# Datenreduktions-Techniken zur netzwerkverteilten Perzeptualisierung von Volumendaten

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades eines

Doktor-Ingenieur

Dr.-Ing.

genehmigte

Dissertation

von

**M. Sc. Dipl.-Inf. (FH) Nils Jensen**
geboren am 14. Januar 1975 in Hamburg

2007

Referentin: Prof. Dr.-Ing. Gabriele von Voigt
Korreferent: Prof. Dr. techn. Wolfgang Nejdl
Tag der Promotion: 26.11.2007

**Abstract**

Scientists use the visual and the haptic sense to discover information in complex data that are the results of numerical simulations by computers. Datasets that describe scalars in three dimensions are ”volume data”, and they are represented as volume pixels or isosurfaces. A sequence comprises an animation.
Software supports the users by way of transforming data to images, to touch, and to other perceptible forms. This process is called ”perceptualisation”, and it is specified by the perceptualisation pipeline.
A challenging problem is to perceptualise data at the speed at which they are provided by the data source when this exceeds the capacity or bandwidth of the hardware on which the pipeline is deployed.
The thesis specifies an approach to analyse the perceptualisation pipeline, specifically to identify spots of limited data throughput, and to propose data reduction techniques that improve it. These contributions are new:

- The design, implementation, and test of a coder/decoder to process volume pixels. It supports lossless and lossy compression.
- The design, setup, and conduct of a psychophysical user study to measure the sensitivity of human beings to variations of computer-synthesised viscosity, using a Phantom 1.5A haptic device.
- The transfer of the results from the user study to a parameter setting for the coder/decoder that enables sensation-preserving lossy volume data compression.
- The design, implementation, and test of a lossy coder to simplify isosurfaces while they are generated. It does not use decompression.
- The integration of the algorithms in an existing visualisation system, and its scientific use in technical mechanics and meteorology.

Keywords: visualization, haptics, performance.

**Zusammenfassung**

Wissenschaftler nutzen den visuellen und den haptischen Sinn, um Informationen in komplexen Daten zu entdecken. Die Daten sind die Resultate numerischer Simulationen durch Rechner. Datensätze, die Skalare in drei Dimensionen beschreiben, sind „Volumendaten", und sie werden als Volumenpixel oder Äquipotenzialflächen repräsentiert. Eine Abfolge bildet eine Animation.
Software unterstützt die Nutzer indem sie Daten in Bilder, in Tasteindrücke und in andere wahrnehmbare Formen umwandelt. Dieser Prozeß wird „Perzeptualisierung" genannt, und er wird durch die Perzeptualisierungs-Pipeline spezifiziert.
Ein herausforderndes Problem ist das Perzeptualisieren von Daten mit der Geschwindigkeit, mit der sie von der Datenquelle bereitgestellt werden, wenn dies die Kapazität oder Bandbreite der Hardware, auf welcher die Pipeline verteilt ist, überschreitet.
Die Arbeit beschreibt die Analyse der Perzeptualisierungs-Pipeline, insbesondere die Identifikation von Punkten begrenzten Datendurchsatzes, sowie Vorschläge für Techniken zur Datenreduktion, die den Datendurchsatz verbessern. Die Arbeit trägt Neues bei:

- Den Entwurf, die Implementierung und den Test eines Coders/Decoders, der Volumenpixel verarbeitet. Er unterstützt verlustloses und verlustbehaftetes Komprimieren.
- Den Entwurf, den Aufbau und die Durchführung einer Psychophysik-Studie, die dazu dient, die Reizempfindlichkeit von Menschen gegenüber Variationen rechnersynthetisierter Viskosität zu messen, die von einer Phantom 1.5A nachgeahmt wird.
- Die Übernahme der Ergebnisse von der Nutzerstudie in eine Parameterbelegung für den Coder/Decoder, welche die reizerhaltende, verlustbehaftete Volumendaten-Komprimierung ermöglicht.
- Den Entwurf, die Implementierung und den Test eines verlustbehafteten Coders, der Äquipotenzialflächen, noch während sie erstellt werden, vereinfacht. Dieser Ansatz benötigt keine Dekomprimierung.
- Die Integration der Algorithmen in ein existierendes Visualisierungs-System und dessen wissenschaftliche Nutzung in der technischen Mechanik und Meteorologie.

Schlagworte: Visualisierung, Haptik, Performance.

# Danksagung

Ich danke jenen, die mich während der Arbeit an der Leibniz-Universität Hannover konstruktiv begleitet haben. Mein besonderer Dank gilt meiner Doktormutter Prof. Dr.-Ing. Gabriele von Voigt für ihr Vertrauen in meine Fähigkeiten, ihre konstruktiven Urteile und die Entfaltungsmöglichkeiten, die mir halfen, meinen Weg konsequent zu beschreiten. Weiterhin geht mein Dank an Prof. Dr. techn. Wolfgang Nejdl. Seine offene Begeisterungsfähigkeit, sein Arbeitseifer und sein Vordenkerwille sind vorbildlich. Mein weiterer Dank geht an Prof. Dr.-Ing. Christian Müller-Schloer, Prof. Dr. Rainer Parchmann und Prof. Dr. Heribert Vollmer für ihre konstruktive Teilnahme am Zulassungs- bzw. Prüfungsprozeß.
Prof. Dr.-Ing. Stephan Olbrich von der Heinrich-Heine-Universität Düsseldorf sei für die langjährige personelle, organisatorische und technische Führung gedankt. Seine Doktorarbeit und die Gespräche mit ihm waren der Ausgangspunkt meiner Themenfindung. Ich habe die von ihm verfaßte Software „DSVR" weiterentwickelt. Dr. Sebastian Manten danke ich für weiterführende Verbesserungen am Polygon-Vereinfachungs-Algorithmus nach Abschluß meiner Arbeit.
Danke an die Kolleginnen und Kollegen am Forschungszentrum L3S und am Regionalen Rechenzentrum für Niedersachsen, mit denen ich zusammengearbeitet habe, sowie an Privatdozent Dr. Siegfried Raasch und Mitarbeiter vom Institut für Meteorologie und Klimatologie der Leibniz-Universität Hannover für die Erlaubnis, die mit der Software „PALM" generierten Daten in meinen Tests verwenden zu dürfen.
Die Entwicklung der in der Doktorarbeit beschriebenen Techniken wurde teilweise vom Bundesministerium für Bildung und Forschung, vom Ministerium für Wissenschaft und Kultur (Projekt VASE 3), von der Deutschen Forschungsgemeinschaft (Projekt EVITA OL 241/1-1) und von der Europäischen Union – Research Infrastructure Action unter dem Forschungsrahmenprogramm 6 „Structuring the European Research Area" (Projekt HPC-Europa RII3-CT-2003-506079, Unterprojekt 326) finanziert. Prof. Dr. Stefan Seipel von der Universität Uppsala und Dr. Luigi Fusco von der European Space Agency (ESA/ESRIN) gewährten mir Gastfreundschaft im Rahmen meiner Forschungsaufenthalte in Schweden und Italien.
Meiner Familie, insbesondere meinen Eltern, danke ich von Herzen für ihre treue Zuneigung und Förderung.

Brigitte und Thomas gewidmet. In Erinnerung an Klara und Roland.

*„Darin besteht das Wesen der Wissenschaft. Zuerst denkt man an etwas, das wahr sein könnte. Dann sieht man nach, ob es der Fall ist, und im allgemeinen ist es nicht der Fall."*
(Bertrand Russell)

# Vorwort

## Zweck

Die Doktorarbeit dokumentiert Teile meiner knapp fünf Jahre dauernden Forschung an der Fakultät für Elektrotechnik und Informatik der Leibniz-Universität Hannover.

## Leser

Die Leser[1], die ich anspreche, sind Wissenschaftler, Entwickler und Studierende, die sich auf Softwaretechnik, verteilte Systeme oder wissenschaftliche Visualisierung spezialisiert haben. Ich nehme an, dass die Leser mit der Notation und Terminologie zur Beschreibung von verteilter Software und von algorithmischer Komplexität vertraut sind. Weiterhin sind Kenntnisse in der Informationstheorie, Netzwerkprogrammierung, Computergrafik und C vorteilhaft.

## Konventionen

### Kapitelstruktur

Jedes Kapitel folgt derselben Struktur: Ich gebe eine Übersicht, diskutiere den Sachverhalt und fasse ihn, mit weiterführenden Referenzen versehen, zusammen.

### Darstellung

Der Text stellt technische Begriffe und Schlüsselworte beim ersten Gebrauch *kursiv* dar. Das Glossar spezifiziert sie, und der Index hilft dem Leser, sie zu lokalisieren.
Ich schreibe Quelltext, Algorithmen und technische Namen in `Courier` und Produktnamen in Sans Serif. Ich zeichne Zitate explizit aus (nach Harvard) und setze sie in Anführungszeichen, wenn sie wortwörtlich übernommen sind.
Pseudo-Code schreibe ich in C++-ähnlicher Notation, und Diagramme zeichne ich in der Unified Modeling Language 2.0 (Rumbaugh et al. 2004).

[1] Ich wähle, um Platz zu sparen, die männliche Form.

# Ausgewählte Publikationen

Die in der Arbeit behandelten Veröffentlichungen sind mit $*$ gekennzeichnet:


1. Stephan Olbrich, Sebastian Manten, Nils Jensen (2007) Scalable Isosurface Extraction in a Parallelized Streaming Framework for Interactive Simulation and Visualization. In: 10th International Conference on Humans and Computers. December 13–15 2007. Düsseldorf, Germany. *

2. Sebastian Manten, Nils Jensen, Stephan Olbrich (2007) Parallele Isosurface-Extraktion mit integrierter flexibler Polygonsimplifizierung. In: 4th GI VR/AR Workshop „Virtuelle und erweiterte Realität". July 15 2007. Weimar, Germany. *

3. Nils Jensen, Gabriel Gaus, Gabriele von Voigt, Stephan Olbrich (2007) Design and Psychophysical Study of Volume Compression for Haptic Rendering. In: Proceedings of the 2nd World Haptics Conference. March 22–24 2007. Tsukuba, Japan. 261–267. IEEE Press. *

4. Nils Jensen (2005) Real-Time Synchronised 3D-Perceptualisation. In Paola Alberigo, Giovanni Erbacci, Francesca Garofalo (eds): Report on Science and Supercomputing in Europe 2005. Casalecchio di Reno, Italy. 224–230. CINECA. *

5. Nils Jensen, Gabriele von Voigt, Wolfgang Nejdl, Johannes Bernarding (2005) Efficient 1-Pass Prediction for Volume Compression. In Heikki Kalviainen, Jussi Parkkinen, Arto Kaarna (eds): Proceedings of the 14th Scandinavian Conference on Image Analysis. June 19–22 2005. Joensuu, Finland. 302–311. Springer. *

6. Nils Jensen, Gabriele von Voigt, Wolfgang Nejdl, Stephan Olbrich (2004) Development of a Virtual Laboratory System for Science Education. In Jennifer Burg, Anne Boyle (eds): The Interactive Multimedia Electronic Journal of Computer-enhanced Learning. 6(2) 2004. Online: http://imej.wfu.edu/articles/2004/2/03/index.asp. Wake Forest University.

7. Denis Göhr, Christian Grimm, Nils Jensen, Stefan Piger, Jan Wiebelitz (2004) A Novel Approach to Protect Grids With Firewalls. In Marian Bubak, Michal Turala, Kazimierz Wiatr (eds): 4th Cracow Grid Workshop. December 12–15 2004. Cracow, Poland. ACK Cyfronet AGH.

8. Nils Jensen, Stefan Seipel, Gabriele von Voigt, Siegfried Raasch, Stephan Olbrich, Wolfgang Nejdl (2004) Development of a Virtual Laboratory System for Science Education and the Study of Collaborative Action. In Lorenzo Cantoni, Catherine McLoughlin (eds): Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications. June 21–26 2004. Lugano, Switzerland. 2148–2153. AACE.

9. Nils Jensen (2004) Supporting Experiential Learning in Virtual Laboratories. In Uwe Matzat et al (eds): 6th International Conference on German Online Research. March 30–31 2004. Duisburg, Germany. (Talk).

10. Nils Jensen, Ralf Einhorn, Gabriele von Voigt (2004) A Supply-Chain for Computer-Mediated Communication and Visualization. In Vaclav Skala (ed): Proceedings of the 12th World Science Conference on Computer Graphics, Visualization and Computer Vision. February 2–6 2004. Plsen, Czech Republic. 69–72 (Poster). UNION Agency Science Press.

11. Lars Winkler Pettersson, Nils Jensen, Stefan Seipel (2003) A Virtual Laboratory for Computer Graphics Education. In Steve Cunningham, Dick Martin (eds): Education Presentations – Eurographics Conference 2003. September 1–6 2003. Granada, Spain. (Poster).

12. Karsten Chmielewski, Stephan Olbrich, Nils Jensen (2003) A Distributed Scientific Visualization System. In: International Mathematica 2003 Symposium. July 7–11 2003. London, United Kingdom. (Poster).

13. Ralf Einhorn, Nils Jensen, Stephan Olbrich, Karsten Chmielewski (2003) Aufbau und Entwicklung von Multimedia-Infrastrukturen und -Services für innovative E-Learning-Anwendungen. In Jan von Knop, Wilhelm Haverkamp, Eike Jessen (eds): Security, E-Learning, E-Services – 17. DFN-Arbeitstagung über Kommunikationsnetze. June 10–13 2003. Düsseldorf, Germany. 541–567. GI.

14. Nils Jensen, Stefan Seipel, Wolfgang Nejdl, Stephan Olbrich (2003) CoVASE – Collaborative Visualization for Constructivist Learning. In Barbara Wasson, Sten Ludvigsen, Ulrich Hoppe (eds): Proceedings of the Conference on Computer Supported Collaborative Learning 2003. June 14–18 2003. Bergen, Norway. 249–253. Kluwer Academic.

15. Stephan Olbrich, Nils Jensen (2003) Lessons Learned in Designing a 3D Interface for Collaborative Inquiry in Scientific Visualization. In Constantine Stephanidis, Julie Jacko (eds): Proceedings of the International Conference on Human Computer Interaction 2003. June 22–27 2003. Hersonissos, Greece. (2)1121–1125. Lawrence-Erlbaum.

16. Nils Jensen, Stephan Olbrich, Helmut Pralle, Siegfried Raasch (2002) An Efficient System for Collaboration in Tele-Immersive Environments. In Dirk Bartz, Xavier Pueyo, Erik Reinhard (eds): Proceedings of the 4th Eurographics / ACM SIGGRAPH Workshop on Parallel Graphics and Visualization. September 9–10 2002. Blaubeuren, Germany. 123–131. 2002. ACM Press. *

# Glossar

**2-D** Zweidimensional

**3-D** Dreidimensional

**Abbilden** Das Überführen gefilterter Daten in eine Geometrie

**Aktuator** s. End-Effektor

**Arithmetische Kodierung** Algorithmus wie Huffman-Kodierung, der allerdings auch nicht-ganzzahlige Codes vergibt

**Äquipotenzialfläche** Fläche, die ein Volumen in zwei disjunkte Klassen teilt, so dass eine der Klassen genau jene Teile des Volumens enthält, an dem die enthaltenen Skalare größer als ein beliebiger aber fester Wert sind

**API** Abk. Application Programming Interface

**ASCII** Abk. American Standard Code for Information Interchange

**Barrier** Synchronisationsstelle für eine Menge von Prozessen, die nur von allen gleichzeitig überschritten wird

**Bibliothek** Zu bindende Programmteile mit einer wohldefinierten API

**Bit-Packing** Ausgewählte Bits aus mehreren Bytes zu einem Byte verdichten. Dabei gehen keine Daten verloren.

**Bitplane** Die k-ten Bits einer Menge von Bytes

**Codec** Abk. für Kodierer/Dekodierer

**CPU** Central Data Processing Unit

**Delta-Kodierung** Kodierung, welche nur den absoluten Unterschied zweier Daten quantifiziert

**Dictionary** Eine aus der Informatik bekannte Datenstruktur, in die man mittels eines Keys einen damit assoziierten Value ins Dictionary schreibt und vom Dictionary unter Angabe des Keys liest

**Digital Pulse Code Modulation** Kodierung, welche nur den relativen Unterschied zweier Daten quantifiziert

**Diskrete Cosinus-Transformation** Technik, die Datenfolgen in Cosinus-Kurven umwandelt und sie diskretisiert

**DPCM** Abk. Digital Pulse Code Modulation

**End-Effektor** Das Ende vom haptischen Gerät, das eine Kraft auf den Nutzer ausübt. Der End-Effektor hat stets eine definierte Position und einen definierten Zustand.

**Endianness** Byte-Folge eines Datums

**Entwurfsmuster** Ein sich wiederholender, bewährter, allgemeiner Lösungsansatz zum Entwerfen von Software

**Feature** Datenmuster in Roh- oder Geometriedaten

**Faktorisieren** Zusammenführen gleichartiger Funktionen oder Datenstrukturen in ein dediziertes Modul

**Filtern** Das Entfernen überflüssiger Rohdaten, das Korrigieren falscher Rohdaten und das Ergänzen von Rohdaten

**Frame** Volumen oder eine komplette Geometrie, als Bestandteil von Szenen, aus denen sich ein Stream zusammensetzt

**Frame-Nummer** Identifikator eines Frames

**Frame-Rate** Die Anzahl der Renderings pro Sekunde

**Frustum** Sicht- oder tastbarer Ausschnitt aus einem Frame

**Grobstruktursimulation** Numerische Simulation von großskaligen Turbulenzstrukturen meteorologischer Phänomene

**GPU** Graphic Processor Unit

**GUI** Abk. für Graphical User Interface, zu deutsch: grafische Nutzungsschnittstelle

**Handshake-Verfahren** Grundprinzip eines Protokolls, bei dem sich zwei Prozesse durch den Austausch von aufeinander bezogene Nachrichten synchronisieren

**Haptisieren** Verwandeln von Rohdaten in ertastbare Figuren

**Huffman-Kodierung** Kodierung von Symbolen, wobei die Länge jedes Codes mit der Häufigkeit des mit ihm assoziierten Symbols abnimmt

**Irregulär** Nicht-äquidistant verteilte Datenpunkte, die jedoch durch einen lineare Transformation in äquidistant verteilte Datenpunkte überführt werden können

**ISO** Abk. International Organisation for Standardisation

**Kodierer/Dekodierer** Ein Kodierer komprimiert (zumeist verlustbehaftet) einen über die Zeit veränderlichen Datenstrom, der multimediale Informationen beinhaltet und ein dazugehörender Dekodierer dekomprimiert den Datenstrom zur Nutzung

**LAN** Abk. Local Area Network

**Lauflängenkodierung** Kodierung der direkten Aufeinanderfolge eines Symbols durch das Symbol und dessen Anzahl

**Least Significant Bit** Bit, das den geringsten Betrag zum Datum liefert

**LZW** Abk. Lempel-Ziv-Welch-Algorithmus, der ein Datum durch die Referenz auf ein früheres Vorkommen im selben Datensatz ersetzt

**Message Passing Interface** Eine API zum standardisierten Austausch von Nachrichten zwischen Prozessoren in Distributed-Memory-Systemen

**Metapher** Eine Klasse von dem Menschen eingängigen, dargestellten geometrischen Repräsentationen eines physikalischen Zustands

**Most Significant Bit** Vom Least Significant Bit gegenteiliges Bit

**MPI** Abk. Message Passing Interface

**Multi-modal** Verschiedene Sinne ansprechend

**OpenGL** Abk. Open Graphics Library; eine API um standardisiert zu Rendern

**Parallelisierung** Aufteilung eines Problems auf mehrere Prozessoren, um den Zeitaufwand bei der Lösung des Problems zu verringern

**Partial Predictor Encoding** Kodierung eines Codes durch weniger Bits, als notwendig ist, um ihn alleine daran zu erkennen

**Performance** Eine Qualitätsmetrik, anhand der Software nach der durch sie verursachten Ausgabequalität, ihren Zeitbedarf, ihren Prozessorzeitbedarf und ihren Speicherbedarf charakterisiert wird

**Perzeptualisieren** Hier: Visualisieren und Haptisieren Allgemein: Überführen von Daten in sinnlich erfaßbare Signale, um die Daten effizient zu verstehen

**PU** Abk. Processor Unit; Prozessor

**Phantom** Ein Roboter zum Ertasten von nicht physikalisch vorhandenen Figuren

**Prädiktor** Voraussager

**Progressives Dekodieren** Dekodieren, bei dem die Zwischenergebnisse bereits dargestellt und inkrementell und iterativ verfeinert werden

**Protocol Data Unit** Eine Protocol Data Unit ist eine Dateneinheit im Netzwerkverkehr, die beim Sender entgegengenommen, mit Kontrollinformationen versehen und an die untere Netzwerk-Schicht weitergegeben wird.

**Regulär** Äquidistant verteilte Datenpunkte

**Rendering** Englisch für „Rendern", auch: Ergebnis des Renderns

**Rendern** Das Anfertigen eines Bildes oder einer ertastbaren Figur von einer Geometrie

**Resampling** Neu-Abtasten von Daten

**Schnittfläche** Planarer Ausschnitt aus einem Volumen

**Sampling** Abtasten (im Sinne von „Messen") von Daten

**SDK** Abk. Software Development Kit

**Software Development Kit** Bibliothek

**STL** Abk. Standard Template Library

**Stream** Verbund von Szenen, der eine Animation enthält und zeitabhängig stückweise transportiert und dargestellt wird

**Technik** Implementierter, integrierter und verteilter Algorithmus

**Texture Mapping** Legen eines Musters auf eine Figur

**UML** Abk. Unified Modeling Language

**Unified Modeling Language** Eine Standard-Notation, um Systeme zu beschreiben

**Update-Rate** Die Anzahl der Ankunft von Frames pro Sekunde, die gerendert werden sollen

**Verteiltes System** Auf mehrere Rechner verteilte Komponenten, die lose gekoppelt miteinander arbeiten

**Virtual Reality Modeling Language** ISO-Standarddatenformat zum Spezifizieren von 3-D-Objekten

**Volume of Fluid** Beschreibung der Durchmischung zweier Medien durch Repräsentation als diskrete Dichtewerte in einem Raum

**Volumen** Hier: Regulär oder irregulär in 3-D angeordnete Skalare

**Volumenperzeptualisierung** Verwandlung eines Volumens in Grafiken und ertastbare Figuren

**Volumenpixel** Räumliches primitives Bildelement, durch das ein Skalar im Raum repräsentiert wird

**Voxel** Abk. Volumenpixel

**VRML** Abk. Virtual Reality Modeling Language

**Wavelet-Analyse** Zerlegung von Daten in Grundfunktionen (Wavelets)

**ZNS** Zentrales Nervensystem

# Inhalt

# Bilder

# Tabellen

# Kapitel 1

# Einleitung

*„Virtual worlds, or synthetic environments, technology holds great promise for medicine, for design, for training, and for science. It is an irony sadly characteristic of our culture that these promising uses will be enabled, if at all, as byproducts of our desire to be entertained."* Frederick P. Brooks, Jr. in seinem Vorwort zu Burdea (1996)

Das Kapitel motiviert und spezifiziert den Zweck, die daraus resultierenden Ziele und die zur Erreichung notwendigen Aufgaben, sowie ihren wissenschaftlichen Innovationsgrad.

## 1.1 Motivation und Stand der Technik

Rechnergestütztes *Perzeptualisieren* ist das Überführen von Daten in sinnlich erfaßbare Signale mittels Software (Card et al. 2001). Es dient dem Veranschaulichen der Daten. Das Überführen in Bilder ist das „Visualisieren". Das Überführen in ertastbare Figuren ist das „*Haptisieren*". Im folgenden geht es beim Verwenden des Wortes „Perzeptualisieren" um diese Spezialfälle.
Der Zweck des wissenschaftlichen Perzeptualisierens, um das es in dieser Arbeit gehen soll, ist das Veranschaulichen komplexer Daten, die einen räumlichen und zeitlichen physikalischen Prozeß beschreiben. Das gestattet Nutzern, Rückschlüsse über Regelhaftigkeiten innerhalb des Prozesses zu ziehen. Die Einbettung ins allgemein bekannte Data-Mining (Hand et al. 2001) geschieht dabei folgendermaßen:

1. Messe die Prozessergebnisse
2. Wähle und korrigiere die Ergebnisse
3. Perzeptualisiere:
   (a) Bilde die Daten auf Instanzen sicht- oder tastbarer *Metaphern* ab
   (b) Stelle die Instanzen dar
   (c) Werte die Instanzen aus
4. Modelliere den Prozeß als neuen oder verfeinerten Satz von Regeln
5. Wiederhole alle Aktivitäten unter Ersetzung des Prozesses durch das Modell

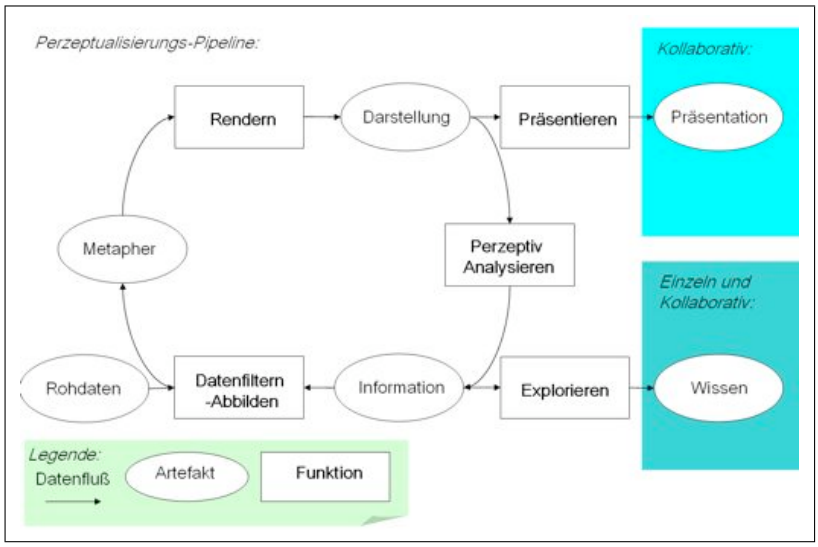


***Bild 1.1:*** *Die Perzeptualisierungs-Pipeline nach Haber und McNabb (1990)*

Bild 1.1 zeigt die Stufen des Perzeptualisierens nach Haber und McNabb (1990). Man erkennt die in Serie geschalteten Stufen vom Datum zum Bild, respektive zur ertastbaren Figur. Die Zwischenstufen unterteilen sich in das *Filtern*, *Abbilden* und *Rendern*. Kapitel 2 wird die Funktionsweise jeder Stufe erläutern.
An der Quelle liegen die Daten als simulierte, oder real vorgenommene, Messungen vor. Sie werden durch *regulär*, *irregulär* oder unstrukturiert im Raum angeordnete Tensorwerte niederdimensionaler Stufe spezifiziert, wie in Bild 1.2 gezeigt ist. Ein

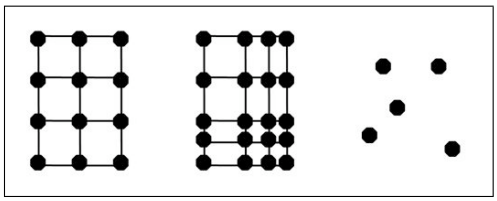

***Bild 1.2:*** *Reguläre, irreguläre und unstrukturierte Datenpunkte in* 2-D

Spezialfall sind Skalare die im dreidimensionalen Raum regulär oder irregulär angeordnet sind. Man spricht dann von *Volumenperzeptualisierung*, für deren Umsetzung man sich Metaphern zur visuellen Ausgabe bedient (Bild 1.3):

- Kolorierte *Schnittfläche*
- *Äquipotenzialfläche*
- Direktes *Volumenrendering*

Äquivalent existieren zur Ausgabe an den Tastsinn (Mark et al. 1996; Avila und Sobierajski 1996):

- Oberfläche bestimmter Rauheit und Härte
- Viskoses Medium

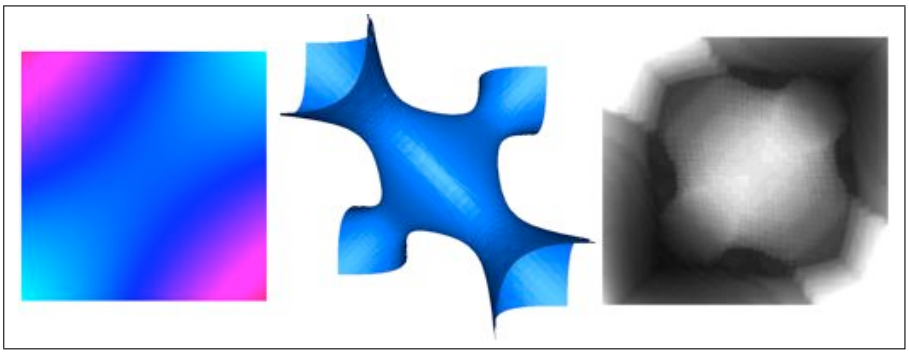

***Bild 1.3:*** *Beispiel für eine Schnittfläche, Äquipotenzialfläche und direktes Rendering*

Die Instanzen jener Metaphern sind die eingangs erwähnten Bilder und ertastbaren Figuren. Ihnen zugrunde liegt die einheitliche Repräsentation als Geometrie. Diese besteht zum Beispiel aus Gittern von *Volumenpixeln* (*Voxel*) oder aus Netzen von Polygonen.
Ein Faktor, der bei der Entwicklung und Nutzung von Perzeptualisierungs-Software berücksichtigt wird, ist deren *Performance*. Performance ist die Qualität der erzeugten Instanzen, bezüglich des Bedarfs an Zeit, Prozessorzeit und Speicher. Das Ziel ist die Steigerung der Performance. Der Nutzer bewertet die Qualität.
In einigen Arbeiten werden Techniken zur Performancesteigerung aufgezeigt: Oft wird der Bedarf an mehr Zeit in Kauf genommen, um die Qualität zu verbessern (Szirmay-Kalos et al. 2005). Eine anderslautende Zielsetzung ist die Optimierung des Zeitbedarfs unter Beibehalt der Qualität, was durch *Parallelisierung* erreicht wird (Olbrich und Pralle 2001; Woop et al. 2005). Prozessorzeit oder Speicher werden indes nicht gespart. Dieses Ziel läuft oft der Parallelisierbarkeit zuwider (Garland und Heckbert 1997; El-Rewini und Lewis 1998).
Bei der Optimierung der Perzeptualisierungs-Pipeline müssen viele Daten direkt an der Quelle entfernt werden. Die von Olbrich und Pralle (2001) spezifizierte Software DSVR verbessert den Zeitbedarf und im geringen Umfang den Speicherbedarf. Zur Performancesteigerung zerlegt sie die Daten in Gebiete und verarbeitet sie gleichzeitig. Das macht sie direkt an der Datenquelle, weil dort die meisten Daten per Zeitschritt anfallen. Orthogonal zur Datenparallelität arbeitet Funktionsparallelität. Ein Administrator verteilt dazu die Komponenten, aus denen DSVR besteht, auf parallel arbeitende Rechner.
DSVR braucht viel Prozessorzeit und Speicher, denn es wandelt lediglich die Rohdaten in binär kodierte Geometriedaten um. Weiterhin erhöht das Aufteilen im Netzwerk die Latenz des Datenaustausches. Zuletzt beherrscht das System nicht die „Graceful Degradation", denn das Bearbeiten von Datenmengen, welche die Prozessor- und Speicherkapazität überschreiten, verschlechtert die Latenz überproportional. Zum Beheben dieser Probleme gibt es Techniken, die in Tabelle 1.1 aufgeführt sind.
Die in der mittleren Spalte benannten Techniken verbessern die Latenz zwischen dem Abrufen der Daten und ihrer Darstellung bei der Ausgabe. Es werden unmittelbar die

| | Latenz | Bandbreite |
|---|---|---|
| Vermeidung | Software-Optimierung<br>Parallelisierung<br>Datenpfadoptimierung | Clipping / Culling<br>Quantisierung |
| Kompensierung | Prefetching<br>Caching<br>Speculation<br>Pipelining | Komprimierung |

***Tabelle 1.1:*** *Techniken zum Vermeiden oder Kompensieren von Latenz und Bandbreitenüberschreitung beim Perzeptualisieren von Daten*

Latenz (Vermeidung), oder die störenden Effekte, die durch die Latenz auftreten, reduziert (Kompensierung). Andere Optimierungen dienen der ökonomischen Bandbreitennutzung. Zu nennen sind Clipping, Culling, Verfahren die Quantisierung einsetzen und Datenkomprimierung. Letzteres erfordert die Daten*de*komprimierung.
Clipping / Culling entfernt sichtabgewandte Geometrien (Sutherland et al. 1974). Quantisierung reduziert Daten, die Farben und Koordinaten beschreiben. Weiterhin findet Quantisierung bei der sogenannten Polygonnetz-Vereinfachung Einsatz, zum Beispiel beim Vertex-Clustering und bei der Vertex-Decimation. Vertex-Clustering quantisiert die Koordinaten der Ecken. Vertex-Decimation entfernt einige Ecken und füllt die erzeugten Lücken.
Bei der Komprimierung unterscheiden wir die verlustlose und die verlustbehaftete Komprimierung. Nur bei der verlustlosen Komprimierung sind die dekomprimierten Daten mit den unkomprimierten Daten identisch. Ein Beispiel ist prädiktorenbasiertes Komprimieren (Salomon 2004). Ein *Prädiktor* $p$ sagt Werte $v_i$ von $m$ vorherigen Werten voraus. Sei $g$ ein Gewichtungssfaktor, dann ist die allgemeine Form

$$v_i = p_i = \sum_{j=1}^{m} g_{ij} v_j \tag{1.1}$$

Die Komprimierung iteriert die Vorhersage über alle Werte und schreibt eine ‚1', falls der vorhergesagte dem echten Wert gleicht, und sie schreibt andernfalls eine ‚0'. Der Bitstrom falsch vorhergesagter Werte wird in einer Liste gespeichert. Die Dekomprimierung liest den Bitstrom und wiederholt die Iteration der Vorhersage $p_i$ über alle Werte. Falls sie eine ‚1' vom Bitstrom liest, gibt sie den vorhergesagten Wert aus, ansonsten schiebt sie den ersten Wert aus der Liste in die Ausgabe. Dies rekonstruiert alle Daten. Man spezialisiert die Prädiktoren mit dem Ziel, deren Trefferquote zu erhöhen.
Eine Literatur-Recherche, Stand Dezember 2006, liefert den Einsatz der genannten Techniken für die Unterstützung der Volumenperzeptualisierung (Tabelle 1.2).

| | Clipping / Culling | Vereinfachung | Komprimierung |
|---|---|---|---|
| Äquipotenzialfläche | (Sutherland et al. 1974) | (Luebke 2001) | (Lindstrom 2000) |
| Direkt. Volumenrendering | (Levoy 1990) | – | (Ibarria et al. 2003) |
| Oberfläche | (Govindaraju et al. 2005) | (Otaduy und Lin 2003) | – |
| Viskoses Medium | – | – | – |

***Tabelle 1.2:*** *Stand performancesteigernder Techniken zur Volumenperzeptualisierung*

In Kapitel 2 wird deutlich werden, dass wenige Algorithmen in funktions- und datenparallel arbeitender Software eingesetzt werden können. Diese müssen zudem eine verläßliche Qualität geben, gut skalieren und garantierte worst-case-Bedarfe aufweisen. Weiterhin muß der konstante Mehraufwand jeder dieser Techniken gering sein, um die interaktive Bedienbarkeit des Gesamtsystems nicht durch ungewünschte Latenzen einzuschränken.

## 1.2 Zweck und Ziel

Der Zweck der Arbeit ist, große zeitveränderliche Datenmengen mit hoher Performance zu perzeptualisieren. Hierbei soll der Stand der Technik in Bezug auf alle zuvor genannten Entwurfsziele, d. h. Qualität, Zeit-, Prozessorzeit- und Speicherbedarf, verbessert werden.
Folgende Forschungsfragen habe ich hierzu im Rahmen der Arbeit verfolgt:

- Welche Optimierungen sind notwendig und hinreichend, um ein Perzeptualisierungs-System zum Explorieren von zeitveränderlichen Volumendaten, die mehrere MByte/sec umfassen, einsetzen zu können?
- Ab wann nehmen Nutzer die Änderung der Ausgabequalität des Perzeptualisierungs-Systems wahr?

Dies führte mich zu den folgenden Hypothesen. Dabei bedeutet „Effizienz", in Anlehnung an den Zweck der Arbeit, lineare Skalierbarkeit mit der Prozessoranzahl und $O(n)$-Komplexität mit der Eingabedatenmenge:

1. Die Techniken zur Performance-Optimierung in einem Haptisierungs-System sind die gleichen Algorithmen, wie in einem Visualisierungs-System. Deshalb können bekannte Techniken adaptiert werden.
2. Es existiert ein effizienter Coder/Decoder (*Codec*), um dynamisch generierte *Volumen*, in Form von Voxeln, so verlustlos zu komprimieren, dass die Bandbreitennutzung pro Zeitschritt reduziert wird, ohne dass der Nutzer Qualitätsverluste sieht.
3. Es existiert ein effizienter Codec, um dynamisch generierte Volumen, in Form von Voxeln, so verlustbehaftet zu komprimieren, dass die Bandbreitennutzung pro Zeitschritt reduziert wird, ohne dass der Nutzer Qualitätsverluste ertastet.
4. Diese Wahrnehmungsschwelle kann durch Nutzerstudien, auf Basis von Kenntnissen der experimentellen Psychophysik, nachgewiesen werden.
5. Es existiert ein effizienter Codec, um dynamisch generierte Volumen in Form von Äquipotenzialflächen so verlustbehaftet zu vereinfachen, dass die Bandbreitennutzung pro Zeitschritt reduziert wird, so dass der Nutzer sichtbare Qualitätsverluste bestimmen kann.
6. Die Verwendung der Codecs hat das Potenzial, neue Data-Mining-Anwendungen zu unterstützen, die mit derzeitigen Systemen nicht, oder nur unzureichend, realisiert werden.

Das Ziel ist, die Hypothesen zu verifizieren. Daraus ergeben sich die Aufgaben, die in meiner Arbeit erfüllt worden sind:

1. Entwerfe, implementiere und teste einen Komprimierer und Dekomprimierer, auf Basis einer bekannten Art der Datenkomprimierungstechnik. Dieser soll in Voxel überführte Volumendaten in $O(n)$ verlustlos verarbeiten.

2. Verbinde Quantisierung und das o. g. Verfahren zu einem verlustbehaftet arbeitenden Codec.

3. Weise den Effekt der Wahrnehmungsneutralität des o. g. Codecs durch einen in der Psychophysik anerkannten Beleg, in Form einer kontrollierten Studie mit Nutzern, nach.

4. Entwerfe, implementiere und teste einen Vereinfacher auf Basis von bekannten Methoden der Polygonnetz-Vereinfachung. Volumendaten werden in $O(n)$ zu vereinfachten Polygonnetzen umgearbeitet, die der Topographie der nichtvereinfachten Polygonnetze nahe bleiben.

5. Integriere diese Ansätze in Software, die es Nutzern erlaubt, zeitveränderliche Volumendaten zu explorieren, und lege dies durch den exemplarischen Einsatz in realen Anwendungsfeldern dar.

## 1.3 Neuheit und Wert der Arbeit

Die Beantwortung der Hypothesen ist von wissenschaftlicher, technischer und anwendungsbezogener Güte:

**Wissenschaftlicher Mehrwert** Die Arbeit stellt einen neuartigen Komprimierer und Dekomprimierer von Volumendaten vor, der das Konzept der Quantisierung mit dem Konzept der prädiktorenbasierten verlustlosen Komprimierung auf Volumendaten verbindet. Dies erlaubt es, im Gegensatz zu anderen Verfahren, Informationsverluste exakt zu definieren, weil sie ausschließlich im Quantisierer auftreten. Nachgeschaltete verlustlose Komprimierung reduziert die Daten zwar weiter, aber stellt sie beim Dekomprimieren bis auf den Quantisierungsverlust wieder her. Ähnliche Arbeiten sind von Fowler und Yagel (1994); Engelson et al. (2000); Ibarria et al. (2003); Ratanaworabhan et al. (2006); Lindstrom und Isenburg (2006) publiziert worden. Meine Arbeit schildert einen Weg, bei dem Prädiktoren, in nur einem Durchlauf, den linearen Funktionsverlauf in der 12-er-Nachbarschaft eines Voxels vorhersagen. Dies allein bringt nur marginale Erfolge. Dafür habe ich folgende Technik erdacht: Sechs der acht Prädiktoren haben denselben Code. Der Dekomprimierer „weiß" nicht beim Auftreten dieses Codes, welchen der sechs Prädiktoren der Komprimierer gewählt hat. Deshalb wird beim Auftreten des Codes durch den Dekomprimierer eine weitere Funktion evaluiert, die aus lokalen, vorher bearbeiteten Voxeln einen Hinweis ableitet, welcher der Prädiktoren der richtige ist. Diese Arbeit demonstriert, dass durch die typische Regelmäßigkeit in Volumendaten Hinweis-Funktionen existieren, die verwendet werden, um durchschnittliche Reduktionen von 1:2 zu erzielen (Jensen et al. 2005). Die Verwendung des Lorenzo-Prädiktors, als einen der acht Prädiktoren, verbessert die Reduktion auf 1:3, indem er quadratische Funktionsverläufe vorhersagt (Ibarria et al. 2003). Die Reduktion durch eine mögliche vorangegangene Quantisierung kommt in beiden Fällen hinzu. Es ist allgemein anerkannt, dass der Mensch pro Farbkanal (Rot, Grün, Blau) 256 Helligkeitsstufen unterscheidet. Das kann in acht Bit pro Kanal kodiert werden und resultiert in 24 Bit pro Pixel. Diesen Aspekt habe ich für die Haptisierung neu untersucht.

Es existiert lediglich eine Arbeit, in der sich Forscher mit der Unterscheidbarkeit von Tasteindrücken in *3-D* beschäftigt haben (Otaduy und Lin 2003). Die Unterscheidbarkeit von Viskositätsstufen in 3-D hat keiner gemessen, obwohl die Umwandlung in Viskositätsstufen pro Dichtewert eines Voxels gängig ist (Avila und Sobierajski 1996). Der genaueste möglicherweise heranzuziehende Wert ist sechs Bit für haptische Geräte mit einem Freiheitsgrad (Burdea 1996, Bild 2.15). Diese Einschränkung und die Ungenauigkeit des verwendeten Meßverfahrens haben mich bewogen, genauere Ergebnisse zu ermitteln. Hierzu stelle ich ein anerkanntes Meßverfahren für Versuche mit einem geeigneten haptischen Gerät vor. Mit einem solchen Verfahren habe ich für mehrere Nutzer in randomisierten Versuchen validiert, dass Viskositätsstufen mit einer Genauigkeit von vier Bit wahrgenommen werden. Andere Forscher können anhand meiner Beschreibungen und der verwendeten Testsoftware die gleichen Studien mit genaueren haptischen Gerät reproduzieren. Danach kann entschieden werden, ob entweder der Nutzer oder das haptische Gerät die von mir ermittelte Genauigkeit bedingen.
Statt Voxel können auch Äquipotenzialflächen aus Rohdaten gewonnen werden. Hierzu nutzt man Marching-Cubes oder Marching-Tetrahedra (Lorensen 1987; Carneiro und Kaufman 1996). Die erzeugten Polygonnetze, die die Flächen annähern, bestehen aus zu vielen Polygonen. Verlustlose Komprimierung löst das Problem nicht, weil die Dekomprimierung die überflüssigen Polygone wiederherstellt. Verlustbehaftete Komprimierung ist geeignet, solange sie sich mit der Datenerzeugung verzahnt, um den Speicheraufwand gering zu halten. Der Nachteil ist, dass dekomprimiert werden muß.
Es bieten sich Verfahren an, welche die Generierung der Polygonnetze mit deren Vereinfachung verzahnen und keine Dekomprimierung erfordern. Die neueste Arbeit ist von Treece et al. (1999). Sie verbinden Marching-Tetrahedra mit Vertex-Clustering, um vereinfachte Äquipotenzialflächen nahe bei der Datenquelle zu generieren. Die Arbeit von Treece et al. (1999) zeigt, dass man nicht das vollständige Polygonnetz erzeugen muß, sondern jeweils Teilstücke erzeugen und vereinfachen kann. Der Polygonnetz-Vereinfacher greift auf topologische Zusatzinformationen in Marching-Tetrahedra zurück, um die Abweichungen von den Originaldaten zu reduzieren. Dennoch bleibt die Arbeit von Treece et al. (1999) hinter dem Anspruch zurück, Performance, wie ich sie definiere, optimieren zu wollen.
In meiner Arbeit wird eine Technik präsentiert, deren Performance optimiert wurde. Sie verbindet Marching-Cubes und Vertex-Clustering, indem sie Ecken zu einer zusammenfaßt und sie mit den umliegenden, nicht zusammengefaßten Ecken, neu vernetzt. Hierdurch entsteht ein Verfahren, das

- die Topographie schont,
- wenig zusätzlichen Rechenaufwand erfordert,
- weniger Speicher als nachgelagertes Vereinfachen braucht,
- mehrere Vergröberungsstufen zuläßt,
- kein Dekomprimieren erfordert und, anders als bei Treece et al. (1999),
- datenparallel ausgeführt werden kann, weil es die Polygonränder zwischen separaten Unterregionen, die nach dem Vereinfachen verbunden werden, konsistent hält.

Der letzte Punkt ist bemerkenswert, weil während des Reduzierens keinerlei Nachrichtenaustausche stattfinden, was die Skalierbarkeit verbessert. Es gibt keine dazu vergleichbaren Ansätze.

**Technischer Mehrwert** Ich stelle vor, wie ich das Visualisierungs-System DSVR Version 1 (DSVR-1) um direktes Volumenrendering, um eine Haptisierungs-Pipeline, und um alle in dieser Arbeit entworfenen Techniken erweitert habe. Die neue Software heißt DSVR-2. Durch sie werden, bei gleicher Hardwarenutzung, komplexere Daten untersucht, als es mit DSVR-1 der Fall sein konnte.
Die neuen Codecs verarbeiten regulär und irregulär angeordnete Datenpunkte mit nur einer Datenzusammenführung über alle Prozessoren per Zeitschritt. Bereits vor dieser Datenzusammenführung werden die meisten Datenreduktionen angewendet, damit weitere Datentransfers, auch über Netzwerke mit geringen Bandbreiten, ungehindert bewerkstelligt werden. Der beim Filtern und Abbilden gemessene Datendurchsatz beläuft sich auf durchschnittlich 4,44 MByte/sec bei einer Reduktion um 97% der Voxel und auf 0,56 MByte/sec bei einer Reduktion um 72% der Polygone. Letztere Implementierung ist noch nicht optimiert.
Alle Implementierungen sind in Portable-C geschrieben. Sie funktionieren unter den aktuellen Versionen der Betriebssysteme Linux, SGI Irix und IBM AIX. Die Codecs werden im Einzel- und Mehrprozessormodus betrieben.

**Anwendungsbezogener Mehrwert** Der praktische Wert ist das Erproben von DSVR-2 in wissenschaftlichen Data-Mining-Anwendungen der technischen Mechanik und Meteorologie. Ich zeige die mit meinen Techniken aufbereiteten Ergebnisse einer *Volume-of-Fluid*-Simulation und einer parallelisierten *Grobstruktursimulation*. Durch die Datenreduktion kann man die zeitveränderlichen Ergebnisse sofort explorieren und präsentieren.

## 1.4 Übersicht

- In Kapitel 2 gebe ich einen Überblick über wissenschaftliche Volumenvisualisierung, spezifiziere die Grundlagen der Psychophysik, stelle die Leistungsmerkmale des haptischen Ausgabegeräts Phantom, sowie dessen *Software Development Kits* Ghost, vor und nenne die Funktionsweise der haptischen Rendering-Pipeline und ihre Anwendung. Es werden danach bekannte, effiziente Techniken zum Clipping, zum Culling, zur Quantisierung, zur Vereinfachung und zur Komprimierung besprochen. Ich schließe mit einer kurzen Einführung in paralleles und verteiltes Rechnen und der Spezifikation ausgewählter Visualisierungs- und Haptisierungs-Systeme.

- Kapitel 3 spezifiziert DSVR-1 und dessen Anwendung. Eine Analyse verdeutlicht den Bedarf an den zu entwerfenden Techniken.

- In Kapitel 4 beschreibe ich den technischen Entwurf von DSVR-2. Hierzu erweitere ich die Architektur zunächst um bekannte Techniken zum direkten Volumenrendering, sowie um die Möglichkeit zur Haptisierung. Weiterhin binde ich ein *Handshake-Verfahren* ein, um sicherzustellen, dass ausgespielte Grafiken und haptische Figuren synchron zueinander ablaufen.

- In Kapitel 5 entwerfe ich Techniken zum Reduzieren von Volumendaten. Die Algorithmen arbeiten, nach Maßgabe der zu Anfang gesetzten Entwurfsziele, effizient. Im Gegensatz zu anderen Verfahren liegt der Schwerpunkt auf symmetrisch arbeitenden Techniken, bei denen eine besondere Rekonstruktion der Daten entweder mit derselben Effizienz wie bei der Datenreduktion durchgeführt

wird oder entfällt. Ich behandle Techniken zum Clippen, Skalieren und Komprimieren von 3-D-Texturen und zum Vereinfachen von Äquipotenzialflächen. Ich schildere die Eingliederung der implementierten Techniken in DSVR-2 für den netzwerkverteilten Einzel- und Multiprozessor-Betrieb.

- In Kapitel 6 kommentiere ich das direkte Volumenrendering mit *OpenGL*-1.2-konformen Erweiterungen. Ich zeige die Nutzung des *Entwurfsmusters* `Adapter`, um damit Ghost in die Entwicklungsumgebung MS Visual Studio 2003 .NET trotz zueinander inkompatibler *`Standard Template Libraries`* (*STL*) einzubinden. Weiterhin gehe ich kurz auf den Unterschied zwischen Unix-Pipes und Linux-Pipes bei der Interprozeßkommunikation in einigen Komponenten ein.

- In Kapitel 7 bewerte ich die durch meine Techniken erzielte haptische Ausgabequalität. Kernstück ist die Spezifikation der Planung, Durchführung und Auswertung einer kontrollierten Nutzer-Studie zum Messen der Qualität der ausgegebenen ertastbaren Figuren. Das durch die Ergebnisse erreichte Ziel ist, ein allgemeingültiges psychophysisches Profil zu erstellen, das Aufschluß über die richtige Parametrisierung der Codecs geben wird.

- Kapitel 8 gibt die Ergebnisse der Lasttests des integrierten Systems wieder. Das Kapitel spezifiziert Pilot-Anwendungen in der technischen Mechanik und der Meteorologie.

- In Kapitel 9 bewerte ich das Erreichte und stelle zukünftige Arbeiten vor, die durch meine Ergebnisse möglich werden.

# Kapitel 2

# Techniken und Systeme zur Volumenperzeptualisierung

*„Du fragst, ob Du mir meine Bücher schicken sollst? Lieber, ich bitte Dich um Gottes willen, lass mir sie vom Halse!"* Johann Wolfgang von Goethe in von Goethe (2001)

Das Kapitel spezifiziert ausgewählte Grundlagen der Visualisierung und Haptisierung. Dann erörtere ich Methoden und Ergebnisse der Psychophysik im Kontext der Haptisierung. Aus einer Klasse von haptischen Geräten beschreibe ich das, was ich in der Arbeit verwendet habe. Weiterhin beschreibe ich Techniken, die Daten reduzieren, zeige Merkmale paralleler und verteilter Systeme und berichte über den Einsatz in Visualisierungs- und Haptisierungs-Software.

## 2.1 Nutzen der Visualisierung am Beispiel

Die Ursprünge der Visualisierung können ins 19. Jahrhundert zurückverfolgt werden. Zu den besten Visualisierungen gehört, neben Charles Joseph Minards zitierten Darstellungen (Minard 1861), eines der ersten Bilder über die Verbreitung von Epidemien. Der Arzt Dr. John Snow hat Bild 2.1 gezeichnet.
Das Bild zeigt das Auftreten von Cholera-Todesfällen (Punkte), in Relation zu den Standorten von öffentlichen Wasserpumpen (Kreuze). Die Zeichen sind auf einen Kartenausschnitt von London aufgetragen. Die unvollständige Legende spezifiziert den Kartenmaßstab und die Bedeutung einiger Zeichen.
Zum Entstehungszeitpunkt war das Erregerbakterium *Vibrio cholerae* unbekannt. (Tufte 1983, S. 24) beschreibt, dass Snow festgestellt hatte, dass man die meisten Cholera-Toten nahe der öffentlichen Wasserpumpe in der Broad Street gefunden hatte. Dies ist durch das Bild nachvollziehbar. Man erkennt an der Dichte der Punktwolken, dass die Anzahl der Cholera-Toten mit der Nähe zur kontaminierten Pumpe zunahm. Die Lücken nahe der Pumpe kann man dadurch erklären, dass es dort Hinterhöfe gab, die eine eigene Frischwasserzufuhr hatten. Gleichzeitig deuten die Lücken darauf hin, dass sich der Erreger nicht über die Luft verbreitete. Snow hatte daraufhin erwirkt, die Pumpe zu schließen. Das half, eine Epidemie zu beenden, die über 500 Menschen das Leben gekostet hatte.
Auch wenn die nachfolgenden Betrachtungen weniger dramatischen Ursprungs sind,

so unterstreicht das Beispiel, welchen Gewinn Nutzer durch den Einsatz guter, anwendungsbezogener Visualisierungen erhalten.


***Bild 2.1:*** *Visualisierung von Todesfällen in einer Karte von London nach Snow (1855)*

## 2.2 Wissenschaftliche Volumenvisualisierung

### 2.2.1 Überblick

Wissenschaftliches Visualisieren ist das Veranschaulichen von Daten, die einen physikalischen Zustand beschreiben. Wenn als Rohdaten Skalare vorliegen, die in $\mathbb{R}^3$ regulär oder irregulär angeordnet sind, spricht man von Volumenvisualisierung. Änderungen der Skalare über die Zeit werden diskretisiert, und aus jedem diskreten Volumen wird ein Bild abgeleitet. Die Bildfolge gibt eine Folge von Zuständen, d. h. den Prozeß, wieder.

### 2.2.2 Visualisierungs-Pipeline

Die Visualisierungs-Pipeline ist eine, auf die grafische Ausgabe spezialisierte, Form der Perzeptualisierungs-Pipeline, deren Architektur wir aus Kap. 1 kennen. Sie besteht aus seriengeschalteten Komponenten. Zu den Komponenten zählen der Datengenerator, das Filter, der Abbilder, der Renderer und die Anzeige.

**Generator** Der Generator erzeugt die Daten in numerischer Form durch Messungen oder Simulationen auf Rechnern.

**Filter** Filtern besteht aus zwei Unterstufen: Dem abbildungsunabhängigen Filtern und dem abbildungsabhängigen Filtern. Das Filter entfernt oder ergänzt die Daten, korrigiert sie und reichert sie anwendungsspezifisch an.

**Abbilder** Der Abbilder erzeugt aus den gefilterten Daten Geometriedaten, die sogenannten Visualisierungsobjekte. Hierbei wird vom Merkmalsraum in den Ausprägungsraum abgebildet. Die Abbildung ist in zwei Stufen zu teilen: Darstellungsunabhängiges und darstellungsabhängiges Abbilden. Zu letzterem zählt unter anderem das Speichern von Meta-Information. Darunter fallen Achsen, Beschriftungen, Orientierungspunkte, Segmentspezifikationen, Farbskalen und die Legende.

**Renderer und Anzeige** Dem Renderer liegt als Architektur die Grafikpipeline zugrunde. Sie besteht aus der Geometrie-, Shader- und Rastereinheit. Die Grafikpipeline verwandelt die Geometriedaten in Bildfolgen (Kaufman 1991). Die Grafikpipeline bildet dabei vom Ausprägungsraum in den Bildraum ab, und die Anzeige stellt die Ergebnisse dar.

**Auswertung und Steuerung durch den Nutzer** Der Nutzer steuert über einen Rückkanal alle Komponenten der Visualisierungspipeline.

### 2.2.3 Metaphern

Metaphern sind Klassen von Abbildungsergebnissen und dem Menschen eingängige, geometrische Repräsentationen eines physikalischen Zustands. Deren konkrete Ausprägungen weisen auf die Anzahl, Art und Relation von Datenregelmäßigkeiten hin, welche den Zustand ausmachen. Tab. 2.1 stellt ausgewählte Metaphern einander gegenüber. Die Spalten geben die Dimension des Merkmalsraums wieder und die Zeilen die Dimension des Ausprägungsraums, siehe Earnshaw und Wiseman (1992).

| | 1-D | 2-D | 3-D | n-D |
|---|---|---|---|---|
| 3-D | | | Volumenrendering<br>Festkörpermodellierung<br>Kuberillen | Ikonen<br>Attribut-Abbildung |
| 2-D | | Höhenfelder<br>Pseudofarbe<br>Konturkarten<br>Bilder | Geschachtelte Oberflächen<br>Gestapelte Texturen<br>Bänder | Attribut-Abbildung |
| 1-D | Linien<br>Kurven | Konturkarten<br>Feldvektoren<br>Raumkurven | 3-D-Vektornetze<br>Feldvektoren<br>Hedge Hogs<br>Bänder | |
| 0-D | Punkte | Punktwolken<br>Partikel<br>Punktoberflächen | Punktwolken<br>Partikelspuren<br>Punktoberflächen | |

***Tabelle 2.1:*** *Gebräuchliche Metaphern nach (Earnshaw und Wiseman 1992, Bild 3.1)*

Die mit „3-D" betitelte Spalte kennzeichnet Metaphern zur Volumenvisualisierung, darunter die in Kap. 1 genannten.

**Nutzen und Einschränkungen**

In diesem Abschnitt soll erläutert werden, warum es mehrere Metaphern zur Volumenvisualisierung gibt. Der Vorteil der Darstellung als Äquipotenzialfläche ist, dass sie die Topologie entlang aller Raumrichtungen verfolgt und darstellt. Der Nachteil von Äquipotenzialflächen liegt darin, dass sie sich gegenseitig verdecken und oft Bildraum leer lassen. Die Darstellung als halbtransparentes Objekt verbessert das Erkennen von verdeckten Details, aber es erschwert einem, den Vordergrund vom Hintergrund zu unterscheiden.
Durch den Zwang zur Wahl eines Schwellwerts blendet man stets Teile des zu untersuchenden Objekts aus. Das erschwert das Erkennen von zusammenhängenden Strukturen. Das Schachteln von Oberflächen und Freilegen relevanter Ausschnitte, sowie das Verändern des Schwellwerts, kompensieren diesen Nachteil unvollständig.
Direktes Volumenrendering beseitigt die Nachteile, bis auf die Tatsache, dass weiterhin Verdeckungen und visuelle Fehldeutungen auftreten. Fehldeutungen sind somit in beiden Metaphern nicht auszuschließen, und der Nutzer vermeidet sie, indem er die Geometrien perspektivisch vergrößert, dreht und verschiebt.
Der folgende Abschnitt beschreibt, wie man Oberflächen und Volumen erzeugt, beziehungsweise rendert.

### 2.2.4 Indirektes Rendering

Die Oberfläche besteht aus Dreiecksnetzen, die zum Beispiel von Marching-Cubes generiert werden (Lorensen 1987). Der Pseudo-Code ist

```
marchingCubes(V) {
 in volume V {
  for k = 1 to z {
   for j = 1 to y {
    for i = 1 to x {
     for c = 1 to 8 {
      if (voxelAt(cube[i,j,k].corner[c].coordinate).value
       > threshold) {
       cube[i, j, k].corner[c].in = true
      }
      else {
       cube[i, j, k].corner[c].in = false
      }
     }
     v = calculateVertices(cube[i, j, k])
     e = interpolateEdges(v, i, j, k)
     mesh = generateTriangles(e)
}}}}}
// render and display mesh
```

Kern von Marching-Cubes ist die Funktion `calculateVertices()`, in welcher der Algorithmus anhand einer Fallunterscheidungstabelle die Kombinationen der binären Zustände der acht Ecken des Kubus zur Definition der Polygone heranzieht. Spiegeln und Rotieren reduziert die 256 Kombinationen auf bis zu 15 generische Fälle (Bild 2.2).

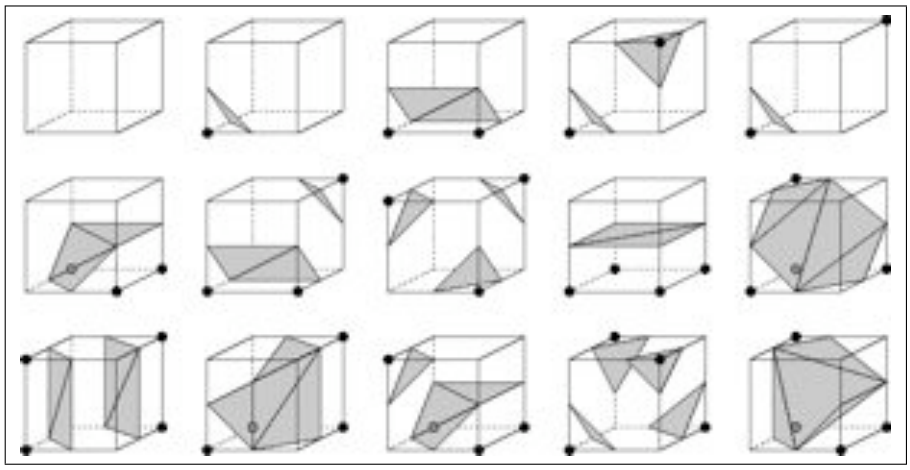

***Bild 2.2:*** *Die von Marching-Cubes unterschiedenen Fälle (Ebner 2006)*

Marching-Tetrahedra nutzt statt eines Kubus ein Tetraeda. Das vermeidet Fehler, die Marching-Cubes wegen Mehrdeutigkeiten bei der Fallunterscheidung begeht. Weiterhin sind die Längenverhältnisse der erzeugten Dreiecke besser (Treece et al. 1999). Der Aufwand an Prozessorzeit steigt um $O(1)$, weil der Algorithmus mehrere Tetraeda anstatt jeweils eines Kubus durchläuft.
Das Rendering (Foley et al. 1997) ist in den Grafikkartenprozessoren (*GPU*) kodiert. Es wird dort hardwarebeschleunigt, unter Zuhilfenahme des karteneigenen Speichers, durchgeführt. Das proprietäre MS Direct3D (Blythe 2006) und der plattformübergreifende Standard OpenGL (Schreiner et al. 1999) geben dem Programmierer Schnittstellen (*APIs*), um die Algorithmen zu bedienen. Die Performance in OpenGL ist Gegenstand von Untersuchungen durch Olbrich (2000).

## 2.2.5 Direktes Rendering

**Strahlenverfolgungsbasierte Verfahren**

Bild 2.3 zeigt als Beispiel ein hochqualitatives direktes Volumenrendering. Die Farbe und Opazität jedes Voxels korreliert mit der Dichte des an dieser Stelle gescannten Materials. Zum Bestimmen dessen Anteils an den Grundfarben und an der Opazität definiert der Nutzer eine Färbungsfunktion.
Levoy (1990) beschreibt das „Brute-Force-Raycasting" und einen davon abgeleiteten optimierten Algorithmus. Der Brute-Force-Ansatz ist für den Fall der Parallelprojektion dargestellt, um das Prinzip zu verdeutlichen.
Voxel werden mit $i, j, k = 1..N$ durch einen Vektor $\vec{i} = [i, j, k]$ indiziert. Parallele Strahlen werden vom Betrachter zum Volumen gesendet. Das zu rendernde Bild mit Breite und Höhe von jeweils $S$ besteht aus $S^2$ Pixeln. Pro Pixel wird ein Strahl gesendet. Jedes Pixel, somit jeder Strahl, wird durch einen Vektor $\vec{u} = [x, y]$, $x, y = 1..S$ indiziert. Entlang jedes Strahls tastet man das Volumen in äquidistanten Abständen ab. Eine Probe wird durch $\vec{U} = [u, v, w]$ indiziert. $[u, v]$ kennzeichnet den Sehstrahl. $w$ kennzeichnet den Abstand $1..W$ zum Betrachter entlang des Sehstrahls. Die Farbe und Opazität an einer Probe sind $C(\vec{U})$ und $\alpha(\vec{U})$. Hinter dem Volumen wird ein un-

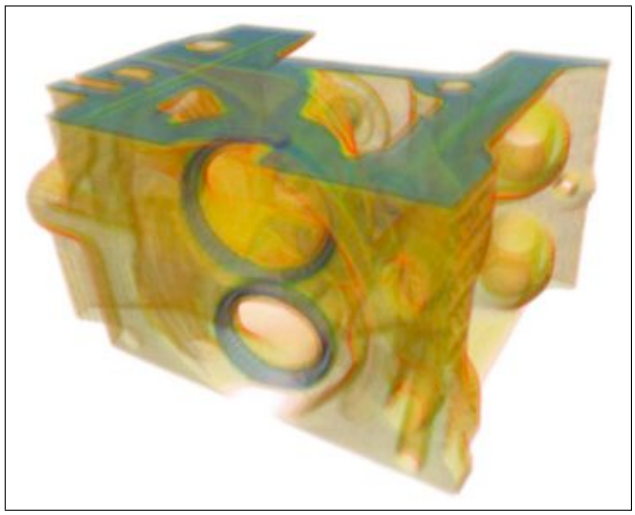

***Bild 2.3:*** *Gerenderte Daten, welche die Geometrie eines Motorblocks spezifizieren (Fraunhofer Institut für graphische Datenverarbeitung 2006)*

durchsichtiger Grund definiert. Alle Farben und Opazitäten entlang $\vec{U}$ werden zu einer Pixelfarbe $C(\vec{u})$ verknüpft.
Seien weiter $C'_{in}(\vec{u};\vec{U}) = C_{in}(\vec{u},\vec{U})\alpha_{in}(\vec{u},\vec{U})$, $C'_{out}(\vec{u};\vec{U}) = C_{out}(\vec{u};\vec{U})\alpha_{out}(\vec{u};\vec{U})$ und $C'(\vec{U}) = C(\vec{U})\alpha(\vec{U})$. Damit ist die Relation zwischen der Farbe und Opazität beim Eintritt in eine Probe und beim Austritt

$$C'_{out}(\vec{u};\vec{U}) = C'_{in}(\vec{u};\vec{U}) + C'(\vec{U})(1 - \alpha_{in}(\vec{u};\vec{U})) \tag{2.1}$$

und

$$\alpha_{out}(\vec{u};\vec{U}) = \alpha_{in}(\vec{u};\vec{U}) + \alpha(\vec{U})(1 - \alpha_{in}(\vec{u};\vec{U})) \tag{2.2}$$

Nach dem Abtasten aller Werte entlang eines Sehstrahls erhält man mit $\vec{W} = [u, v, W]$ das Ergebnis $C(\vec{u}) = C'_{out}(\vec{u};\vec{W})/\alpha_{out}(\vec{u};\vec{W})$.
Zusammenfassend lassen sich Strahlenverfolgung, Abtastung und Verknüpfung algorithmisch darstellen. Es werden dazu Hilfsfunktionen benötigt. `First` und `last` berechnen aus dem Index für einen Sehstrahl die Objektkoordinaten, wo der Sehstrahl in die Daten ein- und aus ihnen austritt. Die Koordinaten haben die Form $\vec{x} = [x, y, z]$, wobei $1 \leq x, y, z \leq N$ gilt. `Object` und `image` konvertieren zwischen den Objekt- und den Bildkoordinaten. `Sample` nimmt ein Array von Farben oder Opazitäten, sowie die Objektkoordinaten eines Punkts. `Sample` gibt die Farbe oder Opazität, indem es trilinear zwischen den acht nächsten Voxelwerten interpoliert. Mit den Hilfsfunktionen wird der Algorithmus spezifiziert:

```
traceRay(U) {
 C'(u)     = 0
 alpha(u) = 0
 x_1 = first(u)
 x_2 = last(u)
 U_1 = ceiling(image(x_1))
 U_2 = floor(image(x_2))
```

```
 for U = U_1 to U_2 {
  x = Object(U)
  alpha(U) = sample(alpha, x)
  if alpha(U) > 0 {
   C'(U) = sample(C', x)
   C'(u) = C'(u) + C'(U)(1 - alpha(u))
   alpha(u) = alpha(u) + alpha(U)(1 - alpha(u))
}}}
```

Anhand der Lichtundurchlässigkeit bestimmt man, ob Strahlen vorzeitig terminiert werden. Das optimiert den Algorithmus, wie Levoy (1990) erläutert.

### Texturbasierte Verfahren

Zum Rendern werden zur Bildebene in Abständen $w$ parallele Schnittflächen $z_w$ erzeugt. Für jeden Schnittpunkt $[x, y, z]$ zwischen den Flächen und dem Volumen ermittelt man den Wert $f(x, y, z)$. Die Funktion $[u, v] = \mathit{image}(w, x, y, z)$ übersetzt die Objekt- in die Bildkoordinaten der Schnittfläche $z_w$. Aus dem Voxelwert wird über die Transferfunktionen $C(x, y, z)$ und $\alpha(x, y, z)$ der Farbwert erzeugt und auf die Schnittfläche an der Stelle $[u, v]$ als Wert eines Texturpixels (Texel) gesetzt, was unter dem Namen *Texture Mapping* bekannt ist (Heckbert 1986). Das Rendern aller Schnittflächen von hinten nach vorne gibt das direkte Volumenrendering.
Ein Punkt schneidet im Regelfall mehrere Voxelwerte. Wie beim direkten Volumenrendering werden deshalb entweder die Voxelwerte (Nachklassifikation), oder die Farbinformationen der Voxelwerte (Vorklassifikation) interpoliert. Nachklassifikation gibt, außer in Spezialfällen, die höherwertigeren Resultate (Engel et al. 2001). Nachteilig ist jedoch die hohe Anzahl an Schnittflächen, die gebraucht wird, um keine Artefakte durch das *Sampling* entstehen zu lassen.
Engel et al. (2001) beschreibt eine Lösung in Form einer Erweiterung zur Vor-Integration von Voxel- oder Farbwerten entlang jedes Sehstrahls zwischen zwei Flächen $z_w$ und $z_{w+\Delta w}$. Die Erweiterung nutzt die Möglichkeit zur verketteten Textur-Indirektion auf Grafikkarten mit Texture- oder Fragment-Shadern, um die Sehstrahlen durch winkelabhängige Texel nachzuahmen. Engel et al. (2001) erreichen dadurch mit weniger Schnittflächen die Ausgabequalität der Nachklassifikation.
Das Verfahren von Engel et al. (2001) ist ein Hybrid zwischen Strahlenverfolgung und Texture Mapping, weil es im Volumen statt zweidimensionalen Schnitten dreidimensionale Schnitte anlegt. Dies ist möglich, weil strahlenverfolgungsbasierte und texturbasierte Verfahren dasselbe mathematische Prinzip haben, wie Cabral et al. (1994) erstmals angemerkt haben. Dort wird direktes Volumenrendering als die Anwendung der Radon-Transformation abstrahiert (Radon 1917). Sei exemplarisch $\mathbb{R}^2$ gewählt, dann definiert man die Radon-Transformation als eine mathematische Abbildung vom physikalischen Raum eines Körpers $[x, y]$ in den Projektionsraum $[s, \theta]$. Die Funktion $f(x, y)$ definiert die Voxelwerte, generell gesprochen die Körperdichte, entlang einer Ebene in $z$. Bild 2.4 verdeutlicht den Zusammenhang. Die Radon-Transformation ist ein Operator auf der Funktion $f$, der die Funktion $h$ ausgibt:

$$h(s, \theta) = \int_{-\infty}^{\infty} f(x(l), y(l))dl \tag{2.3}$$

Die Funktion $h(s, \theta)$ ist das Linienintegral $[x(l), y(l)]$ das im Winkel $\theta$ über $f(x(l), y(l))$

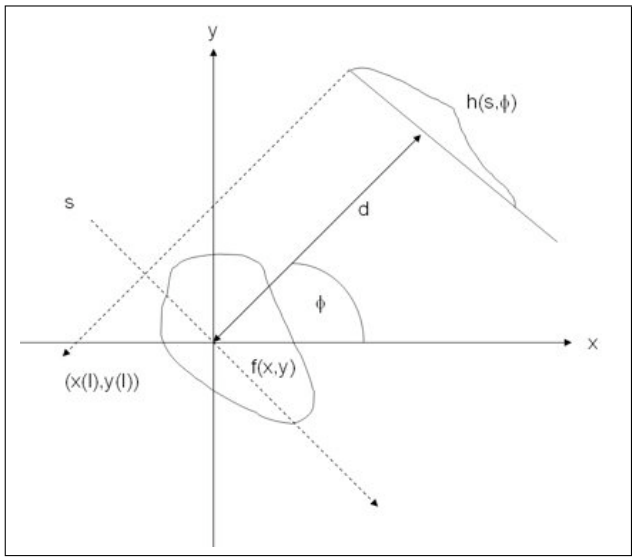


***Bild 2.4:*** *Parallele Strahlengeometrie nach Cabral et al. (1994)*

projiziert wird.

$$\begin{aligned} x(l) &= d\cos\theta + s\sin\theta + l\cos\theta \\ y(l) &= -d\sin\theta + s\cos\theta + l\sin\theta \end{aligned}$$

Das Überführen von $f$ in $h$ ist das direkte Volumenrendering. Die diskrete Form lautet

$$h(s,\theta) \approx \Delta l \sum_{k=1}^{N} f(x(l_k), y(l_k)) \tag{2.4}$$

Die Vorbedingung für die inverse Radon-Transformation ist die eulersche Identität:

$$e^{i\phi} = \cos(\phi) + i\sin(\phi) \tag{2.5}$$

Sei $H$ die Fourier-Transformation der Projektion $h$. Dann ist $\tilde{h}$ die gefilterte Version der Projektion $h$:

$$\begin{aligned} \tilde{h}(s,\theta) &= \int_{-\infty}^{\infty} H(\omega,\theta) e^{i2\pi\omega s} |\omega| d\omega & (2.6) \\ &= \mathcal{F}^{-1}(|\omega|) * h(s,\theta) & (2.7) \end{aligned}$$

Dann ist die inverse Radon-Transformation

$$f(x,y) = \int_0^{\pi} \tilde{h}(s,\theta) d\theta \tag{2.8}$$

Dies nutzt die Eigenschaft, dass $h$ gleich der inversen Fourier-Transformation eines kreisförmigen Querschnitts der Fourier-Transformation $\mathcal{F}$ von $f$ im Winkel $\omega$ ist. Weitere Ausführungen sind Cabral et al. (1994) zu entnehmen.

## 2.3 Haptik

Menschen nutzen den Tastsinn, kurz Haptik, um die Welt zu erforschen. Zunächst führe ich in die Psychophysik ein. Dann erkläre ich, wie das Tasten biologisch-mechanisch beim Menschen funktioniert. Zuletzt spezifiziere ich in diesem Abschnitt Geräte und Systeme zur Haptisierung.

### 2.3.1 Psychophysik

Vermehrt betrachten Wissenschaftler die Vermittlung haptischer Reize als Ergänzung zur Visualisierung, um einem Nutzer Informationen buchstäblich „begreiflich" zu machen. Grundlagen der optischen Psychophysik sind Lehrbüchern zu entnehmen, zum Beispiel Ware (2000). In diesem Abschnitt soll vielmehr der Fokus auf eine Klärung der Grundlagen jener Teile der Psychophysik gelegt werden, die für die Beurteilung der Darstellungsqualität haptischer Darstellungsformen von elementarer Bedeutung sind, und die somit einen Schlüssel zum Erreichen der in dieser Arbeit gesetzten Arbeitsziele darstellen. Jedoch überschneidet sich das Grundlagenwissen der haptischen Psychophysik mit dem der optischen Psychophysik, weil die Mechanismen der Sinneswahrnehmung auf wenige Regeln zurückgeführt werden.
In Deutschland haben Weber und Fechner Anfang des 19. Jahrhunderts die menschliche Sinneswahrnehmung untersucht, um einheitliche Grundregeln zu erkennen, nach denen Menschen die Intensität und Unterschiedlichkeit von Sinneseindrücken ausgehend von Sinnesreizen beurteilen (Fechner 1860). Es wurde also der innere Sinneseindruck in Abhängigkeit des äußeren Sinnesreizes, unter weitestmöglicher Ausblendung höherer kognitiver Prozesse, gemessen. Dies ist die *psychophysische Funktion*. Zwei Parameter sind dabei die stets quantitativ zu bestimmenden Kenngrößen. Der erste Parameter ist die absolute Wahrnehmungsschwelle, auch Nullpunkt der Funktion genannt, abgekürzt mit RL für Reiz-Limes. Der Reiz-Limes spezifiziert, ab welcher Intensität $I$ ein Reiz einen unmittelbaren Eindruck hervorruft. Der zweite Parameter ist die Unterschiedsschwelle $\Delta I$, abgekürzt mit DL für Differenz-Limes. Der Differenz-Limes spezifiziert, wie groß der Unterschied zwischen zwei Intensitäten eines Reizes mindestens sein muß, um die durch sie unmittelbar hervorgerufenen Eindrücke anhand der Intensität zu unterscheiden.
Weber (1834) formuliert im Weberschen Gesetz, dass der Differenz-Limes linear von der Intensität eines Reizes abhängt ($c$ ist konstant):

$$\frac{\Delta I}{I} = c \tag{2.9}$$

Daraus folgt, dass Sinneseindrücke bei sehr niedrigen oder sehr hohen Reizintensitäten kaum unterschieden werden. Fechner (1860) postuliert, unter der Annahme, dass die Differenz-Limen perzeptuell gleich sind, dass mit $P$ als wahrgenommener Intensität des Eindrucks eine logarithmische Relation gilt:

$$P = c \cdot \log I \tag{2.10}$$

Die Werte der Kenngrößen RL und DL für mehrere Arten des Bewegungs- und Tastsinns lernen wir später am Menschen kennen.

### 2.3.2 Methoden zum Bestimmen der psychophysischen Funktion

Die psychophysische Funktion spezifiziert den RL und den DL. Psychophysische, experimentelle Studien dienen dazu, die Funktion zu ermitteln.

Reizempfinden ist weder erlernbar noch bedeutend individuell verschieden (Tan und Pizlo 2003). Somit genügen für psychophysische Studien zum Bestimmen der psychophysischen Funktion, im Gegensatz zu anderen empirischen Studien, ein bis drei Teilnehmer.
Die Studien folgen einem generellen Studienaufbau. Folgende Methoden zum Aufbau sind möglich (Tan und Pizlo 2003):

- Method of adjustment
- Method of limits
- Method of constant stimuli
- Adaptive methods:
  - Up-down methods
  - Transformed up-down methods

Bei der Method of adjustment mißt der Experimentator RL, indem der Teilnehmer (Proband) die Stärke eines Reizes so einstellen muß, dass er den Reiz gerade eben spürt. DL wird gemessen, indem der Proband die Stärke eines Reizes der eines Referenzreizes anpassen muß, so dass beide als gleich wahrgenommen werden. Die Standardabweichung ist der DL und der Durchschnitt ist der Punkt subjektiver Gleichheit (Point of subjective equality, PSE).
Bei den nachfolgenden Methoden, darunter der Method of limits, ergibt eine Folge von Versuchen das Experiment. Die Method of limits beginnt damit, einen Reiz zu präsentieren, der unterhalb des PSE ist. Jeder nachfolgende Reiz wird um einen konstanten Wert stärker ausgeführt als der direkte Vorgänger (ascending series). Der Proband sagt, ab wann der Reiz den Referenzreiz übersteigt. Der PSE ist der Mittelpunkt zwischen zwei Reizesstärken während der letzten negativen und der ersten positiven Antwort vom Probanden. Ähnlich konstruiert der Experimentator eine Folge, bei der jeder Reiz schwächer als der vorhergehende ist (descending series). Mehrere ascending und descending series müssen randomisiert durchgeführt werden. PSE ist der Durchschnitt des Übergangspunktes der geraden Anzahl von ascending und descending series.
Bei der Method of constant stimuli präsentiert der Experimentator dem Probanden einige wenige ausgewählte Reizintensitäten, die sich um ein festes Vielfaches unterscheiden. Jede Stufe wird über hundertmal präsentiert. Welche Stufe jeweils als nächstes benutzt wird, soll der Probanden nicht vorhersehen, um das Ergebnis nicht zu verfälschen. Der Proband muß sagen, wann immer der Reiz stärker ist als ein Referenzreiz. Um DL zu messen, ist der Referenzreiz spürbar. Für RL ist er das nicht. Die Method of constant stimuli ist verlässlicher und genauer als die Method of adjustment oder die Method of limits.
Die Treppen-Methode (simple up-down-method) ist mit der Method of limits identisch, außer, dass eine Folge von Versuchen nicht nach dem ersten Ändern der Antwort endet. Stattdessen wiederholt man die Versuche, um die gesuchten Werte genauer zu ermitteln. Jedoch beeinflußt die Erwartungshaltung des Teilnehmers seine Antwort und konfundiert so das Ergebnis.
Der Experimentator braucht jeweils sehr lange, um eine der Methode durchzuführen. Außerdem sind die Messungen ungenau, falls der Experimentator falsche Abstände zwischen den Reizintensitäten vorsieht. Adaptive Methoden verbinden den Vorteil der Nicht-Vorhersagbarkeit durch den Probanden mit der verringerten Dauer zur Erlangung genauer Ergebnisse. Die Dauer wird verringert, indem der Experimentator die Anzahl

von Versuchen herabsetzt, in denen Probanden die Reizintensitäten überwiegend korrekt wahrnehmen.

### 2.3.3 Aufbau und Funktion des Tastsinns

Nach Burdea (1996) ist Propriozeption das Wahrnehmen der eigenen Körperposition und Bewegung. Diese wird unter anderem durch das Wahrnehmen der Kräfte in den Muskeln und Sehnen ermöglicht. Die freien Nervenenden an den Knochengelenken geben die Körperposition an das zentrale Nervensystem (*ZNS*) weiter. Das ZNS ermittelt die Geschwindigkeit und Bewegung des Körpers, indem es den Wechsel der Winkel der Gelenke pro Einheitszeit auswertet.
Der DL, um zwischen verschiedenen Positionen zu unterscheiden, ist vom jeweiligen Gelenk abhängig. 300 Versuche mit drei Teilnehmern zeigen, dass der DL zwischen $2,0$ Grad am Handgelenk und $0,8$ Grad an der Schulter variiert. Finger haben einen Positions-DL von $2,5$ Grad, der Ellbogen von $2,0$ Grad (Tan et al. (1994), zitiert in (Burdea 1996, Tab. 2.3)).

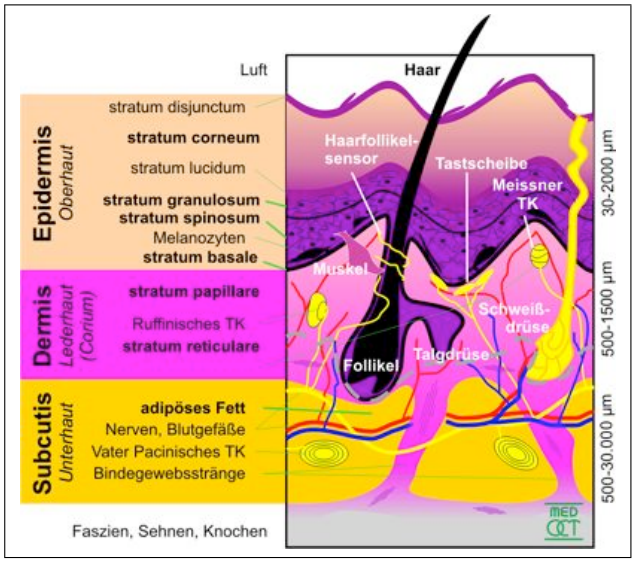


***Bild 2.5:*** *Schematischer Querschnitt durch ein Modell der Haut (Med-OCT-Group 2006)*

Taktile Wahrnehmung wird durch taktile Rezeptoren in der Haut bestimmt (Bild 2.5). Die taktile Wahrnehmung beeinflußt die Propriozeption und Kraftwahrnehmung.

- Die Merkel-Zellen und Ruffini-Körperchen reagieren auf Druck. Sie adaptieren sich langsam auf Druckreize. Damit detektiert ein Mensch das Auftreten von Druck und dessen Intensität.
- Die Meissner-Körperchen und Haarfollikel-Sensoren reagieren auf Berührung. Sie adaptieren sich schnell auf diese Reize, aber unterscheiden deren Intensitäten nicht.
- Die Vater-Pacini-Körperchen reagieren auf Vibration. Sie adaptieren sich sehr schnell auf die ausgeübten Reize. Die höchste Empfindlichkeit liegt bei 300 Hz. Die Detektion der Intensität ist dementsprechend eingeschränkt.

- Die Thermorezeptoren leiten Wärmereize und die Nozizeptoren Schmerzreize an das ZNS weiter.

Allen Rezeptoren gemeinsam ist die Art, wie sie physikalische Reize in Potenzialsentladungen überführen. Die Verformung der Rezeptor-Kapseln ändert den Ausschlag der Potenzialsentladung der Rezeptoren. Das ZNS nimmt die Entladungen entgegen und ermittelt aus ihnen die Körperposition und die auf den Körper wirkenden Kräfte.
Tab. 2.2 nach Burdea (1996) faßt die Namen und Bedeutungen der haptischen Rezeptoren zusammen.

| Qualität | Rezeptor | Adaptation |
|---|---|---|
| Druck | Merkel-Zelle, Ruffini-Körper | Langsam |
| Berührung | Meissner-Körperchen, Haarfollikelrezeptor | Schnell |
| Vibrationen | Vater-Pacini-Körperchen | Sehr schnell |
| Schmerz | Nozizeptor | Nicht |
| Temperatur | Thermorezeptor | (Langsam) |

***Tabelle 2.2:*** *Arten von Nervenzellen nach Burdea (1996)*

Es gibt keine gesicherten Erkenntnisse über das komplexe Zusammenspiel zwischen den Rezeptoren und dem ZNS, wie folgendes Zitat besagt: „[T]he precise contribution of signals arising from joint and cutaneous receptors to proprioception remains unclear because inconsistencies in present experimental results have been reported in the literature." (Burdea 1996, S. 21)
Bekannt ist, dass das ZNS Positionsinformationen voneinander getrennt und in Kombination miteinander verarbeitet. In Kombination verbindet es die Positionsinformationen von den Gelenksensoren mit den Daten von anderen Sensoren, beispielsweise den Golgi-Sehnenorganen und den Muskelspindeln.

- Die Golgi-Sehnenorgane sind zwischen den Muskeln und Sehnen und dienen dem teilweisen Erfassen der Propriozeption und Krafteinwirkung. Als kinetische Sensoren messen sie die Spannung der Sehnen, um die Muskel-Kontraktion und Feinmotorik zu unterstützen.
- Die Muskel-Spindeln zwischen den Muskelfasern werden durch das Spannen der benachbarten Muskelfasern angeregt. Die Spindeln messen die Rate der Muskelverlängerung.

Ermüdung beeinflußt das Kraftempfinden, so dass nach längerer Belastung durch eine konstante Krafteinwirkung diese allmählich als immer stärker empfunden wird (Burdea 1996, S. 21).
Über das Empfinden der Nachgiebigkeit von Objekten, d. h. Viskosität, liegen keine genauen Erkenntnisse vor. Die Diskrepanz zwischen zwei als gleich empfundenen Viskositäten reicht von 34% bei 1024 N-sec/m als Referenz bis 83% bei 2 N-sec/m als Referenz (Burdea 1996, S. 33). Die Schwierigkeit, exakt zu messen, ist mutmaßlich darauf zurückzuführen, dass der Mensch die Kraft des Widerstands und die Tastgeschwindigkeit zur Bestimmung der Viskosität auswertet.

### 2.3.4 Haptische Geräte

Eines der ersten Perzeptualisierungs-Systeme, bestehend aus Rechnern und Roboterarmen, das zur Kraftübertragung auf einen Menschen eingesetzt wurde, war GRO-

PE von Brooks Jr. et al. (1990). Noch im Jahr 1997 schreiben Rosenblum und Cross (1997): „Haptic or force feedback is the least mature of the three information channels discussed [visual, auditory, haptic]. [...] While force feedback devices becoming available, some at 'moderate costs', research efforts are taking place to evaluate the role of haptic feedback as part of multimodal interactions within virtual environments."
Hayward et al. (2004) spezifiziert die seitdem entwickelten haptischen Geräte, deren Klassen wir in Tab. 2.3 sehen. Deren Entwurf umfaßt das Anwenden von Kenntnissen aus der Robotik, experimentellen Psychologie, Biologie, Informatik, Maschinensteuerung und dem Maschinenbau.

| | |
|---|---|
| Programmierbare Tastatur | Exoskelett |
| Desktop-Gerüst | Handschuh |
| Maus | Punkt-Interaktionsgerät |
| Joystick | Pantograph |
| Drehknopf | Trackball |

***Tabelle 2.3:*** *Klassen von haptischen Geräten nach Hayward et al. (2004)*

Ein kommerziell erhältliches Punkt-Interaktionsgerät ist die Sensable *Phantom*, von der ein Exemplar in Bild 2.6 zu sehen ist (Burdea 1996). Die Phantom ist ein in

***Bild 2.6:*** *Die* Phantom *1.5A (Foto: Dipl-Ing. Hassan Mahramzadeh)*

unterschiedlichen Größen erhältlicher Roboterarm, der an einem Ende durch die Hand des Nutzers geführt wird. Das Modell 1.5A wird über ein serielles Kabel an die Karte eines Personalcomputers angeschlossen. Ein Treiber, der die drei Freiheitsgrade des Roboterarms mißt und fallbezogen regelt, bedient sie. Der Treiber mißt die drei Drehmomente des *End-Effektors*, ohne sie zu regeln.
Tab. 2.4 spezifiziert die technischen Details meines Geräts. Die Servomotoren sind so angebracht, dass sie den Roboterarm in Ruheposition auspendeln und damit das Gewicht des menschlichen Arms und der Schwerkraft ausgleichen. Dadurch braucht das Gerät keine aktive Kompensierung für die auf den Roboterarm einwirkende Schwerkraft. Die Wechselstrom-Motoren mit optischen Encodern arbeiten weitgehend ortho-

| Parameter | Wert |
|---|---|
| Arbeitsraum | 19,50 x 27,00 x 37,50 cm |
| Bewegungsraum | Bewegung des Unterarms, Maximum am Ellbogen |
| Nominale Positionsauflösung | 860,00 dpi (0,03 mm) |
| Reibung | 0,04 N |
| Maximale ausgeübte Kraft | 8,50 N |
| Kontinuierlich ausgeübte Kraft | 1,40 N |
| Steifheit | 3,50 N/mm |
| Empfundene Trägheit | $<$ 75,00 g |
| Größe | 25,00 x 33,00 cm |
| Kraftrückkopplung | x, y, z |
| Positionsbestimmung | x, y, z (3 Freiheitsgrade) |
| Schnittstelle | Via Parallelport |

***Tabelle 2.4:*** *Technische Spezifikation der* Phantom *1.5A nach Sensable (2002)*

gonal, d. h. eine Kraft wird in alle Richtungen durch dasselbe Maß ihrer Ansteuerung erzeugt. Die Kraftwiederholrate beträgt circa 800 Hz (Burdea 1996, S. 89). Das liegt oberhalb der Bandbreite von 30 Hz, über der propriozeptive und kinetische Impulsfolgen nicht mehr unterschieden werden (Shimoga (1992), zitiert in Burdea (1996)).

### 2.3.5 Indirektes und direktes Rendering

Haptisches Rendern funktioniert so, dass Software, welche den Treiber nutzt, die Koordinaten des End-Effektors abfragt. Parallel dazu setzt sie einen virtuellen Punkt mit den gleichen Koordinaten in eine 3-D-Szene, welche die Geometrie der zu ertastenden Figur enthält. Die Kollisionsermittlung errechnet, wann der Punkt die Figur berührt. Es wird eine geeignete Kollisionsantwort generiert und an das haptische Gerät zurückgesendet.

#### Kollisionsermittlung

Dadurch, dass die Koordinaten des Punkts sich mit bis zu 800 Hz ändern, bleiben für die Kollisionsermittlung und -antwort pro Zeitschritt $1000$ msec / $800$ Hz $= 1,25$ msec. Der überwiegende Teil wird für das Ermitteln aufgewendet und dessen Zeitbedarf wird wie folgt formalisiert, wenn $N$ die Anzahl der geometrischen Primitive und $t_c$ die Zeit zum Berechnen einer Kollision sind:

$$T_{naive} = N \cdot t_c \tag{2.11}$$

Es ist für große $N$ ein dreistufiges Verfahren notwendig, bei dem zunächst grob vorsortiert wird, welche Objekte dem End-Effektor nahe sind, welche der nahegelegen mit ihm zusammenstoßen könnten und für welche dies zutrifft, so dass eine Kontaktantwort generiert werden muß. Sei $T_d$ die Summe der Zeitaufwände, um aus den $N$ Primitiven die zum End-Effektor $K$ nächstgelegenen Objekte zu ermitteln, um daraus die Kontaktpaare zu isolieren. Sei dabei $t_s$ der Aufwand pro Objekt, dann ist der maximale Zeitaufwand für das Ermitteln von Kontaktpunkten (Gregory et al. 2000)

$$T_d = N \cdot t_s + K \cdot t_c \tag{2.12}$$

Es soll

$$T_d < N \cdot t_c \tag{2.13}$$

gelten. Somit ergibt sich

$$\begin{aligned} N \cdot t_c &> N \cdot t_s + K \cdot t_c \qquad (2.14)\\ N \cdot t_c - K \cdot t_c &> N \cdot t_s \\ t_c - \frac{K}{N} \cdot t_c &> t_s \\ t_s &< t_c - \frac{K}{N} \cdot t_c \qquad (2.15) \end{aligned}$$

Die Verringerung von $t_s$ bedeutet, eine Äquivalenzrelation basierend auf unveränderlichen Relationen zwischen den Primitiven zu bauen und im Term $N \cdot t_s$ nur die Vertreter unterschiedlicher Äquivalenzklassen auf Nähe zum End-Effektor zu testen. Wenn wir $N$ im Term belassen, verteilt sich somit die Zeit $t_s$, um einen Vertreter zu testen, auf alle Primitive, die vom Vertreter repräsentiert werden. Eine gleichwertige Sichtweise wäre, $t_s$ zu belassen und die Anzahl der Vertreter $N'$ einzuführen, so dass $N' < N$ ist. Gleichung 2.13 kann nicht zu $K < N$ abgeschwächt werden. Man betrachte hierzu den Extremfall $K = 0$ (d. h. kein Primitiv in der Nähe). Die starke Bedingung ergibt hierfür

$$\begin{aligned} t_s &< t_c - \frac{0}{N} t_c \\ t_s &< t_c \end{aligned}$$

Eingesetzt in $T_d$ ist das

$$\begin{aligned} T_d &= N \cdot t_s \\ &< T_{naive} \end{aligned}$$

Die schwache Bedingung schlösse nicht aus, dass

$$\begin{aligned} T_d &= N \cdot t_s \\ &\geq T_{naive} \end{aligned}$$

und wäre gleichbedeutend damit, *gleichen und zusätzlichen* Rechenaufwand zu tolerieren, wenn keine Kollision vorliegt. Dies ist inakzeptabel, und deshalb muß die starke Bedingung eingehalten werden.

### Kollisionsantwort

Bei $M$ Kollisionen während eines Zeitschritts werden $M$ Antworten generiert, wovon jede $t_f$ Zeit zur Berechnung erfordert. Somit ist der Aufwand $T_h$ für das Ermitteln *und* Antworten

$$T_h = T_d + M \cdot t_f \tag{2.16}$$

Es soll $M \cdot t_f < K \cdot t_f < N \cdot t_f$ gelten. Die Betrachtungen aus dem letzten Abschnitt sind sinngemäß wieder darauf anzuwenden.
Finite-Elemente-Methoden (FEM) sind wegen des Rechenaufwands nicht praktikabel. Stattdessen nutzen Programmierer zur Generierung der Kontaktantwort mit Oberflächen das Masse-und-Feder-Modell. Die folgende Betrachtung des Problems ist Deutsch (2006) entnommen.

Gegeben sei eine mechanische Feder, welche an einem Ende mit einer Masse und am anderen Ende mit einer Oberfläche verbunden ist. Das Verhalten der Feder, über eine Dauer $T$ zu oszillieren, hängt von der Konstanten $k$ und der Masse $m$ ab. Aus der Physik sei $F = m \cdot a$ (Kraft ist Masse mal Beschleunigung) vorausgesetzt. Es gilt $F = -k \cdot x$. Seien $k$ eine Hilfskonstante und $x$ die Position (wir betrachten den eindimensionalen Fall). Damit ist

$$a = \frac{k}{m} \cdot x \tag{2.17}$$

wobei $k$ frei gewählt wird, um die Materialeigenschaft zu setzen und somit, um abhängig von der Eindringungstiefe $x$, sowie der Richtung und der Masse $m$ des End-Effektors, die Kollisionsantwort zu formulieren. Zusätzlich muß die Ausrichtung der Oberflächennormalen für drei Dimensionen berücksichtigt werden.

Direktes Rendern ist die Anwendung eines anderen Rendering-Stiles, zum Beispiel dem zum Nachahmen von Viskosität. Seien $k$ die Dichte und $v$ die Geschwindigkeit des End-Effektors, dann lautet die Gleichung

$$f = -k \cdot v \tag{2.18}$$

#### Anwendungen

Verglichen mit dem haptischen Rendern von Distanzfeldern oder Polygonen, wie das Brooks Jr. et al. (1990) und Mark et al. (1996) beschrieben haben, bedeutet haptisches, direktes Volumenrendern das Rendern von 3-D-Texturen. Avila und Sobierajski (1996), Huang et al. (1998) und McNeely et al. (1999) berichten davon, Voxelwerte auf Viskosität abzubilden. Ruspini et al. (1997); Lawrence et al. (2000); Vidolm und Nystrom (2005) bauen mit diesen Techniken Modellierungs- und Perzeptualisierungs-Systeme.

### 2.3.6 Erweiterung der Visualisierungs-Pipeline

Um ein Perzeptualisierungs-Systems zu entwerfen, berücksichtige ich Beobachtungen von McNeely et al. (1999); Lawrence et al. (2000); Gregory et al. (2000).

1. Die Geometrien für das grafische und haptische Rendering können unterschiedlich sein. Es ist daher falsch anzunehmen, ausgehend von der Geometrie für das grafische Rendering, die für das haptische Rendering ableiten zu können.

2. Wegen der hohen Kraftwiederholrate für die haptische Ausgabe kann es notwendig sein, das haptische Rendering auf einen anderen Prozessor auszulagern.

Beidem wird man gerecht, indem man die Visualisierungs-Pipeline um einen parallelen Datenkanal zum Generieren von haptischen Ausgaben erweitert. Das ist am einfachsten realisierbar, indem man zwei Instanzen der Visualisierungs-Pipeline nutzt und bei einer den Renderer ersetzt.

Das Rendering für die haptische Ausgabe muß durch die Ansteuerung eines haptischen Geräts erfolgen. Die Treiber und Software Development Kits (*SDKs*) der Geräte sind dafür zuständig, aus den Geometriedaten haptische Figuren zu erzeugen.

## 2.4 Techniken zur Datenreduktion

### 2.4.1 Clipping

Clipping prüft, ob sich ein Objekt im Sichtkegel oder in Tastnähe des Nutzers befindet. Man definiert hierzu den Sichtkegel oder die Tastnähe durch einen Quader mit einem bestimmten Raummaß, auch *Frustum* genannt. Jedes Primitiv wird geprüft, ob es zumindest in Teilen im Quader ist. Für Voxel ist der Test trivial. Sutherland et al. (1974) geben Algorithmen, die Polygone dementsprechend prüfen.
Mit Sutherland-Hodgeman werden die Seiten des Quaders durch Ebenen repräsentiert, die jeweils durch einen Punkt $\vec{p}_0$ und eine Normale $\vec{n}$ definiert werden. Die Normalen der Ebenen sind zueinander gerichtet.
Der Punkt $\vec{p}$ eines Polygons wird nun daraufhin geprüft, ob er im Quader ist (innen), oder nicht (außen). Das ist genau dann der Fall, wenn die Bedingung

$$(\vec{p} - \vec{p}_0) \cdot \vec{n} \geq 0 \tag{2.19}$$

gilt. Der Algorithmus ist aus Foley et al. (1997); Charlton (2006):

```
For each edge, check both nodal values, s and p
If the point values are
 1. Inside-inside, append the second node, p
 2. Inside-outside, compute and append the intersection i
     of edge sp with the clipping plane
 3. Outside-outside, no operation
 4. Outside-inside, compute and append the intersection i
     of edge sp with the clipping plane,
     then append the second node p
```

Die geordnete Liste der Knoten ist das „geclippte" Polygon. Folgende Optimierungen sind angebracht:

1. Wenn die Zelle, die das Polygon umgibt, außerhalb des Quaders ist, wird die leere Menge ausgegeben.

2. Wenn die Zelle, die das Polygon umgibt, komplett innerhalb des Quaders ist, wird das gesamte Polygon ausgegeben.

### 2.4.2 Backface-Culling

Wenn ein Polygon als Gleichung gegeben ist, kann man über den Test

$$Ax + By + Cz + D < 0 \tag{2.20}$$

die Richtung der Oberflächennormale im Vergleich zum Blickwinkel ermitteln. Ein positives Ergebnis bedeutet, dass die Normale vom Betrachter wegzeigt und das Polygon nicht gezeichnet zu werden braucht.
Für das verteilte Filtern und Rendern geeignete Formen werden von Luebke und Erikson (1997); El-Sana et al. (2001) beschrieben. In beiden Varianten wird der aktuelle Blickwinkel genutzt, um verdeckte und rückwärtige Anteile einer virtuellen Szene zu entfernen, und um die verbleibenden Polygone an den Renderer zu senden.

Ein Verfahren für haptische Ausgaben, das Backface-Culling nutzt, ist von Johnson und Willemsen (2003). Die Zielsetzung besteht darin, Geräte mit sechs kontrollierbaren Freiheitsgraden anzusteuern. Angenommen, es bewege sich mit dem End-Effektor ein virtuelles Objekt. Es erlaubt die simultane Rückgabe mehrere Kontaktpunkte und somit eine umfassendere Weitergabe von haptischen Informationen. Der Algorithmus verarbeitet die Geometriedaten, um korrekte Oberflächennormalen zu generieren. Dann bestimmt er iterativ, welche Polygone zum Rendering zugelassen werden. Das Kriterium ist, Polygone zu selektieren, die einander zugewandte Oberflächennormalen haben, weil nur solche miteinander kollidieren.

### 2.4.3 Quantisierung

Quantisierung ist „die Darstellung einer Größe in einem [Zahlens]ystem, in dem sie nur diskrete Werte annehmen kann" (Wikipedia 2006). Es werden die absolute und die relative Quantisierung unterschieden, womit die Veränderlichkeit des Zahlensystems in Abhängigkeit von den benachbarten Größen ausgedrückt wird. Quantisierung wird in der Regel mit anderen Techniken kombiniert, um den Informationsverlust nicht augenscheinlich werden zu lassen. Chiueh et al. (1997) beschreibt ein Beispiel zur Quantisierung von Frequenzanteilen, die mittels der *Wavelet-Analyse* extrahiert wurden.

### 2.4.4 Vereinfachung

Vielfach bestehen die erzeugten Polygonnetze aus zu vielen Dreiecken, um sie interaktiv zu rendern. Das Problem, sie zu vereinfachen, ist in $\mathcal{NP}$. Jeder polynomiale Polygonnetz-Vereinfacher arbeitet somit zwingend sub-optimal. Ein Weg ist, ein Volumen zu subsampeln, um weniger Polygone zu erzeugen, wie Lamphere et al. (1999), Taubin (2000) und Taubin (2002) gezeigt haben. Aber diese Lösung verliert *Features* zwischen zwei adjazente Samples. Außerdem ist die Kontrolle der Qualität schwierig. Varianten des Subsamplings passen sich daher ortsgebunden der Informationsdichte an.
Es gibt Ansätze, welche die Datengrößen auf 1–3% reduzieren. Gelder und Wilhelms (1994), Rossignac (1997), Brodlie und Wood (2000) und Sutton et al. (2000) fassen Entwürfe zusammen, die auf Greedy-Verfahren basieren. Tab. 2.5 vergleicht die Entwurfsarten. Sie beinhalten Klassifizierungs- und Vereinfachungs-Schritte, um zwischen visueller Qualität und Datengröße auszugleichen, siehe Luebke (2001). Fehlermetriken dienen der Kontrolle der Abweichung von den Eingabedaten (Cohen et al. 1996; Ciampalini et al. 1997; Garland und Heckbert 1997; Lindstrom und Turk 1999; Gerstner und Pajarola 2000).
Zuerst vereinfacht man das Problem dadurch, dass man die Polygone in Dreiecke teilt. Vertex-Decimation entfernt die Ecken in einem Dreiecksnetz, die am wenigsten zur Gestalt der Oberfläche oder des Umrisses beitragen. Hierzu wird die Gestalt anhand der Fehlermetrik vor und nach dem probeweisen Entfernen einer Ecke evaluiert. Wenn der Fehler nach dem Entfernen einen beliebigen aber festen Schwellwert nicht überschreitet, wird das Entfernen angenommen. Das Verfahren iteriert, bis alle überflüssigen Ecken weg sind. Um das Prinzip zu verdeutlichen, soll die einfachste Qualitätsmetrik gezeigt werden, die Distance-To-Surface. Hierbei werden die drei nächsten Ecken gewählt, welche die probeweise zu entfernende Ecke umgeben und mit ihr über Kanten verbunden sind. Man bildet zwischen den zu erhaltenden Ecken ein Dreieck. Der Abstand zwischen der probeweise zu entfernenden Ecke und dem Dreieck ist der Fehler. Der Fehler ist genau dann 0, wenn die Ecke auf dem Dreieck liegt. Dieser Fall

| Name | Vorgehen | Vor- und Nachteile |
|---|---|---|
| Vertex-Decimation | Wähle, ordne, entferne Ecke und retrianguliere | + Schnell, topologieschonend<br>- Spezialisiert |
| Iterative Contraction | Reduziere Dreieck zu Ecke und zwei Ecken zu einer | + Anpaßbar<br>- Fehlerminimierung schwierig |
| Edge-Collapsing | Reduziere Kante zu Ecke | + Topologieschonend<br>- Benötigt Topologie |
| Vertex-Clustering | Teile Netz in äquidistante Kuben gleicher Größe und clustere die Ecken pro Kubus | + Vielfältig<br>+ Schnell, parallelisierbar<br>- Nicht topologieerhaltend<br>- Abhängig von Kuben-Ausrichtung |

***Tabelle 2.5:*** *Übersicht über Polygonnetz-Vereinfacher*

tritt bei planaren Oberflächen auf. Eine kompliziertere Qualitätsmetrik spannt zwischen den Ecken eine Funktion höherer Ordnung auf und berechnet den Fehler (Garland und Heckbert 1997).

Es ist wichtig, nicht alle Ecken mit gleicher Priorität probeweise zu entfernen. Ein Ansatz besteht darin, sie anhand einer Heuristik in einer Priority-Queue einzureihen (Garland und Heckbert 1997). Der Nachteil ist, dass dadurch das Vereinfachen serialisiert wird, weil man keine Ecke entfernt, ohne sie vorher zu priorisieren.

Iterative Contraction ist der Sammelname für Algorithmen, die nicht in die anderen Kategorien fallen. Dazu gehören Algorithmen, die Dreiecksnetze neu abtasten und anhand nutzerbestimmter Kriterien über Kanten zusammenhängende Ecken entfernen. Man bevorzugt Edge-Collapsing gegenüber Iterative Contraction, weil man dadurch die Fehler besser kontrolliert.

Garland und Heckbert (1997) beschreiben einen Edge-Collapsing-Algorithmus. Die meisten neueren Arbeiten sind Derivate davon. Edge-Collapsing priorisiert die Kanten und entfernt probeweise jeweils eine, indem es die beiden Eckenkoordinaten auf deren gewichtetes arithmetisches Mittel setzt und danach eine der Ecken durch die andere in allen Kanten ersetzt.

Man nutzt Edge-Collapsing, um topologiebehaftete Dreiecksnetze zu reduzieren. Für andere Dreiecksnetze reicht Vertex-Clustering. Vertex-Clustering funktioniert anhand einer vorgegeben Gittergröße und -Ausrichtung, die über das Dreiecksnetz gelegt wird. Pro Zelle des Gitters werden die Ecken geclustert, d.h. gemittelt. Das Ergebnis weist, abhängig von der Ausrichtung der Zellen, Fehler auf.

Ein Polygonnetz-Vereinfacher bearbeitet entweder ein komplettes, detailliertes Netz, oder er bearbeitet ein partielles Netz, wenn er mit dem Polygongenerator verschränkt wird. Letzteres hat den Vorteil, dass stets nur Teile des detaillierten Netzes im Speicher liegen und der Klassifikationsaufwand exakt einmal vorgenommen wird. Treece et al. (1999) folgen dieser Idee. Sie kombinieren Marching-Tetrahedra und Vertex-Clustering, um Äquipotenzialflächen zu generieren, die topologisch konsistent mit den Daten sind und aus 70% weniger Dreiecken bestehen. Treece et al. (1999) vergleichen weder die Auswirkung mehrerer Cluster-Stufen, noch berichten sie über die datenparallele Nutzung.

Otaduy und Lin (2003) beschreibt eine Arbeit, in der Nutzer fühlbare Objekte aneinander stoßen lassen, was als Basis dafür dient, das Montieren von Teilen zu üben. Die Teile werden als Polygonnetze modelliert, welche ein Vereinfachungs-Algorithmus unter Erhalt ihrer Oberflächenkrümmung reduziert (Otaduy 2004, S. 104).

### 2.4.5 Datenkomprimierung und -dekomprimierung

**Motivation für die Entwicklung spezialisierter Algorithmen**

Zip ist ein weitverbreitetes Werkzeug zum Komprimieren und Dekomprimieren von Daten. Implementierungen von Zip nutzen den *LZW*-Algorithmus von Welch (1984). LZWs Vorteile sind die Verfügbarkeit, Flexibilität, Geschwindigkeit, Robustheit und Effektivität. Aber zum Komprimieren von Volumen oder Polygonnetzen sind LZW und andere generalisierte Verfahren sub-optimal, weil sie die den Datentypen inhärente Information nicht memorisieren. LZW auszuführen braucht somit mehr *CPU*-Zyklen als notwendig wäre. Spezialisierte Verfahren haben den Nachteil nicht.

**Spezialisierte verlustlos arbeitende Algorithmen**

Fowler und Yagel (1994) spezifizieren den Compvox-Algorithmus, der Voxelreihen in 3-D folgt. Er nutzt einen Prädiktor, der einen Voxelwert anhand orthogonaler Nachbarvoxel vohersagt. Compvox errechnet die Gewichtungskoeffizienten des Prädiktors in sieben vollständigen Durchläufen durch die Eingabedaten. Weil die Werte verrauscht sind, kodiert Compvox nur die *Most Significant Bits* über den Prädiktor und die *Least Significant Bits* zusätzlich über die *Digital Pulse Code Modulation*. Die Prädiktorencodes werden mittels *Delta-* und *Huffman-Kodierung* gespeichert. Das Ergebnis ist um 50% reduziert.
Engelson et al. (2000) schlagen vor, ein Volumen zu traversieren und Folgen von Werten zu hashen. Jeder neue Hash-Wert wird gespeichert, jeder alte Hash-Wert wird als Ersatz für die Folge von Werten, die er repräsentiert, geschrieben.
Ratanaworabhan et al. (2006) spezifizieren einen ähnlichen Ansatz, aber sie ersetzen die Berechnung der Hash-Funktion mit einem Prädiktor über zeitveränderliche Daten, der die polynomiale eindimensionale Extrapolation korrespondierender Werte repräsentiert.
Lindstrom und Isenburg (2006) verbessern den Ansatz, indem sie effizient mit Fließkommawerten arbeiten. Sie berichten davon, einige große Datensätze komprimiert zu haben. Kern des Ansatzes ist die Nutzung des Lorenzo-Prädiktors, der erstmals in Ibarria et al. (2003) erwähnt wurde. Er behält die Auflösung der Datensätze und ist somit als Vor- oder Nachbearbeiter geeignet, der mit anderen Datenreduktionen, wie beispielsweise *Resampling*, verbunden wird. Dies wird allerdings nicht im Artikel besprochen.
Forscher haben zudem mit aus der 2-D-Bildverarbeitung bekannten Techniken experimentiert (Hargreaves et al. 1997). Man erzielt Reduktionsraten von 1:3 durch Hintergrundunterdrückung und den Einsatz von Prädiktion innerhalb der Schnittflächen. Die Autoren berichten von *schlechteren* Resultaten wenn sie

- Prädiktoren zwischen den Schnittflächen einsetzen,
- die Schnittflächen durch invertierbare Operatoren in niedrigere Auflösungen umwandeln,
- zwei Scan-Hälften in Symmetrie zueinander setzen und
- Scans als die Perturbation eines Referenzbildes deuten.

Die Tests sind nur für Magnetresonanztomographie-Scans des Hirns aussagekräftig. Die Autoren berichten weder von anderen Anwendungen, noch geben sie die Ausführungszeiten ihrer Algorithmen preis.

Eine konzeptionell einfache Technik stammt von Klappenecker et al. (1998). Sie teilt die Eingabedaten in Unterfrequenzen und blocksortiert sie. Ein arithmetischer Kodierer bearbeitet sie nach. Ein Teil der Technik gleicht der Hintergrundunterdrückung aus Hargreaves et al. (1997), weil sie Eingabedaten in relevante und irrelevante Regionen klassifiziert. Der Vorteil der Technik ist, dass der Dekomprimierer *progressiv* dekodiert. Die Komprimierungsrate von 1:3 wird allerdings für nur einen Datensatz gezeigt.
Steingötter et al. (1998) vergleichen, wie *Lauflängenkodierung*, Huffman-Kodierung und die *Diskrete Cosinus-Transformation* mit eingebauter Quantisierung arbeiten, wenn sie um die Vorhersage zwischen den Schnittflächen erweitert werden. Es wird behauptet, einen medizinischen Scan der menschlichen Brust mit 1:3 komprimiert zu haben, aber es werden keine Details über die Beschaffenheit der Daten oder die Algorithmen gegeben.
Andere Techniken basieren auf Wavelets, zum Beispiel 3-D-SPIHT und sein Nachfolger AT-SPIHT (Cho et al. 2004). SPIHT teilt die Eingabedaten in einen Baum, um die Daten zu sortieren, zu komprimieren und zu dekomprimieren. SPIHT speichert die Wavelet-Koeffizienten, die während der Berechnung erzeugt werden, in *Bitplanes*, sortiert diese nach ihrem Betrag und extrahiert deren Least Significant Bits. Für jede „Wichtigkeitskategorie" werden die Daten in den *Ausgabestream* geschrieben.
AT-SPIHT ist ein in der Performance verbessertes 3-D-SPIHT. Es ändert die Ausgeglichenheit des Wavelet-Koeffizienten-Baums, um bessere Reduktionen zu erzielen. Die Reduktion auf 25% der ursprünglichen Datengröße ist das beste veröffentlichte Ergebnis. Ein anderer Vorteil ist, dass SPIHT, wie jede Wavelet-basierte Technik, progressives Dekodieren unterstützt. SPIHT läuft jedoch mehr als einmal durch die Daten. Weiterhin ist der Algorithmus komplex und daher bedingt für den Einsatz in Systemen konzipiert, die robuste Algorithmen nutzen sollen. Von praktischer Relevanz ist, dass SPIHT patentiert ist.
Polygonnetz-Reduktion ist ein weiteres Einsatzfeld (Taubin und Rossignac 1998; Rossignac 1999; Lindstrom 2000; Chen und Nishita 2002). Edgebreaker ist das wichtigste Verfahren:

- Identifiziere redundante Topologien durch Prädiktoren,
- speichere die Codes und
- komprimiere die Koordinaten der Ecken mittels Huffman- oder *arithmetischer Kodierung*.

Der Algorithmus kodiert, im Mittel, jedes Dreieck mit zwei Bit.

#### Verlustbehaftung und haptische Wahrnehmungsneutralität

Verlustbehaftete Komprimierung wird in Chiueh et al. (1997), Yang (2000) und Krishnan et al. (2004) spezifiziert. Die Daten werden frequenz-, orts- oder merkmalsbezogen zerlegt und quantisiert. Es gibt keine Komprimierer zum Verarbeiten von für das haptische Rendering bestimmten Daten.

## 2.5 Parallele Systeme und Rechnernetze

Ein paralleles System besteht aus miteinander kommunizierenden Prozessoren und Speicher zum gleichzeitigen Ausführen von Algorithmen. Man unterscheidet die folgenden Architekturen:

- Single-Instruction-Multiple-Data (SIMD)
- Multiple-Instruction-Single-Data (MISD)
- Multiple-Instruction-Multiple-Data (MIMD)

SIMD ist die Architektur der Vektorrechner. Ein Vektorrechner lädt Daten in den Speicher und führt auf ihnen dieselbe Instruktion pro Takt aus. MISD wäre die Architektur eines Rechners, der verschiedene Instruktionen auf demselben Datum ausführt. MIMD ist die flexibelste Architektur, bei der mehrere Prozessoren verschiedene Daten bearbeiten. Die heutzutage gängigen Rechencluster unterstützen MIMD.
Es gibt mehrere Arten der *Parallelisierung*. Datenparallelität besagt, dass die Daten zwischen Prozessen aufgeteilt werden, die denselben Algorithmus ausführen. Scatter/Gather sendet die Eingabe durch einen Masterprozeß an die anderen Prozesse, und es sammelt die Ausgaben beim Masterprozeß wieder ein. Der Nachteil ist die mangelnde Ausführbarkeit beim Ausfall des Masterprozesses. Weiterhin ist es schwierig, die Arbeitslast vorab gleichmäßig aufzuteilen, um den Datendurchsatz zu optimieren.
Funktionsparallelität heißt, den Algorithmus auf Prozesse zu verteilen. Ein bekanntes Verfahren ist Pipelining, das verschiedene Instruktionen zeitverschränkt ausführt und somit, nach einer einmaligen Anlaufzeit, pro Takt eine neue Ausgabe erzeugt. Die Schwierigkeit liegt darin, die Instruktionen gleich zeitaufwendig zu halten.
Dynamische Parallelität optimiert die gleichmäßige Aufteilung der Arbeitslast, indem die Datenmenge und die auszuführenden Instruktionen von den Prozessoren bedarfsweise bestimmt werden. Die Nachteile sind der höhere Aufwand und die Fehleranfälligkeit des Ansatzes.
Die Parallelisierung der Ausführung eines Algorithmus auf $P$ Prozessoren erfordert die Minimierung des Teils $a$, der sequentiell ausgeführt werden muß, und des Kommunikationsaufwands $o(P)$. Je besser die Parallelisierung gelingt, desto höher ist nach Amdahl (2000) der Speedup $s$:

$$s = \frac{1}{a + o(P) + \frac{1-a}{P}} \leq \frac{1}{a} \tag{2.21}$$

Die Effizienz ist der Speedup bei $P$ Prozessoren dividiert durch $P$. Das ist das prozentuale Maß für die Auslastung jedes Prozessors.
Ein *verteiltes System* besteht aus Prozessoren und Speicher zur netzwerkverteilten Ausführung von Algorithmen, wobei hier der Schwerpunkt nicht auf der Parallelität, sondern auf der Verteilung großer Datenmengen liegt. Dies erfordert den Transfer von Datenpaketen über ein Netzwerk, bei dem folgende Faktoren beachtet werden:

- Latenz, d. h. die Verzögerung des Empfangs aller Pakete
- Delay, d. h. die Verzögerung des Empfangs einzelner Pakete
- Jitter, d. h. die Schwankungen der Verzögerungen bei der Paketübertragung
- Packet loss, d. h. den Verlust von Paketen
- Datendurchsatz, d. h. die Anzahl der empfangenen Pakete pro Sekunde

Deren Kontrolle wird, in Anlehnung an das ISO/OSI-Modell, mit den Protokollen IP, TCP und UDP realisiert (Grimm und Schlüchtermann 2005, Kap. 3). Der Hauptzweck vom Internet Protocol (IP) ist es, vom Übertragungsmedium abstrahieren zu können.

IP realisiert eine Ende-zu-Ende-Verbindung zwischen Quellen und Senken über heterogene Übertragungsmedien. Weiterhin kontrolliert es die Fragmentierung von Daten, und es ermittelt deren Lebensdauer, um höher gelagerte Protokolle zu unterstützen, beispielsweise TCP und UDP.
Das Transmission Control Protocol (TCP) bietet eine Punkt-zu-Punkt-Verbindung mit verläßlicher Übertragung bei hohem Übertragungsaufwand. Die Daten werden auf einen Stapel gelegt und in Blöcken übertragen. Beim Senden wird eine Empfangsbestätigung angefordert. Falls ein Paket verloren geht, bleibt die Bestätigung aus, und es wird nochmal gesendet. TCP garantiert, dass die ursprüngliche Reihenfolge der Pakete nicht vertauscht wird, indem es auf Pakete in der Folge wartet, in der sie geschickt werden sollen.
Das User Datagram Protocol (UDP) ist ein leichtgewichtiges Datenübertragungsprotokoll und komplementiert TCP. Die Pakete werden sofort abgeschickt und empfangen. Es gibt keinen Schutz gegen Paketverlust und Reihenfolgenverkehrung. TCP ist für die Übertragung von Streams geeignet, UDP ist für die zeitnahe Übertragung von Signalen geeignet, deren Verwendung allein die Anwendungsschicht kontrolliert.
Darauf aufbauende Anwendungsschichten folgen dem Client-/Server- oder dem Peer-to-Peer-Architekturmuster. Kurz gesagt nutzt ein Client eine vom Server bereitgestellte Funktion. Ein Peer vereint beide Rollen. Steinmetz und Wehrle (2005) spezifiziert Grundlagen und Anwendungen.

## 2.6 Software

In dem Abschnitt verdeutliche ich den Einsatz ausgewählter integrierter Systeme, siehe ergänzend Kachina Technologies (2002).

### 2.6.1 Amira

Amira wird seit 1994 am Konrad-Zuse-Zentrum Berlin (ZIB) entwickelt (Kähler et al. 2003). Die Software hat eine grafische Nutzungsoberfläche, die sich aus dem Ausgabefenster, dem Netzplan, dem Modul-Attributfenster und dem Status- und Hilfefenster zusammensetzt. Im Netzplan verbinden die Nutzer Module, um die Ausgaben im Ausgabefenster zu sehen und im Modul-Attributfenster zu verändern.
In der Grundausstattung werden Standard-Visualisierungsmodule und grafische Interaktionsmethoden zum Teilen und Färben von Grafiken geliefert. Zusatzpakete unterstützen das Bilden von Metaphern zur Strömungsvisualisierung, von Animationen und das freihändige Navigieren in großprojizierten Grafiken.
Von besonderem Interesse in meiner Arbeit ist die direkte Skalarvisualisierung mittels 2-D- oder 3-D-Texturen. Hierbei kommt die OpenGL-Erweiterung zum Verwalten und Darstellen von 3-D-Texturen zum Einsatz. Das Large-Data-Add-on verbindet dessen Nutzung mit Out-Of-Core-Rendering, welches Daten von der Festplatte in den Hauptspeicher einlagert und von dort in den Texturspeicher der Grafikkarte schreibt. Dieser Mechanismus basiert auf adaptiven Gitterverfeinerungs-Hierarchien (Berger und Colella 1989). Durch dessen Nutzung verfolgt man das Ziel, nur die für die Anzeige wichtigsten Teile zu rendern und somit den Hauptspeicherbedarf und die Performance des Renderings zu verbessern (Kähler et al. 2003).
Im Vorverarbeitungsschritt durchzieht der Algorithmus das Volumen mit einer groben Gitterstruktur. Diese ist die Wurzel der Gitterverfeinerungs-Hierarchie. An Stellen mit höherer Auflösung wird ein Kindknoten eingefügt, der eine feinere Gitterstruktur als

der Elternknoten aufweist. Der Vorteil gegenüber Octrees ist, dass die Untergitter frei platziert und dadurch wichtige Volumenstellen präzise abgedeckt werden. Man braucht weniger Texturspeicher, weil für freie Stellen kein Texturspeicher alloziert wird.
Für das Darstellen werden die Knoten bedarfsweise eingelagert und von hinten nach vorne gerendert:

- Freie Stellen werden ignoriert, weil es für sie keine Blätter gibt.
- Knoten, die außerhalb des Sichtkegels sind, werden ausgelagert.
- Diejenigen von den anderen Knoten, die kaum zur Darstellung beitragen, werden ausgelagert.

### 2.6.2 AVS/Express Multipipe Edition mit Parallel Support Kit

Das Advanced Visualization System geht aus der Verbindung von Uniras und AVS hervor. AVS ist ein datenflußorientiertes Visualisierungs-System, in dem Module zum Lesen, Filtern und Rendern von Daten zusammengeschaltet werden. Auf diesem Weg werden komplexe Visualisierungslösungen erstellt.
AVS/Express ist eine Weiterentwicklung, für die es die Erweiterungen Multipipe-Edition und Parallel Support Toolkit gibt. Die Multipipe-Edition steuert mehrere Grafikausgabe-Pipes gleichzeitig an. Damit werden mehrteilige Projektionswände gesteuert. Das Parallel Support Toolkit besteht aus Modulen, welche die dem Rendern vorgelagerten Prozesse datenparallelisieren. Beispielsweise wird pro Prozessor eine Instanz von `p_read_field` gestartet, die ein vom Masterprozeß zugewiesenes Datensegment liest. Jeder Prozessor führt `p_crop_struct` und `p_isosurface_struct` aus, um eine Teil-Äquipotenzialfläche zu erzeugen. `UViewer3-D` stellt das Kompositum dar.

### 2.6.3 Covise

Covise wird in Lang et al. (1995) beschrieben. Die verteilte Software besteht aus einem datenflußorientierten Visualisierungskern, der über eine ähnliche Nutzungsschnittstelle wie AVS verfügt und einen vergleichbaren Leistungsumfang an Visualisierungstechniken aufweist. Weitere Module dienen dem gleichzeitigen Replizieren der Grafikausgabe auf mehrere Clients und dem Rückführen der von dort getätigten Eingaben an den Server. Ergänzungen fließen durch Cover und Virvo ein (Wössner et al. 2002).
Cover steuert Projektionsflächen und leitet, von Positionsverfolgern (Trackern) empfangene, Daten an den Visualisierungskern weiter. Entwickler nutzen die Erweiterungs-API, um neue Plugins zu programmieren.
Virvo, die Bibliothek für Virtual-Reality Volume Rendering, implementiert direktes Volumenrendering. Sie kapselt Funktionen von OpenGL und bietet Datenim- und -export.

### 2.6.4 Scirun

Scirun arbeitet datenflußorientiert. Es hat ein Tcl/TK-basiertes Graphical User Interface (*GUI*), mit dem Nutzer Module in der bekannten Weise verbinden, und es stellt Standardmodule zur Volumen- und Strömungsvisualisierung mit OpenGL bereit. Weiterhin bietet die Software Routinen zum Datenimport und -export (of Utah 2002). Das

verteilte System ermöglicht das Steuern von Anwendungen, indem es die vom Nutzer initiierten Kommandos übers Netzwerk sendet.
Scirun verfügt als eines der wenigen Systeme über eine Ergänzung für haptische Geräte (Durbeck et al. 1998). Das haptische Gerät wird mit der grafischen Anzeige auf einem anderen Rechner registriert. Es stellt 3-D-Vektorfelder dar. Die Richtung und Stärke der Vektoren in Nähe des End-Effektors wird, mittels einer nutzerdefinierten Funktion, auf haptische Impulse abgebildet. Durbeck et al. (1998) ergänzt Scirun mit Ghost um die Möglichkeit, Objekte punktweise abzutasten:

```
...
// ljdForceInput derived directly from Ghost SDK CalcEffect
class ljdForceInput : public gstEffect {
public: ...
 virtual gstVector calcEffectForce(void *PHANToMN) {
  ...
  gstPoint pos;
  // force vector components
  double xc, yc, zc;
  xc = scirun_force.x();
  yc = scirun_force.y();
  zc = scirun_force.z();
  return gstVector(xc, yc, zc);
 }
}
void main() {
 gstPHANToM *PHANToM = new gstPHANToM("PHANToM.ini");
 root->addChild(PHANToM);
 scene->setRoot(root);
 // force from SCIRun
 ljdForceInput *forces = new ljdForceInput;
 PHANToM->setEffect(forces);
 PHANToM->startEffect();
 scene->startServoLoop();
 gstPoint pos; // holds current PHANToM position
 double u, v, w;
 sock_init();
 while (!scene->getDoneServoLoop()) {
   pos = PHANToM->getPosition_WC();
   // send position to client, SCIRun
   write_triple(pos.x(), pos.y(), pos.z());
   // read resulting force from SCIRun
   if (read_triple(&u, &v, &w) == 0) {
    // copy to readable location
    scirun_force = gstVector(u, v, w);
    // the next time calcEffectForce() happens,
    // it will see this new force
   }
   else {
    sock_exit();
 }}
```

```
// quit my force input
PHANToM->stopEffect();}
```

### 2.6.5 Vista und Viracocha

Vista ist eine multiplattformfähige C++-*Bibliothek*, welche auf HP-UX, Linux, MS Windows, SGI Irix und Sun Solaris läuft (Gerndt et al. 2004). Die Bibliothek ermöglicht das Modifizieren und Darstellen von Szenengraphen. Es nutzt die Visualisierungs-Pipeline des Visualization Toolkit (VTK) (Schroeder und Lorensen 1996; Law et al. 1999). Zum Basissystem gibt es die Erweiterungen Vista Flow-Lib und Viracocha. Die Flow-Lib dient dem Extrahieren von Strömungsvisualisierungen. Viracocha dient der Parallelisierung und dem optimierten Rendering. Vista Flow-Lib kapselt

1. die Rendering- und Zeitverwaltung,
2. die Verwaltung *multi-modaler* Nutzungsoberflächen und
3. den Parallelisierungs-Adapter.

Komponente 1 enthält einen Visualisierungs-Controller, Visualisierungsobjekte, Zeitabbilder und Ressourcen-Verwalter (Schirski et al. 2003, S. 80). Der Controller verwaltet die Geometriedaten. Der Zeitabbilder bildet die normalisierte Visualisierungszeit auf die absolute Simulationszeit ab, damit die Geometrien eindeutig mit den Zeitschritten assoziiert werden. Der Ressourcen-Verwalter alloziert, deallozierd und kontrolliert Ressourcen nach Maßgabe von den anderen Komponenten.
Komponente 2 ist für den Betrieb mit mehreren Arten von Hardware ausgelegt, wie das bei Amira besprochen wurde.
Komponente 3 und Viracocha bestehen aus drei funktionalen Schichten:

- Kommunikation,
- Scheduling und
- Extraktion.

Der Kommunikator kapselt den Zugriff auf das *Message Passing Interface* (*MPI*). Das Scheduling besteht aus dem eigentlichen Scheduler zum Weiterleiten von Datenextraktions-Anforderungen, einem Server, der den Scheduler zum Visualisierungs-Host hin repräsentiert, den Proxies und den Arbeitsknoten. Proxies lesen auf Anforderung durch Arbeitsknoten Daten von Festplatte und reichen die Daten an die Arbeitsknoten weiter, welche die Geometriedaten aus den Simulationsergebnissen generieren. Sobald ein Arbeitsknoten fertig ist, sendet er die Geometriedaten an den Visualisierungs-Host. Dieses Verhalten ist an keine *Barrier* gekoppelt. Das heißt, dass Teile einer Szene bereits dargestellt werden, während andere Teile noch von Arbeitsknoten berechnet werden. Weiterhin ist erwähnenswert, dass die Software dynamisch parallel arbeitet.
Zur Datenreduktion der gelesenen Ressourcen werden folgende Techniken angeboten:

- Es werden platzsparende Metaphern verwendet.
- Das Rendern wird mithilfe von Billboard-Techniken optimiert, die, ähnlich einer Kulisse, eine Geometrie durch ein texturiertes Polygon ersetzen.
- Das Filtern, Abbilden und Rendern von Äquipotenzialflächen wird optimiert.

Letzteres funktioniert so, dass alle Blöcke in aufsteigender Entfernung zum Betrachter sortiert und danach klassifiziert werden, ob sie leer sein werden oder nicht. Aus den vollen Blöcken werden die Dreiecksnetze erzeugt und an den Renderer gesendet, sobald deren Datengröße einen Wert überschreitet.

### 2.6.6 Volumenrenderer

Engel et al. (1999), Bethel et al. (2000), Park et al. (2000) und Rezk-Salama et al. (2000) spezifizieren Volumenrenderer. Das Volumenrendering wird durch Bildinterpolation nachgestellt, welche visuelle Artefakte erzeugt. Weiterhin erlaubt Volumenrendering nur wenige Änderungen der Darstellung.
Der SGI Viz-Server von SGI (2002) unterstützt das entfernte Rendern mit OpenGL, indem er auf einem Renderer ein Bild erzeugt, es an entfernte Rechner sendet und dort darstellt. Alle Rechner erhalten zeitgleich dasselbe Bild. Der Ansatz ist für das Rendering mit geringen Netzwerk-Bandbreiten geeignet oder wenn die Rendering-Software nur zentral einsetzbar ist.
Chromium ist ein System, das ausgeführte OpenGL-Befehle protokolliert, das Protokoll an entfernte Rechner sendet und dort die Befehle wiederholt (Humphreys et al. 2002). Das System komplementiert den Viz-Server.

### 2.6.7 Prototypen zur Haptisierung von Volumen

**Six Degree-of-Freedom Haptic Display of Polygonal Models** Gregory et al. (2000) spezifizieren einen haptischen Renderer, bei dem die Geschwindigkeit des End-Effektors am haptischen Gerät gemessen wird. Zwischen dem virtuellen Objekt, das sich mit dem End-Effektor bewegt, und den anderen virtuellen Objekten ermittelt das System die die Geometrien umgebenden Zellen. Anhand einer Kohärenzsuche bestimmt es die am nächsten beieinanderliegenden Polygonpaare. Das System bestimmt aus ihnen die wichtigsten Kontaktpunkte, Kontaktnormalen und Eindringungstiefen, und es generiert daraus die Argumente zum Aufruf der Masse-und-Feder-Funktion. Um die Latenz beim Zusammenstellen der Paare zu senken, sagt das System anhand der Position des End-Effektors und dessen Geschwindigkeit die Kollisionsparameter durch lineare Extrapolation über die Zeit voraus.
Wenn sich zwei Zellen erstmalig überlappen, kann nicht kohärenzgetrieben nach Polygon-Paaren gesucht werden. Stattdessen wird nach Polygonen gesucht, indem nur die äußersten Eckpunkte in einer Zelle untersucht werden, weil diese mit höherer Wahrscheinlichkeit in eine Kollision involviert sind als die inneren Eckpunkte. Eine weitere Optimierung ergibt sich daraus, jene Eckpunkte priorisiert in die Kollisionserkennung einzugeben, die am nächsten entlang des Vektors zwischen den Mittelpunkten zweier überlappender Zellen liegen.

**Shock and Vortex Visualization using a Combined Visual/Haptic Interface** Lawrence et al. (2000) spezifizieren die Entwicklung und Nutzung einer grafischen und einer haptischen Komponente zur wissenschaftlichen Perzeptualisierung. Die grafische Komponente ist ein Rechner auf dem AVS/Express läuft. Das haptische Gerät besteht aus fünf Aktuatoren, die an einem hexagonalen Ring befestigt sind. Die Aktuatoren steuern fünf Ruten, die an einen End-Effektor führen, der mit der Hand bewegt wird. Man sieht einen Nutzer mit dem Gerät auf Bild 2.7.
Der Vorverarbeiter teilt zum Auffinden die Daten in Zellen ein und weist ihnen Koordinaten zu. Pro Rendering-Durchlauf findet der Renderer jene Zelle, welche die Position

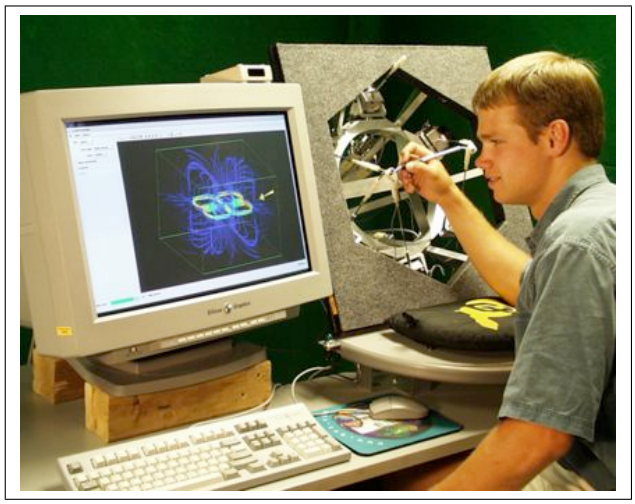

***Bild 2.7:*** *Perzeptualisierungs-System von Lawrence et al. (2000)*

des End-Effektors umgibt. In der Zelle interpoliert der Renderer die Position, um daraus die Koeffizienten zu gewinnen, nach denen er die Daten in den umliegenden Zellen gewichtet. Er interpoliert aus den gewichteten Daten ein Datum, das er dem Nutzer in Form eines haptischen Impulses gibt.
Ein Beispiel zur Anwendung ist das Perzeptualisieren von Schockwellen, indem in den Daten an der Position des End-Effektors der Dichtegradient $p$ berechnet wird. Liegt dieser oberhalb eines Werts $\epsilon$, wird eine Kraft $F_n$ in Gegenrichtung des lokalen Gradientenvektors zugeschaltet. Der Betrag ist die Schätzung der Eindringungstiefe in die implizite Oberfläche. Seien $P_n$ die Position des End-Effektors zur Zeit $n$, $D_n = P_n - P_{n-1}$ und $e_n$ die Projektion von $D_n$ auf die Richtung $p_{n-1}$ von $P_{n-1}$, so folgt die Bestimmung der Kraft der Gleichung

$$F_n = -g \bigtriangledown p_n \left| \sum_{i=0}^{n} \vec{e_i} \right| \tag{2.22}$$

## 2.7 Abschließende Bemerkungen

Ich habe bekannte Ansätze zur Perzeptualisierung (Card et al. 2001), zur Datenreduktion (Salomon 2004) und zur Parallelisierung diskutiert (El-Rewini und Lewis 1998). Aus einigen Teilen werde ich ein neues System zusammensetzen und in bestehende Software integrieren. Hierzu analysiere ich die Software in Kap. 3.

# Kapitel 3

# Technische Analyse

*„Worin, obwohl das Kapitel kurz ist, der Greis Alinardus recht interessante Dinge über das Labyrinth andeutet und über die Art, wie man hineingelangt."* Umberto Eco in (Eco 1984, S. 199)

In dem Kapitel motiviere ich den Einsatz der Visualisierungs-Software DSVR-1 und lege dar, dass die funktionalen Erweiterungen mit dem Erreichen meiner Arbeitsziele zusammenhängen. Zum Schluß schätze ich die Datenmengen, die bei der Nutzung der Software ohne Datenreduktion anfallen.

## 3.1 Nutzung und Schwächen

Die Anwendungsfälle sind das Explorieren und das Präsentieren. Beim Explorieren sucht der Wissenschaftler Zusammenhänge. Beim Präsentieren will er die gefundenen Zusammenhänge seinen Kollegen und Studenten vermitteln. In beiden Anwendungsfällen navigiert er durch die Geometriedaten, aber nur im ersten Fall wird er sich mit der Datenquelle verbinden. Im zweiten Fall wird er gespeicherte Geometriedaten abrufen.
Das parallele Reduzieren und Verteilen der Daten ist wichtig, weil sie bis zu mehrere Terabyte umfassen. Allerdings kann sich der Designer eines Perzeptualisierungs-Systems nur auf das Generieren und Modifizieren der Geometriedaten und den davon abhängigen Daten beschränken. Zwar wäre die Reduzierung der Simulationsdatenmenge vielversprechender, weil sie deutlich größer ist, aber sie ist außerhalb der Verwaltung des Perzeptualisierungs-Systems („read-only"). Die Simulationsdaten werden direkt nach der Analyse gelöscht oder gespeichert. Falls sie gespeichert werden, könnten sie nachbearbeitet werden. Somit wird nur für den aktuellen Zeitschritt der Simulation sichergestellt, dass die unveränderten Rohdaten zur Perzeptualisierung herangezogen werden können.
Um aus den Simulationsdaten Geometriedaten zu machen, haben wir in Kap. 2 bereits Systeme kennengelernt. Ihre Schwächen fasse ich zusammen:

- **Monolithische Hardware** Nur ein Rechner perzeptualisiert Daten: Alle Schritte in der Perzeptualisierungs-Pipeline werden von einem Rechner ausgeführt. Das ist ineffizient, weil eine einzige Hardware-Architektur nicht für alle Stufen der Perzeptualisierung gleich geeignet ist. Zudem impliziert es, dass jeder Nutzer Zugriff auf ein solches System hätte.

- **Keine Parallelisierung** Die Komponenten werden nicht parallel ausgeführt. Es ist dadurch schwierig, die Arbeitslast der Hardware auszugleichen, um Überbeanspruchung oder Leerlaufzeit zu minimieren.

- **Keine effiziente Option für Filtern und Abbilden** Der Nutzer kann auf keine Bibliothek zurückgreifen, um Daten zu filtern und abzubilden.

- **Extreme Bandbreitennutzung** Das System folgt einer Alles-Oder-Nichts-Strategie: Es kann nur einen kompletten Satz an Simulationsdaten perzeptualisieren. Dies erhöht die Last, Daten zu speichern, falls das Resultat mehrere hundert Gigabyte übersteigt. Zumindest werden die Interaktivität und Antwortzeit beeinträchtigt. Schlimmstenfalls übersteigt die Datenmenge die Kapazität der Hardware (Prozessoren, Hauptspeicher, Platten und Netzwerk).

- **Begrenzte Performance** Die Algorithmen sind nicht auf Geschwindigkeit optimiert. Die Datenstrukturen enthalten deutlichen Mehraufwand, zum Beispiel *ASCII*-Repräsentationen oder Baumstrukturen, anstelle von binär kodierten Listen. Dies erschwert das effiziente Bearbeiten von Volumen.

- **Batch-Perzeptualisierung** Simulation und Perzeptualisierung sind separate Prozesse. Der Nutzer hat keine interaktive, intuitive Methode zur Hand, um die Simulation als Reaktion auf die Ausgaben zu steuern.

- **Fixiertes Rendering** Falls das System nur Bilder ausgibt, ist das Rendering auf die während des Simulationslaufs aktiven Einstellungen fixiert.

- **Wenige Metaphern** Einige Systeme implementieren wenige Metaphern, zum Beispiel nur 2-D-Schnittflächen. Dies begrenzt die Möglichkeiten des effektiven Perzeptualisierens.

- **Betrieb für einzelne Nutzer** Die Kollaboration zwischen Nutzern ist begrenzt, so dass die Anzahl der Nutzer auf die Anzahl jener beschränkt ist, die sich um einen Rechner physikalisch versammeln oder auf ergänzende Medien zurückgreifen.

- **Optimierung für grafische Ausgaben** Die Systeme sind so optimiert, dass das Nachrüsten um multi-modale Ausgaben erschwert wird.

## 3.2 Entwurf und Implementierung von DSVR-1

### 3.2.1 Übersicht

DSVR-1 vermeidet die o.g. Schwächen. Es ist ein parallelisiertes, netzwerkverteiltes Visualisierungs-System für das interaktive Steuern von numerischen Simulationen und das Betrachten ihrer Ergebnisse (Olbrich 2000). Das System visualisiert die Ergebnisse parallel zum Fortschritt, indem es unmittelbar aus der Quelle die Geometriedaten erzeugt und verteilt.
DSVRs Vorteil ist, dass dessen Architektur eher einem Rahmenwerk denn einem geschlossenen System gleicht. Die Komponenten werden unabhängig voneinander optimiert:

- **Verteilte Hardware** Die Visualisierungs-Pipeline ist in Form von lose gekoppelten Komponenten implementiert, die zwischen heterogenen Rechnern verteilt werden.
- **Parallelisierung** Die Komponenten arbeiten datenparallel durch Gebietszerlegung und funktionsparallel durch Pipelining.
- **API für Filtern und Abbilden** Die Software hat eine Bibliothek, die das Filtern und Abbilden zerlegt und getrennt.
- **Inkrementeller Transfer von Visualisierungsdaten** Physikalische Modelle sind zeitabhängig. Das System kann diese Eigenschaft derart nutzen, dass es die Geometriedaten der Volumen nacheinander an die Render-Clients transferiert. Der Transfer von Teil-Volumen innerhalb eines Zeitschritts erhöht den Verwaltungsaufwand und ist nicht zwingend nützlich für Animationen.
- **Maximierte Performance** Alle Algorithmen und Protokolle sind optimiert. ASCII-basierte Formate müssen in Binärformate umgewandelt werden.
- **Dialogbetriebene Visualisierung** Das System arbeitet im Batch- und im Dialogbetrieb. Interaktives Steuern impliziert, dass Nutzer mit dem Generator im Dialogbetrieb kommunizieren. Dies ist jedoch nicht immer möglich, weil manche Generatoren nicht im Dialogbetrieb gefahren werden dürfen.
- **Flexibles Rendering** Das 3-D-Streaming erlaubt den Nutzern das Explorieren von Volumen unter verschiedenen Blickpunkten und Rendering-Stilen, indem erst der Empfänger den Inhalt rendert.
- **Viele Metaphern** Das System unterstützt die wichtigsten Datentopologien und die *ISO*-standardisierten Spezifikationen von Grafikprimitiven aus *VRML-1* (Web 3D Consortium 2004). Dessen Notation ist stabil und vielseitig einsetzbar.
- **Kollaboratives Arbeiten** Die Software hilft Nutzern, kollaborative Umgebungen zu ergänzen (Wood et al. 1997; Jensen et al. 2003, 2004a,b).
- **Multi-modale Ausgaben** Die Treiber zum Steuern der multi-modalen Ausgabegeräte werden im 3-D-Renderer aufgerufen.

### 3.2.2 Entwurf

#### Statik

Der Entwurf von DSVR folgt einer dreiteiligen Client/Server-Architektur (Bild 3.1):

- Generator-Client
- Streaming-Server
- Render-Client

Die Komponenten tauschen Geometriedaten und Protokollbefehle über Kanäle aus (Tab. 3.1). C2C erweitert die Software um Kollaboration und C2G unterstützt das Steuern von Simulationen.

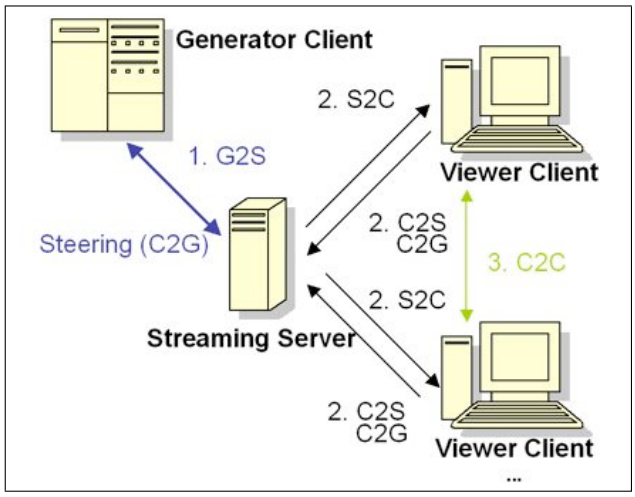


***Bild 3.1:*** *Exemplarische Verteilung von* DSVR-1 *nach Jensen et al. (2002)*

| Von | Bis | Kürzel | Kanal |
|---|---|---|---|
| Generator-Client | Streaming-Server | G2S | 1 |
| Streaming-Server | Rendering-Client | S2C | 2 |
| Rendering-Client | Streaming-Server | C2S | 2 |
| Rendering-Client | Rendering-Client | C2C | 3 |

***Tabelle 3.1:*** *Kommunikationskanäle in* DSVR *nach Jensen et al. (2002)*

### Dynamik

DSVR liest die Rohdaten von regulär und irregulär angeordneten Gittern, die im kartesischen Koordinatensystem vorliegen. Die Software filtert und bildet die Daten dort ab, wo sie generiert werden. Damit wird das Transferieren von Simulationsdaten überflüssig, was den Vorteil hat, die größten Datenmengen im Vorfeld entfernen zu können. DSVR sendet die Geometriedaten an eine Sammelstelle, die über Bus-Systeme oder Local Area Networks (*LAN*) erreicht wird. Dies erfordert, abgesehen vom Betrieb durch mehrere Prozessoren und effiziente Datentransportwege, keine spezialisierte Hardware. Der Nachteil ist die erhöhte Last auf dem Datengenerator.
Der Generator-Client sendet die gesammelten Geometriedaten über eine TCP/IP-Verbindung an einen Server, wo sie gespeichert oder an Render-Clients gesendet werden. Dies geschieht zeitverschränkt mit der Datengenerierung ohne Komprimierung. Der Vorteil ist die Einfachheit und Flexibilität des Verfahrens. Der Nachteil ist, dass die Geometriedaten Redundanzen aufweisen.
Bei DSVR kommt beim Abspielen das sogenannte Push-Verfahren zum Einsatz, d. h. dass der Server bis auf Widerruf Geometriedaten in einem vorher vereinbarten Takt sendet. Beim Pull-Verfahren ist hingegen der Client für das Fordern von Daten zuständig. Das Push-Verfahren ist primär dafür geeignet, Animationen abzuspielen.
Das nachgelagerte Rendern auf den Clients

- erhöht die Wahl der Rendering-Optionen,

- unterstützt das Navigieren in den Geometriedaten mit einem Zugriff auf den Server pro Zeitschritt und
- verschiebt die Rendering-Last vom Generator auf die anderen Clients.

Obwohl der Generator rendern könnte, wäre das weniger effizient als einen Rechner mit Grafikkarte zu nutzen. Auf der anderen Seite ist ein Rechner mit Grafikkarte nicht entworfen, um numerische Simulationen in hoher Genauigkeit durchzuführen, obwohl es mittlerweile Ansätze dafür gibt, die höhere Rechenkapazität von GPUs zu nutzen. Dies arbeitet die Visualisierungs-Pipeline effizient ab, aber hat den Nachteil, dass die Datenmenge durch den Grafikkartenspeicher beschränkt wird. Das Zusammenschalten von Rechnern kompensiert den Nachteil (Strengert et al. 2005). Jedoch verbleibt die Berechnung nur dann auf den Grafikkarten, wenn keine alternativen Rendering-Modi, zum Beispiel zur Ausgabe an haptische Geräte, genutzt werden.

### Datenformate

Das Streaming verbindet die Komponenten. Das Streaming-Format muß die Geometrien spezifizieren, die nach dem Abbilden entstehen. Hierfür gibt es das eingangs erwähnte VRML-Format.
In VRML werden eine oder mehrere Geometrien in einen Szenengraphen eingepaßt. Das Senden und Umwandeln von Szenengraphen in Renderings-Befehle ist ineffizient, weil der Szenengraph in ASCII beschrieben wird und weil der Graph vor dem Rendern geparst werden muß. DSVR nutzt daher das Format DVR, das den serialisierten VRML-Szenengraphen binär repräsentiert.
Der Renderer von DSVR übersetzt die Information vom DVR-Format in OpenGL-Befehle. Vergrößerungen, Drehungen und Verschiebungen werden vorberechnet und beim Abbilden und Rendern relativ zum Blickwinkel eingesetzt.
Eine DVR-Datei besteht aus *Protocol Data Units* (*PDUs*), die den Szenengraphen beschreiben (Bild 3.2):

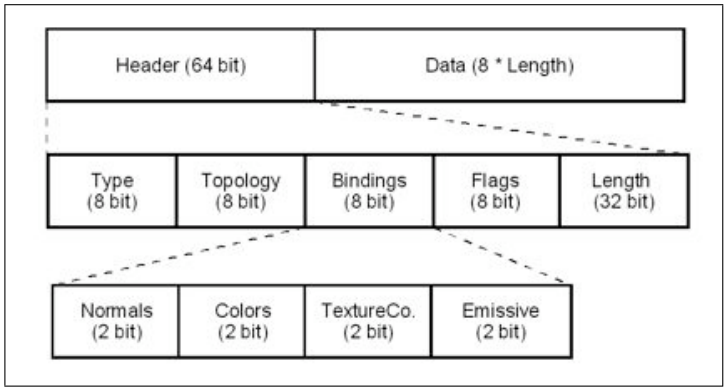


***Bild 3.2:*** *Struktur einer PDU nach (Olbrich 2000, Bild 24)*

1. `Type`: Identifiziert genau eines von
   - `Geometry` (VRML-1-kompatibel)

- `Material` (Shading, Texture, Transparency)
- `Text` (URL)
- `Transformation`
- `Camera`
- `Light`

2. `Topology`: Unterklasse von `Geometry` die durch `Type` definiert wurde
3. `Bindings`: Zusätzliche Parameter
4. `Flags`: Kontrollflags für die Definition der Byte-Reihenfolge von Datentypen
5. `Length`: Anzahl nachfolgender PDUs
6. Nachfolgende PDUs

Das Binärformat wird schnell erzeugt und gelesen. Es reduziert den Netzwerkverkehr um einen konstanten Faktor.
Eine Animation wird als Referenz auf Folgen von DVR-Dateien, auch *Frames* genannt, aufgefaßt. Der Stream besteht aus Szenen, und jede Szene besteht aus Frames.

#### Simultanes Abspielen

Um Szenen gemeinsam in einem Stream abzuspielen, bedarf es in DSVR-1 des manuellen Editierens von sogenannten DVRS-Dateien, welche die Streams als Verbände von Szenenfolgen definieren. Wenn die Szenen live generiert werden, muß man beachten, dass DSVR-1 es auf Seiten der Generatoren nicht erlaubt, mehr als einen Frame pro Generator zu einem Zeitpunkt zu erzeugen und zu versenden. Das System sendet die Frames somit nur nacheinander oder verschränkt. Weiterhin fehlt das synchrone Abspielen der Szenen.

#### Parallelisierung

**Datenparallelität** Beim Aufteilen auf Prozessoren kommt es darauf an, Datenaustausche zwischen Prozessen zu minimieren. Weniger ist die Menge der Datenaustausche entscheidend für die Latenz, als die Anzahl, weil der Mehraufwand zur Initiierung eines Datenaustausches vergleichsweise hoch ist. Bei Shared-Memory-Systemen fällt dies weniger ins Gewicht als bei Distributed-Memory-Systemen, wie sie heutzutage überwiegend in Rechenclustern vorgefunden werden. Somit strebt man in DSVR-1 an, zu Anfang keine Datenaustausche zuzulassen, sondern erst die Abbildungen zu sammeln. Dies impliziert, dass sich die Lastenausgleiche ohne Zutun über mehrere Zeitschritte einstellen, bzw. die Datenquelle dementsprechend für jeden Zeitschritt vom Nutzer vorzerlegt wurde. Jedoch arbeiten viele Abbilder nur auf teilüberlappenden Gebieten korrekt. Dem trägt DSVR-1 Rechnung, indem es teilüberlappende Gebiete akzeptiert (Bild 3.3).

**Funktionsparallelität** Die Funktionsparallelität ergibt sich durch das zeitverschränkte Operieren auf den Daten in der Visualisierungs-Pipeline, wie ich es in Kap. 2.5 vorgestellt hatte.

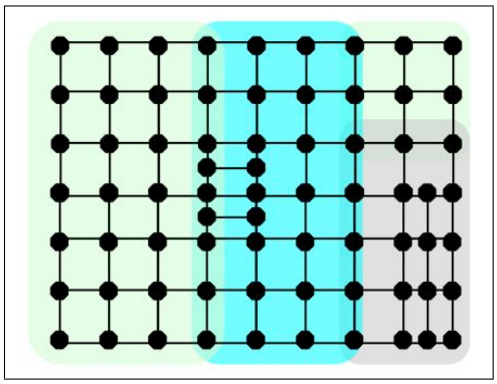

***Bild 3.3:*** *Beispiel für eine 2-D-Gebietszerlegung auf farbig markierte Prozessoren*

### 3.2.3 Implementierung

**Generator-Client**

Die Bibliothek zum Filtern und Abbilden wird in Portable-C und MPI programmiert. Der erste Algorithmus verteilt das Filtern und Abbilden auf alle Prozessoren des Generators:

```
Algorithm 1() {
  for each timestep {
    for process n of N in parallel {
      simulate nth partition
      filter nth partition
      map nth partition
        to nth part of visualization}
    merge N parts of visualization
    forward visualization to server}}
```

Der zweite Algorithmus trennt einen dedizierten Sammel-Prozeß ab, um die Arbeitslast auszugleichen, und um somit einen besseren Datendurchsatz zu erzielen (Jensen et al. 2002).

```
Algorithm 2() { // 'split (pipelining)'
  for each timestep {
    do block 1..2 in parallel {
      block 1:  // PU 1..n
       for process n of N in parallel {
         simulate nth partition
         filter nth partition
         map nth partition
           to nth part of visualization}
      block 2: // PU 0
       merge N parts of visualization
       forward visualization to server}}}
```

#### Streaming-Server

Der Streaming-Server erzeugt für jede Anfrage nach DVR-Dateien einen Prozeß. Jeder Prozeß liest und sendet sie mit einer festen oder einer maximalen Update-Rate (Updates/sec, ups). Bei der festen Rate werden Frames ausgelassen, um die Rate zu halten. Bei der maximalen Rate werden garantiert alle Frames ausgespielt, jedoch nicht notwendigerweise in der maximalen Rate.

#### Render-Client

Die Render-Clients sind Arbeitsplatzrechner, die über ein Browser-Plugin Streams empfangen. Zum Anzeigen schließt man an den Renderer die entsprechenden Ausgabegeräte an (Jensen et al. 2002). Die Frame-Rate (Frames/sec, fps) spezifiziert, wie oft der Renderer Abbildungen an das Ausgabegerät sendet. Die Frame-Rate übersteigt die Update-Rate.

### 3.2.4 Bandbreitennutzung

Volumendatenmengen sind zwischen 8 MByte und 1 GByte groß. Um eine hypothetische Folge von unreduzierten Volumen zu perzeptualisieren, braucht man einen geschätzten, minimalen Datendurchsatz von $8$ MByte $\cdot$ $10$ Frames/sec $=$ $80$ MByte/sec $=$ $640$ Mbit/sec.
Die aus den Volumendaten erzeugten Polygondaten sind kleiner und erfordern weniger Datendurchsatz. Die Ausgabegröße liegt anwendungsabhängig zwischen 1 und 5 MByte, erreicht in pathologischen Fällen aber auch die Größe der Volumendaten. Eine hypothetische Folge von Äquipotenzialflächen erfordert einen minimalen Datendurchsatz von $1$ MByte $\cdot$ $10$ Frames/sec $=$ $10$ MByte/sec $=$ $80$ Mbit/sec.

## 3.3 Gap-Analyse

Bezogen auf die Funktionalität und Performance hat DSVR-1 Schwächen, die teilweise nicht in anderen Systemen zu finden sind:

**A1)** Kein direktes Volumenrendern von 3-D-Texturen

**A2)** Hoher Bedarf an Prozessorzeit und Speicher für den Transfer und das Rendern von unreduzierten Geometriedaten

**A3)** Kein haptisches Rendern

**A4)** Kein synchrones Abspielen von parallel zu betrachtenden Szenen

Die Reduktion ist am wichtigsten (A2), weil ohne sie weder das bestehende System, noch das erweiterte System (A1, A3), effizient genutzt werden. Auf der anderen Seite wird die Art des Renderns bestimmen, wie eine Datenreduktion vorgenommen wird. Die letzte Anforderung (A4) könnte indirekt auf die Datenreduktion Einfluß haben, wenn man sich entschlösse, mehrere Szenen während eines Zeitschritts abhängig voneinander zu reduzieren.

## 3.4 Abschließende Bemerkungen

Ich empfehle, DSVR-1 zur Perzeptualisierungs-Software zu erweitern und die Performance von DSVR-1 zu verbessern. Den Entwurf der Erweiterungen spezifiziere ich in Kap. 4–5.

# Kapitel 4

# DSVR-2 – Systementwurf

*„Designing an object to be simple and clear takes at least twice as long as the usual way. It requires concentration at the outset on how a clear and simple system would work, followed by the steps required to make it come out that way."* (T. H. Nelson in Nelson (1978))

Olbrich (2000) beschreibt den kompletten Entwurf von DSVR-1 nach der Methode des strukturierten Designs (SD) und der Zerlegung in endliche Automaten (EA). Ich übersetze die für meine Arbeit wichtigen Teile in objektorientiertes Design (OOD) und diskutiere Erweiterungen, die mit dem Zusatz „(neu)" gekennzeichnet sind. Den Entwurf der Methoden der Codecs spezifiziere ich in Kap. 5.

## 4.1 Funktionalität

Die neue Funktionalität gliedere ich nach den Anforderungen aus Kap. 3.3:

**F1)** Direktes Volumenrendern von 3-D-Texturen

**F2)** Reduzieren von Geometriedaten

**F3)** Haptisches Rendern

**F4)** Synchrones Abspielen von parallel zu betrachtenden Szenen

Ich diskutiere pro Funktionalität mehrere Alternativen der Modifikation.

## 4.2 Komponenten und Verteilung

Die Komponenten sind die Generator- und Render-Clients, sowie der Streaming-Server. Die Beobachtungen aus Kap. 2.3.6 zeigen, dass die Erweiterungen F1–4 vom bestehenden System aufgenommen, und durch die Mehrfachinstanzierung von Komponenten unterstützt werden. Zur Einführung neuer Komponenten besteht demnach keine Veranlassung. Auch die Verteilung lasse ich unverändert.

## 4.3 Klassen

Jede Operation auf den Daten während der Perzeptualisierung ergibt sich aus der Stufe der Pipeline, der Modalität und der Metapher, die aus den Daten entsteht. Es ist daher falsch, jede Stufe der Pipeline als Klasse zu modellieren, weil

- sie wenige Methoden aufwiese und
- jede Methode eine Fallunterscheidung nach der Geometrie beansprüche.

Letzteres könnte behoben werden, indem geeignete Operationen an die, als Klasse modellierten, Geometrietypen gekoppelt würden. Das heißt aber auch, dass man ohne nennenswerten Verlust auf die explizite Modellierung der Stufen verzichten kann. In diesem Fall wechselt kein Objekt seine eingangs gewählte Klasse, und jede Klasse behält genügend Methoden (Bild 4.1). Zur Detailbestimmung spezifiziere ich in Tab. 4.1 die Class-Responsibility-Collaboration, kurz CRC (Leffingwell und Widrig 1999, S. 144).

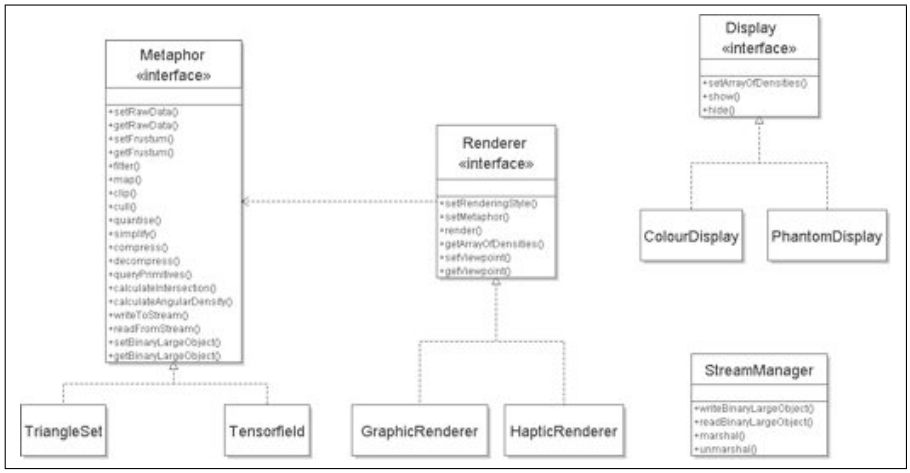


***Bild 4.1:*** *Klassendiagramm (Ausschnitt aus* DSVR-2*)*

### 4.3.1 Metaphern

Die Klasse `Metaphor` ist die Schnittstelle der Metaphern, welche beim Lauf der Daten durch die Pipeline an Form gewinnen. Die Unterklassen `TriangleSet` und `Tensorfield` (neu) repräsentieren Dreiecksnetze und Volumen.

### 4.3.2 Rendern und Anzeigen

Es gibt mehr als eine Art des Renderns, und die hängt gleichsam von der Metapher, der Modalität und dem Rendering-Stil ab. Grafik und Haptik haben gemeinsam, dass sie, ausgehend von einem Frustum, die erfaßbaren Dreiecke und Voxel enumerieren und deren Attribute mittels eines Rendering-Stils zeigen. Die Klasse `Renderer` ist die Schnittstelle des Rendering-Stils, bezogen auf eine Modalität, welche der Nutzer

bestimmt. Die Klassen `GraphicRenderer` und `HapticRenderer` (neu) implementieren die Schnittstelle `Renderer`.
Die Klasse `Display` ist die Schnittstelle, welche die Dichte als Farbe, Transparenz oder Kraft an ein dafür geeignetes Gerät weitergibt. Die Klassen `ColourDisplay` und `PhantomDisplay` (neu) implementieren die Schnittstelle `Display`.

### 4.3.3 Hilfsklassen

Der `StreamManager` abstrahiert die Verwaltung der Streams. Pro Stream existiert eine Instanz.

## 4.4 Aktivitäten

### 4.4.1 F1 – Direktes Volumenrendern

Hier sieht man, wie ein Volumen erzeugt, in einen Stream überführt und gesendet wird. Der Empfänger wandelt den Stream in das Volumen zurück und ruft die Befehle zum Rendern des Volumens auf.

```
Generator-Client {
 extern float source[]
 extern float extent[], depth, constant, factor
 ...
 ubyte blob[]
 StreamManager streamout("ceci.n'est-pas-une-url.eu/f1.dvrs")
 volume = new Tensorfield
 volume.setRawData(source)
 volume.filter(extent, depth, constant, factor)
 volume.map()
 volume.writeToStream(blob)
 streamout.marshal(blob)
 streamout.writeBinaryLargeObject(1, blob)
}
Render-Client {
 extern enumerator ep, ei
 extern ColourDisplay colour3DProjection
 ...
 ubyte densities[]
 ubyte blob[]
 GraphicRenderer renderer
 StreamManager streamin("ceci.n'est-pas-une-url.eu/f1.dvrs")
 streamin.readBinaryLargeObject(1, blob)
 streamin.unmarshal(blob)
 Metaphor metaphor.readFromStream(blob)
 metaphor.setFrustum(renderer.getViewpoint())
 ep = metaphor.queryPrimitives()
 ei = metaphor.calculateIntersection(ep)
 densities[] = metaphor.calculateAngularDensity(ei)
 colour3DProjection.setArrayOfDensities("left", densities)
 metaphor.setFrustum(renderer.getViewpoint())
```

| Name | Kollaboratoren |
|---|---|
| `Metaphor` | `Renderer` |
| **Methoden** | |
| `setRawData()` | `setFrustum()` |
| `getRawData()` | `getFrustum()` |
| `filter()` | `map()` |
| `clip()` | `cull()` |
| `quantise()` | `simplify()` |
| `compress()` | `decompress()` |
| `queryPrimitives()` | `calculateIntersection()` |
| `calculateAngularDensity()` | `writeToStream()` |
| `readFromStream()` | `setBinaryLargeObject()` |
| `getBinaryLargeObject()` | |

| Name | Kollaboratoren |
|---|---|
| `Renderer` | `Metaphor, Display` |
| **Methoden** | |
| `setRenderingStile()` | `setMetaphor()` |
| `render()` | `getArrayOfDensities()` |
| `setViewpoint()` | `getViewpoint()` |

| Name | Kollaboratoren |
|---|---|
| `Display` | `Renderer` |
| **Methoden** | |
| `show()` | `setArrayOfDensities()` |
| `hide()` | |

| Name | Kollaboratoren |
|---|---|
| `StreamManager` | `Metaphor` |
| **Methoden** | |
| `writeBinaryLargeObject()` | `readBinaryLargeObject()` |
| `marshal()` | `unmarshal()` |

***Tabelle 4.1:*** *CRC-Karten (Ausschnitt aus* DSVR-2*)*

```
 ep = metaphor.queryPrimitives()
 ei = metaphor.calculateIntersection(ep)
 densities[] = metaphor.calculateAngularDensity(ei)
 colour3DProjection.setArrayOfDensities("right", densities)
 colour3DProjection.show()
}
```

### 4.4.2 F2 – Reduzieren von Geometriedaten

Die Einbettung der Datenreduktion folgt der Ordnung aus Kap. 2. Das Kap. 5 wird den Feinentwurf der Methoden spezifizieren.

```
Generator-Client {
 ...
 Metaphor metaphor
 ...
 metaphor.map()
 metaphor.clip()
 metaphor.cull()
 metaphor.quantise()
 metaphor.simplify()
 metaphor.compress()
 blob = metaphor.getBinaryLargeObject()
 ...}
Render-Client {
 ...
 Metaphor metaphor.readFromStream(blob)
 metaphor.decompress()
 ...}
```

### 4.4.3 F3 – Haptisches Rendern

Haptisches Rendern bezieht sich auf die irgendwie geartete Darstellung von Dreiecksnetzen oder 3-D-Texturen auf einem haptischen Gerät. Das Vorbild für die Einbindung ist Scirun (Kap. 2.6.4).

```
...
Render-Client {
 extern enumerator ep, ei
 extern PhantomDisplay phantom
 ...
 ubyte densities[]
 ubyte blob[]
 HapticRenderer renderer
 StreamManager streamin("ceci.n'est-pas-une-url.eu/f3.dvrs")
 streamin.readBinaryLargeObject(1, blob)
 streamin.unmarshal(blob)
 Metaphor metaphor.readFromStream(blob)
 metaphor.setFrustum(renderer.getViewpoint())
 ep = metaphor.queryPrimitives()
 ei = metaphor.calculateIntersection(ep) // from frustum
```

```
 densities[] = metaphor.calculateAngularDensity(ei) //contacts
 phantom.setArrayOfDensities(densities) // penetration depths
 // for volumes, the following style would be: "f=-k*v"
 phantom.setRenderingStyle("f=1/v*x") // mass + spring
 phantom.show() // give interpolated force in each direction
}
```

### 4.4.4 F4 – Synchrones Abspielen

DSVR-2 soll die Frames verschränkt oder parallel abspielen. Letzteres erfordert das Anlegen eines Pools von Prozessen, in dem der Generator-Client die Szenenbestandteile sammelt und simultan an Streaming-Server sendet. Diese können die Prozesse eines oder mehrerer Server sein, je nachdem wieviele in einer DVRS-Datei referenziert werden. Anhang A gibt die Details.

Im Streaming-Server habe ich Code eingefügt, der die Server-seitigen Abspielprozesse so synchronisiert, dass kein Prozeß neue Dateien abspielt, bevor nicht alle Peer-Prozesse ihre Dateien mit derselben *Frame-Nummer* abgespielt haben. Dies stellt sicher, dass die Frame-Nummern nicht allmählich auseinanderlaufen. Ich notiere in Anhang A den Pseudo-Code für das dafür einzusetzende bekannte 2-Wege-Handshake.

## 4.5 Zustandsdiagramm von Metaphor

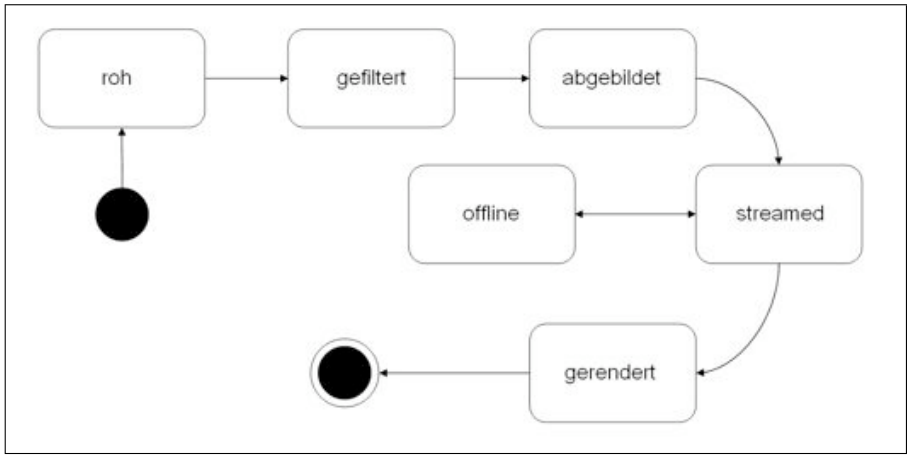


***Bild 4.2:*** *Zustandsdiagramm von* `Metaphor`*-Objekten*

Bild 4.2 ist das Zustandsdiagramm von `Metaphor`-Objekten, anhand dessen man die Stufen der Perzeptualisierungs-Pipeline wiedererkennt. Neu ist die Möglichkeit, partiell vorbereitete Volumen zum Renderer zu senden, um sie nachgelagert abzubilden oder zu rendern. Das erlaubt das Darstellen von 3-D-Texturen, flexibilisiert den Datenfluß und schafft mehr Gestaltungsspielraum bei einer Datenreduktion.

## 4.6 Abschließende Bemerkungen

Das Kapitel hat die technische Umgebung für die Datenreduktionstechniken spezifiziert. Die Gründe für die Wahl der Architektur, ihrer Gliederung und Abstimmung der ineinandergreifenden Verantwortung der Teile sind hierbei Bestandteil. Die Entwurfsziele folgen der Schule von Booch (1993); Jacobson et al. (1993); Coplien (1994); Lakos (1996). Sie propagieren folgende Entwurfsprinzipien: Lesbarkeit, Angemessenheit, starker Zusammenhalt und eine schwache Kopplung zwischen Modulen.

# Kapitel 5

# Codec-Entwurf

*„Shorter response time leads to shorter user think time."* (Ben Shneiderman in Shneiderman (1997))

Das Kapitel spezifiziert neue Datenreduktionstechniken, welche den in Kap. 4 spezifizierten Entwurf nutzbar machen.

## 5.1 Problemanalyse

Der in Kap. 4 spezifizierte Entwurf ist, für sich alleine implementiert, unbrauchbar. Die Latenz ist zu hoch und unterbricht den haptischen Renderer. Ich habe das in Vorgriff auf die nachfolgenden Kapitel in informellen Tests verifiziert. Dies bringt mich zurück auf meine Forschungsfragen aus Kap. 1.2:

- Welche Optimierungen sind notwendig und hinreichend, um ein Perzeptualisierungs-System zum Explorieren von zeitveränderlichen Volumendaten, die mehrere MByte/sec umfassen, einsetzen zu können?
- Ab wann nehmen Nutzer die Änderung der Ausgabequalität des Perzeptualisierungs-Systems wahr?

Ich nehme an, dass man das Perzeptualisierungs-System optimiert und dessen Ausgabequalität schont, indem man die Performance verbessert. Die Qualität soll hierbei gleichbleiben, gleich erscheinen oder vernächlassigbar sinken, sofern sie ihren Ursprungszwecken noch dient. Der Zeitbedarf resultiert aus der Überschreitung der Prozessorkapazitäten und Bandbreiten. Der Prozessorzeitbedarf muß gleichbleiben oder steigen, sofern er die Skalierbarkeit des Gesamtsystems nicht einschränkt, was bei linearer oder superlinearer Skalierbarkeit der Fall ist. Der Speicherbedarf muß tendenziell reduziert werden, sofern das dem Erreichen der anderen Aufgaben dient oder nicht zuwider läuft. Hierfür habe ich im Entwurf aus Kap. 4 folgende Methoden vorgesehen, die von den neuen Codecs in unterschiedlicher Art und Weise genutzt werden:

1. Clip
2. Cull
3. Quantise

4. Simplify
5. Compress
6. Decompress

## 5.2 Lösungsansätze

### 5.2.1 Übersicht der Codecs

- T1: Ein Algorithmus, um 3-D-Texturen aus irregulären Rohdaten zu gewinnen und zu reduzieren
- T2: Eine Technik, um Volumendaten wahrnehmungsneutral und verlustbehaftet zu reduzieren und sie zu haptischen Renderern zu senden
- T3: Ein Algorithmus, um Polygonnetze aus irregulären Rohdaten zu gewinnen und gleichzeitig zu vereinfachen

T1 und T3 sind in dem Sinne optimal, dass sie keine Nachrichtenaustausche verwenden und somit optimal mit der Anzahl an Prozessen skalieren. Sie sind generell auch für den Einsatz in dynamisch zerlegten Gebieten geeignet.

### 5.2.2 T1 – Reduktion von 3-D-Texturen

#### Übersicht

Der Algorithmus muß die Rohdaten in ein beliebiges aber festes Ausmaß bringen, mit dem Ziel, die Performance des Renderings nicht von der Rohdatengröße abhängen zu lassen. Die Technik ist am effektivsten, wenn ich sie unmittelbar beim Filter ansetze. Ich schlage vor, die Rohdaten zu subsamplen, zu quantisieren, und die 3-D-Texturfragmente beim Sammeln zusammenzusetzen. Dies erfordert keine besonderen Schritte, sofern jedes Texturfragment durch einen zusammenhängenden, abgeschlossenen Speicherblock beschrieben wird, was bei einer eindimensionalen Gebietszerlegung in eine beliebige Richtung der Fall ist. Die zusammengesetzte Textur hat für das Rendering geeignete Ausmaße und Voxeltiefen. Dann komprimiere, sende und dekomprimiere ich die Textur mit prädiktorenbasierten Codes, die circa 70% der Texturdaten verlustlos entfernen. Ich optimiere die Komprimierung und Dekomprimierung, so dass beim Render-Client 8,4 Millionen Voxel pro Sekunde erneuert werden.

#### Spezifikation

Das Neue an meinem Ansatz ist das verteilte Verarbeiten nicht-äquidistanter rektilinearer Volumendaten, verglichen mit derem lokalen Verarbeiten (Guthe et al. 2002). Die Daten können nicht direkt in den Speicher der Grafikkarte geschrieben werden. Zuerst wende ich die o. g. Gebietszerlegung über die Rohdaten an. Dann kann ich zwei Alternativen nutzen. Die eine Alternative bedeutet, nicht-äqudistanten Gebiete in äquidistante „Bausteine" zu teilen und auf dem Renderer zusammenzusetzen (Kähler et al. 2003). Die andere Alternative bedeutet, jedes Gebiet auf ein äquidistantes Maß zu bringen und die Resultate beim Filtern oder Abbilden zusammenzusetzen. Der Ansatz mit den Bausteinen zwingt den Renderer, jeden Baustein durch einen Funktionsaufruf in die 3-D-Textur einzupassen. Weiterhin hängt die Effektivität von den Merkmalen

der Daten ab. Der andere Ansatz reduziert die Updates auf je einen Funktionsaufruf, weil jedes rektilineare Volumen vor dem Rendern äquidistant ist. Die Effektivität hängt nicht von den Merkmalen der Daten ab. Außerdem braucht es dafür nur vom Filter oder Abbilder zusätzlichen Aufwand, und es erfordert dort den geringsten relativen Mehraufwand.

Der Pseudo-Code zeigt das Resampling am Beispiel einer Gebietszerlegung in $x$-Richtung:

```
void make_equidistant(UBYTE aTexture[], REAL aData[],
 int tw, int th, int td, int dw, int dh, int dd,
 REAL xcoord[], REAL ycoord[], REAL zcoord[],
 int xsteps, int ysteps, int zsteps)
{
 int  cx = 0; int  cy = 0; int  cz = 0;
 REAL dx = 0; REAL dy = 0; REAL dz = 0;
 int x = 0; int y = 0; int z = 0;
 REAL mx = (xcoord[xsteps-1]-xcoord[0])/tw;
 REAL my = (ycoord[ysteps-1]-ycoord[0])/th;
 REAL mz = (zcoord[zsteps-1]-zcoord[0])/td;
 for(z=0,cz=0,
  dz=zcoord[1]-zcoord[0];z<td;++z) {
  for(y=0,cy=0,
   dy=ycoord[1]-ycoord[0];y<th;++y) {
   for(x=0,cx=0,
    dx=xcoord[1]-xcoord[0];x<tw;++x) {
    aTexture[x+y*tw+z*tw*th] =
     aData[cx+cy*dw+cz*dw*dh];
    dx -= mx;
    while (dx < eps && ++cx < xsteps - 1) {
     dx += xcoord[cx + 1] - xcoord[cx];
    }
   }
   dy -= my;
   while (dy < eps && ++cy < ysteps - 1) {
    dy += ycoord[cy + 1] - ycoord[cy];
   }
  }
  dz -= mz;
  while (dz < eps && ++cz < zsteps - 1) {
   dz += zcoord[cz + 1] - zcoord[cz];
}}}
```

Der Algorithmus geht durch jede Voxelebene, liest die assoziierten Rohdaten und quantisiert sie in Ganzzahlwerte. Ich setze voraus, dass Hardware die Fließkomma-Operationen beschleunigt. Ein Digital Differential Analyser (DDA) oder der Bresenham-Algorithmus können den Algorithmus ersetzen. Über jedes Voxel kann trilinear interpoliert werden. Der Algorithmus arbeitet parallel für jeden Datengenerator. Der Sammler nimmt die Ausgaben und setzt sie zur 3-D-Textur zusammen.

### Nachbearbeitung durch PR0

Die Idee ist, die Topologie von der Topographie der 3-D-Textur zu trennen und gesondert zu komprimieren. Der Algorithmus arbeitet für jede Art von 3-D-Textur, wo große verbundene Regionen von Voxeln ungefähr die gleichen Werte haben. Die Verbundenheit ist die Topologie. Die Werte sind die Topographie. PR0 trennt die 3-D-Textur in einem Durchlauf in ein *Dictionary* von unkomprimierten Werten und eine Folge von Codes gewählter Prädiktoren. Diese verschränken sich bei Bedarf mit DPCM-Bits, um die Prädiktoren nur die Most Significant Bits erkennen lassen zu müssen.
Ich habe Prädiktoren in empirischen Testläufen zum Komprimieren und Dekomprimieren eines repräsentativen Trainings-Datensatzes ausfindig gemacht (Testdatensatz `bonsai` aus Bild B.1 im Anhang). Die statistische Trefferquote habe ich für Anhang B.1 ermitteln lassen. $q$ ist die Queue, welche die unkomprimierten Werte enthält.

$$e(x,y,z) = (v(x-1,y,z-1)+v(x+1,y,z-1))/2 \quad (5.1)$$
$$e(x,y,z) = (v(x,y-1,z-1)+v(x,y+1,z-1))/2 \quad (5.2)$$
$$e(x,y,z) = (v(x-1,y-1,z-1)+v(x+1,y+1,z-1))/2 \quad (5.3)$$
$$e(x,y,z) = (v(x-1,y+1,z-1)+v(x+1,y-1,z-1))/2 \quad (5.4)$$
$$e(x,y,z) = (v(x-1,y,z)+v(x,y-1,z))/2 \quad (5.5)$$
$$e(x,y,z) = v(x,y,z-1) \quad (5.6)$$
$$e(x,y,z) = (v(x-1,y,z)+v(x,y-1,z)+v(x-1,y-1,z))/3 \quad (5.7)$$
$$e(x,y,z) = pop(q) \quad (5.8)$$

Eine Variante ersetzt den siebenten Prädiktor durch den Lorenzo-Prädiktor:

$$\begin{aligned} e(x,y,z) = {} & v(x-1,y,z) - v(x,y-1,z-1) + \\ & v(x,y-1,z) - v(x-1,y,z-1) + \\ & v(x,y,z-1) - v(x-1+y-1,z) + \\ & v(x-1,y-1,z-1) \end{aligned} \quad (5.9)$$

Der Algorithmus probiert für jedes Voxel alle Prädiktoren der Reihe nach durch, ob sie den Originalwert rekonstruieren. Der Algorithmus kodiert den ersten der paßt und modifiziert dessen Code ggfs., um die DPCM-Bits anzuhängen. Der Pseudo-Code für den Komprimierer lautet ($t$ ist eine heuristische Funktion, die ich gleich erkläre)

```
for all v in input, s for output stream, q for dictionary
 for all p in predictors and t in triggers
  if | p(t, x, y, z) - v(x, y, z) | <= epsilon
   and t(x, y, z) == 1,
   fifopush(s, c) where c is the partial code of predictor
  if p is default predictor, fifopush(q, v(x, y, z))
   (p(x, y, z) - v(x, y, z)) = delta
  fifopush(s, tocode(delta))
```

Der Pseudo-Code für den Dekomprimierer ähnelt dem des Komprimierers:

```
read dictionary q
for all c in input stream s, v for output data
 find predictor p that is associated with partial code c
  if p is default predictor, v(x, y, z) = fifopop(q)
```

```
else for all t that belong to p // 'first-fit'
 if t(x, y, z) == 1, v(x, y, z) = p(t, x, y, z)
 if c contains modulation code delta, v(x, y, z)
  += fromcode(delta)
```

$t$ zu nutzen reduziert die Anzahl der Bits, mit denen ein gewählter Prädiktor spezifiziert wird, indem es implizite Informationen in den Eingabedaten evaluiert. Ich nenne das Prinzip *partial predictor encoding*, das wie folgt arbeitet: Einige Prädiktoren (in PR0 sind es die ersten sechs) teilen sich denselben Prädiktor-Code, und der Dekomprimierer muß eindeutig herausfinden, welcher der Unterprädiktoren mit dem Code referenziert werden soll. Der Komprimierer und der Dekomprimierer nutzen in diesem Fall die Heuristik $t$, um den Konflikt beizulegen, indem sie ermitteln, welcher der Prädiktoren der für das aktuelle Datenmuster passendste ist. In PR0 assoziiere ich jeden der ersten fünf Prädiktoren mit einer „Trigger"-Funktion, die entscheidet, welcher der Unterprädiktoren paßt ($\epsilon$ ist ein beliebiger, aber fester Wert, der den Fehler beschränkt).

$$\epsilon \geq abs(v(x-1,y,z-1) - v(x+1,y,z-1)) \tag{5.10}$$
$$\epsilon \geq abs(v(x,y-1,z-1) - v(x,y+1,z-1)) \tag{5.11}$$
$$\epsilon \geq abs(v(x-1,y-1,z-1) - v(x+1,y+1,z-1)) \tag{5.12}$$
$$\epsilon \geq abs(v(x-1,y+1,z-1) - v(x+1,y-1,z-1)) \tag{5.13}$$
$$\epsilon \geq abs(v(x-1,y,z) - v(x,y-1,z)) \tag{5.14}$$

Der siebente und der achte Prädiktor werden explizit in den komprimierten Daten kodiert und brauchen deswegen keine assoziierte Trigger-Funktion.
Der achte Prädiktor paßt immer, weil er den kompletten Wert in der Queue $q$ zum Dekomprimieren speichert. Die Anzahl der Einträge in der Queue gleicht der Anzahl der Male, die der Prädiktor kodiert wurde.
Indem ich die Trigger-Funktion nutze, halbiere ich die Anzahl der Bits um einen Prädiktor zu kodieren und habe damit Platz, einen 1-Bit DPCM-Code anzuhängen, so dass der komplette Code maximal drei Bit pro richtig vorhergesagten Voxelwert einnimmt. Der Komprimierer hängt mehr DPCM-Bits an, wenn er 3-D-Texturen verarbeitet deren Voxelwerte mehr als acht Bit einnehmen, weil dann mehr als ein Bit verrauscht ist. Ich habe gute Komprimierungsraten damit erzielt, fünf zusätzliche Bits einzusetzen, um die komplette Modulierung zu kodieren. Tab. 5.1 spezifiziert die Codes für Prädiktoren, welche 8-Bit-Datentypen vorhersagen (man beachte die Duplikate für die ersten sechs Prädiktoren).

| Prädiktor | Perfekt | Korrektur: -1 | Korrektur: +1 |
|---|---|---|---|
| 1..6 | 01 | 001 | 000 |
| 7 | 110 | 101 | 100 |
| 8 | 111 | – | – |

***Tabelle 5.1:*** *Huffman-Codes (mit Präfix-Eigenschaft) nach Jensen et al. (2005)*

### Weiterverarbeitung in DSVR

Die Bytefolge der Langworte, die die Voxelwerte enthalten, muß beim Transfer von einer CPU zur anderen richtig behandelt werden. Ein naiver Ansatz ist, jedes Langwort

mit `htonl()` vorzuverarbeiten und mit `ntohl()` nachzuverarbeiten. Das verringert die Performance des Renderers beträchtlich. Die Alternative ist, in die korrekte *Endianness* während des Abbildens zu übersetzen und diese beizubehalten. Das vermeidet weiteres Bearbeiten der Byte-Folge, aber es zwingt den Abbilder, 3-D-Texturen in beiden Endianness-Varianten zu erstellen. Die meisten Renderer nutzen praktisch gesehen jedoch nur das Little-Endian-Format. Somit stelle ich die Konvention auf, dass die 3-D-Texturen immer in diesem Format sind.
Die Renderer erhalten Daten und rufen die OpenGL-Funktion `glTexSubImage3D` auf, um die aktuell gerenderte 3-D-Textur zu ersetzen. Das Dekomprimieren eingehender 3-D-Texturen zwingt den Renderer dazu, einen temporären Puffer einzurichten, in den er die dekomprimierten Daten schreibt und diese in den Grafikkartenspeicher kopiert. Ein möglicherweise besserer Weg ist das Dekomprimieren auf der GPU, weil dann keine dekomprimierten Daten über den Bus fließen. Das Feature habe ich in Ermangelung von C-Compilern für Grafikkarten nicht implementiert. Die verbleibenden Features habe ich in DSVR-2 integriert und deren Robustheit validiert.

### 5.2.3 T2 – Volumenreduktion für das haptische Rendering

#### Übersicht

Der Abschnitt diskutiert eine Technik um haptisches Rendern zu beschleunigen, wenn eine oder mehrere 3-D-Texturen, der Stream von Szenen, zum Renderer transferiert werden müssen. In diesem Fall reicht es nicht, das Eintreffen jeder komplette Szene von der Quelle abzuwarten und die Standard-Renderings darauf anzuwenden (Huang et al. 1998). Das hat folgende Gründe:

- Eine komplexe Szene, selbst wenn sie auf das Ausmaß des *Sicht*kegels reduziert wurde, übersteigt die Bandbreite zwischen der Quelle und dem Renderer. Der Nutzer bemerkt die transferbedingte Latenz bei der Update-Rate.

- Komplexes Vorverarbeiten ist ein Nachteil, wenn neue Szenen regelmäßig in kurzen Intervallen eintreffen. Durch den gänzlichen Verzicht auf das Vorverarbeiten werden jedoch ebenfalls die Prozessor- und Speicherkapazität des Renderers überfordert. Die Szenen behindern somit das Erreichen hoher Frame-Raten (Mark et al. 1996).

Um die Probleme zu verstehen, sollten wir uns abermals vergegenwärtigen, dass die Latenz durch das Überbelegen von Prozessorzeit und Bandbreite zustande kommt. Das ist eine andere Latenz, als jene, die der Bearbeitung durch Hard- und Software innewohnt. Vorausgesetzt, dass letztere Latenz nicht das Problem darstellt, reduziere ich das Problem darauf, dass zuviele Daten zum Renderer gesendet werden. Damit lautet das Teilziel, einige der Szenendaten zu entfernen, was ich wiederum darauf reduziere, exakt die Daten zu senden, die zur haptischen Ausgabe gehören. Ich kombiniere Clipping, Quantisierung und die o. g. Komprimierung durch PR0, um einen verlustbehafteten, wahrnehmungserhaltenden Codec zu bauen. In diesem Abschnitt möchte ich als Vorbereitung zeigen, warum der Codec dem Nutzer stets die exakte Kontrolle über die Qualität der Ausgabe läßt. Ich entwerfe den Codec spezifisch dafür, ein einfaches und schnelles Vorverarbeiten für das direkte haptische Rendern von Volumen zu schaffen, das die Engpässe in der Perzeptualisierungs-Pipeline zu berücksichtigen hilft. Den Codec integriere ich in DSVR-2 und validiere dessen Anpaßbarkeit und Robustheit.

### Spezifikation und Integration

Mein Ansatz kombiniert Clipping, Quantisierung und den Einsatz von PR0 ohne das Subsamplen. Stattdessen wird das Tastverhalten des Nutzers zum Clipping herangezogen werden. Ich reduziere die Daten beim Filtern und Abbilden, um hohe Update- und Frame-Raten zu erzielen. Der Ansatz hat Vorteile gegenüber der einfachen verlustbehafteten Komprimierung, insbesondere durch die Eigenschaft, die Verlustrate bei der haptischen Wahrnehmungserhaltung exakt zu wählen. Die Komprimierungskette muß flexibel sein, um die Daten beim Transfer nicht zu beschädigen, wenn Glieder umgangen werden.

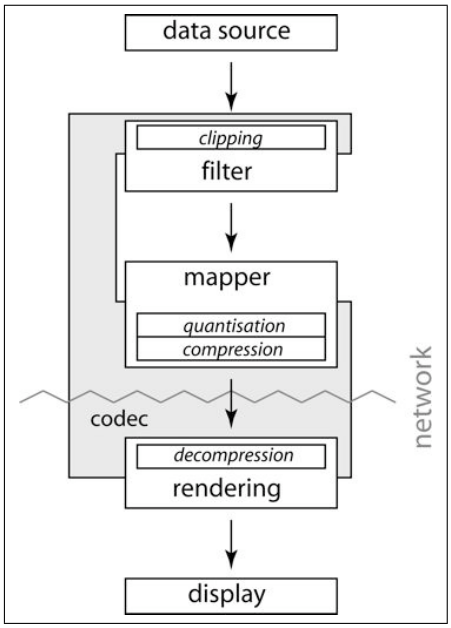


***Bild 5.1:*** *Perzeptualisierungs-Pipeline mit Clipping, Quantisierung, Komprimierung und Dekomprimierung (Bild: Dipl.-Inf. (FH) Gabriel Gaus)*

Die haptische Rendering-Pipeline (Mark et al. 1996) setze ich alternativ zur Grafikpipeline. Die erweiterte haptische Rendering-Pipeline ist auf Bild 5.1. Der erste Teil des Codecs ist die Clipping-Routine, welche Daten wegschneidet, die während der Zeit $t$ außerhalb des haptischen Frustums liegen. Der Quantisierer bildet die Voxelwerte auf Ganzzahlen ab. Der Codec schließt daran die Nachbearbeitung durch PR0 oder durch *Bit-Packing* an. Der Render-Client dekodiert und rendert die Daten nach Maßgabe des Rendering-Stiles. Die Position des End-Effektors wird an den Filter übermittelt. Das Rendern und das Empfangen sind Threads auf dem Render-Client.
Clipping und Quantisierung reduzieren den Aufwand für das Empfangen von Daten und beeinträchtigen die Frame-Rate nicht. Dekomprimierung tut dies und der Mehraufwand sollte daher einem anderen CPU-Core zugewiesen werden.
Das Frustum enthält idealerweise genau jene Voxel, welche die End-Effektoren binnen maximal eines Zeitschritts erreichen. Vereinfachend definiere ich das Frustum von der Gestalt eines Kubus mit den Kantenlängen $k^2 \cdot k/2$ (in Voxel). Bild 5.2 zeigt das Beispiel. Der Generator-Client generiert nur die Voxel im Frustum. Dies reduziert die

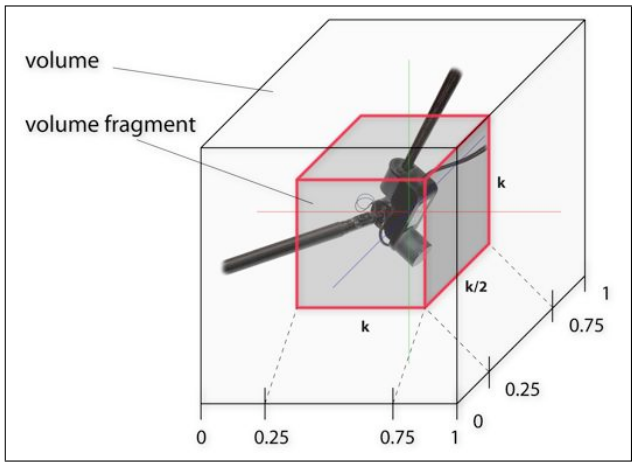


***Bild 5.2:*** *Clipping reduziert die Menge der weiterverarbeiteten Daten (Bild: Dipl.-Inf. (FH) Gabriel Gaus)*

Daten um 87,5%, wenn das Frustum $1/8$ des Gesamtvolumens abdeckt.
Der Quantisierer entfernt die $m$ Least Significant Bits für jeden Voxelwert und schreibt die verbleibenden Bits in den Ausgabestream, was in einer relativen Reduktion zwischen 12,5% und 87,5% resultiert. PR0 kann den Stream weiterverarbeiten. Das resultiert, bei sehr konservativen Schätzungen, in einer Reduktion von 50%. Im Schnitt machen die Ausgabedaten somit nur noch 3% von den Rohdaten aus.
Die Daten können gespeichert oder transferiert werden. Genau die Information geht verloren, die von der Wahl von $k$ und $m$ abhängt. Wenn eine bidirektionale Verbindung zwischen dem Generator und dem Renderer besteht, kann dies benutzt werden, um die Volumendaten progressiv zu versenden. Eine ideale Balance zwischen der Größe des Frustums, den Updates/sec und der Latenz verursacht nur noch $m$ Informationsverluste. Bei $m = 0$ liegt kein und bei $m = 7$ der maximale Verlust vor. Kap. 7 spezifiziert kontrollierte Experimente, in denen ich herausgefunden habe, welche Werte von $m$ ein Nutzer bemerkt.
Der Renderer bildet die Voxelwerte auf Dreiecke oder Viskositäten ab. Viskosität bildet man in Ghost nach, indem man die Methode `gstEffect::calcEffectForce()` überlädt, um die Formel 2.18 für eingehende Geschwindigkeiten des End-Effektors auszurechnen. Die Methode gibt den Kraftvektor (in Newton) zurück. Die Viskosität $= [0..255]$ muß ich vor dem Einsetzen in die Formel auf das Intervall $[0..0,003]$ linear abbilden, um den Betrieb des haptischen Geräts geräusch- und vibrationslos zu halten.

### 5.2.4 T3 – Reduktion von Polygonnetzen

#### Übersicht

Der Abschnitt spezifiziert eine Datenreduktionstechnik für einen Renderer, der viele Polygonnetze pro Sekunde bearbeitet. Ich schlage vor, Marching-Cubes mit topologieschonendem Vertex-Clustering auf dem Abbilder zu verzahnen. Polygone werden in Dreiecke überführt, gesammelt und an die Renderer gesendet. Der Algorithmus ist schneller als qualitätsorientierte Polygonnetz-Vereinfachung und akkurater als

nicht verzahntes Vertex-Clustering und Subsampling, weil die Konnektivitätsinformation über benachbarte Kuben solange erhalten bleibt, wie das vereinfachte Dreiecksnetz im Bau ist. Eine nicht-optimierte Implementierung generiert circa 40000 Dreiecke/sec/Prozessor.

### Spezifikation und Integration

Der Algorithmus verzahnt die Polygonnetzgenerierung mit der Vereinfachung. Marching-Cubes dient als Abbilder, um die Eingabe inkrementell zu erzeugen. Zu jeder Zeit existieren das vereinfachte Ausgabenetz und das partielle dichte Eingabenetz.
Der Vergleich der Vereinfachungs-Algorithmen aus Kap. 2 zeigt die gute Eignung von Vertex-Clustering im Zusammenspiel mit unbekannten Eingabedaten und Marching-Cubes. Besonders sticht hervor, dass beide das Klassifikationsgitter nutzen. Weil dessen Ausrichtung von Marching-Cubes gegeben wird, hat Vertex-Clustering automatisch Zugriff auf die optimale Ausrichtung.
Mein 3-D-Sampling/Vereinfachungs-Konzept wurde durch die Faltungsoperatoren in der Bildverarbeitung inspiriert. Dort wendet man einen Operator wiederholt auf alle Pixel des Bildraums an. In dieser Metapher repräsentiert das Bild das Volumen. Das Ausmaß der Operator-Matrix definiert eine `Gruppe`. Sie enthält eine 2 x 2-Anordnung von `Clustern` mit variabler Größe, die benutzt wird, um beieinanderliegende Dreiecke zu dezimieren. Der Ort der Anwendung des Operators ist der „Vereinfachungs-Cursor”:

```
for each group in sub-volume on process p {
 for each cluster in group {
  for each cube in cluster {
   apply marching cubes to the cube
   store vertices
 }}
 cluster vertices
 generate triangles
}
merge triangles from processes 0..p
```

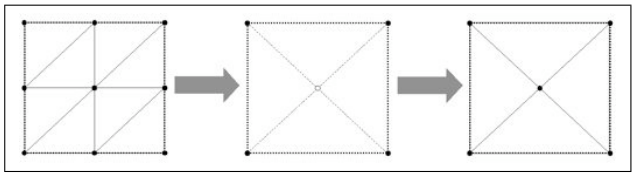

***Bild 5.3:*** *Schema der Dreiecksersetzung in einer Gruppe mit Clustergröße 1*

Das Clustering entwerfe ich so, dass es Teile der Topographie erhält, indem es einige Ecken dort beläßt wo sie sind, einige andere mittelt und die verbleibenden entfernt (Bild 5.3). Jeder Prozeß hat per Definition Zugriff auf ganze Gruppen, somit kann er unabhängig von anderen Prozessen Ecken clustern, eliminieren und neu vernetzen.
Das Vertex-Clustering an Gebietsrändern arbeitet ohne Nachrichtenaustausche. Das zeige ich mittels eines Beweises durch Widerspruch. Eine „Zelle” ist hierbei dual zum Kubus in Marching-Cubes.

**Beweis 5.1** *Man nehme an, jeder Prozeß greife auf mindestens eine Gruppe plus einen Rand von Zellen zu, der durch die Sampling-Auflösung von Marching-Cubes gegeben ist. Man nehme weiter an, dass adjazente Prozesse derart unterschiedlich vereinfachen, dass dies zu unterbrochenen Übergängen an den Rändern führt. Das widerspricht jedoch der Definition eines Algorithmus als deterministische Anweisungsfolge, also auch Prozeßunabhängigkeit, weil jeder Prozeß die gesamte erforderliche Information bei der vorgegebenen Konfiguration hat.* □

Der Algorithmus hat zwei Teile: Das zu iterierende, inkrementelle Marching-Cubes, und den Clusterer, der die reduzierten Dreiecksmengen in den Stream schreibt. Er behandelt auch Ränder, die kleiner als die vorgegebene Clustergröße sind, sofern sie mindestens eine Gruppe von `Clustern` der Größe 1 ausfüllen.

```
for each volume in sequence {
 for each group in volume {
  for each cluster in group {
   for each cube in cluster {
    calculate current x, y, z in volume
    if (is_boundary(x, y, z)) {
     marching cubes({x,y,z}..border limit)
     for each triangle
      push vertices on stack v
    }
    else {
     marching cubes({x,y,z}..cluster limit)
     for each triangle
      push vertices on stack v
  }}}
  commit_clustered_polys(...)
}}
```

Das Clustern und Vernetzen nimmt einen Teil der Dreiecke und berechnet das lokale arithmetische Mittel der Koordinaten. Die Fortsetzung zwischen den Ecken adjazenter `Cluster` bleibt erhalten (Bild 5.3). Ich gebe den Pseudo-Code. `j` ist die Nummer des `Clusters` und `PREV_X` das vorher bearbeitete `Cluster` in $x$-Richtung.

```
commit_clustered_polys(PolygonData *pd,group_t *g,
                       int groupnumber,
                       int xdim,int ydim,int zdim)
{
 while (!is_empty(&g[groupnumber].vertexstack)) {
  retrieve_polygon(&g[groupnumber],
   p[j],p[j+1],p[j+2]);
  // calculate cluster point
  for i=j..j+2 {
   maxs[0] = MAX(p[i][0], maxs[0]);
   maxs[1] = MAX(p[i][1], maxs[1]);
   maxs[2] = MAX(p[i][2], maxs[2]);
   mins[0] = MIN(p[i][0], mins[0]);
   mins[1] = MIN(p[i][1], mins[1]);
   mins[2] = MIN(p[i][2], mins[2]);
```

```
  cp[0]+=p[i][0];cp[1]+=p[i][1];cp[2]+=p[i][2];
 }
 ++polys;
}
g[groupnumber].maxx..z[0] = nil
if (polys >= 1) {
 cp[0] /= polys * 3;
 cp[1] /= polys * 3;
 cp[2] /= polys * 3;
 // find all vertices at group boundary
 // is boundary in x-direction?
 if (g[groupnumber].xb) {
   g[PREV_X(groupnumber,
    xdim,ydim,zdim)].maxx[0]=nil
 }
 ... same for y, z [skipped] ...
 for (--i; i >= 0; --i) {
  if (fabs(p[i][0] - maxs[0]) < eps) {
   g[groupnumber].maxx[x] = p[i]
   x += 3;
  }
 ... same for y, z [skipped] ...
 if (g[groupnumber].xb
  && fabs(p[i][0] - mins[0]) < eps) {
  g[PREV_X(groupnumber,
   xdim,ydim,zdim)].maxx[xm]=p[i]
  xm += 3;
 }
 ... same for y, z [skipped] ...
}
// mark end of lists with <end vertices>
g[groupnumber].maxx[x] = nil
sort_vector(g[groupnumber].maxx, x, z);
... same for y, z [skipped] ...
remove_doubles(g[groupnumber].maxx, x+3);
... same for y, z [skipped] ...
if (g[groupnumber].xb) {
 g[PREV_X(groupnumber,
  xdim,ydim,zdim)].maxx[xm]=nil
 sort_vector(g[PREV_X(groupnumber,
  xdim,ydim,zdim)].maxx,xm,z);
 ... same for y, z [skipped] ...
 remove_doubles(g[PREV_X(groupnumber,
  xdim,ydim,zdim)].maxx,xm+3);
}
... same for y, z [skipped] ...
// connect cluster point cp with corners
remesh(pd, g[PREV_X(groupnumber,
 xdim,ydim,zdim)].maxx,cp,corners);
remesh(pd, g[groupnumber].maxx, cp, corners);
```

```
 ... same for y, z [skipped] ...
}}
```

Das effektive Vernetzen erfolgt durch das Wählen einer neuen Ecke, die der Durchschnitt der entfernten Ecken ist. Es werden, im Uhrzeigersinn entlang jeder Dimension, von dort zu jedem umrandenden Paar Kanten gespannt. Dann werden die umrandenden Paare ebenfalls über Kanten verbunden. Das Ergebnis ist eine valide Dreiecksmenge mit guten Seitenverhältnissen und hat lokal vier Ecken mit dem originalen Netz gemein. Wegen der Art der Verarbeitung nenne ich diesen Remeshing-Ansatz die „Wendeltreppen-Methode". Das Ergebnis ist genau dann mit der Original-Topologie konsistent, wenn die originale Oberfläche keine Löcher hat und an jeder Stelle genau eine Dimension durchläuft. Der Algorithmus tendiert dazu, die Topographie zu ändern, wenn er die Oberfläche an Stellen steiler Krümmung abflacht. Die Topologie wird dort verändert, wo die originale Oberfläche Löcher hat, oder mehr als eine Dimension durchläuft. Eine topologieschonende Näherung bestünde im adaptiven Ordnungstausch der traversierten Dimensionen während des Vernetzens.
Ein Nachteil ist der Zwang, die Ecken zu sortieren, weil das die teuerste Operation im Algorithmus ist. Das Umgehen des Sortierens würde den Algorithmus verbessern. Das erreicht man ohne Mehraufwand, indem man Marching-Cubes programmiert, die Ecken in garantierter Ordnung auszugeben. Die Zeit- und Speicherkomplexität ist $O(n)$.

## 5.3 Abschließende Bemerkungen

Ich habe beschrieben, wie die aus Kap. 3 genannten Anforderungen berücksichtigt und in die Architektur aus Kap. 4 eingegliedert werden. Ich habe neue Algorithmen zur mehrstufigen Volumenvereinfachung und zur Polygonmengenvereinfachung vorgestellt. Diese sind für den Einsatz in DSVR konzipiert worden, aber gleichzeitig von genereller Relevanz und auf verwandte Einsatzfelder übertragbar.

# Kapitel 6

# Implementierung des Prototyps

*„Ruf eine von den Dirnen; der Sturm drückt uns die Scheiben ein, die Luken müssen angeschroben werden!"* (Theodor Storm in Storm (1988))

Das Kapitel enthält Kommentare zu Auszügen aus DSVR-2.

## 6.1 Implementierung

### 6.1.1 Übersicht

Ich habe den objektorientierten Entwurf in C unter Verwendung von MPI und OpenGL in Funktionen implementiert und in DSVR-1 eingegliedert. Funktionen kapseln auch den Zugriff auf die Klassen des Ghost-SDK, die ich jedoch, wegen eines Entwurfs-Fehlers vom Hersteller, nur mit dem Adapter Ghostwrap unter neueren Entwicklungsumgebungen nutzen kann.
Ein weiteres Problem trat dadurch auf, den Streaming-Server für Linux portieren zu wollen. Linux bietet keine Bidirectional pipes, die der Server zur Interprozeß-Kommunikation verwendet hat. Ich habe daher die Bidirectional pipes durch Unidirectional pipes ersetzt.

### 6.1.2 Spezifikation

**Funktionen**

Das Eingliedern von `Tensorfield`s gestaltet sich als unproblematisch. Man modelliert einzig die Unterklasse aus und überträgt deren Methoden auf die bestehende funktionale Implementierung im Generator-Client, der durch die Bibliothek DVRP implementiert und an C- und Fortran-Programme gebunden wird:

1. Bestimme eine Gebietszerlegung und die Rohdaten
2. Spezifiziere ein `Tensorfield` auf den Rohdaten
3. Rufe die Perzeptualisierungs-Routine auf
   (a) Filtere die Rohdaten und bilde sie auf ein `Tensorfield`-Fragment ab
   (b) Sammle die Fragmente über alle Prozessoren

(c) Sende das `Tensorfield`

Das ist der Auszug aus der Implementierung einer Anwendung (Fortran-90), welcher der Instanzierung eines `Metaphor`-Objekts und dem Aufruf der Methode `setRaw-Data()` entspricht:

```
! (1)
call dvrp_grid(stream,nx1,ny1,nz1,xcoor,ycoor,zcoor)
call dvrp_data(stream,data(ix1,iy1,iz1),...,nx1,ny1,nz1,...)
! (2) (new)
call dvrp_tensorfield(stream,packed,256,256,128/PUs,
                      0,0,0,256,256,128,0.0,1.0)
! (3)
call dvrp_visualize(stream,dvrp_vis_tensorfield,nr)
```

In eine `switch-case`-Anweisung wird die Marke `dvrp_vis_tensorfield` gesetzt, von der die Funktion `tensorfield()` aufgerufen wird:

```
/* scale rectilinear to uniform data grid and
   computes for every voxel v:
    aTexture[v] = (aData[v + 'offset'] * factor + sum)
   implements: filter(),clip(),cull(),quantise(),map()
*/
static void tensorfield(
 unsigned char aTexture[], // output
 REAL aData[],             // input
 int32 aMask[],            // reserved
 int tw, int th, int td,   // output dimension
 int dw, int dh, int dd,   // input dimension
 int xq, int yq, int zq,   // 'offset' in input
                           // coordinates...
 REAL xcoord[], REAL ycoord[], REAL zcoord[],
 int xsteps, int ysteps, int zsteps,
 int ascending,            // order of coordinates
 REAL sum, REAL factor);   // simple filter
```

Vor dem Transfer ruft das System `write_dvr_tensorfield()` auf, welches die Methoden `compress()` und `writeToStream()` implementiert:

```
void write_dvr_tensorfield(
 FILE *outfile,int type,int vx,int vy,int vz,
 int xq, int yq, int zq,unsigned char *field,
 int *nbytes) {
 ...
 head.record_type = RT_TensorField;
 ...
 nlen = compressVolume(field, vx, vy, vz, vs, nlen);
 field = vs;
 type = htonl(type);
 vx = htonl(vx); vy = htonl(vy);
 vz = htonl(vz); xq = htonl(xq);
 yq = htonl(yq); zq = htonl(zq);
```

```
 // turn to little endian
 dvr_swap(field, nlen / sizeof(int32));
 fwrite(&type, sizeof(int), 1, outfile);
 fwrite(&vx, sizeof(int), 1, outfile);
 fwrite(&vy, sizeof(int), 1, outfile);
 fwrite(&vz, sizeof(int), 1, outfile);
 fwrite(&xq, sizeof(int), 1, outfile);
 fwrite(&yq, sizeof(int), 1, outfile);
 fwrite(&zq, sizeof(int), 1, outfile);
 fwrite(field, nlen, 1, outfile);
 *nbytes += len + sizeof(head);
}
```

Der Render-Client, DocShow-VR, entnimmt die PDU (`readFromStream()`) und übersetzt sie nach OpenGL:

```
void cbTensorField(..., RecordHeader *head, long *ml, ...) {
 ...
 if (type & packed) {
  decompressVolume(ml,
   ntohl(head->record_length)-sizeof(int)*7,voxel,vx,vy,vz);
  texData = (void *)voxel;
  type &= ~packed;
 }
 else {
  texData = (void *)ml;
 }
 glTexSubImage3DEXT(GL_TEXTURE_3D_EXT,0,xq,yq,zq,vx,vy,vz,
  GL_COLOR_INDEX,GL_UNSIGNED_BYTE,texData);
 ...
}
```

Die Textur wird auf eine Reihe von Schnittflächen gelegt, die als Polygone generiert werden und stets zur Blickrichtung orthogonal stehen. Man invertiert hierzu die Projektionsmatrix und wendet sie auf die Polygone an. Zum Einfärben der Textur wird eine Farbindextabelle erzeugt und an OpenGL übergeben. Ein Beispiel ist

```
 ...
 GLubyte rgbaTable[256 * 4];
 int i = 0;
 glEnable(GL_SHARED_TEXTURE_PALETTE_EXT);
 for (; i < 256 * 4; i += 4) {
  const int j = i / 4;
  rgbaTable[i] = (j >= 128 ? (j - 128) * 2 - 1 : 0);
  rgbaTable[i+1] =(j<=128 ?(j * 2 + 1): ((256-j)*2-1));
  rgbaTable[i+2] =(j<=128 ?((((255 - j)-128)*2-1)): 0);
  rgbaTable[i+3] =alpha;
 }
 glColorTableEXT(GL_SHARED_TEXTURE_PALETTE_EXT,
  GL_RGBA,256,GL_RGBA,GL_UNSIGNED_BYTE,rgbaTable);
}
```

Die weiteren Implementierungen orientieren sich an den Pseudo-Codes aus Kap. 4–5.

### Ghost

Der Renderer berechnet die Kraft $f = -u_{xyz} \cdot v$, mit den Koordinaten x, y und z. $u_{xyz}$ ist der Voxelwert. $v$ ist die Geschwindigkeit des End-Effektors. $f$ ist die direktionale Kraft in Newton, die vom Ghost-SDK gelesen und ausgeübt wird.
Man initialisiert das haptische Rendering für einen neuen Thread:

```
// core class that depends on Ghost SDK - forward declaration
class gstEffect;
// user-defined callback to calculate force
typedef void (*UH_PFUNC_FORCE)(void *p,       // i/o reserved
                               void *pPhantom,// in phantom
                               double dPos[3],// in position
                               double d[3]);  // out force
class uhGwpGenericEffect
{
public:
 uhGwpGenericEffect(void *pData = 0);
 ~uhGwpGenericEffect();
 void setCallbackForce(UH_PFUNC_FORCE pCallback);
 void calculateForce(void *pPhantom,
                     double dPos[3],
                     double dForce[3]);
 // gstEffect is its superclass, not shown here
 gstEffect *self() {return m_pSelf;} // <Adapter> pattern
private:
 gstEffect *m_pSelf;
 UH_PFUNC_FORCE m_pCalculateForce;  // <Callback> pattern
 void *m_pData;
};
...
extern "C" DWORD WINAPI runHaptic(LPVOID p)
{
 ...
 uhViscosityEffectData visc;
 uhGwpGenericEffect effect(&visc);
 visc.getMagnitudeData = p->positionCB;
 effect.setCallbackForce(calculateViscosityForce);
 myPhantom->setEffect(effect.self());
 myScene.startServoLoop();
 ...
}
```

`runHaptic()` nimmt den Zeiger `positionCB` auf die von DSVR bereitgestellte Funktion zum Lesen der Voxelwerte und gibt ihn an das Objekt `visc`. Ein `Ghost-wrap`-Objekt kapselt und versteckt `visc`, um den Zugriff außerhalb Ghosts zu ermöglichen. Ghost ruft bei jedem Rendern `calculateViscosityForce()`, das wiederum `*positionCB()` anspringt. Der Rückkanal C2G (nicht gezeigt) dient zum Senden der Koordinaten, um die Voxelwerte zu lesen.

#### Unix-Pipes

Eine Bidirectional pipe wird durch zwei gegenläufige Unidirectional pipes ersetzt. Über die Pipes erhalten jene Server-Prozesse, welche Szenen replizieren, Nachrichten von einem Master-Prozeß (Dispatcher). Dadurch laufen die Nachrichten, als Graph vorgestellt, sternförmig. Der Nachteil der geringeren Skalierbarkeit wird durch mehrere bekannte Vorteile aufgewogen:

- Konstante Latenz
- Zentraler Verwaltungsaufwand
- Flexibler Nachrichtenfluß

Die Nachrichten zum Frame-genauen Synchronisieren von Szenen werden zwischen allen Prozessen ausgetauscht, die dieselbe sogenannte Session-ID haben. Diese wird vorher von dem oder den Clients festgelegt und an den Server gesendet. Der Server bestimmt einen der Prozesse als Taktgeber. Die zufällige Wahl der Session-ID schließt ungewünschtes Einklinken in eine bestehende Sitzung mit einer Wahrscheinlichkeit von $1/4.294.967.296$ aus.
Jeder Client synchronisiert die ankommenden Szenen mittels einer Bitmaske. Jede Position in der Bitmaske repräsentiert genau eine auszuspielende Szene. Für jeden ankommenden Frame wird das Bit an der entsprechenden Position auf 1 gesetzt. Erst wenn alle Bits auf 1 sind, wird das Bild neu gezeichnet und der Bitmaskeninhalt gelöscht:

```
...
 static int receiveMask = -1;
 receiveMask |= (1 << scene);
 if (-1 == receiveMask) {
  PostMessage(pThis->WndProp->win,WM_COMMAND,
              CM_END_FRAME,scene);
  receiveMask = implicitUpdateMask;
 }
}
...
```

Die Lösung setzt synchrone Szenen zu Anfang des Abspielens voraus.

#### Portierungen

Die Makefiles habe ich so geändert, dass das System unter Cygwin, Linux und AIX läuft. Die Makefiles unterstützen das Kompilieren für 32- und 64-Bit-Rechencluster, sowie für 64-Bit-Vektorrechner.

## 6.2 Abschließende Bemerkungen

Das Kapitel hat Aspekte der Implementierung spezifiziert. Beiliegender Datenträger enthält Quellen (5 Mbyte), Makefiles, Kompilate und Demos.

# Kapitel 7

# Messung der psychophysischen Funktion

*„The real voyage of discovery consists not in seeking new landscapes but in having new eyes."* (Marcel Proust)

Das Kapitel spezifiziert eine Studie, deren zugrundeliegende Zielsetzung das Untersuchen der erforderlichen Ausgabqualität haptischen Renderings für einen ausgewählten Stil ist.

## 7.1 Aufbau

Meine Studie realisiert 3-Interval-Forced-Choice (3-IFC), dessen Aufbau ich gleich erkläre. Damit ermittle ich den Einfluß von Quantisierungsverlusten $m = [1..7]$ in T2 (Kap. 5). 3-IFC ist effizient und gibt, verglichen mit der Method of adjustment und der Method of limits, genauere Resultate (Leek 2001). Der generelle Aufbau entspricht dem einer adaptiven Treppen-Methode.

Ein Proband erhält jede Sekunde ein neues Volumen, dessen Dichte er als Viskosität ertastet. Die Software gibt pro Versuch drei Volumen aus. Zwei Volumen bestehen aus einer gleichmäßigen Dichte $d = [0..(2^8 - 2^m - 1)]$, die von der Software per Zufall gewählt wird. Das dritte Volumen hat die Dichte $d + 2^m$ und wird als „Signal" bezeichnet. Für jede haptische Präsentation wird die Nummer 1–3 auf dem Monitor gezeigt. Der Nutzer gibt die Nummer ein, die angezeigt wurde, während er das Signal ertastet hat. Wann das Signal erscheint, ist zufallsgesteuert. Nach jedem Versuch liest der Proband, ob er das Signal richtig erraten hat. Um die 50%-Marke der psychometrischen Funktion ($X_{50}$) zu ermitteln, wird $m$ nach jeder falschen Antwort inkrementiert und nach jeder richtigen Antwort dekrementiert. Um die 84%-Marke der psychometrischen Funktion ($X_{84}$) zu ermitteln, wird $m$ nach jeder falschen Antwort inkrementiert und nach vier nacheinander richtigen Antworten dekrementiert. In jedem Fall terminiere ich das Experiment mit der zehnten Umkehr von schwachen zu starken Dichten. Das ist ein Kompromiss zwischen der erwarteten experimentellen Genauigkeit und der Ermüdung der Probanden.

## 7.2 Durchführung

Außer meiner Person hatten sich ein Mann und eine Frau aus meinem Institut sich freiwillig gemeldet, an der Studie teilzunehmen. Die Probanden waren zum Zeitpunkt der Studie 31, 34 und 19 Jahre alt. Jeder führte die Experimente zu einem anderen Zeitpunkt durch. Ich instruierte die Probanden bzgl. der Ziele und der Art der Studie, und jeder nutzte das System für circa 3 Minuten, um sich damit vertraut zu machen. Am Ende des Trainings folgte jeder Proband dem Experiment, um $X_{50}$ und dann $X_{84}$ festzustellen. Ich schätzte von den protokollierten Antworten den RL und den DL[1] für jeden Probanden.

## 7.3 Ergebnisse

Bild 7.1 spezifiziert die Ergebnisse. Proband A hat schätzungsweise einen RL von $X_{50} = 16$ und einen DL von $X_{84} - X_{50} = 16$. Proband B und C haben einen RL von 48 und einen DL von 16.

## 7.4 Diskussion

Inkorrekte Messungen sind inkonsistent zur S-förmigen psychometrischen Funktion und können nicht zwischen den Probanden reproduziert werden. Die Erfolgsrate von Proband A für eine Signalintensität von 4 in $X_{50}$ resultiert daraus, dass dieser Wert nur dreimal erreicht wurde. Jedesmal wurde das Signal erraten. Den Effekt konnte ich weder für 8 in $X_{50}$ noch für 4..8 in $X_{84}$ reproduzieren. Auch hat keiner der anderen Probanden ähnliche Ergebnisse gezeigt. Das gleiche Argument gilt für die Erfolgsrate von Proband C für 16 oder 32 in $X_{50}$, wo die Diskrepanz darin besteht, dass beide Wahrscheinlichkeiten vertauscht werden müßten, um die S-förmige Kurve besser anzunähern. Dieser Einzeleffekt kann das Ergebnis davon sein, dass der Proband kurzzeitig abgelenkt war oder ermüdete. Insgesamt zeigen die Daten, dass das Experiment das relevante Intervall erfolgreich gemessen hat, dass alle Probanden demselben psychometrischen Modell beim Evaluieren von Viskositäten folgen, und dass die Genauigkeit, mit der Viskositäten unterschieden werden, nahe bei vier Bit auf dem Testgerät liegt. Dieser Wert ist kleiner als die sechs Bit, die in (Burdea 1996, Bild 2.15) zitiert wurden, weil ich

- 3-IFC statt der Method of Adjustment benutzt habe,
- ein anderes haptisches Gerät mit unterschiedlicher Genauigkeit eingesetzt habe,
- die Kraft auf das Intervall beschränkt habe, die vom haptischen Gerät ausführbar ist und
- eine andere Gleichung zum Synthetisieren der Viskosität habe ausführen lassen.

Von den neuen Ergebnissen lernen wir die höchstwahrscheinlichst wahrnehmungsneutrale Konfiguration für den Codec T2. Der Quantisierungsfaktor ist maximal $8 - 4 = 4$ Bit, ohne dass er die haptische Wahrnehmung konfundiert. Mit anderen Worten: Ein Informationsverlust von 50% vermindert nicht die Ausgabequalität des sich anschließenden haptischen Renderings.

[1] Zur Erinnerung: Reiz- und Differenz-Limes

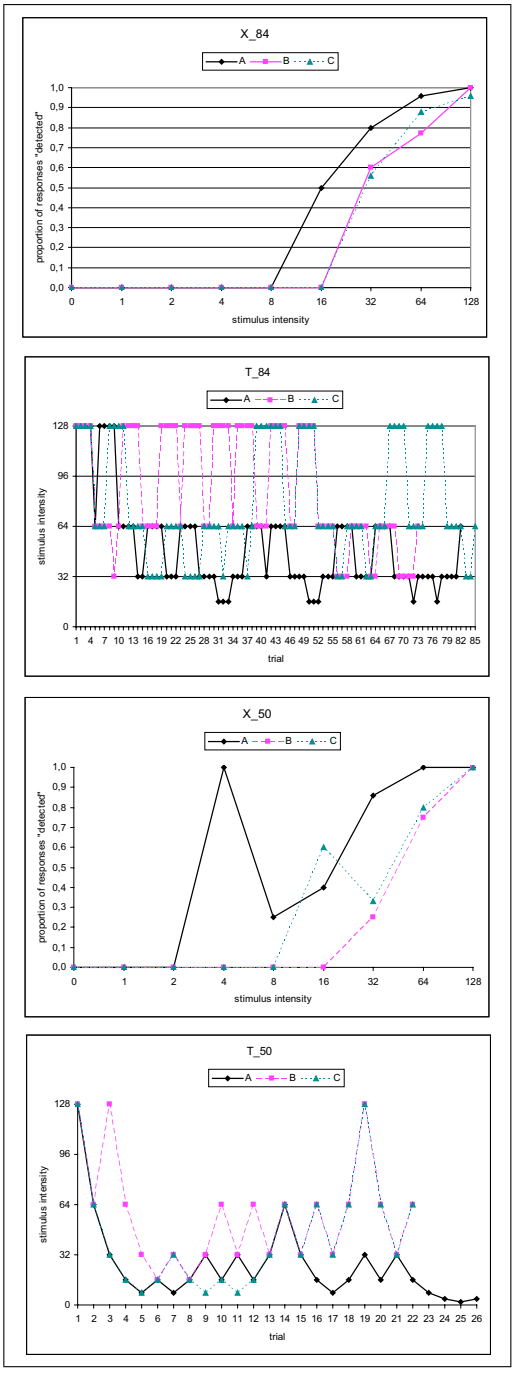


***Bild 7.1:*** *Ergebnisse der 3-IFC-Studie*

## 7.5 Abschließende Bemerkungen

In diesem Kapitel habe ich eine Studie beschrieben, um den Effekt des Quantisierens beim direkten haptischen Rendering zu quantifizieren.

# Kapitel 8

# Test und Diskussion

*„Avoid inlining or detailed tuning until performance profiles prove the need."* (Herb Sutter in (Sutter 2002, S. 86))

Das Kapitel dokumentiert die Modul-, Integrations- und Lasttests.

## 8.1 PR0

### 8.1.1 Aufbau

Ich habe MS Windows XP und MS C++ Version 13.10.3077 genutzt. Mein Arbeitsplatzrechner war ein Personal Computer (PC) von Dell mit 37 GByte Festplattenplatz, 512 MByte Hauptspeicher, und einem mit 2,4 GHz getakteten Pentium 4 mit eingeschaltetem Cache. Ich habe darauf PR0 in beiden Varianten (ohne/mit Lorenzo-Prädiktor) in C implementiert und anhand der Performance-Profile optimiert. Der Compiler hat für einen Intel Pentium 4 die Teile `PREDICT_INCORE` und `PREDICT_FAST` übersetzt. `PREDICT_SHORTDICT` wurde wahlweise mit typedefs eingeschaltet, um 16-Bit-Voxelwerte zu erzeugen.
Ich habe die Testdaten von Engel (2001a,b); Levoy (2004a,b) genommen, sie ins Little-Endian-Format konvertiert und zu je einer Datei verbunden. Weitere Testdaten waren ein 24,5 MByte großer Computertomographie-Scan (`crt-2`) und eine 225 MByte umfassende Volumensequenz (`funcbrain`). Die Sequenz ist eine funktionale Messung der Hirnaktivität[1]. Tab. 8.1 spezifiziert die Größe jedes Volumens als die Zahl der enthaltenden Voxel in jede Dimension.

### 8.1.2 Durchführung

Ich habe jede Datei einmal im Modus $-1$ (Worst-Zip) und im Modus $-9$ (Best-Zip) mit Info-Zip Version 2.3 unter Cygwin 1.3 auf XP komprimiert und dekomprimiert. Dort war das Zip-Programm vorinstalliert und konnte ohne Mehraufwand ausgeführt werden. Ich habe PR0 ohne Lorenzo-Prädiktor für jede Datei benutzt und jede `.pr0`-Datei mit Info-Zip einmal im Modus $-1$ und im Modus $-9$ komprimiert (letzteres ist nicht in den Tabellen, weil es dreimal langsamer als $-1$ arbeitete und nur wenige

[1] Die beiden letztgenannten Testdaten wurden von Prof. Dr. Dr. J. Bernarding, Universität Magdeburg, zur Verfügung gestellt.

| Name | X | Y | Z | Bits/Voxel | # Volumen |
|---|---|---|---|---|---|
| bonsai | 256 | 256 | 128 | 8 | 1 |
| brain | 256 | 256 | 109 | 16 | 1 |
| engine | 256 | 256 | 128 | 8 | 1 |
| head | 256 | 256 | 113 | 16 | 1 |
| crt-2 | 256 | 256 | 192 | 16 | 1 |
| funcbrain | 64 | 64 | 32 | 16 | 900 |

***Tabelle 8.1:*** *Testdaten; Bild B.1 im Anhang zeigt die Bilder von links n. r., o. u. (Jensen et al. 2005)*

KByte sparte). Ich habe die Dauer für jede Tätigkeit pro Datei gemessen. Der komplette Prozeß wurde noch zweimal ohne `funcbrain` wiederholt, und die Ergebnisse arithmetisch gemittelt. Für die Feinmessungen habe ich in PR0 `PREDICT_BENCHMARK` eingeschaltet, um durch `GetTickCount()` der Windows-API die Durchnittsdauer des Verarbeitens jeder Schnittfläche der `crt-2`-Datei abzufragen. Die Feinmessungen waren Teil eines separaten Testlaufs.

### 8.1.3 Ergebnisse

PR0 gibt an, dass das Verarbeiten von $256^2 \cdot 16$ Bit acht Millisekunden dauert. crt-2 zu komprimieren dauert 896 msec. Tab. 8.2 gibt die Komprimierung als die prozentuale Größe der komprimierten Datei, bezogen auf die unkomprimierte Datei.

| Name | PR0 | Worst-Zip | Best-Zip | Worst-Zip, PR0 nachbearbeitend |
|---|---|---|---|---|
| bonsai | 58 | 50 | 47 | 50 |
| brain | 58 | 57 | 55 | 53 |
| engine | 46 | 40 | 38 | 34 |
| head | 52 | 51 | 50 | 44 |
| crt-2 | 56 | 49 | 48 | 45 |
| average | 54 | 49 | 48 | 45 |
| funcbrain | 59 | 49 | 46 | 46 |

***Tabelle 8.2:*** *Vergleich der Komprimierungsraten nach Jensen et al. (2005)*

Tab. 8.3 spezifiziert die Dauer zum Komprimieren/Dekomprimieren. Die Zeiten von `funcbrain` habe ich auf ganze Sekunden gerundet.

### 8.1.4 Diskussion

Tab. 8.2 und 8.3 spezifizieren, dass PR0 im Vergleich zum Standardprogramm gut abschneidet. PR0 gibt fast dieselbe Komprimierungsrate wie Zip. Die Dekomprimierungsdauer ist proportional zur Dateigröße. Je besser die Komprimierung ist, desto schneller ist das Dekomprimieren. Das wird durch Best-Zip gezeigt, das generell eine sehr gute Komprimierungsrate gibt, lange zum Komprimieren und kurz zum Dekomprimieren braucht. Best-Zips Leistungseinbruch beim Komprimieren von `crt-2` und `funcbrain` führe ich darauf zurück, dass dort das Verwalten des Dictionaries in LZW

| Name | PR0 | Worst-Zip | Best-Zip | Worst-Zip, PR0 nachbearbeitend |
|---|---|---|---|---|
| bonsai | 1,8 / 1,5 | 0,3 / 1,0 | 3,7 / 1,0 | 0,7 / 0,3 |
| brain | 1,7 / 2,0 | 2,3 / 1,0 | 5,0 / 1,0 | 1,7 / 1,3 |
| engine | 1,5 / 1,2 | 1,0 / 0,7 | 3,7 / 1,0 | 1,0 / $< 0,1$ |
| head | 1,7 / 2,0 | 2,3 / 0,7 | 4,0 / 1,0 | 1,3 / 1,0 |
| crt-2 | 2,9 / 3,7 | 2,0 / 3,7 | 19,7 / 2,3 | 4,0 / 1,7 |
| average | 1,9 / 2,1 | 1,6 / 1,4 | 7,2 / 1,3 | 1,7 / 0,9 |
| funcbrain | 49,0 / 43,0 | 39,0 / 53,0 | 215,0 / 42,0 | 30,0 / 22,0 |

***Tabelle 8.3:*** *Vergleich der Ausführungsdauer nach Jensen et al. (2005)*

zuviel Zeit braucht und es das 16-Bit-Format der Voxelstreams und die Schnittflächenfolge nicht erkennt. PR0 behandelt sie, per Definition, optimal.

## 8.2 T2 – Volumenreduktion für haptisches Rendering

### 8.2.1 Aufbau

Die folgenden Rechner wurden über ein lokales 100 Mbit/sec Ethernet verbunden:

- Der Generator-Client war ein Prozeß auf dem HLRN (HLRN 2003). Der HLRN war ein Hochleistungs-Rechencluster mit 1024 Prozessoren. Es bestand aus 24 Knoten mit je 32 PowerPC-Prozessoren (PU), einem durchschnittlichen Speicherplatz von 64 GByte und insgesamt 26 TByte Festplattenplatz. Das Rechencluster arbeitete mit einer peak-performance von 2 TFlop/sec (HLRN 2003). Eine Hälfte des HLRN stand im Regionalen Rechenzentrum für Niedersachsen (RRZN), die andere im ZIB. Gerechnet wurde nur auf der RRZN-eigenen Hälfte. Als Betriebssystem diente AIX.
- Der Streaming-Server war ein Pentium III mit 0,8 GHz, 256 MByte Speicher und 254,9 GByte Festplattenplatz, auf dem Redhat Linux 3.2.3-47 lief.
- Der haptische Renderer war der erwähnte Arbeitsplatzrechner, an dem ich die Phantom bediente. Der Netzwerk-Adapter war ein eingebauter Fast Ethernet Controller des Typs 3C905C-TX-kompatibel von 3COM.

TCP wurde statt UDP genutzt, um keine Bitfehler beim Dekomprimieren auftreten zu lassen. Für den Rückkanal nutzte ich den seit DSVR-1 bestehenden Steuerungskanal, obwohl ein neuer Kanal mit UDP geeigneter gewesen wäre. Dennoch erwies sich die TCP-Verbindung für prototypische Tests als brauchbar. Für beide Kanäle habe ich die Socketbuffer-Größe bei 262144 Byte und `NODELAY` bei 0 belassen.
Die Entwicklungswerkzeuge waren MS Visual Studio .Net 2003, Ghost SDK 3.1, Ghostwrap und die Standard-Gnu-C-Compiler von den Cygwin- und Linux-Installationen.

### 8.2.2 Durchführung

Für den Test habe ich Clipping, Komprimierung und 4-Bit-Quantisierung auf unterschiedliche Weise kombiniert. Ich variierte die Volumenauflösung von $16^2 \cdot 8$ bis $256^2 \cdot 128$ gleichmäßig angeordnete Voxel. Ich maß die Updates/sec auf dem Renderer.

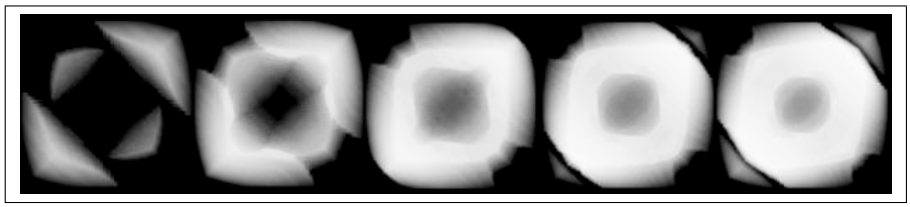

***Bild 8.1:*** *Ein Beispiel für eine mit T2 verarbeitete Volumenfolge; gezeigt sind die Frames für t=9, 19, 29, 39 und 48*

Als Beispiel ließ ich einen Generator-Client Volumen erzeugen, von denen einige Frames auf Bild 8.1 sind. Das Bild zeigt ein direktes visuelles Rendering der Cayley-Oberfläche, die durch $g(x, y, z) = t/4(x^2 + y^2 + z^2) + txyz$ definiert und über die Zeit $t = [0..48]$ moduliert wird. Für jedes $t$ wurde $g$ diskretisiert, für $40^3$ Voxel evaluiert (um das Generieren zu Beschleunigen) und auf die Auflösung von $256^2 \cdot 128$ Voxel resamplet. Die akkumulierte Datengröße betrug 392 MByte (unkomprimiert), 151 MByte (komprimiert mit PR0 ohne Lorenzo-Prädiktor) und 117 MByte (quantisiert, komprimiert). Der Transfer zum Render-Client erfolgte auf dem bekannten Wege. Ich schaltete das grafische Rendering auf demselben Prozessor zu. Alternativ nimmt man dazu einen weiteren CPU-Core oder zusätzlichen Rechner, von dem die Bilder, zum Beispiel mit VNC (RealVNC 2002), gesendet werden.

### 8.2.3 Ergebnisse

Ich untersuche primär die Update-Rate, weil mein Ansatz so konfiguriert werden kann, dass er die Frame-Rate nicht beeinträchtigt. Das habe ich beim Entwurf des Codecs, Kap. 5.2.3, erläutert.

| $k$ | Größe (in Voxel) | ups | Min. ups | Max. ups |
|---|---|---|---|---|
| 16 | $16^2 \cdot 8$ | 27,8 | 25,1 | 32,3 |
| 32 | $32^2 \cdot 16$ | 22,3 | 20,8 | 23,3 |
| 64 | $64^2 \cdot 32$ | 12,7 | 12,5 | 13,0 |
| 128 | $128^2 \cdot 64$ | 3,7 | 3,5 | 3,9 |
| 256 | $256^2 \cdot 128$ | 1 | 1 | 1 |
| 16 | $16^2 \cdot 8$ | 12,8 | 11,8 | 14,1 |
| 32 | $32^2 \cdot 16$ | 13,6 | 13,2 | 14,1 |
| 64 | $64^2 \cdot 32$ | 10,5 | 10,5 | 10,5 |
| 128 | $128^2 \cdot 64$ | 3,9 | 3,7 | 4,2 |
| 256 | $256^2 \cdot 128$ | 0,6 | 0,6 | 0,7 |
| 16 | $16^2 \cdot 8$ | 17,7 | 15,6 | 20,8 |
| 32 | $32^2 \cdot 16$ | 16,1 | 15,5 | 16,5 |
| 64 | $64^2 \cdot 32$ | 10,5 | 10,4 | 10,8 |
| 128 | $128^2 \cdot 64$ | 5,3 | 5,1 | 5,5 |
| 256 | $256^2 \cdot 128$ | 0,8 | 0,7 | 0,8 |

***Tabelle 8.4:*** *Updates/sec (ups) für einige Konfigurationen des Codecs T2*

Tab. 8.4 zeigt die gemessenen ups für einen Clipping-Faktor $k$ pro Zeile. Der erste Teil der Tabelle zeigt die ups wenn die Komprimierung und Quantisierung aus sind. Der zweite Teil zeigt die ups wenn die Komprimierung an und die Quantisierung aus ist. Der letzte Teil zeigt die ups wenn die Komprimierung und die Quantisierung an sind. Ich ermittelte die Zeiten dreimal, um das arithmetische Mittel, das Minimum und das Maximum zu finden.

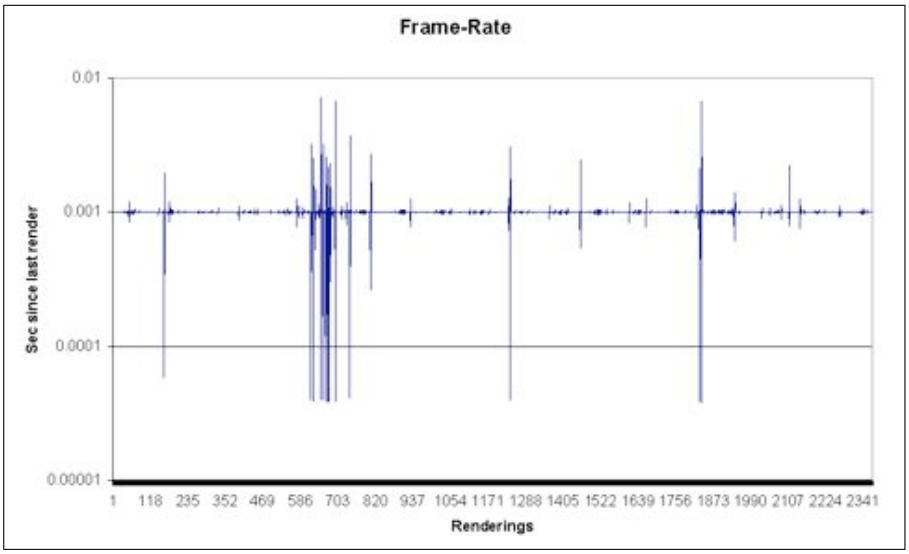


***Bild 8.2:*** *Mit* `gstTimer::secondsSinceLastQuery()` *gemessene Frame-Rate*

Bild 8.2 zeigt die erwartete durchschnittliche Frame-Rate von 973 Hz für den Clipping-Faktor $k = 64$ ohne Dekomprimierung. Die Anzahl der Abweichungen ist gering.

### 8.2.4 Diskussion

Die Ergebnisse zeigen, dass $k = 32..64$ das interaktive Empfangen von Geometriedaten im LAN unterstützt. Clipping verbessert die ups im Mittel, aber die zu kleinen Volumen beeinträchtigen die Update-Rate. Der Grund könnte das Delay zum Füllen der TCP-Pakete sein. Eine Lösung ist, `NODELAY` einzuschalten, das die Latenz reduziert, aber den Datendurchsatz nicht verbessert, weil die Pakete nicht voll ausgelastet gesendet werden. Weiterhin ist das anfällig gegenüber Jitter und einem langsamen Netzwerk. Ein Kompromiss ist, eine Volumengröße von $32 \leq k \leq 128$ zu wählen, welche durch die Bandbreite und der Anzahl der CPU-Cores bedingt wird.

Quantisierung verbessert die Performance, weil es zuverlässig, effizient und nur seitens des Generators ausgeführt wird. Komprimierung/Dekomprimierung bringt Mehraufwand, der nicht immer durch die gesparte Bandbreite im LAN kompensiert wird. Bei einer Bandbreite von 2 Mbit/sec und weniger ist nach meiner Schätzung dieser Fall aber noch gegeben.

## 8.3 T1 und T3 – Reduktion von 3-D-Texturen und Polygonnetzen

### 8.3.1 Aufbau

- Der Generator-Client war ein Prozessor auf dem HLRN und mit der Möglichkeit versehen, die 3-D-Texturen optional zu komprimieren. PR0-New ist PR0 mit eingeschaltetem Lorenzo-Prädiktor. Beide wurden in den Tests benutzt.

- Der Render-Client (ohne Haptik) war der bereits spezifizierte Arbeitsplatzrechner. Die Grafikkarte des Rechners war eine NVidia Quadro4 700 XGL mit AGP 4x, 350 MHz RAMDAC und 64 MByte DDR SDRAM. Der Arbeitsplatzrechner wurde mit Servicepack 2 bespielt und mit dem NVidia Grafiktreiber Release 6.14.10.5303 versehen. Die Bildauflösung war 1280x1024x32 bit.

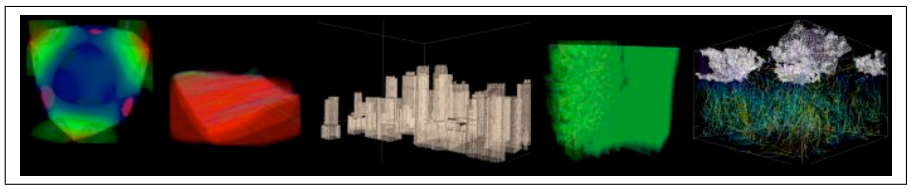

***Bild 8.3:*** *Ausgewählte Frames aus Testdaten verschiedenen Ursprungs*

Die Testdaten (Bild 8.3) wurden zumeist während der Tests erzeugt. Die Generatoren, bzw. Testdaten, stammen teils vom RRZN, vom Institut für Meteorologie und Klimatologie der Leibniz-Universität Hannover und vom Institut für technische Mechanik der Technischen Universität Clausthal.

- `demo4` war der bereits spezifizierte Cayley-Datensatz, der durch ein Fortran-90-Programm erzeugt wurde.

- Der Datensatz `vof` war ein 50 x 100 x 100 Voxel großes Volumen, das 10 Dateien enthielt, wovon jede 15 MByte einnahm und einen Zeitschritt repräsentierte. Die Daten wurden als einzige im Test vorher offline erzeugt. Hierzu kam der Strömungssimulator Star-CD zum Einsatz.

- `shinjuku` war ein Geoinformationsmodell eines Stadtteils von Tokio. Die Datenmenge der 180 x 180 x 100 Voxel betrug 3,23 MByte.

- Der Datensatz `example` war das Ergebnis des Simulierens von zeitveränderlichem Wärmefluß bei einer Auflösung von 40 x 40 x 40 Voxel. Die Simulationssoftware PALM agierte als dialogbetriebene Live-Datenquelle (Raasch und Schröter 2002).

- Der Datensatz `small` beschrieb in derselben Software eine atmosphärische Konvektion über 80 x 80 x 80 Voxel.

### 8.3.2 Durchführung

Eine Protokollierungs-Funktion von DSVR habe ich genutzt, um die Ausführungsdauer der Teile von T1 und T3, ohne PR0, zu messen. Das ist die Dauer für jeden Prozessor, ein Gebiet der ungefilterten Voxel zu resamplen, zu filtern und abzubilden (T1), bzw. mit Marching-Cubes (MC) und Vertex-Clustering (VC) auf Dreiecke abzubilden (T3). Die Protokollierungs-Funktion von DSVR speicherte die durchschnittliche, minimale und maximale Dauer pro Frame. Ich habe die Ergebnisse aus den Protokollen manuell extrahiert.
Die Ausführungsdauer für das Konvertieren der `vof`- in DVR-Dateien habe ich weggelassen. Konzentriert habe ich mich auf die Fälle, in denen die Visualisierung erzeugt, repliziert und parallel zum Generieren der Rohdaten betrachtet wurde. Die ungefilterten Voxel waren Skalare in 8-Byte Fließkommawerten.
Den Versuch, das Ergebnis einem nicht-verzahnten Vertex-Decimation-Algorithmus gegenüberzustellen (Jensen 1995), verwarf ich, weil der Vergleichs-Algorithmus die Volumen nur um 50% reduzierte und somit deutlich schlechter als mein Ansatz arbeitete. Die Arbeit von Manten (2006) war zum Zeitpunkt meiner Untersuchungen noch nicht verfügbar.

### 8.3.3 Ergebnisse

Tab. 8.5 spezifiziert die Ausführungsdauer für die Codecs T1 und T3. Die Simulationsergebnisse aus PALM verändern sich mit der Anzahl der Prozessoren, weil die Genauigkeit der Berechnung der Konvektionen mit dieser Zahl steigt. Dennoch ähneln sich die Features in den 3-D-Frames genug, um die Ergebnisse der Dauer und Datengröße untereinander vergleichen zu können.
Zum Beispiel lautet die erste Zeile („–” bedeutet „übersprungen”): `demo4` generierte einen Frame, lief auf vier PUs und generierte eine komprimierte 3-D-Textur mit 256 x 256 x 128 Voxeln, einem Byte pro Voxel. PR0 komprimierte die Daten auf etwas mehr als 3 MByte, PR0-New lag knapp unter dieser Datenmengengröße. Alle PUs resampleten die Sub-3-D-Texturen in 116 msec/PU, bevor sie der Sammler entgegennahm und an PR0 oder PR0-New weiterreichte. Alternativ generierte das System die Dreiecksnetze in 2 msec/PU und erzeugte 500 KByte, das sind circa 11250 Dreiecke. Clustergröße 1 halbierte die Menge und bedurfte eines Mehraufwands von 56 msec/PU. Der Trend setzte sich mit dem Erhöhen der Clustergröße $c$ fort.

| Name (Frames) | PU | PR0 | PR0-New | MC | MC und VC | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | $c$=1 | $c$=2 | $c$=3 | $c$=4 |
| demo4 (1) | 4 | – 3,13 | 115,99 2,78 | 2,01 0,53 | 58,21 0,38 | 67,30 0,15 | 76,95 0,10 | 166,58 0,09 |
| vof (10) | 2 | – – | – 2,29 | – 2,73 | – 0,85 | – 0,15 | – 0,06 | – 0,03 |
| shinjuku (1) | 4 | – – | – – | 32,31 3,86 | 736,01 2,8 | 710,52 0,75 | 743,38 0,32 | 782,82 0,18 |
| example (30) | 1 | – – | 318,80 2,41 | – – | 450,04 – | 594,55 – | 511,80 – | 5621,71 – |
| | 4 | – – | 89,44 2,41 | 6,13 3,21 | 69,70 1,17 | 115,27 0,16 | 106,81 0,04 | 5889,98 0,02 |
| | 8 | – – | 45,67 2,41 | – – | – – | – – | – – | – – |
| | 10 | – – | 34,04 2,41 | – – | – – | – – | – – | – – |
| small (7200) | 4 | – – | – – | 3,83 0,61 | 99,16 0,30 | 83,14 0,06 | 112,20 0,02 | – 0,01 |
| | 8 | – – | – – | – – | 38,13 0,31 | 32,27 0,05 | 58,78 0,02 | – – |
| | 16 | – – | – – | 1,46 0,64 | 22,05 0,27 | 20,45 0,05 | – – | – – |

***Tabelle 8.5:*** *Ausführungsdauer in msec/Frame und Datengröße in MByte/Frame*

### 8.3.4 Diskussion

Die Ergebnisse deuten auf eine lineare Skalierung der durchschnittlichen Ausführungsdauer für vier bis 16 Prozessoren hin. Das zeigen die Läufe mit `example` und `small`. PR0-New reduziert 8 MByte verlustlos auf 2,78 MByte (`demo4`), 2,29 MByte (`vof`) und 2,41 MByte (`example`). D. h. der Lorenzo-Prädiktor verbessert die Komprimierungsrate auf 1:3. Das kann ich auch mit dem Datensatz `bonsai` reproduzieren (nicht gezeigt). PR0-New kann ich unter Verschlechterung der o. g. Rate parallelisieren, indem ich es vor dem Sammeln starte. Die Verschlechterung tritt auf, weil dadurch mehr unkomprimierte Gebietsränder entstehen und ich auf kompensierende Nachrichtenaustausche verzichte.
Die Dreiecksnetz-Routinen skalieren nichtlinear mit der Clustergröße[2]. Die Messung mit `example` scheint darauf hinzudeuten, dass bei $c$=4 ein Parallelisieren die durchschnittliche Dauer ausdehnt. Das kann ich durch einen Blick auf die minimale Dauer (nicht in der Tabelle) jedoch nicht erhärten, weil diese wie erwartet gesunken ist. Das war somit mutmaßlich nur ein punktueller, Plattform-bedingter Leistungseinbruch.
Eine interessante Regelmäßigkeit sieht man beim Ändern der Clustergröße bei einer beliebigen festen Anzahl von PUs. Daten mit vielen Dreiecken zeigen eine Leistungserhöhung von c=2 auf c=3 und von c=2 auf c=1. Daten mit wenigen Dreiecken zeigen die Leistungserhöhung für die umgekehrten Fälle. Dass die größeren `Cluster` mehr Prozessorzeit brauchen, hat vermutlich zwei Gründe:

- Die MC-Implementierung berechnet für Gruppen dieselben Daten mehrfach, um dadurch eine vermeintliche Optimierung zu erzielen, die in Verbindung mit VC nicht mehr gegeben ist.

- die VC-Implementierung vermeidet nicht das Sortieren von Ecken.

Nutzer können nicht alle Clustergrößen beliebig anschalten, weil für jede Gruppe von `Clustern` exakt ein Prozeß zuständig sein muß und er keine Informationen darüber weiterleitet. Der Verzicht auf Nachrichtenaustausche hat den Vorteil, die Skalierbarkeit zu verbessern.
Die Ausgabequalität (Bilder B.2–B.5) ist verhältnismäßig hoch, insbesondere weil jeder Frame in höchstens wenigen Sekunden erzeugt wurde. Der Ansatz scheint die Topographie und die Topologie zu schonen. Um die Topologie weiter zu verbessern, ist es durch meinen Ansatz möglich, aus Marching-Cubes die Kubus-Konfigurationen zu importieren und nach leeren und gefüllten Kuben vorzusortieren. Der einfachste Fall wäre, leere Kuben als Barrieren zu verwenden, durch die niemals Kanten gespannt werden. Allerdings erhöht sich dadurch die Kopplung zu Marching-Cubes (Manten 2006). Mit meiner derzeitigen Implementierung könnte auch Marching-Tetrahedra abgefragt werden, ohne das Vertex-Clustering zu ändern.
Der Vergleich direkter und indirekter Volumenvisualisierung deutet auf die Speichereinsparnis durch den Einsatz von Äquipotenzialflächen hin. Diese resultiert in der Unterstützung höherer Update-Raten. Ausnahmen liegen dort vor, wo die Geometrien mit vielen Faltungen mehr Speicher brauchen, als die komprimierten 3-D-Texturen. Das Beispiel `vof` dokumentiert eine solche Ausnahme.
Insgesamt stützen die Resultate die Behauptung, dass meine entwickelten Codecs effektiv, robust, generell einsetzbar und ausreichend effizient sind, um in einem Perzeptualisierungs-System eingesetzt werden zu können. Ich habe Optimierungen zum wei-

[2] Ich meine stets Vertex-Cluster, nicht Rechencluster

teren Verbessern der Ausführungsdauer angeregt und somit eine Perspektive für die Erweiterung, Pflege und den Einsatz meiner Techniken aufgezeigt.

## 8.4 Abschließende Bemerkungen

Die Performance der Codecs T1–3 genügt den in Kap. 1 notierten technischen Anforderungen. Weiterhin habe ich den praktischen Einsatz mehrfach unter Einsatz verschiedener Ursprungsdaten und System-Konfigurationen demonstriert.

# Kapitel 9

# Abschluß und Ausblick

*„Glück heißt, seine Grenzen kennen und sie lieben."* (Romain Rolland)

Das Kapitel faßt den erreichten Arbeitsstand zusammen und gibt Anknüpfungspunkte für zukünftige Vorhaben.

## 9.1 Ziele und Erreichtes

Ich habe die gesteckten Ziele erreicht:

1. Der neue Komprimierer und Dekomprimierer PR0-New nutzt Prädiktoren und verarbeitet speziell Volumendaten in $O(n)$ auf verlustlosem Wege.

2. Ein vorgeschalteter Quantisierer reduziert jeden Wert um vier Bit.

3. Die Studie nach 3-IFC hilft zu zeigen, dass Nutzer die Quantisierung nicht bemerken, wenn sie die Volumen ertasten.

4. Der neue Dreiecksnetz-Vereinfacher läuft in $O(n)$ auf topologieschonendem Wege.

5. Die Techniken sind modul-, integrations-, system- und lastgetestet. Für die Tests habe ich Datensätze mittels DSVR-2 exploriert, die aus realen Anwendungsfeldern stammen.

## 9.2 Forschungsergebnisse

Das Erreichen meiner Ziele bestätigt die Hypothesen, und ich beantworte damit die Forschungsfragen:

- Die in dieser Arbeit angesprochenen Techniken sind zur Optimierung eines Perzeptualisierungs-Systems notwendig und hinreichend.

- Folgende Beobachtungen scheinen bzgl. der subjektiven Wahrnehmungserhaltung zu gelten:

- Nutzer einer Phantom 1.5A nehmen Änderungen von mehr als vier Bit pro Voxel durch ein Haptisierungs-System wahr, wenn es die Voxel als Viskosität rendert.
- Für die grafische Qualität von Polygonnetzen ist die Topologie, neben der Topographie, wichtig. Wenn die Topologie zerstört wird, gibt es spätestens ab einer Vereinfachung von 30% sichtbare Artefakte, zumindest wenn Vertex-Clustering ohne weitere Qualitätsmetriken eingesetzt wird. Dennoch können deutlich stärker vereinfachte Polygonnetze dem Nutzer noch Anhaltspunkte über die Existenz, Größe, Anzahl und Relation von Objekten geben. Für diesen Zweck genügt, abhängig von der Anzahl der Objekte, weniger als 1% der Ursprungsmenge.

## 9.3 Lernergebnisse

Während meiner Forschungsvorhaben habe ich gelernt, wie man

- neue Forschungsaufgaben identifiziert,
- Lösungen entwickelt, indem man sie auf bekannte Ansätze reduziert und
- das Erproben von Ansätzen systematisch priorisiert.

Für die Arbeit war es erforderlich,

- Fortran zu lernen,
- mich mit DSVR vertraut zu machen,
- Wissen über die Psychophysik und das Gestalten von Nutzerstudien anzueignen,
- Algorithmen zur Polygonnetz-Vereinfachung zu taxonomieren und
- Anwendungsfelder des Data-Mining zu erschließen.

## 9.4 Weitere Arbeiten

Quantisierte Dateien kann man mit PR0-New nachbearbeiten und dadurch Speicher sparen. Die zweifach komprimierten Dateien mit LZW zu reduzieren bringt jedoch nochmals eine deutliche Einsparung mit sich. D. h. PR0-New muß optimiert werden, quantisierte Volumen zu reduzieren. Man kann im Dictionary Bit-Packing durchführen, um leere Bits zu entfernen.
Forscher müssen meine Studie zum haptischen Rendern mit Geräten anderer Hersteller reproduzieren und zur Anwendung bringen. Für den Einsatz in Perzeptualisierungs-Systemen kommen nur Geräte mit sechs Freiheitsgraden und mehreren End-Effektoren infrage. Geräte mit weniger Freiheitsgraden oder End-Effektoren sind nach meiner Erfahrung zu beschränkt, um damit Anwendungsfelder erschließen zu wollen.
Die Effizienz des Verbundes zwischen Marching-Cubes und Vertex-Clustering muß verbessert werden, indem man

- redundante Berechnungen entfernt,
- die Daten so berechnet, dass sie automatisch vorsortiert sind,

- topologisch unzulässige Kanten verbietet und
- das Verfahren so ändert, dass es die Dichte und Krümmung der Dreiecksnetze beim Reduzieren berücksichtigt.

Kap. 1 hat Techniken spezifiziert, um die Latenz herabzusetzen. Entwickler müssen diese Techniken in DSVR implementieren und für Anwendungen auf GBit/sec-Netzen evaluieren.
Ich habe in meiner Arbeit nur die Volumenperzeptualisierung thematisiert. Forscher müssen die diskutierten Ansätze für den Einsatz mit zeitveränderlichen 3-D-Vektorfeldern erweitern.

## 9.5 Spekulation und Ausblick

Die Architektur von DSVR-2 liegt *allen* zur Perzeptualisierung geeigneten Systemen zugrunde. Hierbei steht sie abseits dessen, was Forscher meines Fachgebietes traditionell mit den Begriffen „virtuelle Umgebungen" und „virtuelle Realität" verbunden haben, aber bei näherer Erprobung als zu anwendungsspezifisch identifiziert wurde. Hingegen erkennt man bei der *anwendungsunabhängigen* Architektur von DSVR, dass sie mit den steigenden Bedarfen an die räumliche, zeitliche und qualitative Auflösung von Perzeptualisierungen skalieren wird:

- Keine Beschränkung auf bestimmte Grafikprimitive oder ganze Figuren
- Verfeinerte Gitterauflösungen in den Eingabedaten durch das Anschließen von mehr Prozessoren
- Durchgehende Perzeptualisierung von neuen Daten im Takt von 100–800 Hz durch das Einsetzen von GBit–TBit/sec-Netzwerken
- Ersetzen der punktbasieren haptischen Geräte durch motorgeregelte Handschuhe, deren Oberflächen von Netzen von End-Effektoren überdeckt sind
- Instanzierung einer Auralisierungs-Pipeline
- Instanzierung einer Olfaktorisierungs-Pipeline

Diesbezüglich sehe ich meine Arbeit als einen Beitrag zum generellen Bestreben des Menschen, die Welt in Modellform besser zu erfassen, als wir es bislang vermögen.

# Anhang A

# Ergänzender Entwurf

## A.1 Statik

### A.1.1 Pakete

Der Coder wird ins bestehende Generator-Client-Paket eingefügt (DVRP). Der Decoder wird in das Render-Client-Paket eingefügt (DocShow-VR).

## A.2 Dynamik

### A.2.1 Aktivitäten

**F4 – Synchrones Abspielen** Ich erweitere den Generator-Client um die Fähigkeit, einen Prozessor pro Szene zu belegen, der sie sammelt und versendet (Jensen 2005). Für den häufigsten Fall von zwei Szenen alloziere ich je einen Sammler pro MPI-pool:

```
...
if (myid == 0) { // dispatcher and gatherer for scene 1
  color = 0;
  leader_rank = 1;
}
else if (myid == size - 1) { // gatherer for scene 2
  color = 0;
  leader_rank = 0;
}
else {
  color = 1;
  leader_rank = 0;
}
MPI_Comm_split(dvrp_all, color, myid, comm_split);
MPI_Comm_dup(*comm_split, &dvrp_comm);
MPI_Intercomm_create(*comm_split, 0, dvrp_all,
  leader_rank, 0, &dvrp_intercomm);
...
```

Das 2-Wege-Handshake hat auf dem Streaming-Server folgenden Aufbau:

```
playMaster {
 PLAYING = TRUE
 while (nr < lastFramenr) {
  prepareFrame(nr++)
  sendFrameTo(remote clients)
  sendFramenrTo(dispatcher)
  waitForAckFrom(dispatcher)
 }
 PLAYING = FALSE
}
dispatcher {
 while (TRUE) {
  receiveFramenrFrom(playMaster)
  for each playSlave s {
   sendFramenrTo(s)
  }
  for each playSlave s {
   receiveAckFrom(s)
  }
  sendAckTo(playMaster)
 }
}
playSlave { // run by n processes
 while (PLAYING) {
  prepareFrame(receiveFramenrFrom(dispatcher))
  sendFrameTo(remote clients)
  sendAckTo(dispatcher)
}}
```

## A.3 Parallelisierung

Ich behalte die Daten- und Funktionsparallelisierung von DSVR-1 bei.

```
Generator-Client {
 extern float source[]
 extern float extent[]
 ...
 ubyte blob[]
 StreamManager streamout("ceci.n'est-pas-une-url.eu/par.dvrs")
 StreamManager split("mpi://localhost/cpu/0")
 Metaphor metaphor
 ...
 in parallel {
  metaphor.setRawData(source)
  metaphor.filter(extent, ...)
  metaphor.map()
  blob = metaphor.getBinaryLargeObject()
  split.writeBinaryLargeObject(MPI_PID, blob)
 }
```

```
 int i = 0;
 while ((blob = split.readBinaryLargeObject(i))) {
  metaphor.setBinaryLargeObject(i++, blob)
 }
 metaphor.writeToStream(blob)
 streamout.marshal(blob)
 streamout.writeBinaryLargeObject(1, blob)
}
```

## A.4 Komplexitätsanalyse

Die Zeit- und Speicherkomplexität ist für alle Algorithmen, sofern nicht anders angegeben, $O(n)$. Für die Analyse des Vertex-Clustering setze ich $n$ als die Anzahl der Eingabewerte voraus, und $c$ proportional zum Aufwand, pro Clustergruppe die Ecken längs einer Dimension zu traversieren. Vor einen Code-Block setze ich dessen Komplexität ($T$). Zeilen mit $T(1)$ entferne ich teilweise vorab, um den Code übersichtlicher zu machen.

```
T_all:
for each volume in sequence {
 for each group in volume {
  for each cluster in group {
   for each cube in cluster {
    ...
    T(3 * c): marching cubes({x,y,z}..cluster limit)
    T(3 * c): for each triangle
      push vertices on stack v
  }}}
  T_commit: commit_clustered_polys(...)
}}

commit_clustered_polys(PolygonData *pd,group_t *g,
                       int groupnumber,
                       int xdim,int ydim,int zdim)
{
 T(3 * c):
 while (!is_empty(&g[groupnumber].vertexstack)) {
  retrieve_polygon(&g[groupnumber],
   p[j],p[j+1],p[j+2]);
  for i=j..j+2 {
   ...
  }
  ...
 }
 ...
 T(3 * c):
 if (polys >= 1) {
  ...
  T(3 * c): // copy the vertices
  ...
```

```
 }
 ...
 T(3 * c^2): sort_vector(g[groupnumber].maxx, x, z);
 ... same for y, z [skipped] ...
 T(3 * c): remove_doubles(g[groupnumber].maxx, x+3);
 ... same for y, z [skipped] ...
 T(3 * c^2 + 3 * c):
  ...
  T(3 * c^2):
  sort_vector(g[PREV_X(groupnumber,
   xdim,ydim,zdim)].maxx,xm,z);
  ... same for y, z [skipped] ...
  T(3 * c):
  remove_doubles(g[PREV_X(groupnumber,
   xdim,ydim,zdim)].maxx,xm+3);
   ... same for y, z [skipped] ...
 }
 T(3):
 remesh(pd, g[PREV_X(groupnumber,
  xdim,ydim,zdim)].maxx,cp,corners);
 remesh(pd, g[groupnumber].maxx, cp, corners);
 ... same for y, z [skipped] ...
}}
```

Der Gesamtaufwand für `commit_clustered_polys()` beträgt

$$\begin{aligned} T_{commit} &= T(3 \cdot c) + T(3 \cdot c) + \\ &\quad T(3 \cdot c^2) + T(3 \cdot c) + T(3 \cdot c^2 + 3 \cdot c) + T(3) \qquad \text{(A.1)} \\ &= O(3 \cdot c^2 + 3 \cdot c) \\ &= O(c^2) \qquad \text{(A.2)} \end{aligned}$$

Der Gesamtaufwand für den kompletten Algorithmus ist

$$\begin{aligned} T_{all} &= T(n) \cdot T(6 \cdot c) \cdot T_{commit} \qquad \text{(A.3)} \\ &= T(n) \cdot O(c^2) \\ &= O(n) \qquad \text{(A.4)} \end{aligned}$$

Der best-, worst- und average-case unterscheiden sich im Wert für $c$:

- Best: $c = 1$
- Worst: $c \geq n$ (degenerierte Fälle für kleine $n$)
- Average: $c < n$

Der average-case ist derjenige, für den die gewonnenen Polygone zusammen weniger Speicher belegen als die Rohdaten, aus denen sie erzeugt werden. Tab. 8.5 belegt dies als den Normalfall während der Tests.

# Anhang B

# Ergänzende Testdaten

## B.1 Trefferquote der Prädiktoren in PR0

Protokoll der Häufigkeiten beim Bearbeiten von `bonsai`, ohne Lorenzo-Prädiktor:

| P0 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | Zeilensumme |
|---:|---:|---:|---:|---:|---:|---:|---:|---:|
| 64516 | 0 | 0 | 0 | 0 | 0 | 0 | 1020 | 65536 |
| 129032 | 0 | 0 | 0 | 0 | 0 | 0 | 2040 | 131072 |
| 193548 | 0 | 0 | 0 | 0 | 0 | 0 | 3060 | 196608 |
| 193548 | 0 | 0 | 0 | 0 | 0 | 55560 | 13036 | 262144 |
| 231311 | 9750 | 2122 | 1705 | 738 | 481 | 61727 | 19846 | 327680 |
| 264410 | 21076 | 5123 | 3809 | 1845 | 1171 | 69404 | 26378 | 393216 |
| 300177 | 31139 | 7698 | 5956 | 3184 | 1788 | 76953 | 31857 | 458752 |
| 336014 | 40835 | 10076 | 7737 | 3942 | 2476 | 83687 | 39521 | 524288 |
| 370240 | 51926 | 13159 | 9892 | 5146 | 3229 | 90555 | 45677 | 589824 |
| 407186 | 61515 | 15465 | 12069 | 6543 | 3860 | 97154 | 51568 | 655360 |
| 441336 | 71358 | 17915 | 14053 | 7432 | 4413 | 105234 | 59155 | 720896 |
| 478174 | 81100 | 20900 | 16000 | 8882 | 5273 | 111754 | 64349 | 786432 |
| 516115 | 90237 | 23200 | 17788 | 9854 | 5890 | 117850 | 71034 | 851968 |
| 549237 | 101101 | 26226 | 19847 | 10860 | 6623 | 125271 | 78339 | 917504 |
| 584738 | 110259 | 28898 | 21713 | 12526 | 7440 | 133152 | 84314 | 983040 |
| 619606 | 119728 | 31259 | 23525 | 13385 | 8152 | 140559 | 92362 | 1048576 |
| 652172 | 130960 | 34527 | 25674 | 14801 | 8959 | 148764 | 98255 | 1114112 |
| 687555 | 140406 | 36847 | 27544 | 15808 | 9595 | 156583 | 105310 | 1179648 |
| 718988 | 150818 | 39801 | 29673 | 16754 | 10435 | 165614 | 113101 | 1245184 |
| 752423 | 160947 | 43264 | 31579 | 18073 | 11228 | 174146 | 119060 | 1310720 |
| 784497 | 170066 | 45608 | 33220 | 18851 | 12034 | 183098 | 128882 | 1376256 |
| 811590 | 181861 | 48884 | 35522 | 20358 | 13330 | 193134 | 137113 | 1441792 |
| 841493 | 191301 | 51796 | 37319 | 21547 | 14172 | 204336 | 145364 | 1507328 |
| 869274 | 200436 | 54406 | 39370 | 22533 | 15241 | 215030 | 156574 | 1572864 |
| 892577 | 210870 | 57905 | 41501 | 24779 | 16518 | 228875 | 165375 | 1638400 |
| 920011 | 220011 | 60368 | 43177 | 25849 | 17525 | 240168 | 176827 | 1703936 |
| 942834 | 229839 | 63269 | 45437 | 27386 | 19041 | 252378 | 189288 | 1769472 |
| 965725 | 239033 | 66785 | 47419 | 29664 | 20397 | 266489 | 199496 | 1835008 |
| 988544 | 247244 | 68887 | 48972 | 30980 | 21577 | 279752 | 214588 | 1900544 |
| 1007797 | 257666 | 72144 | 51386 | 33543 | 23827 | 292960 | 226757 | 1966080 |
| 1030669 | 266605 | 75285 | 53254 | 35623 | 25213 | 306605 | 238362 | 2031616 |
| 1053218 | 275174 | 77677 | 55226 | 37007 | 26921 | 318069 | 253860 | 2097152 |

*Fortsetzung folgt...*

*...Fortsetzung*

| P0 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | Zeilensumme |
|---|---|---|---|---|---|---|---|---|
| 1072263 | 284519 | 81065 | 57611 | 40019 | 28872 | 332717 | 265622 | 2162688 |
| 1095468 | 292921 | 83409 | 59439 | 41587 | 30228 | 345431 | 279741 | 2228224 |
| 1115787 | 302228 | 86123 | 61971 | 43467 | 32419 | 357866 | 293899 | 2293760 |
| 1136327 | 311090 | 89454 | 64140 | 46223 | 34173 | 372381 | 305508 | 2359296 |
| 1157031 | 318913 | 91564 | 65832 | 47588 | 35648 | 385578 | 322678 | 2424832 |
| 1174246 | 327945 | 94523 | 68260 | 50236 | 38420 | 399443 | 337295 | 2490368 |
| 1193961 | 335987 | 97467 | 70286 | 52682 | 40283 | 413906 | 351332 | 2555904 |
| 1212428 | 343446 | 99751 | 72151 | 54215 | 42206 | 427262 | 369981 | 2621440 |
| 1228099 | 351772 | 102744 | 74315 | 57825 | 44528 | 443118 | 384575 | 2686976 |
| 1247544 | 359545 | 105195 | 75972 | 59679 | 46173 | 456698 | 401706 | 2752512 |
| 1263784 | 367187 | 107837 | 77988 | 61882 | 48573 | 471161 | 419636 | 2818048 |
| 1280169 | 374220 | 110577 | 79970 | 65243 | 50500 | 488194 | 434711 | 2883584 |
| 1297957 | 381021 | 112740 | 81545 | 67210 | 52216 | 502047 | 454384 | 2949120 |
| 1313070 | 388317 | 115391 | 83823 | 70574 | 54989 | 517508 | 470984 | 3014656 |
| 1331924 | 395878 | 118027 | 85652 | 72975 | 56946 | 532150 | 486640 | 3080192 |
| 1349846 | 402995 | 120400 | 87489 | 74634 | 59188 | 544475 | 506701 | 3145728 |
| 1365252 | 409997 | 123164 | 89560 | 78167 | 61524 | 560527 | 523073 | 3211264 |
| 1383833 | 417035 | 125465 | 91080 | 80052 | 63578 | 573946 | 541811 | 3276800 |
| 1399342 | 423945 | 127851 | 93039 | 82184 | 66311 | 588206 | 561458 | 3342336 |
| 1415635 | 430652 | 130604 | 94961 | 85768 | 68632 | 603772 | 577848 | 3407872 |
| 1431968 | 436901 | 132565 | 96519 | 87541 | 70581 | 617589 | 599744 | 3473408 |
| 1445248 | 443604 | 135145 | 98788 | 90765 | 73690 | 633202 | 618502 | 3538944 |
| 1460548 | 450124 | 137754 | 100538 | 93604 | 75991 | 649863 | 636058 | 3604480 |
| 1475173 | 456052 | 140001 | 102157 | 95847 | 78054 | 665538 | 657194 | 3670016 |
| 1489067 | 463203 | 142917 | 104145 | 99534 | 80565 | 682402 | 673719 | 3735552 |
| 1506900 | 470568 | 145195 | 105940 | 101597 | 82408 | 696742 | 691738 | 3801088 |
| 1522685 | 478087 | 147740 | 108213 | 103889 | 85067 | 711153 | 709790 | 3866624 |
| 1539404 | 485956 | 150899 | 110388 | 107129 | 87404 | 726521 | 724459 | 3932160 |
| 1557394 | 493436 | 153266 | 112101 | 108981 | 89333 | 739838 | 743347 | 3997696 |
| 1573066 | 502212 | 156307 | 114646 | 112056 | 92359 | 753721 | 758865 | 4063232 |
| 1592137 | 510716 | 159236 | 116847 | 114738 | 94358 | 767896 | 772840 | 4128768 |
| 1611110 | 518772 | 161779 | 118969 | 116479 | 96501 | 780456 | 790238 | 4194304 |
| 1628058 | 527554 | 164972 | 121399 | 119968 | 98960 | 795153 | 803776 | 4259840 |
| 1649144 | 536490 | 167515 | 123280 | 121789 | 100844 | 807786 | 818528 | 4325376 |
| 1668042 | 545685 | 170419 | 125689 | 123793 | 103301 | 820429 | 833554 | 4390912 |
| 1687205 | 554512 | 173724 | 127763 | 126725 | 105270 | 835133 | 846116 | 4456448 |
| 1707610 | 562598 | 176017 | 129437 | 128262 | 106972 | 848244 | 862844 | 4521984 |
| 1725519 | 572548 | 178961 | 131937 | 131065 | 109677 | 861139 | 876674 | 4587520 |
| 1746387 | 581364 | 181945 | 133936 | 133411 | 111314 | 875204 | 889495 | 4653056 |
| 1767541 | 589753 | 184467 | 135956 | 135027 | 113056 | 888049 | 904743 | 4718592 |
| 1786286 | 599510 | 187606 | 138324 | 137994 | 114886 | 902475 | 917047 | 4784128 |
| 1808670 | 608204 | 189892 | 140138 | 139780 | 116268 | 916305 | 930407 | 4849664 |
| 1830172 | 617909 | 192576 | 142625 | 141630 | 118108 | 928884 | 943296 | 4915200 |
| 1851383 | 626888 | 195750 | 144526 | 144091 | 119492 | 944730 | 953876 | 4980736 |
| 1876464 | 635295 | 197836 | 146271 | 145438 | 120652 | 958352 | 965964 | 5046272 |
| 1900595 | 645597 | 200569 | 148698 | 147137 | 122090 | 970835 | 976287 | 5111808 |
| 1929409 | 655052 | 203560 | 150442 | 148640 | 123096 | 981675 | 985470 | 5177344 |
| 1959601 | 664003 | 206005 | 152063 | 149648 | 124269 | 990880 | 996411 | 5242880 |
| 1988045 | 674828 | 209034 | 153937 | 151072 | 125502 | 1000078 | 1005920 | 5308416 |
| 2020377 | 684772 | 211402 | 155642 | 152220 | 126632 | 1007343 | 1015564 | 5373952 |
| 2050612 | 694580 | 213970 | 157543 | 153465 | 127789 | 1016076 | 1025453 | 5439488 |
| 2082894 | 704680 | 217117 | 159116 | 154797 | 128891 | 1023842 | 1033687 | 5505024 |
| 2115681 | 714113 | 219411 | 160596 | 155840 | 130005 | 1031125 | 1043789 | 5570560 |
| 2145263 | 725110 | 222350 | 162647 | 157206 | 131160 | 1039451 | 1052909 | 5636096 |

*Fortsetzung folgt...*

| | | | | | | | | *...Fortsetzung* |
|---|---|---|---|---|---|---|---|---|
| P0 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | Zeilensumme |
| 2178078 | 735118 | 225155 | 164349 | 158470 | 132103 | 1047466 | 1060893 | 5701632 |
| 2211031 | 744614 | 227603 | 166077 | 159518 | 133096 | 1055009 | 1070220 | 5767168 |
| 2242147 | 755843 | 230805 | 167967 | 160837 | 134154 | 1062887 | 1078064 | 5832704 |
| 2276945 | 765937 | 233187 | 169566 | 161805 | 135146 | 1069362 | 1086292 | 5898240 |
| 2308984 | 775910 | 235857 | 171398 | 162881 | 136104 | 1077576 | 1095066 | 5963776 |
| 2343082 | 786348 | 239053 | 173058 | 164139 | 137005 | 1084556 | 1102071 | 6029312 |
| 2378587 | 795645 | 241266 | 174684 | 165084 | 137816 | 1091159 | 1110607 | 6094848 |
| 2410434 | 806716 | 244011 | 176673 | 166297 | 138769 | 1098924 | 1118560 | 6160384 |
| 2445209 | 816615 | 246883 | 178263 | 167543 | 139590 | 1106372 | 1125445 | 6225920 |
| 2479763 | 825996 | 249387 | 179888 | 168426 | 140423 | 1113329 | 1134244 | 6291456 |
| 2512112 | 837235 | 252599 | 181755 | 169747 | 141446 | 1120993 | 1141105 | 6356992 |
| 2548231 | 847232 | 255037 | 183425 | 170761 | 142250 | 1127507 | 1148085 | 6422528 |
| 2581398 | 857257 | 257704 | 185353 | 171809 | 143140 | 1135671 | 1155732 | 6488064 |
| 2616063 | 867948 | 261079 | 186876 | 173124 | 143961 | 1142963 | 1161586 | 6553600 |
| 2652776 | 877477 | 263355 | 188399 | 173936 | 144706 | 1149339 | 1169148 | 6619136 |
| 2685569 | 888637 | 266323 | 190507 | 175069 | 145609 | 1156912 | 1176046 | 6684672 |
| 2721582 | 898614 | 269145 | 192047 | 176275 | 146274 | 1164483 | 1181788 | 6750208 |
| 2758282 | 907864 | 271571 | 193594 | 177023 | 147027 | 1171040 | 1189343 | 6815744 |
| 2792219 | 919338 | 274945 | 195479 | 178251 | 147883 | 1178246 | 1194919 | 6881280 |
| 2830207 | 929314 | 277359 | 197003 | 179203 | 148502 | 1184376 | 1200852 | 6946816 |
| 2865102 | 939426 | 279955 | 198995 | 180185 | 149208 | 1192074 | 1207407 | 7012352 |
| 2901773 | 950134 | 283258 | 200651 | 181356 | 149847 | 1198826 | 1212043 | 7077888 |
| 2940182 | 959501 | 285487 | 202096 | 182098 | 150359 | 1204777 | 1218924 | 7143424 |
| 2973903 | 970961 | 288422 | 204214 | 183218 | 151062 | 1212383 | 1224797 | 7208960 |
| 3011472 | 981023 | 291434 | 205818 | 184298 | 151613 | 1219691 | 1229147 | 7274496 |
| 3049824 | 990761 | 293814 | 207337 | 184977 | 152150 | 1225834 | 1235335 | 7340032 |
| 3085619 | 1002447 | 297153 | 209148 | 186066 | 152853 | 1232407 | 1239875 | 7405568 |
| 3125638 | 1012382 | 299469 | 210611 | 186861 | 153368 | 1238049 | 1244726 | 7471104 |
| 3162250 | 1022598 | 301869 | 212358 | 187640 | 153890 | 1245581 | 1250454 | 7536640 |
| 3200570 | 1033392 | 305171 | 213846 | 188696 | 154500 | 1251819 | 1254182 | 7602176 |
| 3240674 | 1043000 | 307443 | 215213 | 189309 | 154911 | 1257257 | 1259905 | 7667712 |
| 3275524 | 1054345 | 310369 | 217143 | 190164 | 155604 | 1264731 | 1265368 | 7733248 |
| 3313579 | 1064468 | 313497 | 218576 | 191386 | 156108 | 1272078 | 1269092 | 7798784 |
| 3353216 | 1073994 | 315922 | 219986 | 192024 | 156592 | 1277861 | 1274725 | 7864320 |
| 3389762 | 1085683 | 319079 | 221799 | 192989 | 157231 | 1284356 | 1278957 | 7929856 |
| 3430387 | 1095599 | 321385 | 223322 | 193713 | 157650 | 1290021 | 1283315 | 7995392 |
| 3467460 | 1106067 | 323880 | 225042 | 194453 | 158141 | 1297367 | 1288518 | 8060928 |
| 3467460 | 1106067 | 323880 | 225042 | 196169 | 158141 | 1360167 | 1289538 | 8126464 |
| 3531976 | 1106067 | 323880 | 225042 | 196169 | 158141 | 1360167 | 1290558 | 8192000 |
| 3596492 | 1106067 | 323880 | 225042 | 196169 | 158141 | 1360167 | 1291578 | 8257536 |
| 3661008 | 1106067 | 323880 | 225042 | 196169 | 158141 | 1360167 | 1292598 | 8323072 |
| 44% | 13% | 4% | 3% | 2% | 2% | 16% | 16% | 100% |

## B.2 Bilder der Testdaten

S. nächste Seiten.

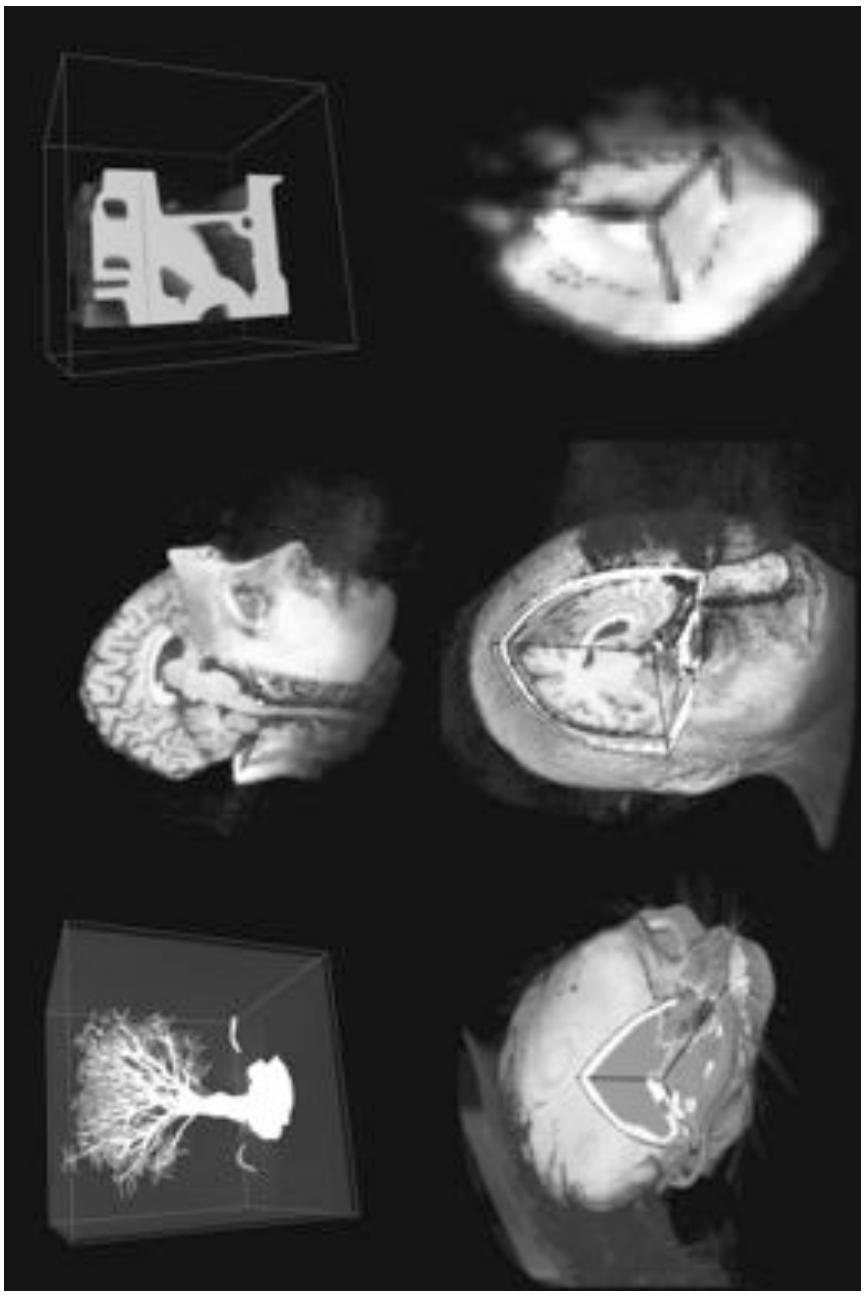

***Bild B.1:*** *Die Volumen für den Test; visualisiert mit* OpenQVis *und* MRIcro *(Jensen et al. 2005, Bild 1)*

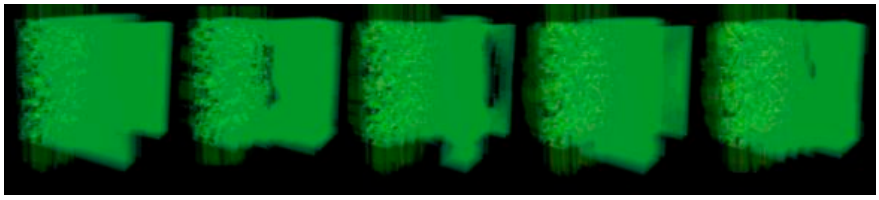

***Bild B.2:*** *Frames 2, 6, 12, 18 und 24 des Datensatzes* `example`*, die mit PR0-New verlustlos komprimiert und dekomprimiert wurden.*

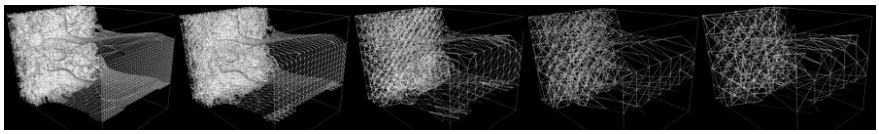

***Bild B.3:*** *Frame 2 des Datensatzes* `example`*, der mit Marching-Cubes und mit Vertex-Clustering bei Clustergröße 1–4 verarbeitet wurde.*

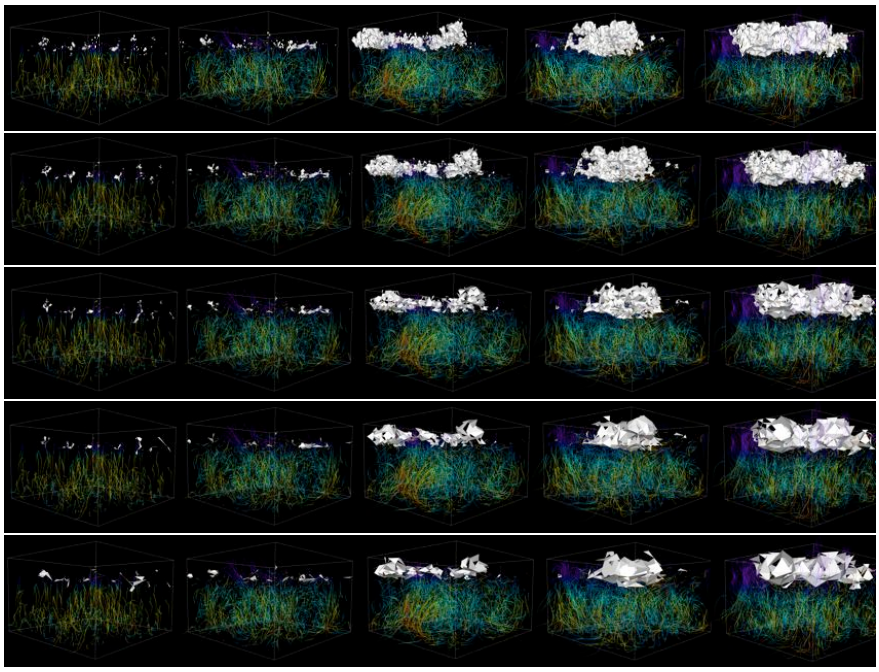

***Bild B.4:*** *Für jede Zeile: Frames 1000, 2000, 3000, 4000 und 5000 des Datensatzes* `small`*, die mit Marching-Cubes (zuoberst) und mit Vertex-Clustering in vier Clustergrößen verarbeitet wurden. Stromlinien sind als zweite Szene enthalten.*

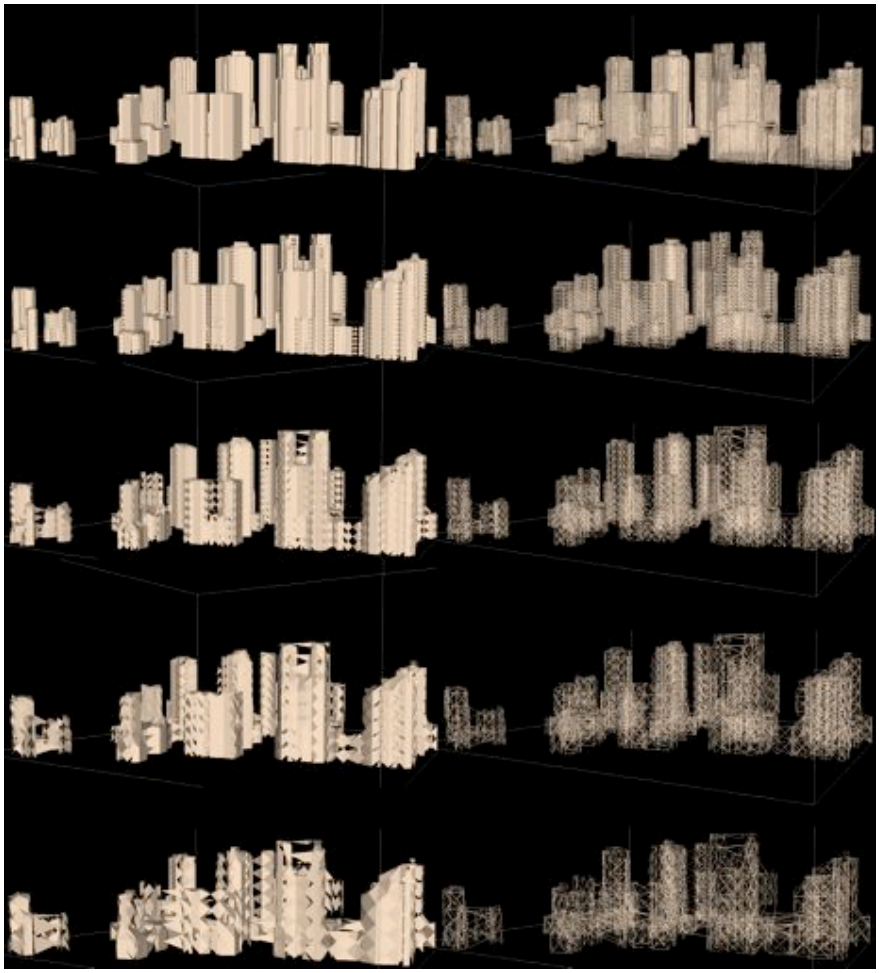

***Bild B.5:*** *Für jede Zeile: Der Frame* `buildings`*, der Teil von* `shinjuku` *ist, und zuoberst mit Marching-Cubes und mit Vertex-Clustering in vier Clustergrößen verarbeitet wurde. Die rechte Spalte zeigt das zugehörige Polygonnetz.*

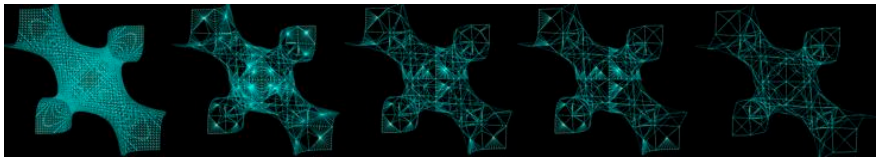

***Bild B.6:*** *Das Bild zeigt die Auswirkung des progressiven Abschaltens von Ecken an Cluster- und Prozeß-Grenzen beim Vertex-Clustering. Der rechteste Frame zeigt die Vernetzung, die der Algorithmus bei allen Beispielen genutzt hat.*

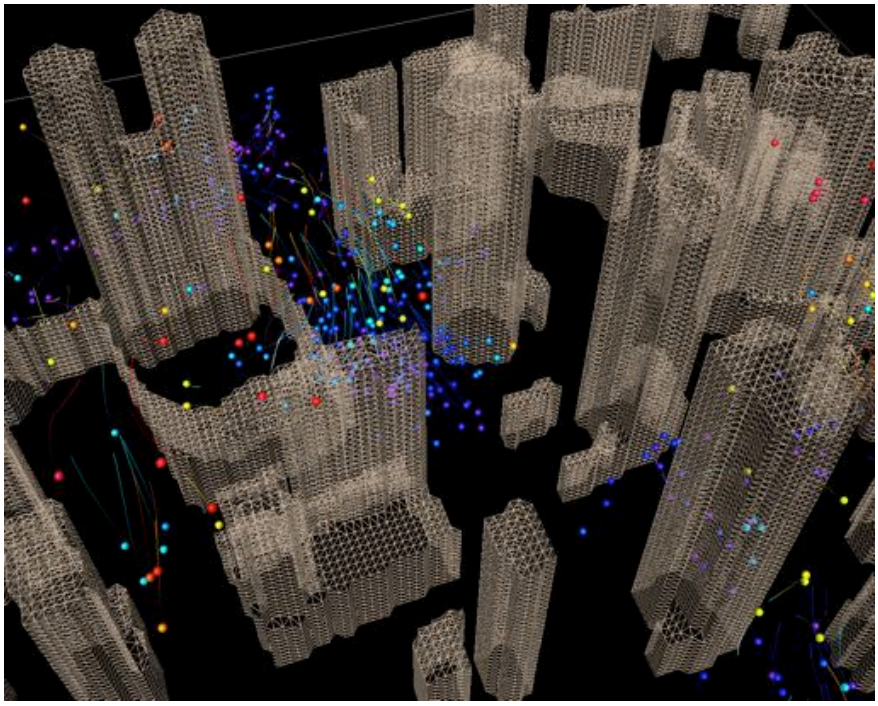

***Bild B.7:*** *Frame 1510 des Datensatzes* `shinjuku`*, in dem Partikelsimulationsergebnisse enthalten sind und bei dem das Gebäudemodell mit Marching-Cubes ohne Vertex-Clustering generiert wurde.*

# Referenzen


Amdahl, G.M. 2000. Readings in computer architecture, chapter Validity of the single processor approach to achieving large scale computing capabilities, 79–81. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Avila, R.S.; Sobierajski, L.M. 1996. A haptic interaction method for volume visualization. VIS '96: Proceedings of the 7th conference on Visualization '96, 197–ff., Los Alamitos, CA, USA. IEEE Computer Society Press.

Berger, M.J.; Colella, P. 1989. Local adaptive mesh refinement for shock hydrodynamics. Journal of Computational Physics, 82(1), 64–84.

Bethel, W.; Tierney, B.; lee, J.; Gunter, D.; Lau, S. 2000. Using high-speed WANs and network data caches to enable remote and distributed visualization. Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), 28, Washington, DC, USA. IEEE Computer Society.

Blythe, D. 2006. The Direct3D 10 system. SIGGRAPH '06: ACM SIGGRAPH 2006 Papers, 724–734, New York, NY, USA. ACM Press.

Booch, G. 1993. Object-oriented analysis and design with applications (2nd edition). Addison-Wesley Professional. 608 S.

Brodlie, K.; Wood, J. 2000. Recent advances in visualization of volumetric data. Eurographics State of the Art Reports, 64–84.

Brooks Jr., F.; Ouh-Young, M.; Batter, J.; Kilpatrick, P. 1990. Project GROPE – haptic displays for scientific visualization. SIGGRAPH 90, 177–185, New York, NY, USA.

Burdea, G. 1996. Force and touch feedback for virtual reality. Wiley. 339 S.

Cabral, B.; Cam, N.; Foran, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. VVS '94: Proceedings of the 1994 symposium on Volume visualization, 91–98, New York, NY, USA. ACM Press.

Card, S.; Mackinlay, J.; Shneiderman (Eds.), B. 2001. Readings in information visualization – using vision to think. Morgan Kaufman, San Francisco, CA. 686 S.

Carneiro, B.; Kaufman, A.E. 1996. Tetra-cubes: An algorithm to generate 3D isosurfaces based upon tetrahedra. Anais do SIBGRAPI, 205–210.

Charlton, E.F. Sutherland-Hodgman polygon clipper. Online: http://hpcc.engin.umich.edu/cfd/users/charlton/thesis/html/node90.html (Zugriff: 15. Jan. 2007), 2006.

Chen, B.Y.; Nishita, T. 2002. Multiresolution streaming mesh with shape preserving and QoS-like controlling. Web3D '02: Proceeding of the seventh international conference on 3D Web technology, 35–42, New York, NY, USA. ACM Press.

Chiueh, T.c.; Yang, C.k.; He, T.; Pfister, H.; Kaufman, A. 1997. Integrated volume compression and visualization. *In* Yagel, R.; Hagen, H., ed., IEEE Visualization '97, 329–336.

Cho, S.; Kim, D.; Pearlman, W. 2004. Lossless compression of volumetric medical images with improved three-dimensional SPIHT algorithm. Journal of Digital Imaging, 17(1), 57–63.

Ciampalini, A.; Cignoni, P.; Montani, C.; Scopigno, R. 1997. Multiresolution decimation based on global error. The Visual Computer, 13(5), 228–246.

Cohen, J.; Varshney, A.; Manocha, D.; Turk, G.; Weber, H.; Agarwal, P.; Brooks, F.; Wright, W. 1996. Simplification envelopes. Computer Graphics, 30(Annual Conference Series), 119–128.

Coplien, J.O. 1994. Advanced C++ programming styles and idioms. Addison-Wesley. 544 S.

Deutsch, J. Mass and spring. Online: http://physics.ucsc.edu/ josh/6a/book/harmonic/-node2.html (Zugriff: 19. Dez. 2006), 2006.

Durbeck, L.; Macias, N.; Weinstein, D.; Johnson, C.; Hollerbach, J. Scirun haptic display for scientific visualization. Phantom users group meeting, Dedham, MA, September 1998., 1998.

Earnshaw, R.A.; Wiseman, N. 1992. An introductory guide to scientific visualization. Springer, Berlin. 156 S.

Ebner, M. 5. Übungsblatt. Online: http://www2.informatik.uni-wuerzburg.de/staff/ebner/teaching/computergraphics/exercisesws2002/blatt5/-marchingcubes2.gif(Zugriff: 15. Jan. 2007), 2006.

Eco, U. 1984. Der Name der Rose. Hanser. 654 S.

El-Rewini, H.; Lewis, T.G. 1998. Distributed and parallel computing. Manning Publications Co., Greenwich, CT, USA. 447 S.

El-Sana, J.; Sokolovsky, N.; Silva, C.T. 2001. Integrating occlusion culling with view-dependent rendering. VIS '01: Proceedings of the conference on Visualization '01, 371–378, Washington, DC, USA. IEEE Computer Society.

Engel, K. Pre-integrated volume rendering – bonsai. Online: http://wwwvis.informatik.uni-stuttgart.de/~engel/bonsai.zip (Zugriff: 28. Jan. 2005), 2001a.

Engel, K. Pre-integrated volume rendering – engine. Online: http://wwwvis.informatik.uni-stuttgart.de/~engel/engine.zip (Zugriff: 18. Jan. 2007), 2001b.

Engel, K.; Kraus, M.; Ertl, T. 2001. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, 9–16, New York, NY, USA. ACM Press.

Engel, K.; Sommer, O.; Ernst, C.; Ertl, T. 1999. Remote 3D visualization using image-streaming techniques. Proceedings of the International Symposium on Intelligent Multimedia and Distance Education, 91–96.

Engelson, V.; Fritzson, D.; Fritzson, P. 2000. Lossless compression of high-volume numerical data from simulations. DCC '00: Proceedings of the Conference on Data Compression, 574, Washington, DC, USA. IEEE Computer Society.

Fechner, G. 1860. Elemente der Psychophysik. Breitkof und Hartel.

Foley, J.D.; van Dam, A.; Feiner, S.K.; Hughes, J.F. 1997. Computer graphics – principles and practice. Addison-Wesley, Reading, Massachusetts. 1175 S.

Fowler, J.E.; Yagel, R. 1994. Lossless compression of volume data. VVS '94: Proceedings of the 1994 symposium on Volume visualization, 43–50, New York, NY, USA. ACM Press.

Fraunhofer Institut für graphische Datenverarbeitung. Direkte Volumenvisualisierung von einem CT-Scan eines Motorblocks. Online: http://www.igd.fhg.de/igd-a2/projects/cfd/volvis_engine2.jpg (Zugriff: 15. Jan. 2007), 2006.

Garland, M.; Heckbert, P. 1997. Surface simplification using quadric error metrics. Computer Graphics, 209–216.

Gelder, A.V.; Wilhelms, J. 1994. Topological considerations in isosurface generation. ACM Transactions on Graphics, 13(4), 337–375.

Gerndt, A.; Hentschel, B.; Wolter, M.; Kuhlen, T.; Bischof, C. 2004. Viracocha: An efficient parallelization framework for large-scale CFD post-processing in virtual environments. SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, 50, Washington, DC, USA. IEEE Computer Society.

Gerstner, T.; Pajarola, R. 2000. Topology Preserving and Controlled Topology Simplifying Multiresolution Isosurface Extraction. *In* Ertl, T.; Hamann, B.; Varshney, A., ed., Proceedings IEEE Visualization 2000, 259–266. IEEE Computer Society Press. (also as SFB 256 report 31, Univ. Bonn, 2000).

Govindaraju, N.K.; Lin, M.C.; Manocha, D. 2005. Quick-cullide: Fast inter- and intra-object collision culling using graphics hardware. vr, 00, 59–66, 319.

Gregory, A.; Mascarenhas, A.; Ehmann, S.; Lin, M.; Manocha, D. 2000. Six degree-of-freedom haptic display of polygonal models. IEEE Visualization 2000, 139–146, New York, NY, USA.

Grimm, C.; Schlüchtermann, G. 2005. Verkehrstheorie in IP-Netzen – Modelle, Berechnungen, statistische Methoden. Hüthig. 525 S.

Guthe, S.; Roettger, S.; Schieber, A.; Strasser, W.; Ertl, T. 2002. High-quality unstructured volume rendering on the pc platform. HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, 119–125, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Haber, R.; McNabb, D. 1990. Visualisation idioms: a conceptual model for scientific visualization systems, 74–93. IEEE Computer Society Press, Los Alamitos, CA, USA.

Hand, D.J.; Smyth, P.; Mannila, H. 2001. Principles of data mining. MIT Press, Cambridge, MA, USA. 546 S.

Hargreaves, B.; Johanson, B.; Nayak, K. Lossless compression of 3D MRI brain images. Online: http://ise.stanford.edu/class/ee392c/demos/hargreaves_johanson_%-nayak/ (Zugriff: 2. Dez. 2004), 1997.

Hayward, V.; Astley, O.R.; Cruz-Hernandez, M.; Grant, D.; Robles-de-la Torre, G. 2004. Haptic interfaces and devices. Sensor Review, 24(1), 16–29.

Heckbert, P.S. 1986. Survey of texture mapping. *In* Green, M., ed., Proceedings of Graphics Interface '86, 207–212.

HLRN. Norddeutscher Verbund für Hoch- und Höchstleistungsrechnen. Online: http://www.hlrn.de/ (Zugriff: 18. Jan. 2007), 2003.

Huang, C.; Qu, H.; Kaufman, A. 1998. Volume rendering with haptic interaction. 3rd Phantom User Group Workshop, 14–17.

Humphreys, G.; Houston, M.; Ng, R.; Frank, R.; Ahern, S.; Kirchner, P.D.; Klosowski, J.T. 2002. Chromium: a stream-processing framework for interactive rendering on clusters. SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, 693–702, New York, NY, USA. ACM Press.

Ibarria, L.; Lindstrom, P.; Rossignac, J.; Szymczak, A. 2003. Out-of-core compression and decompression of large n-dimensional scalar fields. Proceedings of EUROGRAPHICS 2003, 343–348.

Jacobson, I.; Christerson, M.; Jonsson, P.; Övergaard, G. 1993. Object-oriented software engineering – a use case driven approach. Addison-Wesley. 528 S.

Jensen, J.J. Software, ISP IMM DTU, human brain project, Denmark. Online: http://hendrix.ei.dtu.dk/software/polyr/code/polyr.tar.gz (Zugriff: 18. Jan. 2007), 1995.

Jensen, N. 2005. Real-time synchronised 3D-perceptualisation. *In* Alberigo, P.; Erbacci, G.; Garofalo, F., ed., Science and Supercomputing in Europe, 224–230, Casalecchio die Reno (Bologna), Italy.

Jensen, N.; Olbrich, S.; Pralle, H.; Raasch, S. 2002. An efficient system for collaboration in tele-immersive environments. *In* Bartz, D.; Pueyo, X.; Reinhard, E., ed., 4th Eurographics/ACM SIGGRAPH Workshop on Parallel Graphics and Visualization 2002, 123–131.

Jensen, N.; von Voigt, G.; Nejdl, W.; Bernarding, J. 2005. Efficient 1-pass prediction for volume compression. *In* Kalviainen, H.; Parkkinen, J.; Kaarna, A., ed., SCIA 2005, 302–311, Berlin, Germany.

Jensen, N.; Seipel, S.; Nejdl, W.; Olbrich, S. 2003. Covase Â— collaborative visualization for constructivist learning. *In* Wasson, B.; Ludvigsen, S.; Hoppe, U., ed., Proceedings of the Conference on Computer Supported Collaborative Learning 2003, 249–253. Kluwer Academic.

Jensen, N.; Seipel, S.; von Voigt, G.; Raasch, S.; Olbrich, S.; Nejdl, W. 2004a. Development of a virtual laboratory system for science education and the study of collaborative action. *In* Cantini, L.; McLoughlin, C., ed., Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications, 2148–2153. AACE.

Jensen, N.; von Voigt, G.; Nejdl, W.; Olbrich, S. 2004b. Development of a virtual laboratory system for science education. The Interactive Multimedia Electronic Journal of Computer-enhanced Learning, 6(2), http://imej.wfu.edu/articles/2004/2/03/index.asp.

Johnson, D.; Willemsen, P. 2003. Six degree-of-freedom haptic rendering of complex polygonal models. Haptics Symposium 2003, 229–235, Los Alamitos, CA, USA. IEEE Computer Society Press.

Kachina Technologies. Sal: scientific data processing & visualization – software packages. Online: http://sal.kachinatech.com/d/1/index.shtml (Zugriff: 19. Feb. 2002), 2002.

Kähler, R.; Simon, M.; Hege, H.C. 2003. Interactive volume rendering of large sparse data sets using adaptive mesh refinement hierarchies. IEEE Transactions on Visualization and Computer Graphics, 09(3), 341–351.

Kaufman, A. 1991. Volume visualization (tutorial). IEEE Computer Society Press, Los Alamitos, CA, USA. 479 S.

Klappenecker, A.; May, F.; Beth, T. 1998. Lossless compression of 3d MRI and CT data. Proceedings of SPIE on Wavelet Applications in Signal and Image Processing 6, 140–149. SPIE.

Krishnan, K.; Marcellin, M.; Bilgin, A.; Nadar, M. 2004. Compression / decompression strategies for large volume medical imagery. Proceedings of SPIE Medical Imaging 2004: PACS and Imaging Informatics, 152–159. SPIE.

Lakos, J. 1996. Large scale C++ software design. Addison-Wesley Professional Computing Series. Addison-Wesley. 852 S.

Lamphere, P.; Linebarger, J.; Breckenridge, A. 1999. Dynamic isosurface extraction and level-of-detail in voxel space. Sandia Report SAND98-1224.

Lang, U.; Peltier, J.; Christ, P.; Rill, S.; Rantzau, D.; Nebel, H.; Wierse, A.; Lang, R.; Causse, S.; Juaneda, F.; Grave, M.; Haas, P. 1995. COVISE: Perspectives of Collaborative Supercomputing and Networking in European Aerospace Research and Industry, 11. Future Generation Computer Systems (FGCS) Elsevier Science.

Law, C.C.; Schroeder, W.J.; Martin, K.M.; Temkin, J. 1999. A multi-threaded streaming pipeline architecture for large structured data sets. VIS '99: Proceedings of the conference on Visualization '99, 225–232, Los Alamitos, CA, USA. IEEE Computer Society Press.

Lawrence, D.A.; Lee, C.D.; Pao, L.Y.; Novoselov, R.Y. 2000. Shock and vortex visualization using a combined visual/haptic interface. VIS '00: Proceedings of the conference on Visualization '00, 131–137, Los Alamitos, CA, USA. IEEE Computer Society Press.

Leek, M. 2001. Adaptive procedures in psychophysical research. Perception & Psychophysics 63, 1279–1292.

Leffingwell, D.; Widrig, D. 1999. Managing software requirements: A unified approach (the Addison-Wesley object technology series). Addison-Wesley Professional. 528 S.

Levoy, M. 1990. Efficient ray tracing of volume data. ACM Transactions on Graphics, 9(3), 245–261.

Levoy, M. The stanford volume data archive – cthead. Online: http://graphics.stanford.edu/data/voldata/cthead.tar.gz (Zugriff: 28. Jan. 2005), 2004a.

Levoy, M. The stanford volume data archive – mrbrain. Online: http://graphics.stanford.edu/data/voldata/mrbrain.tar.gz (Zugriff: 28. Jan. 2005), 2004b.

Lindstrom, P.; Turk, G. 1999. Evaluation of memoryless simplification. IEEE Transactions on Visualization and Computer Graphics, 5(2), 98–115.

Lindstrom, P. 2000. Out-of-core simplification of large polygonal models. *In* Akeley, K., ed., Siggraph 20000, Computer Graphics Proceedings, 259–262. ACM Press / ACM SIGGRAPH / Addison Wesley Longman.

Lindstrom, P.; Isenburg, M. 2006. Fast and efficient compression of floating-point data. IEEE Transactions on Visualization and Computer Graphics, 12(5), 1245–1250.

Lorensen, W. 1987. Marching cubes: A high resolution 3D surface construction algorithm. Computer Graphics, 21(4), 163–169.

Luebke, D.P. 2001. A developer's survey of polygonal simplification algorithms. IEEE Computer Graphics and Applications, 21(3), 24–35.

Luebke, D.; Erikson, C. 1997. View-dependent simplification of arbitrary polygonal environments. SIGGRAPH 97, 199–208, New York, NY, USA.

Manten, S. 2006. Isosurface-Extraktion mittels parallelisierter, eng gekoppelter Marching-Cube and Vertex-Clustering-Algorithmen. Fakultät für Mathematik und Informatik, Fernuniversität Hagen, Germany. MSc thesis.

Mark, W.; Randolph, S.; Finch, M.; van Verth, J.; Taylor, R. 1996. Adding force feedback to graphics systems: Issues and solutions. SIGGRAPH 96, 447–452, New York, NY, USA.

McNeely, W.; Puterbaugh, K.; Troy, J. 1999. Six degree-of-freedom haptic rendering using voxel sampling. SIGGRAPH 99, 401–408.

Med-OCT-Group. Aufbau der Haut. Online: http://de.wikipedia.org/wiki/bild:haut-aufbau.png (Zugriff: 15. Jan. 2007), 2006.

Minard, C.J. Carte figurative des pertes succesives en hommes de l'Armée Française dans le campagne de Russie 1812–1813. Online: http://upload.wikimedia.org/wikipedia/en/2/29/minard.png (Zugriff: 15. Jan. 2007), 1861.

Nelson, T.H. 1978. The home computer revolution. Distributors.

of Utah, U. Scirun: A scientific computing problem solving environment. Online: http://software.sci.utah.edu/scirun.html, 2002.

Olbrich, S.; Pralle, H. 2001. A tele-immersive, virtual laboratory approach based on real-time streaming of 3D scene sequences. ACM Multimedia 2001, 534–537.

Olbrich, S. 2000. Ein leistungsfähiges System zur Online-Präsentation von Sequenzen komplexer virtueller 3D-Szenen. Fachbereich Elektrotechnik und Informationstechnik, Universität Hannover, Germany. PhD dissertation.

Otaduy, M.A.; Lin, M.C. 2003. Sensation preserving simplification for haptic rendering. ACM Transactions on Graphics, 22(3), 543–553.

Otaduy, M.A. 2004. 6-DoF haptic rendering using contact levels of detail and haptic textures. Department of Computer Science, University of North Carolina at Chapel Hill, USA. PhD dissertation.

Park, K.S.; Cho, Y.J.; Krishnaprasad, N.K.; Scharver, C.; Lewis, M.J.; Leigh, J.; Johnson, A.E. 2000. Cavernsoft g2: a toolkit for high performance tele-immersive collaboration. VRST '00: Proceedings of the ACM symposium on Virtual reality software and technology, 8–15, New York, NY, USA. ACM Press.

Raasch, S.; Schröter, M. PALM – a parallel LES model. Online: http://www.muk.uni-hannover.de/~raasch/palm-1/intro_e.html (Zugriff: 18. Jan. 2007), 2002.

Radon, J. 1917. Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten. Berichte über die Verhandlungen der Sächsischen Akademien der Wissenschaften, Leipzig. Mathematisch-physische Klasse, 69, 262–267.

Ratanaworabhan, P.; Ke, J.; Burtscher, M. 2006. Fast lossless compression of scientific floating-point data. DCC '06: Proceedings of the Data Compression Conference (DCC'06), 133–142, Washington, DC, USA. IEEE Computer Society.

RealVNC. RealVNC. Online: http://www.realvnc.com/ (Zugriff: 18. Jan. 2007), 2002.

Rezk-Salama, C.; Engel, K.; Bauer, M.; Greiner, G.; Ertl, T. 2000. Interactive volume on standard pc graphics hardware using multi-textures and multi-stage rasterization. HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, 109–118, New York, NY, USA. ACM Press.

Rosenblum, L.; Cross, R.A. 1997. Visualization and modeling, chapter The Challenge of Virtual Reality, 324–341. Academic Press.

Rossignac, J. Geometric simplification and compression, course notes #25 SIGGRAPH '97, 1997.

Rossignac, J. 1999. Edgebreaker: Connectivity compression for triangle meshes. IEEE Transactions on Visualization and Computer Graphics, 5(1), 47–61.

Rumbaugh, J.; Jacobson, I.; Booch, G. 2004. Unified modeling language reference manual, the (2nd edition) (addison-wesley object technology series). Addison-Wesley Professional. 550 S.

Ruspini, D.; Kolarov, K.; Khatib, O. 1997. The haptic display of complex graphical environments. Computer Graphics Proceedings 1997, 345–352, New York, NY, USA.

Salomon, D. 2004. Data compression – the complete reference (3rd ed.). Springer. 899 S.

Schirski, M.; Gerndt, A.; van Reimersdahl, T.; Kuhlen, T.; Adomeit, P.; Lang, O.; Pischinger, S.; Bischof, C. 2003. Vista flowlib - framework for interactive visualization and exploration of unsteady flows in virtual environments. EGVE '03: Proceedings of the workshop on Virtual environments 2003, 77–85, New York, NY, USA. ACM Press.

Schreiner, D.; Schreiner, D.E.; Shreiner, D. 1999. OpenGL reference manual: The official reference document to OpenGL, version 1.2. Addison-Wesley, Boston, MA, USA. 761 S.

Schroeder, W.; Lorensen, B. 1996. Visualization toolkit: An object-oriented approach to 3-D graphics. Prentice Hall PTR, Upper Saddle River, NJ, USA. 600 S.

Sensable. Ghost SDK 3.1 manual, 2002.

SGI. SGI professional services: Collaborative visualization solutions. Online: http://www.sgi.com/services/professional/solutions/col_vis/ (Zugriff: 30. Apr. 2002), 2002.

Shimoga, K. 1992. Finger force and touch feedback issues in dextrous telemanipulation. Proceedings of NASA-CIRSSE International Conference on Intelligent Robotic Systems for Space Exploration, Greenbelt, MD, USA. NASA.

Shneiderman, B. 1997. Designing the user interface. Addison Wesley. 639 S.

Snow, J. The clusters of cholera cases in the London epidemic of 1854. Online: http://upload.wikimedia.org/wikipedia/en/2/27/snow-cholera-map-1.jpg (Zugriff: 15. Jan. 2007), 1855.

Steingötter, A.; Werner, C.; Sachse, F.; Dössel, O. 1998. Kompression drei- und vierdimensionaler medizinischer Bilddaten. Biomedizinische Technik, 478–479. Schiele & Schön.

Steinmetz, R.; Wehrle, K., ed. Peer-to-peer systems and applications, 3485 of *Lecture Notes in Computer Science*. Springer, 2005.

Storm, T. 1988. Der Schimmelreiter. Reclam. 41 S.

Strengert, M.; Magallón, M.; Weiskopf, D.; Guthe, S.; Ertl, T. 2005. Large Volume Visualization of Compressed Time-Dependent Datasets on GPU Clusters. Parallel Computing, 31(2), 205–219. doi:10.1016/j.parco.2005.02.006.

Sutherland, I.E.; Sproull, R.F.; Schumacher, R.A. 1974. A characterization of ten hidden-surface algorithms. ACM Computing Surveys, 6(1), 1–55.

Sutter, H. 2002. More exceptional C++: 40 new engineering puzzles, programming problems, and solutions. AW C++ in Depth Series. Addison Wesley. 279 S.

Sutton, P.; Hansen, C.; Shen, H.W.; Schikore, D. 2000. A case study of isosurface extraction algorithm performance. Data Visualization 2000, 259–268.

Szirmay-Kalos, L.; Antal, G.; Sbert, M. 2005. Go with the winners strategy in path tracing. WSCG (Journal Papers), 49–56. UNION Agency Press.

Tan, H.; Srivasan, M.; Eberman, B.; Cheng, B. 1994. Human factors for the design of force-reflecting haptic interfaces. Proceedings of ASME WAM, 353–360, New York, NY, USA. ASME.

Tan, H.Z.; Pizlo, Z. Psychophysics for user interface design and evaluation. the 10th international conference on human-computer interaction (hci international 2003), crete island, greece, 9:00 - 13:00, june 23 2003, 2003.

Taubin, G. 2000. Geometric signal processing on polygonal meshes. Eurographics State of the Art Reports.

Taubin, G. 2002. Blic: bi-level isosurface compression. Proceedings of the conference on Visualization '02, 451–458. IEEE Press.

Taubin, G.; Rossignac, J. 1998. Geometric compression through topological surgery. ACM Transactions on Graphics, 17(2), 84–115.

Treece, G.M.; Prager, R.W.; Gee, A.H. 1999. Regularised marching tetrahedra: improved iso-surface extraction. Computers and Graphics, 23(4), 583–598.

Tufte, E.R. 1983. The visual display of quantitative information. Graphics Press, Cheshire. 197 S.

Vidolm, E.; Nystrom, I. 2005. A haptic interaction technique for volume images based on gradient diffusion. World Haptics Conference 2005, 336–341, Los Alamitos, CA, USA.

von Goethe, J.W. 2001. Die Leiden des jungen Werther. Reclam. 159 S.

Ware, C. 2000. Information visualization – perception for design. Morgan Kaufmann Publishers. 438 S.

Web 3D Consortium. Web3D consortium: specifications. Online: http://www.vrml.org/fs_specifications.htm (Zugriff: 19. Feb. 2002), 2004.

Weber, E. 1834. De Pulsu, Resorptione, Auditu et Tactu. Annotationes Anatomicae Et Physiologicae.

Welch, T.A. 1984. A technique for high-performance data compression. IEEE Computer, 17(6), 8–19.

Wikipedia. Quantisierung. Online: http://de.wikipedia.org/wiki/quantisierung (Zugriff: 15. Jan. 2007), 2006.

Wood, J.; Wright, H.; Brodlie, K. 1997. Collaborative visualization. IEEE Proceedings – Visualization, 253–259.

Woop, S.; Schmittler, J.; Slusallek, P. 2005. RPU: a programmable ray processing unit for realtime ray tracing. ACM Trans. Graph., 24(3), 434–444.

Wössner, U.; Schulze, J.P.; Walz, S.P.; Lang, U. 2002. Evaluation of a collaborative volume rendering application in a distributed virtual environment. EGVE '02: Proceedings of the workshop on Virtual environments 2002, 113–ff, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Yang, C. 2000. Integration of volume visualization and compression: a survey. Computer Science Department, State University of New York at Stony Brook, New York, NY, USA. Technical Report Research Proficiency Report.

# Index

# Lebenslauf

| | |
|---|---|
| Name: | Nils Jensen |
| Geburtsdatum, -ort: | 14. Januar 1975, Hamburg |
| Eltern: | Brigitte Jensen (geb. Erdmann), Thomas Jensen |
| Familienstand: | Keine Geschwister, ledig |
| Epost: | nils_jensen@acm.org |
| | |
| Studienabschlüsse: | Doktor-Ingenieur („Magna cum laude”)<br>Master of Science („Sehr gut mit Auszeichnung”)<br>Diplom-Informatiker (FH) („Sehr gut”) |
| | |
| Studien: | Aufbaustudium (Master) in Computergrafik und parallelem Rechnen an der Universität Hull, Kingston upon Hull, Großbritannien (2000 – September 2001) (Regelstudienzeit) |
| | |
| | Fachhochschulstudium Softwaretechnik an der HAW Hamburg (März 1996 – April 2000) (Regelstudienzeit) |
| | |
| Wehrdienst: | Zivildienst (Krankenpflege) im Krankenhaus „Alten Eichen”, Hamburg<br>(Juli 1994 – September 1995) |
| | |
| Schulbildung: | Emilie-Wüstenfeld-Gymnasium, Hamburg<br>(1991 – Juni 1994) (Abitur mit „Gut” bestanden) |
| | |
| | Realschule Halstenbek, Halstenbek<br>(1985 – Juni 1991) (Realschulabschluß mit „Gut” bestanden) |
| | |
| | Grundschule Nord, Halstenbek<br>(1983 – Juni 1985) |
| | |
| | Grundschule Hinschenfelde, Hamburg<br>(1981 – Juni 1983) |

| | |
|---|---|
| Berufstätigkeiten: | sd&m AG, Hamburg<br>(seit Februar 2007)<br>Software-Ingenieur |
| | Forschungszentrum L3S / Regionales Rechenzentrum für Niedersachsen,<br>Leibniz-Universität Hannover<br>(Dezember 2001 - Januar 2007)<br>Wissenschaftlicher Mitarbeiter für wissenschaftliche Visualisierung |
| | NxN Software AG, München<br>(November 1999 - August 2000)<br>Diplomand und Mitarbeiter für Content-Management-Server-Entwicklung |
| | IBM Informationssysteme Deutschland GmbH, Hamburg<br>(Februar - September 1998)<br>Praktikant und freier Mitarbeiter für Qualitätssicherung |
| | IAP GmbH, Hamburg<br>(Oktober 1995 – Februar 1996, August – September 1996, März 1997)<br>Praktikant und studentischer Mitarbeiter für Datenbankentwicklung |
| Auszeichnungen: | „HPC-Europa Fellowship Research Grant for Computational Science"<br>(Oktober 2005) |
| | „ED-Media 2004 best works"-Award für das Paper<br>„Development of a Virtual Laboratory System for Science Education<br>and the Study of Collaborative Action" (September 2004) |
| | „Best student"-Award der British Computer Society für den<br>„M. Sc. in Computer Graphics and Virtual Environments"<br>(Februar 2002) |
| Sonstiges: | Zertifizierter Massage-Praktiker (seit Oktober 2006) |