei all denen bedanken, die mich bei der Anfertigung dieser Diplomarbeit unterstützt haben.

Ein ganz besonderer Dank gilt Herrn Prof. Dr. Andreas Kolb und Dr. Christof Rezk-Salama der Fachgruppe Computergraphik und Multimediasysteme an der Universität Siegen für ihre hilfreichen Anregungen bei der Konstruktion einer Aufgabenstellung, sowie die Bereitschaft die Arbeit als Gutachter zu betreuen. Weiterhin möchte ich mich bei Dipl.-Inf. Nicolas Cuntz für die Betreuung meiner Arbeit bedanken, der mir in der Phase der Implementierung eine große Hilfe war und viele wertvolle Hinweise gegeben hat. Des weiteren möchte ich mich bei meiner Nichte Julia Pritzkau, und meinen Geschwistern Artur Pritzkau und Victor Penner für das Korrekturlesen, die konstruktive Kritik und das rege Interesse bedanken. Ein herzliches Dankeschön geht an meine Freundin Barbara Hench, die mich während der gesamten Diplomarbeitsphase moralisch unterstützt hat und mir bei Formulierungsproblemen zur Seite stand.

Nicht zuletzt möchte ich mich bei meinen Eltern und der ganzen Familie für ihre Liebe, ihr Vertrauen und den bedingungslosen Rückhalt bedanken, den ich immer wieder bei ihnen finden konnte.

Albert Pritzkau

Vorwort

Die vorliegende Arbeit beschreibt die Ergebnisse meiner Diplomarbeit, welche an der Fachgruppe für Computergraphik und Multimediasysteme unter der Leitung von Prof. Dr. Andreas Kolb der Universität Siegen erstellt wurde. Unter ständiger Betreuung von Nicolas Cuntz, ebenfalls von der Fachgruppe für Computergraphik und Multimediasysteme der Universität Siegen, wurde die Aufgabenstellung entwickelt und ausgeführt.

Diese Arbeit ist in 6 Kapitel unterteilt. Nach einer Einleitung in das aktuelle Thema der Strömungsvisualisierung folgt im ersten Kapitel eine Beschreibung der zugrunde liegenden Motivation dieser Arbeit.

Im darauf folgenden Kapitel werden hinsichtlich der Visualisierung anhand von Strömungslinien zunächst grundlegende Aspekte der Realisierung in theoretischer und praktischer Hinsicht beschrieben. Dazu werden Merkmale eingesetzter Algorithmen und Hardware aufgrund verwandter Beiträge sowie Hardwarespezifikationen evaluiert.

Als Resultat der gewonnenen Erkenntnisse, werden in Kapitel 3 konkrete Ansätze zur Visualisierung von Stream- und Streaklines dargestellt. Die Strömungslinien ergeben sich grundsätzlich durch eine geeignete Verbindung zuvor ermittelter Partikelpositionen. Ausgehend von der Vorstellung eines Partikelsystems zur Steuerung der Partikelpositionierung und -bewegung, wird zunächst jeweils ein Index-basiertes Verfahren vorgestellt. Man beschränkt sich bei der Realisierung zunächst auf die traditionellen Funktionalitäten der programmierbaren Rendering-Pipeline. Darüber hinaus werden dann jeweils sogenannte Geometry Shader-basierte Ansätze präsentiert, die sich aktuelle Erweiterungen moderner Graphikhardware zunutze machen. Im Zuge der Konstruktion der Linienprimitive wird eine angemessene Fehlerkontrolle der Partikelbewegung berücksichtigt, was sich entscheidend auf die Präzision der resultierenden Repräsentation auswirkt.

In Kapitel 4 werden einzelne Implementierungsaspekte wie die Beschreibung des zugrunde liegenden Framework dargestellt. Dazu zählen beispielsweise auch grundlegende Algorithmen und Vorgehensweisen.

In Kapitel 5 wird ein Vergleich der eingesetzten Verfahren vor dem Hintergrund der allgemeinen Anforderungen an Visualisierung diskutiert.

Zuletzt werden in Kapitel 6 weiterführende Ansätze angesprochen, welche jedoch über das Spektrum dieser Arbeit hinausgehen. Im Anhang befinden sich Informationen zum Datensatz, der im Rahmen dieser Arbeit als Grundlage diente, sowie verwendete Formeln.

Abkürzungen

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
CFD	Computational Fluid Dynamics, numerische Strömungsmechanik
CPU	Central Processing Unit, Prozessor auf dem Mainboard
FBO	Frame Buffer Object
FS	Fragment Shader
GLew	OpenGL Extension Wrangler Library
GPGPU	General-Purpose computation on GPUs
GPU	Graphics Processing Unit, Prozessor auf der Graphikkarte
GS	Geometry Shader
LIC	Line Integral Convolution
LOD	Level Of Detail
MDTF	Multidimensionale Transferfunktion
PBO	Pixel Buffer Object
RGB	Red, Green, Blue
RGBA	Red, Green, Blue, Alpha
RK	Runge-Kutta Integration
SIMD	Single-Instruction, Multiple-Data
TF	Transformation Feedback
VBO	Vertex Buffer Object
VS	Vertex Shader

Inhaltsverzeichnis

1 Motivation **4**

1.1 Anforderungen an Visualisierungssysteme 4
1.2 Strömungsvisualisierung 5
1.3 Beitrag der Diplomarbeit 6

2 Grundlagen **8**

2.1 Differenzierung 8
2.2 Partikelsysteme 9
2.2.1 Integrationsverfahren 10
2.2.2 Zeitschrittadaptivität 11
2.3 Strömungsprimitive 12
2.3.1 Strömungslinien 12
2.3.2 Stream Balls, Stream Ribbons, Stream Tubes 13
2.3.3 Strömungsflächen , -volumina 14
2.4 Einfache Beleuchtungsberechnung für Strömungslinien 14
2.5 GPUs und GPGPU 15
2.5.1 Traditionelle Graphik-Pipeline (VS, GS, FS) 16
2.5.2 Geometry Shader 17
2.5.3 Stream-Output/Transform-Feedback 17
2.5.4 Texturen, FBOs 18
2.5.5 GPGPU 18
2.6 GPU-basierte Partikelsysteme 19
2.6.1 Graphikspeicher 20
2.6.2 Initialisierung 21
2.6.3 Integration 22
2.6.4 Rendering 22
2.7 Transfer-Funktionen 23

3 GPU-basierte Strömungslinien **27**
3.1 Problemdifferenzierung . . . 27
3.2 Index-basierter Ansatz . . . 28
3.2.1 Streaklines . . . 29
3.2.2 Streamlines . . . 31
3.3 Zeitschrittadaptivität . . . 32
3.3.1 Streamlines . . . 32
3.3.2 Streaklines . . . 34
3.3.3 Preprocessing der Daten . . . 35
3.4 Geometry-Shader-basierte Streamlines . . . 36
3.5 Geometry-Shader-basierte Streakline . . . 38
3.6 Darstellung . . . 40
3.6.1 Stylized Line . . . 40
3.6.2 Visualisierung von Strömungseigenschaften . . . 42

4 Implementierungsaspekte **45**
4.1 Softwareumgebung . . . 45
4.2 Geometry Shader/Transform Feedback . . . 47

5 Resultate **49**
5.1 Anwendungsszenarien . . . 49
5.1.1 Analytisch generierte Strömungsfelder . . . 49
5.1.2 Typhoon-Datensatz . . . 50
5.2 Effizienz-Messungen . . . 50
5.2.1 Texturzugriffe . . . 51
5.2.2 Geometry Shader/Transform Feedback . . . 53
5.2.3 Adaptiver Integrationszeitschritt . . . 54
5.3 Präzision der Runge-Kutta-Integration . . . 55
5.3.1 Visueller Vergleich von Strömungslinien mit und ohne Zeitschrittadaptivität . . . 55

6 Ausblick **57**
6.1 Kurveninterpolation . . . 57

A Formeln und Codefragmente **59**
A.1 Das klassische Runge-Kutta Verfahren 4. Ordnung . . . 59
A.2 Das Phong-Modell . . . 59
A.3 Einsatz des Transform Feedback Buffers . . . 60

B Datensatz **61**
B.1 Typhoon-Datensatz 61
B.1.1 Höhenschichten 61
B.1.2 Gitterpunkte 62

Verzeichnis der Bilder **64**

Verzeichnis der Tabellen **66**

Listings **67**

Literaturverzeichnis **68**

Kapitel 1

Motivation

Den Ausgangspunkt jeglicher Visualisierung stellt stets eine bestimmte Datenmenge dar. Um bestimmte Phänomene zu erkennen oder nachzuvollziehen, erhofft man sich durch die visuelle Darstellung einen essentiellen Vorteil gegenüber der rein numerischen. Dazu werden prinzipiell einzelne Aspekte der Daten extrahiert und auf geometrische Objekte abgebildet. Aus diesem Vorgehen resultiert eine Herausforderung, welche maßgeblich durch die Komplexität zugrunde liegender Daten und Algorithmen bedingt wird. Die wissenschaftliche Forschung im Bereich der Computergraphik und eine rasant steigende Leistungsfähigkeit entsprechender Hardware korreliert mit der nachhaltigen Identifikation weiterer Einsatzgebiete, deren Verfahren aufgrund dieser Entwicklung evaluiert und verbessert werden können. Eine ständig steigende Datenmenge steht dabei einer deutlich mäßigeren Entwicklung der zur Berechnung verwendeten Hardware gegenüber, und erfordert im Vorfeld schon eine Optimierung, sowohl der Berechnung als auch der Darstellung betreffender Phänomene. Vor diesem Hintergrund zählt die Visualisierung von dicht besetzten 3D Vektorfeldern zu den Hauptschwierigkeiten der wissenschaftlichen Visualisierung. Mit der Verdeckung als Hauptproblem gewinnen neben dicht besetzten Repräsentationen, wie sie beispielsweise durch 3D LIC-Verfahren vertreten werden, auch Verfahren für schwach besetzte Darstellungen an Bedeutung. Insbesondere im Kontext von 3D Vektorfeldern stellen dabei Linienprimitive aussagekräftige Mittel zur Abbildung von Strömungseigenschaften dar. Mit diesem Beitrag werden dazu existierende Verfahren zur Visualisierung unter der Prämisse aktueller Graphikhardware untersucht. Vor allem bei der Ermittlung der Kontrollpunkte entsprechender Linien sowie bezüglich der Darstellung weiterer Strömungseigenschaften ergeben sich eine Reihe von Einsatzmöglichkeiten, die vorgestellt und evaluiert werden.

1.1 Anforderungen an Visualisierungssysteme

Mit dem Ziel der Darstellung durchlaufen die Rohdaten eine Reihe von Verarbeitungsschritten, welche anhand einer so genannten Visualisierungspipeline veranschaulicht werden können. Aus Ansprüchen der wissenschaftlichen Tauglichkeit ergeben sich eine Reihe von Anforderungen, die sich im Wesentlichen auf die Prämissen der eingesetzten Verfahren in den einzelnen Schritten beziehen. Die Ein-

gabe der Visualisierungspipeline bilden also Rohdaten, welche aus analogen Messungen, komplexen Modellbeschreibungen oder aus Simulationen akquiriert wurden, und meist einen mehrdimensionalen Definitionsbereich beschreiben. Eine steigende Komplexität bezüglich räumlicher und zeitlicher Abtastfrequenz sowie steigender Detailgrad auf der einen Seite fördert auf der anderen Seite den Einsatz beliebig komplizierter Verfahren. Unter dem Gesichtspunkt der Effizienz dienen beide Seiten als Ansatzpunkte möglicher Optimierungen, was im Kontext interaktiver Systeme ein besonderes Gewicht bekommt. Je nach Fragestellung der Visualisierung sind beispielsweise nicht zwangsläufig alle Bestandteile des realen Modells relevant und erlauben daher eine entsprechende Abstraktion. In jedem Fall müssen aber alle notwendigen Zusammenhänge zur Interpretation eines Sachverhalts dargestellt werden. Um nachstehende Verarbeitungsschritte weiter zu beschleunigen, können darüber hinaus ergänzende Ableitungen ermittelt werden (Filtering). Zur Auswahl einer geeigneten Darstellung der aufbereiteten Daten, stehen entsprechende Primitive sowie zusätzliche Attribute zur Verfügung. Die Effektivität einer Repräsentation ist ausschlaggebendes Qualitätskriterium, und basiert im Wesentlichen auf dem eigentlichen Ziel der Darstellung. Im Kontext wissenschaftlicher Nutzung wird dieses durch eine angemessene Fehlerkontrolle eingesetzter Verfahren ergänzt. Schließlich können die erzeugten geometrischen Primitive in Bilddaten umgesetzt werden.

1.2 Strömungsvisualisierung

Die Grundlage zur Visualisierung von Strömungsvorgängen bilden (zeitabhängige) 2D- und 3D-Vektordaten. Diese ergeben sich aus numerischen Strömungssimulationen (CFD) beziehungsweise aus experimentellen Messungen. Beispielsweise basiert die aerodynamische Konstruktion von Flug- und Fahrzeugen heutzutage überwiegend auf computergestützten Simulationen. Zur Untersuchung komplexer, natürlicher Phänomene hingegen bezieht man sich heute in vielen Anwendungen auf experimentelle Messungen.

Zur Visualisierung der Vektordaten stehen eine Reihe unterschiedlicher Methoden zur Verfügung. Durch den Einsatz numerischer sowie graphischer Verfahren können geometrische Repräsentationen wie Pfeile, Glyphen, Partikel sowie Strömungslinien, -bänder und -röhren erzeugt werden. Die Konstruktion dieser Darstellung leitet sich in vielerlei Hinsicht aus der Vorstellung einer experimentellen Untersuchung des Strömungsfeldes ab, bei der eine bestimmte Substanz in das Strömungsfeld injiziert wird. Aufgrund der Advektion dieser Substanz lassen sich dann beispielsweise Strömungslinien erzeugen, die ein bestimmtes lokales Strömungsmuster veranschaulichen. Die Platzierung der Substanz hat dabei entscheidenden Einfluss auf das resultierende Ergebnis.

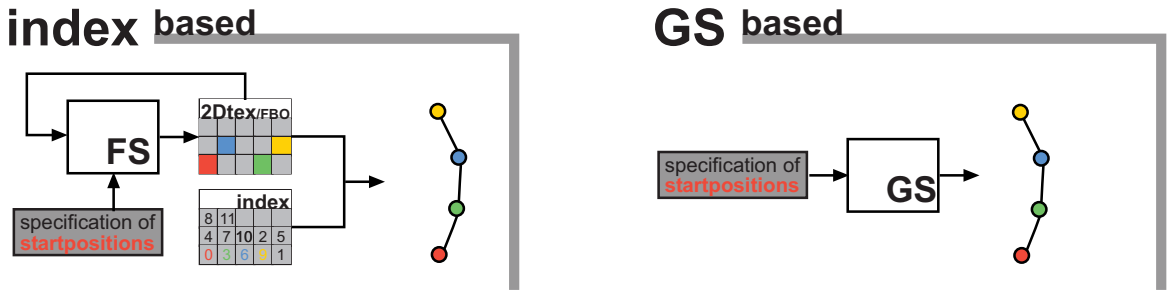
Eine Reihe weiterer Visualisierungsansätze für Vektordaten verwenden globale, bildbasierte Verfahren. Beispielsweise werden dazu Texturen im Vektorfeld angeordnet und entlang des Strömungsverlaufs gefiltert (LIC). Im Gegensatz zur geometrischen Abbildung generieren diese Verfahren eine kontinuierliche Repräsentation der gesamten Strömungsdaten und ermöglichen die Analyse globaler Vektorfeldeigenschaften. Auf diese Weise erübrigt sich die diskrete Platzierung und Advektion einzel-

ner Startpunkte der Untersuchung. Jedoch ergeben sich andererseits aufgrund der Informationsdichte dieser Methoden Verdeckungsprobleme, die für eine effektive Darstellung gelöst werden müssen. Für einen umfassenden Überblick dieser Verfahren sei auf [Lar04] verwiesen.

1.3 Beitrag der Diplomarbeit

Im Rahmen dieser Arbeit werden existierende Verfahren zur Visualisierung von Strömungsfeldern anhand von Strömungslinien dargestellt. Wesentliche Charakteristika dieser Verfahren werden als Basis zur Identifikation und Umsetzung von Optimierungen unter Verwendung aktueller Graphikhardware genutzt. Dafür werden zunächst Leistungsmerkmale und -anforderungen aktueller Graphikkarten (Shader Model 4.0) ermittelt, was die Basis für deren Einsatz im Rahmen der Visualisierung bildet.

Bei der Konstruktion der Strömungslinien werden unterschiedliche Strategien verfolgt. Sie weichen bezüglich der eingesetzten technischen Funktionalitäten von einander ab. In jedem Fall wird aber die Kontrolle der Partikelpositionen über ein Partikelsystem realisiert. Eine benutzerdefinierte Spezifikation der Startpositionen der Partikel erlaubt eine interaktive Einflussnahme auf den Prozess der Visualisierung. Durch die Realisierung der folgenden Konfigurationen auf der Graphikhardware kann die Berechnung wesentlich beschleunigt werden.



(a) (b)

Bild 1.1: Prinzipielle Vorstellung der unterschiedlichen Strategien.

Zur Speicherung der Strömungsdaten sowie der aktuellen Partikelpositionen dienen zunächst Speicherstrukturen der Graphikhardware. In Form von Texturen können diese sowohl die Eingabe als auch das Ergebnis von Berechnungen aktivierter Shaderprogramme bilden. Jedem Partikel wird innerhalb dieser Struktur eine eindeutige Position zugewiesen. Die Komponenten der Texturfragmente werden in diesem Zusammenhang als räumliche Dimensionen interpretiert. Zur Advektion werden Shaderprogramme verwendet, welche die Folgeposition aufgrund der Vektorfelddaten sowie eines festgelegten Integrationsschemas ermitteln. Als Ergebnis enthält eine Partikeltextur alle benötigten Informationen zur Darstellung von Strömungslinien. Zur Synthese der einzelnen Linienprimitive werden die betreffenden Partikelpositionen aufgrund ihrer Platzierung innerhalb der Partikeltextur identifiziert und in

entsprechender Reihenfolge zusammengefasst. Da die Partikel einer Strömungslinie generell nicht in sequentieller Reihenfolge vorliegen wird eine entsprechende Korrektur vorgenommen. Ein Indexbuffer beschreibt in diesem Zusammenhang die gewünschte Reihenfolge (vgl. Bild 1.1(a)).

Der Geometry Shader, als neue Stufe innerhalb der traditionellen Rendering-Pipeline, ermöglicht eine völlig neue Strategie. In einem Durchlauf der Pipeline können dynamisch Primitive erzeugt werden (vgl. Bild 1.1(b)). Das Partikelsystem kann aufgrund dieser Tatsache prinzipiell ohne den Umweg einer Partikeltextur realisiert werden. In der Praxis unterliegt der Einsatz des GS jedoch bislang einer Reihe von Beschränkungen. Anhand der konkreten Anwendung im Kontext der Visualisierung von Strömungslinien werden die Konditionen der Verwendung des GS erörtert. Verschiedene Strömungslinien stellen dabei unterschiedliche Anforderungen an die Ermittlung relevanter Partikelpositionen. Unter Berücksichtigung dieser Anforderungen werden zunächst Verfahren für Streak- beziehungsweise Streamlines umgesetzt. Aus der Evaluation ergibt sich schließlich ein differenziertes Nutzungspotential für die entsprechenden Strategien bezüglich der unterschiedlichen Strömungslinien.

Um den Ablauf der Berechnung zu kontrollieren, werden einzelne Berechnungsschritte zeitlich synchronisiert. In der Ausgangskonfiguration ist das Zeitintervall dieser Synchronisation an eine global festgelegte Schrittweite gekoppelt. Zur Ermittlung der Partikelpositionen werden Integrationsverfahren eingesetzt. Um eine gewisse Präzision der Darstellung zu gewährleisten, wird für die Integration der Partikelpositionen eine adaptive Integrationsschrittweite vorausgesetzt, was im Widerspruch zur bisherigen Konfiguration steht. Durch eine Trennung der zeitlichen Synchronisation des Berechnungsablaufs von einer globalen Integrationsschrittweite, wird der Einsatz einer variablen Integrationsschrittweite ermöglicht. Diese kann aus lokalen Vektorfeldeigenschaften abgeleitet werden.

Schließlich werden Ansätze zur Visualisierung weiterer Strömungseigenschaften auf Basis der Strömungslinien dargestellt und in das Gesamt-Framework integriert. Entsprechende Merkmale können im Zuge der Integration ermittelt werden und daraufhin auf bestimmte visuelle Attribute abgebildet werden. Um auch Abhängigkeiten zwischen einzelnen Eigenschaften zu diagnostizieren wird eine multidimensionale Transferfunktion vorgestellt, die interaktiv beeinflusst werden kann. Abschließend wird im Kontext des Renderings eine weitere Möglichkeit der Integration von Strömungseigenschaften dargestellt. Dabei wird die zugrunde liegende Strömungslinie zu einer Strömungsröhre erweitert, deren Radius als zusätzlicher Informationsträger herangezogen werden kann. Eine adäquate Beleuchtungsberechnung rundet das Gesamtbild dieser Arbeit ab.

Kapitel 2

Grundlagen

2.1 Differenzierung

Um das Thema dieser Arbeit genauer zu differenzieren, soll der Ausgangspunkt zunächst innerhalb des Feldes der Visualisierung kategorisiert werden. Dies setzt zunächst die Definition einer sachgerechten Klassifikation voraus. Ein entsprechendes Konzept wird von [Lar04] vorgeschlagen und soll im Rahmen dieser Arbeit als Bezug dienen.

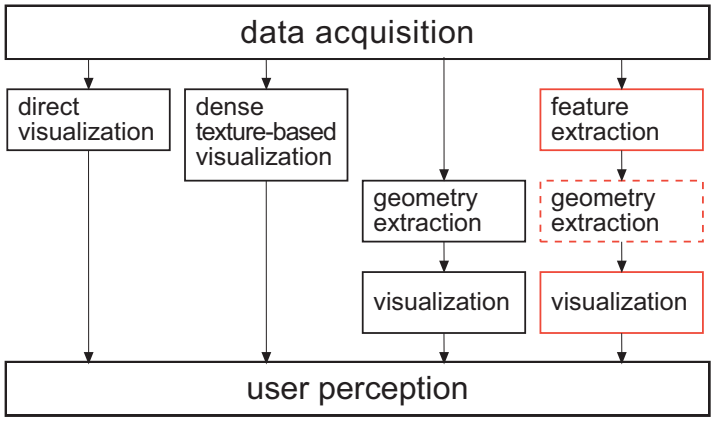


Bild 2.1: Die Graphik zeigt eine Klassifikation von Visualisierungsverfahren, wie sie von [Lar04] vorgeschlagen wird. Die rote Kennzeichnung beschreibt die grundsätzliche Vorgehensweise, die dieser Arbeit zugrunde liegt.

Grundsätzlich kann das Feld der Strömungsvisualisierung in vier verschiedene Methoden der Auswertung von Daten klassifiziert werden: direkte, texturbasierte, integrationsbasierte, sowie merkmalsbasierte Strömungsvisualisierung. Diese schließen sich nicht notwendigerweise gegenseitig aus, sondern ergänzen sich vielmehr. Die direkte Strömungsvisualisierung übersetzt die vorhandenen Strömungsdaten unmittelbar in bestimmte visuelle Informationsträger, wie dies beispielsweise bei einer Farbkodierung der Geschwindigkeit der Fall ist. Als weitere Informationsträger können Linien, Pfeile

oder auch Glyphen herangezogen werden. Diese werden üblicherweise an diskreten Abtastpunkten platziert und veranschaulichen Richtungsinformationen. Glyphen erweitern diese Darstellung durch die Einbeziehung weiterer Strömungseigenschaften.

Das Ziel der texturbasierten Strömungsvisualisierung ist es, eine möglichst dichte Repräsentation des Vektorfeldes zu erzeugen. Zu diesem Zweck werden hochfrequente Texturen im Vektorfeld angeordnet, um sie entlang der Strömungsrichtung zu advektieren. Durch die Anordnung der Texturen kann so das gesamte Vektorfeld abgedeckt werden.

Die geometrische oder auch integrationsbasierte Strömungsvisualisierung setzt üblicherweise einen Vorverarbeitungschritt zur Integration des Vektorfelds voraus. Das Resultat sind geometrische Objekte, die die Basis der Visualisierung bilden und Eigenschaften des Vektorfeldes wiedergeben.

Eine merkmalsbasierte Visualisierung untersucht das Strömungsfeld auf bestimmte Charakteristika. Logischerweise setzt dies einen Verarbeitungsschritt voraus, in dem diese Eigenschaften extrahiert werden. Zur Darstellung bedient man sich üblicherweise direkter beziehungsweise geometrischer Visualisierungsverfahren.

Für einen wissenschaftlichen Zugang zur Strömungsvisualisierung ist es wichtig, das zugrunde liegende Modell zu untersuchen. Die Basis bilden hier Vektordaten, welche üblicherweise als Richtungen interpretiert werden. Diese können beispielsweise das Ergebnis einer Ableitung von Messwerten sein. Gegebenenfalls kann an diesen Richtungsvektor über die entsprechende Länge eine weitere Eigenschaft gebunden werden. Im Folgenden geht man beim Vektorfeld von Geschwindigkeitsdaten aus, die sich beispielsweise aus der Ableitung von Positionen $\vec{x}$ nach der Zeit t ergeben. Geht man von masselosen Partikeln in einem derartigen Vektorfeld aus, so kann die Herleitung des Vektorfelds anhand folgender Gleichung beschrieben werden:

$$d\vec{x} = \vec{v}dt \Leftrightarrow \vec{v} = \frac{d\vec{x}}{dt} \tag{2.1}$$

$$\vec{x}(t, x_0) = \vec{x_0} + \int_{\tau=0}^{t} \vec{v}(\tau)d\tau \tag{2.2}$$

Um die Differentialgleichung 2.1 auszuwerten, wird sie in Form eines Integrals ausgedrückt. Man erhält damit die Formel, welche die Grundlage jeglicher Strömungsfelder bildet. Ausgehend von einer Anfangsposition $\vec{x_0}$ eines Partikels können Folgepositionen in Abhängigkeit von t bestimmt werden.

2.2 Partikelsysteme

Die Modellierung spezieller Effekte wie Wolken, Feuer, Rauch und weiterer Naturerscheinungen gehört zu den aktuellen Problemstellungen in der Computergraphik. Das Ausmaß und die Komplexität derartiger Objekte begrenzen oft die Darstellbarkeit anhand eines Polygonmodells. Anstelle des Polygonmodells treten dann üblicherweise Partikelsysteme, die aus einer Menge geometrisch voneinander unabhängiger Einzelteile bestehen. Im Laufe der Berechnung werden Partikel in das System eingefügt, fortbewegt und gelöscht. Der entscheidende Vorteil gegenüber einer oberflächenbasierten

Repräsentation resultiert aus der Abstraktion einzelner Partikel. Die Position eines Partikels ergibt sich aus einem Integrationsverfahren, und kann unabhängig von anderen Partikeln ermittelt werden. Die Bewegung einzelner Partikel wird durch ein Geschwindigkeitsfeld bestimmt. Dieses kann entweder analytisch oder als Ergebnis eines Messverfahrens gegeben sein.

Die Idee der Verwendung von Partikelsystemen ist alles andere als neu. Schon 1983 wird durch [Ree83] ein adäquates Konzept vorgestellt. Mittlerweile kommt kaum ein Computerspiel ohne den Einsatz derartiger Systeme aus. Wurden diese Partikelsysteme zunächst auf der CPU realisiert, so geht man heute zunehmend dazu über, den Ablauf vollständig auf der GPU zu realisieren, um so den zeitintensiven Datentransfer auf die Graphikkarte zu umgehen.

Die Konfiguration eines Partikelsystems erstreckt sich über unterschiedliche Phasen der Partikelberechnung. In den einzelnen Phasen ergeben sich unterschiedliche Steuerungsmöglichkeiten des Berechnungsprozesses. In diesem Zusammenhang kann zwischen der Initialisierung, der Positionsänderung und der Darstellung von Partikeln unterschieden werden. Die Initialisierung wird durch festgelegte Emitterkonfigurationen bestimmt. Die Positionierung dieser Emitter innerhalb eines Definitionsbereichs kann dabei entscheidenden Einfluss auf das Ergebnis haben. Die Änderung der Position aller existierender Partikel unterliegt dann einem konkreten Integrationsverfahren, welches die Folgepositionen der Partikel innerhalb des gegebenen (zeitabhängigen) Geschwindigkeitsfeldes bestimmt. Die Auswahl eines Integrationsverfahrens der Berechnung kann sich entscheidend auf die Effizienz und Präzision des Gesamtablaufs auswirken. Zuletzt können alle vorhandenen Informationen auf geeignete Weise gerendert werden. Neben der Position können auch weitere Partikeleigenschaften über visuelle Attribute veranschaulicht werden.

In Kapitel 2.6 wird die konkrete Implementierung eines Partikelsystems vorgestellt, das zur Visualisierung von Strömungsfeldern eingesetzt wird. Hier ergeben sich unterschiedliche Strömungslinie aus der integrationsbasierten Bestimmung einzelner Partikelbahnen aufgrund bestimmter Startpositionen. Da Kollisionen oder andere Interaktionen zwischen einzelnen Partikeln dabei nicht relevant sind, spricht man von einem lokalen Partikelsystem.

2.2.1 Integrationsverfahren

Ein Vektorfeld $\vec{F}$ im dreidimensionalen Raum wird durch eine Funktion definiert, die jeder Position des Definitionsraumes einen eindeutigen Vektor zuweist. In einem Strömungsfeld repräsentieren die einzelnen Vektoren $\vec{v}$ lokale Geschwindigkeiten. Diese können sich beispielsweise durch die Ableitung von gemessenen Positionen $x = x(t)$ nach der Zeit t ergeben.

$$\dot{\vec{x}}(t) = \vec{v}(t) = \frac{\mathrm{d}\vec{x}(t)}{\mathrm{d}t} \tag{2.3}$$

Im Rahmen der Visualisierung solcher Strömungsdaten werden einzelne Partikelbahnen $\vec{p}(s)$ extrahiert und dargestellt. Da diese Partikelbahnen in der Regel nicht in Form einer analytischen Differentialgleichung gegeben sind, müssen sie über eine numerische Integration von Partikelpositionen ermittelt werden. Üblicherweise wird dieser Integration ein bestimmtes Zeitintervall s zugrunde gelegt.

Man beachte in diesem Zusammenhang die Ähnlichkeit der folgenden Gleichung mit der Gleichung 2.2.

$$\vec{p}(s) = \vec{p_0} + \int_{\tau=0}^{s} \vec{v}(\vec{p}(\tau), \tau + t_0) d\tau \tag{2.4}$$

$\vec{p_0}$ ist die Startposition der Partikelbahn, wobei t_0 dem entsprechenden Emissionszeitpunkt entspricht. Zur Integration werden eine Reihe unterschiedlicher Verfahren vorgeschlagen. Diese approximieren das Ergebnis meist aufgrund einer Diskretisierung durch eine Taylorentwicklung (vgl. 2.5) und unterscheiden sich im Wesentlichen durch ihre Effizienz und Stabilität.

$$\vec{x}(t_0 + \tau) = \vec{x}(t_0) + \tau \frac{\partial \vec{x}}{\partial t}(t_0) + ... \tag{2.5}$$

Die Auswahl eines Integrationsverfahrens setzt dementsprechend eine sorgfältige Analyse und Anpassung der Einflussfaktoren auf den lokalen Diskretisierungsfehler, der sich innerhalb eines Schrittes durch den Abbruch der Taylorreihe ergibt, voraus. Dieser ist ausschlaggebend für die Stabilität und Zuverlässigkeit der entsprechenden Methode. In diesem Zusammenhang muss der Fehler, der sich aus der numerischen Integration ergibt, in Relation zum lokalen Interpolationsfehler des diskreten Vektorfeldes gesetzt werden. [TGE97]

2.2.2 Zeitschrittadaptivität

Für den zuvor erwähnten lokalen Diskretisierungsfehler erweist sich die Auswahl der Integrationsschrittweite h als maßgeblich entscheidend. Dieser muss aufgrund von lokalen Vektorfeldeigenschaften variieren, um bestimmten Genauigkeitsanforderungen zu entsprechen. Die Entwicklung von Algorithmen für eine adaptive Kontrolle des Diskretisierungsfehlers von Integrationsverfahren kann inzwischen auf eine lange Tradition zurückblicken, und fand im Laufe der Zeit eine breite Verwendung. Repräsentativ für die Aktivität der Entwicklung können eine Reihe von Beiträgen genannt werden. [Feh70] schlägt beispielsweise ein heuristisches Verfahren zur Ermittlung der optimalen Schrittweite vor. [CP92] erweitert dieses Konzept durch Verfahren dynamischer Systemtheorien, um die Ermittlung zu beschleunigen. Ein neuerer Ansatz realisiert eine entsprechende Abschätzung beispielsweise auf Basis linearer Kontrolltheorien [Söd02].

Im Falle des Runge-Kutta-Verfahrens 4. Ordnung (vgl. A.1), kann eine Regel zur Bestimmung der optimalen Schrittweite angegeben werden. Im Prinzip vergleicht man hier Näherungen, die im Laufe des Verfahrens ermittelt werden:

$$Q = |\frac{k_3 - k_2}{k_2 - k_1}|$$

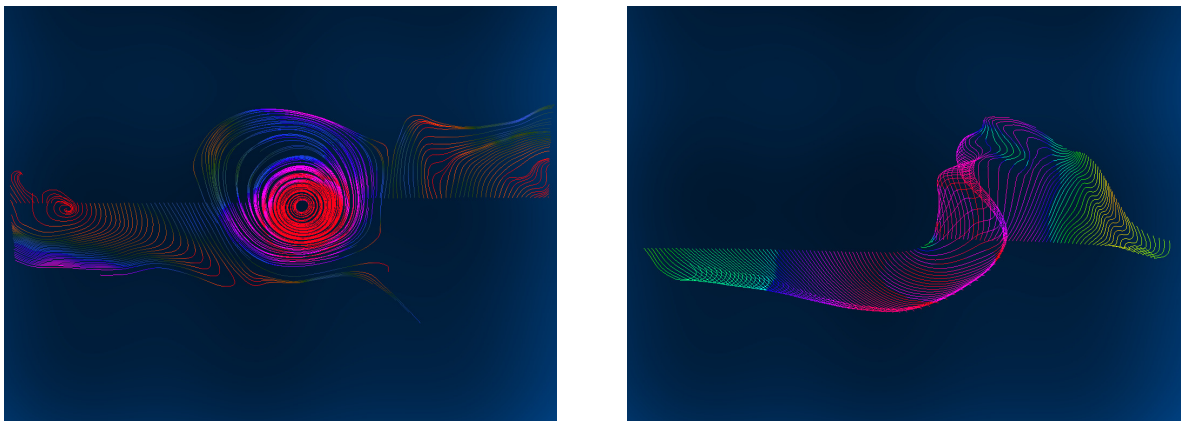
Befindet sich der Betrag dieses Quotienten innerhalb vorgegebener Schranken von $Q \in (0.025, 0.1)$ ist der numerische Fehler als minimal anzunehmen [Stö95]. Um den Rechenaufwand während der Ausführung so gering wie möglich zu halten, kann diese Schrittweite in einem iterativen Vorverarbeitungsschritt ermittelt werden.

2.3 Strömungsprimitive

Bei der Untersuchung von zeitabhängigen Strömungsfeldern spielt die Wahl des geometrischen Strömungsprimitivs eine wesentliche Rolle. Aufgrund ihrer Beschaffenheit können unterschiedliche Zusammenhänge dargestellt werden.

2.3.1 Strömungslinien

Über Strömungslinien können strukturelle Eigenschaften eines Strömungsfeldes analysiert werden. Bei der Generierung werden einzelne Partikelbahnen verfolgt. Die dabei ermittelten Partikelpositionen werden dann auf geeignete Weise miteinander verbunden, um bestimmte Charakteristika hervorzuheben. Im Allgemeinen wird zwischen so genannten Stream-, Streak-, Path- und Timelines unterschieden.



(a) (b)

Bild 2.2: Streamlines (a) und Streaklines (b)

Streamlines setzen sich aus Positionen zusammen, die sich aus der Partikelintegration zu einem festen Zeitpunkt ergeben. Aufgrund der Zeitkonstante kann diese nicht experimentell erzeugt werden. Es handelt sich vielmehr um eine Momentaufnahme des Vektorfeldes. Ein Windbändsel an Wanten oder Achterlieken eines Segelbootes zur Anzeige der Windrichtung verdeutlicht die Eigenschaften einer Streamline. Da es sich jeweils an der aktuellen Windrichtung ausrichtet verläuft es in jedem Punkt parallel zum zugehörigen Geschwindigkeitsfeld, was in diesem Fall durch Windgeschwindigkeiten gegeben ist. Im Rahmen der Visualisierung werden von einer Startposition ausgehend von Geschwindigkeitsvektoren sukzessiv Folgepositionen ermittelt. Aus dem Linienverlauf lassen sich zudem bestimmte Strömungseigenschaften ableiten. Beispielsweise verlaufen Druckänderungen üblicherweise senkrecht zur Streamline.

Streaklines (Streichlinien) ermöglichen im Gegensatz zu Streamlines eine Analyse zeitabhängiger Eigenschaften. Experimentell ergibt sich eine Streakline durch kontinuierliches Einführen von Partikeln in eine Strömung an einem festen Startpunkt. Da sie über einen gewissen Zeitraum betrachtet werden, beschreibt die Darstellung die Dynamik des zugrunde liegenden Vektorfeldes. Bei der Konstruktion der Streaklines im Rahmen der Visualisierung verbindet man Positionen von Partikeln, die in aufeinander folgenden, diskreten Zeitpunkten am selben Ort gestartet sind. Die Positionen aller vorhandenen Partikel eines Zeitschrittes bilden die Grundlage der Integration der entsprechenden Positionen im nächsten Zeitschritt. Streaklines benötigen im Bezug auf die Integration einen erheblichen Rechenaufwand. Deren interaktive Darstellung ist eines der Hauptziele dieser Arbeit.
Die Pathline (Pfadlinie) verfolgt die Spur eines Partikels innerhalb der Strömung über einen gewissen Zeitraum hinweg. In jedem diskreten Zeitschritt wird die zuletzt ermittelte Position jeweils um einen Schritt fortbewegt. Eine Linie setzt sich jeweils aus aufeinander folgenden Positionen eines Startpunktes zusammen.

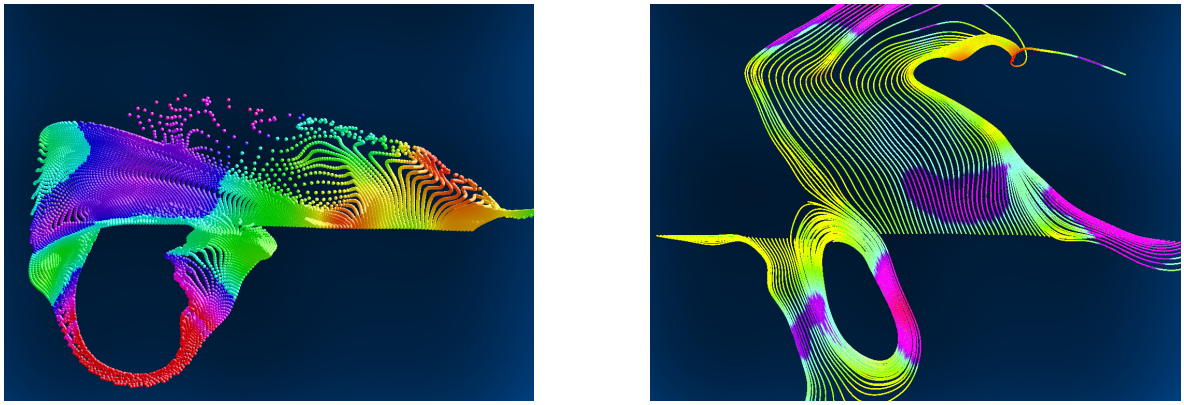
Zuletzt fügt eine so genannte Timeline die analog zu Pathlines ermittelten Partikelpositionen mehrerer Startpositionen in der Form zusammen, dass jeweils zum gleichen Zeitpunkt eingeführte Partikelpositionen miteinander verbunden werden. Sie veranschaulichen unterschiedliche Geschwindigkeitsentwicklungen innerhalb der Strömung.

Die erzeugten Primitive können als Träger weiterer Informationen eingesetzt werden. Beispielsweise können Geschwindigkeitsverhältnisse anhand einer entsprechenden Farbe der Linie dargestellt werden. Durch einfache Beleuchtungseffekte kann die räumliche Wahrnehmung der beschriebenen Strömungsprimitive deutlich gesteigert werden. Eine entsprechende Berechnung wird in Kapitel 2.4 dargestellt.

2.3.2 Stream Balls, Stream Ribbons, Stream Tubes

Stream Balls, Stream Ribbons (Strömungsbänder) und Stream Tubes (Strömungsröhren) steigern die Qualität der visuellen Darstellung durch die Integration der räumlichen Perzeption anhand von echten 3D-Geometrien.

Stream Balls stellen die ermittelten Partikelpositionen anhand von (beleuchteten) Kugeln dar. Diese können beispielsweise zusätzliche Strömungsinformationen über unterschiedliche Farben oder Radien veranschaulichen (vgl. Bild 2.3(a)). Werden Glyphen anstelle dieser Kugeln verwendet, können neben skalaren Größen gegebenenfalls auch vektorielle Eigenschaften dargestellt werden. Analog zu Stream Balls erweitern Stream Ribbons die Vorstellung der Streamlines durch Strömungsbänder, die aufgrund der lokalen Rotation gedreht werden können. Zusätzlich stehen mit der Breite und der Oberfläche weitere Informationsträger zur Verfügung. Ein ähnlicher Ansatz wird auch bei Streamtubes verfolgt (vgl. Bild 2.3(b)). Anstelle der Strömungsbänder treten Strömungsröhren, welche entsprechende Informationen über den Radius und Oberflächeneigenschaften wie Farbe oder Texturierung visualisieren können. Eine entsprechende Realisierung soll in späteren Abschnitten noch dargestellt werden.



(a) (b)

Bild 2.3: Stream Balls/Stream Tubes

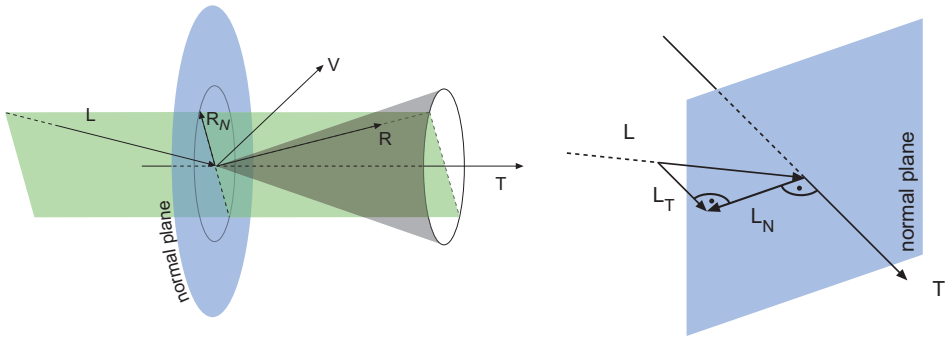
2.3.3 Strömungsflächen , -volumina

Flächen- und volumenbasierte Verfahren erweitern die bisher beschriebenen Primitive um jeweils eine weitere räumliche Dimension. Indem Streamlines zusätzlich entlang der schon erwähnten Timelines verbunden werden, erhält man eine Approximation der Strömungsflächen. Im Idealfall verläuft diese in jedem Punkt parallel zur Strömungsrichtung. Daneben sind Timesurfaces die äquivalente Erweiterung der zuvor erwähnten Timelines auf Flächen. Unter Strömungsvolumina werden nun zuletzt Objekte als Teilmenge des gesamten Vektorfeldes verstanden. Als Ausgangspunkt für die Integration wird eine Fläche in 2D angenommen. Für detaillierte Beschreibung sei auf [PLV+02] verwiesen.

2.4 Einfache Beleuchtunsberechnung für Strömungslinien

Durch eine relativ einfache Beleuchtungsberechnung können Linienprimitive um einen räumlichen Tiefeneindruck ergänzt werden. Dazu wird in [ZSH96] ein entsprechendes Verfahren dargestellt, welches die Beleuchtungsberechnung auf Basis des Phong-Modells realisiert (siehe Anhang A.2). Es generalisiert das oberflächenbasierte Phong-Beleuchtungsmodell auf Linienprimitive.

Wie aus dem Modell ersichtlich ist, wird die resultierende Intensität durch die Summe dreier unabhängiger Terme approximiert. Während der erste Term durch globale Konstanten bestimmt wird, sind für die übrigen Terme lokale Beleuchtungseigenschaften ausschlaggebend. In der Regel lässt sich aus Oberflächeneigenschaften ein Normalenvektor ableiten. Daraus kann anhand der Lichtrichtung L ein so genannter Reflexionsvektor R ermittelt werden. Mit der Blickrichtung V sind alle Voraussetzungen zur Berechnung dieses Modell geschaffen. Für oberflächenbasierte Strukturen können die benötigten Normalen- und Reflexionsvektoren eindeutig abgeleitet werden. Für linienbasierte Strukturen in 3D kommen hingegen unendlich viele Möglichkeiten der Auswahl dieser Vektoren in Betracht (siehe Bild 2.4(a)).



(a) (b)

Bild 2.4: Herleitung der Beleuchtungsberechnung

Aus allen möglichen Normalenvektoren wählt man aus diesem Grund jene aus, welche sich koplanar zu L und T verhält. Ein entsprechendes Kriterium wird auch zur Auswahl des Reflexionsvektors eingesetzt. Durch eine Zerlegung des Lichtvektors entsprechend Bild 2.4(b) lassen sich die beiden Terme auch ohne den Umweg einer Normalen- beziehungsweise Reflexionsvektorberechnung ermitteln.

$$L \cdot N = |L_N| = \sqrt{1 - |L_N|^2} = \sqrt{1 - (L \cdot T)^2} \tag{2.6}$$

$$\begin{aligned} V \cdot R &= V \cdot (L_T - L_N) && (2.7)\\ &= V \cdot ((L \cdot T) \cdot T - (L \cdot N) \cdot N) && (2.8)\\ &= (L \cdot T)\,(V \cdot T) - (L \cdot N)\,(V \cdot N) && (2.9)\\ &= (L \cdot T)\,(V \cdot T) - \sqrt{1 - (L \cdot T)^2}\sqrt{1 - (V \cdot T)^2} && (2.10) \end{aligned}$$

Eine entsprechende Vereinfachung lässt sich direkt auf trigonometrische Gesetzmäßigkeiten zurückführen. Für eine entsprechende Berechnung genügt also die Interpolation der Tangente. Diese ergibt sich für eine Partikelposition jeweils aus den adjazenten Partikelpositionen und kann jeweils dem entsprechendem Vertex als zusätzliche Variable hinzugefügt werden.

2.5 GPUs und GPGPU

Die Entstehung von GPU-basierten Anwendungen, die über die Absicht der fotorealistischen Szenendarstellung hinausgehen, ist im Wesentlichen auf die rasante Verbesserung und Erweiterung der Programmierbarkeit und Effizienz aktueller Graphikhardware zurückzuführen. Die Eigenschaften der GPUs sowie die damit einhergehenden Programmstrukturen, welche grundsätzlich auf der Rende-

ring Pipeline basieren, sollen im Folgenden anhand eines kurzen Überblicks dargestellt werden. Diese Ausführung bildet die Grundlage für die darauf folgende Nutzung im Bereich der Strömungsvisualisierung. Für detaillierte Hintergrundinformationen im Bezug auf die Rendering-Pipeline sei auf [FvDFH96] verwiesen.

2.5.1 Graphik-Pipeline (VS, GS, FS)

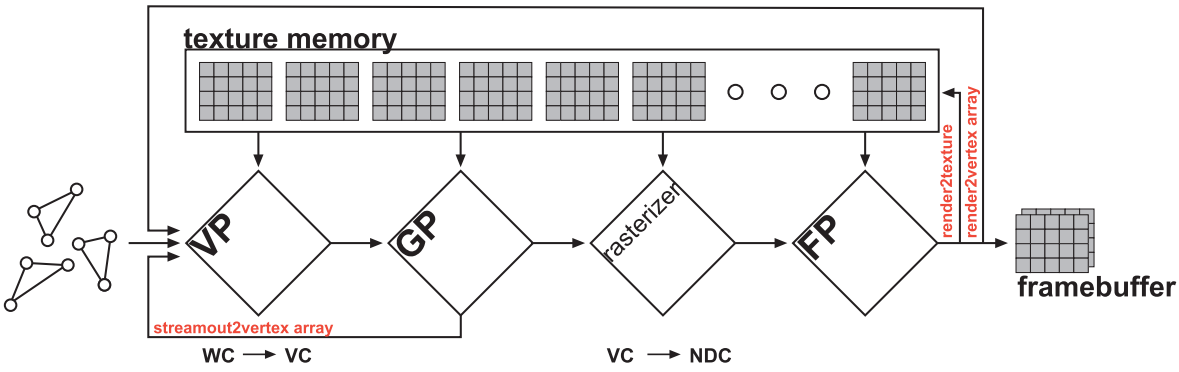


Bild 2.5: Konzeptionelle Darstellung der Graphik-Pipeline. Die graphische Verarbeitung unterliegt in den einzelnen Phasen der Berechnung unterschiedlichen Voraussetzungen an Eingabeprimitiv und zusätzlicher Berechnungsinformation, die in Form von Texturen bereitgestellt werden können.

Die Rendering-Pipeline bildet den Kern der 3D Computergraphik. Hinter diesem Begriff verbirgt sich eine festgelegte Abfolge bestimmter Phasen zur Produktion eines gerasterten Bildes auf Basis von 3D-Repräsentationen einer Szene. Diese soll im Folgenden grob dargestellt wird. Die Eingabe bildet eine Liste bestimmter Geometrien, die durch Punkte in Objektkoordinaten angegeben werden. Verschiedene Modellierungstransformationen setzen die einzelnen Geometrien in Beziehung zueinander. Über die Viewingtransformation wird die gesamte Szene in Beziehung zur Kameraposition gesetzt. In weiteren Teilschritten werden die Eingabepunkte zu zuvor spezifizierten Primitiven zusammengesetzt. Anhand der Licht- und Kameraposition kann eine Berechnung von Beleuchtungseigenschaften auf Punkteebene durchgeführt werden. In der zweiten Phase wird die exakte Position einzelner Oberflächenpolygone auf dem Bildschirm ermittelt. Über die Projektions- sowie Viewport-Transfornmation werden dazu alle Werte entsprechend skaliert, sodass sie direkt auf Pixelkoordinaten des Framebuffers übertragen werden können. Zudem werden einzelne Punktparameter über das gesamte Polygon interpoliert, sodass in der folgenden Phase eine Beleuchtungsberechnung auf Fragmentebene durchgeführt werden kann. Das Ergebnis ist ein Fragment für jedes Pixel des Framebuffers, das von einem der Polygone verdeckt wird. In der folgenden Phase kann dann aufgrund der interpolierten Punktparameter für jedes Fragment eine entsprechende Farbe ermittelt werden. In der letzten Phase findet dann die Komposition der einzelnen Fragmente zu einem rasterisierten Bild statt. Das Ergebnis wird in den Framebuffer geschrieben, der dann auf den Bildschirm projiziert wird.

Der enorme Geschwindigkeitsvorteil gegenüber der CPU, der aus einer Auslagerung von Graphikprozessen in eine eigene Hardwareeinheit (GPU) resultiert, kann hauptsächlich auf die Parallelisierung einzelner Teilprozesse zurückgeführt werden. Durch die Verarbeitung von mehreren Streams (Datenströmen) ist es möglich, bestimmte Verarbeitungsschritte auf eine Menge gleichwertiger Eingabeelemente zu optimieren. Ordnet man diese Streams dann in so genannten Pipelines an, kann eine parallele Verarbeitung von Befehlssequenzen realisiert werden (vgl. Bild 2.6).

Die Integration der benutzerdefinierten Programmierung, Vertex-Shader (VS) und Fragment Shader (FS), für einzelne Phasen der Pipeline geben dieser Entwicklung eine weitere Dimension der Flexibilität. Es ermöglicht eine adaptive Einflussnahme auf den Prozess der Berechnung. Durch einen einheitlich gestalteten Speicherzugriff in den einzelnen Phasen der Pipeline, können in Texturen gespeicherten Informationen als Grundlage für und Ergebnis von Berechnungen dienen.

2.5.2 Geometry Shader

Der Geometry Shader (GS) stellt innerhalb der klassischen Rendering-Pipeline einen relativ neuen Shadertyp dar. Er wurde erst mit dem Shader Model 4.0 (2006) eingeführt [PBSN06]. Als eine weitere programmierbare Phase kann dieser nach dem VS aktiviert werden und akzeptiert als Eingabe einzelne Punkte oder auch vollständige Primitive wie sie durch die entsprechende Anwendung spezifiziert werden. Bei der Ausführung des Programms können Eingabeprimitive auf beliebige Weise modifiziert werden.

Als einziger Shadertyp ist der GS in der Lage neue Geometrie dynamisch hinzuzufügen oder zu löschen. Analog zum VS und FS kann sich auch die GS-Einheit dabei auf Texturen als Grundlage von Berechnungen stützen. Im Gegensatz zum VS hat er jedoch simultanen Zugriff auf alle Punkte eines Eingabeprimitivs. Die Spezifikation dieser Erweiterung stellt eine Reihe möglicher Eingabeprimitive zur Verfügung. Die Angabe des Eingabeprimitivs bestimmt die Anzahl der in einem Durchlauf verfügbaren Punkte. Die Ausführung des Shaders setzt sowohl die Angabe eines Ein- und Ausgabeprimitivs voraus. Generell ist die Anzahl der Punkte, die während eines Programmaufrufs ausgegeben werden hardwaremäßig beschränkt. Das bisher bekannte Prinzip der Parallelität wird somit durch Konditionen erweitert, die eine weitere Phase der Filterung bzw. der Modifikation der Ausgabe ermöglichen.

2.5.3 Stream-Output/Transform-Feedback

Bei der Verwendung des GS im Zusammenhang einer fortlaufenden Modifikation von Primitiven spielt die Nutzung des Transformation Feedbacks (TF) eine ausschlaggebende Rolle. Anhand des TF können zuvor ausgewählte Attribute einzelner Punkte während der Verarbeitung von Primitiven in ein oder mehrere Bufferobjekte geschrieben werden, ohne die gesamte Pipeline durchlaufen zu müssen. In Folgeschritten können diese Daten wiederum die Eingabe der Rendering-Pipeline bilden. Als hilfreiches Kontrollinstrument erweist sich in diesem Zusammenhang die Option einer asynchronen Abfrage, mittels derer spezielle Informationen über den Status einer Feedbackfolge ermittelt werden

können. So kann beispielsweise schon die Anzahl der verarbeiteten Primitive Aufschluss über die Struktur des modifizierten Bufferobjekts geben und die Grundlage der Auswahl von Folgeschritten in anschließenden Pipelinedurchläufen bilden. Ohne weitere Einflussnahme der CPU sind damit Eingabedaten sukzessiv modifizierbar. Ein derartiger Mechanismus stellt die Basis iterativer Algorithmen auf der GPU dar. Eine entsprechende Verwendung wird im späteren Verlauf dieses Beitrags noch dargestellt (siehe Kapitel 3.4).

2.5.4 Texturen, FBOs

Der Zugriff auf Texturen gilt auch über den Bereich der Visualisierung hinaus als eines der wichtigsten Verfahren des Echtzeitrenderings. Traditionell dienten zu diesem Zweck direkte Zugriffsoperationen auf den Framebuffer oder die so genannten Pixelbuffer.

Bei der Konstruktion von Frame Buffer Objects (FBOs) löst man sich in vielerlei Hinsicht von den Beschränkungen dieser existierenden Lösungen und eröffnet damit eine völlig neue Form des Off-Screen-Renderings. FBOs kapseln in diesem Zusammenhang viele Eigenschaften des bisher bekannten Framebuffers durch eine neuartige Speicherstruktur. Im Gegensatz zur festgelegten Struktur des Frambuffers bieten FBOs die Gelegenheit unabhängig voneinander Color-, Depth- und Stencilbuffer zu erzeugen und einzubinden. Die Einbindung mehrerer Colorbuffer erlaubt zum Beispiel das gleichzeitige Rendern in unterschiedliche Buffer. Die Flexibilität ergibt sich hauptsächlich aus der Möglichkeit zwischen verschiedenen Renderzielen zu wechseln.

2.5.5 GPGPU

Das Leistungsspektrum heutiger Graphikhardware ist das Ergebnis einer im hohen Maße spezialisierten Bauweise, die ursprünglich aus Charakteristika der graphischen Aufgabenstellungen resultiert. Durch eine parallele Anordnung eng gekoppelter Prozessoren, eignet sich diese Bauweise vorzugsweise für die Anwendung von ähnlichen Operationen auf eine große Menge von Daten (SIMD). Bild 2.6 verdeutlicht das generelle Prinzip der parallelen Verarbeitung. Innerhalb der Graphik-Pipeline bildet das Rechenaufkommen bei der Rasterisierung, beziehungsweise der Fragmentberechnung, die Motivation für derartige Entwicklungen. Das zugrunde liegende Modell der Programmierung und Verarbeitung genügt jedoch auch Anforderungen aus einer Reihe von unterschiedlichen Anwendungsbereichen, die nicht zwangsläufig der Computergraphik zuzuordnen sind [OLG+07]. Die rasche Entwicklung im Hinblick auf Bandbreite, Durchsatz und Flexibilität der Programmierung steigern in zunehmenden Maß das Interesse der Nutzung. Die Berechnung wird dazu in unabhängige Phasen unterteilt, die die Verarbeitung einer Menge von Daten ohne einen Zustandswechsel ermöglichen. Die Vermittlung von Zwischenergebnissen einzelner Phasen wird durch eine geeignete Synchronisation gewährleistet. Im Kontext der Strömungsvisualisierung wird eine entsprechende Konfiguration durch ein GPU-basiertes Partikelsystem repräsentiert. Da es die Ausgangslage dieser Arbeit darstellt, soll es im Folgenden näher erläutert werden.

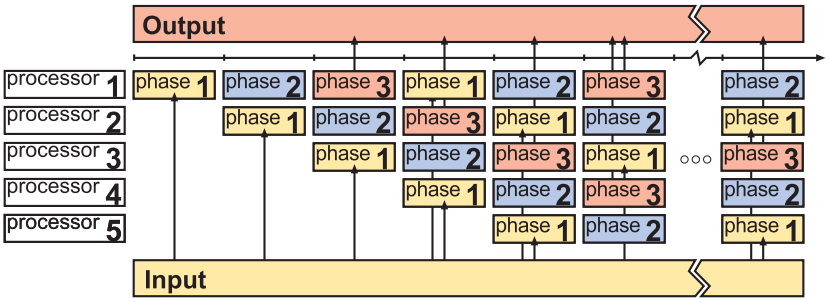
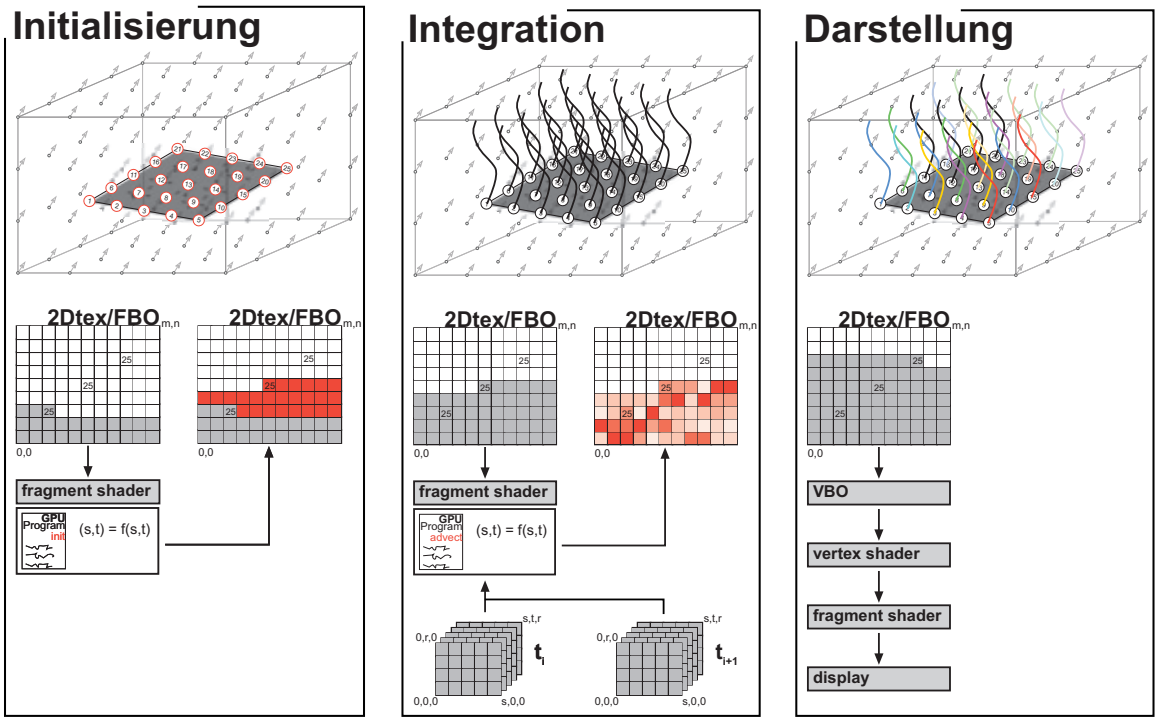


Bild 2.6: Parallelität/Pipelining. Hier wird das Prinzip des Pipelining dargestellt. Einzelne Phasen der Berechnung werden sequenziell abgearbeitet. Durch ein parallele Anordnung und eine Synchronisation der entsprechenden Prozessoren (Pipelining) wird jedoch faktisch die gleichzeitige Verarbeitung ganzer Befehlsfolgen erreicht.

2.6 GPU-basierte Partikelsysteme



(a) (b) (c)

Bild 2.7: GPU-basierte Partikelsysteme. Aus einzelnen Phasen der Verarbeitung lassen sich entsprechende Anforderungen an Rechen- und Speicherbedarf ableiten.

Partikelsysteme werden üblicherweise dazu genutzt, die Dynamik einer großen Partikelmenge zu kontrollieren. Die Bewegung einzelner Partikel unterliegt dabei einer festgelegten Transformations-

vorschrift, die übergreifend für alle Partikel des Systems definiert wird. Durch zusätzliche (zeitabhängige) Parameter, die sich üblicherweise auf diese Vorschrift beziehen, kann das Systemverhalten gesteuert werden. Aus dieser Tatsache lassen sich Analogien zur graphischen Verarbeitung ableiten. Anstelle der Partikel dienen hier üblicherweise Punkte beziehungsweise Fragmente als Grundlage von Berechnungen, die aufgrund vorgegebener Shaderprogramme durchgeführt werden. Das zugrunde liegende Konzept der graphischen Verarbeitung kann deshalb relativ intuitiv auf die Verarbeitung von Partikelsystemen angewandt werden. Hieraus ergibt sich die Affinität der Umsetzung von Partikelsystemen auf der Graphikhardware. In [KSW04] sowie [KLRS04] werden Partikelsysteme vorgestellt, welche vollständig auf der Graphikkarte realisiert wurden und dadurch im Vergleich zur konventionellen, CPU-basierten Lösung wesentlich beschleunigt werden konnten. Alle Partikel eines derartigen Systems durchlaufen üblicherweise die in Kapitel 2.2 erwähnten Phasen der Verarbeitung: Initialisierung, Positionsänderung und Darstellung. Dabei werden unterschiedliche Anforderungen an Rechen- und Speicherbedarf gestellt.

2.6.1 Graphikspeicher

Um den Datentransfer zwischen CPU und GPU zu minimieren, müssen möglichst alle relevanten Informationen auf der Graphikkarte vorgehalten werden. Dazu zählen im Kontext von Vektorfeldern neben den Vektorfelddaten selbst auch Partikelpositionen, sowie die eingesetzten Shaderprogramme. Partikelpositionen werden in diesem Zusammenhang anhand von 2D-Texturen gespeichert.

Üblicherweise werden Texturen als Datenstrukturen verstanden, deren Elemente (Fragmente) sich aus Farbinformationen zusammensetzen. Die Definition eines internen Datenformats legt dabei die Anzahl sowie die Datentiefe der zur Verfügung stehenden Farbkomponenten fest. Zur Speicherung von Partikelpositionen werden die Farbkomponenten als räumliche Dimensionen interpretiert. Die maximale Anzahl möglicher Partikel wird durch die Größe der Partikeltextur, beziehungsweise durch die maximale Größe entsprechender Texturen abgesteckt. Für den Zugriff müssen diese Texturen zunächst aktiviert und an entsprechende Ziele der Graphikhardware gebunden werden. Diese Konfiguration gibt eine strikte Trennung zwischen Lese- und Schreibzugriff vor. Da Partikelpositionen sukzessiv ermittelt werden, benötigt man zumindest eine Anordnung von zwei Texturen, die ihren Zustand als Eingabe- beziehungsweise Ausgabetextur im Laufe der Ausführung gegenläufig wechseln. Auf diese Weise wird ein iterativer Berechnungsprozess gewährleistet. Erweitert man diese Vorstellung auf eine ringförmige Anordnung weiterer Texturen, so können Informationen mehrerer Texturen in jeweils eine Berechnung einfließen. Die beschriebenen Texturen können anhand von FBOs realisiert werden. Im Gegensatz zu traditionellen Texturen bieten diese einen wesentlich einfacheren Zugriffsmechanismus für den Off-Screen-Rechenprozess der Partikelintegration (siehe Kapitel 2.5.4).

Zur Speicherung der Vektorfelddaten werden 3D-Texturen genutzt. Generell kann man sich diese als strukturierte Anordnung gleich großer 2D-Texturen vorstellen. Jede dieser 2D-Texturen speichert jeweils alle diskreten Positionen in Richtung zweier Koordinatenachsen bei einer festgelegten dritten Dimension. In einer zeitabhängigen Konfiguration wird für jeden diskreten Zeitschritt eine entspre-

chende Textur benötigt.

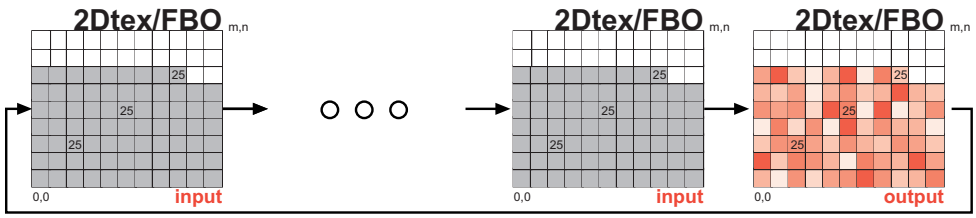


Bild 2.8: Texturkonfiguration. Hier wird eine mögliche Ringkonfiguration dargestellt. Ein Zustandswechsel erfolgt entlang dieser Anordnung.

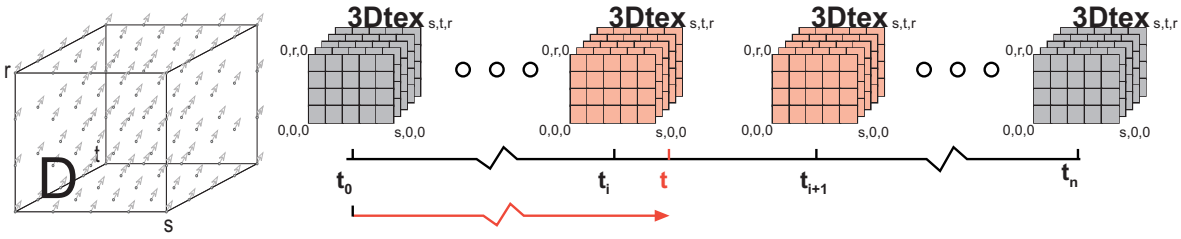
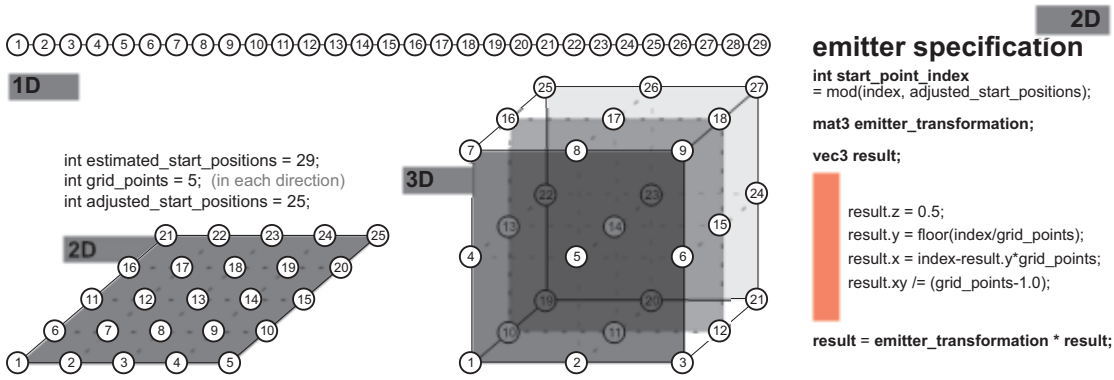


Bild 2.9: Für einen Definitionsbereich D des Vektorfeldes werden für jeden diskreten Zeitschritt die Vektorfelddaten in 3D-Texturen gespeichert.

2.6.2 Initialisierung

Die Initialisierung neuer Partikelpositionen wird über bestimmte Emitterkonfigurationen realisiert. Die einzelnen Startpositionen ergeben sich aus Transformationen bezüglich eines lokalen Koordinatensystems nach einem festgelegten Algorithmus. Das Resultat kann beliebig innerhalb des Definitionsbereichs des Vektorfeldes positioniert werden. Über die Dichte und räumliche Verteilung entsprechender Emitter können bestimmte Bereiche des Vektorfeldes für detaillierte Untersuchungen extrahiert werden.

Prinzipiell kann die Anzahl der Startpositionen beliebig gewählt werden. Jedoch muss dieser Wert gegebenenfalls an den Konstruktionsalgorithmus der Emitterkonfiguration angepasst werden. Nimmt man beispielsweise die in Bild 2.10(a) definierten Konfigurationen an, so wird die Anzahl der Startpositionen im 2D-Fall auf die nächst kleinere Zweierpotenz angepasst. Analog wird im 3D-Fall auf die nächst kleinere Dreierpotenz angeglichen. Neben den dargestellten Konfigurationen sind auch komplexere sowie zufällige Anordnungen denkbar. Die Berechnung der Startpositionen, sowie deren Einträge in die Partikeltextur, werden über Fragmentprogramme realisiert. Dabei muss die Konsistenz der Partikeltextur zu jedem Zeitpunkt der Berechnung gewährleistet sein. Partikelpositionen werden aus diesem Grund sequenziell in die entsprechende Textur geschrieben. Über Zeiger, die bei allen Schreiboperationen mitgeführt werden, werden Startpunkte folgender Operationen kontrolliert.



(a) (b)

Bild 2.10: Emitterkonfigurationen. 2.10(a) stellt mögliche Emitterkonfiguration dar. Eine mögliche Spezifikation für ein 2D Gitter wird in 2.10(b) veranschaulicht.

Gelangt man durch fortlaufende Emissionsschritte an die Grenze der verfügbaren Texturfragmente, ist die maximale Lebensdauer der anfangs emittierten Partikel erreicht. In den folgenden Schritten können diese überschrieben werden.

2.6.3 Integration

Die Positionsänderung einzelner Partikel ergibt sich aus einem Integrationsverfahren. Dieses wird analog zur Initialisierung über entsprechende Fragment Shader verwirklicht. Zusätzlich zur Partikeltextur dienen Vektorfelddaten als Grundlage der Integration. An beliebigen Positionen des Definitionsbereichs können relevante Vektoren über eine trilineare Interpolation aus den nächstgelegenen diskreten Abtastwerten ermittelt werden.

Eine zeitabhängige Integration erfordert zusätzlich eine Interpolation zwischen den Ergebnissen $\vec{v_i}$ und $v_{i+1}\vec{}$ der trilinearen Interpolation an den diskreten Zeitschritten (vgl. Bild 2.9 und Bild 2.11). Für einen beliebigen Zeitpunkt t mit $t_i \leq t \leq t_{i+1}$ ergibt sich das gewünschte Ergebnis aus $(1-\alpha)\cdot\vec{s}+\alpha\cdot\vec{t}$ mit $\alpha = \frac{t-t_i}{t_{i+1}-t_i}$.

2.6.4 Rendering

Nach der Advektion werden Partikelpositionen schließlich in ein Vertex Bufffer Object (VBO) kopiert. Diese Speicherstruktur der Graphikhardware enthält die Eingabepunkte für den folgenden Renderprozess. Dazu werden die gespeicherten Vertex-Informationen durch die Graphik-Pipeline anhand von Vertex- und Fragment Shader zur Darstellung gebracht.

Eine konkrete Nutzung des beschriebenen Partikelsystems im Kontext der Strömungsvisualisierung wird in [KKKW05] vorgestellt. Ein statischer 3D-Strömungsdatensatz auf einem uniformen Gitter stellt dabei die Integrationsgrundlage dar. Neben einem Ansatz zur Schrittweitenkontrolle für die

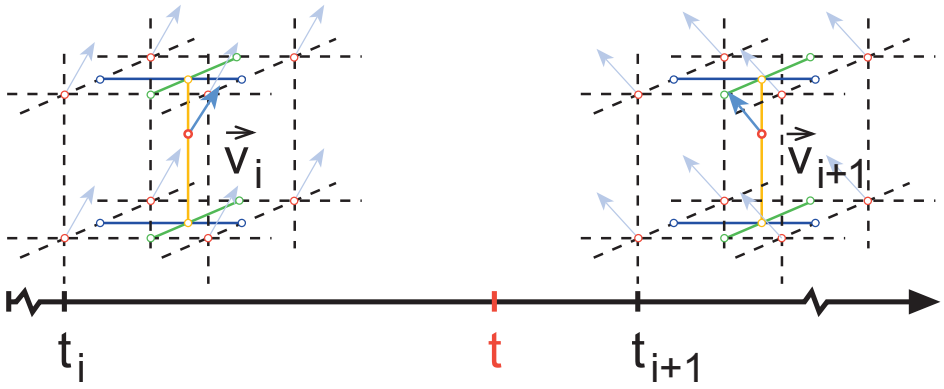


Bild 2.11: Quadrilineare Interpolation. Die Interpolation der zur Integration verwendeten Geschwindigkeitsvektoren in einem zeitabhängigen Vektorfeld dar.

Integration, werden eine Reihe unterschiedlicher Darstellungsprimitve zur Auswahl gestellt. Neben direkten Visualisierungsverfahren über beleuchtete Stream Balls, die praktisch über so genannte Point Sprites realisiert werden können, werden auch geometrische Verfahren wie Streamlines und Stream Ribbons berücksichtigt. [CKL+07] erweitert diese Vorstellung um dynamische Strömungsdaten. Die hier eingesetzten Verfahren bilden den Ansatzpunkt meiner Arbeit.

2.7 Transfer-Funktionen

Um spezifische Informationen eines Datensatzes innerhalb der Repräsentation hervorzuheben, beziehungsweise unwichtige Details auszublenden, erweisen sich Transferfunktionen als effizientes und anpassungsfähiges Hilfsmittel. Sie haben sich vor allem im Bereich des Volume-Rendering als Standardverfahren der Visualisierung etabliert.

Im Prinzip setzt eine Transferfunktion lediglich einen skalaren Definitionsbereich $s \in [0, ..., 2^x - 1]$ in Relation zu einem Farbwert.

$$T_{F_\alpha}(s) = \alpha \text{ mit } 0 \leq \alpha \leq 1 \tag{2.11}$$

$$T_{F_{RBG}}(s) = \begin{pmatrix} R \\ G \\ B \end{pmatrix} \text{ mit } 0 \leq R, G, B \leq 255 \tag{2.12}$$

Über eine geschickte Definition der Formel 2.11 können zusätzlich Bereiche bestimmter Eigenschaften durch die Opazität unterschiedlich gewichtet werden. Um die Ausführung zu verdichten, werden in der folgenden Darstellung die Gleichungen 2.11 und 2.12 zu $T_{F_{RBGA}} : \mathbb{R}^N \rightarrow \mathbb{R}^M$ mit N = 1 und M = 4 zusammengefasst. Das Ergebnis einer Transferfunktion ist dementsprechend ein RGBA-Wert, der

direkt in die Darstellung einfließen kann. Im Kontext von Strömungsfeldern dient üblicherweise der Geschwindigkeitsbetrag als Definitionsbereich einer solchen so genannten eindimensionalen Transferfunktion. Darüber hinaus können eine Reihe weiterer Strömungseigenschaften ermittelt werden, deren Untersuchung einen tieferen Einblick verspricht.

Durch mehrdimensionale Transferfunktionen MDTF erweitert man die beschriebene Vorstellung, indem mehrere Transferfunktionen unterschiedlicher Merkmale in die Konstruktion einbezogen werden. Bei Strömungsfeldern kommen beispielsweise der Geschwindigkeitsbetrag, der Gradient als Ableitung der Geschwindigkeiten, die Divergenz, sowie Rotationseigenschaften in Betracht. Zur Konstruktion einer entsprechenden Funktion MDTF muss zunächst für jede der N ermittelten Skalarwerte eine eindimensionale Transferfunktion der folgenden Form definiert werden: $T_i : \mathbb{R} \rightarrow \mathbb{R}^4, i = 1, ..., N$. Anschließend werden diese zur n-dimensionalen Transferfunktion der Form $T : \mathbb{R}^N \rightarrow \mathbb{R}^4$ zusammengesetzt. Dazu werden die Farbwerte, die sich aus der Auswertung der einzelnen Transferfunktionen ergeben, mit dem entsprechenden Opazitätswert gewichtet und aufsummiert. Den resultierenden Farbwert erhält man durch eine Division durch die Anzahl der beteiligten Transferfunktionen. Analog werden für die Berechnung des resultierenden Opazitätswertes die einzelnen Alpha-Werte gewichtet aufsummiert.

$$T_{RGB}(x) = \frac{1}{N} \sum_{i=1}^{N} T_{i,RGB}(x_i) \cdot T_{i,\alpha}(x_i) \tag{2.13}$$

$$T_{\alpha}(x) = \frac{1}{N} \sum_{i=1}^{N} T_{i,\alpha}(x_i) \tag{2.14}$$

$$\text{mit } x = (x_1, ..., x_N) \in \mathbb{R}^N \tag{2.15}$$

Zur Ermittlung der einzelnen Strömungseigenschaften wird in [PBL+04] ein adäquater Ansatz vorgeschlagen, welcher auch im Rahmen dieser Arbeit berücksichtigt wird. Generell bieten Strömungslinien gegenüber direkten Visualisierungsverfahren den Vorteil der Darstellung nicht nur lokaler sondern auch globaler wie zeitabhängiger Eigenschaften. Dabei stellt sich jedoch vor allem bei dicht besetzten Ausgangsdaten die Verdeckung als größte Herausforderung dar. Transferfunktionen begegnen diesem Problem, ohne zuvor die Datendichte insgesamt zu reduzieren. Grundsätzlich werden dazu zunächst bestimmte Eigenschaften des zugrunde liegenden Datensatzes ermittelt, welche als Kriterien der nachfolgenden Darstellung dienen.

Eine Gewichtung einzelner Eigenschaften wird durch einen Opazitätswert realisiert, welcher in eine direkte Beziehung zur Stärke der Ausprägung einer ermittelten Eigenschaft gesetzt werden kann. Im Kontext von 3D Skalarfeldern respektive der Volumenvisualisierung, haben sich betreffende Verfahren längst als Standard durchgesetzt. Jedoch existieren auch Ansätze, eine entsprechende Vorgehensweise auch bei der Visualisierung von 3D Vektorfeldern zu nutzen [PBL+04].

Zur Ermittlung von Vektorfeldeigenschaften bedient man sich im Wesentlichen der Vektoranalysis. Prinzipiell kann dies während der Ermittlung der Partikelpositionen oder in Vorverarbeitungs-

schritten erledigt werden. Für ein einheitliches Verständnis zunächst die Definition des Vektorfeldes:

$$F(x,y,z) = F(r) = \begin{pmatrix} F_x(r) \\ F_y(r) \\ F_z(r) \end{pmatrix} \tag{2.16}$$

Die Vektorkomponenten der Funktion F geben also die Komponenten in Richtung der kartesischen Koordinatenachsen x, y, z an. Aufgrund dieser Annahme kann innerhalb eines Strömungsfeldes ein Geschwindigkeitsbetrag direkt über die Länge der entsprechenden Geschwindigkeitsvektoren interpoliert werden. Dieser Wert kann nun unmittelbar als Eingabe einer geeigneten Transferfunktion dienen. Auf diese Weise lassen sich innerhalb des Strömungsfeldes Bereiche unterschiedlicher Geschwindigkeit extrahieren. Um die Umgebung einer diskreten Position in die Betrachtung einzubeziehen, ist zusätzlich der so genannte Vektorgradient zu berechnen. Dies läuft im Prinzip auf eine Richtungsableitung des Vektorfeldes hinaus.

Dazu wird der Tensor $\nabla F(x,y,z) = \left(\frac{\delta F(x,y,z)}{\delta x}, \frac{\delta F(x,y,z)}{\delta y}, \frac{\delta F(x,y,z)}{\delta z}\right)$ definiert. Die partiellen Ableitungen dieser Matrix werden an diskreten Positionen über zentrale Differenzen approximiert. Die Bestimmung des Gradienten würde nun eine umfassende Eigenwertanalyse dieser Matrix voraussetzen. Innerhalb einer interaktiven Anwendung mit zeitabhängigen Vektorfeldern würde die Eigenwertanalyse den Rahmen der Realisierbarkeit sprengen. Dies ist im Wesentlichen auf aufwändige Matrixdekompositionen zurückzuführen. Um trotzdem einen Begriff für Strömungsänderungen in der Nähe diskreter Punkte zu erhalten, genügt die Bestimmung der Länge dieses Gradienten. Durch diese Approximation abstrahiert man von der Richtung des Gradienten und beschränkt sich auf das Ausmaß einer lokalen Strömungsänderung. Der Betrag ergibt sich ohne komplexe Matrixdekompositionen aus der Determinante des Vektorfeldgradienten.

$$|\nabla F(x,y,z)| = \begin{vmatrix} \frac{\delta F_1(x,y,z)}{\delta x} & \frac{\delta F_1(x,y,z)}{\delta y} & \frac{\delta F_1(x,y,z)}{\delta z} \\ \frac{\delta F_2(x,y,z)}{\delta x} & \frac{\delta F_2(x,y,z)}{\delta y} & \frac{\delta F_2(x,y,z)}{\delta z} \\ \frac{\delta F_3(x,y,z)}{\delta x} & \frac{\delta F_3(x,y,z)}{\delta y} & \frac{\delta F_3(x,y,z)}{\delta z} \end{vmatrix} \tag{2.17}$$

Als weitere Strömungseigenschaft wird aus $\nabla F(x,y,z)$ die Divergenz als Summe der Hauptdiagonalen ermittelt: $divF(x,y,z) = \frac{\delta F_1(x,y,z)}{\delta x} + \frac{\delta F_2(x,y,z)}{\delta y} + \frac{\delta F_3(x,y,z)}{\delta z}$
Für eine diskrete Stelle innerhalb des Strömungsfeldes stellt dieser Wert eine bestimmte Bewegungstendenz in dessen Nähe dar. Über eine Transferfunktion können so beispielsweise Quellen (Divergenz größer Null) und Senken (Divergenz kleiner Null) unterschieden werden. Rotationen können ebenso aus der oben dargestellten Matrix aufgrund folgender Kombination der einzelnen Terme abgeleitet werden.

$$rotF\left(x,y,z\right)=\begin{pmatrix}\frac{\delta F_3(x,y,z)}{\delta y}-\frac{\delta F_2(x,y,z)}{\delta z}\\ \frac{\delta F_1(x,y,z)}{\delta z}-\frac{\delta F_3(x,y,z)}{\delta x}\\ \frac{\delta F_2(x,y,z)}{\delta x}-\frac{\delta F_1(x,y,z)}{\delta y}\end{pmatrix} \tag{2.18}$$

Die Länge des resultierenden Vektors, der auch als Rotationsachse interpretiert werden kann, wird als Ausgangswert für entsprechende Transferfunktionen herangezogen. Zuletzt können auch Kombinationen verschiedener Strömungseigenschaften einen aufschlussreichen Untersuchungsgegenstand bilden. So kann beispielsweise die Rotation in Strömungsrichtung analysiert werden. Rechnerisch lässt sich ein entsprechender Wert über die Projektion der Rotationsachse auf den entsprechenden Geschwindigkeitsvektor ermitteln: $heliF\left(x,y,z\right)=rotF\left(x,y,z\right)*F\left(x,y,z\right)$.

Kapitel 3

GPU-basierte Strömungslinien

3.1 Problemdifferenzierung

Den Ausgangspunkt für den Beitrag dieser Arbeit bilden die in Kapitel 2.6 aufgeführten Verfahren. Für die zeitabhängige Integration von Kontrollpunkten wird ein Partikelsystem eingesetzt. Die ermittelten Positionen werden dann zu Linien zusammengefasst. Die Berechnungsergebnisse des Partikelsystems werden in Form einer 2D Textur vorgehalten. Verschiedene Strömungslinien setzen dabei unterschiedliche Vorgehensweisen bei der Ermittlung und Platzierung der benötigten Partikelpositionen innerhalb dieser Textur voraus. Die in dieser Textur gespeicherten Berechnungsergebnisse können als Eingabe folgender Berechnungen dienen. Die Berechnung selbst wird üblicherweise durch entsprechende Shaderprogramme übernommen. Eine konsistente Struktur der jeweiligen Ergebnistextur wird durch eine Lese- und Schreibzugriffkontrolle im Hauptprogramm gewährleistet. Diese Textur wird schließlich als Ausgabe des Partikelsystems in Form eines VBOs zur Verfügung gestellt, welches alle benötigten Vertexpositionen enthält. Da die einzelnen Linienelemente dann aber nicht zwangsläufig in sequenzieller Reihenfolge vorliegen, muss eine entsprechende Korrektur über einen Indexbuffer durchgeführt werden. Im nächsten Abschnitt werden zunächst entsprechende Vorgehensweisen zur Konstruktion von Stream- und Streaklines vorgestellt. Neben der Berechnung der Partikelpositionen wird auch die Konstruktion der zugehörigen Indexbuffer vorgestellt. Dabei ergibt sich die Zugehörigkeit und Reihenfolge der Linienpunkte aus der Position innerhalb der Ergebnistextur, beziehungsweise des daraus resultierenden VBOs.

Die vollständige Realisierung auf der Graphikhardware schafft die Voraussetzung für eine interaktive Ausführung dieses Verfahrens. Jedoch beschränkt sich die bisherige Konfiguration bei der Berechnung der Partikelpositionen auf einen global festgelegten Integrationszeitschritt. Diese Prämisse ergibt sich aus der Konstellation einer parallelen Verarbeitung. Die Berücksichtigung eines adaptiven Zeitschrittes gilt als Voraussetzung einer adäquaten Fehlerkontrolle und -reduktion von Integrationsverfahren, und findet im Rahmen dieser Arbeit besondere Beachtung. Darüber hinaus wird mit dem GS das Leistungsspektrum aktueller Graphikhardware in die Betrachtung einbezogen und eingesetzt.

Schließlich werden unterschiedliche Aspekte der Darstellung der beschriebenen Linienprimitive

in bisherige Überlegungen einbezogen. Dazu zählt beispielsweise eine mehrdimensionale Transferfunktion. Diese kann dazu genutzt werden Regionen besonderen Interesses herauszufiltern, um auf diese Weise von der Darstellung des gesamten Strömungsfeldes zu abstrahieren. Insbesondere bei dichtbesetzten Repräsentationen von Strömungsfeldern kann man anhand dieser Strategie dem Problem der Verdeckung begegnen.

Um neben dem Strömungsverlauf zusätzlich weitere Strömungsinformationen zu visualisieren, wird das ursprüngliche Linienprimitv zuletzt zu einer Strömungsröhre erweitert, um damit eine weitere Informationsdimension auf dem Träger zu generieren.

3.2 Index-basierter Ansatz

Der so genannte Index-basierte Ansatz macht sich die in Bild 2.7 dargestellte Konfiguration zu Nutze. Der Einsatz dieser Konfiguration setzt zunächst eine Untersuchung der Abhängigkeiten des Integrationsprozesses voraus. Die Integration der Partikelpositionen wird in diesem Kontext über die Hauptschleife der zugrunde liegenden Applikation ausgelöst. Die Basis der Berechnung von Partikelpositionen bilden zum einen die Partikelpositionen vorangegangener Berechnungen sowie die zugehörigen Integrationsparameter. Dazu werden die entsprechenden Eingabepositionen in Form von Texturen an den Prozess gebunden (vgl. Bild 2.5). Die zugehörigen Strömungsdaten bilden das zentrale Element der Integration und werden in Form von 3D-Texturen eingebunden. Aus Effizienzgründen kann jedoch nicht der gesamte Texturstack gleichzeitig bereitgestellt werden, da entsprechende Zugriffsoperationen dann jeweils den gesamten Stack abtasten würden. Dies führt zu einer deutlichen Beeinträchtigung der Geschwindigkeit der gesamten Berechnung. Diese Tatsache konnte durch entsprechende Laufzeit-Messungen auf der Geforce 880 GTX bestätigt werden. Die Testsituation bestand darin, innerhalb eines Shaderdurchlaufs aufgrund einer zuvor festgelegten Bedingung, jeweils eine Textur auszuwählen, die zur Ausgabe beitragen sollte. Die verglichenen Konfigurationen ergaben sich aus einer unterschiedlichen Anzahl von 4 beziehungsweise 16 gebundenen RGB-Texturen (Auflösung: 512x512, Datentiefe: 16 Bit). Der Geschwindigkeitsunterschied konnte mit einem Verhältnis von 163 FPS : 52 FPS in einer GLSL-Programmumgebung und 192 FPS : 53 FPS in einer Cg-Programmumgebung angegeben werden.

Folglich werden nur die im aktuellen Zeitschritt relevanten Texturen eingebunden. Eine Zeitvariable, welche durch die Applikation verwaltet wird, steuert die Auswahl der relevanten Texturen. Sind die Strömungsdaten jeweils einem diskreten Abtastzeitpunkt zuzuordnen, so sind für einen Berechnungszeitpunkt jeweils die beiden angrenzenden Abtastzeitpunkte relevant. Bei der Berechnung werden diese Texturen über eine lineare Interpolation unterschiedlich gewichtet. Der Interpolationsfaktor ergibt sich aus der zeitlichen Distanz des Berechnungszeitpunktes zu den angrenzenden diskreten Abtastpunkten (vgl. Kapitel 2.6.3). Nach jedem Integrationsschritt wird der Berechnungszeitpunkt jeweils um die Integrationsschrittweite inkrementiert. Ein Wechsel der gebundenen Texturen findet nur statt, wenn die Zeitvariable nach der Inkrementierung das jeweils aktuelle diskrete Intervall verlässt.

Ist das Ende des Abtastzeitraums erreicht, wird die Zeitvariable über eine Modolo-Operation wieder an den Anfang zurückgesetzt. Auf diese Weise wird die Simulation in einer Schleife ausgeführt.

Neben den Strömungsdaten wird auch die zur Integration verwendete Schrittweite, sowie das eingesetzte Integrationsschema, über das Hauptprogramm ausgewählt. Als Resultat wird der entsprechende Shader zur Berechnung der Integrationsergebnisse gebunden. Zur Darstellung gewünschter geometrischer Primitive wie Streak- oder Streamlines, ist die Vorgehensweise zur Erzeugung der zugrunde liegenden Partikeltextur entscheidend und soll im Folgenden näher erläutert werden.

3.2.1 Streaklines

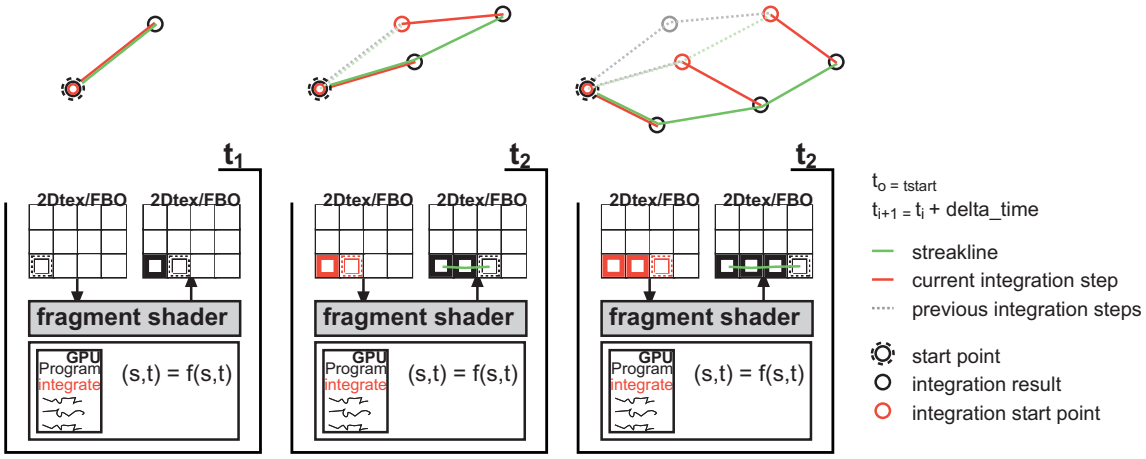
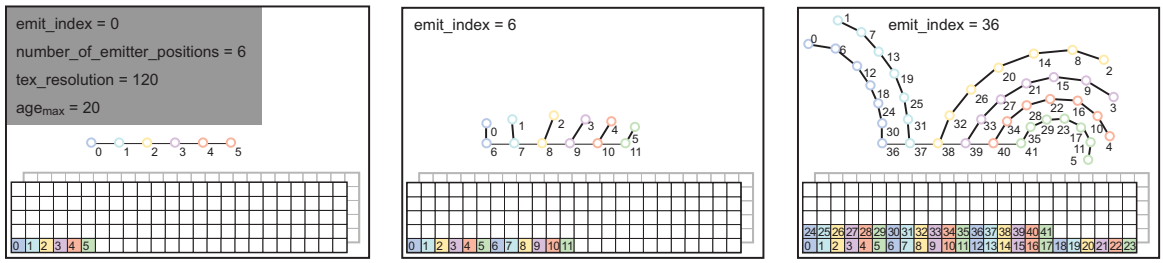


Bild 3.1: Generierung einer Streakline. Die relevanten Partikelpositionen eines Zeitschrittes ergeben sich aus dem jeweils vorhergehenden Zeitschritt. Zusätzlich zur Integration aller vorhandenen Partikelpositionen wird in jedem Zeitschritt ein neues Startpartikel eingefügt. $delta\_time$ entspricht hier der Integrationsschrittweite.

Zur Konstruktion von Streaklines werden einzelne Partikel, die zu unterschiedlichen Zeitpunkten an derselben Position emittiert wurden, in sequenzieller Reihenfolge zu einer Linie zusammengefasst. Die Punkte einer Streakline werden also durch unterschiedlich viele Integrationsschritte aus demselben Startpunkt abgeleitet. Dabei werden die einzelnen Integrationen in aufeinander folgenden Integrationszeitschritten durchgeführt. Zur Integration kann jeweils die aktuelle Partikeltextur als Eingabe gebunden werden. Alle Partikelpositionen, die sich im Definitionsbereich befinden, werden jeweils einen Schritt fortbewegt. Um die Konsistenz der Ausgabetextur zu gewährleisten, werden die neu berechneten Partikelpositionen wieder am gleichen Ort der Ausgabetextur platziert. Zusätzlich wird anschließend für jede Streakline eine weitere Startposition emittiert.

Damit keine relevanten Partikelpositionen überschrieben werden, erfordert deren Platzierung eine genaue Kontrolle. Die Applikation verwaltet dazu einen Index, der die Anzahl der gültigen Partikelpositionen enthält. Dieser Index wird jeweils um die Anzahl der emittierten Partikel inkrementiert. Wird dieser Index als Position innerhalb der Partikeltextur interpretiert, entspricht er der Startposition



(a) (b) (c)

Bild 3.2: Generierung von Streaklines. 3.2(a) stellt den Zustand der Partikeltextur und der entsprechenden Darstellung am Start dar. 3.2(b) und 3.2(c) veranschaulichen die fortlaufende Veränderung.

folgender Partikelplatzierungen. Bild 3.1 beschreibt eine entsprechende Konstellation zur Generierung einer Streakline. Um dieses Konzept auf eine Menge von Streaklines zu erweitern, wird in jedem Verarbeitungsschritt eine Sequenz von Startpositionen emittiert. Diese ergeben sich aus der aktuell ausgewählten Emitterform (vgl. Kapitel 2.10). Als Ergebnis erhält man eine Partikeltextur, deren Partikelsequenzen entsprechend ihres Alters absteigend sortiert sind. Dabei ergibt sich das Alter aus der Summe der Integrationsschritte.

Die maximale Lebensdauer der Partikel ist durch die Auflösung der Partikeltextur sowie der aktuellen Anzahl von Emitterpositionen beschränkt $age_{max} = \frac{max\_particle\_positions}{number\_of\_emitter\_positions}$. In der Darstellung macht sich diese Tatsache durch eine entsprechende Linienlänge bemerkbar. Wird die maximale Lebensdauer eines Partikels überschritten, kann die entsprechende Texturposition durch die darauf folgenden Schreiboperationen überschrieben werden. Diese Situation entsteht, wenn sich der Index zur Kontrolle der Schreiboperationen an die Grenze der Texturauflösung bewegt. Indem der entsprechende Index zurückgesetzt wird, kann in diesem Fall an den Anfang der Textur zurückgekehrt werden. Alte Partikelpositionen können hier überschrieben werden, da sie nunmehr weder für die Darstellung noch für folgende Berechnungen relevant sind.

Auf diese Weise erhält man einen fortlaufenden Algorithmus. Da die Partikeltextur in jedem Schritt um Sequenzen von Partikelstartpositionen erweitert wird, liegen die Linienpunkte nicht in sequenzieller Reihenfolge vor. Eine entsprechende Korrektur wird durch einen Indexbuffer verwirklicht. Dieser legt die exakte Reihenfolge der Eingabepunkte der Pipeline fest. Die Konstruktion dieses Indexbuffers wird primär durch die Anzahl der Emitterpunkte n bestimmt. Dazu werden die einzelnen Partikelpositionen über ihren Index in der Partikeltextur referenziert. Die Menge $M_{P,i}$ aller Partikelpositionen P_{tex_index}, die zur Darstellung einer Streakline $L_{STREAK,i}$ relevant sind, kann also auf folgende Weise definiert werden: $M_{P,i} = \{P_{tex\_index} | tex\_index = k \cdot n + i\}$ mit $k \in \{0, ..., k-1\}$. Dabei entspricht $i \in \{0, ..., n-1\}$ dem Index der Streakline, welche sich jeweils aus der i-ten Emitterposition ergibt. k beschreibt die zuvor ermittelte maximale Linienlänge und entspricht der festgelegten Lebensdauer von Partikeln.

Da sich die Startposition einer Streakline in der Partikeltextur nach jedem Emissionsschritt um n

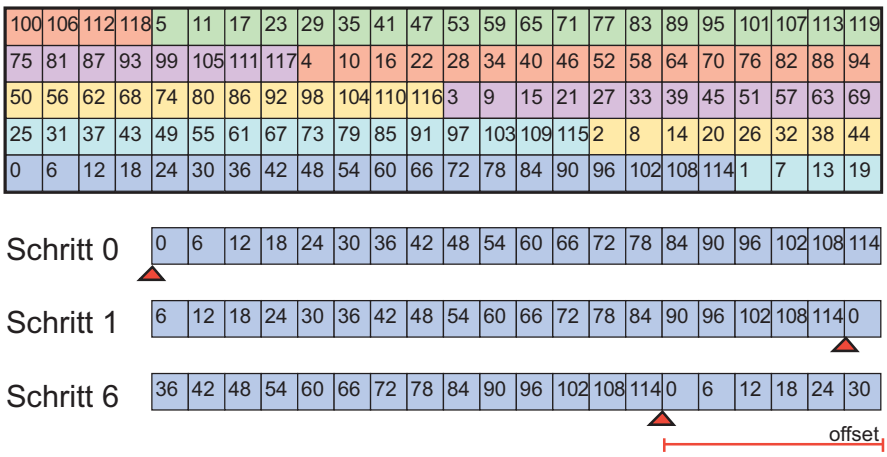
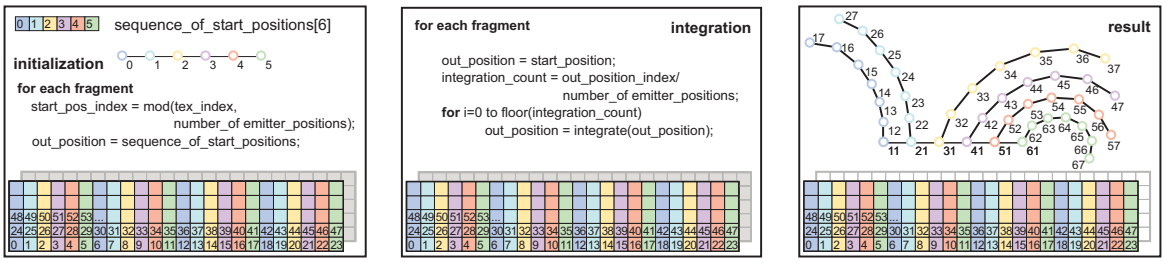


Bild 3.3: Die Struktur des Indexbuffers einer Streakline zur Konfiguration in Bild 3.2. Bei der Darstellung einer Linie muss jeweils ein Offset für den Start einer Streakline berücksichtigt werden.

nach vorne bewegt, muss dieser Offset bei der Verwendung des Indexbuffers berücksichtigt werden (vgl. Bild 3.3). Dazu muss entsprechend ein Zähler $offset$ mit $0 \leq offset \leq k$ in jedem Emissionsschritt inkrementiert werden. Vor dem Zugriff auf einen Index I_{offset}, muss der ursprüngliche Index I_{orig} also um den aktuellen Offset verschoben werden und ergibt $I_{offset} = (I_{orig} + offset \cdot n) \pmod{(n \cdot k)}$. Dabei entspricht $n \cdot k$ der maximal möglichen Partikelanzahl. Die Modolo-Operation verhindert den Zugriff auf undefinierte Speicherplätze.

3.2.2 Streamlines



(a) (b) (c)

Bild 3.4: Generierung von Index-basierten Streamlines. 3.4(a) zeigt die Partikeltextur nach der Initialisierung durch Sequenzen von Emitterstartpunkten. Das Ergebnis der folgenden Integration wird in 3.4(c) dargestellt.

Streamlines geben im Gegensatz zu Streaklines einen zeitlichen Schnappschuss des Vektorfeldes wieder. Aus diesem Grund können alle Partikelpositionen ohne einen Wechsel der zeitabhängigen Strömungsdaten bewerkstelligt werden. Diese werden durch unterschiedlich viele Integrationsschritte

aus den aktuell gültigen Emitterstartpositionen bestimmt. Zu diesem Zweck wird zunächst die gesamte Textur in einem Emissionsschritt mit Sequenzen dieser Startpositionen gefüllt. Über den Index eines Texturfragments kann die entsprechende Startposition identifiziert werden (vgl. Bild 3.4(a)). Als Ergebnis erhalten wir eine vollständige Partikeltextur, die bisher jedoch nur mit k Folgen derselben Startpositionen gefüllt ist, mit $k = \frac{max\_particle\_positions}{number\_of\_emitter\_positions}$. Im darauf folgenden Schritt wird dann eine erforderliche Integration durchgeführt werden. Ebenfalls in Abhängigkeit von dem Index des betreffenden Texturfragments werden auf den Startpositionen unterschiedlich viele Integrationsschritte durchgeführt. Die erforderliche Anzahl von Integrationsschritte entspricht der Emissionsfolge, der die entsprechende Partikelposition zuzuordnen ist. Diese ergibt sich aus der entsprechenden Texturkoordinate (vgl. Bild 3.4(b)). Damit enthält die Partikeltextur alle zur Darstellung benötigten Partikelpositionen.

Die Konstruktion des Indexbuffers verläuft analog zu Streaklines. Die Partikelintegration setzt dabei in jedem Zeitschritt wieder am Anfang der Textur an. Aus diesem Grund entfällt die Berücksichtigung eines Offsets. Im Vergleich zu Streaklines, bei denen für jede Partikelposition nur jeweils ein Integrationsschritt durchgeführt wird, hängt die Anzahl der erforderlichen Integrationsschritte von der Länge der entsprechenden Streamline ab und entspricht $\sum_{i=1}^{n} i$ mit $n = line\_length - 1$. Die zur Integration benötigten Texturinterpolationen wirken sich beachtlich auf die Geschwindigkeit aus.

3.3 Zeitschrittadaptivität

Mit dem geschilderten Mechanismus lassen sich bereits beeindruckende Ergebnisse insbesondere hinsichtlich der Effizienz der Partikelintegration erzielen. Jedoch bleibt die Fehlerkontrolle eingesetzter Integrationsverfahren stark eingeschränkt. In [TGE97] werden Integrationsmethoden für diskrete Daten analysiert. Die eingesetzte Integrationsschrittweite beeinflusst demnach die Präzision der Integrationsmethoden. Eine optimale Schrittweite kann im Allgemeinen jedoch nicht übergreifend für das gesamte Vektorfeld angegeben werden, da sie an lokale Charakteristika des Vektorfeldes gebunden ist. Die beschriebene Vorgehensweise geht jedoch von einem konstanten Zeitschritt aus. Dieser wird insbesondere zur zeitlichen Synchronisation der Erzeugung von Streaklines eingesetzt und ist eine Prämisse der parallelen Verarbeitung. Der beschriebene Mechanismus verwendet dafür eine globale Integrationsschrittweite. Im Folgenden sollen die Bedingungen der Integrationen einer adaptiven Schrittweite in das aktuelle Framework evaluiert werden.

3.3.1 Streamlines

Die dargestellten Verfahren zur Erzeugung von Stream- und Streaklines unterscheiden sich deutlich bezüglich ihrer Affinität zur Integration eines adaptiven Zeitschrittes. Diese resultiert im Wesentlichen aus den theoretischen Umständen dieser Linienprimitiven, welche die Grundlage der folgenden Erörterung bilden. Streamlines können durch die folgende Gleichung definiert werden:

$$\frac{d\vec{x}}{ds} = \vec{v}\left(\vec{x}\right) \tag{3.1}$$

$$\vec{x}(s, x_0) = \vec{x_0} + \int_{s=0}^{t} \vec{v}(s)ds \tag{3.2}$$

x_0 entspricht in diesem Zusammenhang dem Startpunkt der Streamline, die über einen Integrationszeitraum t definiert ist. Aus 3.1 lässt sich mit $s \mapsto \vec{x}\left(s\right)$ eine Parametrisierung der Streamline ableiten. Die Länge der Streamline ergibt sich somit aus dem Zeitintervall t, über welches jeweils ein Partikel verfolgt werden soll. Da die Streamline zu einem festen Zeitpunkt generiert wird, bestehen keinerlei weitere Abhängigkeiten. Diese Tatsache ermöglicht die unmittelbare Implementierung eines adaptiven Zeitschrittes. Im diskreten Fall ergibt sich der Definitionsbereich von $\vec{x}\left(s\right)$ aus den Integrationszeitschritten, die bei der Partikelintegration verwendet wurden. In der bisherigen Vorstellung wird die Streamline an äquidistanten Zeitpunkten ausgewertet. Durch die Verwendung eines adaptiven Zeitschrittes wird die Auflösung der Auswertung an entsprechende Vektorfeldcharakteristika gebunden. Eine Anpassung der Schrittweite ermöglicht die Gewährleistung eines gewissen Maßes an Präzision der Auswertung. Der Interpolationsfehler, der sich aus dem diskreten Vektorfeld ergibt, muss in diesem Zusammenhang als obere Schranke des Integrationsfehlers vorausgesetzt werden.

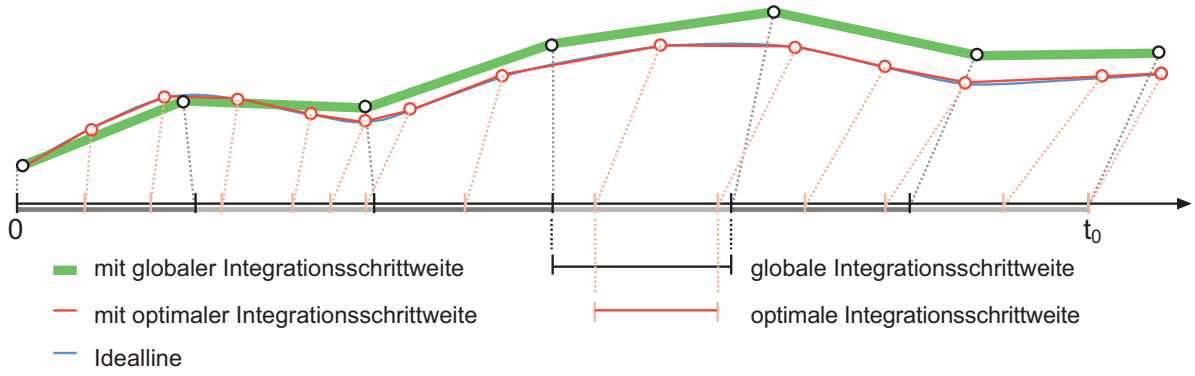


Bild 3.5: Adaptive Streamlines. Vergleich der Streamlinegenerierung mit und ohne Zeitschrittanpassung.

Da sich die optimale Schrittweiten aus lokalen Vektorfeldeigenschaften ergeben, ist auch die Anzahl der benötigten Integrationsschritte nicht festgelegt. Diese ergibt sich aus der Positionierung der relevanten Partikel innerhalb des Vektorfeldes und kann sich aufgrund eines dynamischen Vektorfeldes zeitabhängig ändern. Derartige laufzeitabhängige Anpassungen sind anhand der in Kapitel 3.2.2 dargestellten Konfiguration nicht ohne weiteres realisierbar. Mit dem Geometry Shader wird jedoch relativ zur traditionellen Rendering-Pipeline eine weitere Dimension der Kontrolle des Berechnungsprozesses verwirklicht. Diese ermöglicht das Hinzufügen und Löschen einzelner Primitive aufgrund von Laufzeiteigenschaften.

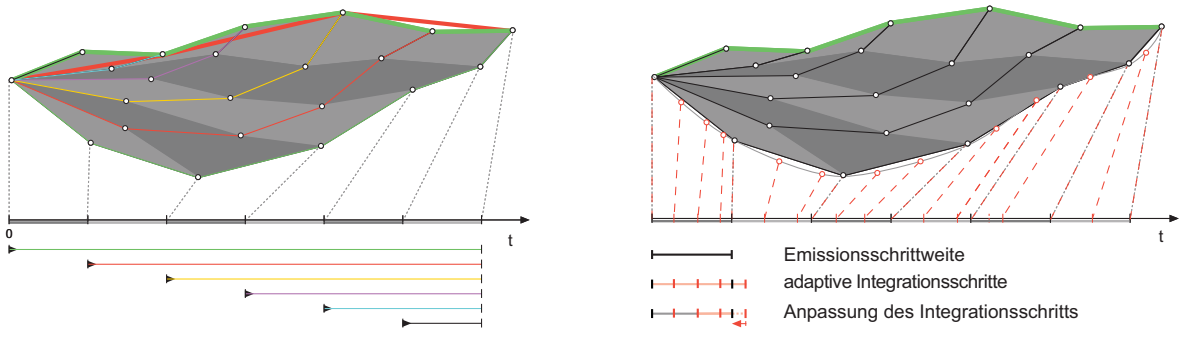
Der entscheidende Vorteil, der sich aus der Verwendung des GS ergibt, liegt in der Ausgliederung des Entwicklungsprozesses einzelner Strömungslinien in separate GS-Durchläufe. Als Eingabe erhält der GS jeweils nur einen Startpunkt der Streamline. In einem Durchlauf kann nun theoretisch die gesamte Streamline erzeugt werden. Man löst sich dadurch grundsätzlich von der Synchronisation zwischen unterschiedlichen Strömungslinien und erhält eine Reihe individueller Kontroll- und Steuerungsoptionen wie eine unabhängige Schrittweitenkontrolle der Integration, sowie eine variable Anzahl von möglichen Integrationsschritten. Das Ausgabeprimitiv ist nicht durch eine feste Anzahl zugehöriger Punkte beschränkt.

3.3.2 Streaklines

Streaklines verfolgen einzelne Partikel, die an einer konkreten Position in der Strömung ausgesetzt werden, über einen festgelegten Zeitraum t_0 hinweg. Sie können generell mit folgender Gleichung definiert werden:

$$\frac{d\vec{x}_P}{dt} = \vec{u}_P(\vec{x}_P, t) \quad \vec{x}_P(t = \tau_P) = \vec{x}_{P0} \tag{3.3}$$

Die Streakline $\vec{x}_P(t, \tau_P)$ wird durch τ_P mit $0 \leq \tau_P \leq t_0$ parametrisiert. Im diskreten Fall wird das Zeitintervall t_0 in entsprechende Zeitintervalle unterteilt. Die Unterteilungspunkte können in diesem Zusammenhang auch als Emissionszeitpunkte der für die Streakline relevanten Partikelpositionen interpretiert werden. Die in aufeinander folgenden Zeitschritten emittierten Partikel werden jeweils entsprechend dieses Zeitintervalls fortbewegt.



(a) (b)

Bild 3.6: Integration eines adaptiven Integrationszeitschrittes für Streaklines. 3.6(a) zeigt die zeitlichen Abhängigkeiten der Streakline-Generierung. Die einzelnen Integrationsschritte sind durch Grauabstufungen abgesetzt. Die farblich gekennzeichneten Linien unterhalb der Abbildung 3.6(a) beschreiben das Zeitintervall, das der Integration eines Streaklinepunktes zugrunde liegt. Dieses Intervall wird in Bezug zum gesamten zeitlichen Ablauf gesetzt. Der Vergleich der roten bzw. grünen Streakline veranschaulicht den Einfluss des Emissionsintervalls auf die Stetigkeit der Kurve. Schließlich wird in 3.6(b) ein möglicher Ansatz zur Berücksichtigung einer adaptiven Integrationsschrittweite vorgeschlagen. Dazu wird der Emissionsschritt in adaptive Integrationsschritte unterteilt.

Die Präzision der Streakline ergibt sich im Wesentlichen aus der Größe der Zeitintervalle zwischen aufeinander folgenden Emissionszeitpunkten (vgl. Bild 3.6(a)). In der bisherigen Vorstellung entspricht die Schrittweite, die zur Partikelintegration verwendet wird, diesem Zeitintervall.

Generell lässt sich analog zu Streamlines auch für Streaklines ein Algorithmus entwickeln, der einen adaptiven Zeitschritt berücksichtigt. Die Punkte der Streakline ergeben sich hier aus der Partikelbahnberechnung einzelner Partikel, die zu unterschiedlichen Zeitpunkten an der gleichen Position gestartet sind. Der Startzeitpunkt bestimmt dabei die Länge der Integration (vgl. Bild 3.6(b)). Dieses Intervall kann beliebig in adaptive Zeitschritte unterteilt werden. Die Berechnung einzelner Streaklines kann sich je nach Streaklinelänge auf mehrere diskrete Zeitschritte erstrecken, und setzt aus diesem Grund die Verfügbarkeit entsprechender Strömungsdaten voraus. Stellt man aber den gesamten Texturstack an Strömungsdaten bereit, führt dies bei der Texturinterpolationen wie schon erwähnt zu erheblichen Geschwindigkeitseinbußen (vgl. Kapitel 3.2). Die Realisierung von Streaklines mit einer adaptiven Schrittweitenkontrolle, erfordert also vorab die Lösung dieser Problematik.

Der in Kapitel 3.5 geschilderte Algorithmus geht von der Generierung von Streaklines aufgrund der in Kapitel 3.2.1 geschilderten Konfiguration aus. Diese beruht auf einer Synchronisation von Teilberechnungen in unterschiedlichen Zeitschritten. Die Taktfrequenz ist dabei durch das Emissionsintervall festgelegt. Die Integrationsschrittweite ist in der bisherigen Konstellation direkt von dieser Taktfrequenz der Synchronisation abhängig.

Für die Umsetzung einer adaptiven Schrittweite müssen die Abhängigkeiten der Integrationsschrittweite von dem Emissionsintervall gelöst werden. Indem das gewählte Emissionsintervall lediglich als obere Grenze der Summe der zu verwendenden Integrationsintervalle interpretiert wird, lassen sich beispielsweise variable Integrationszeitschritte realisieren (vgl. Bild 3.6(b)). In einem Verarbeitungsschritt wird durch eine variable Anzahl von Integrationsschritten das gesamte Emissionsintervall integriert. Geht die Summe der optimalen Integrationsschrittweiten über das Emissionsintervall hinaus, findet eine entsprechende Korrektur der aktuellen Integrationsschrittweite statt, sodass lediglich über den Emissionszeitraum integriert wird. Auf diese Weise kann neben der Auflösung der Kurve auch eine adäquate Fehlerkontrolle des Integrationsprozesses realisiert werden.

3.3.3 Preprocessing der Daten

Die Genauigkeit der Partikelintegration basiert maßgeblich auf der eingesetzten Integrationsschrittweite. Im Rahmen dieser Arbeit wird für eine adäquate Fehlerkontrolle auch ein heuristisches Verfahren eingesetzt. Dabei wird zunächst vom Integrationsschema der klassischen Runge-Kutta-Integration 4. Ordnung ausgegangen. In diesem speziellen Fall lässt sich eine einfache Regel zur Ermittlung des optimalen Integrationsschrittes angeben. Dieser wird in einem Vorverarbeitungsschritt aufgrund der in Kapitel 2.2.2 vorgeschlagenen Regel iterativ ermittelt.

Als Eingabe des folgenden Algorithmus dient der gesamte Strömungsdatensatz. Dieser wird bezüglich einer Standardintegrationsschrittweite ausgewertet. Der optimierte Integrationsschritt wird dem Strömungsdatensatz hinzugefügt und analog zur Geschwindigkeit interpoliert. Natürlich müssen

innerhalb der Applikation entsprechende Speicherstrukturen angepasst werden, damit auf diese Komponenten zugegriffen werden kann.

```
1  foreach discrete position at every time_step
2          q_factor = generateQ( position, standard_step );
3          optimized_integration_step = standard_step;
4
5          while ( q_factor > top_threshold )
6
7                  optimized_integration_step /= 2.0f;
8                  q_factor = generateQ( optimized_step );
9
10         if (q_factor < bottom_threshold)
11                 while ( q_factor < bottom_threshold)
12
13                         optimized_integration_step *= 2.0f;
14                         q_factor = generateQ( optimized_step );
15
16         save(optimized_integration_step);
```

Listing 3.1: Berechnung des optimierten Integrationsschrittes.

3.4 Geometry-Shader-basierte Streamlines

Die Berechnung von Streamlines basiert konzeptionell auf Strömungsdaten zu einem festen Zeitpunkt. Dabei besteht grundsätzlich keine Relation zwischen Partikeln unterschiedlicher Streamlines. Da eine Synchronisation also ohnehin überflüssig ist, findet sich die ideale Basis für die Integration eines adaptiven Zeitschrittes.

Die Darstellung von Streamlines unter Verwendung des GS verläuft prinzipiell analog zur Vorgehensweise, wie sie für das Index-basierte Verfahren vorgestellt wurde. Die wesentlichen Schritte sind demnach durch Initialisierung, Integration und schließliche Darstellung relevanter Partikelpositionen charakterisiert. Durch den Einsatz des GS ist man in der Lage, die einzelnen Streamlines unabhängig voneinander zu behandeln. Als Eingabeprimitiv wird jeweils ein Startpunkt der Streamline übergeben. Das Ausgabeprimitiv soll zuletzt die gesamte Strömungslinie enthalten. Für jeden Startpunkt wird jeweils ein Aufruf des GS-Programms initiiert. Ausgehend von dem Startpunkt können sukzessive Integrationsschritte durchgeführt werden, bis die gewünschte Linienlänge erreicht ist. Wie üblich kann das Ausgabeprimitiv anschließend im Fragmentshader verarbeitet werden.

In der Praxis ist der GS während eines Durchlaufs durch eine hardwareabhängige Obergrenze der maximal möglichen Ausgabeelemente beschränkt. Aus diesem Grund muss die Generierung einer Strömungslinie gegebenenfalls auf mehrere Durchläufe verteilt werden. Im vorliegenden Fall wurde eine segmentweise Unterteilung der Erzeugung gewählt. Dabei besteht ein Segment aus vier Punkten.

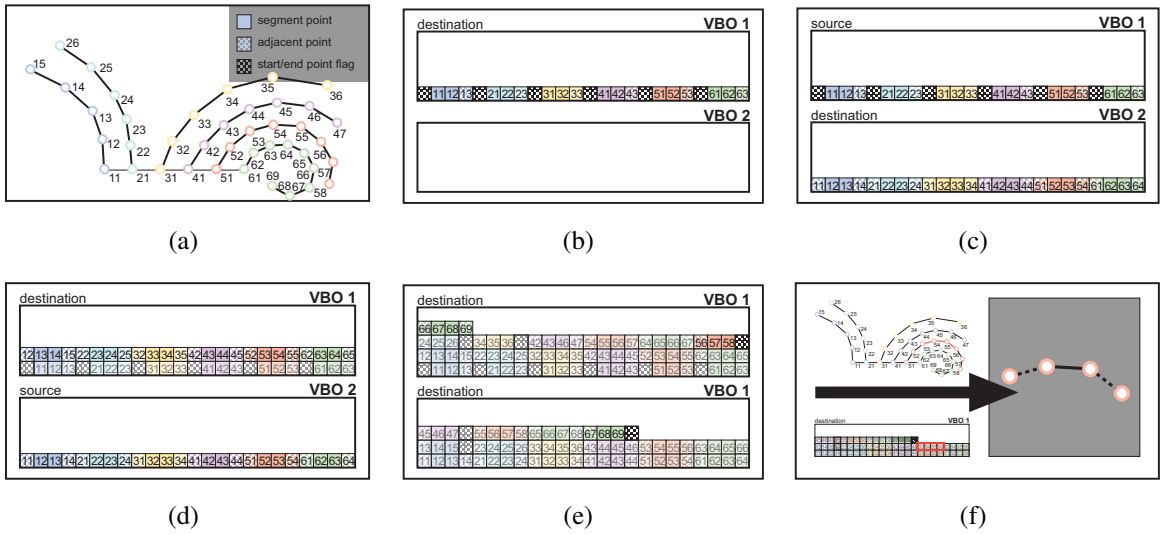


Bild 3.7: Generierung von Streamlines anhand des GS. Bild 3.7(a) veranschaulicht zunächst das erwartete Ergebnis. Bild 3.7(b) - Bild 3.7(d) beschreiben die ersten drei Schritte beim abwechselnden Schreiben des entsprechenden VBOs. Bild 3.7(e) verdeutlicht dann den letzten Schritt. Bild 3.7(f) zeigt die Zusammenhänge zwischen VBO, Segment und Streamline.

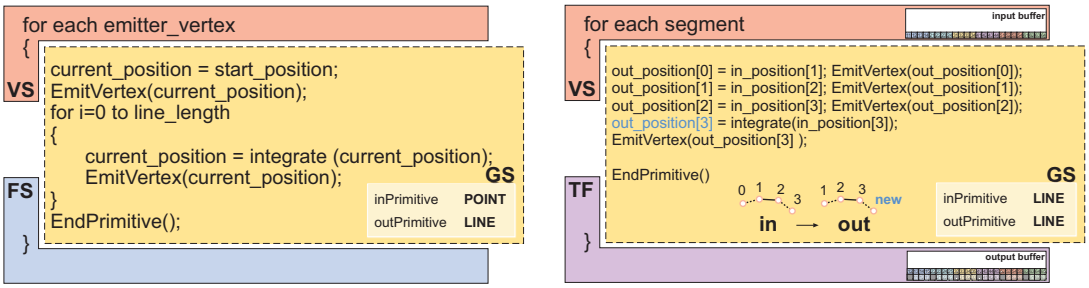


Bild 3.8: Aufruf und Durchlauf des GS.

Die beiden inneren Punkte stellen jeweils die Endpunkte des Liniensegments dar. Darüber hinaus werden zwei weitere angrenzende Punkte gespeichert, welche im weiteren Verlauf beispielsweise zur Berechnung von Tangenten genutzt werden können. Auf diese Weise erhalten wir die kleinste Einheit unabhängiger Liniensegmente.

Zur Speicherung der Startpositionen sowie folgender Integrationsergebnisse dient ein so genannter Transform Feedback Buffer. Dieser wird in Form eines Doublebuffers realisiert, um einen gegenläufigen Zustandswechsel analog zum Framebuffer zu gewährleisten. Das Ergebnis eines GS-Durchlaufs bilden vier Punkte, die sich aus aufeinander folgenden Integrationsschritten ergeben. In jedem weiteren Durchlauf wird die Integration jeweils einen Schritt fortgesetzt. Dabei dient das Segment eines

GS-Durchlaufs als Eingabe des folgenden. Die letzten drei Punkte des Eingabesegments können direkt zum Ausgabeprimitiv hinzugefügt werden. Lediglich der letzte Punkt ergibt sich aus der Integration der letzten Eingabeposition. Um diese Vorgehensweise generieren zu können, muss der TF-Buffer zunächst anhand eines GS-Programms initialisiert werden. Anhand eines VBOs werden dazu die Integrationsstartpositionen in die Rendering-Pipeline übergeben. Der GS erzeugt jetzt das erste Grundelement einer Linie. Um Start- und Endsegmente zu kennzeichnen wird der erste beziehungsweise letzte Punkt des entsprechenden Segments durch eine Position gekennzeichnet, die außerhalb des Einzugsbereichs der Integration liegt. Auf diese Weise können diese im späteren Verlauf als spezielle Punkte identifiziert und individuell behandelt werden. Das bisherige Ergebnis wird nun in den momentan gebundenen Zielbuffer ausgegeben. Dazu muss vor der Aktivierung des entsprechenden Shaders die Funktionalität des TF aktiviert werden. Um eine Manipulation des aktuellen Framebuffers zu unterbinden, muss zusätzlich die Rasterisierung deaktiviert werden. Diese Funktionalität wird durch die Erweiterung des TF zur Verfügung gestellt. Das Ergebnis der Initialisierung kann nun in Form eines VBOs als Eingabe der nachfolgenden Integration dienen. Um einen Zustandswechsel der betreffenden Buffer zu minimieren, wird dieser erst durchgeführt, nachdem alle Strömungslinien jeweils einen Integrationsschritt durchgeführt haben. Die Sicherung der Konsistenz der Buffer wird zum einen durch die Ausgabe von vollständigen Primitiven gewährleistet. Zum anderen kann die Anzahl der in einem Integrationsschritt erzeugten Primitive durch eine asynchrone Abfrage ermittelt werden, was die korrekte Positionierung des Zeigers für folgende Schreib- und Leseoperationen sichert. Für jeden Buffer wird dazu die Anzahl der enthaltenen Elemente gespeichert. Die jeweils nächste Position kann für folgende Schreiboperationen genutzt werden, ohne Positionen zu überschreiben. Die Anzahl der im letzten Schritt erzeugten Elemente kann zur Positionierung des Lesezugriffs in nachfolgenden Schritten genutzt werden. Dieser Vorgang kann sich je nach gewünschter Linienlänge beliebig oft wiederholen. Die Größe des Buffers fungiert dabei als obere Grenze. Abschließend kann der Inhalt des gesamten Buffers in gewünschter Form dargestellt werden. Die einzelnen Segmente können dabei unabhängig voneinander behandelt werden.

3.5 Geometry-Shader-basierte Streakline

Die Integration eines variablen Zeitschrittes zur Erzeugung von Streaklines setzt eine Reihe von Anpassungen voraus. Eine direkte Umsetzung der von GS-basierten Streamlines bekannten Vorgehensweise würde wie schon erwähnt die simultane Verfügbarkeit der gesamten Strömungsdaten voraussetzen. Da dies jedoch im Allgemeinen aus Gründen der Geschwindigkeit und des Speicherbedarfs nicht realisierbar ist, bedarf es bei der Generierung der Streaklines wieder einer zeitlichen Synchronisation. In den einzelnen Synchronisationsschritten ist dann wieder nur ein Bruchteil der gesamten Strömungsdaten relevant. Im folgenden Ansatz wird die zeitliche Synchronisation nicht mehr an eine globale Integrationsschrittweite gebunden, sondern basiert auf einer prinzipiell frei wählbaren zeitlichen Distanz. Idealerweise wird hier ein Bruchteil der zeitlichen Auflösung des zugrunde liegenden

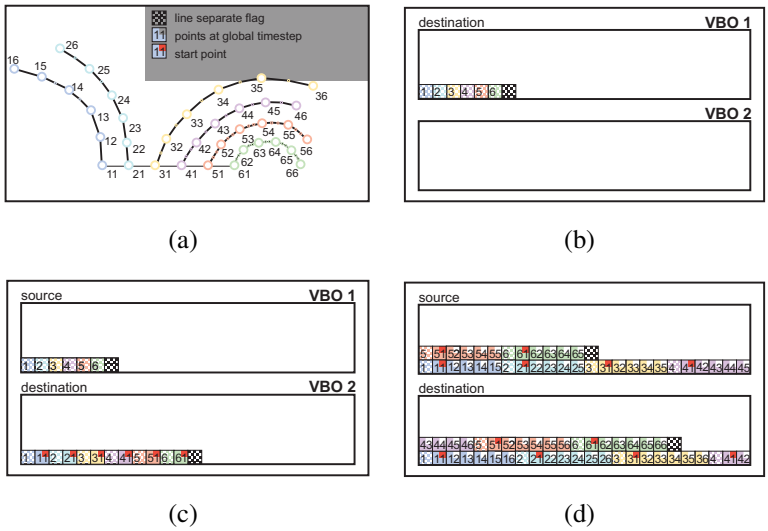


Bild 3.9: Generierung von GS-basierten Streaklines mit adaptivem Integrationszeitschritt. Bild 3.9(a) zeigt zunächst das erwartete Ergebnis nach sechs Integrationsschritten. Die darauf folgenden Abbildungen beschreiben den prinzipiellen Ablauf der Initialisierung und Advektion. Bild 3.9(d) verdeutlicht dem Zustand des TF Buffers entsprechend zu Bild 3.9(a).

Datensatzes angenommen. Auf diese Weise können für jeden Synchronisationsschritt die jeweils relevanten Strömungsdaten eindeutig identifiziert werden. Innerhalb eines Synchronisationsschrittes kann dann eine variable Anzahl adaptiver Integrationsschritte vorgenommen werden. Eine entsprechende Vorgehensweise wird in Bild 3.9 dargestellt.

Grundsätzlich kommt auch hier analog zu Streamlines der Geometry Shader sowie eine Konfiguration von TF-Buffern zum Einsatz. Eine Unterteilung des entsprechenden Buffers in unterschiedliche Streaklines wird durch speziell gekennzeichnete Punkte vorgenommen. Durch eine eindeutige Indizierung dieser Punkte kann jede Streakline einer entsprechenden Emitterposition zugeordnet werden. Zunächst werden also die Unterteilungspunkte in den dafür vorgesehenen Buffer geschrieben. Ein weiterer spezieller Punkt kennzeichnet die Berechnungsgrenze. Über die Berechnungsgrenze hinaus sind keine gültigen Positionen mehr zu erwarten. Im folgenden Synchronisationsschritt dient der soeben erzeugte TF-Buffer als Eingabe. Für jeden Unterteilungspunkt wird nun über den zugehörigen Index eine Startposition erzeugt. Neben der Positionsinformation wird das entsprechende Alter der Partikel gespeichert. Um die beschriebene Struktur des Buffer zu erhalten wird zunächst eine Kopie des Unterteilungspunktes in den Ausgabebuffer emittiert. Daraufhin wird der erzeugte Startpunkt geschrieben. Das Ergebnis (vgl. Bild 3.9(d)) enthält nun Unterteilungspunkte sowie gültige Startpunkte. Diese werden in folgenden Schritten unterschiedlich behandelt. Für jeden Unterteilungspunkt wird wie bisher eine Kopie des Unterteilungspunktes sowie ein zugehöriger Startpunkt erzeugt. Alle weiteren Punkte werden als Ausgangspositionen für die folgende Integration interpretiert. Ausgehend von dieser Position, wird über die gesamte Synchronisationsschrittweite integriert. Dazu werden

sukzessive optimale Schrittweiten genutzt, bis das Ende des Synchronisationsintervalls erreicht ist. Nach jedem Integrationsschritt wird das entsprechende Intervall um die Integrationsschrittweite verringert. Ist der optimale Zeitschritt größer als das übrige Intervall, wird dieses als Integrationsschrittweite verwendet. Das Ergebnis der letzten Integration wird schließlich in den Speicher geschrieben. Das Alter des Ausgabepunktes ergibt sich aus der Summe des Alters des Eingabepunktes und der Synchronisationsschrittweite. Entspricht das Alter eines Punktes einer zuvor spezifizierten maximalen Lebensdauer, wird der betreffende Punkt nicht mehr weiterverfolgt und verworfen. Natürlich wechseln nach jedem Synchronisationsschritt die beiden TF-Buffer gegenläufig ihren Zustand als Eingabe- beziehungsweise Ausgabebuffer.

Der Ausgabebuffer eines Synchronisationsschrittes kann direkt zur Darstellung der Streakline verwandt werde. Wie in Kapitel 3.3.2 dargestellt, kann die Synchronisationsschrittweite als Emissionsfrequenz interpretiert werden. Somit bestimmt dieses Intervall direkt die Auflösung der entsprechenden Streakline. Die Darstellung des gesamten Buffers entspricht der Vorgehensweise bei Streamlines.

3.6 Darstellung

Die bisherige Darstellung beschreibt zunächst nur die Ermittlung der relevanten Partikelpositionen, die zur Darstellung verschiedener Strömungslinien benötigt werden. Die Linienprimitive ergeben sich dabei durch eine entsprechende Verbindung der erzeugten Partikelpositionen. Über die unmittelbare Darstellung hinaus, können diese Linienprimitive durch weitere Verfahren um eine Reihe visueller Effekte erweitert werden, die den Informationsgehalt sowie die Wahrnehmung der Visualisierung deutlich steigern können.

3.6.1 Stylized Line

Darüber hinaus kann ein effizientes Verfahren zur Strömungsvisualisierung genutzt werden, welches die ursprüngliche Auffassung einfacher Linienprimitive um visuelle Attribute wie beispielsweise einer definierbaren Liniendicke erweitert [SGS05]. Generell wird die Linie durch ein geschlossenes 2D-Profil zu einer zylinderförmigen Röhre erweitert.

Die Konstruktion verläuft segmentweise auf Basis zweier jeweils aufeinander folgender Linienpunkte. Dazu wird zunächst das lokale Koordinatensystem an den Segmentendpunkten ermittelt, welches zur Synchronisation entsprechender Profile dient. Die Ermittlung des lokalen Koordinatensystems kann aufgrund der Tangente und der Betrachterposition auf Basis folgender Gleichungen bestimmt werden (siehe Bild 2.4(a)).

$$\hat{y}_i = \frac{\hat{t}_i \times \vec{u_\perp}}{\|\hat{t}_i \times \vec{u_\perp}\|} \tag{3.4}$$

$$\hat{x}_i = \frac{\hat{t}_i \times \hat{y}_i}{\|\hat{t}_i \times \hat{y}_i\|} \tag{3.5}$$

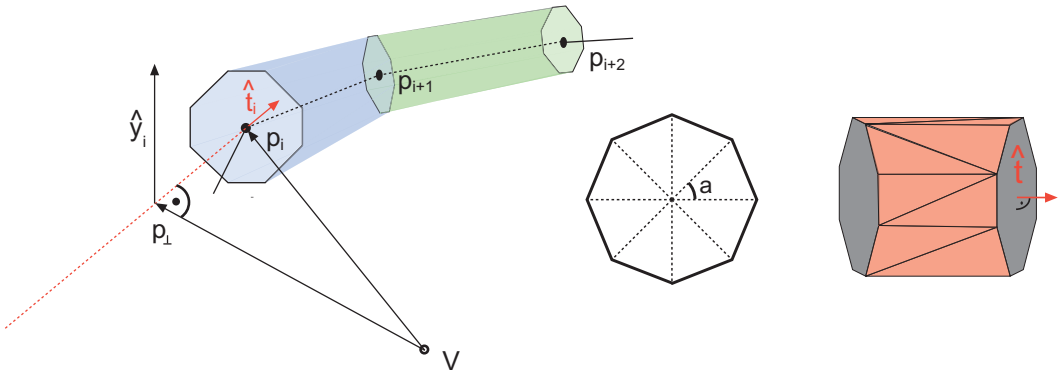


Bild 3.10: Generierung von Stylized Lines

Zur Ermittlung der Tangente können jeweils adjazente Linienpunkte herangezogen werden. Die in Kapitel 3.4 dargestellte Konfiguration stellt diese jeweils segmentweise zur Verfügung. Aufgrund einer Profildefinition, beispielsweise $\Pi(\phi) = r_i \cdot \begin{pmatrix} \sin(\phi) \\ \cos(\phi) \end{pmatrix}$ mit $0 \leq \phi \leq 2\pi$, können diskrete Profilpunkte ermittelt werden. Die generierten Profilpunkte können dann in entsprechender Reihenfolge zu Polygonen zusammengesetzt werden, und beschreiben die Oberflächenstruktur der Segmente.

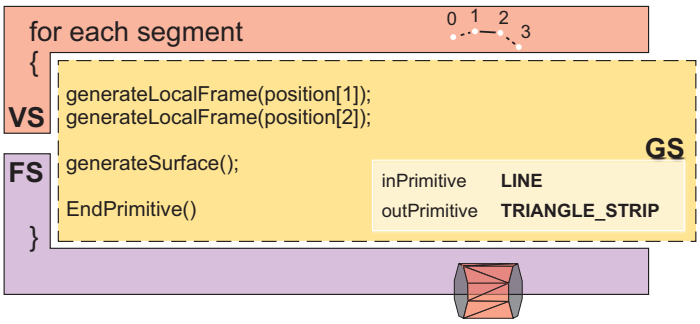


Bild 3.11: Listing Stylized Lines

Zur Implementierung kann in diesem Kontext auch der GS eingesetzt werden. Dabei kann in einem Durchlauf der Zugriff auf alle zuvor generierten Segmentpunkte gewährleistet werden. Die Ausgabe eines Durchlaufs bilden dann alle Oberflächenpolygone des Segments. Im Fragmentshader kann dann eine entsprechende Beleuchtungsberechnung vorgenommen werden. Soll über die Liniendicke eine bestimmte Strömungseigenschaft kodiert werden, wird während der Berechnung jeweils der Radius r_i der Profildefinition angepasst. Der entsprechende Wert muss dann aber in der Vertexdefinition vorliegen.

Zur Beschleunigung dieses Verfahrens wird in [SGS05] darüber hinaus eine Approximation der Oberflächenstruktur durch ein einziges Polygon realisiert. Die Polygoneckpunkte können als Silhouettepunkte eines Segments interpretiert werden. Zusätzlich zur Position der Eckpunkte werden alle zur

Texturierung und Beleuchtungsberechnung benötigten Variablen ermittelt und an den Fragmentshader übergeben. Da eine entsprechende Reduktion auf eine 2D-Fläche nicht immer möglich ist, muss in diesem Fall wieder auf die bekannte Methode der Generierung einer vollständigen Oberflächenstruktur zurückgegriffen werden.

3.6.2 Visualisierung von Strömungseigenschaften

Wie in Kapitel 2.7 dargestellt, lassen sich bestimmte Strömungseigenschaften direkt aus lokalen Gegebenheiten ermitteln. Zur Bestimmung der Strömungsgeschwindigkeit genügt es beispielsweise, den zugrunde liegenden Datensatz einmal an entsprechender Stelle abzutasten. Weitere Strömungseigenschaften ergeben sich aus einer geeigneten Kombination lokaler Ableitungen.

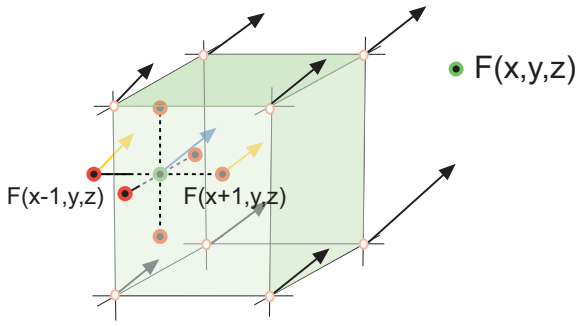


Bild 3.12: Abtastpunkte für die Approximation des Gradienten durch zentrale Differenzen in der Umgebung eines Punktes.

An diskreten Positionen innerhalb des Vektorfeldes kann zu diesem Zweck ein Tensor in Form einer 3×3-Matrix bestimmt werden, welcher sich aus partiellen Ableitungen in Richtung der kartesischen Koordinatenachsen zusammensetzt. Eine Approximation der partiellen Ableitungen wird über zentrale Differenzen realisiert. Dazu müssen die Strömungsdaten jeweils in der Umgebung der betreffenden Position mehrfach abgetastet werden. Bild 3.12 veranschaulicht die entsprechenden Samplepositionen. Dabei muss eine geeignete Abtastdistanz zur Ursprungsposition gewählt werden. Bei einer uniformen Verteilung der zugrunde liegenden Strömungsdaten entspricht diese Distanz idealerweise dem Abstand der Messpositionen, welcher sich aus der Auflösung des Datensatzes ergibt. Um bei der Partikelintegration von der Auflösung und Verteilung des zugrunde liegenden Datensatzes zu abstrahieren, wird das Vektorfeld $F(x, y, z)$ auf einen Bereich mit $x, y, z \in [0, 1]$ abgebildet. Die Sampledistanz ergibt sich aus der Ausdehnung des Vektorfeldes $fieldSize$ sowie einer gewünschten Sampledistanz $realDistance$ zu $sampleDistance = \frac{fieldSize}{realDistance}$. Eine entsprechende Berechnung des Gradienten kann direkt an die Integrationsberechnung jeweils aktueller Partikelpositionen angeschlossen werden (vgl. Listing 3.2). Dem betreffenden Shader muss dazu die Ausdehnung des Vektorfeldes, sowie die gewünschte Sampledistanz zur Verfügung gestellt werden.

```
delta_x = real_distance/vFieldSize.x;
delta_y = real_distance/vFieldSize.y;
delta_z = real_distance/vFieldSize.z;
foreach position as result of particle tracing
        sample_f_x_-_1 = sampleTField(position-vec(delta_x,0,0), currentTime);
        sample_f_x_+_1 = sampleTField(position+vec(delta_x,0,0), currentTime);
        sample_f_y_-_1 = sampleTField(position-vec(0,delta_y,0), currentTime);
        sample_f_y_+_1 = sampleTField(position+vec(0,delta_y,0), currentTime);
        sample_f_z_-_1 = sampleTField(position-vec(0,0,delta_z), currentTime);
        sample_f_z_+_1 = sampleTField(position+vec(0,0,delta_z), currentTime);
        gradientTensor = mat3(
                0.5*(sample_mag_x-sample_min_x),
                0.5*(sample_mag_y-sample_min_y),
                0.5*(sample_mag_z-sample_min_z)
                );
```

Listing 3.2: Berechnung des Gradienten.

```
getHSVColor(float x){
        vec3 result;
        float sat = 1.0;
        float value = 1.0;
        float scale = 1.0;
        float bias = 0.0;
        float hue = 6.0*scale*x+bias;
        if (hue > 6.0) hue = 0.0;
        //convert to RGB
        float i = floor(hue);
        float f = hue-i;
        float p = value*(1.0-sat);
        float q = value*(1.0-(sat*f));
        float t = value*(1.0-(sat*(1.0-f)));
        if (i < 1.0)
        result = vec3(value, t, p);
        else if (i < 2.0)
        result = vec3(q, value, p);
        else if (i < 3.0)
        result = vec3(p, value, t);
        else if (i < 4.0)
        result = vec3(t, p, value);
        else
        result = vec3(value, p, q);
        return result;
        }
```

Listing 3.3: 1D-Farbkodierung anhand der HSV-Skala und Abbildung RGB-Farbwerte.

Die Ergebniswerte können unmittelbar auf Farbwerte abgebildet werden. Dazu kann generell jede beliebige Farbskala eingesetzt werden. Üblicherweise wird ein entsprechender Wertebereich zunächst auf einen Aktionsradius zwischen 0 und 1 abgebildet. Über diesen Wert kann ein entsprechender Farbwert aus der zugrunde liegenden Skala bestimmt werden (vgl. Listing 3.3).

Zusätzlich zum Farbwert kann analog auch ein Opazitätswert angegeben werden. Anschließend können die aus den einzelnen Transferfunktionen resultierenden RGBA-Werte über eine Mittelwertbildung kombiniert werden. Die Implementierung besteht lediglich aus der Umsetzung der beschriebenen Gleichungen zur Ermittlung des resultierenden RGBA-Wertes. Die Herausforderung im Bezug auf MDTFs besteht in der Regel in der Wahl und Einstellung geeigneter Transferfunktionen, welche bislang hier noch aussteht. Bezüglich einer zielgerichteten, merkmalsorientierten Visualisierung haben sich MDTFs bereits als mächtiges Interaktionswerkzeug erwiesen. Hinsichtlich der Verdeckung als Hauptproblem der linienbasierten Strömungsvisualisierung, verspricht man sich durch der Einsatz solcher MDTFs einen entscheidenden Vorteil. Auf diese Weise könnten merkmalsbasiert nicht relevante Bereiche ausgeblendet werden, ohne von vornherein eine Reduktion der zugrunde liegenden Repräsentationsdichte vorzunehmen.

Kapitel 4

Implementierungsaspekte

4.1 Softwareumgebung

Die Implementierung wurde in eine bestehende Softwareumgebung integriert. Innerhalb dieses Frameworks werden die grundsätzlichen Aufgabenstellungen durch unabhängige Module realisiert und über entsprechende Schnittstellen eingesetzt. Dies begünstigt die Abstraktion von einer konkreten Implementierung einzelner Module.

Bei der Realisierung einer GPU-basierten Strömungsvisualisierung können unterschiedliche Verwaltungsebenen identifiziert werden, die sich auch in der Struktur der Anwendung wiederfinden. Den Ausgangspunkt für GPU-basierte Berechnungen bilden Daten- und Zugriffsstrukturen, deren Administration durch die Applikation realisiert werden muss. In unserem Fall werden entsprechende Funktionalitäten von OpenGL zur Verfügung gestellt. Das Leistungsspektrum dieser Bibliothek kann durch aktuelle Weiterentwicklungen in Form von Erweiterungen ergänzt werden. In diesem Zusammenhang übernimmt GLew eine hardwareunabhängige Verwaltung dieser Erweiterungen und vereinfacht zudem deren Handhabung.

Zur Instantiierung und Manipulation von Speicherplatz auf der Graphikhardware als Berechnungsgrundlage werden im Kontext dieser Arbeit generell FBOs eingesetzt. Als Erweiterung der Standardfunktionalität von OpenGL ermöglichen sie die Durchführung komplexer Off-Screen-Rechenprozesse. Dabei können sie als Eingabe oder Ausgabespeicher von Berechnungen fungieren. Um die komplexen Berechnungszusammenhänge konsistent zu steuern, wurden Programmklassen entwickelt, welche benötigte Funktionalitäten und Datenstrukturen der Bibliothek kapseln. Diese können innerhalb der Anwendung beliebig erstellt und angeordnet werden. Beispielsweise ergibt sich das eingesetzte Partikelsystem aus einer bestimmten Anordnung von „FBOTextures“. Diese Programmklasse kapselt die Funktionalität der beschriebenen FBOs. Innerhalb der Anwendung werden sie als Container der ermittelten Partikelpositionen interpretiert. Darüber hinaus realisiert diese Klasse einen angepassten Bindemechanismus an entsprechende Berchnungsprozesse, sowie die zum Rendern benötigte PBO-VBO-Transferfunktionalität. Eine Steuerung des Berechnungsprozesses selbst wird von übergeordneten Programmstrukturen übernommen.

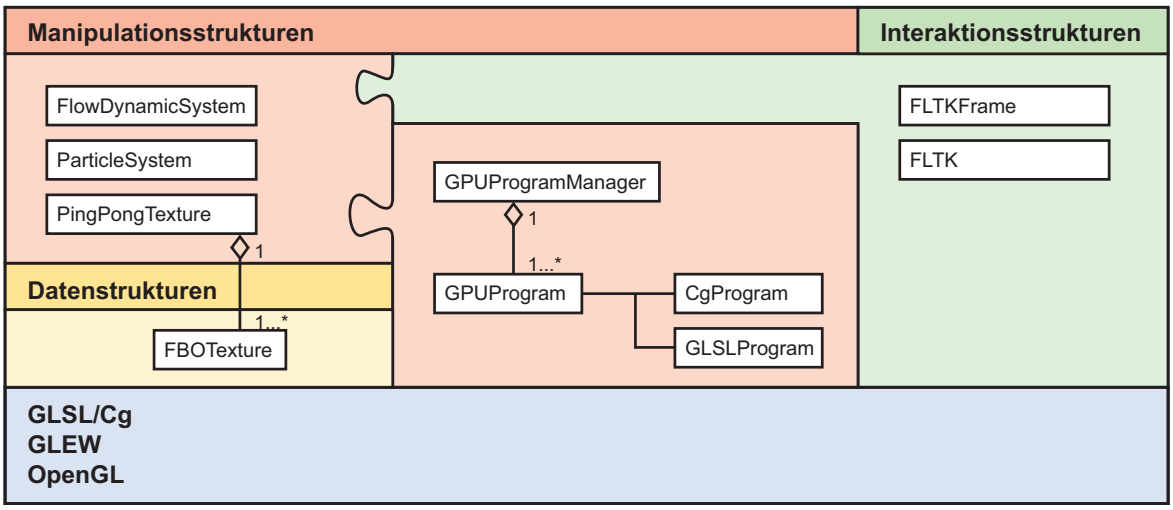


Bild 4.1: Das Framework illustriert die grundsätzlichen Abhängigkeiten der Programmkomponenten

So veranlasst das Partikelsystem die Erzeugung und Anordnung betreffender Strukturen. Um erzeugte FBO-Texturen kontrolliert als Renderquelle oder -ziel der Partikelintegration nutzen zu können, werden eine bestimmte Anzahl von FBOs in einer Ringstruktur angeordnet. Jeweils ein FBO wird als Renderziel definiert. Alle anderen FBOs, können als Eingabe des Berechnungsprozesses eingesetzt werden. Ein Zustandswechsel kann so kontrolliert entlang dieser Anordnung stattfinden. Eine entsprechende Struktur wird durch die Programmklasse „PingPongTexture“ realisiert.

Schließlich erweitert die Klasse „FlowDynamicSystem“ die Vorstellung traditioneller Partikelsysteme um strömungsspezifische Funktionalitäten. Dazu zählt beispielsweise die Akkumulation einzelner Partikelpositionen zu entsprechenden Primitiven.

Der Berechnungsprozess an sich setzt eine weitere Verwaltungsebene voraus. Hier müssen Werkzeuge zur Manipulation des Renderprozesses bereitgestellt werden. Wie oben beschrieben wird dies durch den Einsatz spezieller Shaderprogramme realisiert. Zu diesem Zweck stellt das Framework einen geeigneten Mechanismus zum Laden, Kompilieren und Binden dieser Programme zur Verfügung. Dieser Mechanismus umfasst auch die Definition benötigter Shadervariablen, sowie eine globale Verwaltung dieser Programme. Zur Interaktionskontrolle bedient man sich in diesem Kontext einer externen Bibliothek namens FLTK. Sie stellt alle Funktionalitäten einer graphischen Benutzerschnittstelle (GUI) zur Verfügung. Über entsprechende Benutzereingaben können Berechnungsprozesse über eine Shaderauswahl und zugehörige Variablen beeinflusst werden. Der Beitrag dieser Arbeit betrifft vor allem die Erweiterung eines existierenden Partikelsystems um eine strömunsspezifische Visualisierung anhand von Strömungsprimitiven. Dazu werden entsprechende Shaderprogramme zur Erzeugung der erforderlichen Partikelpositionen verwendet. Darüber hinaus werden über passende Programme auch die Visualisierung zusätzlicher Attribute realisiert. Im Folgenden werden besondere Aspekte der Implementierung eruiert.

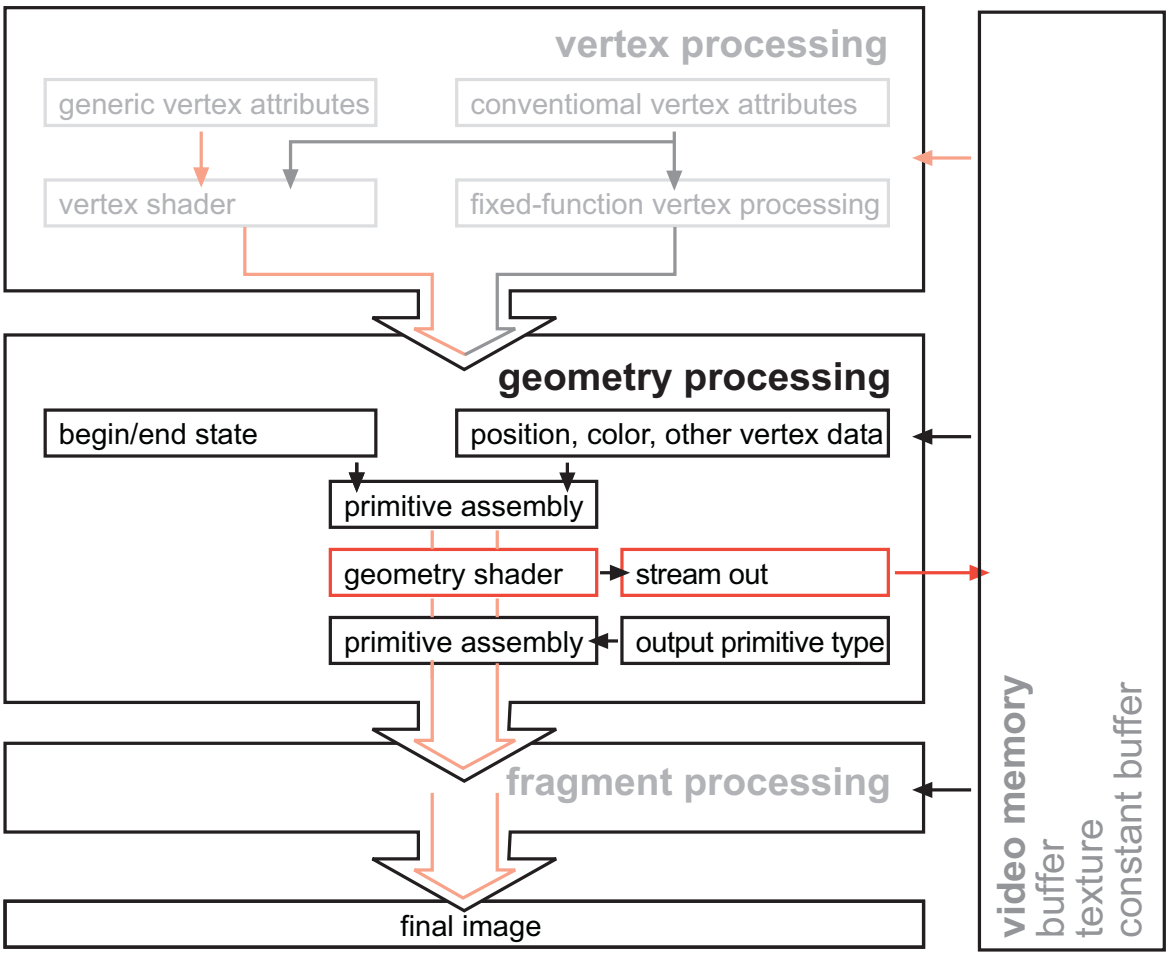


Bild 4.2: Ein Ausschnitt der Render-Pipeline veranschaulicht die Platzierung des GS sowie TF

4.2 Geometry Shader/Transform Feedback

Da es sich beim GS sowie beim TF um relativ neue Erweiterungen der traditionellen Pipeline handelt, soll die Handhabung anhand der konkreten Anwendung in unserem Kontext veranschaulicht werden. Der GS stellt dabei einen neuen Shadertyp dar, dessen Verwendung sich nur geringfügig zu bisherigen Typen unterscheidet. Entsprechend der Position in der Rendering Pipeline müssen vor der Initialisierung sowohl Eingabe- als auch Ausgabeprimitiv festgelegt werden.

```
1 glProgramParameteriEXT(gpu_program, GL_GEOMETRY_INPUT_TYPE_EXT, GL_LINES_ADJACENCY_EXT);
2 glProgramParameteriEXT(gpu_program, GL_GEOMETRY_OUTPUT_TYPE_EXT, GL_LINE_STRIP);
3 glProgramParameteriEXT(gpu_program, GL_GEOMETRY_VERTICES_OUT_EXT, vertices_per_call);
```

Listing 4.1: Notwendige Programmparameter des Geometry Shaders.

Die Angabe des Eingabeprimitivs bestimmt dabei die Anzahl verfügbarer Eingabepunkte eines Durchlaufs. Durch den Zugriff auf mehrere Punkte eines Primitivs können geometrische Abhängigkeiten ab-

geleitet werden. Beispielsweise kann die Reihenfolge der Eingabepunkte eine bestimmte Anordnung beschreiben. Die Angabe des Ausgabeprimitivs ist dagegen für die Zusammensetzung der resultierenden Punkte des GS zu Primitiven erforderlich. Man beachte, dass die Anzahl der verfügbaren Eingabepunkte durch die Angabe eines Eingabeprimitiv festgelegt wird. Die Anzahl der Ausgabepunkte hingegen ist lediglich durch eine hardwaremäßige Obergrenze limitiert oder kann explizit angegeben werden.

Der Geometry Shader wird im Kontext dieser Arbeit im Wesentlichen dazu genutzt, sukzessiv einzelne Liniensegmente zu erzeugen. Die iterative Vorgehensweise setzt die Möglichkeit voraus, Berechnungsergebnisse zwischenzuspeichern, um im weiteren Verlauf darauf zurückzugreifen. Diese Anforderung wird durch den Transform Feedback Buffer erfüllt. Dazu wird zunächst Speicherplatz im Videospeicher instantiiert, was identisch zur traditionellen Methode funktioniert. Über die Definition einer bestimmten Speicherstruktur kann dieser Speicher die Ausgabe des GS aufnehmen, ohne dazu die gesamte Pipeline zu durchlaufen. Dabei werden jeweils Attribute der erzeugten Punkte ausgegeben. Bezüglich der Streamlines werden beispielsweise zunächst über einen entsprechenden Algorithmus die Startsegmente der Streamlines in einen Buffer geschrieben und dienen so als Grundlage der Integration. Der entsprechende Programmcode wird im Anhang A.2 dargestellt. Das Resultat kann dann in Form eines VBOs wieder als Eingabe der Pipeline dienen. Da ein entsprechender Buffer nicht gleichzeitig als Ein- und Ausgabebuffer fungieren kann, wird für die Umsetzung eine Anordnung mehrerer Buffer vorausgesetzt. Da entsprechende Prozesse idealerweise im Off-Screen-Modus durchgeführt werden, besteht die Möglichkeit, durch eine Deaktivierung der Rasterisierung die Manipulation des aktuellen Framebuffers zu unterbinden.

Kapitel 5

Resultate

5.1 Anwendungsszenarien

Die dargestellten Algorithmen wurden auf einem System mit einer AMD64 X2 DualCore 4200+ 2.21 GHz Prozessorkonfiguration, 3.50 GB RAM, sowie einer NVIDIA Geforce 8800 GTX Graphikkarte implementiert und getestet. Die Graphikkarte gehört zu den ersten ihrer Generation, welche das Shader Model 4.0 und somit den Geometry Shader unterstützt. Die verwendeten Szenarien setzen sich aus Visualisierungsansätzen für Stream- und Streaklines zusammen. Die Generierung dieser Strömungslinien wird jeweils Index- beziehungsweise GS-basiert durchgeführt. Bezüglich der Berechnung sowie der Darstellung werden unterschiedliche Möglichkeiten der interaktiven Einflussnahme zur Verfügung gestellt. So kann zur Integration der Linienkontrollpunkte eine globale also auch eine adaptive Integrationsschrittweite eingesetzt werden. Darüber hinaus kann im Zuge des Renderings zwischen einfachen und erweiterten Linienprimitiven ausgewählt werden. Die folgende Untersuchung findet auf Basis analytischer sowie simulierter Datensätze statt.

5.1.1 Analytisch generierte Strömungsfelder

Die analytisch generierten Strömungsfelder basieren auf einer festgelegten Funktion, die den Strömungsvektor $\vec{v}$ on-the-fly aufgrund der aktuellen Position (x, y, z) berechnet. Ein gewähltes zirkulares Feld wird durch

$$\vec{v}\,(x, y, z) = \begin{pmatrix} -y \\ x \\ z \end{pmatrix} \tag{5.1}$$

definiert. Da zur Ermittlung der Strömungsdaten der zeitintensive Texturzugriff entfällt, laufen bezüglich der Partikelintegration die Berechnungen der optimalen Schrittweite (vgl. Kapitel 2.2.2) simultan ab.

Über die analytischen Funktionen besteht die Freiheit beliebige Vektorfeldeigenschaften zu induzieren. Man abstrahiert auf diese Weise von den komplexen Eigenschaften simulierter Vektorfelder.

Diese eignen sich aus diesem Grund insbesondere für qualitative Untersuchungen. Unter der Voraussetzung einer gewählten Spezifikation können dann weitere Abhängigkeiten abgeleitet werden. In Bild 5.4 wird der Einfluss der Integrationsschrittweitenanpassung beispielhaft veranschaulicht. Dazu wird zunächst ein zirkulares Vektorfeld generiert, mit dem die Richtungsänderung der Strömung als konstant angenommen werden kann. Dieses Strömungsfeld wird dann in Richtung einer Koordinatenachse ansteigend gewichtet ($\vec{v}_{result} = x \cdot \vec{v}\,(x, y, z)$). Bei konstanter Richtungsänderung ist der Einfluss der Schrittweitenanpassung vom Betrag der Geschwindigkeit abhängig.

5.1.2 Typhoon-Datensatz

Der in [CKL+07] erzeugte Datensatz dient im Rahmen dieser Arbeit als Beispiel für eine Anwendung des dargestellten Frameworks auf Basis simulierter Daten. Er enthält einen ausgewählten Zeitabschnitt von 6 Stunden aus der Simulation eines tropischen Typhoons. Ohne auf Details der Erzeugung einzugehen, besteht der Definitionsraum aus $106 \times 53 \times 39$ Gitterpunkten, welche jeweils an 32 diskreten Zeitpunkten abgetastet wurden. Die räumliche Ausdehnung beträgt ca. $6660,0\ km \times 3238,0\ km \times 20,0\ km$. Der Datensatz liegt in Form eines binären Dateiformates vor, welches neben den Strömungsdaten auch über die Auflösung sowie über die Anzahl der enthaltenen Zeitschritte verfügt.

Für jeden diskreten Punkt in Raum und Zeit sind jeweils drei Gleitkommazahlen gegeben, aus denen sich entsprechende Geschwindigkeitsvektoren ergeben. Der gesamte Datensatz umfasst ungefähr 80 MB. Die Datensatzgröße ergibt sich aus der Multiplikation der Anzahl der Zeitschritte, der Auflösung des Datensatzes, der Anzahl der Vektorkomponenten sowie der Datentiefe. Zur Berücksichtigung einer adaptiven Integrationsschrittweite wird jedes Datenelement in einem Vorverarbeitungsschritt um jeweils eine Vektorkomponente mit dem optimalen Zeitschritt erweitert. Der Umfang des Datensatzes vergrößert sich daraufhin um ein Drittel der Ausgangsgröße.

Zu jedem Zeitpunkt der Ausführung werden mindestens zwei Zeitschritte für die Partikelintegration benötigt. Diese diskreten Zeitschritte werden vor der Verarbeitung in Form von 3D-Texturen in den GPU-Speicher übertragen. In der vorliegenden Implementierung findet diese Übertragung einmalig statt, und betrifft den gesamten Datensatz. Um Speicherbedarf zu reduzieren, kann durch geringfügige Modifikationen auch ein dynamisches Nachladen über die Pixel-Buffer-Extension realisiert werden, was jedoch zu Inkompatibilitäten bezüglich unterschiedlicher Graphikhardware führen kann.

5.2 Effizienz-Messungen

Die Grundlage der Geschwindigkeitsmessungen bildet der zuvor beschriebene simulierte Typhoon-Datensatz. Die Vergleichskonditionen bestehen in einem einheitlich gestalteten Emitteraufbau sowie einer geeigneten Emitterplatzierung. Für die Partikelintegration wurde zudem durchgehend das klassische Runge-Kutta Verfahren 4. Ordnung als Integrationsschema eingesetzt (vgl. A.1). Der Emitteraufbau besteht aus einer 1D-Konfiguration mit 32 Startposition. Die einzelnen Verfahren werden aufgrund unterschiedlicher Linienlängen verglichen. In diesem Zusammenhang spielt die Platzierung

des Emitters eine wesentliche Rolle. Wird während der Berechnung der Linienprimitive der Rand des Definitionsbereichs des Vektorfeldes erreicht, bricht jede weitere Berechnung ab, was die Vergleichsbedingungen der einzelnen Verfahren inhomogen beeinflussen kann. Um diesen Einfluss so weit wie möglich zu minimieren, ist für vorliegende Daten experimentell eine geeignete Position gefunden worden. Für jeden anderen Datensatz wäre diese experimentelle Position erneut individuell festzulegen.

Für einen Vergleich der unterschiedlichen Szenarien wurde die Geschwindigkeit der Berechnung bezüglich eines statischen Vektorfeldes gemessen (vgl. Bild 5.2 und Bild 5.3). Durch die identische Platzierung des Emitters ergibt sich auch ein gleichförmiger Einfluss der Adaptivität in unterschiedlichen Konfigurationen.

Die im Folgenden dargestellten Messergebnisse ermöglichen Aussagen über ein differenziertes Nutzungspotential der vorgestellten Konfigurationen. Dieses ergibt sich aus dem zugrunde liegenden Berechnungsablauf der einzelnen Linienprimitive. Mehrere Faktoren beeinflussen den Verlauf der Frameraten.

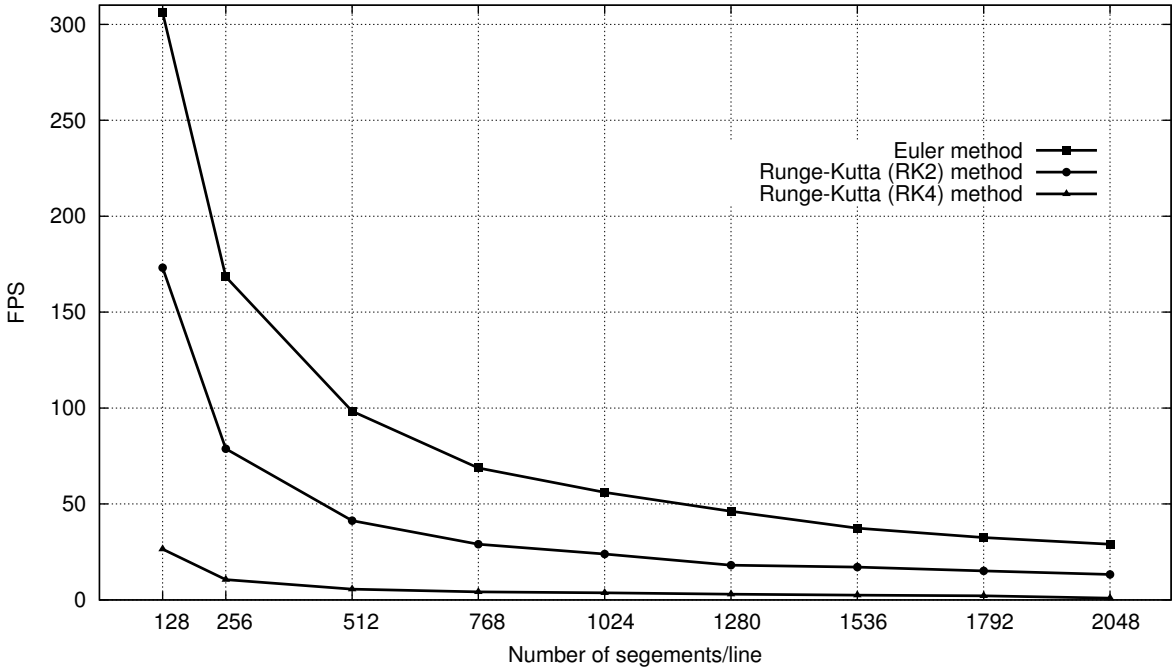


Bild 5.1: Messergebnisse. Frameratenverlauf bei unterschiedlichen Linienlängen aufgrund unterschiedlicher Integrationsschemata in diesem Fall für Index-basierte Streamlines auf Basis des simulierten Typhoon-Datensatzes.

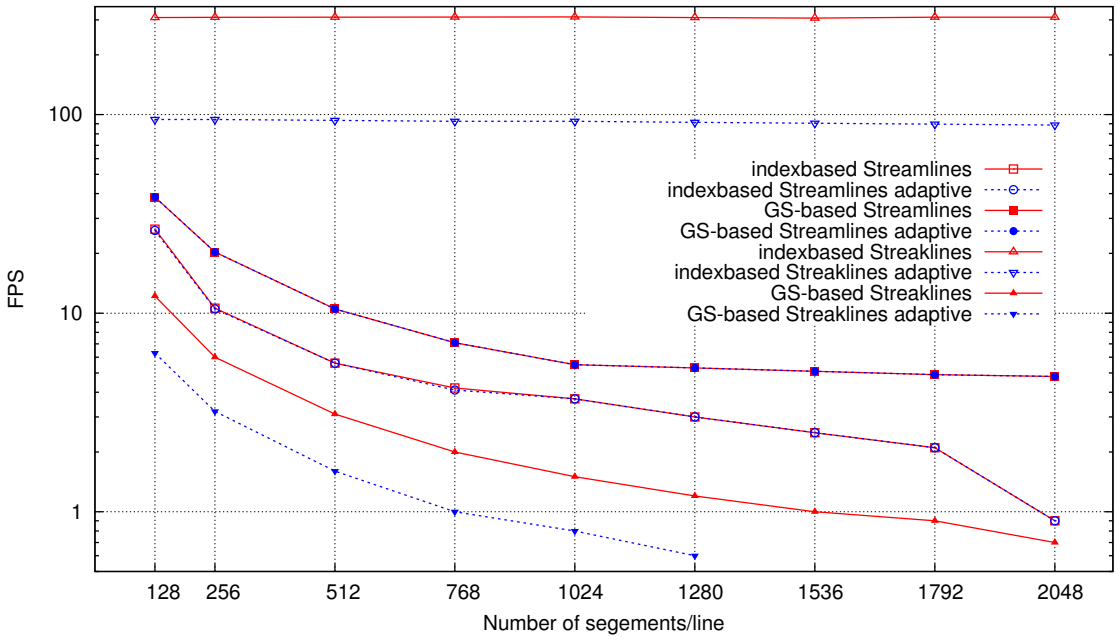


Bild 5.2: Messergebnisse. Frameratenverlauf bei unterschiedlichen Linienlängen.

5.2.1 Texturzugriffe

Zunächst kann offensichtlich ein direkter Zusammenhang zwischen der Performanz und der Anzahl durchgeführter Texturzugriffe der Renderdurchläufe hergestellt werden. Dieser Zusammenhang lässt sich anhand der Ergebnisse der Index-basierten Verfahren ohne Zeitschrittadaptivität ablesen. Der Ablauf der Berechnung ist in beiden Fällen durch eine Folge von Integrationsschritten charakterisiert. Je nach Integrationsverfahren (Euler, RK2, RK4) sind dabei unterschiedlich viele Zugriffe auf die Strömungsdaten notwendig. Bei Index-basierten Streaklines wird in jedem Zeitschritt die gesamte Partikeltextur rasterisiert. Für jede Rasterposition ergibt sich das Berechnungsergebnis aus einem Integrationsschritt. Die gleich bleibende Anzahl benötigter Berechnungen und damit der erforderlichen Texturzugriffe ergibt den konstanten Frameratenverlauf in Bild 5.2.

Im Gegensatz zur Streaklinegenerierung, die sich auf Berechnungsergebnisse unmittelbar vorangehender Zeitschritte stützt, werden alle Kontrollpunkte einer Streamline in einem einzigen Zeitschritt generiert. Diese Punkte ergeben sich durch unterschiedlich viele Integrationen aus den zugehörigen Startpositionen. Die Anzahl m der benötigten Texturzugriffe ist in diesem Fall direkt von der Linienlänge n abhängig und ergibt sich zu $m = \sum_{k=1}^{n} k \cdot i$, wobei i der Anzahl der Texturzugriffe eines Integrationsschrittes entspricht. Die Anzahl i ergibt sich aus dem jeweils eingesetzten Integrationsschema. So werden für die klassische Runge-Kutta Integration, entsprechend der 2. beziehungsweise 4. Ordnung, zwei beziehungsweise vier Texturzugriffe benötigt, wohingegen die Eulerintegration lediglich einen Zugriff voraussetzt. Bild 5.1 veranschaulicht eingehend den Einfluss der Texturzugriffe im Bezug auf die Index-basierte Streamlinegenerierung.

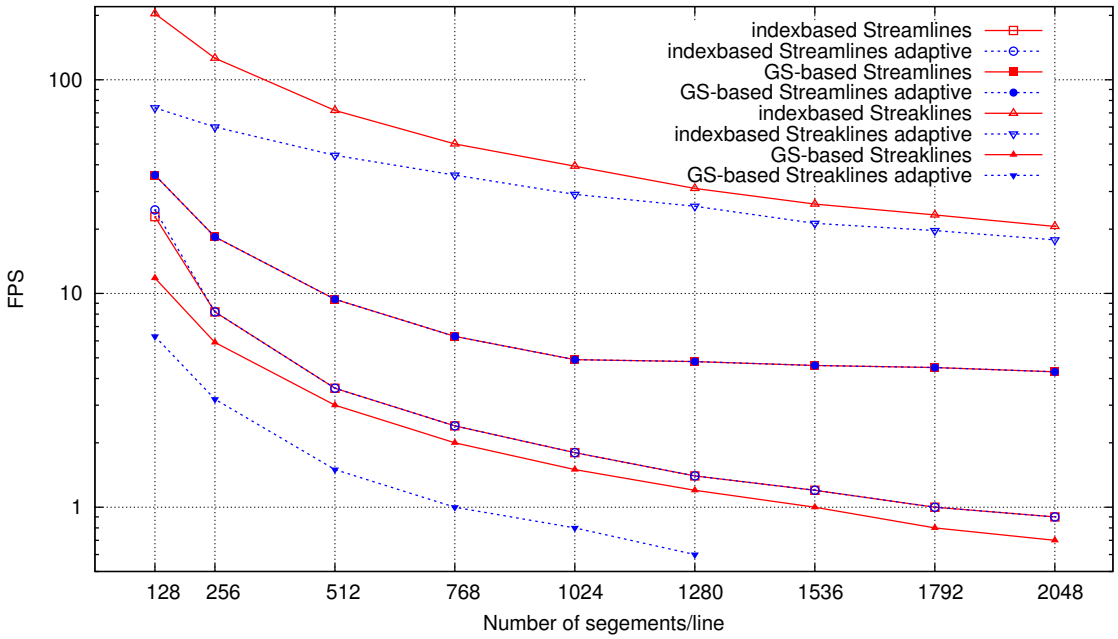


Bild 5.3: Messergebnisse. Frameratenverlauf aufgrund unterschiedlicher Linienlängen. Die Darstellung wird durch erweiterte Linienprimitive realisiert.

5.2.2 Geometry Shader/Transform Feedback

Wie in Kapitel 3.4 und 3.5 dargestellt, wird durch den Einsatz des GS die Anpassungsfähigkeit des Berechnungsprozesses deutlich gesteigert. So kann beispielsweise die Generierung der Streamlines ohne jegliche Synchronisation der Strömungslinien miteinander verwirklicht werden. Durch die Möglichkeit der dynamischen Erzeugung von Primitiven ist auch die Anzahl relevanter Partikelpositionen im Laufe der Berechnung nicht von vornherein festgelegt und kann daher aufgrund verschiedener Kriterien variieren. Dies führt zum besonderen Vorteil einer flexiblen Kontrolle der Linienlänge, was folglich eine Steigerung der Rechengeschwindigkeit erwirken kann.

Insbesondere auch bei Erzeugung der erweiterten Linienprimitive (vgl. Kapitel 3.6.1) macht sich der Vorteil des GS bemerkbar. Durch den Zugriff auf mehrere Punkte eines Eingabeprimitivs, wird erst die Verarbeitung geometrischer Abhängigkeiten ermöglicht. Im Gegensatz zu [SGS05], kann nun dadurch die Berechnung der erweiterten Linienprimitive in einem einzigen Renderpass erfolgen. Einzelne Linienprimitive bilden in diesem Fall die Eingabe der Berechnung durch den dargestellten Algorithmus (vgl. [SGS05]). Aufgrund der Eingabepunkte eines Liniensegments können alle relevanten Ausgabepunkte sowie zugehörige Punktparameter wie Normalen und Texturkoordinaten ermittelt und durch die traditionell, schnelle Rendering-Pipeline verarbeitet werden. Der Einfluss des GS auf den Geschwindigkeitsverlauf ist in Bild 5.2 und Bild 5.3 aufgeführt; übereinstimmende Linientypen sind dabei zu vergleichen. Durch die ständige Steigerung der Recheneffizienz zukünftiger Graphikhardware ist eine tangentiale Annäherung an die Echtzeitfähigkeit zu erwarten.

Da der GS derzeit einer hardwaremäßigen Beschränkung der maximalen Anzahl der Ausgabepunkte innerhalb eines Durchlaufs unterliegt, muss speziell bei Streamlines die Generierung auf mehrere Durchläufe verteilt werden. Die Speicherung der Zwischenergebnisse wird durch ein VBO in Form eines TF-Buffers realisiert. Bei GS-basierten Streaklines werden ebenfalls die Ergebnisse von Partikelberechnungen vorangegangener Zeitschritte in solchen TF-Buffern vorgehalten. Der Einsatz dieses Buffers scheint jedoch die Performanz maßgeblich zu bestimmen, was sich bei Streaklines mehr als bei Streamlines bemerkbar macht. Der Geschwindigkeitsunterschied von Index- und GS-basierten Verfahren mit adaptivem Integrationszeitschritt ist in Bild 5.2 und Bild 5.3 veranschaulicht. Obwohl in beiden Fällen identische Berechnungen durchgeführt werden, kann ein abweichender Frameratenverlauf festgestellt werden.

5.2.3 Adaptiver Integrationszeitschritt

Die Berücksichtigung eines adaptiven Integrationszeitschrittes dient der Fehlerreduktion eingesetzter Integrationsverfahren. Die Auswirkung dieser Anpassung wird durch lokale Faktoren wie Geschwindigkeit und Ausmaß der Richtungsänderung der Strömung bestimmt. Wie in Bild 5.4 dargestellt, ergeben sich daraus in bestimmten Bereichen des Strömungsfeldes kürzere Liniensegmente, welche die Stetigkeit der Kurve erhöhen. Festgestellt wird weiterhin, dass sich zwischen dem adaptiven Zeitschritt und der Kurvenverfeinerung generell kein direkter Zusammenhang zur Kurvenverfeinerung herstellen lässt. Als Nachweisführung wurde in Bild 5.5 mit einer Integrationsschrittweite von einer Sekunde im Gegensatz zur Konfiguration in Bild 5.4 mit einer Integrationsschrittweite von 0,2 Sekunden ein deutlich höherer Wert angesetzt. Es wird deutlich, dass in Bereichen geringer Geschwindigkeiten die Anpassung nicht zur erwarteten Linienverfeinerung führt. Der übereinstimmende Linienverlauf in Bereichen hoher Geschwindigkeiten weist aber gleichzeitig eine hohe Berechnungspräzision nach.

Darüber hinaus wird mitgeteilt, dass in einem zeitabhängigen Umfeld adaptiver Integrationsschrittweiten der Geschwindigkeitsverlauf der Visualisierung insbesondere bei Streaklines aufgrund unterschiedlich vieler Texturzugriffe deutlich variiert. Diese Variation ergibt sich aus zeitabhängigen Veränderungen der Vektorfeldeigenschaften, aus denen sich die oben genannte Adaptivität ableitet.

5.3 Präzision der Runge-Kutta-Integration

5.3.1 Visueller Vergleich von Strömungslinien mit und ohne Zeitschrittadaptivität

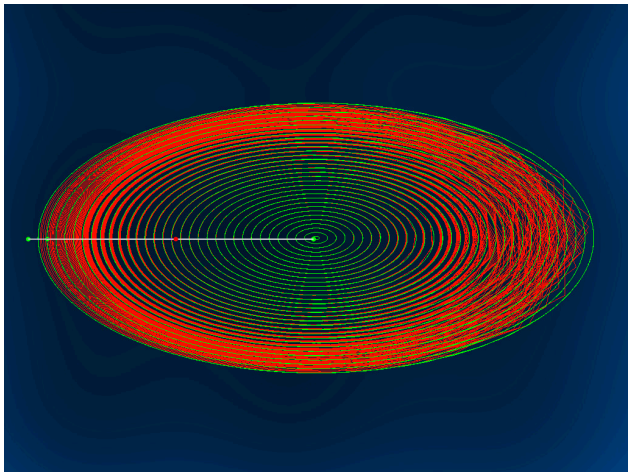


Bild 5.4: Vergleich von Streamlines bezüglich des adaptiven Zeitschrittes. Als globale Integrationsschrittweite wurde ein Zeitschritt von 0,2 Sekunden angenommen. Bezüglich eines gewichteten zirkularen Vektorfeldes (vgl. Kapitel 5.1.1) wird die Auswirkung der Adaptivität dargestellt. Die grüne beziehungsweise rote Färbung kennzeichnet das Resultat unter Verwendung einer adaptiven beziehungsweise globalen Schrittweite.

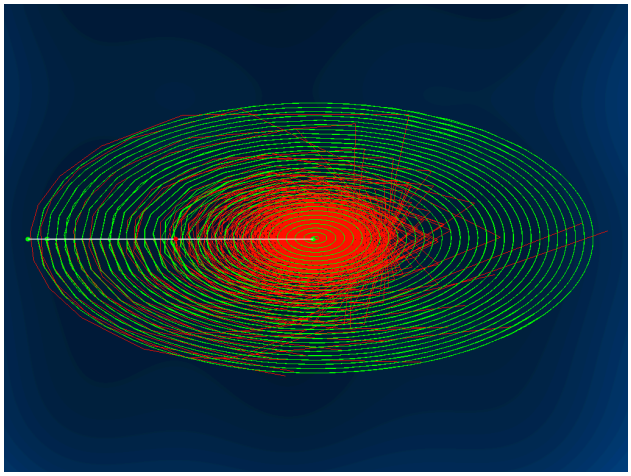


Bild 5.5: Vergleich von Streamlines bezüglich des adaptiven Zeitschrittes. Als globale Integrationsschrittweite wurde ein Zeitschritt von einer Sekunde angenommen. Bezüglich eines gewichteten zirkularen Vektorfeldes (vgl. Kapitel 5.1.1) wird die Auswirkung der Adaptivität dargestellt. Die grüne beziehungsweise rote Färbung kennzeichnet das Resultat unter Verwendung einer adaptiven beziehungsweise globalen Schrittweite.

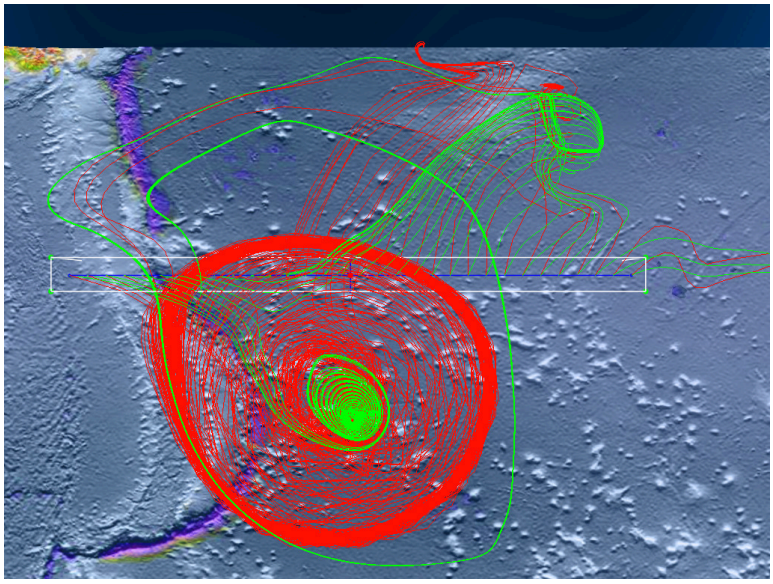


Bild 5.6: Vergleich von Streamlines bezüglich des adaptiven Zeitschrittes. In diesem Fall wurde ein globaler Zeitschritt von einer Sekunde angenommen. Das Resultat einer adaptiven Schrittweite ist durch die grüne Färbung gekennzeichnet. Die roten Linien sind das Resultat unter Verwendung einer globalen Schrittweite.

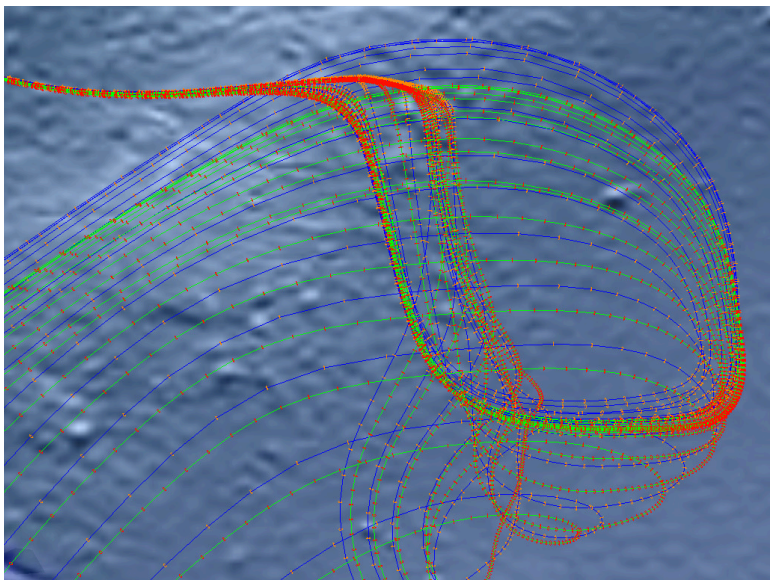


Bild 5.7: Vergleich von Streamlines bezüglich des adaptiven Zeitschrittes. In diesem Fall wurde ein globaler Zeitschritt von 0,2 Sekunden angenommen. Das Resultat einer adaptiven Schrittweite ist durch die grüne Färbung gekennzeichnet. Die blauen Linien sind das Resultat unter Verwendung einer globalen Schrittweite. Die einzelnen Liniensegmente werden durch Separatoren in der jeweiligen Komplementärfarbe gekennzeichnet.

Kapitel 6

Ausblick

6.1 Kurveninterpolation

Anhand eines adaptiven Zeitschrittes wird zunächst der Fehler der numerischen Integration minimiert [TGE97]. In unserem Fall wird dies durch einen Vergleich unterschiedlicher Näherungen des Integrationsergebnisses realisiert. Natürlich beeinflusst diese Korrektur auch die räumliche Ausdehnung eines Integrationsschrittes und damit die Stetigkeit des Kurvenverlaufs. Jedoch besteht keine Möglichkeit den Abstand aufeinander folgender Linienpunkte konkret zu steuern, was teilweise zu einem recht kantigen Kurvenverlauf führt. Um diesem Problem zu begegnen, wird eine kubische Kurveninterpolation vorgeschlagen. Aus der Segmentstruktur, wie sie in Kapitel 3.4 beschrieben wird, können zunächst die zu interpolierenden Punkte P_0 und P_1 bestimmt werden. Da diese bekanntlich auch adjazente Punkte eines Segments beinhalten, können auch die zugehörigen Tangenten $\vec{t_0}$ und $\vec{t_1}$ direkt abgeschätzt werden. Da in diesem Fall ein Strömungsfeld zugrunde liegt, können alternativ auch entsprechende Geschwindigkeitsvektoren als Tangenten verwendet werden, welche idealerweise in jedem Punkt tangential zur Strömung verlaufen. Damit liegen alle Kontrollparameter der Kurveninterpolation vor. Hierzu kann eine Hermite-Interpolation herangezogen werden:

$$H(u) = P_0 h_0(u) + \vec{t_0} h_1(u) + \vec{t_1} h_2(u) + P_1 h_3(u) \tag{6.1}$$

Die zugehörigen Basisfunktionen zur Gewichtung der einzelnen Kontrollparameter sind gegeben durch:

$$h_0(u) = 2u^3 - 3u^2 + 1 \tag{6.2}$$

$$h_1(u) = u^3 - u^2 + u \tag{6.3}$$

$$h_2(u) = u^3 - u^2 \tag{6.4}$$

$$h_3(u) = -2u^3 - 3u^2 \tag{6.5}$$

Eine entsprechende Berechnung könnte in den Fragmentshader verlagert werden, wenn es gelingt, die entsprechenden Kontrollparameter zu übertragen. Andernfalls müsste im VS beziehungsweise im GS eine feinere Unterteilung der Kurve vorgenommen werden. Die Tiefe der Unterteilung könnte dann durch ein LOD-Verfahren in Abhängigkeit zur Segmentlänge und Beobachterdistanz gesetzt werden.

Anhang A

Formeln und Codefragmente

A.1 Das klassische Runge-Kutta Verfahren 4.Ordnung

$$
\begin{aligned}
y_{i+1} &= y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(h^4) && \text{(A.1)}\\
k_1 &= f(x_i, y_i) && \text{(A.2)}\\
k_2 &= f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_1) && \text{(A.3)}\\
k_3 &= f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_2) && \text{(A.4)}\\
k_4 &= f(x_i + h, y_i + hk_3) && \text{(A.5)}\\
y_{i+1} &= y_i + hs(x_i) && \text{(A.6)}
\end{aligned}
$$

A.2 Das Phong-Modell

$$
\begin{aligned}
I_{out} &= I_a \cdot k_{ambient} + I_{in} \cdot k_{diffus} \cdot \cos\phi + I_{in} \cdot k_{specular} \cdot \cos^n\theta && \text{(A.7)}\\
&= I_a \cdot k_{ambient} + I_{in} \cdot (k_{diffus} \cdot \cos\phi + k_{specular} \cdot \cos^n\theta) && \text{(A.8)}\\
&= I_a \cdot k_{ambient} + I_{in} \cdot (k_{diffus} \cdot (\vec{L} \cdot \vec{N}) + k_{specular} \cdot (\vec{R} \cdot \vec{V})^n) && \text{(A.9)}
\end{aligned}
$$

A.3 Einsatz des Transform Feedback Buffers

```
glGenBuffers(2, transform_feedback_vbo);
glBindBuffer(GL_ARRAY_BUFFER, transform_feedback_vbo[0]);
glBufferData(GL_ARRAY_BUFFER,
                    transform_feedback_vbo_size*4*sizeof(GLfloat),
                    buffer,GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, transform_feedback_vbo[1]);
glBufferData(GL_ARRAY_BUFFER,
                    transform_feedback_vbo_size*4*sizeof(GLfloat),
                    buffer,
                    GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
GLint attribs[] = { GL_POSITION, 4, 0};
glTransformFeedbackAttribsNV(1, attribs, GL_INTERLEAVED_ATTRIBS_NV);
```

Listing A.1: Generierung des Transform Feedback Buffers.

```
GLenum feedback_primitive = GL_POINTS;
int global_points_count[2];
int points_written[2];
int current_output_buffer = 0;
for ( int i = 0; i < line_length; i++){
  glBindBufferOffsetNV(GL_TRANSFORM_FEEDBACK_BUFFER_NV,
                                    0,
                                    transform_feedback_vbo[output_buffer],
                                    global_points_count[output_buffer]*4*sizeof(GLfloat));
  glBeginTransformFeedbackNV(feedback_prim);
  glEnable(GL_RASTERIZER_DISCARD_NV);
  glBeginQuery(GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN_NV, transform_feedback_query);
  glEnableClientState( GL_VERTEX_ARRAY );
  glBindBuffer(GL_ARRAY_BUFFER, transform_feedback_vbo[1 - current_output_buffer]);
  glVertexPointer( 4, GL_FLOAT, 4*sizeof(float), 0 );
  int offset = global_points_count[1-current_output_buffer]
              -points_written[1-current_output_buffer];
  glDrawArrays(GL_LINES_ADJACENCY_EXT,
                          offset,
                          points_written[1 - current_output_buffer]);
  glDisableClientState( GL_VERTEX_ARRAY );
  glEndQuery(GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN_NV);
  GLuint primitives_written;
  glGetQueryObjectuiv(transform_feedback_query, GL_QUERY_RESULT, &primitives_written);
  global_points_count[current_output_buffer]+=primitives_written;
  points_written[current_output_buffer]=primitives_written;
  glDisable(GL_RASTERIZER_DISCARD_NV);
  glEndTransformFeedbackNV();
  current_output_buffer = 1 - current_output_buffer;
}
```

Listing A.2: Stream Out anhand des Transform Feedbacks.

Anhang B

Datensatz

B.1 Typhoon-Datensatz

Aus zeitlicher Sicht enthält der Datensatz 32 Zeitschritte im Abstand von 6 Stunden. Der Gitterpunktabstand während eines Zeitschrittes beträgt in der Horizontalen 0,5625 Grad. Am Äquator würde das in etwa 62,44 km entsprechen.

B.1.1 Höhenschichten

Schicht	***Höhe***
1	*25.0*
2	*50.0*
3	*75.0*
4	*100.0*
5	*150.0*
6	*200.0*
7	*250.0*
8	*300.0*
9	*350.0*
10	*400.0*

11	*450.0*
12	*500.0*
13	*600.0*
14	*700.0*
15	*800.0*
16	*900.0*
17	*1000.0*
18	*1100.0*
19	*1200.0*
20	*1300.0*

21	*1400.0*
22	*1500.0*
23	*1750.0*
24	*2000.0*
25	*2500.0*
26	*3000.0*
27	*3500.0*
28	*4000.0*
29	*4500.0*
30	*5000.0*

31	*6000.0*
32	*7000.0*
33	*8000.0*
34	*9000.0*
35	*10000.0*
36	*12500.0*
37	*15000.0*
38	*17500.0*
39	*20000.0*

Tabelle B.1: Höhenschichten in Meter

B.1.2 Gitterpunkte

B.1.2.1 Längengrade

Längengrade						
	120.375	131.625	142.875	154.125	165.375	176.625
	120.9375	132.1875	143.4375	154.6875	165.9375	177.1875
	121.5	132.75	144	155.25	166.5	177.75
	122.0625	133.3125	144.5625	155.8125	167.0625	178.3125
	122.625	133.875	145.125	156.375	167.625	178.875
	123.1875	134.4375	145.6875	156.9375	168.1875	179.4375
	123.75	135	146.25	157.5	168.75	
	124.3125	135.5625	146.8125	158.0625	169.3125	
	124.875	136.125	147.375	158.625	169.875	
	125.4375	136.6875	147.9375	159.1875	170.4375	
	126	137.25	148.5	159.75	171	
	126.5625	137.8125	149.0625	160.3125	171.5625	
	127.125	138.375	149.625	160.875	172.125	
	127.6875	138.9375	150.1875	161.4375	172.6875	
	128.25	139.5	150.75	162	173.25	
	128.8125	140.0625	151.3125	162.5625	173.8125	
	129.375	140.625	151.875	163.125	174.375	
	129.9375	141.1875	152.4375	163.6875	174.9375	
	130.5	141.75	153	164.25	175.5	
	131.0625	142.3125	153.5625	164.8125	176.0625	

Tabelle B.2: Längengrade östlicher Länge

Der Abstand der Gitterpunkte beträgt in West-Ostrichtung etwa 60 Grad. Am Äquator entspricht ein Grad in Ost-West-Richtung etwa 111 km. Dies ergibt sich durch die Division des Erdumfangs durch 360 Grad. Zu den Polen hin wird der Abstand immer kleiner. In Nord-Süd-Richtung entspricht ein Grad immer 111 km.

B.1.2.2 Breitengrade

Breitengrade			
	34.5397336010701	23.3073022554465	12.0748680800568
	33.9781121366611	22.7456805974447	11.5132463283931
	33.41649065912	22.1840589328381	10.9516245738869
	32.8548691688896	21.6224372618593	10.3900028166909
	32.2932476663968	21.060815584735	9.82838105695561
	31.731626152053	20.4991939016858	9.26675929482939
	31.1700046262551	19.9375722129269	8.7051375304588
	30.6083830893862	19.3759505186683	8.14351576398857
	30.0467615418161	18.8143288191146	7.58189399556176
	29.4851399839021	18.2527071144659	7.02027222531987
	28.9235184159897	17.6910854049175	6.45865045340297
	28.3618968384128	17.1294636906605	5.89702867994982
	27.8002752514945	16.5678419718816	5.33540690509797
	27.2386536555477	16.0062202487637	
	26.6770320508755	15.4445985214858	
	26.1154104377714	14.8829767902235	
	25.55378881652	14.3213550551488	
	24.9921671873977	13.7597333164304	
	24.4305455506724	13.1981115742342	
	23.8689239066043	12.636489828723	

Tabelle B.3: Breitengrade nördlicher Breite

Tabelle B.1 beschreibt die nicht-uniforme Verteilung der Höhenschichten

Verzeichnis der Bilder

1 Geometry Shader-basierte Streamlines mit adaptiver Integrationsschrittweite iii

1.1 Prinzipielle Vorstellung der unterschiedlichen Strategien 6

2.1 Generierung von GS-basierten Streaklines 8
2.2 Streamlines/Streaklines 12
2.3 Stream Balls/Stream Tubes 14
2.4 Herleitung der Beleuchtungsberechnung 15
2.5 Konzeptionelle Darstellung der Graphik-Pipeline 16
2.6 Parallelität/Pipelining 19
2.7 GPU-basierte Partikelsysteme 19
2.8 Texturkonfiguration 21
2.9 Vektorfelddaten 21
2.10 Emitterkonfigurationen 22
2.11 Quadrilineare Interpolation 23

3.1 Generierung einer Streakline 29
3.2 Generierung von Index-basierten Streaklines 30
3.3 Indexbuffer einer Streakline 31
3.4 Generierung von Index-basierten Streamlines 31
3.5 Adaptive Streamlines 33
3.6 Integration eines adaptiven Integrationszeitschrittes für Streaklines 34
3.7 Generierung von Streamlines anhand des GS 37
3.8 Listing GS-basierte Streamlines 37
3.9 Generierung von GS-basierten Streaklines mit adaptivem Integrationszeitschritt . . . 39
3.10 Generierung von Stylized Lines 41
3.11 Listing Stylized Lines 41
3.12 Transferfunktion Samples 42

4.1 Framework 46
4.2 Render Pipeline 47

5.1 Messergebnisse - Einfluss des Integrationsverfahrens 51
5.2 Messergebnisse - Frameratenverlauf bei unterschiedlichen Linienlängen 52
5.3 Messergebnisse - Frameratenverlauf bei unterschiedlichen Linienlängen (erweitert) . 53
5.4 Vergleich von Streamlines bezüglich des adaptiven Zeitschrittes analytisch 55
5.5 Vergleich von Streamlines bezüglich des adaptiven Zeitschrittes analytisch 55
5.6 Vergleich von Streamlines bezüglich des adaptiven Zeitschrittes 56
5.7 Vergleich von Streamlines bezüglich des adaptiven Zeitschrittes. 56

Verzeichnis der Tabellen

B.1 Höhenschichten in Meter 61
B.2 Längengrade östlicher Länge 62
B.3 Breitengrade nördlicher Breite 63

Listings

3.1 Berechnung des optimierten Integrationsschrittes 36
3.2 Berechnung des Gradienten 43
3.3 1D-Farbkodierung anhand der HSV-Skala. 43
4.1 Notwendige Programmparameter des Geometry Shaders 47
A.1 Generierung des Transform Feedback Buffers 60
A.2 Stream Out anhand des Transform Feedbacks 60

Literaturverzeichnis

[CKL+07] Nicolas Cuntz, Andreas Kolb, Martin Leidl, Christof Rezk-Salama, and Michael Böttinger. GPU-based dynamic flow visualization for climate research applications. In Thomas Schulze, Bernhard Preim, and Heidrun Schumann, editors, *SimVis*, pages 371–384. SCS Publishing House e.V, 2007.

[CP92] J. H. E. Cartwright and O. Piro. The dynamics of runge-kutta methods. *Int. J. of Bifurcation and Chaos*, 2(3):427–449, 1992.

[Feh70] E. Fehlberg. Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme. (German) [Classical Runge-Kutta formulas of fourth and lower order with stepsize control and their use for heat problems]. *Computing*, 6(1–2):61–71, 1970.

[FvDFH96] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics (2nd ed. in C): principles and practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[KKKW05] Jens Kruger, Peter Kipfer, Polina Kondratieva, and Rudiger Westermann. A particle system for interactive visualization of 3d flows. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):744–756, 2005.

[KLRS04] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 123–131, New York, NY, USA, 2004. ACM.

[KSW04] Peter Kipfer, Mark Segal, and Rüdiger Westermann. Uberflow: a gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, New York, NY, USA, 2004. ACM.

[Lar04] R. S. Laramee. *Interactive 3D Flow Visualization Using Textures and Geometric Primitives*. PhD thesis, Vienna University of Technology, Institute for Computer Graphics and Algorithms, Vienna, Austria, December 2004.

[OLG+07] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Kruger, Aaron E. Lefohn, and Timothy J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26:80–113(34), March 2007.

[PBL+04] Sung W. Park, Brian Budge, Lars Linsen, Bernd Hamann, and Kenneth I. Joy. Multi-dimensional transfer functions for interactive 3d flow visualization. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference on (PG'04)*, pages 177–185, Washington, DC, USA, 2004. IEEE Computer Society.

[PBSN06] Suryakant Patidar, Shiben Bhattacharjee, Jag Mohan Singh, and P. J. Narayanan. Exploiting the shader model 4.0 architecture. Technical report, Center for Visual Information Technology, IIIT Hyderabad, 2006.

[PLV+02] F. Post, R. Laramee, B. Vrolijk, H. Hauser, and H. Doleisch. Feature extraction and visualisation of flow fields, 2002.

[Ree83] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2):91–108, 1983.

[SGS05] Carsten Stoll, Stefan Gumhold, and Hans-Peter Seidel. Visualization with stylized line primitives. *vis*, 00:88, 2005.

[Söd02] Gustaf Söderlind. Automatic control and adaptive time-stepping. *Numerical Algorithms*, 31(1–4):281–310, December 2002.

[Stö95] Horst Stöcker. *Taschenbuch mathematischer Formeln und moderner Verfahren*. Verlag Harri Deutsch, 1995.

[TGE97] C. Teitzel, R. Grosso, and T. Ertl. Efficient and reliable integration methods for particle tracing in unsteady flows on discrete meshes. In W. Lefer and M. Grave, editors, *Eighth Eurographics Workshop on Visualization in Scientific Computing*, pages 49–56, 1997.

[ZSH96] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Interactive visualization of 3d-vector fields using illuminated stream lines. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 107–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.