TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR SOFTWARE- UND MULTIMEDIATECHNIK
PROFESSUR FÜR COMPUTERGRAPHIK UND VISUALISIERUNG
PROF. DR. STEFAN GUMHOLD

Großer Beleg

Gestengesteuerte visuelle Datenanalyse einer Laser-Plasma-Simulation

Benjamin Schneider
(Mat.-Nr.: 3425954)

Betreuer:

Prof. Dr. rer. nat. Stefan Gumhold
Dr. Michael Bussmann
Dipl.-Inf. Marcel Spehr

Dresden, 30. Juni 2012

Aufgabenstellung

Zur Untersuchung der Wechselwirkungen von Laserpulsen mit Materie werden am HZDR Laser-Plasma-Simulationen durchgeführt und ausgewertet. Um die große Menge an Daten, die bei solchen Simulationen erzeugt wird, effektiv zu analysieren soll in dieser Arbeit, aufbauend auf der Diplomarbeit von Lukas Zühl, zusätzlich zur Visualisierung eine Interaktion mit den gezeigten Daten ermöglicht werden.

Im ersten Schritt werden die zu visualisierenden Daten durch einen Konfigurationsdialog ausgewählt. Hierbei kommt die deklarative Visual Query Language (VQL) zum Einsatz. Der ausgewählte Datensatz soll nun mit Hilfe von VTK in einem separaten Fenster stereoskopisch auf eine Rückprojektionsleinwand gezeichnet werden. Hierbei gibt es für die verschiedenen Typen von Datensätzen (Partikel-, Felddaten) unterschiedliche Arten der Visualisierung (Partikelplot, Isoflächen etc.). Die Interaktion mit den gezeigten Daten soll über Gesten erfolgen. Zur Erkennung der Gesten wird eine Microsoft Kinect verwendet. Damit soll das Navigieren in einer Visualisierung sowie das Selektieren von interessanten Teilmengen der abgebildeten Daten ermöglicht werden. Beispiele für Eingabegesten und deren Effekt sind das ”Greifen und Auseinanderziehen” als Zoomgeste oder das Zeichnen einer virtuellen Ebene um eine Teilmenge der gezeigten Daten auszuwählen. So könnten beispielsweise einzelne Partikel selektiert werden um sie über einen bestimmten Zeitabschnitt zu verfolgen und eine Visualisierung ihrer Bahnkurve(n) entstehen.

Um geeignete Gesten zur Steuerung des Programms zu finden, können verschiedene Eingabegesten in Hinblick auf einige Gesichtspunkte wie Ergonomie und Konformität zur Nutzerintuition evaluiert werden.

Zentraler Bestandteil dieser Arbeit ist die Implementierung eines Backends zur Visualisierung mit VTK und Gestenverarbeitung sowie dessen Verbindung zum Frontend, welches die Gestenerkennung verwirklicht.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Arbeit zum Thema:

Gestengesteuerte visuelle Datenanalyse einer Laser-Plasma-Simulation

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 30. Juni 2012

Benjamin Schneider

Inhaltsverzeichnis

Imagination or visualization, and in particular the use of diagrams, has a crucial part to play in scientific investigation.

René Descartes, 1637

1 Einleitung

1.1 Motivation

Am Helmholtz Zentrum Dresden-Rossendorf wird die Wechselwirkung zwischen hochenergetischen Laserpulsen und Materie theoretisch und experimentell untersucht. Um schneller neue Erkenntnisse zu gewinnen und geeignete Parameter für reale Experimente zu bestimmen, wird die Laser-Plasma-Kollision mit GPU-Codes wie PIConGPU [BWH+10] simuliert. Diese Simulationen laufen auf Hochleistungsrechnern und erzeugen aufgrund ihrer hohen räumlichen und zeitlichen Auflösung sehr große Mengen an Daten. Das Ausmaß dieser Datenmengen bewegt sich im Bereich von vielen Terabyte bis zu einigen Petabyte. Auf Papier ausgedruckt könnte man damit wohl einen Turm bis zum Mond aufschlichten. Nur das Lesen aller gedruckten Blätter einer einzelnen Simulation würde Monate dauern - keine effiziente Methode der Datenauswertung. Abgesehen davon ist Speicherplatz und -zeit auf Großrechenanlagen sehr begrenzt und teuer[1].

Um also die entstandenen Daten effizient auszuwerten, bedarf es Werkzeugen und Methoden, welche es dem Wissenschaftler erlauben seine kognitiven Fähigkeiten und sein Expertenwissen optimal zu nutzen. Ein solches Werkzeug sollte die Daten nicht nur in einer ansprechenden Form darstellen, also visualisieren, sondern auch intuitiv bedienbar sein. Es existiert bereits eine Vielzahl freier sowie kommerzieller Programme zur wissenschaftlichen Visualisierung [Kit12] [SWH05]. Die meisten dieser Tools setzen noch auf ein klassisches WIMP-Interface[2] und sind in ihrer Funktionalität sehr allgemein gehalten, was in einem hohen Einarbeitungsaufwand resultiert. Selbst um einfachste Aufgaben, wie das Laden und Darstellen der Daten sowie das Selektieren von Teilmengen dieser Daten, zu bewältigen bedarf es einiger Zeit die Tools kennenzulernen. Zeit die von Wissenschaftlern sinnvoller verwendet werden könnte und die Studenten während einer Studienarbeit nicht haben.

Zunächst stellen sich zwei grundlegende Fragen: Warum ist eine mathematisch unpräzise Visualisierung zum Verständnis großer Datenansammlungen besser geeignet als eine Tabelle oder andere numerische Darstellungen? Und: Welche Vorteile bringt eine Steuerung durch Gesten an Stelle von Tastatur und Maus?

1.2 Warum Visualisierung ?

Es gibt zahlreiche Beispiele für Quellen großer Datenmengen. Neben den bereits genannten Simulationscodes wie *PIConGPU* [BWH+10] sind dies u.a. Laser-Scanning-Systeme, die in weniger als 15 Sekunden über 500.000 Punkte und mehr erkennen [WT91], Wettersimulationen die globale Wettermuster

[1] z.B. http://www.rrze.uni-erlangen.de/dienste/konditionen/preise/ Abruf: 05.05.2012
[2] http://de.wikipedia.org/wiki/WIMP_(Benutzerschnittstelle)

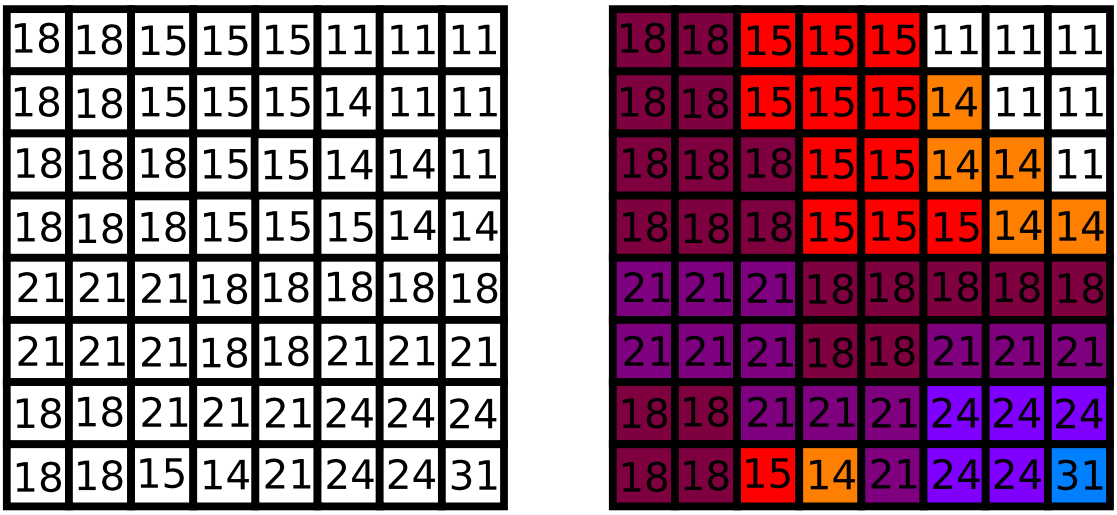


Abbildung 1.1: Leichteres Erfassen und Verstehen von Informationen durch farbliches Hervorheben

berechnen [Che93] und viele weitere Datenquellen aus Physik, Biologie, Medizin, Astronomie und Wirtschaft. Zur Speicherung dieser Datenmengen wurden immer größere Speichermedien entwickelt. Meist werden diese Daten ohne Vorverarbeitung und Filterung als Rohdaten abgespeichert. Diese Rohdaten selbst haben keine besondere Aussagekraft. Lediglich die aus den Daten extrahierten Informationen sind von Wert [KKEM10].

Visualisierung gibt uns die Möglichkeit aus dieser Flut an Daten die interessanten und wichtigen Informationen zu extrahieren [SML98]. Sie gibt uns einen komprimierten Überblick über die Daten, die verschiedenen Parameter und ihre Wertebereiche. Dadurch entsteht ein Gesamteindruck, den man aus numerischen Listen und Tabellen nur schwer gewinnen kann. So lassen sich bspw. Trends erkennen oder verschiedene Darstellungen nebeneinander oder übereinander legen und vergleichen. Beispiele hierfür sind die Visualisierungstechniken *Linking*[3] und *Brushing*[4].

Neben dieser makroskopischen Darstellung, die Effekte im Großen zeigt, können durch genauere Betrachtung auch Phänomene im Kleinen ausgemacht werden. Dazu werden Visualisierungstechniken eingesetzt, die eine Fokussierung auf Teilbereiche der Daten erlauben ohne sie aus dem Kontext zu reißen.

Dabei unterstützt die Visualisierung den leistungsfähigsten Bildverarbeitungsapparat den wir kennen: das menschliche (visuelle) Wahrnehmungssystem. Es ist hervorragend darauf trainiert Muster zu erkennen und Regionen schneller Veränderung auszumachen. Zusammen mit dem Expertenwissen des Anwenders können so sehr schnell Ergebnisse bewertet, neue Hypothesen aufgestellt und vorhandene Annahmen überprüft werden.

Nichtzuletzt dient die Visualisierung auch der Präsentation von Ergebnissen. Aussagekräftige Abbildungen werden häufig schneller und besser verstanden als ausschließlich Fließtext und Zahlenkolonnen (siehe Abb. 1.1).

[3] die gleichzeitige Darstellung mehrerer Ansichten
[4] die Selektion in einer Ansicht mit Markierung in allen Ansichten

1.3 Warum Gestensteuerung ?

Um den Nutzer des Visualisierungssystems nicht zum passiven Beobachter zu machen und nur mit statischen Bildern zu konfrontieren, bedarf es Möglichkeiten zur Interaktion mit der virtuellen Welt, die vor ihm liegt. Schon die freie Wahl des Blickpunktes kann einen enormen Verständniszuwachs für die räumliche Struktur der visualisierten Daten bedeuten und den Nutzer zum Experimentieren animieren. In diesem Zusammenhang bietet die Steuerung durch Gesten in Verbindung mit Echtzeit-Rendering die Möglichkeit zur direkten Manipulation und unmittelbares Feedback über die vom Nutzer ausgeführten Aktionen. Dies führt zu einer Verkürzung der kognitiven Distanz zwischen Aktion des Nutzers und Reaktion des Systems, was der intuitiven Bedienung und dem Erlernen des zugrunde liegenden Interaktionskonzeptes zu Gute kommt. Somit bieten gut gestaltete, auf die ergonomischen Voraussetzungen und intuitiven Erwartungen des Anwenders abgestimmte, natürliche Benutzerschnittstellen eine sehr flache Lernkurve. Aufwändige Schulungen oder Trainings sind nicht mehr erforderlich. Man kann sofort anfangen mit dem Visualisierungssystem zu arbeiten. Und das ohne sich komplizierte Maus- und Tastatureingabefolgen einprägen oder durch zahlreiche Untermenüs klicken zu müssen, nur um eine einfache Betrachtung der Visualisierung durchzuführen.

Der gewählte Interaktionsmodus bestimmt nicht nur das ”Vokabular” der möglichen Eingaben durch das Eingabegerät, sondern beeinflusst auch sehr stark die Gestaltung der grafischen Benutzeroberfläche (engl. *Graphical User Interface*, kurz: GUI). Während Maus und Tastatur häufig in Verbindung mit WIMP-Interfaces - bestehend aus Buttons und Boxes - genutzt werden, erlaubt ein natürliches, intuitives Interaktionskonzept den Einsatz einer minimaleren GUI. Sie dient vorrangig der Unterstützung des Nutzers, indem sie Aufschluss über mögliche Aktionen gibt oder Feedback über den Zustand des Systems, wie z.B. den Tracking-Status (Wie nimmt mich das System wahr? Welche Eingabe erkennt es gerade?), vermittelt.

Nichtzuletzt ist die Umsetzung einer Gestensteuerung auf Grund der jetzt verfügbaren Technologie erst möglich geworden [BKJJLP04]. Moderne Hardware erlaubt das Erkennen von Personen und Gesten auf Basis ausgefeilter Bildverarbeitungsalgorithmen. Schon zu einem Preis von etwa € 100 kann man eine Microsoft Kinect sein Eigen nennen[5]. Die benötigten Treiber[6] und Bibliotheken[7] sind als Open-Source Software im Internet frei verfügbar.

1.4 Gestengesteuerte visuelle Datenanalyse

> *Exploratory data analysis (EDA) means open-minded looking at data with the aim to detect and describe patterns, trends, and relationships and to generate hypotheses, which can then be tested using mathematical methods.* [AA07]

Ziel dieser Arbeit ist es, die Möglichkeiten einer Visualisierung zur Datenanalyse mit der intuitiven Bedienbarkeit eines Gesteninterfaces zu verbinden. Dazu ist eine Kombination aus immersiver - meist stereoskopischer - 3D-Grafikausgabe, einem Eingabegerät zur direkten dreidimensionalen Interaktion und

[5] http://amazon.de Abruf: 14.05.2012
[6] Kinect Treiber: https://github.com/avin2/SensorKinect
[7] OpenNI & NITE: http://openni.org oder http://www.microsoft.com/en-us/kinectforwindows/

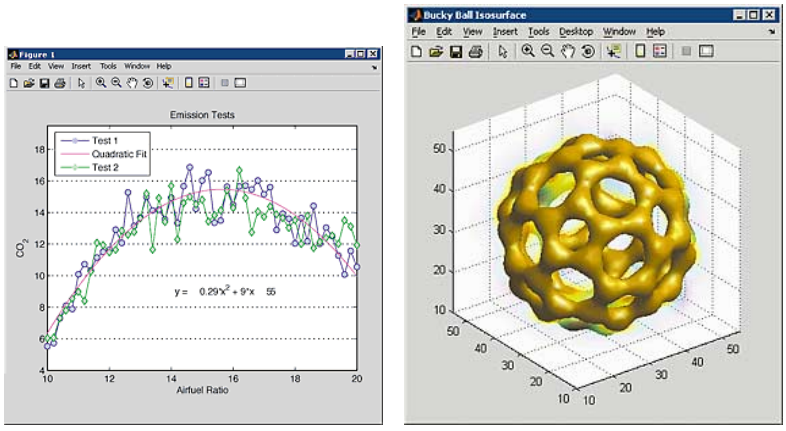
der entsprechenden Rechnerhardware zur Auswertung der Nutzereingaben und Darstellung der virtuellen Welt (engl. *VR - Virtual Reality* oder auch *VE - Virtual Environment*) nötig [Bry96].

Es wird ein auf die Bedürfnisse des HZDR zugeschnittenes System entwickelt, welches dem Nutzer erlaubt aus vorhandenen Simulationsdaten die für ihn interessanten Datensätze auszuwählen und diese durch Zuordnung der Beobachtungs- und Merkmalsdaten (z.B. Zeitschritt, Partikel-ID, Position, Feldvektor, Energie) zu den Visualisierungsattributen (z.B. Raumachsen, Vektorwerte, Skalare) darzustellen. Um die Immersion zu steigern wird die 3D Szene stereoskopisch auf eine Rückprojektionsleinwand (sog. Powerwall) gerendert. Außerdem soll der Nutzer in der Lage sein sich frei in der Visualisierung zu bewegen und für ihn wichtige Datenbereiche zu selektieren und auszuwerten. Ergebnis soll eine intuitiv benutzbare Datenanalyse- und Präsentationssoftware sein, die es erlaubt große Datenmengen auf einfachem Wege darzustellen und zu bearbeiten. Ein einfaches Werkzeug für einfache - aber wichtige - Aufgaben. Nach dieser initialen Betrachtung können auf Basis eines stark reduzierten Datensatzes genauere Untersuchungen mit Werkzeugen wie MatLab o.ä. auf evtl. gefundene Hypothesen vorgenommen werden.

1.5 Verwandte Arbeiten

Um die interaktive Analyse großer Datenmengen effektiver zu gestalten gibt es eine Fülle an Ansätzen. Die Wichtigsten sollen hier kurz vorgestellt werden.

Als Beispiel für die ”klassische” Vorgehensweise der mathematisch-statistischen Datenauswertung soll MatLab herangezogen werden. MatLab bietet eine Hochsprache für das wissenschaftliche und technische Rechnen, mit der man Aufgaben aus diesen Bereichen schneller und einfacher bewältigen kann als mit herkömmlichen Programmiersprachen wie C, C++ oder Fortran. Darüber hinaus bietet es auch die Möglichkeit 2D und 3D Diagramme zu erstellen. [Mat12]



(a) 2D-Liniendiagramm (b) 3D-Isoflächendiagramm

Abbildung 1.2: Visualisierung mit MatLab

Ein moderner Ansatz besteht in der Verbindung von Echtzeit-Interaktion und High-Performance-Visualisierung. Oft ist es möglich, große Datensätze von mehreren Prozessoren gleichzeitig verarbeiten zu lassen. Als Beispiel einer visuell programmierbaren, hochparallelen Datenverarbeitungsanwendung soll

ParaView vorgestellt werden. ParaView ist eine freie auf VTK basierende Open-Source-Anwendung zur (dreidimensionalen) Visualisierung. Sie bietet eine grafische Benutzeroberfläche, um die Visualisierungspipeline zu konfigurieren und das Resultat interaktiv zu betrachten und auszuwerten. Durch die eingesetzten Mechanismen zur Parallelverarbeitung und die Client-Server-Architektur lassen sich große Datensätze problemlos darstellen. [Kit12]

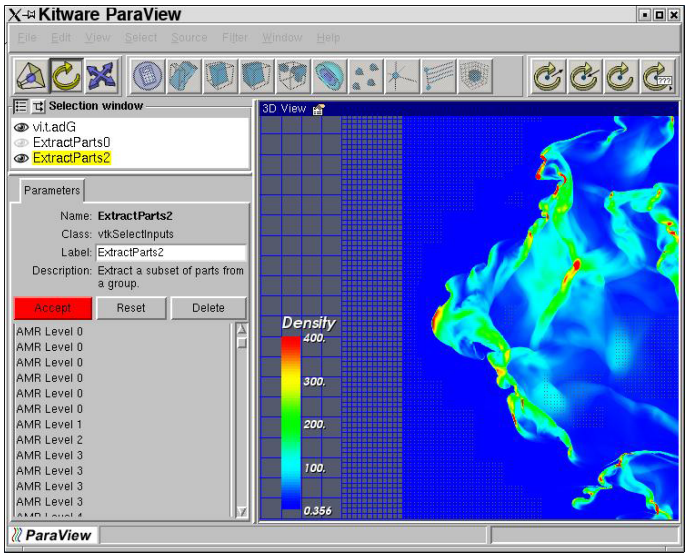


Abbildung 1.3: ParaView

Auch die virtuelle Visualisierungsumgebung (engl. *virtual visualization environment*) ist als Kombintion aus Mensch-Computer-Interaktion und immersiver Darstellungstechnik zu einem wichtigen Mittel der interaktiven Visualisierung geworden [DLS02, s. 535-555]. Der Grad der Immersion wird dabei stark durch die verwendete Displaytechnik und -größe bestimmt. Die Bandbreite reicht von nichtimmersiven Desktopsystemen bis hin zu halb- oder voll-immersiven stereoskopischen Powerwalls und CAVEs [CNSD93].

Kommen mehrere Interaktionstechniken, wie z.B. direkte Manipulation, Spracherkennung und Haptik, gleichzeitig zum Einsatz, so spricht man von multimodalen Systemen. Diese bieten eine natürlichere Interaktion und erlauben dem Nutzer mehr seiner Artikulations- und Kommunikationsfähigkeiten einzusetzen [OFD04]. Das *Multimodal Scientific Visualization Tool* bietet eine virtuelle Arbeitsumgebung zur Visualisierung von Flüssigkeitsströmungen. Die Interaktion erfolgt dabei im Gegensatz zum eingangs erwähnten ParaView nicht durch Maus und Tastatur, sondern multimodal mit Sprach- und Gesteneingabe. Dies soll ein Erfahren der Daten auf natürliche Art und Weise ermöglichen. Die dreidimensionale Darstellung wird durch das eingesetzte Stereodisplay noch unterstützt. Zur Erkennung der Handgesten nutzt MSVT den Fakespace Pinch Glove[8]. [LJ99]

Vermischt man virtuelle und reale Welt bzw. überlagert man die reale Welt mit digitalen Inhalten, so spricht man von *Augmented Reality*. Diese findet vorallem in der Medizin und dem Ingenieurwesen Anwendung [ZSAL09, s. 8].

Zuletzt soll noch kurz der Aspekt der gemeinsamen Arbeit an einer Visualisierung angesprochen werden, die sog. *Collaborative Visualization*. Sie kann zusammen in einem Raum (VR-Labor) oder an verschie-

[8] http://www.fakespacelabs.com/tools.html

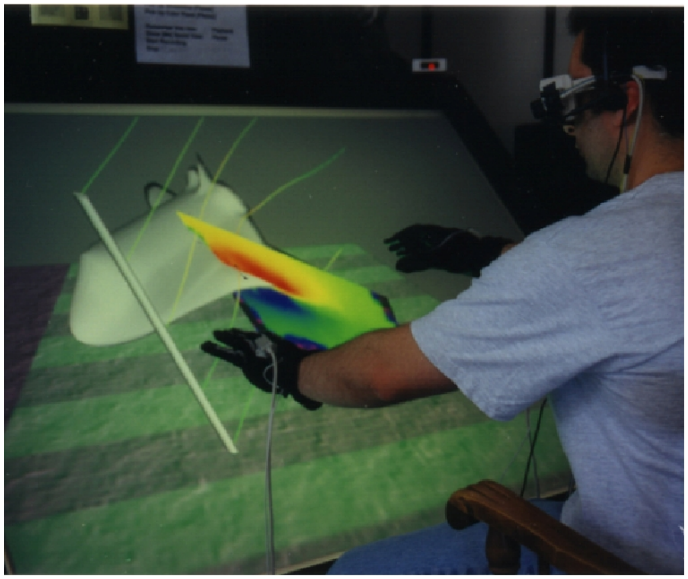


Abbildung 1.4: Multimodal Scientific Visualization Tool im Einsatz (`http://cs.brown.edu/people/jjl/msvt.html`)

denen Orten verteilt über ein Netzwerk an Einzelplatzrechnern erfolgen und dient dem Austausch von Ideen sowie der Diskussion und Kommunikation von Ergebnissen [BDG+04].

Es existiert also eine Vielzahl an Anwendungen und Systemen zur wissenschaftlichen Visualisierung und Datenauswertung. Deren Spektrum an Funktionalität und Bedienkonzepten reicht dabei von immersiven Betrachtungsprogrammen, über mathematisch-statistische Analysewerkzeuge, bis hin zu (visuell) programmierbaren, hochparallelen Datenverarbeitungsanwendungen und Toolkits mit klassischen WIMP-Interfaces oder multimodalen Nutzerschnittstellen, mit Darstellung auf mobilen Endgeräten oder mehrseitigen Projektionsleinwänden (CAVE).

1.6 Abgrenzung

Die entstandene Software soll kein Werkzeug zur exakten Datenanalyse bieten. Für genaue mathematisch-physikalische Berechnungen und statistische Auswertungen existiert bereits eine Vielzahl an Programmen[9]. Ebenso wenig kann eine vollständige Palette an Tools (siehe Abschnitt 4.4.4) implementiert oder der Funktionsumfang etablierter Anwendungen erreicht werden (vgl. ParaView).

1.7 Aufbau dieser Arbeit

Nachdem diese Arbeit nun motiviert und gegenüber vorhandenen Projekten abgegrenzt wurde, folgt in Kapitel 2 eine Einführung in die grundlegenden Begriffe der Visualisierung und der Gesteninteraktion.

In Kapitel 3 werden die Anforderungen an eine intuitive und immersive Arbeitsumgebung zur wissenschaftlichen Datenanalyse geschildert. Zuvor wird das Projekt *VIPER*, hervorgegangen aus der Diplom-

[9] z.B. MatLab http://www.mathworks.de/products/matlab/

arbeit von Lukas Zühl [Zü11], als Ausgangspunkt dieser Arbeit beschrieben.

Wie das Projekt konkret umgesetzt wurde und mit Hilfe welcher Bibliotheken wird in Kapitel 4 dargelegt. Hier werden die einzelnen Systemkomponenten im Detail beschrieben. Ebenso wurden einige Erweiterungen am vorhandenen VIPER vorgenommen, die hier erläutert werden.

Anschließend werden in Kapitel 5 die Erfahrungen einiger Testpersonen bei ihrem ersten Kontakt mit dem System ausgewertet. Das Publikum reichte dabei von unerfahrenen Nutzern, die das System auf öffentlichen Präsentationen vorgeführt bekamen und anschließend selbst ausprobierten bis hin zu technisch versierten Testpersonen (Mitarbeiter der TU Dresden und des HZDR).

In den Kapiteln 6 und 7 wird noch einmal kurz zusammengefasst, was das System leistet, wo sich mögliche Einsatzgebiete finden und wo Ansatzpunkte zur Weiterarbeit liegen.

2 Grundlagen und Definitionen

In diesem Kapitel sollen die wichtigsten Begriffe der Visualisierung und der Gesteninteraktion geklärt werden. Allen voran der Begriff *Visualisierung* selbst.

2.1 Visualisierung

> *Both scientific visualization and information visualization create graphical models and visual representations from data that support direct user interaction for exploring and acquiring insight into useful information embedded in the underlying data.* [OL03]

Visualisierung ist also die Transformation von Daten oder Informationen aus einem *Beobachtungsraum* in eine bildliche Darstellung, die möglichst *expressiv* und *effektiv* ist und dabei mit *angemessenem* Aufwand durchgeführt werden kann. Eine direkte Interaktion mit der Visualisierung gibt dem Nutzer eines Visualisierungssystems die Möglichkeit, seine Daten interaktiv zu erforschen und die Parameter der *Visualisierungspipeline* seinen Vorstellungen und Untersuchungszielen entsprechend anzupassen.

Effektivität Effektiv ist eine Visualisierung, wenn sie die Gegebenheiten und charakteristischen Eigenschaften des Betrachters und des Ausgabegerätes optimal ausnutzt. Mit anderen Worten ist eine Darstellung dann effektiv, wenn sie intuitiv wahrgenommen werden kann.

Expressivität Expressivität oder *Ausdrucksfähigkeit* bedeutet, dass die dargestellte Datenmenge unverfälscht wiedergegeben wird. Es sollen nur die Informationen visualisiert werden, die auch in den Daten enthalten sind. Ein Beispiel zur Verletzung des Expressiviätskriteriums findet sich in [Mac86]. Hier wird gezeigt, wie die Verwendung eines Balkendiagramms an Stelle eines Scatterplots eine Ordnung zwischen Merkmalen der Daten suggeriert, die in Wirklichkeit nicht vorhanden oder nicht Bestandteil der Datenmenge ist.

Angemessenheit Da die Transformation großer Datenmengen in hochauflösende Bilder ein u.U. sehr ressourcenaufwändiger Prozess sein kann, muss bei der Suche nach der optimalen Visualisierung auch Aufwand und Nutzen abgewägt werden. Angemessenheit beschreibt also die Kosten eines Visualisierungsprozesses in Hinblick auf das zu erwartende Ergebnis. In der Praxis sind Effektivität und Angemessenheit häufig voneinander abhängig. So müssen für eine effektive, interaktive Datenanalyse ausreichend Ressourcen verfügbar sein, um eine (nahezu) flüssige Bildfolge zu generieren bzw. muss die Visualisierung so gestaltet werden, dass dies mit den vorhandenen Ressourcen möglich ist.

Visualisierungspipeline Die Visualisierungspipeline stellt den schrittweisen Prozess der Transformation von abstrakten Daten in Bilder dar. Die drei wesentlichen Schritte dabei sind:

Das *Filtering* oder auch Datenaufbereitung. Eine Daten-zu-Daten-Abbildung deren Ausgangspunkt die (gemessenen oder berechneten) Rohdaten sind. Häufig werden hier Operationen zur Vervollständigung oder Reduktion der Daten angewandt. Fehlende Datenwerte können interpoliert oder für eine Untersuchung irrelevante Variablen von der Visualisierung ausgeschlossen werden.

Das *Mapping*. Es dient dazu, den aufbereiteten Daten eine geometrische Repräsentation zu zuweisen, indem sie auf Primitive wie Punkte, Linien, Polygone oder geometrische Formen einschließlich der zugehörigen Attribute (wie z.B. Farbe) abgebildet werden.

Das *Rendering*. Im letzten Schritt wird aus den geometrischen Primitiven durch Methoden der Computergraphik ein Bild generiert, das durch gängige Anzeigegeräte ausgegeben und vom Nutzer betrachtet werden kann. [SM00, s. 15 ff]

In Abbildung 2.1 ist die Visualisierungspipeline nach Haber und McNabb (1990) noch einmal schematisch dargestellt.

Abbildung 2.1: Visualisierungspipeline nach [HM90]

Integriertes Referenzmodell Das integrierte Referenzmodell zeigt, welchen Einfluss der Nutzer auf die Visualisierung hat. Es verdeutlicht, an welchen Stellen eine interaktive Anpassung der Visualisierungspipeline durch den Nutzer möglich ist, sofern er mit dem System vertraut ist oder vom System beispielsweise durch eine Benutzeroberfläche unterstützt wird. Die Zyklen innerhalb der Abbildung 2.2 zeigen, dass es sich hierbei um einen iterativen Prozess handelt. Der Nutzer interagiert mit dem Visualisierungssystem, um eine für ihn interessante Darstellung der gegebenen Datenmenge zu finden. Dazu kann er verschiedene Filter anwenden und die Attribute der Visualisierung beeinflussen (z.B. verschiedene Farbskalen für das Mapping von Skalarwerten ausprobieren).

Visualisierungsszenario Es gibt mehrere Möglichkeiten, wie der Prozess der interaktiven Datenanalyse und Visualisierung gestaltet werden kann, sog. Visualisierungsszenarien. Diese sind der *Bewegungsmodus*, das *Tracking*, das *Interaktive Postprocessing* und die *Interaktive Steuerung*. Für diese Arbeit interessant ist das Szenario des *Interaktiven Postprocessing* (siehe Abb. 2.3). Dabei werden die Daten im ersten Schritt aus Messungen und Beobachtungen sowie aus mathematischen Modellen und Simulationen gewonnen und dann im zweiten Schritt visualisiert und interaktiv bearbeitet. Die im ersten Schritt bereitgestellten Daten können also nicht mehr verändert sondern lediglich anders dargestellt werden.

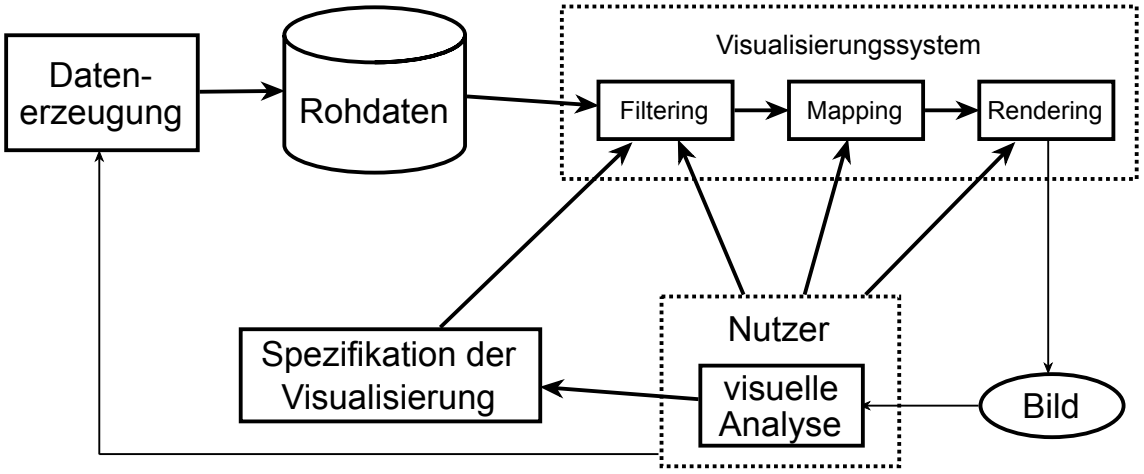


Abbildung 2.2: Integriertes Referenzmodell angelehnt an [RF94]

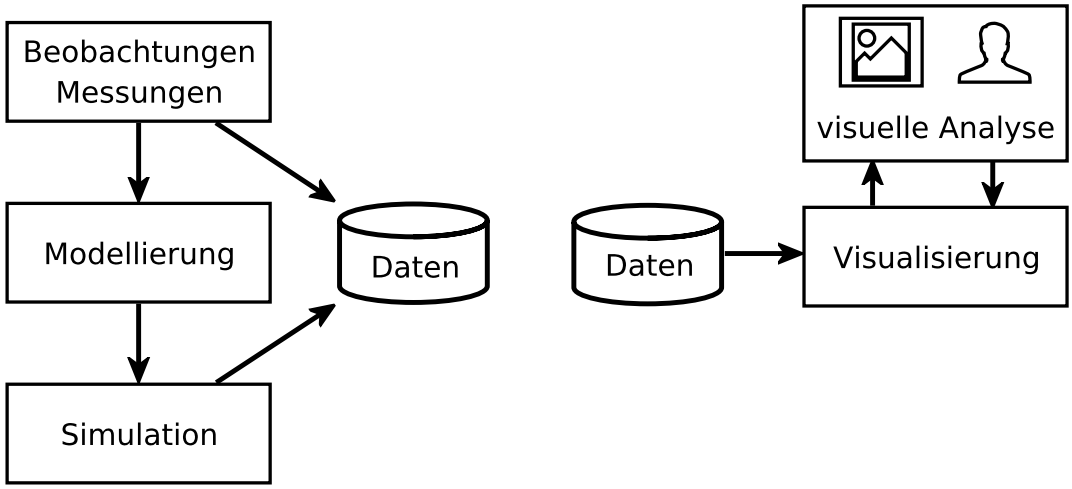


Abbildung 2.3: Schritt 1: Datenerzeugung (links), Schritt 2: interaktive Bearbeitung (rechts)

Referenzarchitektur Zur softwaretechnischen Umsetzung eines Visualisierungssystems kann auf die in Abbildung 2.4 gezeigte Referenzarchitektur zurückgegriffen werden. Sie zeigt die Kernkomponenten und ihre (möglichen) Beziehungen untereinander. Einer der wesentlichen Vorteile dieser Architektur ist die Trennung von Datenmodell und visuellem Modell, was die Visualisierung ein und derselben Datenquelle auf verschiedene Weisen ermöglicht. Ebenso werden Visualisierung und Sicht (engl. *View*) auf eben diese voneinander getrennt. Dies ermöglicht mehrere Sichten mit einer Visualisierung zu verknüpfen. Durch den Einsatz modularer Controller können Nutzereingaben von einer Vielzahl von Geräten verarbeitet und flexibel eingesetzt werden. [HA06]

Beobachtungsraum Der Beobachtungsraum bezeichnet den Raum, in dem die Daten erhoben werden. Dabei wird bewusst von der Art der Erhebung - ob gemessen, beobachtet oder berechnet - abstrahiert. Ebenso wird von der Beschaffenheit des Raumes abstrahiert, also ob es sich um einen physikalischen Raum mit drei Orts- und evtl. einer Zeitachse handelt oder einen abstrakten Raum mit beliebig vielen Dimensionen. Die Punkte des Beobachtungsraumes, für die Daten erhoben wurden, werden als

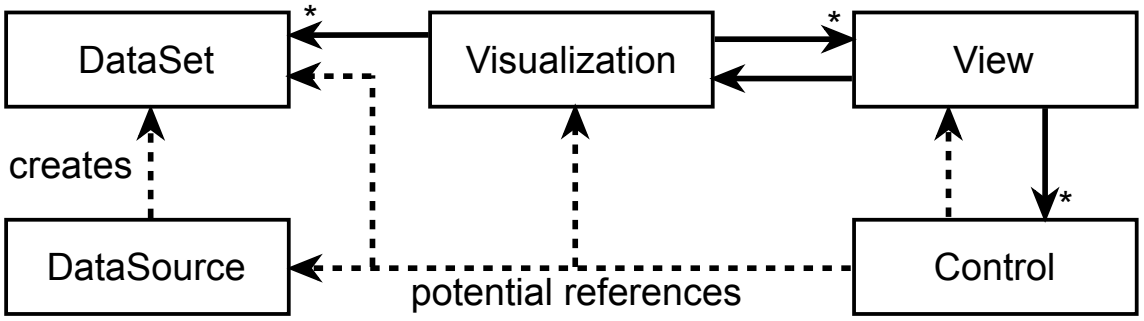


Abbildung 2.4: Referenzarchitektur der Visualisierung

Beobachtungspunkte bezeichnet. Interessant ist dabei auch der Verbund der Beobachtungspunkte. Einerseits kann es sich dabei um beliebig verteilte Beobachtungspunkte handeln. Dann spricht man von *gitterfreien Daten*. Andererseits kommt es häufig vor, dass zwischen den Beobachtungspunkten ein gewisser Zusammenhang besteht. Dann kommen Gitter zum Einsatz um die Daten zu strukturieren. Dabei wird zwischen verschiedenen Gittertypen unterschieden, wie z.B. regelmäßige, unregelmäßige, blockstrukturierte, strukturierte oder hybride Gitter. [SM00, s. 29 ff]

Merkmal Die verschiedenen Größen, die an den Beobachtungspunkten erhoben werden, werden als Merkmale bezeichnet. Da ihre Werteverteilung von ihrer Position im Beobachtungsraum abhängt, werden Merkmale auch abhängige Variablen genannt. Für die Visualisierung interessant sind vor allem Datentyp, Dimensionalität, Wertebereich und Strukturierung der Merkmale. [SM00, s. 35]

Visualisierungsattribut Die Merkmale eines Datensatzes müssen zum Zweck der Visualisierung auf bestimmte Attribute des Visualisierungsraumes abgebildet werden. Beispiele für Visualisierungsattribute sind Position, Orientierung und Farbe. So kann etwa die Energie eines Teilchens auf eine Farbskala, die Position im physikalischen Raum auf die Position im virtuellen Raum und der Geschwindigkeitsvektor auf die Orientierung der geometrischen Repräsentation des Teilchens abgebildet werden.

Visualisierungsnetzwerk Da die Visualisierungspipeline aus einer Vielzahl von Komponenten, wie Datenquellen und Filtern, bestehen kann, die komplexe Verzweigungen und Verbindungen aufweisen können, spricht man meist von einem Visualisierungsnetzwerk statt einer ”eingleisigen” Pipeline.

Stereopsis/Stereoskopie Um die Immersion zu steigern und das Verständnis bzw. die Wahrnehmung der räumlichen Struktur der Visualisierung zu unterstützen, wird in virtuellen Arbeitsumgebungen häufig auf die Stereoskopie zurückgegriffen. Das stereoskopische (auch räumliche) Sehen entsteht, indem unsere Augen leicht verschiedene Bilder betrachteter Objekte sehen. Diese parallaktisch verschobenen Bilder werden von unserem Gehirn zusammengeführt und es entsteht ein Tiefeneindruck. Die Stereoskopie (griechisch $\sigma\tau\varepsilon\rho\varepsilon\acute{o}\varsigma$ stereos = Raum/räumlich, fest – $\sigma\kappa\pi\acute{\varepsilon}\omega$ skopeo = betrachten) ist die Wiedergabe von Bildern mit einem räumlichen Eindruck von Tiefe, der physikalisch nicht vorhanden ist [Wik12b]. Um diese Stereobilder zu erzeugen, wird die Szene aus zwei leicht unterschiedlichen Blickpunkten aufgenommen. In der Computergraphik spricht man von Stereo-Rendering. Um die entstehen-

den Teilbilder auf dem jeweiligen Auge anzuzeigen gibt es verschiedene Möglichkeiten, die in Abschnitt 3.2.1 genauer betrachtet werden. Ist das Bild eines Auges auch auf dem anderen Auge (teilweise) sichtbar, so spricht man von *Ghosting* oder auch *Crosstalk*.

2.2 Gesteninteraktion

Tracking Mit dem Begriff *Tracking* beschreibt man alle Bearbeitungsschritte, die zur Verfolgung von (bewegten) Objekten notwending sind. Ziel ist es, die beobachtete Bewegung zur technischen Verwendung abzubilden [Wik12a]. Ein Beispiel ist das in dieser Arbeit eingesetzte Skelett-Tracking, bei dem markante Punkte - sog. *Joints* - einer Person getrackt werden. Dies sind meist Hände, Elbogen, Schultern und Kopf. Dadurch kann eine digitale Repräsentation des Skeletts rekonstruiert werden.

Geste Eine allgemeine Definition für den Begriff Geste findet sich bei Kurtenbach und Hulteen (1990): *A gesture is a motion of the body that contains information. Waving goodbye is a gesture. Pressing a key on a keyboard is not a gesture because the motion of a finger on its way to hitting a key is neither observed nor significant. All that matters is which key was pressed.* [KH90, s. 309-317] Harling und Edwards verstehen unter einer Geste nicht nur Bewegungen, sondern auch statische Posen [HE97, s. 75-88]. Außerdem kann man zwischen zwei Arten von Gesten unterscheiden. Einerseits kontinuierliche Gesten, bei denen die beobachtete Bewegung mit einem Zustand im Computer verbunden ist. Damit kann bspw. ein Zeiger (engl. *Cursor* oder *Pointer*) über den Bildschirm bewegt werden. Andererseits diskrete Gesten, die mit einer Aktion im Computer verknüpft sind und eine Zustandsänderung herbeiführen. [HH98, s. 292-307]

Gestenerkennung Bei der Erkennung von Gesten werden die Beobachtungsdaten der Sensorik durch verschiedene Algorithmen analysiert, um bestimmte Merkmale zu extrahieren und zu klassifizieren. Zur Aufbereitung der Beobachtungsdaten (z.B. Kamerabilder) kann ein Kalman-Filter [WB06] eingesetzt werden. Zur Klassifizierung werden dann Mustererkennungsalgorithmen und Methoden der künstlichen Intelligenz, wie Hidden-Markov-Models und neurale Netze, eingesetzt. Unterscheiden kann man zwischen geräte- und kamerabasierter Erkennung. Bei der gerätebasierten Erkennung trägt der Nutzer die Sensorik am Körper. Beispiele hierfür sind Datenhandschuhe[1] oder die Controller der Nintendo Wii[2]. Bei der kamerabasierten Erkennung ist der Nutzer frei von jeglicher Peripherie. Die Gestenerkennung erfolgt auf Basis von Einzelbildern oder Bildsequenzen, die durch eine oder mehrere Kameras aufgenommen werden. Ein solches Aufnahmegerät ist die in dieser Arbeit eingesetzte Microsoft Kinect.

[1] http://www.vrealities.com/P5.html
[2] http://www.nintendo.de/NOE/de_DE/wii_54.html

3 Konzeption einer Arbeitsumgebung zur visuellen Datenanalyse

In diesem Kapitel werden die grundlegenden Anforderungen an eine Arbeitsumgebung zur gestengesteuerten, visuellen Datenanalyse beschrieben. Zunächst wird in Abschnitt 3.1 das Ergebnis der Diplomarbeit von Lukas Zühl vorgestellt. Es bildet den Ausgangspunkt für diese Arbeit. Anschließend werden in Abschnitt 3.2 die wichtigsten Kriterien für eine immersive und intuitive Benutzeroberfläche und deren Auswirkungen auf den Entwurf des Systems erläutert. Abschnitt 3.3 gibt einen kurzen Einblick in die wissenschaftliche Datenanalyse und die damit verbundenen Anwendungsfälle. Abschließend wird der konkrete Hardware-Aufbau der Arbeitsumgebung mit seinen Kernkomponenten beschrieben.

3.1 Ausgangspunkt VIPER

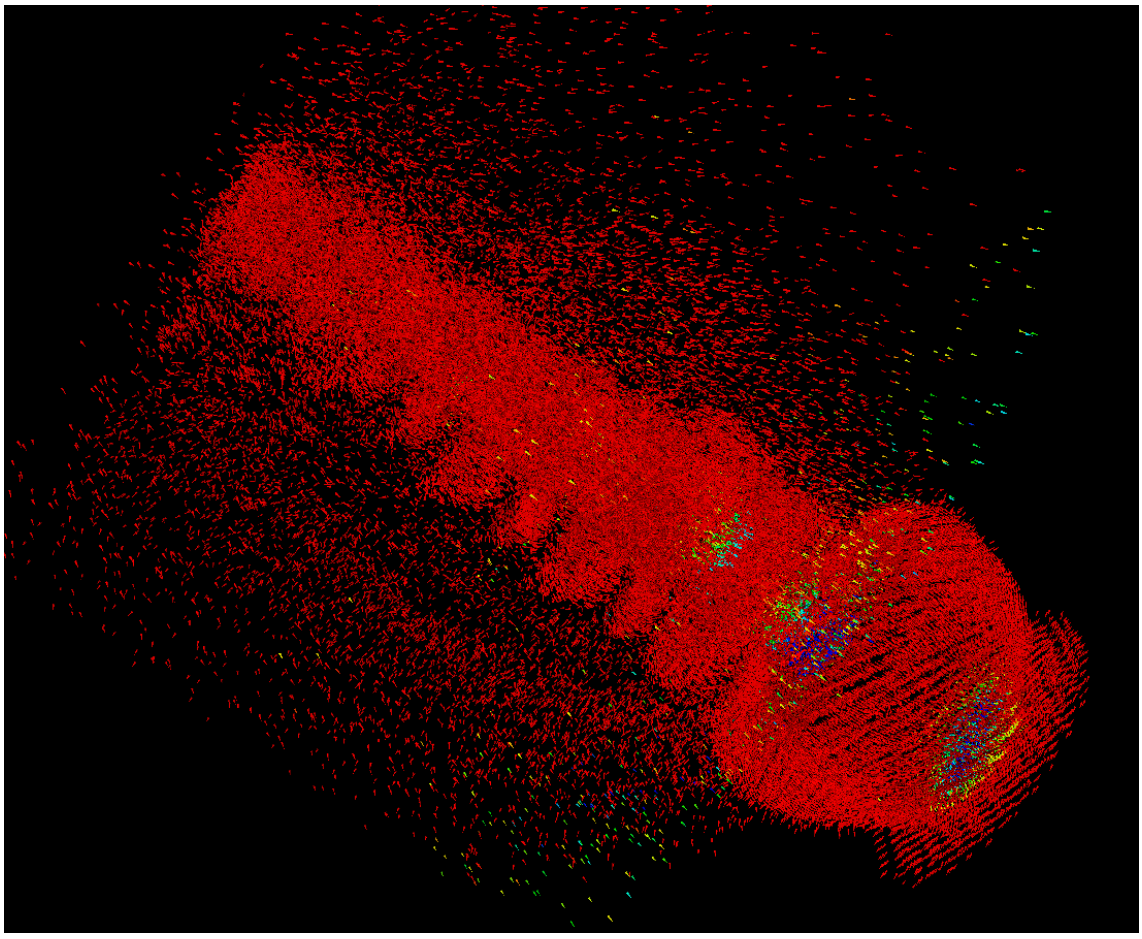


Abbildung 3.1: Ein Partikelstreudiagramm

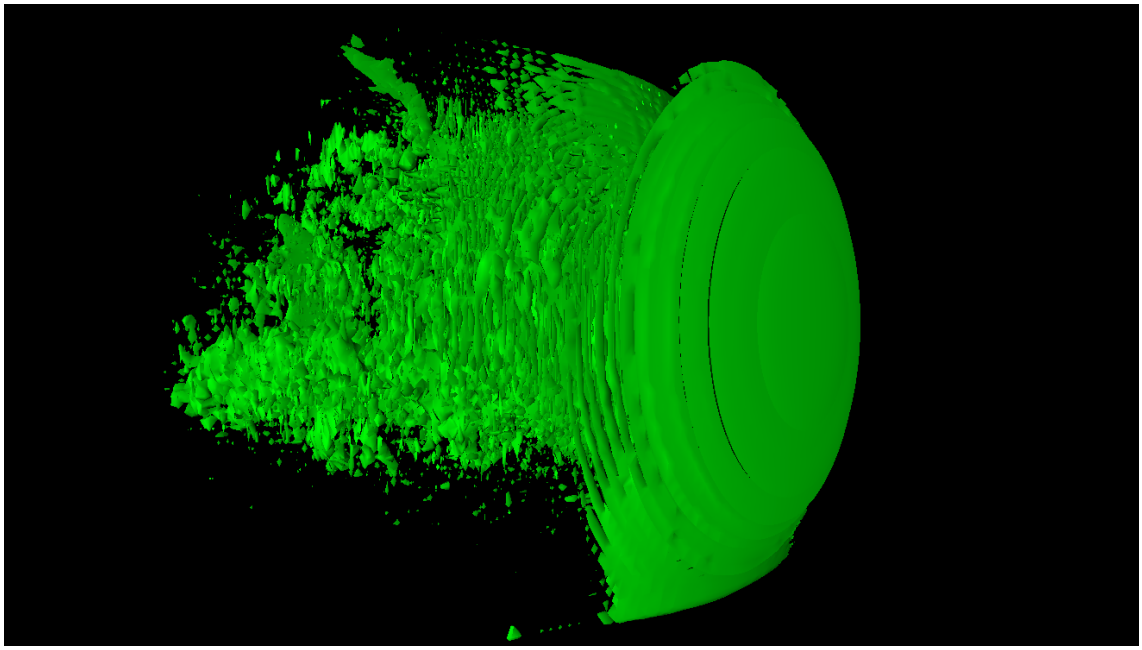


Abbildung 3.2: Eine Isoflächendarstellung eines (E-)Feldes

Wie bereits eingangs erwähnt bildet das Resultat aus der Diplomarbeit von Lukas Zühl die Grundlage dieser Arbeit. Das Projekt *Visual Interactive Plasma Computer Experiments for Researchers* (kurz VIPER) bietet Klassen zum Laden und Darstellen von Simulationsdatensätzen der Laser-Plasma-Kollision. Dazu wird zunächst ein Programm in der an SQL angelehnten und eigens für VIPER entwickelten *Visualization Query Language* (kurz VQL) erstellt. Darin wird spezifiziert, welche Daten aus welchem Datensatz geladen werden sollen und wie diese den Visualisierungsattributen zugeordnet werden. Außerdem wird die Art der Darstellung festgelegt. Auf diese Weise kann eine Visualisierung deklarativ beschrieben werden. Im Gegensatz zu VTK, wo per Programmier- oder Skriptsprache explizit die Visualisierungspipeline beschrieben wird, wird mit VQL das *Was* und nicht das *Wie* bestimmt. Die VIPER-Klasse `Visualizer` ist für den Aufbau des Visualisierungsnetzwerkes zuständig. Sie parst dazu das VQL-Programm mit Hilfe der Klasse `vql::Parser`. Als Rückgabe erhält sie ein Objekt vom Typ `vql::Program`, welches sie als Parameter dem Konstruktor der `Visualization`-Klasse übergibt. Diese erstellt nun eine oder mehrere Instanzen der konkreten Unterklassen von `AbstractDataSource` sowie `AbstractVisPipe`. Die von `AbstractVisPipe` abgeleiteten Klassen implementieren verschiedene Visualisierungen in VTK. Beispielsweise wird durch die `ScatterPlotVisPipe`-Klasse ein 3D-Punkt-Diagramm mit (orientierten) Glyphen realisiert. Die Abbildungen 3.1, 3.2 und 3.3 zeigen drei Visualisierungen die mit VIPER entstanden sind.

Die VIPER-Visualisierungspipeline ist in Abbildung 3.4 dargestellt. Sie implementiert die in Kapitel 2 beschriebene Referenzpipeline mit eigenen Klassen und unter Nutzung vorhandener VTK-Algorithmen und Filter. Dabei lassen sich die einzelnen Anweisungen im VQL-Programm direkt auf die Stufen der Pipeline abbilden (siehe Abb. 3.4). Die von `AbstractDataSource` abgeleiteten Klassen implementieren Laderoutinen für verschiedene Simulationsdatenformate. In dieser Arbeit wird hauptsächlich die `illumination::DataSource` genutzt. Diese nutzt wiederum Klassen wie `illumination::VtkGridToImageDataSource` oder `illumination::VtkParticleToPolyDataSo-`

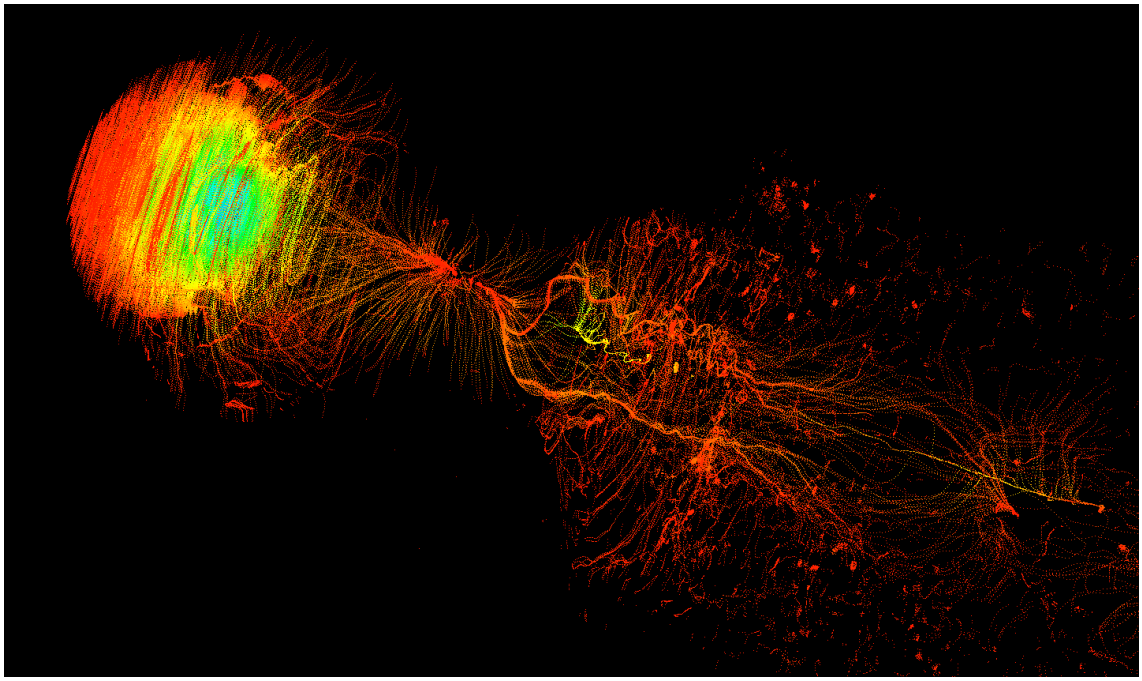


Abbildung 3.3: Ein Vektorfeld visualisiert durch Stromlinien

`urce` zum Laden der verschiedenen Datenfelder der Simulationsdatensätze in die entsprechenden VTK-Datenstrukturen (`vtkImageData` oder `vtkPolyData`).

Mit VIPER ist es also möglich, Simulationsdaten auf flexible Weise zu visualisieren und zu betrachten. Was noch fehlt sind Möglichkeiten zur interaktiven Bearbeitung der Daten und ein natürliches Nutzerinterface zur Steuerung des Visualisierungssystems.

3.2 Anforderungen an Immersion, Intuitivität und Usablility

Beim Entwurf einer immersiven, virtuellen Umgebung und einer intuitiven Benutzerschnittstelle gilt es eine Vielzahl von Faktoren zu beachten. Im Folgenden wird erläutert, welche Richtlinien und Kriterien die Gestaltung des User Interface und die Wahl der Eingabegesten beeinflusst haben.

3.2.1 User Interface & Stereo-Rendering

Bei der Gestaltung der Benutzeroberfläche mussten zwei gegensätzliche Aspekte vereint werden. Einerseits soll das Interface so minimal wie möglich gehalten werden, um die Aufmerksamkeit des Nutzers nicht vom eigentlichen Untersuchungsobjekt abzulenken und die Immersion aufrecht zu halten. Andererseits muss es auch ein ausreichendes Feedback bieten, da die Gestenerkennung nicht immer hundertprozentig korrekt funktioniert sowie ein Minimum an Hilfe bieten um die Interaktionsmöglichkeiten aufzuzeigen. Als Designrichtlinien wurden dazu die zehn Usability-Heuristiken von Jakob Nielsen herangezogen:

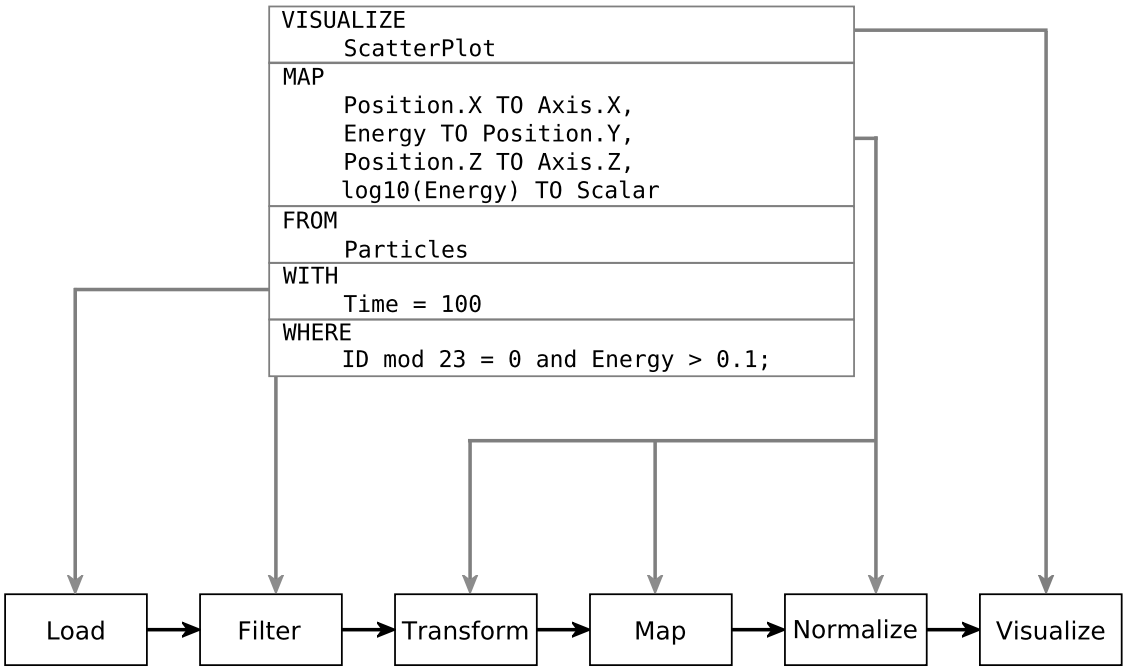


Abbildung 3.4: Die Visualisierungspipeline von VIPER mit zugehörigen VQL-Statements

Visibility of system status

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Match between system and the real world

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

User control and freedom

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

Consistency and standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

Error prevention

Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Recognition rather than recall

Minimize the user's memory load by making objects, actions, and options visible. The user should

not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Flexibility and efficiency of use

Accelerators - unseen by the novice user - may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

Aesthetic and minimalist design

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Help users recognize, diagnose, and recover from errors

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Help and documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large. [Nie12]

Wie sich diese Heuristiken in Funktionalität und Gestaltung des Systems wiederspiegeln wird im Folgenden beschrieben.

Da durch fehlerhaft erkannte Gesten ungewollt Aktionen ausgelöst werden können, muss ein Undo-Mechanismus umgesetzt werden, der Aktionen wie das Erstellen und Anwenden von Tools (siehe Abschnitt 4.4.4) rückgängig machen kann.

Um den Nutzer ausreichend Feedback über den Systemzustand zu liefern werden Icons und ein Mini-Skelett am unteren Bildrand angezeigt, die aktive und mögliche Interaktionen sowie den Trackingstatus wiedergeben.

Um eine größtmögliche Immersion zu erreichen, wird die Visualisierung stereoskopisch auf eine Powerwall gerendert. Es gibt verschiedene Ansätze um das entsprechende Teilbild eines Stereopaares für das jeweilige Auge sichtbar zu machen und für das andere Auge auszublenden. Beim aktiven Stereorendering werden abwechselnd die Teilbilder für jeweils ein Auge gezeigt. Zum Betrachten eines aktiven Stereodisplays wird eine sog. Shutter-Brille benötigt, die synchron zum angezeigten Teilbild das entsprechende Auge durch die Brille blicken lässt und gleichzeitig das andere Auge abdunkelt. Hierfür sind Anzeigegeräte mit einer hohen Bildwiederholfrequenz erforderlich, da durch das Umschalten zwischen den Augen die Bildrate halbiert wird. Um also eine Nettobildfrequenz von 60 Hz zu erreichen muss das Display eine Bildwiederholrate von 120 Hz bieten. Im Fall der sog. anaglyphen Stereotechnik dagegen wird das Bild für beide Augen gleichzeitig angezeigt. Die Teilbilder des Stereopaares werden dazu in einem Vorverarbeitungsschritt farbkodiert (meist in Komplementärfarben) und dann übereinandergelegt. Das Trennen der Bilder erfolgt mit einer speziellen Brille, deren Gläser aus Farbfiltern bestehen. Eine dritte Technik, die im in Abschnitt 3.4 beschriebenen Aufbau Anwendung findet ist das passive Stereorendering. Dabei werden die Stereobilder durch zwei Beamer auf eine Leinwand (Powerwall) projiziert. Direkt hinter den

Beamerlinsen befinden sich Polarisationsfilter, die das ausgestrahlte Licht orthogonal zueinander linear polarisieren. Zur Betrachtung reicht nun eine gleichartig polarisierte Brille aus. Dadurch wird ein minimal invasives Stereodisplay realisiert. Lediglich autostereoskopische Displays, welche mit speziellen Gittern aus Linsen auf der Displayoberfläche arbeiten, kommen ganz ohne Brille aus.

In jedem Fall ist zu beachten, dass es zu unerwünschten Effekten wie der sog. *Stereo Window Violation* kommen kann. Dabei ragt das gerenderte 3D-Objekt über die Ränder des Anwendungsfensters hinaus und wird in Folge dessen abgeschnitten. Dieser Effekt stört den Tiefeneindruck besonders bei Objekten die aus der Leinwand herauszukommen scheinen, da diese vom Leinwandrahmen verdeckt werden, obwohl sie vor ihm zu liegen scheinen. In Abbildung 3.5 ist ein Beispiel hierfür zu sehen.

Abbildung 3.5: Stereo Window Violation in Anaglyph-Darstellung (Effekt sichtbar mit Rot-Blau- oder Rot-Grün-Brille)

Ein weiterer Effekt, der die Augen belastet, ist die falsche Ausrichtung der virtuellen Aug- und Fokuspunkte. Der naive Ansatz ist, die beiden Augpunkte um einen bestimmten Wert nach rechts bzw. links von der virtuellen Beobachterposition zu verschieben. Dies führt zur sog. Offset-Technik. Durch die versetzten Sichtkegel entstehen Regionen, welche nur von jeweils einem Auge gesehen werden. Das belastet die Augen, da manche Objekte nur auf einem Auge sichtbar sind (siehe Abb. 3.6). Richtet man die virtuellen Kameras nun auf einen gemeinsamen Fokuspunkt auf der optischen Achse in Blickrichtung aus, so spricht man von der *Toe-In*-Technik (siehe Abb. 3.7). Diese führt zu einer ungewollten vertikalen Parallaxe und strapaziert die Augen. Um dies zu vermeiden, sollte die *Off-Axis*-Methode eingesetzt werden. Dabei sind die Sichtstrahlen der virtuellen Kameras parallel, wobei die Sichtkegel (engl. *View Frustum*)

so geschert werden, dass sie nicht mehr symmetrisch sondern schief sind (siehe Abb. 3.8). [Gra]

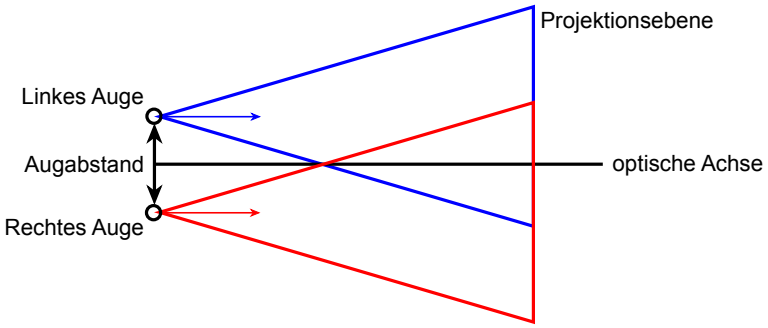


Abbildung 3.6: Offset-Methode mit versetzten Sichtkegeln

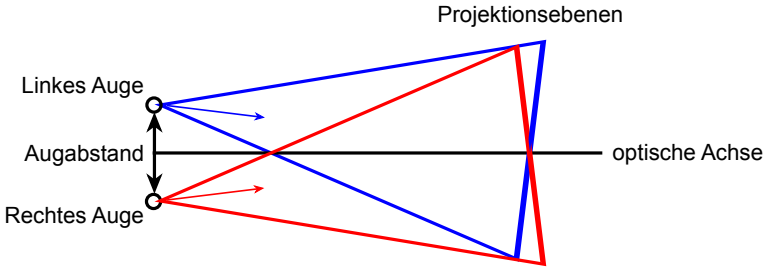


Abbildung 3.7: Toe-In-Methode mit verschiedenen Projektionsebenen

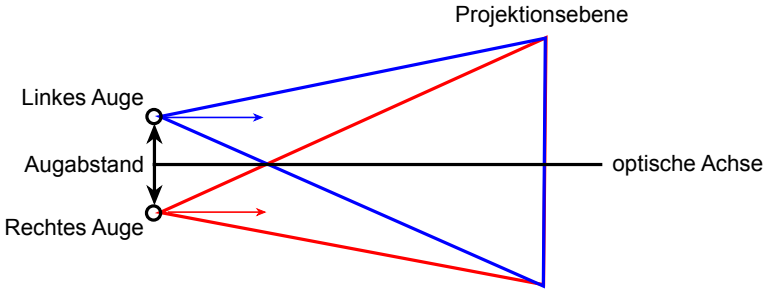


Abbildung 3.8: Off-Axis Methode mit gemeinsamer Projektionsebene beider Augen und schiefen Sichtkegeln

Die perspektivische Projektionsmatrix ist dabei wie folgt definiert:

$$P_{sym} = \begin{pmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & \frac{n+f}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Beschrieben wird sie durch folgende Parameter, welche die Lage der das Sichtvolumen begrenzenden Ebenen (engl. *clipping plane*) definieren:

- $z = n$ Abstand der Near Clipping Plane
- $z = f$ Abstand der Far Clipping Plane
- $y = b$ Y-Koordinate der Bottom Clipping Plane am Schnittpunkt mit der Near Clipping Ebene

- $y = t$ Y-Koordinate der Top Clipping Plane am Schnittpunkt mit der Near Clipping Ebene
- $x = l$ X-Koordinate der Left Clipping Plane am Schnittpunkt mit der Near Clipping Ebene
- $x = r$ X-Koordinate der Right Clipping Plane am Schnittpunkt mit der Near Clipping Ebene

Wobei für ein symmetrisches Sichtvolumen gilt:

- $b = -t$
- $l = -r$.

Diese wird mit einer Scherung um den negativen halben Augabstand für das linke Auge bzw. den positiven halben Augabstand für das rechte Auge entlang der X-Achse

$$M_{shear} = \begin{pmatrix} 1 & 0 & \mp\frac{d}{2} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

mit $d = Augabstand$, wie folgt verkettet:

$$P_{oblique} = P_{sym} * M_{shear}.$$

Daraus ergibt sich die gescherte Projektionsmatrix, welche ein schiefes Sichtvolumen beschreibt:

$$P_{oblique} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \mp\frac{d}{2} & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & \frac{n+f}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

3.2.2 Wahl der Gesten

Bei der Gestaltung des Interaktionskonzeptes wurden vorallem die Aspekte Ergonomie, Konformität zur Intuition des Nutzers und technische Machbarkeit berücksichtigt.

Zur intuitiven Steuerung des Systems, empfiehlt es sich Eingabegesten zu verwenden, die dem Nutzer aus der realen Welt bekannt sind (siehe Nielsen-Heuristiken *'Match between system and the real world'*). So werden für die Steuerung der virtuellen Kamera - also die Navigation - eine Kombination aus Ein- und Zweihandgesten und Fingererkennung verwendet. Dabei wird nur eine Geste erkannt, wenn eine oder beide Hände geschlossen sind. Dies vermittelt dem Nutzer den Eindruck, er könne die virtuellen Objekte greifen und direkt manipulieren. Beispiele dafür sind das zweihändige Rotieren um die X- und Y-Achse durch entgegengesetztes Auf- und Ab- bzw. Vor- und Zurückbewegen der geschlossenen Hände. Oder auch das einhändige "Greifen" und Bewegen zur Translation. Diese Gesten sind intuitiv, da sie ein physikalisches Äquivalent in der realen Welt besitzen.

Der Nutzer soll außerdem in der Lage sein, die verfügbaren Aktionen schnell zu identifizieren und nicht durch zu viele gleichzeitig mögliche Gesten überfordert werden. Um die Anzahl an Eingabegesten mög-

lichst gering zu halten, können dieselben Gesten für mehrere Aktionen verwendet werden. Gleiche Gesten sollten dabei gleiche oder ähnliche Aktionen auslösen. Um eine solche ”Mehrfachbelegung” von Gesten zu verarbeiten, muss dem System die Intention des Nutzers bekannt sein, mit welcher er die Geste ausführt. Zur Bestimmung der Intention muss ein *Kontext* festgelegt werden, repräsentiert durch einen bestimmten Systemzustand. Dazu wurden in dieser Arbeit sog. *Modi* eingeführt. Diese werden in Abschnitt 4.4.3 genauer beschrieben.

Einer dieser Modi ist für die Manipulation des Blickpunktes zuständig, der *Navigation Mode*. Hier wird die Einhand-Translationsgeste zum Verschieben der virtuellen Kamera verwendet. Im *Manipulation Mode*, welcher zum Bearbeiten von Tools dient, kann diese Geste dagegen zum Verschieben einer Schnittebene benutzt werden. Dabei wird immer nur ein Freiheitsgrad manipuliert. D.h. wenn eine Translation entlang der X-Achse erfolgt, wird die Y- und Z-Koordinate konstant gehalten. Dies soll das präzise Navigieren unterstützen und ungewollte Manipulationen, hervorgerufen durch Rauschen auf den Kinect Trackingdaten, minimieren. Zudem kann der Nutzer eine größere Anzahl an sich gleichzeitig verändernden Freiheitsgraden nur schwer beherrschen [BKJJLP04].

Das Wechseln zwischen den Modi erfolgt über einen zentralen Steuermodus, *Task Selection Mode* genannt. Da abstrakte Aktionen wie das Wechseln in einen Navigationsmodus keine Entsprechung in der realen Welt haben und in dieser Arbeit kein multimodales Interface eingesetzt wird, welches z.B. eine Sprachsteuerung nutzen könnte, wird hier ein Umweg über symbolische Gesten gemacht. Dabei werden im Task Selection Modus die Bewegungen der geschlossenen rechten Hand aufgezeichnet und als ein gezeichnetes Symbol erkannt. Ein gemaltes ”N” wechselt in den Navigation Mode, ein ”M” in den Manipulation Mode, ein ”C” in den Creation Mode und ein ”L” in den Load/Configure Mode. Das Kreuzen der Arme vor der Brust in Form eines ”X” führt vom jeweiligen Modus immer zurück zum Task Selection Mode. Für diesen speziellen Fall spielt die Handhaltung keine Rolle.

Wie bereits erwähnt werden (fast) alle Interaktionen mit geschlossenen Händen ausgeführt. Dies fördert einerseits das Gefühl der direkten Interaktion mit den virtuellen Objekten (das Greifen), andererseits erlaubt es dem Nutzer seine Arme auszuruhen und hängen zu lassen, ohne unbeabsichtigt Aktionen auszulösen. Dabei ist es wichtig, dass sich der Nutzer nicht zu weit vom Kinect Sensor entfernt befindet. Ansonsten reicht die Auflösung der Kamera nicht mehr aus um eine stabile Fingererkennung zu garantieren, da die Handkontur mit zunehmender Entfernung immer weniger Pixel auf dem Kamerabild einnimmt. Auf Grund der technischen Einschränkungen der Kinect und je nach Physiologie des Benutzers (Größe der Hand) und Armhaltung beim Ausführen der Gesten sollte ein Abstand von 1.2 m bis 2 m eingehalten werden. Es hat sich gezeigt, dass ein Abstand von etwa 1.4 m optimal ist, da die Leinwand dann das Blickfeld des Nutzers gut ausfüllt, ohne das dieser übermäßig große Kopfbewegungen ausführen muss. Dies fördert die Immersion ebenso wie das verwendete kamerabasierte Tracking, welches ohne Sensorik am Nutzer auskommt.

3.3 Wissenschftliche Datenanalyse

Wissenschaftliche Datenanalysen sind nach Meyers Lexikon *”numerische und statistische Verfahren zur Aufdeckung von Strukturen in großen Datenmengen”* [mey02]. Im Kontext dieser Arbeit soll es genauer

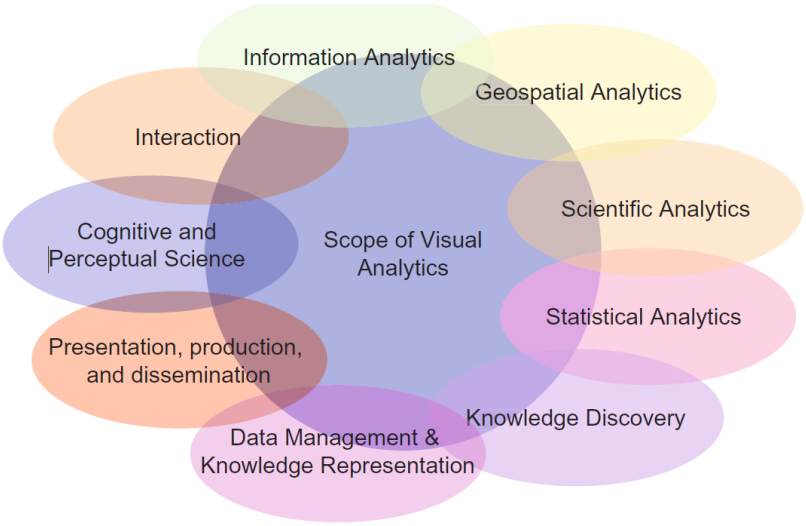


Abbildung 3.9: Visual Data Analytics nach [KMS+08]

um die visuelle Analyse von Daten gehen. Unter dem Stichwort *Visual Analytics* verstehen Thomas und Cook (2005) *"the science of analytical reasoning facilitated by interactive visual interfaces."*[TC05] Der Aspekt Interaktivität spielt also eine gewichtige Rolle bei der visuellen Datenanalyse. Der Nutzer soll nicht mit statischen Diagrammen und fertigen Analyseresultaten konfrontiert werden, sondern selbst den computergestützten Analyseprozess steuern.

Einen Überblick über das interdisziplinäre Feld der Visual Data Analytics gibt Abbildung 3.9.

Es gibt also eine Vielzahl von Anwendungsfällen bzw. Bearbeitungszielen, die man mit einer interaktiven Visualisierung erfüllen möchte. An erster Stelle ist dies die Exploration, d.h. das freie Betrachten und Erforschen der virtuellen Umgebung in welcher sich die Visualisierung befindet. Diese initiale Betrachtung der Daten dient dazu, einen Überblick zu gewinnen, evtl. Trends und Ausreißer auszumachen und Hypothesen aufzustellen.

Außerdem versucht man durch Messung von bestimmten Größen und das Abschätzen von Verhältnissen, aufgestellte und vorhandene Annahmen zu überprüfen. Man sucht also nach einer Bestätigung für Vermutungen die man über die innere Struktur der Daten hat.

Nichtzuletzt dient die Visualisierung und die visuelle Datenanalyse der Kommunikation von Methoden und Forschungsergebnissen. Eine klare Darstellung der gesammelten Informationen zur Kommunikation mit Dritten und eine Beschreibung der eigenen Vorgehensweise ist das Ziel.

Um den Anwender bei der Bewältigung dieser Aufgaben so gut wie möglich zu unterstützen, braucht er eine Palette an Werkzeugen (engl. *Tools*), mit der er in der virtuellen Umgebung arbeiten kann. Dazu gehört die Möglichkeit zur freien Navigation durch die virtuelle Umgebung, welche Funktionen zum Verschieben, Rotieren und Zoomen des virtuellen Beobachter- und Fokuspunktes einschließt.

Desweiteren werden Werkzeuge zur Datenreduktion und -auswertung gebraucht. Diese sollten die Reduktion der Daten auf visuell ansprechende und verständliche Weise umsetzten.

Selbstverständlich kann die Exploration, Reduktion und Auswertung der Daten auch als Präsentation vor einem größeren Publikum durchgeführt werden. Trotz des verwendeten Single-User-Ansatzes ist eine

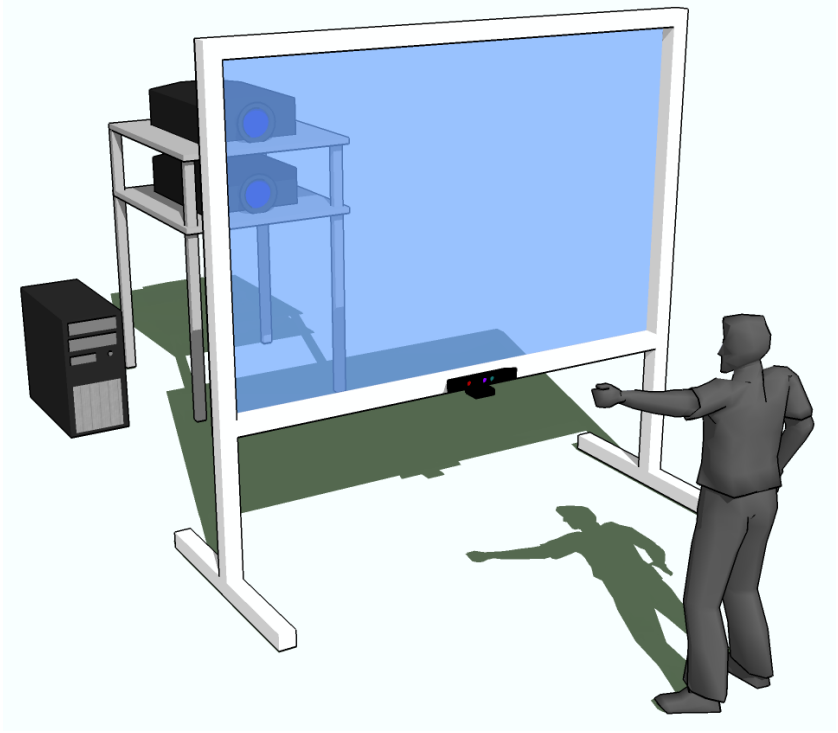


Abbildung 3.10: Aufbau der Hardwarekomponenten im VR-Labor des HZDR

Zusammenarbeit bspw. in Form einer Diskussionsrunde möglich und sinnvoll.

3.4 Aufbau der Arbeitsumgebung

In Abbildung 3.10 ist der Aufbau einer zur visuellen Datenanalyse geeigneten Arbeitsumgebung schematisch dargestellt. Das Stereobild wird dabei mit Hilfe zweier Projektoren (engl. *Beamer*) auf einer sog. *Powerwall* angezeigt. Die als Trackinggerät eingesetzt Microsoft Kinect befindet sich mittig am unteren Rahmen der Powerwall und ist leicht nach oben ausgerichtet, um den Bereich vor der Wand optimal erfassen zu können. Hinter der Powerwall stehen die beiden Beamer. Diese sollten möglichst gerade und orthogonal zur Projektionsleinwand positioniert werden, um ein unverzerrtes Bild zu zeigen. Die Beamer und der Kinect Sensor sind mit einem leistungsstarken Rechner verbunden, auf welchem das Gestentracking und das Rendering der Visualisierung erfolgt. Der Nutzer steht im Raum vor der Leinwand, welcher möglichst frei von anderen Personen und Gegenständen sein sollte. Im Folgenden sollen die wesentlichen Hardwarekomponenten kurz beschrieben werden.

Powerwall Mit dem Begriff Powerwall wird i. Allg. eine große und hochauflösende Anzeigefläche bezeichnet, auf der computergenerierte Bilder dargestellt werden. In unserem Fall wird diese Anzeige durch eine 2 m $\times$ 1.26 m große Rückprojektionsleinwand realisiert. Das Material der Leinwand ist dabei so beschaffen, dass es keinen Hotspot entstehen lässt. Desweiteren ist es polarisationserhaltend, was Voraussetzung für das passive Stereorendering ist. Angestrahlt wird die Wall von zwei Projektoren der

Marke *Christie*[1] mit einer maximalen Auflösung von 1920×1200 Pixel. Direkt vor deren Linsen befinden sich Polarisationsfilter, welche das von den Beamern ausgestrahlte Licht horizontal respektive vertikal polarisieren, bevor es auf die Powerwall trifft. Um das Stereobild zu betrachten reicht nun eine einfache Polbrille aus.

Microsoft Kinect Die Kinect wurde ursprünglich als Steuerung (engl. *Controller*) für die Microsoft XBox 360 konzipiert. Entwickelt wurde sie in Zusammenarbeit mit PrimeSense[2] unter dem Codenamen "Project Natal". Mit ihr lassen sich Spiele auf der XBox durch Körperbewegungen steuern. Mittlerweile gibt es auch ein offizielles SDK für die Windows-Plattform sowie offene Treiber und Bibliotheken, welche die Entwicklung von Software auf allen Plattformen unterstützen. Die Kinect besteht aus einer 32 Bit RGB-Kamera mit einer Auflösung von 640×480 Pixel bei einer Framerate von 30 Hz. Dazu kommt eine 830 nm Laserdiode, welche als Projektor dient und die Umgebung vor der Kinect mit strukturiertem Infrarotlicht ausstrahlt. Dieses IR-Licht wird von einer weiteren Kamera zur 3D Rekonstruktion und Tiefenbildgenerierung genutzt. Das Tiefenbild wird dabei 30 mal in der Sekunde erzeugt und hat eine Auflösung von 320×240 Pixel mit einer Farbtiefe von 16 Bit. Der nutztbare Tiefenbereich liegt bei etwa 1.2 m bis 3.5 m Entfernung zur Kamera. Hinzu kommen noch vier Mikrofone die einen 16 Bit Audiostream mit einer Frequenz von 16 kHz liefern. Diese können zur Sprachsteuerung und für den Live-Chat genutzt werden. [pla12]

PC-Hardware Als Testsystem kommt ein High-End-PC mit einem Ubuntu 11.10 Linux-Betriebssystem zum Einsatz, auf dem alle benötigten Treiber und Bibliotheken installiert sind. Verbaut sind ein Intel Xeon X5650 Prozessor, eine Nvidia Quadro 5000 sowie 12 GB RAM. Die leistungsstarke CPU ermöglicht im Zusammenspiel mit der High-End-Grafikkarte das Laden und Filtern bzw. Rendern großer Datensätze. Aufgrund der Streaming-Fähigkeit der VTK-Pipeline (umgesetzt durch die spezialisierte *vtkExecutive*-Klasse *vtkStreamingDemandDrivenPipeline*, welche die Pipelineausführung steuert) können theoretisch auch Datenmengen verarbeitet werden, die die Größe des Hauptspeichers überschreiten. Allerdings können dann nicht immer interaktive Frameraten gewährleistet werden und es kommt zu einem sichtbar langsamen bzw. stückweisen Bildaufbau.

[1] http://www.christiedigital.com/en-us/business/products/projectors/1-chip-dlp/pages/christie-dwu670-e.aspx
[2] http://www.microsoft.com/en-us/news/press/2010/mar10/03-31PrimeSensePR.aspx

4 Implementierung

4.1 Systemarchitektur

In diesem Abschnitt werden zunächst die Komponenten, aus welchen das Gesamtsystem besteht, und deren Zusammenspiel erläutert. Anschließend werden die zu deren Umsetzung genutzten Bibliotheken kurz in ihrer Funktionalität und Verwendung beschrieben.

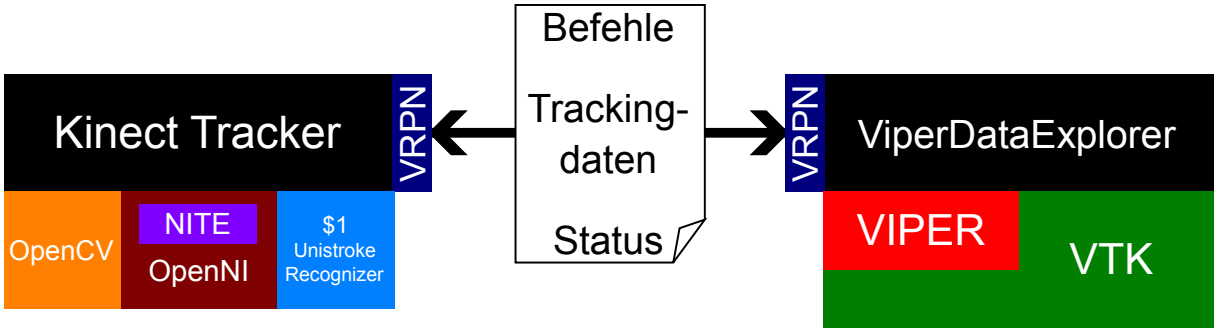


Abbildung 4.1: Systemkomponenten

4.1.1 Komponenten

Das zur gestengesteuerten visuellen Datenanalyse entwickelte System besteht im Wesentlichen aus zwei Softwarekomponenten, gezeigt in Abbildung 4.1. Zum einen der Kinect-Tracker, der für Erkennung von Gesten und Nutzer sowie für die Erzeugung von Befehlen zuständig ist. Diese durch ein Kommunikationsprotokoll definierten Befehle werden an die zweite Systemkomponente - die Visualisierung - gesendet und dort verarbeitet.

Die Trennung von Frontend und Backend, also Kinect-Tracker und Visualisierung, und deren Kommunikation über ein Netzwerkprotokoll - in unserem Fall UDP bzw. die VRPN-Middleware - ermöglicht eine Verteilung des Gesamtsystems auf mehrere Computer. So kann der Kinect-Tracker auf einem Rechner mit geeigneter Hardware zur Bildverarbeitung laufen, während die Visualisierung auf einer leistungsstarken Maschine mit professioneller Grafikhardware und hoher Speicherkapazität ausgeführt werden kann. Die beiden Komponenten müssen sich auf diese Weise nicht die vorhandene Rechenkapazität auf einem PC teilen, sondern können die volle Leistung zweier Einzelrechner nutzen. Performancelastige Kontextwechsel auf der Grafikkarte (wie z.B. das Neuladen von Shader-Programmen u.ä.) können so minimiert werden.

Ein weiterer Vorteil besteht in der Entkopplung und Wiederverwendbarkeit der Komponenten. Der Tracker

kann theoretisch jede beliebige Softwarekomponente steuern, die das verwendete Kommunikaitonsprotokoll implementiert. Ebenso kann die Visualisierung von jeder beliebigen Frontend-Software gesteuert werden, die die entsprechenden Befehle sendet. Ein Beispiel hierfür liefert ein zu Testzwecken implementierter Client, der die in ein Textfeld eingegebenen Kommandos per UDP an den Standard-Port der Visualisierungskomponente sendet (siehe Abb. 4.2).

Abbildung 4.2: Test-Client zur Steuerung der Visualisierung

4.1.2 Bibliotheken

Zur Implementierung der Tracker- und Visualisierungskomponente sowie zur Umsetzung deren Kommunikation über ein lokales Netzwerk wurden die im Folgenden beschriebenen Bibliotheken (engl. *libraries*) verwendet.

VTK Das *Visualization Toolkit* ist eine frei verfügbare, Open-Source-Software die von der Firma *Kitware Inc.* entwickelt wird. Sie bietet eine umfangreiche Klassenbibliothek zur Bildverarbeitung und wissenschaftlichen Visualisierung. Neben der nativen C++-Schnittstelle, gibt es auch die Möglichkeit über interpretierte Sprachen wie Java, Python und Tcl auf die umgesetzten Routinen und Algorithmen zuzugreifen. Dies sind u.a. Visualisierungs- und Modellierungsalgorithmen wie Mesh Glättung, Polygonreduktion, Cutting, Contouring und Delaunay Triangulation. Dazu bietet VTK eine Reihe von 3D Widgets und unterstützt die parallele Verarbeitung von Daten. VTK ist plattformübergreifend und läuft auf Windows, Linux, Mac und Unix-Plattformen. [SML98] Wie aus dem Komponentenschema in Abbildung 4.1 hervorgeht, wird VTK indirekt, über die Verwendung von VIPER, sowie direkt in der Visualisierung selbst benutzt.

VRPN Das *Virtual Reality Peripheral Network* ist eine von der University of North Carolina entwickelte Netzwerk-Middleware. Sie bietet für eine Vielzahl von Peripheriegeräten aus dem Umfeld der virtuellen Realität eine einfache und transparente Netzwerkschnittstelle. Dazu wird eine Client-Server-Architektur zwischen Applikation und Trackinggerät umgesetzt. Auf diese Weise bleibt die Netzwerktopologie vor den (VR-)Anwendungen verborgen, sodass Client und Server auf verschiedenen oder aber auch auf ein und demselben Rechner laufen können. VRPN stellt verlorene Verbindungen von selbst wieder her und bietet die Möglichkeit zur Synchronisation zwischen verschiedenen Anwendungen. Übertragene Nachrichten können durch Zeitstempel datiert werden. Dabei laufen Client und Server in getrennten Prozessen, asynchron und nicht-blockierend [IHS+01]. Eingesetzt wird VRPN als Netzwerkschnittstelle

zwischen Kinect-Tracker und Visualisierung. Befehle, Trackingdaten und Statusinformationen werden damit ausgetauscht.

OpenNI *Open Natural Interaction* ist eine von mehreren Unternehmen gegründete Non-Profit-Organisation, mit dem Ziel die Kompatibilität und Interoperabilität zwischen Geräten zur natürlichen Interaktion, Anwendungen und Middleware zu fördern und zertifizieren. Hauptanliegen ist dabei die Einführung solcher Geräte in den Markt zu beschleunigen, indem eine Standard-API für Entwickler spezifiziert wird. Wichtige Partner in dieser Kooperation sind die Firma PrimeSense, welche die Technologie für die Kinect entwickelt hat sowie ASUS, die inzwischen in Zusammenarbeit mit PrimeSense ein Kinect-ähnliches Gerät für den PC veröffentlicht hat[1].

NITE Die *NITE* Middleware implementiert die OpenNI API und stellt Funktionalität, wie die Erkennung von Gesten und das Skelett-Tracking, zur Verfügung.

OpenCV Die *Open Source Computer Vision Library* ist eine freie, quelloffene Bibliothek zur (Echtzeit-) Bildanalyse und -verarbeitung (engl. *computer vision*)[2]. Eingesetzt wird sie im Tracker, um aus dem Kinect-Kamerabild die Handkontur zu extrahieren und die Fingererkennung zu realisieren.

$1 Unistroke Recognizer Der *One-Dollar Unistroke Recognizer* bietet eine leichtgewichtige Möglichkeit, simple zweidimensionale Gesten zu erkennen. Konzipiert wurde er für den Einsatz in interaktiven Anwendungen bzw. deren User Interfaces. [WWL07] In unserem Fall kommt er im Kinect-Tracker zum Einsatz, um einhändig ”gezeichnete” Gesten zu erkennen.

4.2 Kinect-Tracker

Der Kinect-Tracker stellt das Frontend des Gesamtsystems dar. Er ist verantwortlich für die Erkennung von Nutzereingaben in Form von Gesten und für das Tracking des Nutzers als solchen. Die erkannten Eingabegesten werden in Kommandos, entsprechend dem Kommunikationsprotokoll, umgewandelt und per VRPN an das Visualisierungsbackend gesendet. Die durch Skelett-Tracking erfassten Joint-Positionen werden als Rohdaten übertragen. Im Gegenzug empfängt der Tracker von der Visualisierungskomponente Nachrichten über Statusänderungen.

Das Interaktionskonzept beinhaltet eine Vielzahl von Interaktionsszenarien, welche direkt mit dem Modus des Viper Data Explorer verknüpft sind. Um ein optimales Tracking zu garantieren und gleichzeitig die Komplexität der Tracker-Implementierung minimal zu halten, verwendet der Tracker wie auch der VDX intern in Klassen gekapselte Zustände (engl. *state*, siehe State-Pattern [GHJV94]). Diese Tracking-States verwenden verschiedene Tracking-Strategien. In unserem Fall werden die `SkeletonTrackingStrategy` und die `HandTrackingStrategy` genutzt. Diese sind nach dem Strategy-

[1] http://www.asus.de/Multimedia/Motion_Sensor/Xtion_PRO/
[2] http://opencv.willowgarage.com/wiki/

Pattern aus [GHJV94] implementiert. Sie legen fest, wie die Trackingdaten erhoben und wie aus diesen Rohdaten Gesten extrahiert werden.

Der Kinect-Tracker ist dabei nur ein Teil des am HZDR entwickelten Frameworks zur natürlichen Interaktion, genannt VRIF (*Virtual Reality Interaction Framework*, siehe [Skr12]).

4.3 Befehlsprotokoll zur Steuerung der Visualisierung

Die Kommunikation zwischen Frontend und Backend, also der Systemkomponente, welche die Nutzereingaben entgegen nimmt und verarbeitet, und der Visualisierung, welche die vom Nutzer gewünschten Aktionen ausführt und die Daten darstellt, erfolgt über einen definierten Satz an Kommandos. Die werden entweder per UDP oder VRPN übertragen. Dabei werden vom Tracker zwei Arten von Befehlsdaten an die Visualisierung gesendet.

Erstens vom Tracker erkannte Gesten wie *SwipeLeft* für ein Schwenken der rechten Hand nach links oder *Push* für eine schnelle Vorwärtsbewegung. Desweiteren gibt es Kommandos für Gesten die zusätzliche Parameter haben wie z.B. *spread: 0.1*, welche eine beidhändige Geste beschreibt, wobei beide Hände geschlossen werden und anschließend voneinander weg oder aufeinander zu bewegt werden. Der Parameter variiert hierbei nach dem pro Zeit veränderten Abstand der Hände. Diese Art der Befehle verarbeitet also die vom physischen Trackinggerät gelieferten Daten und fügt ihnen Semantik hinzu, indem sie verschiedene Trackingstrategien einsetzt um Gesten zu extrahieren und zu parametrisieren.

Hierbei wurde darauf geachtet, die Kommandos so generisch und abstrakt wie möglich zu halten, d.h. der Tracker liefert als Kommando nur was er erkennt, sagt jedoch nicht, welche Aktion darauf folgen soll. Anders wäre dies wenn der Tracker konkrete Kommandos wie ”Bewege Schnittebene” oder ”Bewege Kamera” generieren würde. Diese Abstraktion erlaubt es dem konkreten Backend (hier der Visualisierung) die Kommandos selbst zu interpretieren und nach dem jeweiligen Systemzustand und Kontext für die gleichen Kommandos verschiedene Aktionen auszuführen.

Die zweite Art von Daten werden direkt als Rohdaten gesendet. Auf diese Weise werden die Joint-Positionen des vom Tracker implementierten Skelett-Tracking übertragen.

Das Befehlsprotokoll spielte also eine wesentliche Rolle zur Entkopplung der Interaktions- und Visualisierungskomponente. Eine vollständige Liste aller Kommandos mit Beschreibung findet sich in Anhang A.

4.4 Viper Data Explorer

Der Viper Data Explorer (VDX) bildet das Backend des Systems und realisiert alle Funktionen, die zur Darstellung und Bearbeitung der Simulaitonsdaten nötig sind. Zentrale Komponente ist die Klasse `ViperDataExplorer`, welche wie in Abbildung 4.1 illustriert auf VTK und dem bestehenden Viper aufbaut. Sie wird beim Programmstart als erstes instanziiert und erzeugt ihrerseits Instanzen der Klassen `ViperStereoVisualizer` und `GUIManager`.

Dem Hauptprogramm können per Kommandozeile bis zu drei Parameter übergeben werden. Mit dem

Parameter *+resolution* `BreitexHöhe` kann die Größe des erzeugten `vtkRenderWindows` festgelegt werden (z.B. *+resolution 1920x1080*). Wird keine Auflösung angegeben, so wird standardmäßig ein Fenster der Größe 640×480 Pixel geöffnet. Mit `+Modus` wird bestimmt, ob und welcher Stereomodus verwendet wird. Mögliche Werte für `Modus` sind: *mono* (kein Stereo, Standard), *anaglyph*, *redblue*, *active* und *passive*. Als letztes kann mit *+udpPort* `Portnummer` oder *+vrpnIP* `IP` noch die Netzwerkverbindung zum Frontend konfiguriert werden. Mit *+udpPort* wird der Viper Data Explorer so konfiguriert, dass er auf dem durch `Portnummer` definierten Port auf UDP-Pakete wartet die Kommandos enthalten. Standardmäßig wird Port `8123` verwendet. Wird stattdessen *+vrpnIP* angegeben, so versucht der VDX eine Verbindung zu einem VRPN-Server mit der spezifizierten IP aufzubauen. Die IP `127.0.0.1` ist als Defaultwert festgelegt (d.h. Frontend und VDX laufen auf einem Rechner). Wird weder *+udpPort* noch *+vrpnIP* als Kommandzeilenparameter angegeben, wird die Standard-UDP-Konfiguration verwendet. Ein Beispielaufruf des Data Explorer mit VRPN Verbindung für die Powerwall mit passivem Stereo-Rendering in Full-HD-Auflösung sieht dann wie folgt aus (unter Linux):

```
./ViperDataExplorer +resolution 1920x1080 +passive +vrpnIP 127.0.0.1
```

Die Reihenfolge der Parameterangabe spielt keine Rolle.

Die per UDP- oder VRPN-Verbindung übertragenen Kommandos werden zuerst vom VDX empfangen, evtl. vorverarbeitet und dann an den konkreten Arbeitsmodus weitergereicht. Eine Vorverarbeitung wird dabei für `move`-Kommandos durchgeführt. Die vom Tracker gesendeten Bewegungsdaten beziehen sich auf das Weltkoordinatensystem und müssen zur Verwendung im Kamerakoordinatensystem (bspw. zur Steuerung der Kamera und zur Manipulation von Tools) noch transformiert werden. Um die korrekte (vom Nutzer erwartete) Verschiebung zu erhalten muss der Bewegungsvektor $\vec{t}_{world}$ mit der inversen View-Matrix M_{view}^{-1} multipliziert werden.

$$\vec{t}_{view} = M_{view}^{-1} * \vec{t}_{world}$$

Die Arbeitsmodi wurden nach dem State-Pattern [GHJV94] umgesetzt und werden in Abschnitt 4.4.3 detailliert beschrieben. Der konkrete Modus führt nun eine dem Kommando entsprechende Aktion aus oder leitet das Kommando an das aktive Tool (siehe Abschnitt 4.4.4) weiter. Somit entsteht eine Chain-of-Responsibility im Sinne von [GHJV94].

Wie in Abschnitt 3.2.1 dargelegt, können auf Grund von Trackingfehlern oder aus Versehen ungewollte Aktionen ausgelöst werden. Deshalb sollte es dem Nutzer möglich sein, bestimmte Aktionen rückgängig zu machen. Dazu wurde das in [GHJV94] eingeführte Command-Pattern umgesetzt. Es wandelt Prozeduraufrufe in Objekte um und ermöglicht so eine interne Speicherung in einer Aufrufhistorie sowie das einfache Rückgängigmachen von Aktionen. Abbildung 4.3 zeigt die Klassenhierarchie mit den konkreten Command-Klassen für das Erstellen und Anwenden von Tools. Diese beiden Aktionen haben sich im Laufe der Entwicklung als am fehleranfälligste herausgestellt. Hingegen können fälschliche Moduswechsel und Navigationsbefehle leicht durch den Nutzer selbst korrigiert werden. Der programmatische Ablauf bei der Ausführung und dem anschließenden Rückgängigmachen des `vdxApplyToolCommand` sieht wie folgt aus: Zuerst empfängt der Viper Data Explorer das `apply`-Kommando und leitet es an den aktuellen Modus weiter. Befindet sich der VDX gerade im Manipulation-Mode, so wird ein

Objekt der Klasse `vdxApplyToolCommand` erzeugt. Dieses bekommt per Konstruktor die Instanz des VDX, die aktuelle Visualisierung und das aktive Tool übergeben. Der Manipulationsmodus ruft dann die `ExecuteCommand`-Methode des VDX mit dem erzeugten Command-Objekt auf.

```
/// in ManipulationMode::HandleCommand
if ( strstr( cmdString, "apply" ) )
{
        m_context->ExecuteCommand( new vdxApplyToolCommand( m_context,
                                m_context->GetVisualizer()->getActiveVisualization(),
                                m_activeTool ) );
}

/// in ViperDataExplorer
void ViperDataExplorer::ExecuteCommand( vdxCommand * command )
{
        command->Execute();

        vdxUndoableCommand * undoCmd = dynamic_cast< vdxUndoableCommand* >( command );

        if ( undoCmd != NULL )
        {
                m_commandHistory.push( undoCmd );
        }
}
```

Im VDX selbst wird dann die `execute`-Methode des Command-Objektes aufgerufen und dadurch die gewünschte oder ungewünschte Aktion ausgeführt. Handelt es sich beim übergebenen Command um ein Objekt vom Typ `vdxUndoableCommand`, so wird es intern auf einem Stack abgelegt (engl. *push*). Soll nun die eben durchgeführte Aktion wieder rückgängig gemacht werden, so wird das oberste Element vom Command-Stack heruntergenommen (engl. *pop*) und dessen `undo`-Methode aufgerufen.

```
void ViperDataExplorer::Undo()
{
        if ( m_commandHistory.size() > 0 )
        {
                vdxUndoableCommand * cmd = m_commandHistory.top();
                cmd->Undo();
                m_commandHistory.pop();
        }
}
```

Analog wird das Erstellen und Entfernen von Tools mit `vdxCreateToolCommand` durchgeführt.

Schließt der Nutzer am Ende seiner Arbeit den VDX, so wird das Resultat seiner Sitzung (engl. *session*) in einer VQL-Datei gespeichert. Dazu wird die `Save()`-Methode des VDX aufgerufen. Es wird eine Datei angelegt, deren Name sich aus ”session__” und der aktuellen Systemzeit sowie der Endung ”.vql” zusammensetzt. Der Inhalt der Datei (also das VQL-Programm) wird durch die Klasse `viper::Configurator` generiert. Wird ein Tool angewendet, so wird dessen `AbstractTool::applyToConfigurator(viper::Configurator * config)`-Methode aufgerufen. Diese bewirkt je nach Implementierung im konkreten Tool einen Aufruf der `setFilter*`-Methode(n) auf dem übergebenen Configurator-Objekt. Folgender Codeausschnitt zeigt dies für das Halfspace-Select-Cut Tool:

```
void HalfspaceSelectCutTool::applyToConfigurator( viper::Configurator * config )
{
        // get plane position on the axis the tool is aligned to
        double axisPos;
```

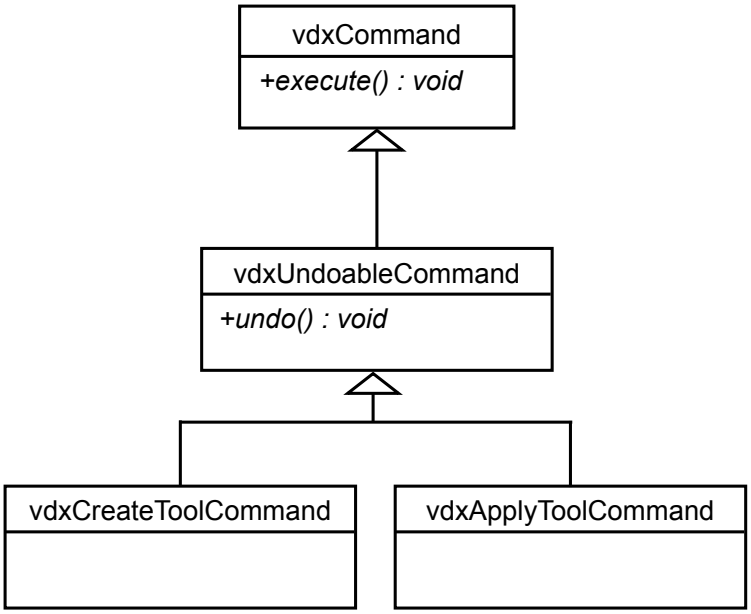


Abbildung 4.3: Klassendiagramm: Implementierung Command-Pattern

```
        std::string axis;

        // ...

        if ( m_alignment == PositiveX || m_alignment == PositiveY || m_alignment == PositiveZ )
        {
                config->setFilterMin( axis, axisPos );
        }
        else
        {
                config->setFilterMax( axis, axisPos );
        }
}
```

Zuletzt wird die `writeVqlProgramm`-Methode mit dem zuvor angelegten Filehandle als Parameter aufgerufen.

4.4.1 Erweiterung der Visualisierung

Zur Realisierung der interaktiven Bearbeitungsfunktionen und um die Visualisierung so effektiv und expressiv wie möglich darzustellen, wurde das bestehende VIPER-Projekt in einigen Punkten verändert bzw. erweitert.

4.4.1.1 ViperStereoVisualizer

Die Klasse `viper::Visualizer` ist die zentrale Klasse um aus VQL-Programmen Visualisierungen zu generieren. Ihre Funktionalität wurde weitestgehend in die neu-implementierte Klasse `ViperStereoVisualizer` übernommen. Diese ersetzt somit die Originalklasse aus VIPER. Wie ihre Name schon verrät, unterstützt sie stereoskopisches Rendering in mehreren Varianten. Desweiteren setzt sie ein

erweitertes Beleuchtungskonzept um (siehe Abschnitt 4.4.1.2).

Implementiert wurden die Rendermodi Anaglyph-, Rot-Blau-, aktives Shutter- und passives Side-by-Side-Stereo sowie nicht-stereoskopisches Mono-Rendering wie zuvor. Die drei erstgenannten Stereomodi werden direkt von VTK unterstützt. Initialisiert werden können sie durch die entsprechenden Aufrufe von `vtkRenderWindow::SetStereoModeToAnaglyph()`, `vtkRenderWindow::SetStereoModeToRedBlue()` bzw. `vtkRenderWindow::SetStereoModeToCrystalEyes()`. Zusätzlich muss das Renderfenster noch als stereofähig deklariert werden. Die geschieht über den Aufruf von `vtkRenderWindow::StereoCapableWindowOn()` und `vtkRenderWindow::StereoOn()`.

Zur Umsetzung des passiven Stereo-Rendering wird die Side-by-Side Technik eingesetzt. Dazu werden die Stereoteilbilder von zwei `vtkRenderer`n nebeneinander in ein Renderwindow doppelter Breite gezeichnet. Folgender Quelltextausschnitt zeigt, wie der ViperStereoVisualizer dafür konfiguriert wird:

```
void ViperStereoVisualizer::configPassive()
{
        m_parallax = 0.001;

        m_renderWindow->SetSize( 2 * m_renderWindowWidth, m_renderWindowHeight );
        m_renderWindow->SetPosition( 0, 0 );

        m_leftRenderer->SetViewport( 0, 0, 0.5, 1 );
        m_renderWindow->AddRenderer( m_leftRenderer );

        m_rightRenderer->SetViewport( 0.5, 0, 1, 1 );
        m_renderWindow->AddRenderer( m_rightRenderer );
}
```

Zuerst wird der Augabstand festgelegt. Er wird später als Offsetwert für die virtuellen Augpunkte verwendet. Danach wird das Renderwindow auf die doppelte Breite skaliert und am linken oberen Bildschirmrand ausgerichtet. Anschließend werden durch die Darstellungsfelder (engl. *Viewport*) die Bereiche festgelegt, in welche die Renderer ihr Bild zeichnen. Der in Abschnitt 3.4 beschriebene Beameraufbau sorgt nun dafür, dass beide Teilbilder auf der Leinwand übereinander liegen und bei Betrachtung mit einer Polbrille ein Stereobild entsteht.

Ein Problem besteht nun darin, dass der standardmäßig zur Nutzerinteraktion eingesetzte `vtkRenderWindowInteractor` nur Events auf einem Renderer auslöst. Um die synchrone Steuerung der beiden virtuelle Kameras der Renderer zu gewährleisten implementiert der ViperStereoVisualizer ein eigenes Kamera-Interface. Dieses beinhaltet Methoden zum Zoomen, Verschieben und Rotieren einer dritten, internen Kamerainstanz, welche die Beobachterposition repräsentiert. Nach jedem Aufruf einer dieser Methoden wird die `updateCameras`-Funktion aufgerufen:

```
Vector3 xaxis, yaxis, zaxis;

Vector3 eyePos, focalPoint;
m_camera->GetPosition( eyePos.X, eyePos.Y, eyePos.Z );
m_camera->GetFocalPoint( focalPoint.X, focalPoint.Y, focalPoint.Z );

// Compute z axis (negative axis along which we are looking)
zaxis = eyePos - focalPoint;

// Normalize z axis
```

```
zaxis.Normalize();

// Now Compute the x axis (right vector) from world up and z axis
Vector3 up( 0.0, 1.0, 0.0 );

xaxis = Vector3::CrossProduct( up, zaxis );

// Normalize x axis
xaxis.Normalize();

// compute correct up (y) axis
yaxis = Vector3::CrossProduct( zaxis, xaxis );
yaxis.Normalize();

double halfParallax = m_parallax * 0.5;

Vector3 camLeftFP, camRightFP, camLeftPos, camRightPos;

camLeftPos = eyePos - xaxis * halfParallax;
camRightPos = eyePos + xaxis * halfParallax;
camLeftFP = focalPoint - xaxis * halfParallax;
camRightFP = focalPoint + xaxis * halfParallax;

m_leftRenderer->GetActiveCamera()->SetPosition( camLeftPos.X, camLeftPos.Y, camLeftPos.Z );
m_leftRenderer->GetActiveCamera()->SetFocalPoint( camLeftFP.X, camLeftFP.Y, camLeftFP.Z );

m_rightRenderer->GetActiveCamera()->SetPosition( camRightPos.X, camRightPos.Y, camRightPos.Z );
m_rightRenderer->GetActiveCamera()->SetFocalPoint( camRightFP.X, camRightFP.Y, camRightFP.Z );

m_leftRenderer->GetActiveCamera()->SetViewShear( -halfParallax, 0, 0.2 );
m_rightRenderer->GetActiveCamera()->SetViewShear( halfParallax, 0, 0.2 );
```

Sie berechnet die lokalen Koordinatenachsen der internen Kamera und setzt anschließend Position und Fokuspunkt der beiden Augkameras, versehen mit einem dem halben Augabstand entsprechenden Offset nach links und rechts. Hier werden auch gleich die Projekionsmatizen der Kameras mit der in Abschnitt 3.2.1 hergeleiteten Scherung multipliziert. Dazu bietet VTK die Methode `SetViewShear`.

4.4.1.2 Beleuchtung mit VTK

Wie bereits im vorigen Abschnitt erwähnt, implementiert der ViperStereoVisualizer ein verbessertes Beleuchtungskonzept. Die zu Grunde liegende Motivation ist, aus der Visualisierung so viel Information wie möglich zu extrahieren. Dazu kann eine gut arrangierte Beleuchtung einen wichtigen Beitrag leisten, indem sie bspw. Konturen und Oberflächeneigenschaften besser hervorhebt. Das Problem besteht nun darin, dass weder der Entwickler des Visualisierungssystems noch dessen Nutzer die entstehende Szene bzw. Visualisierung kennen kann. Damit ist eine spezialisierte Lichtpositionierung ausgeschlossen. Ein weiteres Problem sind das meist fehlende Expertenwissen sowie keine Restriktionen. Anwender eines Visualisierungssystems verfügen meist nicht über das nötige Know-How (wie etwa Fotografen) zur Ausleuchtung von Szenerien. Ebenso sind sie in der virtuellen Welt weit weniger eingeschränkt als in der realen Welt (z.B. auf der Theaterbühne). So kann es den Nutzer viel Zeit kosten, eine effektive Beleuchtung für seine Visualisierung zu finden. Die einfachste Lösung ist, das Standardlicht von VTK zu verwenden. Es handelt sich dabei um ein einzelnes Headlight, welches der Kamerabewegung folgt

und immer in Blickrichtung strahlt. Problematisch ist jedoch, dass ein solches Licht keine Information über Orientierung und Richtung wiedergibt. Es kann Spots bzw. Highlights auf der Oberfläche erzeugen und so zu Fehlinterpretationen führen. Ein klarer Verstoß gegen das Expressivitätskriterium. Eine andere Möglichkeit besteht darin, mehrere Lichter zu benutzen. Diese effektiv zu positionieren und auszurichten ist, wie gesagt, nicht trivial. Außerdem geht jedes weitere Licht zu Lasten der Performance. Ein undurchdachtes Beleuchtungskonzept mit mehreren Lichtern ist also weder effektiv noch angemessen.

Deshalb wurde in dieser Arbeit ein fertiges und einfach parametrisierbares Licht-Arrangement verwendet. Das `vtkLightKit`[HM03] bietet eine solche Lösung mit insgesamt fünf Lichtern. Es besteht aus einem Key-, Fill-, Head- und zwei Backlights.

Das Keylight ist leicht seitlich und deutlich oberhalb des betrachteten Objektes platziert. Es simuliert den Lichteinfall durch Sonne oder Deckenbeleuchtung und vermittelt so einen Orientierungs- bzw. Richtungssinn.

Das Filllight ist fast genau entgegengesetzt ausgerichtet. Es hat nur eine halb so große Leuchtkraft wie das Keylight und dient dazu, übermäßig starke Hell-Dunkel-Kontraste zu vermeiden. Simuliert wird damit das von anderen Objekten, Boden und Himmel reflektierte Licht.

Das Headlight ist dem Licht der Standardbeleuchtung sehr ähnlich und bewegt sich stets mit der Kamera.

Die beiden Backlights dienen dem Hervorheben von Kanten nahe der Silhouette des Beobachtungsobjekts. Sie sind symmetrisch hinter dem Objekt angeordnet und besitzen etwa ein Drittel der Leuchtkraft des Keylights.

Das LightKit bietet eine sog. Wärmeskala zur Einfärbung der Lichter, welche von dunkelblau über weiß bis rot reicht. Da viele Visualisierungen selbst Gebrauch von Farbskalen machen um bestimmte Merkmale hervorzuheben, sollte als Lichtfarbe immer weiß gewählt werden. Andernfalls kann eine unvorteilhafte Farbmischung die Visualisierung verfälschen [Gum02].

Der Effekt, den das verbesserte Beleuchtungskonzept hat, ist in Abbildung 4.4 anhand eines Partikelplots zu sehen. Die blauen Partikel, die hier offensichtlich sehr signifikant sind, verschwinden in der linken Abbildung fast vollständig. Mit dem neuen Beleuchtungssystem sind sie wesentlich besser erkennbar.

Ein weiteres Beispiel liefert der Vergleich von einfach, verbessert und mit angepassten Materialeigenschaften gerendertem Skalarfeld (siehe Abb. 4.5). Im linken Bild ist die Standardbeleutung zu sehen. Konturen sind hier schwer nach zu verfolgen und starke Kontraste lassen das Bild eher nach einem ungeordneten Haufen Polygone, statt einem elektro-magnetischen Feld, aussehen. Im mittleren Bild erscheint die Darstellung schon wesentlich homogener. Im rechten Bild wurden die Materialeigenschaften des `vtkActor` mit den Methoden von `vtkProperty` wie folgt angepasst:

```
m_actor->GetProperty()->SetSpecularColor( 1.0, 1.0, 1.0 );
m_actor->GetProperty()->SetSpecularPower( 76.0 );
m_actor->GetProperty()->SetAmbient( 0.15 );
m_actor->GetProperty()->SetDiffuse( 0.35 );
m_actor->GetProperty()->SetSpecular( 0.4 );
```

So hebt sich die gesamte Darstellung noch besser vom schwarzen Hintergrund ab und Konturen bzw. Kanten können besser zugeordnet und mit dem Auge verfolgt werden.

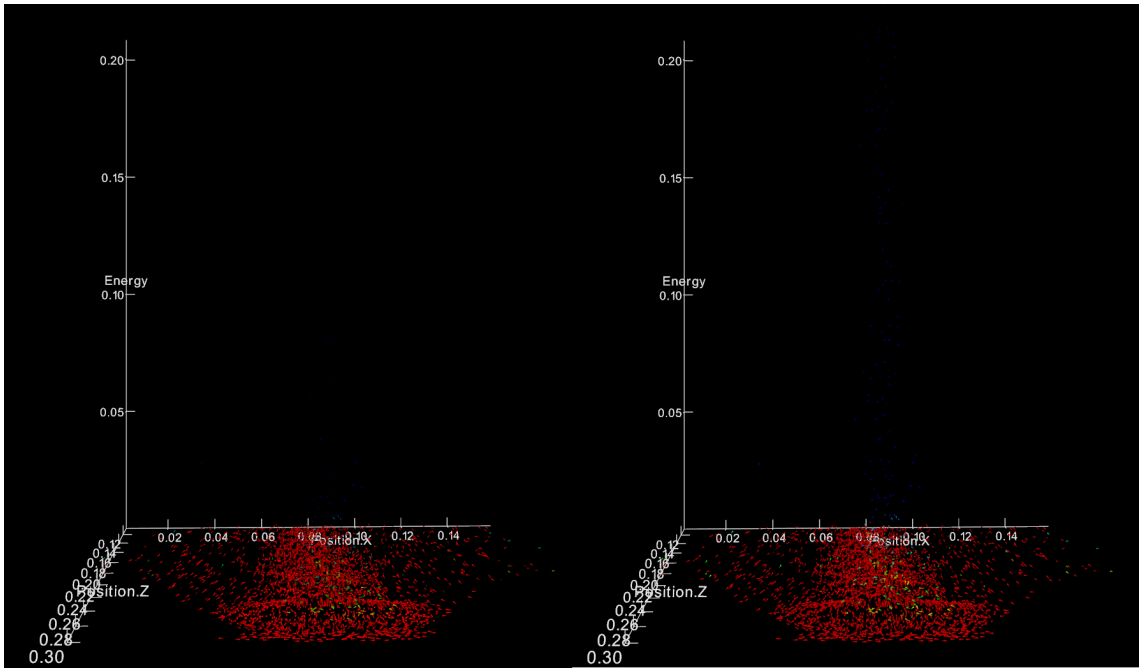


Abbildung 4.4: Partikelplot ohne und mit verbesserter Beleuchtung (blaue Partikel sichtbar)

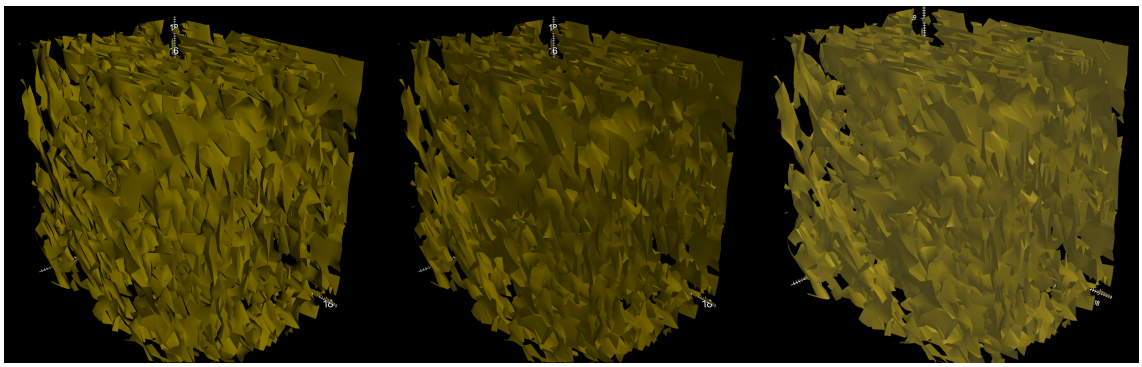


Abbildung 4.5: Isoflächen mit Standard- (links), verbesserter (Mitte) und spekularer Beleuchtung (rechts)

4.4.1.3 VIPER-Visualisierungspipeline

Um die in Abschnitt 4.4.4 beschriebenen Tools zu implementieren, mussten die Klassen `viper::Visualization` und `viper::AbstractVisPipe` erweitert werden. Zur visuellen Darstellung der Tools ist es nötig, die durch die Visualisierungspipeline generierten geometrischen Daten vor dem Renderingschritt noch weiter zu filtern. Dazu wurde der `AbstractVisPipe`-Klasse die Methode `GetOutputPort` hinzugefügt. Sie liefert die zum Rendern gemappten Daten des letzten (VTK-)Filters zurück, bevor diese mit einem vtkActor dargestellt werden. Dies ermöglicht den Tools die Visualisierungspipeline um Filter zu ihrer eigenen visuellen Repräsentation zu erweitern und schließlich das Gesamtergebnis zu rendern. Verdeutlicht wird dies in Abbildung 4.6.

Nachdem ein Tool erstellt und damit gearbeitet wurde kann es entweder wieder entfernt oder angewendet werden. Wird es entfernt, so wird die Visualisierung wieder ohne das Tool gerendet. Wird es angewendet, so wird dessen visuelle Repräsentation ebenfalls wieder aus der Pipeline gelöscht. Je nach Implementie-

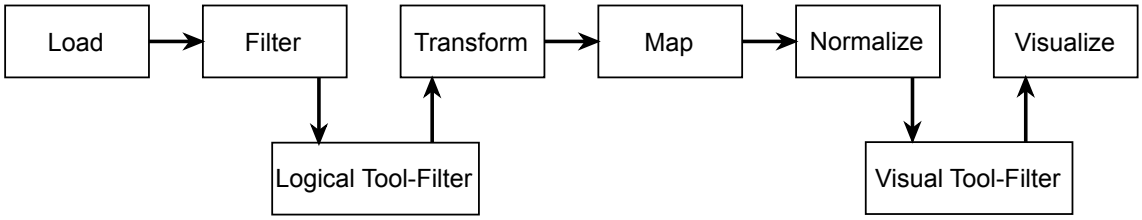


Abbildung 4.6: Visualisierungspipeline mit Eingriffspunkten der Tools

rung des Tools werden nun aber die Erweiterungsmethoden der Klasse Visualization eingesetzt, um das Tool als Datenfilter in die Pipeline einzubauen. Dazu erstellt das Tool intern eine Instanz von `VtkDataFilter` oder `vtkImageAlgorithm`. Diese werden dann durch einen Aufruf von `pushPolyDataFilter` bzw. `pushImageDataFilter` auf der aktuellen Visualisierung in den Filterschritt der VIPER-Pipeline eingebaut. Damit wird die für die nachfolgenden Schritte zu verarbeitende Menge an Daten reduziert. Werden nun weitere Tools erstellt und angewendet, so werden diese nach den zuvor angewandten Tools auf den internen Filter-Stack gelegt. Die Implementierung der einzelnen Tools wird im jeweiligen Unterabschnitt genauer erläutert.

4.4.2 User Interface

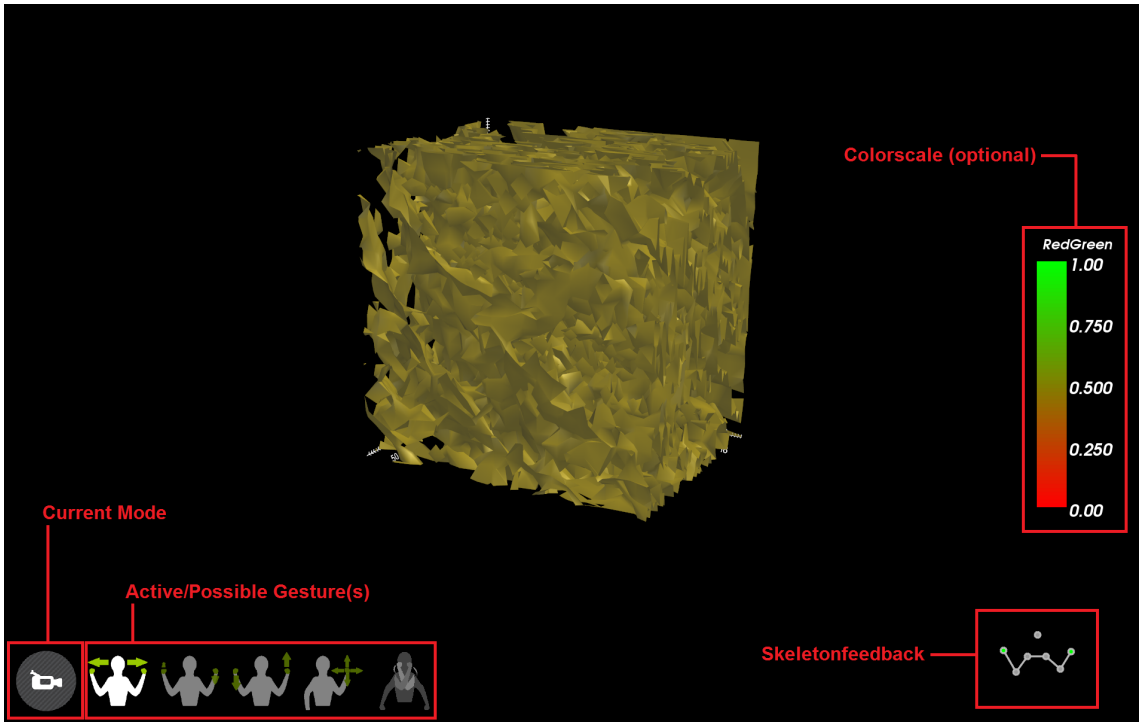


Abbildung 4.7: Die Benutzeroberfläche im Navigationsmodus

Bei der Gestaltung der Benutzeroberfläche wurde - wie in Kapitel 3 diskutiert - auf Minimalität sowie ausreichend Hilfe und Feedback für den Nutzer Wert gelegt. Abbildung 4.7 zeigt die GUI. In der linken unteren Ecke wird immer der aktuelle Modus angezeigt (hier Navigaitonsmodus). Direkt daneben werden die möglichen Gesten als Icons schematisch dargestellt. Die gerade vom System erkannte Geste wird

dabei hervorgehoben. In der unteren rechten Bildschirmecke gibt ein Mini-Skelett Aufschluss über den Skelett-Tracking-Status. Die vom Tracker erkannten Joint-Bewegungen werden hier durch kleine Kreise für Hände, Ellbogen, Schultern und Kopf angezeigt. Um besser nachvollziehen zu können, welcher Kreis welches Körperteil symbolisiert, wurden sie mit entsprechenden Linien für Arme und Schlüsselbein verbunden.

Abbildung 4.8: Erste Version der Gestenicons

Bei der Gestaltung der Icons wurde darauf geachtet, dass sich die Gesten so ausführen lassen wie abgebildet. Eine frühere Version der Icons ist in Abbildung 4.8 zu sehen. Die Nutzer führten die Gesten wie abgebildet mit hoch erhobenen Arme aus, was weder notwendig noch ergonomisch und auf Dauer ermüdend ist. Deshalb wurden die Icons neu gestaltet und zudem in Einklang mit dem Mini-Skelett gebracht. Dies äußert sich vorallem in der Farbgebung. Um eine Interaktion anzustoßen müssen die Hände geschlossen sein. Dies wird der Visualisierung durch die Befehle *Hand_Left_Closed* und *Hand_Right_Closed* bzw. *Hand_Left_Opened* und *Hand_Right_Opened* vom Tracker mitgeteilt. Daraufhin wird der Kreis für die entsprechende Hand grün statt grau gefüllt. Dadurch wird eine bessere Konsistenz im User Interface erreicht (siehe Abb. 4.9).

Bei der Gestaltung des Interface wurde so weit als möglich auf Text verzichtet. Erstens eignen sich bildliche Darstellungen wesentlich besser zur Erklärung von Gesten als Textbeschreibungen. Zweitens verleitet Text bei jedem hinsehen zum Lesen. Er prägt sich nicht so leicht ein und kann nicht so schnell als bekannt wahrgenommen werden wie ein einfaches Icon.

Am rechten Bildrand wird die verwendete (optionale) Farbskala (engl. *color scale*) vom `ViperStereoVisualizer` mit Hilfe des `vtkScalarBarActor`s angezeigt.

Implementiert wurde die GUI unter Einsatz der 2D Rendering API von VTK. Deren wichtigste Klassen sind `vtkContext2D`, `vtkContextActor`, `vtkContextScene` und `vtkAbstractContextItem`. Mit ihrer Hilfe wurde die `GUIManager`-Klasse implementiert. Sie dient als Verwaltungsinstanz für das User Interface und als Container aller GUI-Elemente. Intern enthält sie eine Referenz auf einen `vtkContextActor`. Dieser wird bei der Initialisierung dem Renderer (bzw. den Renderern) im Visualizer hinzugefügt. Vom Context-Actor kann mit `GetScene` eine Instanz der `vtkContextScene` bezogen werden. Dieser Szene wird der von `vtkAbstractContextItem` abgeleitete GUI-Manager hinzugefügt. Nun kann durch Überladen der `Paint`-Methode das 2D Rendering mit Klassen wie `vtkPen` und `vtkBrush` genutzt werden.

4.4.3 Arbeitsmodi

Zur Bewältigung der in Abschnitt 3.3 diskutierten Untersuchungsziele der visuellen Datenanalyse wurde das Konzept der Bearbeitungsmodi eingeführt. Es entlastet den Nutzer, da dieser nicht mit einer zu großen Zahl an möglichen Eingabegesten überfordert wird. Desweiteren ermöglicht es die Verwendung

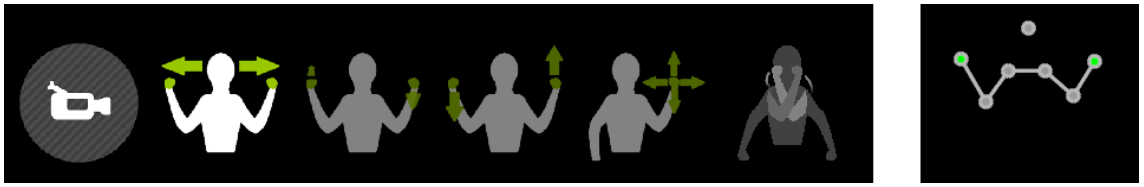


Abbildung 4.9: Korrespondenz zwischen Gestenicons und Skelett-Feedback

gleicher Gesten in einem anderen Kontext zur Bewältigung ähnlicher Aufgaben (siehe Abs. 3.2.2). Nicht-zuletzt vereinfacht es das Tracking und die Gestenerkennung, da für jede Aufgabe bzw. jeden Modus ein spezialisierter Zustand im Tracker initialisiert werden kann. Dieser verwendet dann die bestmöglichen Tracking-Strategien für die möglichen Nutzereingaben. Die Reduktion der Gesten pro Modus verringert auch die Fehleranfälligkeit, da sich die einzelnen Gesten deutlicher unterscheiden und so vom Tracker besser klassifiziert werden können.

Die Modi wurden dem State-Pattern[GHJV94] entsprechend implementiert. Abbildung 4.10 zeigt die einzelnen Modi und ihr Zusammenwirken in einem Zustandsdiagramm.

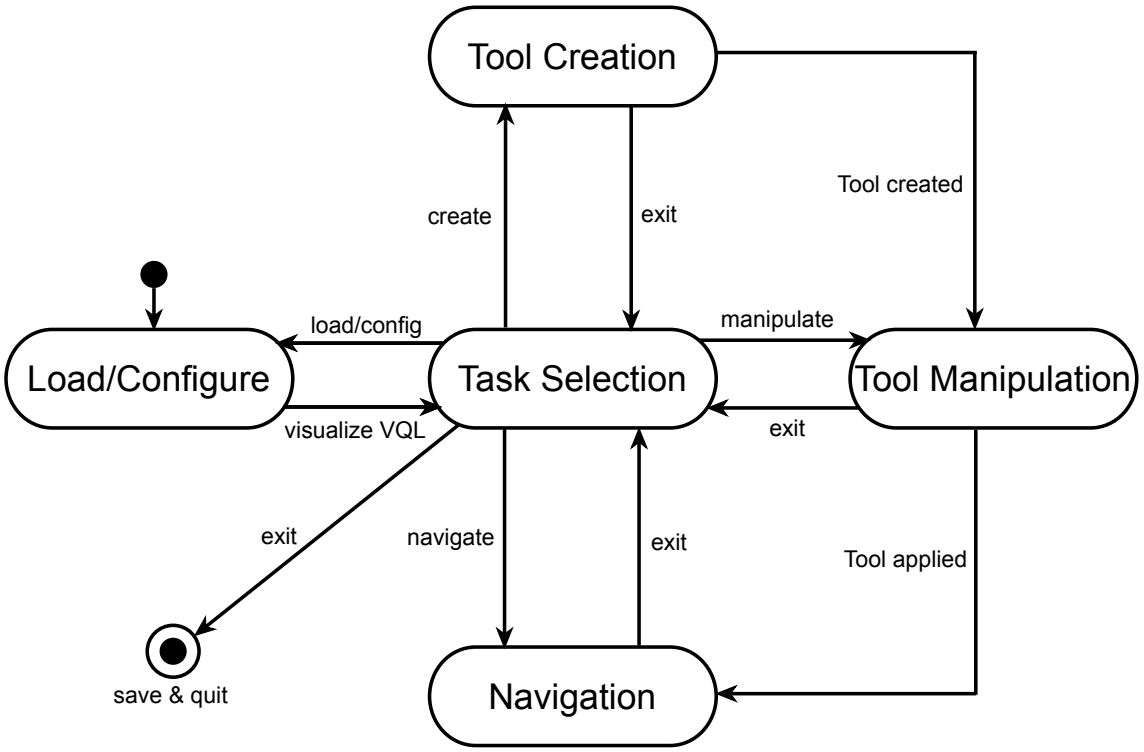


Abbildung 4.10: Zustandsdiagramm der Arbeitsmodi

4.4.3.1 Load/Configure Mode

Beim Start des VDX gelangt man zunächst in den Load/Configure-Modus. Hier hat der Nutzer die Option ein vorhandenes VQL-Programm zu laden oder mit Hilfe eines Konfigurationsdialogs ein neues zu erstellen. Wählt der Nutzer die erste Option, so werden ihm vier VQL-Programme zur Auswahl angeboten. Ein Partikelplot und ein Energie-Positions-Diagramm eines Partikeldatensatzes sowie eine Isoflächendarstellung und ein Stromliniendiagramm eines Felddatensatzes.

Entscheidet sich der Nutzer für die Neukonfiguration eines VQL-Programms, so erfolgt dies in drei Schritten: Zuerst muss ein Datensatz selektiert werden (i.d.R. Partikel- oder Felddaten). Danach wird eine Liste aller Datenfelder neben einer Liste der Visualisierungsattribute angezeigt. Durch *Swipe*, *Push*, *Move* und *Spread* Gesten ordnet der Anwender die Datenfelder den gewünschten Visualisierungsattributen zu und legt die Wertebereiche fest. Ein Datenfeld kann dabei mehreren Visualisierungsattributen zugeordnet werden, allerdings kann jedem Visualisierungsattribut nur ein Datenfeld zugewiesen werden (1:n-Abbildung von Datenfeldern zu Visualisierungsattributen). Im letzten Schritt werden dem Nutzer mögliche Visualisierungen für seine Konfiguration angezeigt, aus welchen er eine auswählt. Daraufhin werden die Daten geladen und der Data Explorer wechselt in den Task Selection Mode.

Zur Umsetzung der GUI wurden die Klassen `vdxButton`, `vdxIntervalSlider` sowie `vdxVerticalList` und `vdxHorizontalList` implementiert. Die Anzeige der Datenfelder und Visualisierungsattribute erfolgt mit Hilfe der Button-Klasse, die für jedes Datenfeld instanziiert und entsprechend beschriftet wird. Die IntervalSlider dienen zur Bestimmung der zu ladenden Wertebereiche.

Durch *Swipe* und *Push* kann der Nutzer durch die GUI-Elemente navigieren. Ein typischer Konfigurationsablauf sieht wie folgt aus: Der Nutzer navigiert per *Swipe* zu einem Datenfeld, welches er als Merkmal in die Visualisierung einbeziehen möchte. Hat er den Fokus darauf gesetzt, führt er eine *Push*-Geste aus um es zu selektieren. Daraufhin springt der Fokus automatisch auf den sich rechts vom Datenfeld-Button befindlichen IntervalSlider. Hier passt der Nutzer per *Move* und *Spread* den Wertebereich wie gewünscht an. Ein weiteres *Push* lässt den Fokus in die Liste der Visualisierungsattribute springen. Per *SwipeUp* und *SwipeDown* kann nun der Fokus auf das gewünschte Visualisierungsattribut gelegt werden. Durch ein weiteres *Push* wird das Merkmal dem Attribut zugewiesen, dargestellt durch eine Verbindungslinie. Abbildung 4.11 zeigt den Zuweisungsschritt. Die vorgenommene Zuordnung entspricht folgendem VQL-Programm:

```
VISUALIZE

MAP
        Energy TO Axis.Y,
        Position.X TO Axis.X,
        Position.Z TO Axis.Z,
        Energy TO Scalar
WITH
        Time in range 0 56
WHERE
        ID > 100 and ID < 1.4e7
        and Position.X > 6.5 and Position.X < 35.14;
```

Im letzten Schritt wird eine Visualisierung ausgewählt und dadurch das `VISUALIZE`-Statement ausge-

füllt.

Die einzelnen Ladeschritte wurden ähnlich den Arbeitsmodi mit dem State-Pattern implementiert. Sie bilden also eine Art Sub-State-Hierarchie für die `LoadVisMode`-Klasse.

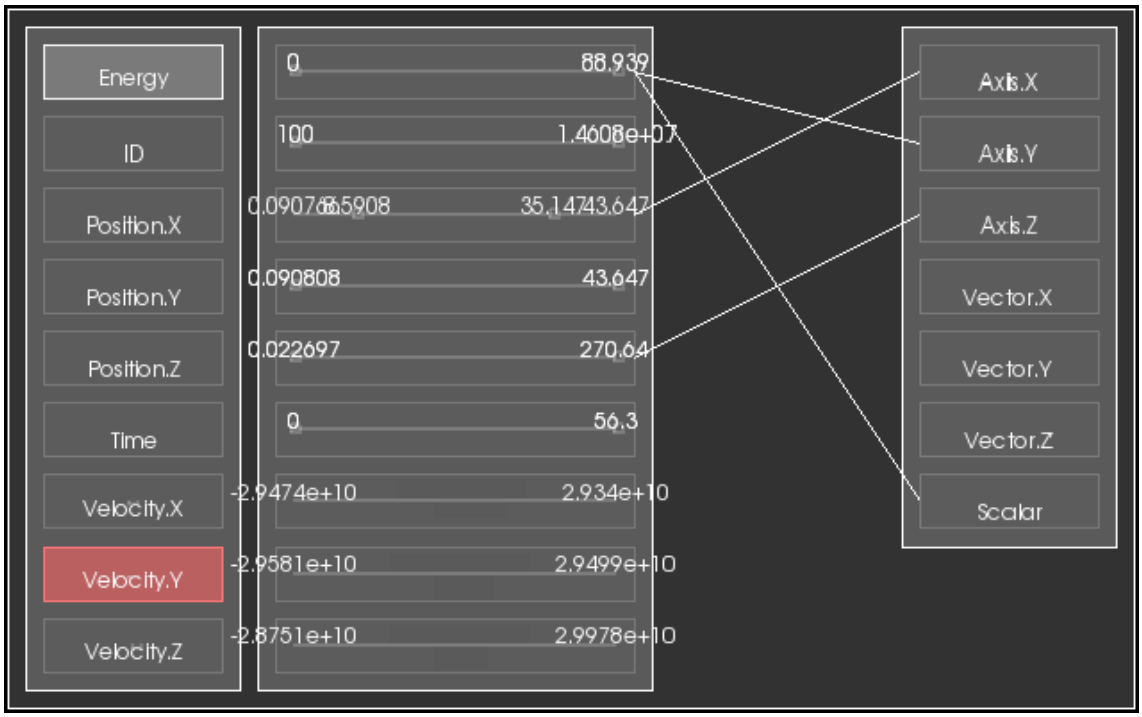


Abbildung 4.11: Parameterzuweisung und Einstellung der Datenbereiche mit dem Konfigurationsdialog

4.4.3.2 Task Selection Mode

Der Task Selection Mode fungiert als zentraler Auswahlmodus. Hier wird über die einhändigen symbolischen Gesten einer der anderen Modi angewählt. Wird ein 'C' gezeichnet wechselt der VDX in den Tool Creation Modus (im Zustandsdiagramm Übergang mit '*create*' bezeichnet). Ein 'N' führt in den Navigationsmodus ('*navigate*'-Übergang). Gibt es ein aktives Tool, so führt eine 'M'-Geste in den Manipulation Mode. In den Load/Configuration Modus gelangt man durch eine 'L'-Geste. Wird die Exit-Geste ausgeführt, so speichert der VDX das aktuelle Bearbeitungsresultat in eine VQL-Datei und schließt sich.

4.4.3.3 Navigation Mode

Der Navigation Mode dient zur freien Betrachtung der Visualisierung. Es erfolgt eine direkte Manipulation der Beobachterposition (ohne 3D Cursor o.ä.). Druch einhändiges Greifen kann der Nutzer die Visualisierung im Raum verschieben. Schließt der Nutzer beide Hände, so kann er durch entgegengesetztes Auf- und Abbewegen der Hände eine Rotation um die X-Achse ausführen. Durch Vor- und Zurückbewegen erfolgt eine Rotation um die Y-Achse. Eine Rotation um die Z-Achse wurde bewusst ausgeschlossen, da dies oft zum Orientierungsverlust führt. Bewegt der Nutzer die geschlossenen Hände voneinander weg,

so fährt die Kamera näher an ihren Fokuspunkt heran, zoomt also. Führt er die Hände zusammen zoomt die Kamera heraus. Von den fünf Freiheitsgraden kann immer nur einer gleichzeitig verändert werden. Um einen anderen zu manipulieren muss der Nutzer die Hände zuerst öffnen und kann durch erneutes Schließen die nächste Aktion ausführen. Per Exit-Geste gelangt er zurück zum Task Selection Mode.

4.4.3.4 Tool Creation Mode

Im Tool Creation Modus kann der Nutzer verschiedene Werkzeuge zur Auswahl von Datenbereichen und Auswertung von Merkmalen erstellen. Auf die verschiedenen Werkzeuge wird in Abschnitt 4.4.4 genauer eingegangen.

Als Eingabe werden einhändig gezeichnete Gesten erkannt. Um beispielsweise eine Auswahlbox zu erstellen führt der Nutzer eine einhändige Zeichengeste in Form eines Quadrats aus. Daraufhin wird im Mittelpunkt der Visualisierung ein würfelförmiger Bereich farbig dargestellt, während alles außerhalb der Auswahlbox in transparentem Grau gerendert wird. Gleiches gilt für eine Auswahlebene. Sie wird erstellt, indem der Nutzer eine möglicherweise rotierte und gespiegelte 'L'-förmige Geste ausführt. Dabei definiert die Richtung der zuerst gezeichneten Linie des Ls die Achse, welche die entstehende Ebene orthogonal schneiden soll. Die als zweites gezeichnete Linie gibt die Richtung der Normalen an. Somit können Auswahlebenen für alle sechs Halbachsen erstellt werden (also auf X-, Y- und Z-Achse mit Orientierung der Normalen in positiver oder negativer Richtung).

Ein Tool wird nach seiner Erstellung zunächst am Ende der Visualisierungspipeline eingefügt, ist also rein visuell (siehe Abb. 4.6). Die `addToVisualization`-Methode wird dazu aufgerufen.

4.4.3.5 Tool Manipulation Mode

Mit den im Creation Mode erzeugten Tools kann nun im Manipulation Mode gearbeitet werden. Auswahlebenen können auf der Achse in Richtung ihrer Normalen verschoben werden. Auswahlboxen dagegen können in allen drei Raumachsen ausgerichtet und zusätzlich noch skaliert werden. Hierzu verwendet der Nutzer die gleichen Gesten wie schon im Navigationsmodus zur Kamerasteuerung. Die in Abschnitt 3.2.2 angesprochene Konsistenz zwischen ähnlichen Aufgaben und gleichen Gesten wird so umgesetzt.

Außerdem können im Manipulationsmodus Werkzeuge angewandt werden. Das bedeutet beispielsweise für das Auswahlbox-Tool, dass dessen visuelle Repräsentation vom Ende der Pipeline entfernt und ein Datenfilter wie in Abbildung 4.6 gezeigt in die Pipeline eingebaut wird. Dazu werden intern die `removeFromVisualization` und `applyToVisualization`-Methoden aufgerufen.

4.4.4 Tools

Die bereits im vorigen Abschnitt erwähnten Tools lassen sich in zwei Kategorien unterteilen. Einerseits die interaktiven Datenfilter. Sie dienen dazu, die zu bearbeitende Datenmenge durch visuelle Parametrisierung (Selektion) und anschließendes Anwenden (Filtern) zu reduzieren. Diese Art von Tools ändert al-

so dynamisch die Visualisierungspipeline. Anderseits die Werkzeuge zur Auswertung. Mit ihnen werden Merkmalsausprägungen untersucht, indem Schnitte durch Felddaten gemacht sowie Beträge bestimmter Größen gemessen werden.

4.4.4.1 Select-and-Cut Tool

Das Select-and-Cut Tool schneidet Teilbereiche der Daten in einer oder drei Dimensionen aus und erhält dabei die Dimensionalität der Daten. Es gibt dabei zwei verschiedene geometrische Formen, mit denen man Daten auswählen kann. Eine einfache Ebene und eine Box. Die Auswahlebene kann für alle sechs Halbachsen erstellt werden. Dazu wird wie in Abschnitt 4.4.3.4 beschrieben eine L-förmige Geste erkannt. Die Ebene ist an der X-, Y- oder Z-Achse ausgerichtet. Die Richtung ihrer Normalen (positive oder negativ entlang der Achse) gibt an was ”vor” und was ”hinter” ihr liegt respektive was ”ausgewählt” und was ”weggeschnitten” wird.

Die Auswahlbox ist eine an den Raumachsen ausgerichteter Quader. Er kann beliebig skaliert und verschoben, jedoch nicht rotiert, werden.

Zur visuellen Darstellung des Tools wurde mit den Klassen `vtkClipPolyData`, `vtkPolyDataMapper` und `vtkActor` die Visualisierungspipeline erweitert. Abbildung 4.12 zeigt die entstehende Pipeline. Dabei wird die Ebene bzw. Box als implizite Funktion an die Instanz von `vtkClipPolyData` übergeben. Diese liefert dann an zwei Output-Ports die geclippten und nicht-geclippten Polydaten, welche dann jeweils mit Hilfe eines PolyDataMappers und Actors gerendet werden. Der Actor der die geclippten (also weggeschnittenen) Daten verarbeitet wird dabei in transparentem Grau dargestellt.

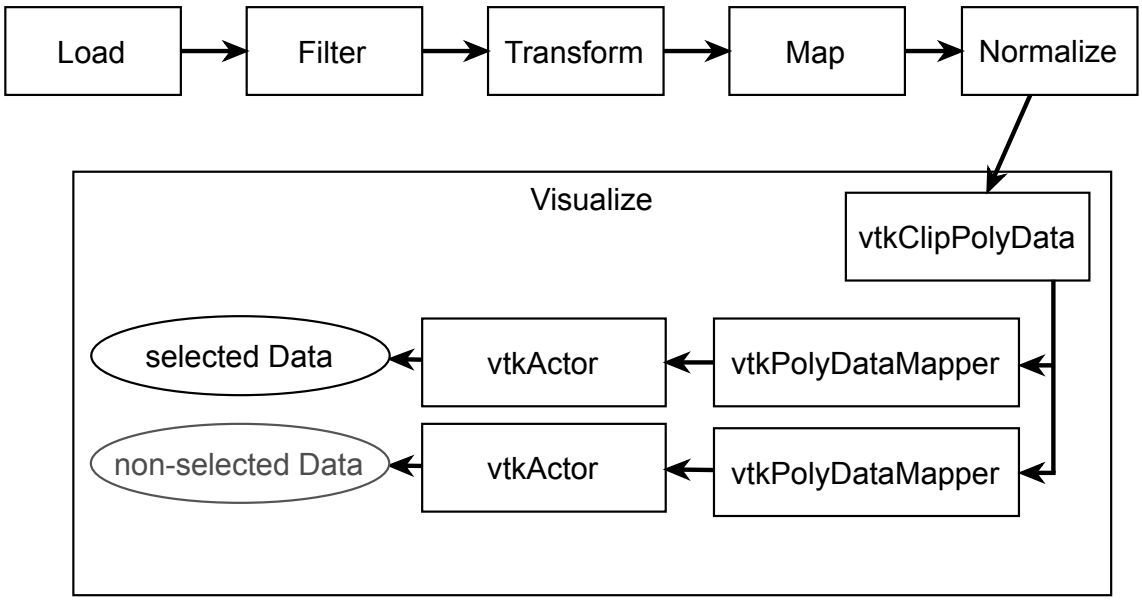


Abbildung 4.12: Die Viper-Pipeline mit angefügtem Select-Cut-Tool

In Abbildung 4.13 ist Auswahlebene während der Manipulation zu sehen.

Wird das Tool zum Selektieren von Partikeldaten (welche VIPER als `vtkPolyData` darstellt) verwendet, so kann es frei verschoben und skaliert werden. Wird es dagegen zur Bearbeitung von Felddaten (intern per `vtkImageData` gespeichert) eingesetzt, so wird das Tool per Snapping-Mechanismus immer

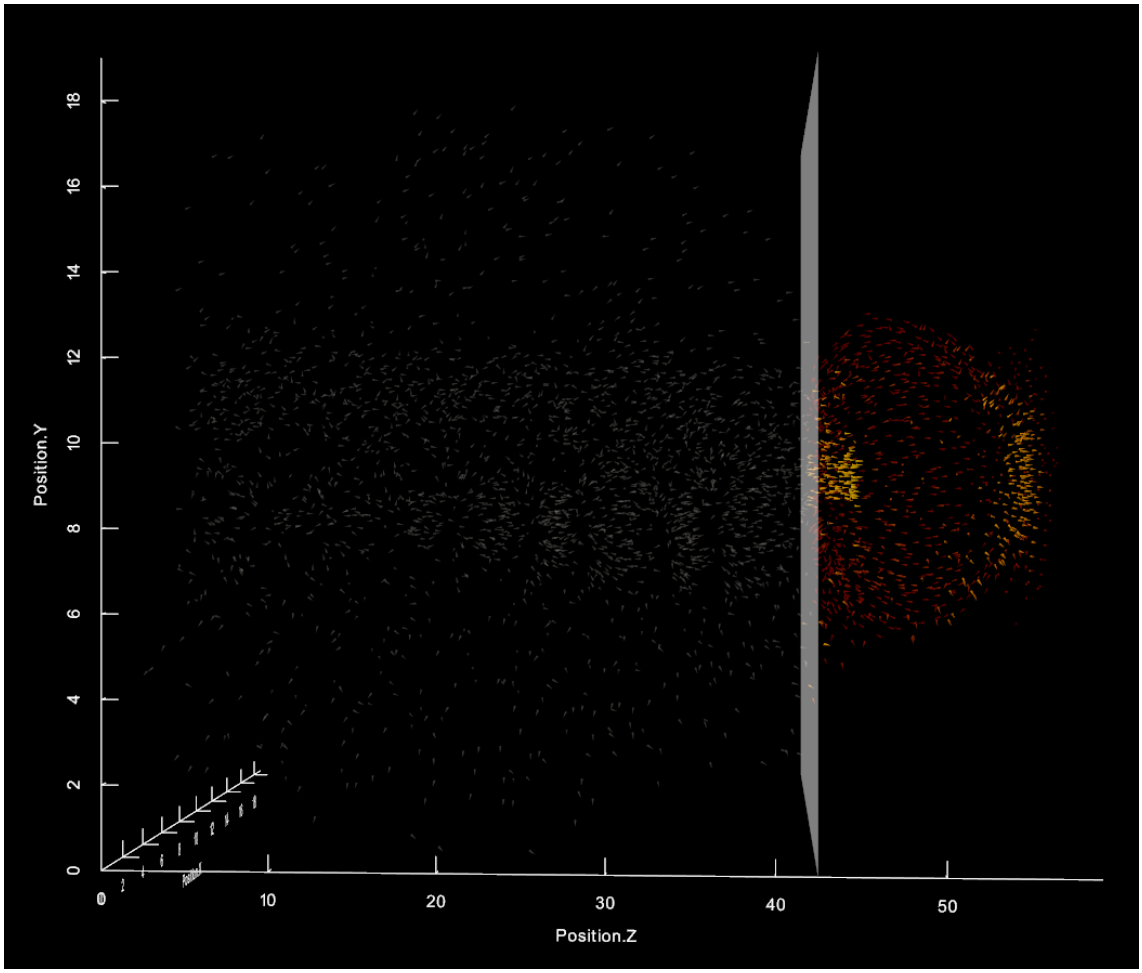


Abbildung 4.13: Das Select-and-Cut Tool als Ebene

an den Gitterpunkten ausgerichtet (vgl. [FSSB06]). Die implizite Speicherung in einer Gitterdatenstruktur ist in Hinblick auf die Datenhaltung wesentlich effizienter als das explizite Speichern der Gitterpunkte und -zellen. So muss nur der Ursprung des Gitters (engl. *origin*), die Dimensionen (engl. *dimensions*) und der Abstand der Gitterpunkte (engl. *spacing*) gespeichert werden. Um nach der Selektion keine halben Zellen zu erhalten, muss für die gewünschte Auswahl immer der nächstgelegene Gitterpunkt gefunden und die Auswahlbox oder -ebene daran ausgerichtet werden. Um den entsprechenden Gitterpunkt zu finden wurde die Klasse `vtkPointLocater` verwendet.

Wird das Tool nun angewendet, kommt es wieder darauf an welcher Typ an Daten bearbeitet wird. Handelt es sich um Partikeldaten (also PolyData), so wird eine Instanz von `VtkDataFilter` erstellt und in die Pipeline eingebaut. Dieser Filter wird durch einen VQL-Ausdruck parametrisiert, welcher schließlich als Auswahlkriterium für verarbeitete Daten dient. Folgender Codeabschnitt zeigt, wie der VQL-Ausdruck für eine Auswahlebene erzeugt wird:

```
if ( m_alignment == PositiveX || m_alignment == PositiveY || m_alignment == PositiveZ )
{
        m_polyFilter->SetFilter
        (
                new viper::vql::Less<double>
                (
                        new viper::vql::FieldVariable<double>(axis),
```

```
                        new viper::vql::Const<double>(axisPos)
                )
        );
}
else
{
        m_polyFilter->SetFilter
        (
                new viper::vql::Greater<double>
                (
                        new viper::vql::FieldVariable<double>(axis),
                        new viper::vql::Const<double>(axisPos)
                )
        );
}

visualization->pushPolyDataFilter( m_polyFilter );
```

Der Bezeichner `axis` speichert dabei den Namen des Merkmals, welches auf die Achse des Tools gemappt wurde. `axisPos` bestimmt die Position der Ebene auf der Achse. Für die X-Achse werden diese Parameter wie folgt bestimmt:

```
double origin[3];
m_plane->GetOrigin( origin );

axisPos = origin[0];
axis = (*visualization->getAssigns())["Axis.X"];
```

Handelt es sich um Felddaten (also ImageData), so wird statt `VtkDataFilter` die Klasse `vtkExtractVOI` instanziiert. Diese wird mit den entsprechenden Indizes parametrisiert:

```
int imin = 0, jmin = 0, kmin = 0;
int imax = visualization->getImageDataDimensions()[0] - 1;
int jmax = visualization->getImageDataDimensions()[1] - 1;
int kmax = visualization->getImageDataDimensions()[2] - 1;

int * ijk = findStructuredGridCoords( origin, visualization ); /// uses vtkPointLocator

switch(m_alignment)
{
        case PositiveX: imax = ijk[0]; break;
        case NegativeX: imin = ijk[0]; break;
        case PositiveY: jmax = ijk[1]; break;
        case NegativeY: jmin = ijk[1]; break;
        case PositiveZ: kmax = ijk[2]; break;
        case NegativeZ: kmin = ijk[2]; break;
        default: break;
}

m_imageFilter->SetVOI( imin, imax, jmin, jmax, kmin, kmax );

visualization->pushImageDataFilter( m_imageFilter );
```

4.4.4.2 Probe-Field Tool

Das Probe-Field Tool legt eine Schnittebene durch die visualisierten Felddaten. Auf dieser Ebene werden die Merkmale des Skalarfeldes farbkodiert dargestellt. Abbildung 4.15 zeigt das Tool im Einsatz. Es kann

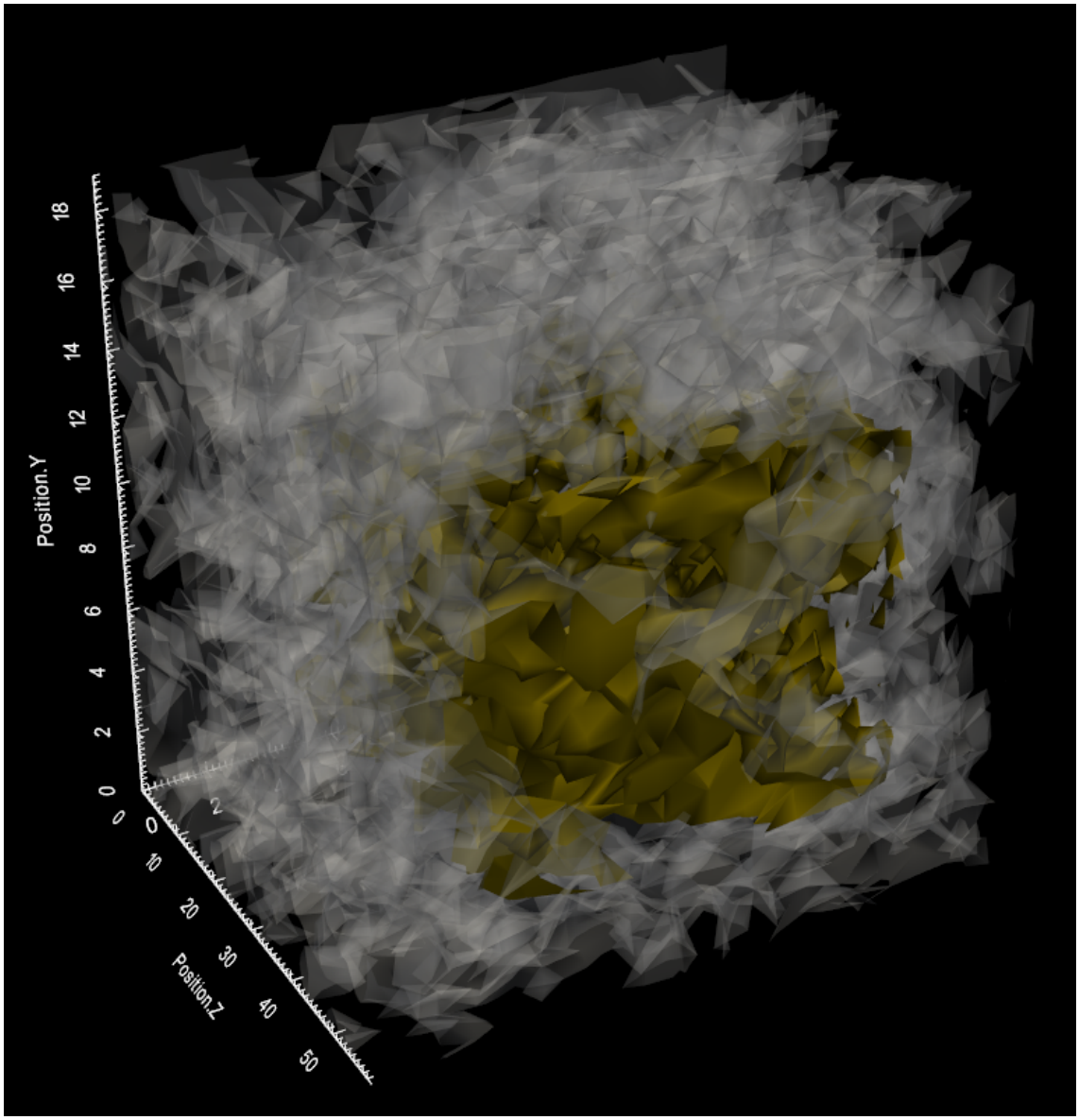


Abbildung 4.14: Das Select-and-Cut Tool als Box

im Manipulationsmodus mit Hilfe der Einhand-Translations-Geste (`move`-Befehl) verschoben werden. Die Verschiebung ist auf die beim Erstellen des Tools festgelegte Raumachse beschränkt. Folgender Codeausschnitt zeigt, wie der `move`-Befehl vom Tool verarbeitet wird:

```
void PlaneCutTool::handleCommand( char * cmdString )
{
        if ( strstr( cmdString, "move: ") )
        {
                double x, y, z;
                sscanf( cmdString, "move: %lf %lf %lf", &x, &y, &z );

                this->move(-x * 0.5, -y * 0.5, -z * 0.5);
        }
}

void PlaneCutTool::move( double x, double y, double z )
{
```

```
        double origin[3];
        m_plane->GetOrigin( origin );

        // move only on the axis the tool is aligned to
        switch ( m_alignment )
        {
                case PositiveX:
                case NegativeX:
                        origin[0] += x; break;
                case PositiveY:
                case NegativeY:
                        origin[1] += y; break;
                case PositiveZ:
                case NegativeZ:
                        origin[2] += z; break;
                default: break;
        }

        m_plane->SetOrigin( origin );
}
```

Die vom VDX empfangenen Befehle werden also wie in Abschnitt 4.4 beschrieben an das Tool weitergereicht und dort entsprechend verarbeitet (oder verworfen).

Implementiert wurde das Tool mit Hilfe der VTK-Klassen `vtkCutter`, `vtkPolyDataMapper` und `vtkActor`. Es handelt sich hierbei um ein rein visuelles Tool. Wird es angewendet, so verschwindet es wieder aus der Pipeline und die Visualisierung wird wieder dargestellt wie vor der Erstellung des Tools.

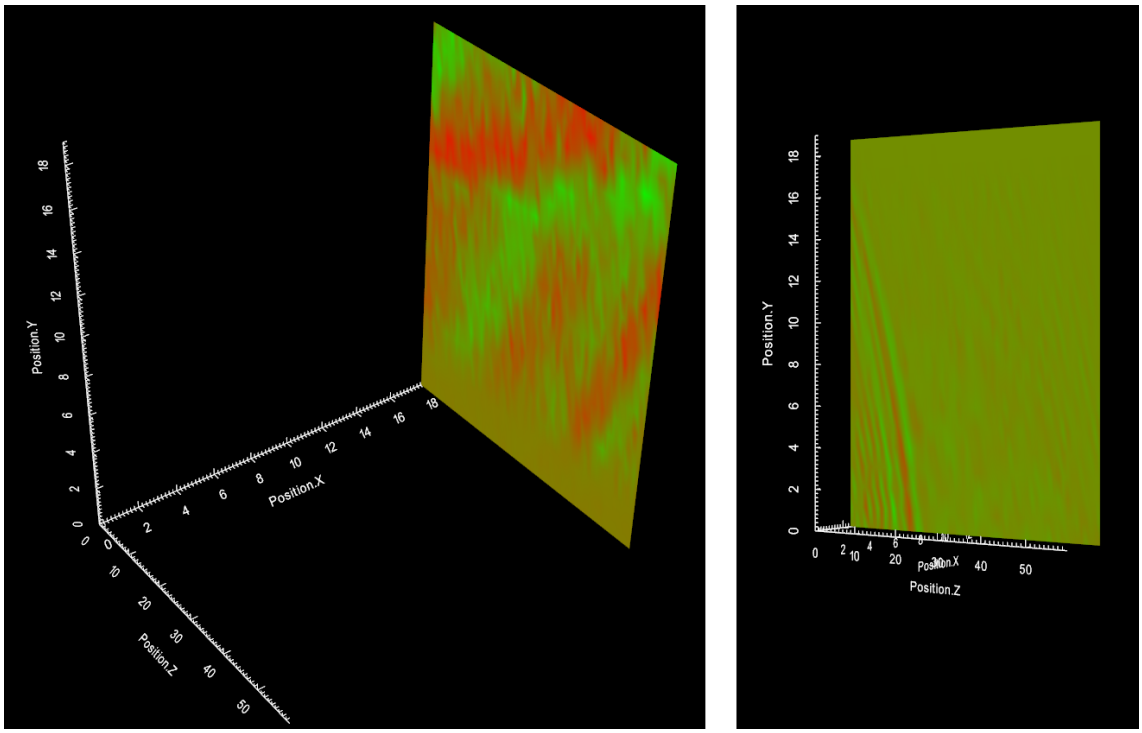


Abbildung 4.15: Zwei Schnitte durch ein (E-)Feld mit dem Probe-Field-Tool

4.4.4.3 Measure Tool

Das Measure Tool stellt ein achsenausgerichtetes 3D Lineal dar. Es kann zur Messung von Merkmalen verwendet werden, die einer Raumachse zugeordnet sind. Dabei erfolgt die Messung immer entlang einer festen Achse, für die das Tool erstellt wurde. Eine Messung in allen drei Dimensionen wäre für den Nutzer schwerer zu handhaben und macht in Fällen, in welchen den Raumachsen unterschiedliche Merkmale zugeordnet sind, keinen Sinn. Ist beispielsweise der X-Achse die Teilchenposition, der Y-Achse die Energie und der Z-Achse die ID des Teilchens zugeordnet, so ist ein Messwert durch alle drei Dimensionen nicht sinnvoll interpretierbar.

Die zur Umsetzung genutzte Klasse `vtkDistanceRepresentation` zeichnet eine Art 3D-Lineal mit Textausgabe für dessen Länge. Zusätzlich wurde ähnlich dem Select-Cut-Tool eine Erweiterung für die Visualisierungspipeline implementiert, welche den Teil der Visualisierung in dem sich das Tool befindet, transparent darstellt. Andernfalls wäre das Tool inmitten einer Isoflächendarstellung verdeckt. Abbildung 4.16 zeigt den Einsatz des Werkzeugs zur Messung einer Energiedifferenz entlang der Y-Achse.

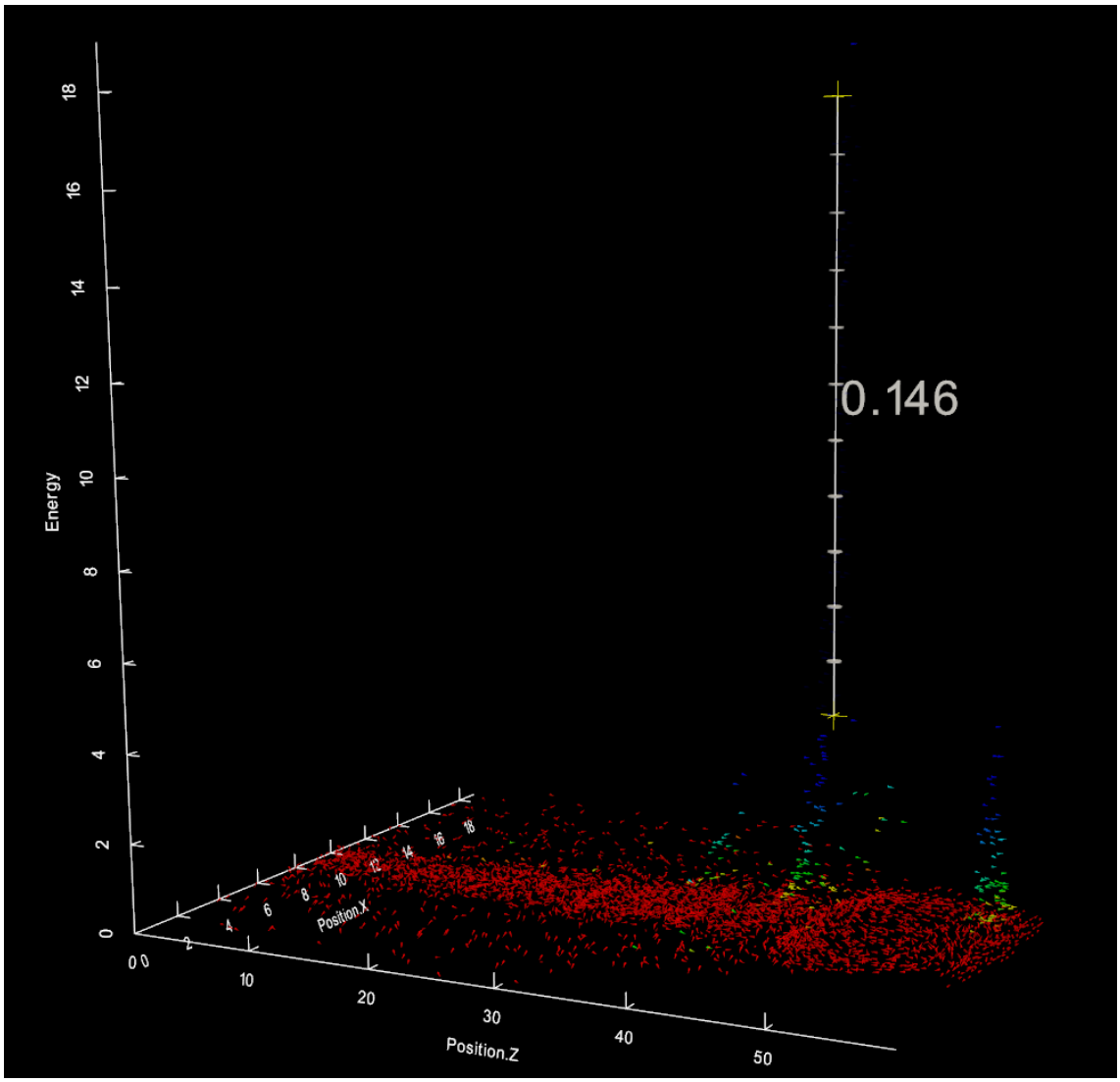


Abbildung 4.16: Das Measure-Tool im beispielhaften Einsatz zur Messung einer Energiedifferenz

5 Evaluation

In diesem Kapitel sollen auf informelle Art und Weise die Erfahrungen und Beobachtungen beschrieben werden, welche beim Test und bei der Vorführung des Systems gemacht wurden. Abschließend werden an einem Beispielszenario die Laufzeit- und Speicheranforderungen analysiert.

5.1 Aufbau & Testpersonen

Der Aufbau des Systems entsprach stets dem in Abschnitt 3.4 beschriebenen Powerwall-Setup, welches sich im VR-Labor des HZDR befindet, bzw. dem sehr ähnlichen Aufbau im Computergraphik-Labor der Fakultät Informatik an der TU Dresden.

Als Probanden fungierte eine Vielzahl an Personen. Darunter Mitarbeiter der Technischen Universität Dresden und des Helmholtz-Zentrum Dresden-Rossendorf sowie Workshop-Teilnehmer und Besucher am Tag des offenen Labors. Damit ergibt sich ein breites Spektrum von technisch-versierten bis hin zu unerfahrenen Nutzern.

5.2 Testdurchführung

In den meisten Fällen erfolgte der Selbstversuch durch die Probanden nach einer Präsentation des Systems. Damit kann also stets ein Vorwissen über die grundlegende Funktionsweise und das umgesetzte Interaktionskonzept vorausgesetzt werden. Nichtsdestotrotz wurden die Nutzer auch vor und während des Tests eingewiesen.

5.3 Auswertung

Beim Neustart des Systems sieht der Anwender als erstes den Konfigurationsdialog. In dessen erster Version waren die dargestellten GUI-Elemente noch am oberen linken Bildrand ausgerichtet. Einige Nutzer versuchten daraufhin ihre Hand über den GUI-Elementen zu platzieren. Da kein virtueller Cursor vorhanden ist, diente ihnen ihre Hand als indirekter Cursor-Ersatz. Die Folge war eine sehr anstrengende, unergonomische Körperhaltung für den Nutzer sowie Fehler beim Tracking auf Seiten des Systems, da sich die Hand am Rande oder außerhalb des von der Kamera erfassbaren Bereichs befand. Als Konsequenz dessen wurden alle interaktiven GUI-Elemente in späteren Versionen auf der Leinwand zentriert. Eine minimalistische GUI in Verbindung mit einer Gestensteuerung kann also viel schneller erlernt werden, als ein überladenes WIMP-Interface, gesteuert durch eine Maus. Ein gewisser Lernaufwand ist

dennoch erforderlich um ein effektives Arbeiten zu gewährleisten. Dabei darf die Tatsache, dass auch der Erstkontakt mit Maus und Tastatur oft mit Schwierigkeiten behaftet ist, nicht vernachlässigt werden.

Es hat sich gezeigt, dass nach einer Lern- und Übungsphase von wenigen Minuten (im Fall eines Probanden waren fünf Minuten bereits ausreichend) Nutzer und System aufeinander eingespielt sind. Der Anwender hatte das Interaktionskonzept verinnerlicht und der Tracker war ausreichend gut auf die neue Person kalibriert, sodass ein selbstständiges, effektives Arbeiten möglich war.

Als besonders eindrucksvolles Feature hat sich das implementierte Stereo-Rendering erwiesen. Es ist nicht nur effektvoll, sondern fördert auch die räumliche Wahrnehmung und somit das Verständnis bzw. die Expressivität der Visualisierung. Dies ist besonders hilfreich bei abstrakten Darstellungen (z.B. Zeit-Energie-Positions-Diagramm), in denen die Tiefenwahrnehmung aus Vorwissen über die Beschaffenheit der Daten fehlt (wie z.B. beim Partikelplot mit Positionsdaten auf den drei Raumachsen). Zu beachten ist noch, dass sich die Nutzer immer möglichst genau vor der Leinwand positionieren sollten und den Kopf nicht in Schräglage bringen dürfen. Andernfalls treten beim eingesetzten passiven Stereo-Rendering Ghostingeffekte auf. Dies und die Tatsache, dass nicht überall der kostenintensive Powerwall-Aufbau vorhanden ist, machten die Implementierung der anderen Stereo-Techniken notwendig. Der Vorteil der Stereodarstellung liegt in der gesteigerten Immersion. Der Nutzer wird viel stärker in die virtuelle Umgebung involviert, als es durch ein Desktopsystem möglich wäre. Diese Vertiefung in die Arbeit mit den Daten regt zum Experimentieren an und kann auch die Konzentration und Motivation steigern. Ein Zustand der in der Psychologie als ”Flow” bezeichnet wird [Csi97].

Eine weitere Fehlerquelle, die sich bei den Vorführungen gezeigt hat, ist die Erkennung bzw. Verarbeitung mehrerer Personen in der vom Tracker erfassten Umgebung. Standen viele Personen hinter dem Nutzer, so hatte der Tracker Probleme die große Anzahl erkannter Skelette voneinander zu trennen und schnell genug zu verarbeiten. Was für einen Nutzer funktionierte, stellte in solchen Demonstrationsszenarios ein großes Problem dar. Die Trackingdaten waren stark verrauscht und die Gestenerkennung instabil. Deshalb wurde seitens des Trackers die Anzahl aktiv getrackter Skelette auf eines reduziert. Zusätzlich wurde der getrackte Tiefenbereich eingegrenzt, um in einem gewissen Abstand hinter dem Nutzer vorbeigehende Personen gar nicht erst zu erkennen. Somit bietet das entwickelte System auch die Möglichkeit, gemeinsam über die visualisierten Daten zu diskutieren und Ergebnisse interaktiv zu präsentieren.

5.4 Performance-Analyse

In einem beispielhaften Arbeitsszenario wurden mit einer Partikel- und einer Feldvisualisierung (Isoflächen) Laufzeit und Speicherverbrauch gemessen. Die Testhardware entsprach dabei der aus Abschnitt 3.4, soll aber hier aus Gründen der Vollständigkeit nochmals aufgelistet werden:

- Prozessor: Intel Xeon X5650 @ 2,67 GHz Hexa-Core (12 Hardware-Threads)
- Grafikkarte: NVidia Quadro 5000 @ 510 MHz GPU Takt mit 2,5 GB GDDR5 Grafikspeicher @ 1,5 GHz
- Arbeitsspeicher: 12 GB

Es wurde die Zeit, die zum Laden der Daten nötig war und die durchschnittliche Framerate im Navigationsmodus gemessen. Da in der Feldvisualisierung abhängig von der Beobachterposition starke Schwankungen in der Bildrate zu beobachten waren, wurde hier zusätzlich die minimale und maximale Framerate gemessen. Die Tabellen 5.1 und 5.2 geben die genauen Messwerte wieder. Die Diagramme aus den Abbildungen 5.1 und 5.2 zeigen die Abhängigkeit zwischen Anzahl geladener Partikel bzw. geladenem Feldbereich und den Größen Ladezeit, Hauptspeichernutzung sowie minimaler, maximaler und durchschnittlicher Framerate.

Testnr.	Anzahl Partikel	Ladezeit [s]	Ø FPS	RAM [MB]
1	46560	2,6	231,0	216
2	60032	2,8	219,2	228
3	84797	3,5	192,8	276
4	131112	4,5	155,2	360
5	180822	5,5	117,7	444
6	240209	6,9	93,3	564
7	298597	8,4	74,9	672
8	361592	9,5	65,5	780
9	480874	12,3	48,2	1008
10	720640	18,3	35,5	1452

Tabelle 5.1: Frameraten und Speicherbedarf der Partikelvisualisierung

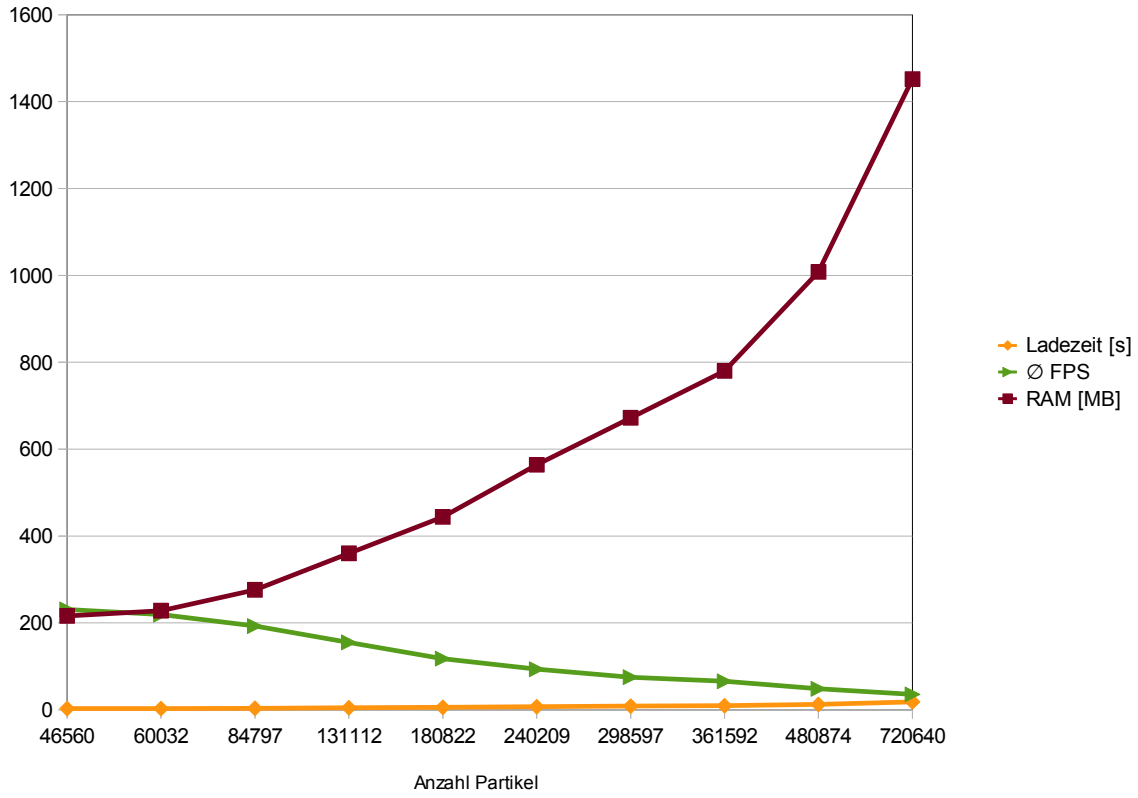


Abbildung 5.1: Ladezeit, durchschnittliche Framerate und genutzer RAM in Abhängigkeit von der Partikelanzahl

Testnr.	Feldbereich (X,Y,Z)	Ladezeit [s]	min. FPS	max. FPS	Ø FPS	RAM [MB]
1	(0,59) (0,59) (0,40)	2,8	39	293	145	204
2	(0,59) (0,59) (0,80)	7,0	19	220	133	360
3	(0,59) (0,59) (0,140)	9,1	12,3	180	104	468
4	(0,59) (0,59) (0,190)	20,0	10,1	160	85,3	816
5	(0,59) (0,59) (0,250)	20,1	6,9	131	73,6	1092
6	(0,59) (0,59) (0,300)	23,2	5,7	120	71,5	1260
7	(0,59) (0,59) (0,340)	24,3	5,3	106	69,6	1284
8	(0,59) (0,59) (0,385)	25,8	4,1	90	60,3	1356

Tabelle 5.2: Frameraten und Speicherbedarf der Feldvisualisierung

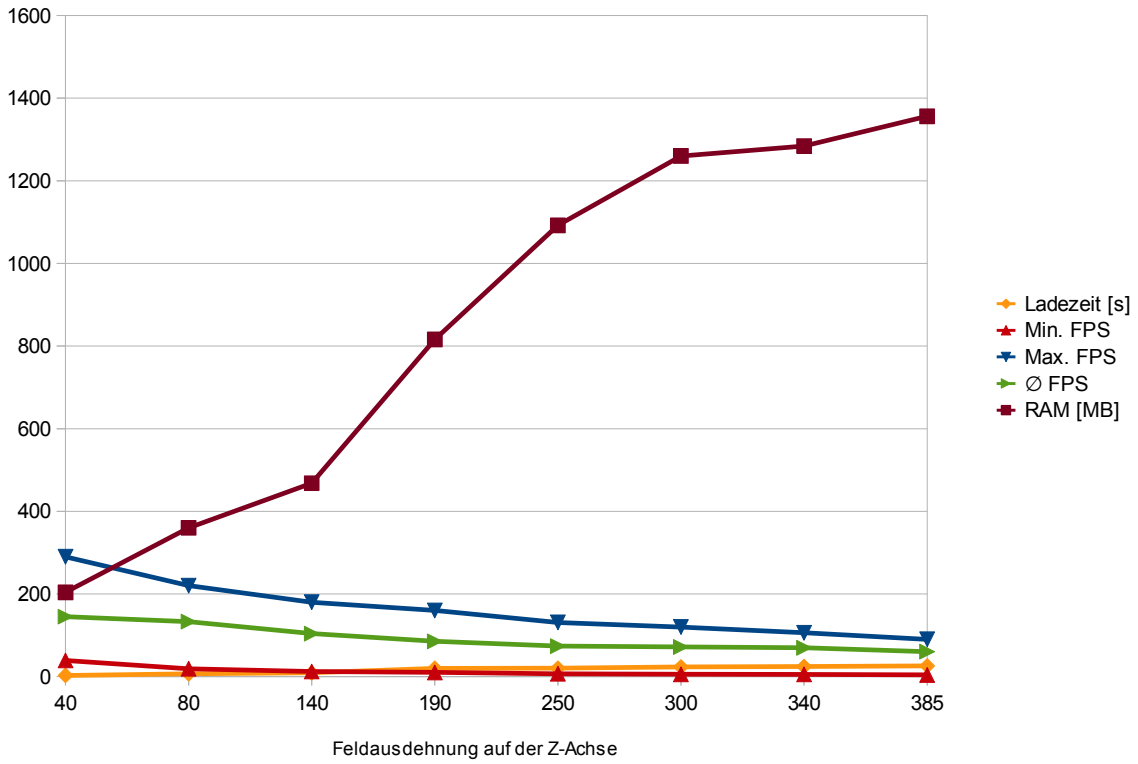


Abbildung 5.2: Ladezeit, minimale, maximale und durchschnittliche Framerate sowie genutzer RAM in Abhängigkeit von der Feldgröße

6 Ausblick

Während Konzeption, Entwicklung und Test des Systems erwiesen sich einige Funktionen und Charakteristiken des Systems als verbesserungs- bzw ausbaufähig.

Der Viper Data Explorer wurde lediglich als Prototyp implementiert. Er zeigt, wie eine gestengesteuerte Datenanalyse ablaufen kann. Aus diesem Grund wurde auch nur eine begrenzte Anzahl an Tools umgesetzt. Weitere Tools und Utilities, wie ein Minitaschenrechner oder eine Exportfunktion für Präsentationsvideos sind nur einige Ideen die (noch) nicht im System implementiert wurden. Spezialisierte Werkzeuge, die kontextsensitiv Funktionalität anbieten und so mehr "Intelligenz" im System unterbringen, erfordern eine genaue Analyse der Anforderungen und Arbeitsabläufe.

Gegenwärtig wird die Gestensteuerung als einziger Interaktionsmodus genutzt. Diese eignet sich hervorragend zur Navigation und direkten Manipulation der Visualisierung. Für Aufgaben wie das wechseln der Bearbeitungsmodi ist sie jedoch eher umständlich (Umweg über symbolische Gesten). Wie schon Bill Buxton feststellte: *Everything is best for something and worst for something else.* [Bux12] Bestimmte Eingabemodalitäten sind also für bestimmte Aufgaben besser geeignet als andere. Ein multimodales Interaktionskonzept könnte die Usability des Systems stark verbessern. In diesem Zusammenhang sollte untersucht werden, welcher Modus für welche Interaktionen am besten geeignet ist. Ein erster Ansatz wäre, Spracherkennung für die Moduswechsel zu implementieren und anschließend die Usability erneut zu evaluieren.

Durch die Entkopplung von Tracker und Visualisierungskomponente ist eine Multi-User-Session mit mehreren Kinect-Kameras und -Trackern leicht umzusetzen. Dazu kann ein Publish-Subscribe-Mechanismus für die Synchronisation der internen Zustände von Trackern und VDX eingesetzt werden. Desweiteren sollte immer nur ein User gleichzeitig aktiv sein bzw. nur Eingaben von einem Nutzer verarbeitet werden. Beginnt eine Person die Interaktion, so sollten alle anderen passiv geschaltet werden. Erst wenn dieser die Interaktion beendet hat, kann ein anderer Nutzer Befehle an das System senden.

Zur Verarbeitung sehr großer Datenmengen kann das Visualisierungssystem verteilt werden. Sowohl das Laden und Filtern der Daten sowie das Rendern bietet Ansatzpunkte zur Parallelisierung. [ALS+00] und [SMS03] stellen geeignete Herangehensweisen und Architekturen vor. Die gesamte Verarbeitung kann dabei auf einem Großrechner erfolgen. Das gerenderte Bild wird dann als Videostream an einen Client übertragen, auf dem das User Interface angezeigt wird und die Interaktion durch den Nutzer erfolgt (sog. *Remote Desktop Delivery* [MT03]).

7 Zusammenfassung

In dieser Arbeit wurden Konzepte der natürlichen Interaktion mit denen der wissenschaftlichen Visualisierung kombiniert, um eine virtuelle Arbeitsumgebung zur visuellen Datenanalyse zu schaffen. Dabei stand die intuitive Nutzbarkeit des Systems und das immersive Erforschen der vorliegenden Laser-Plasma-Simulationsdaten im Vordergrund. Es galt, ein Interaktionskozept zu entwickeln, welches vom Nutzer leicht erlernt werden kann, indem es an ihm bekannte Metaphern aus der realen Welt oder andere weitverbreitete Systeme angelehnt ist. Es sollte den Nutzer nicht durch zu viel gleichzeitig verfügbare Funktionalität überfordern und dennoch die wichtigsten Mittel zur Datenauswertung bereitstellen.

Die Ergebnisse der Diplomarbeit von Lukas Zühl dienten als Grundlage für das Visualisierungssystem. Durch Anpassung, Erweiterung und Ergänzung einiger Klassen, wurde das flexible Visualisierungssystem zu einem einfachen Datenanalysewerkzeug weiterentwickelt. So wurden Bearbeitungswerkzeuge und -modi eingeführt, die Exploration und Auswertung der Daten unterstützen. Mit Hilfe des implementierten Stereo-Rendering können die Daten noch besser erfahren und verstanden werden.

Ergebnis dieser Arbeit ist ein Software-Tool, das zur Erfüllung einfacher, aber essentieller, Aufgaben aus dem Bereich der wissenschaftlichen Datenanalyse und Visualisierung genutzt werden kann. Ein Beispiel für das Finden und Präsentieren eines Untersuchungsergebnisses ist in Abbildung 7.1 zu sehen. Zunächst wurde das elektro-magnetische Feld im Raum visualisiert. Durch Navigation und Selektion wurde dann ein interessanter Datenbereich ausgewählt (in der Abb. links). Dazu kam das Select-Cut Tool zum Einsatz. Mit Hilfe des Probe-Field Tools wurden schließlich die beiden dargestellten Schnitte durch das Feld gemacht (mittlere Bilder der Abb.). In einer neuen Visualisierung wurde dann derselbe Raumausschnitt wie zuvor in der Isoflächendarstellung ausgewählt. Dieses Mal wurden die Partikeldaten als 3D-Plot visualisiert, wobei eine Farbskala die Teilchenenergie auf Farben abbildet. Nun ist auch für Nicht-Physiker der Zusammenhang zwischen Raumladungen - sichtbar in den Feldschnitten - und hochenergetischen Teilchen leicht erkennbar.

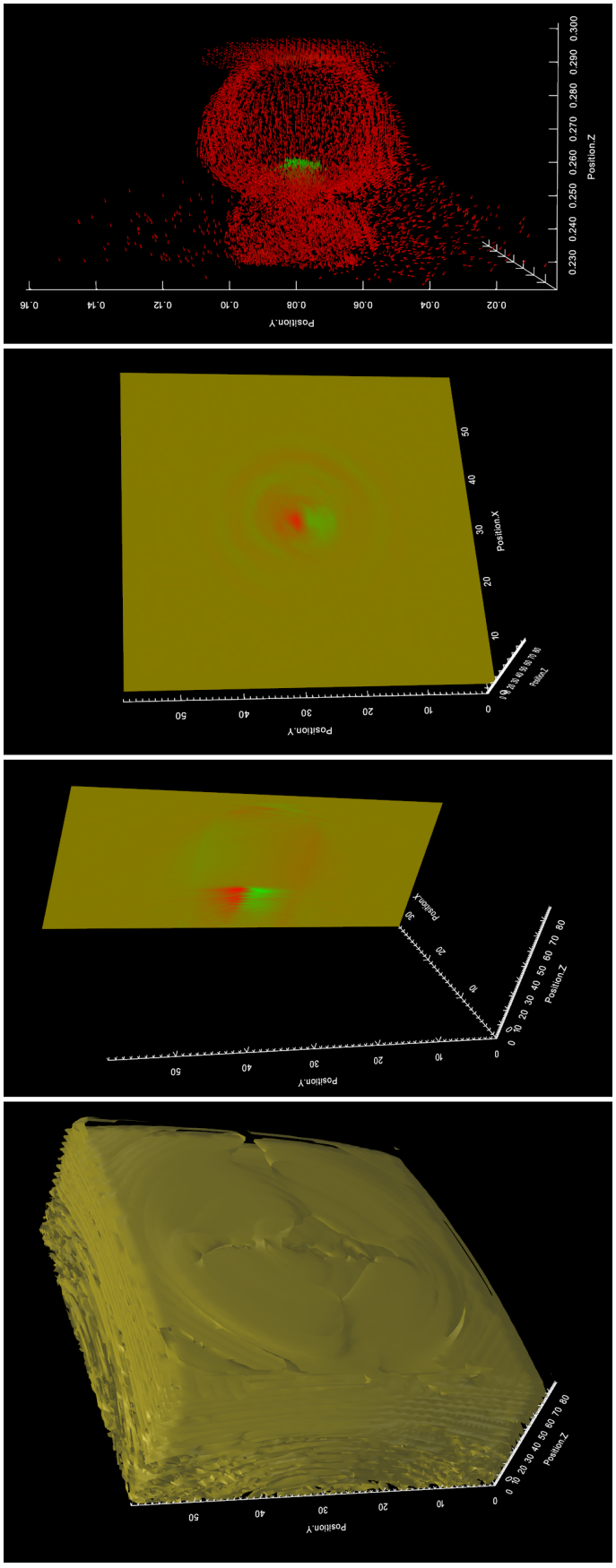


Abbildung 7.1: Korrelation zwischen Partikeln und Feldern

Literaturverzeichnis

[AA07] ANDRIENKO, Gennady L. ; ANDRIENKO, Natalia V. *Visual Data Exploration: Tools, Principles, Problems*. Fisher, P.F.: Classics from IJGIS. Twenty years of the International journal of geographical information science and systems, Boca Raton, Taylor & Francis. 2007

[ALS+00] AHRENS, James ; LAW, Charles ; SCHROEDER, Will ; MARTIN, Ken ; PAPKA, Michael. *A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets*. Los Alamos National Laboratory - Technical Report #LAUR-00-1620. 2000

[BDG+04] BRODLIE, K. W. ; DUCE, D. A. ; GALLOP, J. R. ; WALTON, J. P. R. B. ; WOOD, J. D.: Distributed and Collaborative Visualization. In: *Computer Graphics Forum* 23 (2004), Nr. 2, S. 223–251. – ISSN 1467–8659

[BKJJLP04] BOWMAN, Doug A. ; KRUIJFF, Ernst ; JOSEPH J. LAVIOLA, Jr. ; POUPYREV, Ivan: *3D User Interfaces: Theory and Practice*. Addison-Wesley, 2004

[Bry96] BRYSON, Steve. *Virtual Reality in Scientific Visualization*. Communications of the ACM, Vol. 39, No. 5. Mai 1996

[Bux12] BUXTON, Bill. *Bill Buxtons Homepage*. `http://www.billbuxton.com/`. Mai 2012

[BWH+10] BURAU, Heiko ; WIDERA, Renée ; HÖNIG, Wolfgang ; JUCKELAND, Guido ; DEBUS, Alexander ; KLUGE, Thomas ; SCHRAMM, Ulrich ; COWAN, Tomas E. ; SAUERBREY, Roland ; BUSSMANN, Michael. *PIConGPU: A Fully Relativistic Particle-in-Cell Code for a GPU Cluster*. IEEE Transactions on Plasma Science, Vol. 38, No. 10,. Oktober 2010

[Che93] CHEN, P. C. *A Climate Simulation Case Study*. Proceedings of Visualization '93, IEEE Computer Society Press. 1993

[CNSD93] CRUZ-NEIRA, Carolina ; SANDIN, Daniel J. ; DEFANTI, Thomas A.: Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1993 (SIGGRAPH '93). – ISBN 0–89791–601–8, S. 135–142

[Csi97] CSIKSZENTMIHALYI, Mihaly: *Finding Flow: The Psychology of Engagement with Everyday Life*. 1997

[DLS02] VAN DAM, Andries ; LAIDLAW, David H. ; SIMPSON, Rosemary M. *Experiments in Immersive Virtual Reality for Scientific Visualization*. Computer and Graphics. August 2002

[FSSB06] FABRE, Arnaud ; STERNBERGER, Ludovic ; SCHRECK, Pascal ; BECHMANN, Dominique. *Constrained Gesture Interaction in 3D Geometric Constructions*. 2006

[GHJV94] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph E. ; VLISSIDES, John: *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994

[Gra] GRASBERGER, Herbert. *Introduction to Stereo Rendering*

[Gum02] GUMHOLD, Stefan. *Maximum Entropy Light Source Placement*. IEEE Visualization. Oktober 2002

[HA06] HEER, Jeffrey ; AGRAWALA, Maneesh. *Software Design Patterns for Information Visualization*. IEEE Transactions on Visualization and Computer Graphics, Vol. 12, No. 5,. Oktober 2006

[HE97] HARLING, P.A. ; EDWARDS, A.D.N. *Hand Tension as a Gesture Segmentation Cue*. Progress in Gestural Interaction. 1997

[HH98] HUANG, Chung-Lin ; HUANG, Wen-Yi. *Sign language recognition using model-based tracking and a 3D Hopfield neural network*. Machine Vision and Applications, Vol. 10. 1998

[HM90] HABER, R. B. ; MCNABB, D. A. *Visualization idioms: A conceptual model for scientific visualization systems*. Visualization in Scientific Computing, pages 74-93, IEEE Computer Society Press. 1990

[HM03] HALLE, Michael ; MENG, Jeanette. *LightKit: A lighting system for effective visualization*. IEEE Visualization. 2003

[IHS+01] II, Russell M. T. ; HUDSON, Thomas C. ; SEEGER, Adam ; WEBER, Hans ; JULIANO, Jeffrey ; HELSER, Aron T. *VRPN: A Device-Independent, Network-Transparent VR Peripheral System*. VRST 2001 Conference. 2001

[KH90] KURTENBACH, Gord ; HULTEEN, Eric. *Gestures in Human-Computer Communication*. Addison-Wesley Publishing Co. 1990

[Kit12] KITWARE. *ParaView Homepage*. http://www.paraview.org/. April 2012

[KKEM10] KEIM, Daniel A. ; KOHLHAMMER, Jörn ; ELLIS, Geoffrey ; MANSMANN, Florian: *Mastering the Information Age - Solving Problems with Visual Analytics*. Eurographics, 2010

[KMS+08] KEIM, Daniel ; MANSMANN, Florian ; SCHNEIDEWIND, Jörn ; THOMAS, Jim ; ZIEGLER, Hartmut: Visual Analytics: Scope and Challenges. In: SIMOFF, Simeon (Hrsg.) ; BÖHLEN, Michael (Hrsg.) ; MAZEIKA, Arturas (Hrsg.): *Visual Data Mining* Bd. 4404. Springer Berlin / Heidelberg, 2008. – 10.1007/978-3-540-71080-6_6. – ISBN 978–3–540–71079–0, S. 76–90

[LJ99] LAVIOLA, Joseph J. ; JR. *MSVT: A Virtual Reality-Based Multimodal Scientific Visualization Tool*. 1999

[Mac86] MACKINLAY, Jock: Automating the design of graphical presentations of relational information. In: *ACM Trans. Graph.* 5 (1986), April, Nr. 2, S. 110–141. – ISSN 0730–0301

[Mat12] MATHWORKS. *MATLAB - Die Sprache fÃ¼r technische Berechnungen*. `http://www.mathworks.de/products/matlab/`. 2012 Abruf: 21.05.2012

[mey02] *Meyers Lexikon, 9. Auflage*. Bibliographisches Institut, Mannheim, November 2002. – ISBN 3411012501

[MT03] MORELAND, Kenneth ; THOMPSON, David. *From Cluster to Wall with VTK*. IEEE Symposium on Parallel and Large-Data Visualization and Graphics. 2003

[Nie12] NIELSEN, Jakob. *Ten Usability Heuristics*. `http://www.useit.com/papers/heuristic/heuristic_list.html`. Mai 2012

[OFD04] OVIATT, S. ; FLICKNER, M. ; DARRELL, T.: Multimodal interfaces that flex, adapt, and persist. In: *Communications of the ACM* 47 (2004), Nr. 1, S. 30–33

[OL03] DE OLIVEIRA, Maria Cristina F. ; LEVKOWITZ, Haim. *From Visual Data Exploration to Visual Dta Mining: A Survey*. IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 3. 2003

[pla12] PLAY.COM. *Kinect Technical Details*. `http://www.play.com/Games/Xbox360/4-/10296372/Project-Natal/Product.html`. Abruf: 23.05.2012

[RF94] ROBERTSON, P. ; FERRARI, L. D. *Systematic Approaches to Visualization: Is a Reference Model Needed?* Scientific Visualization Advances and Challenges Academic Press. 1994

[Skr12] SKRIPCAK, Tomas. *Design and Implementation of Natural User Interaction Framework for Virtual Reality and Visualisation Applications*. 2012

[SM00] SCHUMANN, Heidrun ; MÜLLER, Wolfgang: *Visualisierung - Grundlagen und allgemeine Methoden*. Springer-Verlag, 2000

[SML98] SCHROEDER, Will ; MARTIN, Ken ; LORENSEN, Bill: *The Visualization Toolkit, 2nd Edition*. Prentice Hall PTR, 1998

[SMS03] SCHNEIDER, Sascha ; MAY, Thorsten ; SCHMIDT, Michael. *Parallel architecture of an interactive scientific visualization system for large datasets*. OpenSG Symposium. 2003

[SWH05] STALLING, Detlev ; WESTERHOFF, Malte ; HEGE, Hans-Christian. *Amira - A Highly Interactive System for Visual Data Analysis*. `http://www.amira.com/`. 2005

[TC05] THOMAS, J.J. ; COOK, K.A. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE CS Press. 2005

[WB06] WELCH, Greg ; BISHOP, Gary. *An Introduction to the Kalman Filter*. `http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf`. 2006

[Wik12a] WIKIPEDIA. *Tracking*. `http://de.wikipedia.org/wiki/Tracking`. 2012, Abruf: 20.05.2012

[Wik12b] WIKIPEDIA. *Stereoskopie*. `de.wikipedia.org/wiki/Stereoskopie`. Abruf: 30.05.2012

[WT91] WATERS, K. ; TERZOPOULOS, D. *Modeling and Animating Faces Using Scanned Data*.

Visualization and Computer Animation. 1991

[WWL07] WOBBROCK, Jacob O. ; WILSON, Andrew D. ; LI, Yang. *Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes*. Proceedings of the ACM Symposium on User Interface Software and Technology. 2007

[ZSAL09] ZUDILOVA-SEINSTRA, Elena ; ADRIAANSEN, Tony ; VAN LIERE, Robert: *Trends in Interactive Visualization*. 2009

[Zü11] ZÜHL, Lukas. *Visualisierung von Laser-Plasma-Simulationen*. 2011

A Befehlsprotokoll

move: *x y z*	Bewegung der geschlossenen rechten Hand in den drei Raumkoordinaten
spread: *amount*	Abstandsänderung der geschlossenen Hände
azimuth: *amount*	entgegengesetzte Vor- und Rückwärtsbewegung der geschlossenen Hände
elevation: *amount*	entgegengesetztes Auf- und Abbewegen der geschlossenen Hände
SwipeUp	schnelle Aufwärtsbewegung der rechten Hand
SwipeDown	schnelle Abwärtsbewegung der rechten Hand
SwipeLeft	schnelle Wischbewegung der rechten Hand nach links
SwipeRight	schnelle Wischbewegung der rechten Hand nach rechts
Push	schnelle Bewegung der rechten Hand vom Körper weg
Sign: N	geschlossene rechte Hand zeichnet ein N
Sign: M	geschlossene rechte Hand zeichnet ein M
Sign: C	geschlossene rechte Hand zeichnet ein C
Sign: L	geschlossene rechte Hand zeichnet ein L
Sign: X	Kreuzen der Arme vor der Brust
Sign: Box	geschlossene rechte Hand zeichnet ein Viereck
Sign: Left	geschlossene rechte Hand zeichnet eine Linie von rechts nach links
Sign: Right	geschlossene rechte Hand zeichnet eine Linie von links nach rechts
Sign: Up	geschlossene rechte Hand zeichnet eine Linie von unten nach oben
Sign: Down	geschlossene rechte Hand zeichnet eine Linie von oben nach unten
Sign: DownLeft	geschl. rechte Hand zeichnet in einem Zug von oben nach unten nach links
Sign: DownRight	geschl. rechte Hand zeichnet in einem Zug von oben nach unten nach rechts
Sign: UpLeft	geschl. rechte Hand zeichnet in einem Zug von unten nach oben nach links
Sign: UpRight	geschl. rechte Hand zeichnet in einem Zug von unten nach oben nach rechts
Sign: RightDown	geschl. rechte Hand zeichnet in einem Zug von links nach rechts nach unten
Sign: RightUp	geschl. rechte Hand zeichnet in einem Zug von links nach rechts nach oben
Sign: LeftDown	geschl. rechte Hand zeichnet in einem Zug von rechts nach links nach unten
Sign: LeftUp	geschl. rechte Hand zeichnet in einem Zug von rechts nach links nach oben
Sign: SpreadHorizontal	geschlossene Hände werden horizontal auseinander bewegt
Sign: SpreadVertical	geschlossene Hände werden vertikal auseinander bewegt
Hand_Left_Closed	linke Hand wurde geschlossen
Hand_Left_Opened	linke Hand wurde geöffnet
Hand_Right_Closed	rechte Hand wurde geschlossen
Hand_Right_Opened	rechte Hand wurde geöffnet

Tabelle A.1: Kommandos mit Beschreibung der vom Tracker erkannten Gesten

Danksagung

An dieser Stelle möchte ich all denen meinen Dank aussprechen, die mich während meiner Belegarbeit betreut und auf jede erdenkliche Weise unterstützt haben. An erster Stelle sind dies mein betreuender Hochschullehrer Prof. Dr. Stefan Gumhold sowie mein unmittelbarer Betreuer Marcel Spehr. Ihnen verdanke ich viele Anregungen und Hinweise um diese Arbeit erfolgreich abzuschließen. Desweiteren möchte ich Dr. Michael Bussmann für die vielen Ideen und aufschlussreichen Gespräche als auch die Möglichkeit, diese interessante Aufgabenstellung am HZDR zu bearbeiten, danken. Gleiches gilt für Nils Schmeißer und viele weitere Kollegen aus der Abteilung für Informationstechnologie, die stets für Fragen und Gespräche offen waren.

Einen besonderen Dank möchte ich Tomáš Skripčák aussprechen. Die Entwicklung des Interaktionsframeworks VRIF und damit des Kinect-Trackers im Rahmen seiner Dissertaion am HZDR hat diese Arbeit erst möglich gemacht. Er stand mir stets mit Rat und Tat zur Seite und hat all die vielen Wünsche zur Umsetzung des Interaktionskonzepts im VRI-Framework implementiert.

Außerdem möchte ich all jenen danken, die mein Projekt während der Entwicklung getestet haben. Dazu gehören nicht nur Mitarbeiter der TU Dresden und des Helmholtz-Zentrum Dresden-Rossendorf, sondern auch Besucher, die zum Tag des offenen Labors in Rossendorf Interesse für meine Arbeit gezeigt haben. Ihre Fragen, ihr positives Feedback aber vorallem ihre Schwierigkeiten im Umgang mit dem System haben wichtige Hinweise zur Verbesserung gegeben.

Zu guter Letzt möchte ich meinen Eltern für ihre immerwährende Unterstützung danken.