Jül-4307

Jülich Supercomputing Centre (JSC)

Entwicklung einer Software zur Bestimmung morphologischer Informationen von Wurzelsystemen mit Virtual Reality Techniken

Sandra Wienke

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

Entwicklung einer Software zur Bestimmung morphologischer Informationen von Wurzelsystemen mit Virtual Reality Techniken

Sandra Wienke

Berichte des Forschungszentrums Jülich; 4307
ISSN 0944-2952
Jülich Supercomputing Centre (JSC))
Jül-4307

Vollständig frei verfügbar im Internet auf dem Jülicher Open Access Server (JUWEL) unter
http://www.fz-juelich.de/zb/juwel

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek, Verlag
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461 61-5220 · Telefax: 02461 61-6103 · e-mail: zb-publikation@fz-juelich.de

Zusammenfassung

Fruchtbare Böden und sauberes Trinkwasser sind weltweit knappe Ressourcen. Zur ressourcenschonenden Boden- und Wassernutzung wird u.a. der Wasserfluss im Boden und die Wasseraufnahme durch Pflanzenwurzeln erforscht. Die unterirdischen Boden-Wurzel-Wechselwirkungen können seit einigen Jahren mit Hilfe von nicht-invasiven 3D Magnetresonanztomographie-Messungen (MRT) untersucht werden. Außerdem wurde kürzlich ein dreidimensionales Simulationsmodell zur Beschreibung der Bodenwasseraufnahme entwickelt. Dieses basiert auf hierarchisch strukturierten morphologischen Informationen von Wurzelsystemen, die bisher jedoch nur durch Wachstumssimulationen erzeugt werden konnten. Um nun real gemessene Wurzelstrukturen mit dem Simulationsprogramm zu verknüpfen, müssen die entsprechenden morphologischen Daten aus dem MRT-Datensatz strukturiert extrahiert werden. Diese Aufgabe übernimmt die im Rahmen dieser Arbeit entwickelte Software mit Hilfe von Virtual Reality (VR) Techniken.
Die entwickelte Software visualisiert den MRT-Datensatz der Wurzeln im dreidimensionalen Raum mittels stereoskopischer Projektion. Zusätzlich stellt sie dem Benutzer Funktionalitäten zur Verfügung, um das dargestellte Wurzelsystem manuell und interaktiv zu rekonstruieren. Die Interaktion im Raum wird durch optisches Tracking, das die Bewegungen von Kopf und 3D Maus wahrnimmt, realisiert. Aus einer vollständigen Wurzelrekonstruktion können alle für die Simulation benötigten morphologischen Daten bestimmt werden. In dieser Arbeit werden Rekonstruktionsschritte und entsprechende Simulationsergebnisse der Bodenwasseraufnahme beispielhaft anhand der Nutzpflanze Lupine diskutiert.

Abstract

Fertile soils and clean ground- and drinking water are becoming worldwide scarce resources. Research on soil water transport and water uptake by plant roots may lead to an improved sustainable use of soil and water. Below ground soil-root-interactions can be investigated by noninvasive 3D magnet resonance imaging (MRI) since a couple of years. Furthermore, modeling approaches have recently been developed in order to predict the root water uptake distribution based on a hierarchically structured root architecture. Until now, this morphological information could however only be generated by root growth simulation programs. To link the measured MRI root systems to the modeling software, the morphological data must be extracted in a hierarchically structured way from the MRI datasets. This task is achieved by the software, which was developed within this thesis, by dint of Virtual Reality (VR) techniques.
This software visualises the MRI dataset of the roots in the three-dimensional space by using stereoscopic projections. Additionally, it provides features to reconstruct manually and interactively the visualised root skeleton. Interaction in space is supported by an optical tracker which observes movements of the user's head and 3D input device. A fully reconstructed root system makes the necessary morphological information available for the simulation program. Reconstruction steps and corresponding simulation results are discussed in this thesis based on the root skeleton of the plant lupine.

Inhaltsverzeichnis

1 Einleitung **1**

2 Bodenwasseraufnahme durch Wurzelsysteme **5**
2.1 Grundlagen 5
2.1.1 Wurzelsysteme 6
2.1.2 Boden 8
2.1.3 Boden-Pflanze-Atmosphäre-Kontinuum 11
2.2 3D tomographische Messungen 14
2.2.1 Messtechnik MRT 14
2.2.2 Boden-Wurzel-Probe 18
2.3 Simulationen mit R-SWMS 19
2.3.1 Modellierung der Boden-Pflanze-Wechselwirkung 19
2.3.2 Parametrisierung durch Eingabedateien 20
2.3.3 Visualisierung der Simulationsergebnisse 22

3 Virtual Reality Techniken **23**
3.1 Hardware 25
3.1.1 Visualisierungssystem PI-casso 25
3.1.2 Eingabegeräte 30
3.2 Software 32
3.2.1 OpenGL 32
3.2.2 OpenSG 33
3.2.3 ViSTA 34

4 Software-Implementierung **41**
4.1 Zielsetzung und Problematik 41
4.2 Konfiguration der Hardwarekomponenten 43
4.2.1 Monoskopische Darstellung 44
4.2.2 Stereoskopische Darstellung 44
4.3 Visualisierung der Volumendatensätze 49
4.3.1 Umwandlung ins VTK-Legacy-Format 50
4.3.2 Volume Rendering 51

4.4 Rekonstruktion der Wurzelstrukturen . . . 56
4.4.1 Datenstrukturen und Attribute . . . 57
4.4.2 Verarbeitung der eingehenden Sensorwerte . . . 58
4.4.3 Zeichenroutinen . . . 59
4.4.4 Erzeugung und Manipulation der Wurzelstruktur . . . 60
4.4.5 Steuerung durch Aufrufargumente . . . 64
4.4.6 Speichern und Laden . . . 65
4.5 Extraktion der morphologischen Informationen . . . 67
4.5.1 Voraussetzungen an die Wurzelnachbildung . . . 67
4.5.2 Ausgabedatei im R-SWMS-Format . . . 68
4.6 Einschränkungen . . . 69

5 Evaluation am Anwendungsbeispiel Lupine **71**
5.1 Lupinen in der Landwirtschaft . . . 71
5.2 Rekonstruktion der Lupine . . . 72
5.3 Simulationsparameter . . . 74
5.4 Simulationsergebnisse . . . 74

6 Zusammenfassung und Ausblick **77**

Literaturverzeichnis **81**

A Softwarebedienung **85**
A.1 Aufrufargumente . . . 85
A.2 Manipulationsaktionen . . . 86

B Konfiguration der VR-Hardware **87**
B.1 XML-Interaktionsdateien . . . 87
B.2 System-Konfiguration . . . 96

C Rekonstruktion der Lupine **101**
C.1 Datensatz im VTK-Legacy-Format . . . 101
C.2 Wurzelsystem im internen XML-Format . . . 101
C.3 Wurzelsystem im R-SMWS-Format . . . 102

Kapitel 1

Einleitung

Fruchtbare Böden und sauberes Trinkwasser sind weltweit knappe Ressourcen. In Deutschland dient zur Trinkwasserversorgung hauptsächlich das Grundwasservorkommen [26]. Dieses wird jedoch durch landwirtschaftliche und industrielle Bodennutzung belastet. Der Boden nimmt für das Grundwasser eine besondere Bedeutung ein, da dieser eine Art Pufferzone bildet, die einen Teil der Schadstoffe filtert. Neben dieser Schutzfunktion, ist der Boden der wichtigste Produktionsfaktor zur Ernährung der Weltbevölkerung [26]. Die grundlagen- und anwendungsorientierte Forschung im Bereich Boden und Grundwasser hilft dabei das Agrarökosystem ressourcenschonend zu nutzen. Ein Teilgebiet dieser Forschungen bezieht sich auf den Wasser- und Stofftransport im oberflächennahen Bodenbereich, der durch die Bodenwasseraufnahme von Pflanzenwurzeln stark beeinflusst wird. Die Wechselwirkungen zwischen Boden und Pflanze wirken sich aber nicht nur auf den direkten Stoff- und Wasserfluss zwischen Boden und Wurzel, sondern auch auf die gesamte Wurzelmorphologie und die Bodeneigenschaften aus [26].
Trotz der Wichtigkeit von Pflanzenwurzeln für den Wasser- und Nährstoffwirkungsgrad, konnten sie lange Zeit nur dürftig erforscht werden. Ein Grund lag hier in der Schwierigkeit, das Wurzelsystem innerhalb des Bodens zu untersuchen. Die unterirdischen Pflanzenwurzeln werden bezeichnenderweise auch oft die „hidden half“ des Ökosystems genannt. Seit einiger Zeit existieren jedoch nicht-invasive, dreidimensionale Messtechniken – wie die Magnetresonanztomographie (MRT) – für Böden und Pflanzen, die ein Wurzelsystem unzerstört auf Daten im dreidimensionalen Raum abbilden können. Desweiteren ermöglichen kürzlich entwickelte Simulationsmodelle, die auf bestimmten morphologischen Informationen der Wurzelstruktur basieren, die Vorhersage der dreidimensionalen Bodenwasseraufnahme von Wurzelsystemen. Diese konnten bisher jedoch nur auf Grundlage von stochastischen Wachstumsmodellen für Wurzelsegmente angewendet werden. Um diese bisher theoretisch eingesetzten Simulationsmodelle zu verifizieren, müssen experimentell gemessene Wasseraufnahmen der Pflanzen mit den Simulationsergebnissen verglichen werden. Für die Erzeugung realitätsnaher Simulationsergebnisse müssen jedoch zunächst die realen Wurzelsystem-Informationen der MRT-Datensätze mit dem Simulationsmodell verbunden werden. Dazu muss die Wurzelarchitektur der Pflanze aus den Volumendaten der MRT-Datensätze, die oft als Graustufenbilder interpretiert werden, extrahiert werden. Im

Gegensatz zum sequentiellen Aufbau des MRT-Datensatzes muss dazu ein Datensatz erzeugt werden, der die Wurzel-Morphologie hierarchisch und logisch strukturiert beschreibt, sodass er als Grundgerüst für die Simulationen dienen kann. Diese Aufgabe übernimmt die im Rahmen dieser Arbeit entwickelte Software mit Hilfe von Virtual Reality (VR) Techniken.

Die entwickelte Software stellt den MRT-Datensatz eines Wurzelsystems zunächst auf einem stereoskopischen Visualisierungssystem dar. Wegen Messungenauigkeiten kann die Visualisierung des Datensatzes jedoch stark verrauscht oder die abgebildeten Wurzelsegmente unterbrochen sein. Diese Unstetigkeiten können von einem automatischen Algorithmus nur schwer oder gar nicht registriert werden. Aus diesem Grund muss der Benutzer selbst die Bestimmung der morphologischen Wurzelinformationen aus den dargestellten Messdaten übernehmen. Die implementierte Software bietet ihm die notwendigen Funktionalitäten, um mit 3D Interaktionsgeräten die Wurzelstruktur manuell zu rekonstruieren. Bei der Rekonstruktion baut der Benutzer interaktiv durch Erzeugung und Manipulation von Knotenelementen eine hierarchische Baum-Datenstruktur auf. Die baumartige Datenstruktur speichert die architektonischen und morphologischen Attribute wie Raumkoordinaten oder Wurzelsegmentlängen. Diese Eigenschaften können entsprechend für das Simulationsprogramm zur Vorhersage der Bodenwasseraufnahme formatiert extrahiert werden und dienen so als Grundlage für diese Modelle.
Das Simulationsprogramm namens *R-SWMS* benötigt bestimmte morphologische Informationen für die Berechnungen von radialen und axialen Wasserflüssen innerhalb der Wurzel, die ebenfalls die wurzelangrenzende Bodenfläche beeinflussen. Der Wasserfluss im Boden ist von dessen Zusammensetzung abhängig. Der Boden wird über seine so genannte Wasserspannungskurve charakterisiert. Die beschreibenden Parameter dieser Kurve gehen ebenfalls in die Wasserflusssimulation ein. Zusammen mit weiteren Eigenschaften des Wurzelsystems und des Bodens können dadurch die Wassergehalte und -potentiale im Boden und schließlich die Bodenwasseraufnahme durch die Pflanzenwurzeln kalkuliert werden. Die Bodenwasseraufnahme einer Pflanze kann jedoch auch über MRT-Messungen ermittelt werden. Bei diesen Experimenten wird der Wassergehalt im Boden mit Hilfe von Kalibrierungsröhren untersucht. Die Kalibrierungsröhren besitzen einen bekannten Wassergehalt, der Rückschlüsse auf den Wassergehalt bzw. die Wassergehaltänderung des betrachteten Bodens erlaubt. Die Wassergehaltänderung wird hier ausschließlich durch die Bodenwasseraufnahme der Wurzeln hervorgerufen. Durch einen Vergleich dieser experimentellen Ergebnisse mit den Simulationsresultaten, die beide auf dem selben realen Pflanzenwurzelsystem basieren, soll schließlich eine Evaluation des Simulationsmodells bzw. der darin verwendeten physikalischen Annahmen ermöglicht werden. Die Vorhersage der Bodenwasseraufnahme wird in dieser Arbeit vornehmlich dazu verwendet die entwickelte Software zu beurteilen, indem die entsprechenden Simulationsergebnisse auf Plausibilität des Wasseraufnahmeverhaltens untersucht werden.
Die implementierte Software nutzt zur Erzeugung der für *R-SWMS* benötigten Daten verschiedene Virtual Reality Techniken wie die Stereoskopie zum räumlichen Sehen oder das optische Tracking. Die VR-Hardwaregrundlage bildet dabei ein Visualisierungssystem

vom Typ *PI-casso*. Dieses Visualisierungssystem erzeugt mittels zweier Projektoren und polarisiertem Licht ein stereoskopisches Bild, sodass der Benutzer die virtuelle 3D Welt durch eine Stereobrille immersiv erleben kann. Die zwei Tracking-Kameras des Systems nehmen Bewegungen des Kopfes des Benutzers und des mausähnlichen 3D-Eingabegerätes namens *Gyrostick* wahr. Dadurch kann der Benutzer unmittelbar mit der abgebildeten Welt interagieren, was wiederum die Immersion verstärkt. Die Navigation durch die Szene wird mit einer Art Joystick – dem *Spacenavigator* – umgesetzt, der eine freie Bewegung durch den Raum mittels seiner sechs Freiheitsgrade erlaubt.
Softwaretechnisch wird die Stereoprojektion und das Trackingverhalten durch das VR-Toolkit *ViSTA (Virtual Reality for Scientific Technology Applications)* unterstützt, das Anwendungsprogrammierern Schnittstellen für den Zugriff auf VR-Komponenten zur Verfügung stellt. So werden auch die Sensoren der verwendeten VR-Geräte mit Hilfe von Treiber-Konfigurationsdateien über *ViSTA* kontrolliert. Zusätzlich implementiert das von der RWTH Aachen entwickelte Softwarepaket *ViSTA* einen Szenengraph, der durch *OpenSG* realisiert wird. Zeichenroutinen werden hier durch die Programmierschnittstelle *OpenGL* verwirklicht. Die entwickelten Rekonstruktions-, Manipulations- und Extraktionsmethoden der Wurzelstruktur integrieren folglich auch *ViSTA*-Anweisungen zur Realisierung ihrer Funktionalitäten.

Diese Arbeit legt zunächst in Kapitel 2 die Grundlagen der Bodenwasseraufnahme durch Pflanzenwurzeln dar. Dabei werden die Boden-Wurzel-Wechselwirkungen, ihre dreidimensionale nicht-invasive Messmethode MRT und die Simulationsberechnungen der Bodenwasseraufnahme mit *R-SWMS* behandelt. Kapitel 3 beschäftigt sich mit den verwendeten Virtual Reality Techniken. Hier wird auf das stereoskopische Tracking- und Visualisierungsystem *PI-casso*, den *Gyrostick* und den *Spacenavigator* eingegangen. Ebenfalls wird die Architektur des VR-Toolkits *ViSTA* erläutert. Der zentrale Teil dieser Arbeit beschreibt die manuelle interaktive Rekonstruktion des Wurzelsystems und die Extraktion der morphologischen Informationen. Die dazu implementierten Funktionalitäten und die verwendete Konfiguration des VR-Systems werden in Kapitel 4 vorgestellt. In Kapitel 5 wird ein konkretes Anwendungsbeispiel der entwickelten Software betrachtet. Hier wird die schrittweise Rekonstruktion des Wurzelsystems der Nutzpflanze Lupine dokumentiert und das Resultat zur Simulation ihres Wasseraufnahmeverhaltens genutzt. Mit Hilfe der Simulationsergebnisse wird die entwickelte Software evaluiert. Kapitel 6 schließt die Arbeit mit einer Zusammenfassung und einem kurzen Ausblick über mögliche Erweiterungen der Software ab.

Kapitel 2

Bodenwasseraufnahme durch Wurzelsysteme

Die im Rahmen dieser Arbeit entwickelte Software zur Bestimmung morphologischer Informationen von Wurzelsystemen fand in Kooperation mit dem *Institut für Chemie und Dynamik der Geosphäre*, Teilbereich Agrosphäre (ICG-4), des Forschungszentrums Jülich statt. Das ICG-4 untersucht Stofftransport und Stoffumsätze in Böden und oberflächennahem Grundwasser um die ressourcenschonende Nutzung von Agrarökosystemen zu fördern [26]. Im Besonderen beschäftigt sich die Forschungsgruppe *Boden-Pflanze-Wechselwirkungen von der einzelnen Wurzel bis zum Feldmaßstab* mit neuartigen Versuchs- und Modellierungskonzepten zur Betrachtung der Verteilung des Wassers, der Reaktion von Pflanzen auf Wasserknappheit, oder der Auswirkung der Wasseraufnahme durch die Wurzeln auf die Variabilität des Wasserflusses und der Ausbreitung gelöster Stoffe [26].
In diesem Zusammenhang wurden zur Messung von Boden-Wurzel-Prozessen, deren Grundlagen in Unterkapitel 2.1 beschrieben werden, nicht-invasive Techniken wie Magnetresonanztomographie (MRT) verwendet. Die dreidimensionale MRT-Messtechnik und der Aufbau der Wurzelproben, die zusammen die dieser Arbeit zugrunde liegenden Messdatensätze erzeugten, werden in Unterkapitel 2.2 dargestellt. In Abschnitt 2.3 wird auf das Simulationsprogramm namens *R-SWMS* eingegangen. Dieses führt numerische Simulationen zur Vorhersage der Bodenwasseraufnahme durch Wurzeln auf Grundlage der morphologischen Informationen, die im Rahmen dieser Arbeit ermittelt werden, durch. Die entsprechenden Ergebnisse tragen später zu der Bewertung der Software bei.

2.1 Grundlagen

Die Forschungsgruppe des ICG-4 betrachtet die Wurzelsysteme von Nutzpflanzen wie Mais, Rizinus oder Lupine, die im landwirtschaftlichen Anbau von Interesse sind. Deswegen wurde diese Arbeit anhand von Messdatensätzen, die die Wurzelsysteme der genannten Pflanzen in Sandböden beschreiben, getestet. Die aus dem Datensatz extrahierten morphologischen Eigenschaften dienen zur Untersuchung der Boden-Pflanze-Interaktion.

Um diese Boden-Pflanzen-Zusammenhänge nachvollziehen zu können, werden in diesem Unterkapitel die Grundlagen über Wurzelsysteme, Böden und ihre Wechselwirkungen bereitgestellt.

2.1.1 Wurzelsysteme

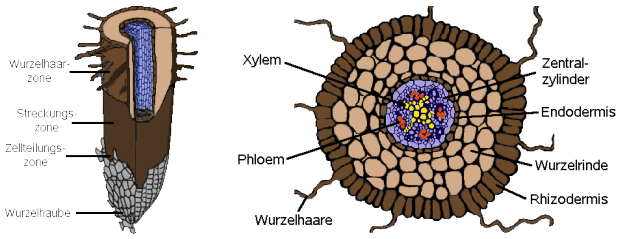
Als Wurzelsystem wird die Gesamtheit der Wurzeln einer Pflanze bezeichnet. Die Wurzel gehört neben Spross und Blatt zu den drei Hauptorganen der höheren Pflanzen. Das Wurzelsystem dient zur Verankerung der Pflanze, zur Speicherung von Wasser und Nährstoffen und zur Wasser- und Nährstoffaufnahme aus dem Boden. In dem Forschungsprojekt, auf dem diese Arbeit basiert, wird die Funktion der Wasseraufnahme der Wurzeln betrachtet.

Entwicklung und Radikation

Die Radikation, das heißt der Bewurzelungstyp, beschreibt die Ausgestaltung des Wurzelsystems. Hier wird zwischen allorhizen und homorhizen Wurzelsystemen unterschieden.
Der Samen einer Pflanze ist die Quelle ihres genetischen Materials. Bei günstigen Bedingungen absorbiert der Samen Wasser und Sauerstoff aus dem Boden und kann keimen. Bei den meisten Pflanzen erscheint dabei die so genannte Keimwurzel zuerst, die den Keimling im Boden verankert. Bei zweikeimblättrigen Pflanzen entwickelt sich normalerweise ein allorhizes Wurzelsystem. Bei allorhizen Wurzelsystemen bildet sich eine Hauptwurzel aus und verzweigt sich. Sie ist die Ausgangsbasis für ein hierarchisch über Seitenwurzeln aufsteigender Ordnung verzweigtes Wurzelsystem. Als Beispiel einer allorhizen Bewurzelung steht die in dieser Arbeit als Datensatz vorliegende Lupine.
Dagegen stirbt die Primärwurzel von Homorhizen meist ab oder bleibt kurz und unfunktionell. Dafür entwickelt sich ein sekundäres Wurzelsystem aus vielen gleichwertigen sprossbürtigen Wurzeln aus. Dieser Bewurzelungstyp tritt meist bei einkeimblättrigen Pflanzen wie bei der Mais-Pflanze auf, deren Wurzelstruktur ebenfalls als Datensatz für die Visualisierung vorliegt.

Anatomie

Die Wurzelachsen von Primär- und Seitenwurzeln können meist in vier Regionen unterschieden werden: die Wurzelhaube, die Zellteilungszone, die Streckungszone und die Zone der ausdifferenzierten Wurzelgewebe (Abbildung 2.1(a)). Hier wird besonders die letzte Region näher betrachtet.
Im Allgemeinen haben die Wurzeln eine zylindrische Form [8] und in ihrem Querschnitt ist ihre typische Anatomie gut erkennbar (vergleiche Abbildung 2.1(b)). Das Zentrum der Wurzel bildet der Zentralzylinder, in dem der Wasser- und Stofftransport zum und vom Spross stattfindet. Im Zentralzylinder existiert dafür ein radiales Leitbündel, das aus Xylem- und Phloemelementen besteht. Das Xylem erscheint im Querschnitt sternförmig und sorgt für den Transport des Wassers und darin gelösten Nährsalzen von den Wurzeln zu den Blättern. In den Buchten zwischen den Sternarmen befinden sich die Phloemelemente, die die in den Blättern durch Photosynthese hergestellten Stoffe und unverbrauchte Nährsalze zu den Speicherzellen transportiert. Der Zentralzylinder wird durch einen Kranz



(a) Längsschnitt (b) Querschnitt einer Wurzel

Abbildung 2.1: Anatomie einer Wurzel [23, modifiziert]

von Zellen – die Endodermis – von der mehrschichtigen Wurzelrinde abgegrenzt. Die Endodermis beinhaltet in ihren Zellwänden den so genannten Caspary-Streifen, mit dessen Hilfe sie den Übergang von Wasser und Nährstoffen zum Zentralzylinder steuert. Die Wurzelrinde – auch Cortex genannt – befestigt die Wurzel und kann Nährstoffe speichern. Die äußerste Schicht bildet die Rhizodermis, die zum Schutz der Wurzel dient und Ansatzpunkt für die Wurzelhaare ist.

Morphologie

In der äußeren Gestalt und auch in der Anatomie unterscheiden sich die allorhizen oder homorhizen Wurzelsysteme nur kaum [8]. Zur Differenzierung der verschiedenen Pflanzen sind die morphologischen Charakteristika der Wurzeln und vor allem das Verzweigungsmuster des Gesamtwurzelsystems einer Pflanzenart von Bedeutung.
Wichtige morphologische Merkmale sind der Durchmesser der Wurzel und die Oberflächenstruktur [8]. Bei der manuellen Rekonstruktion des Wurzelsystems ist dabei im Besonderen der Durchmesser der Wurzel von Interesse. Artspezifisch unterschiedliche Wurzeldurchmesser befinden sich bei krautigen Pflanzen zwischen 30 µm und einigen mm [8]. In der Oberflächenstruktur unterscheiden sich die Wurzeln hauptsächlich durch die Dichte und Länge der Wurzelhaare. Allerdings sind die Wurzelhärchen zu fein, um bei MRT-Messungen registriert zu werden. So werden diese Eigenschaften auf das ausdifferenzierte Wurzelgewebe übertragen. Hierfür extrahiert das im Rahmen dieser Arbeit entstandene Programm neben dem Durchmesser auch die Länge und die Oberfläche jedes Wurzelsegments.
Das eigentliche Muster der Wurzelverzweigungen spezifiziert eine Pflanzenart besser, ist für die Aufnahme von Bodenwasser und Nährstoffen aber nur im Rahmen der morphologischen Eigenschaften interessant. Abbildung 2.2 zeigt die Silhouetten von verschiedenen Wurzelverzweigungen. Zwischen der starken vertikalen Bodendurchwurzelung und der Exploration in die Fläche nahe der Bodenoberfläche gibt es alle denkbaren Übergänge [8]. Je nach Pflanzenart unterscheiden sich dementsprechend das Volumen des durchwurzelten Bodens, die dabei erreichte Bodentiefe, die Gesamtlänge aller Wurzeln und deren für die Wasser- und Mineralstoffaufnahme verfügbare Oberfläche. Auch die Bodentextur (vergleiche Kapitel 2.1.2) beeinflusst das Wachstum der Wurzeln, indem sie die Ausdehnung der einzelnen Wurzelstränge erleichtert oder erschwert.

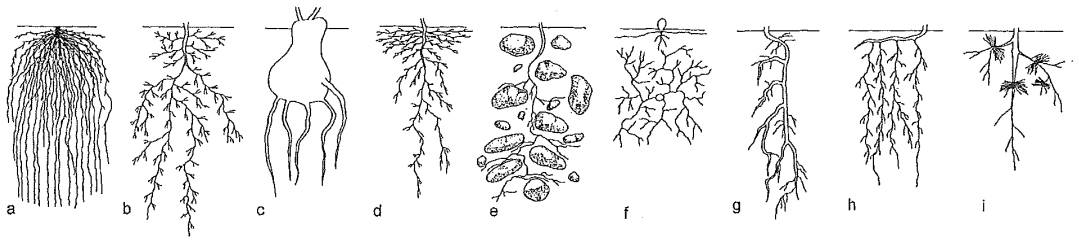


Abbildung 2.2: Typen der Wurzelverzweigungen (a,b: Gräser, c: Knolle, d,e: dikotyle Krautfplanzen, f: Sukkulenten, g,h: rhizombürtige Wurzeln, i: proteoide Bewurzelung) [8]

2.1.2 Boden

Der Boden ist ein komplexes System, das sich in Porenvolumen, Textur und Struktur unterscheiden kann. Zudem kann der Boden je nach Typ einen anderen Wassergehalt, ein anderes Wasserpotential und eine andere hydraulische Leitfähigkeit haben.

Zusammensetzung und Porenraum

Der Boden setzt sich aus vier Hauptkomponenten zusammen: mineralische Partikel, organisches Material, Wasser und Luft. Wie in Abbildung 2.3 verdeutlicht ist, bilden die Mineralpartikel die Hauptkomponente der festen Bodensubstanz (Bodenmatrix) und die tote organische Substanz einen kleinen Anteil. Luft füllt die Hohlräume, die nicht von der wässrigen Bodenlösung eingenommen werden [8]. Der Boden besteht somit aus festen Teilchen und einem flüssigkeits- bzw. gasgefüllten Porensystem, das in Abbildung 2.4 beispielhaft dargestellt ist. Dieser Porenraum kann unterschiedlich voluminös sein und gibt die Porosität des Bodens an [16]. An seiner großen Oberfläche finden Austauschvorgänge statt und dort kann beispielsweise durch Niederschläge zugeführtes Wasser in großen Quantitäten vom Boden festgehalten werden [8]. Die Wasser- und Nährstoffversorgung der Pflanzenwurzeln hängt insgesamt vom Porenvolumen, der Porengröße (Leitungs-, Speicher- oder Restporen) und der Durchlässigkeit des Bodens ab.

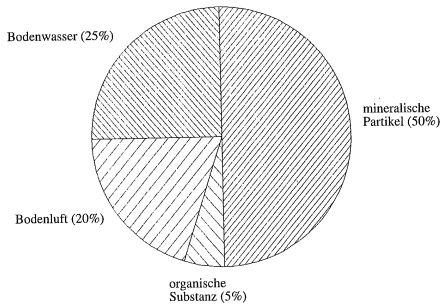


Abbildung 2.3: Bodenzusammensetzung [16]

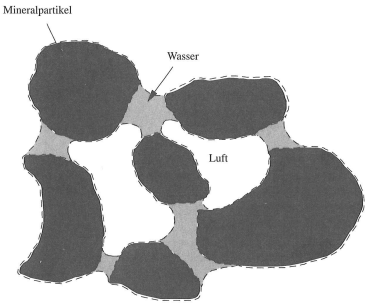


Abbildung 2.4: Mögliches Bodenporensystem [16]

Bodentextur und -struktur

Die mineralische Festsubstanz des Bodens ist ein Gemenge aus Sand, Ton und Schluff. Die Komponenten unterscheiden sich in ihren Korngrößen (von < 2 µm bei Ton bis > 63 µm bei Sand). Ihre Mengenanteile beschreiben die Textur, das heißt die Körnung, des Bodens. Je nach Verteilung der drei Komponenten werden Sand-, Lehm- oder Tonböden sowie Übergangsformen unterschieden. Diese können zwar unterschiedlich gut „bearbeitet“ werden [16], aber die Wasserdurchlässigkeit des Bodens hängt stärker vom Ausmaß des Zusammenschlusses der mineralischen Partikel und der organischen Masse zu Struktureinheiten und Hohlräumen ab. Diese Anordnung von Bodenpartikeln wird durch das Bodengefüge (Bodenstruktur) beschrieben. Ein Einzelkorngefüge besteht zum Beispiel bei einer Sanddüne. Das bedeutet ihre Teilchen sind nicht miteinander verklebt. So ein Boden wird auch als strukturlos bezeichnet. Ein krümeliger Boden bietet meist den Pflanzenkeimlingen eine gute Basis und kann die Pflanzenwurzeln gut mit Wasser und Sauerstoff versorgen [16].

Bodenwassergehalt und -potential

Der Gesamtwassergehalt des Bodens ist meist erheblich größer als das pflanzenverfügbare Bodenwasser [8]. Dies hängt von verschiedenen Faktoren ab. Die Menge des „Haftwassers“, das in den oberen Bodenschichten gegen die Schwerkraft festgehalten wird und somit den Pflanzenwurzeln zur Verfügung steht, wird durch das Ausmaß der kapillaren Bindungskräfte der Bodenporen und der Bodenquellung bestimmt. Im gleichen Maß wird die Menge des „Sickerwassers“, das über die Bodenoberfläche zugeführt wird (wie zum Beispiel Regen), beeinflusst. Inwieweit die Pflanzenwurzeln dann das Haftwasser dem Boden entziehen können, ist wiederum abhängig von der Stärke der adhäsiven Kräfte und damit von den Oberflächenstrukturen der Bodenteilchen und Porenwände [8]. Das Bodenwasser ist folglich nicht völlig frei beweglich und seine Verfügbarkeit für die Pflanzen hängt weniger von der absoluten Bodenwassermenge als vom energetischen Aufwand, mit dem die Kontakte der Wassermoleküle vom Boden gelöst werden müssen, ab. Der Bodenwassergehalt kann in Bezug zum Volumen einer Bodenprobe oder zu ihrer Masse berechnet werden. Dabei wird der Wassergehalt nach Konvention mit Θ angegeben:

$$\Theta_V = \frac{V_{\mathrm{H_2O}}}{V_{\mathrm{Boden}}} \left[\frac{\mathrm{m}^3}{\mathrm{m}^3}\right].$$

Das Bodenwasser lässt sich aber auch energetisch beschreiben, was mit Hilfe des Potentialkonzepts geschieht. Die Kräfte, die auf das Bodenwasser einwirken, verändern seinen Energiezustand, das heißt seine Fähigkeit Arbeit zu leisten. Als Bezugswert wurde reinem Wasser unter bestimmten Standardbedingungen ein Wasserpotential von 0 zugewiesen. Dementsprechend wird stets das Potentialgefälle dazu betrachtet und die Boden- und Wurzelpotentiale liegen im negativen Bereich. Mehrere Kräfte wirken auf das Bodenwasser und tragen zu seinem Gesamtpotential Ψ, das je nach Bodenart unterschiedlich groß sein kann, bei:

$$\Psi = \Psi_m + \Psi_s + \Psi_g + \Psi_p \ [\mathrm{Pa}]\,.$$

Hierbei beschreibt Ψ_p den hydrostatischen Druck auf das Porenwasser und ist meist vernachlässigbar gering [8]. Die Schwerkraft bewirkt das Gravitationspotential Ψ_g und die Osmose das meist vernachlässigbar kleine Osmosepotential Ψ_s. Zusätzlich verursachen Kapillarkräfte das Kapillarpotential Ψ_m, das besser als Matrixpotential bekannt ist und bei Wasserentzug aus dem Boden überwunden werden muss. Das Matrixpotential ist folglich ein Maß für den Einfluss der Bodenmatrix. Diese Komponente macht den Hauptteil des Gesamtpotentials aus [8] und wirkt dem Gravitationspotential entgegen. Der Betrag des Matrixpotentials wird oft als Wasserspannung oder Saugspannung verwendet [25], die in der Literatur auch mit dem Buchstaben h – gemessen in Zentimeter Wassersäule (cmWS) – angegeben wird. Die Abhängigkeit der Wasserspannung vom Wassergehalt stellt die so genannte Wasserspannungskurve (Abbildung 2.5) dar, die eine wichtige bodenphysikalische Eigenschaft ist.

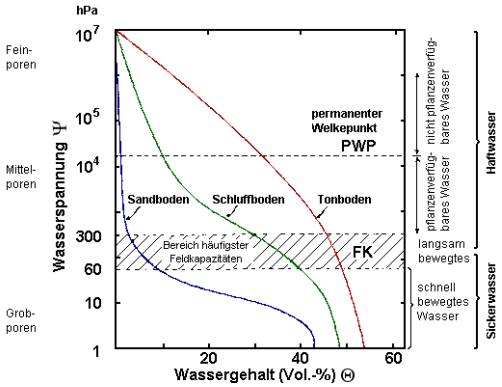


Abbildung 2.5: Wasserspannungskurven für Sand, Schluff und Ton [25, modifiziert]

Anhand ihres Verlaufs kann zum Beispiel die Verteilung der Porengrößen eines Bodens und die Menge des pflanzenverfügbaren Wassers je nach Wassergehalt bestimmt werden.

Hydraulische Leitfähigkeit

Aufgrund von Potentialdifferenzen innerhalb des Bodens befindet sich das Wasser nur selten in einem statischen Gleichgewicht. Das Potentialgefälle hat eine ausgleichende Wasserbewegung vom höheren zum niedrigeren Potential zur Folge. So verlagert sich das Wasser beispielsweise entlang der Schwerkraft von wassergesättigten Bodenbereichen zu solchen mit noch ungesättigten Wasseraufnahmekapazitäten. Dabei spielt die Durchlässigkeit des Bodens eine wichtige Rolle. Diese hydraulische Leitfähigkeit wird durch einen subtratspezifischen Proportionalitätsfaktor [25] angegeben, der folglich von Bodeneigenschaften wie Körnung und Struktur abhängig ist. Dieser Wasserleitfähigkeitskoeffizient k kann auch als Widerstand aufgefasst werden, den der Boden dem Wasserfluss entgegensetzt. Der eindimensionale stationäre hydraulische Wasserfluss im gesättigten Boden kann durch die Gleichung von Darcy quantifiziert werden:

$$q = -k \cdot \frac{d\Psi}{dz}. \tag{2.1}$$

Hierbei beschreibt die Flussdichte q die Wassermenge, die je Zeiteinheit durch einen Fließquerschnitt strömt. Das Wasser fließt entlang des hydraulischen Gradienten $d\Psi$ in z Richtung.
Die in Gleichung (2.1) beschriebene stationäre Strömung gilt nur in bestimmten Bodenbereichen. Für die Wasserbewegung im ungesättigten Bereich spielen jedoch auch zeitabhängige Größen wie Niederschlag, Versickerung und pflanzliche Wasseraufnahme eine große Rolle. Hierzu beschreibt Richards den Zusammenhang zwischen der Veränderung des Wassergehaltes eines Bodenvolumens in einer Zeiteinheit und dem Fluss, der durch einen Potentialgradienten hervorgerufen wird, folgendermaßen [25]:

$$\begin{aligned} \frac{\partial \Theta}{\partial t} &= -\frac{\partial q}{\partial z} \\ &\overset{(2.1)}{=} \frac{\partial}{\partial z}\left[k(\Psi_m)\cdot\left(\frac{\partial \Psi_m}{\partial z} - 1\right)\right]. \end{aligned} \tag{2.2}$$

Hierbei wurde das Gesamtpotential Ψ durch die Summe des Matrix- und Gravitationspotentials angenähert: $\Psi = \Psi_m + \Psi_g = \Psi_m + (h - z)$.
Der durch die Richards-Gleichung gegebene Zusammenhang zwischen Wassergehalt und Wasserpotential wurde von Van Genuchten und Mualem parametrisiert. Die entsprechenden Funktionsparameter können durch nichtlineare Kurvenanpassung der Funktionen an Messdaten oder durch inverse Modellierung von Fließexperimenten bestimmt werden [38].
Das Modell von Van Genuchten beschreibt den Wassergehalt Θ in Abhängigkeit des Wasserpotentials (Wasserspannung) Ψ:

$$\Theta(\Psi) = \Theta_r + \frac{\Theta_s - \Theta_r}{[1 + |\alpha\Psi|^n]^m}. \tag{2.3}$$

Dabei stellt Θ_s den gesättigten Wassergehalt, Θ_r den residualen Wassergehalt und α, m, n die empirischen Parameter dar.
Das Modell von Mualem beschreibt die Porengrößenverteilung, das heißt die Wasserleitfähigkeit:

$$k(\Psi) = k_s S_e^l \left[1 - (1 - S_e^{1/m})^m\right]^2. \tag{2.4}$$

Hier ist k_s die gesättigte hydraulische Leitfähigkeit, $S_e = \frac{\Theta - \Theta_r}{\Theta_s - \Theta_r}$ der effektive Wassergehalt und l ein empirischer Parameter. l beschreibt die Porenkonnektivität und wird für die meisten Böden auf $0,5$ geschätzt.
Auf diesem Genuchten-Mualem-Modell basiert die Bestimmung der Wasserspannungskurve und damit auch die Simulationen zur Bodenwasseraufnahme in *R-SWMS* (Kapitel 2.3).

2.1.3 Boden-Pflanze-Atmosphäre-Kontinuum

Im Boden herrschen negative Wasserpotentiale zwischen wenigen bis einigen 100 KPa [8]. Dagegen besitzt die Luft, das heißt die Atmosphäre, negative Potentiale in der Größenordnung von einigen bis etlichen 100 MPa (vergleiche Abbildung 2.7). Dieser steile Wasserpotentialgradient zwischen Boden und Atmosphäre fördert die Wasserbewegung vom Ort hohen

Potentials zum Ort niedrigeren Potentials: Es entsteht eine Saugspannung. Ein Teil des Potentialausgleichs geschieht über die Evaporation, die zum Weg eines Wasserflusses (Abbildung 2.6) gehört. Die Evaporation beschreibt die energieverbrauchende Überführung des Wassers aus dem flüssigen in den gasförmigen Aggregatzustand.

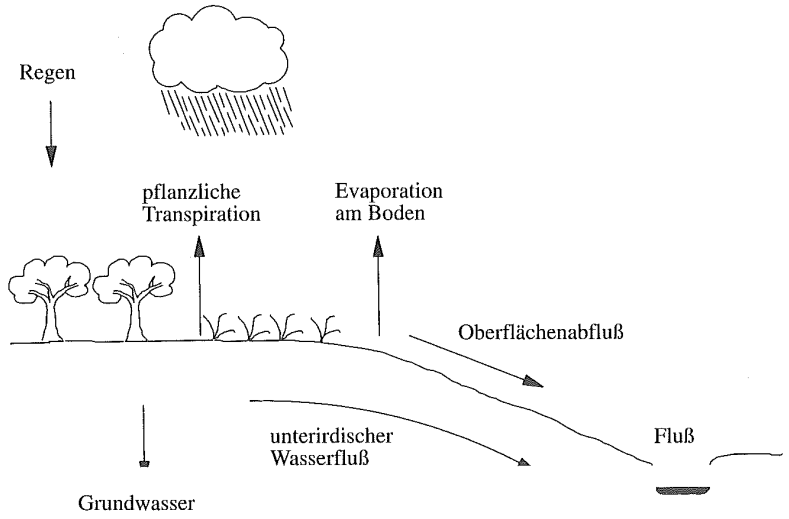


Abbildung 2.6: Wege des Wasserflusses [16]

Zwischen den Potentialen des Bodens und der Atmosphäre liegen die Gewebewasserpotentiale von Pflanzen (0 - 6 MPa). Die Pflanzen setzen dem Wasserstrom wenig Widerstand entgegen [8]. Die kontrollierte Abgabe von Wasserdampf in den Blättern wird Transpiration genannt und dient der Pflanze vor allem zum Schutz vor Überhitzung bei starker Sonneneinstrahlung. Die Transpiration kann einen Transpirationsdruck aufbauen, der das Wasser aus den Wurzeln in die Blätter der Pflanze befördert. Die Pflanze verbraucht von dem von ihr transportierten Wasser nur einen kleinen Anteil und ist für den größten Teil des vom Boden zur Atmosphäre transportierten Wassers nur Transitstrecke [8]. Diese ganze Strecke des Wassers vom Boden durch die Pflanze bis zur Atmosphäre wird als Boden-Pflanze-Atmosphäre-Kontinuum (engl.: soil-plant-atmosphere-continuum, SPAC) bezeichnet.
Der der Saugspannung folgende Wasserstrom passiert Teilstrecken mit unterschiedlichen Leitfähigkeiten. Oft wird statt der Leitfähigkeit der reziproke Wert, das bedeutet der Strömungswiderstand $R = \frac{1}{k(\Psi)}$, betrachtet. Die Widerstände im Transportweg von Wasser im SPAC sind in Abbildung 2.7 veranschaulicht. Im Boden wird der Wasserfluss durch den Nachleitwiderstand gebremst (Abbildung 2.7:A). Der radiale Wurzelwiderstand (Abbildung 2.7:B, C) tritt dem Wasserstrom bei der Aufnahme in die Wurzel und in den Wurzelzentralzylinder entgegen. Der axiale Widerstand beschränkt den Fluss in den Xylem-Leiterbahnen hin zu den Blättern durch Reibung und Gravitation (Abbildung 2.7:D). Beim Austritt des Wassers aus den Blättern (Abbildung 2.7:E,F) und in der Atmosphäre (Abbildung 2.7:G) treten ebenfalls Widerstände auf. Im Folgenden werden kurz die Wi-

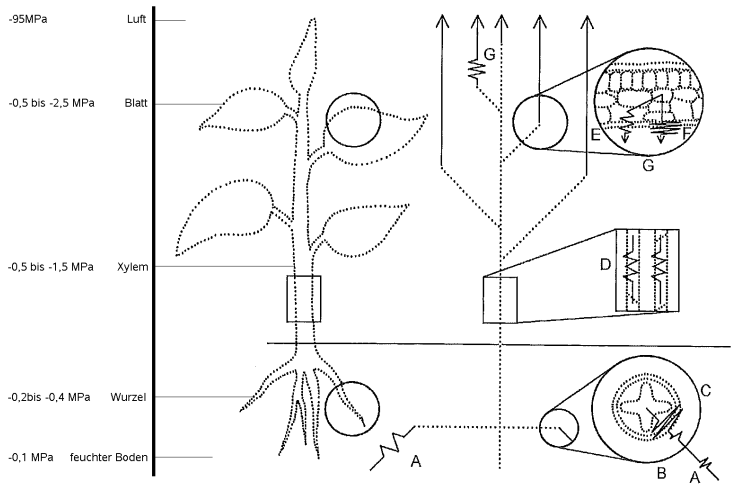


Abbildung 2.7: Potentiale und Widerstände im Wasser-Transportweg im SPAC [8, modifiziert]

derstände A-D erläutert, da diese in das Simulationsprogramm *R-SWMS* (Kapitel 2.3) eingehen. Dabei sind die Widerstände im Boden maßgeblich für die Menge der Wasseraufnahme gegenüber den kleinen radialen und axialen Widerständen.

Widerstände im Boden

Im Boden setzen sich dem Wasserstrom Nachleitwiderstände entgegen (Abbildung 2.7:A). Diese entstehen durch die Wasserentnahme, die beispielsweise im Nahbereich der Wurzeln geschieht, und hat die Anziehung von Wasser aus feuchteren Bodenbezirken zur Folge. Diese Wassernachleitung erfolgt kapillar. Mit zunehmender Ausschöpfung des Wassers im Porensystem nimmt der Nachleitwiderstand stark zu. Der Nachleitwiderstand ist dabei von der Zusammensetzung des Bodens und damit auch von seiner hydraulischen Leitfähigkeit abhängig.

Radiale Widerstände

Radiale Widerstände treten bei der Aufnahme von Wasser in die Wurzelrinde (Abbildung 2.7:B) und beim Übergang in die Endodermis-Passage (Abbildung 2.7:C) auf.
Der Übergang des Wassers vom Boden in die Wurzeln wird hauptsächlich durch den Transpirationssog hervorgerufen, aber auch durch die Osmose. Um die mehrschichtige Wurzelrinde zu durchqueren stehen dem Wasser zwei Möglichkeiten zur Verfügung. Zum einen ist ein symplastischer Transport möglich, wobei das Wasser durch das Innere der Zellen gelangt. Denn ist das Wasser einmal in einer Zelle angekommen, kann es mit seinen gelösten Stoffen leicht von Zelle zu Zelle diffundieren, da die Zellen über Kanäle miteinander verbunden sind [23]. Zum anderen kann ein apoplastischer Transport („Zellwandtransport“) stattfinden. Hier bewegen sich die Wassermoleküle zwischen den Hohlräumen der

Zellwände bis hin zur Endodermis-Schicht.
Der Caspary-Streifen, der sich in den Zellwänden der Endodermis befindet, verhindert jedoch den Weitertransport des Wassers, um die Pflanze vor unkontrolliertem Wasserzufluss zu schützen. Dies hat zur Folge, dass das Wasser den Apoplastenbereich verlassen und in den Symplastenbereich eintreten muss. So gelangt es schließlich in die Xylem-Leiterbahnen des Zentralzylinders.

Axiale Widerstände

Durch das Xylem kann das Wasser von den Wurzeln bis hin zu den Blättern mit Hilfe des Transpirationssogs transportiert werden. Hier spielen nur Reibungskräfte, die in die Leitfähigkeit des Xylems einfließen, und das Arbeiten gegen die Schwerkraft eine Rolle.

2.2 3D tomographische Messungen

Als Eingabe benötigt die im Rahmen dieser Arbeit erstellte Software Informationen über das zu visualisierende Wurzelsystem. Diese Informationen werden in Form von ASCII-Datensätzen zur Verfügung gestellt. Die Datensätze der verschiedenen Wurzelsysteme wurden mittels der nicht-invasiven Messtechnik der Magnetresonanztomographie (MRT) erstellt. Der Aufbau des verwendeten Magnetresonanztomographen des ICG, seine Funktionsweise und der Aufbau der resultierenden Datensätze werden in Unterkapitel 2.2.1 beschrieben. Die Wurzelproben, die mit Hilfe dieser Messungen zu analysieren sind, werden in Unterkapitel 2.2.2 erläutert.

2.2.1 Messtechnik MRT

Untersuchungen der Bodenwasseraufnahme durch Wurzeln wurden lange Zeit durch die geringe räumliche Auflösung der Messverfahren und die Möglichkeit nur Aussagen über zwei Dimensionen machen zu können eingeschränkt [11]. Zugleich bestand die Schwierigkeit das Wurzelsystem und die Interaktionen mit dem Boden durch den Erdboden hindurch zu analysieren. Heutige nicht-invasive dreidimensionale Messtechniken wie die Neutronenradiographie, die Röntgentomographie und die Magnetresonanztomographie beseitigen diese Probleme. So wird auch für die Erforschung der Boden-Pflanze-Wechselwirkungen MRT als bildgebendes Verfahren eingesetzt.
Magnetresonanztomographen machen sich die kernmagnetische Resonanz zu eigen und werden meist nach Feldstärke der verwendeten Magnete (gemessen in Tesla (T)) klassifiziert. Sie wurden hauptsächlich durch ihren Einsatz in der Medizin bekannt. Seit Mitte der 80er Jahre wurden die ersten klinischen MRT-Systeme bei relativ geringen Feldstärken (0,35 T und 0,5 T) verwendet. Danach folgten Geräte mit etwas höheren Feldstärken (1,0 T - 1,5 T), die in den vergangenen 20 Jahren den Hauptanteil der klinisch eingesetzten Systeme stellten [36]. Seit einigen Jahren werden zunehmend weiterentwickelte Hochfeldscanner mit Feldstärken von 3 T angeschafft, da sich die Steigerung der Magnetfeldstärke durch ein besseres Signal-Rausch-Verhältnis unmittelbar auf die Qualität der Signalstärke der gemessenen Daten auswirkt. Auch Ultrahochfeld-Systeme werden für

den Einsatz in der Humanmedizin getestet, werden bisher jedoch kaum verwendet. Eine Schwierigkeit liegt dabei in der Größe der Öffnungen, die für den Menschen benötigt werden, da dadurch schnell inhomogene Felder entstehen. Ebenfalls muss der Einfluss von sehr hohen magnetischen Feldstärken auf den Menschen noch weiter untersucht werden.
Die in der Medizin eingesetzten Tomographen können zwar auch für Untersuchungen von Boden-Pflanzen-Interaktionen verwendet werden, aber aufgrund anderer Gewebeeigenschaften des Boden-Pflanzen-Bereichs gegenüber Menschen, wie zum Beispiel höhere Wasseranteile, ergaben die üblichen Magnetfeldstärken nur stark verrauschte Wurzelbilder. Deswegen werden seit einiger Zeit MRT-Scanner mit sehr hohen Feldstärken und kleinen Öffnungen für Boden- oder Pflanzenproben in diesem Bereich eingesetzt.

Magnetresonanztomograph des ICG

Dem ICG des Forschungszentrums Jülich steht seit zwei Jahren ein eigenes MRT-System der Firma *Varian* zur Verfügung (Abbildung 2.8). Dieser Magnetresonanztomograph misst mit 4,7 T und hat einen Durchmesser und eine Höhe von je ca. 1,5 m. Er kann Proben mit

Abbildung 2.8: Magnetresonanztomograph des ICG

einem Durchmesser von maximal 17 cm scannen, wobei die Höhe des effektiven Messfeldes 20 cm beträgt. Aufgrund des Hochbaus des Scanners können allerdings auch Probensäulen bis zu einer Länge von 2 m durch den MR-Tomographen geschoben werden.
Der Zylinder des Tomographen ist schematisch in drei Schichten unterteilt, die in Abbildung 2.9 dargestellt sind. Die innerste Schicht, die sich an die Öffnung für die Probe anschließt, bilden die Gradientenspulen, die zur Modellierung des Magnetfelds in drei Dimensionen dienen. Darauf folgt der röhrenförmige Heliumtank, der die Hochfrequenzspulen mit supraleitender Magnetwirkung enthält. Damit sich das System nicht zu stark erhitzt, wird es mit Hilfe von Stickstoff in der äußersten Schicht gekühlt. Die Funktionsweise eines solchen Magnetresonanztomographen wird im folgenden Unterkapitel erläutert.

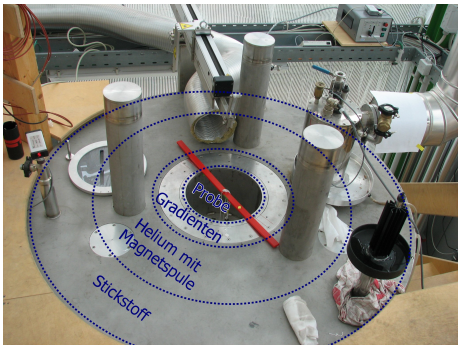


Abbildung 2.9: Schematische Darstellung des Magnetresonanztomographen

Funktionsweise der Magnetresonanztomographie

Die Magnetresonanztomographie, auch Kernspintomographie oder magnetische Kernresonanztomographie genannt, beruht auf Magnetfeldern, die den Kern von bestimmten Atomen – meist Wasserstoffkerne – resonant anregen. Dabei nutzt die MRT-Messtechnik den Eigendrehimpuls (Spin, Abbildung 2.10) dieser Atomkerne aus. Die Spins der Atomkerne besitzen zwei stabile Energieniveaus: den Spin nach oben (engl.: spin up) und den Spin nach unten (engl.: spin down).

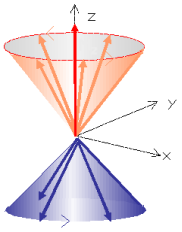


Abbildung 2.10: Spin Up/Down von Atomkernen [13]

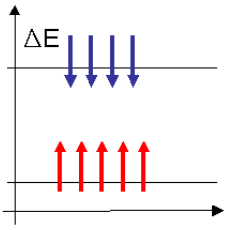


Abbildung 2.11: Stabile Spin-Energieniveaus [13]

Durch Anlegen eines starken statischen Magnetfeldes richten sich die Spins parallel (↑ spin up) oder antiparallel (↓ spin down) aus. Bei diesem thermischen Gleichgewicht kommt es jedoch zu einer leichten Überbesetzung des unteren Energieniveaus (parallele Ausrichtung), wie Abbildung 2.11 verdeutlicht. Diese bewirkt eine makroskopische Magnetisierung in Richtung des statischen Feldes. Mit Hilfe eines zusätzlichen elektromagnetischen Wechselfeldes, erzeugt durch Hochfrequenzspulen, werden die Atomkerne angeregt und so die vorherige Polarisierung aufgehoben. Als Folge der Auslenkung erfolgt eine resonante Präzessionsbewegung des Kernspins, die in einer Spule eine Spannung induziert. Diese Spannung kann gemessen werden und wird als das freie Induktionssignal bezeichnet. Wird das wechselnde Magnetfeld der Hochfrequenzspulen abgeschaltet, geben die Atomkerne nach und nach ihre aufgenommene Energie wieder ab, sodass langsam der Gleichgewichtszustand wieder hergestellt wird. Dieser Vorgang wird Relaxation genannt. Die resultierende Relaxationszeit beinhaltet die Spin-Gitter-Relaxation (T_1-Zeit) und die

Spin-Spin-Relaxation (T_2-Zeit). Die T_2-Zeit bezieht sich auf den freien Induktionszerfall (engl.: free induction decay, FID) und ist meist wesentlich kürzer als die T_1-Zeit. Beide Relaxationszeiten sind charakteristisch für die Gewebeart, wodurch Stoffe unterschieden werden können.
Ein einzelnes Magnetfeld reicht jedoch nicht aus um räumliche Informationen aus den gemessenen Signalen zu bestimmen. Deswegen werden linear ortsabhängige Magnetfelder, das heißt Gradientenfelder mit variabler Feldstärke, eingesetzt um eine Ortskodierung zu erzeugen. Der erste Gradient wird während der hochfrequenten Anregung geschaltet und selektiert eine bestimmte Schicht (Schichtselektion). Der zweite Gradient wirkt quer zum Ersten und wird direkt nach der Anregung eingesetzt. Dadurch erhalten die Atomkerne in jeder Bildzeile unterschiedliche Präzessionsfrequenzen während der Wirkung des Gradienten und damit auch eine ortsabhängige Phasenverdrehung bei der Datenaufnahme (Phasenkodierung). Der dritte Gradient wird während der Datenaufnahme rechtwinklig zu den beiden anderen angelegt. Er sorgt dafür, dass die Signale für jede Bildspalte unterschiedliche Frequenzen liefern. Diese Frequenzen sind proportional zur Ortskoordinate in Gradientrichtung (Frequenzkodierung). Durch eine Frequenzanalyse dieser Signale mittels zweidimensionaler Fourier Transformation kann dann pro Schicht ein 2D-Helligkeitsbild ermittelt werden.

Format der Messdatensätze

Die Gradienten erzeugen Sampling-Punkte (x, y, z) auf einem strukturierten Gitter entlang dreier orthogonaler Koordinatenachsen. Aufgrund der unterschiedlichen Auflösungen und Abmessungen des Pflanzencontainers pro Dimension misst auch der Abtastabstand auf den Achsen unterschiedliche Größen und es entsteht ein anisotropes Gitter. Die diskreten Gitterpunkte werden durch Voxel (Volumenelemente) repräsentiert, denen eine Eigenschaft – hier ein skalarer Wert – zugewiesen wird. Volumendaten werden dementsprechend formal als eine Menge S von Abtastpunkten (x, y, z, v) definiert, wobei der Wert v eine Eigenschaft am 3D-Raumpunkt (x, y, z) angibt. Diese Verhaltensweise kann auch als Funktion $S(x, y, z) = v$ geschrieben werden. Der Wert v wird oft als Helligkeit des Voxels interpretiert. Er kann aber auch die Opazität des Voxels symbolisieren, die die Transparenz bzw. die Lichtundurchlässigkeit des Materials festlegt. Skalare mit dem gleichen Wert werden häufig als Isowerte bezeichnet.
Die Eigenschaft v wird mit Hilfe der zweidimensionalen Fourier Transformation aus den gemessenen Signalen ermittelt und liegt zunächst im „float complex“ Wertebereich, das heißt Fließkommazahlen mit einfacher Genauigkeit für Real- und Imaginärteil mit einer Gesamtgröße von 8 Bytes. Bei den für diese Arbeit relevanten Versuchsreihen des ICG-4 wird entweder nur der Realteil des Messsignals oder der Absolutbetrag der komplexen Zahl betrachtet. Nach dem Messen wird das Datenvolumen mit Hilfe eines Schwellenwerts grob in Boden- und Wurzelbereich geteilt und der entsprechende Bodenanteil durch Setzen des skalaren Attributs auf Null eliminiert. Desweiteren werden die Fließkommazahlen, wenn möglich, auf Bytezahlen reduziert und damit auf den Bereich [0, 255] abgebildet. Diese werden fortlaufend (in x-, y-, z-Reihenfolge) in eine ASCII-Datei geschrieben, die der im Rahmen dieser Arbeit entwickelten Software als zu visualisierender Eingabedatensatz

dient.
Die (Datei-)Größe eines Datensatzes unterscheidet sich folglich anhand der Anzahl der Sampling-Punkte. Bei einer Messpunktzahl von 256 x 256 x 256 (entspricht je 256 Punkte in x-, y- und z-Richtung) wird ein Datensatz, bei dem nur der Realteil verwendet wird, von 64 MB erzeugt. Bei Abspeicherung als Bytezahlen Arrays kann das Datenvolumen auf 16 MB verkleinert werden. Diese Zahlen machen deutlich, dass bei der 3D Visualisierung die Echtzeit-Simulation vor allem von den (großen) Datenmengen abhängt. Deswegen werden für die Visualisierung solcher Daten schnelle Volume Rendering Algorithmen (vergleiche Abschnitt 4.3.2) und Graphikkarten mit schnellen GPUs und großem Speicher benötigt.

2.2.2 Boden-Wurzel-Probe

Das ICG-4 untersucht vor allem die Wurzelsysteme von Nutzpflanzen. Zu den betrachteten Wurzelstrukturen zählen die von Mais, Rizinus und Lupine und es werden Untersuchungen über Gerste und Zuckerrübe folgen. Die bisher beobachteten Systeme werden meistens in zylindrischen, durchsichtigen Pflanzencontainern in der Größenordnung von maximal 10 cm x 10 cm angepflanzt (siehe Abbildung 2.12). Als Boden dient dabei oft ein Sandboden, da dieser gute Eigenschaften für die tomographischen Messungen besitzt. So sind seine langen Relaxationszeiten mit dem MR-Tomographen gut messbar. Wurde die einige Wochen alte Probe im Tomographen platziert, können mit Hilfe einer Lampe und eines Belüftungssystems noch bestimmte klimatische Eigenschaften simuliert werden. Die Auflösung, mit der der Magnetresonanztomograph die Boden-Wurzel-Probe abtastet, ist von der Größe und Struktur der Wurzeln abhängig. Eine Auflösung von 128 Messpunkten auf je 10 cm ist meist schon nicht ausreichend um ein detailliertes Volumen zu erhalten. So wird hier zumeist eine Messpunktzahl von 256 pro Dimension verwendet.

Abbildung 2.12: Rizinus [12] und Lupine

Für diese Arbeit lagen je ein Messdatensatz des Rizinus, des Mais und der Lupine als Testobjekt vor. Der Rizinus-Datensatz wurde mit einer Auflösung von 128 x 128 x 128 Messpunkten auf 10 cm x 10 cm x 10 cm aufgenommen. Abbildung 2.13 zeigt das mit der entwickelten Software visualisierte Wurzelsystem des Rizinus. Dort sind neben der Wurzelstruktur auch zwei Kalibrierungsröhren zu sehen. Diese Röhren enthalten eine Bodenprobe mit einem vorgegebenen Wassergehalt (hier 39 %) und werden zur Ermittelung aller Wassergehalte im Datensatz verwendet. Seine Darstellung ist jedoch stark verrauscht. Beim Mais-Datensatz wurde die Messpunktanzahl bei gleichem Volumen auf

256 x 256 x 256 erhöht. In Abbildung 2.14 sind ebenfalls die zwei Kalibrierungsröhren erkennbar, sowie eine Reduzierung der Rauschanteile gegenüber dem Rizinus-Datensatz. Die 10 cm x 8 cm x 5,5 cm große Probe des Wurzelsystems der Lupine wurde 256 x 256 x 110 mal abgetastet. Der Datensatz wird in Abbildung 2.15 dargestellt.

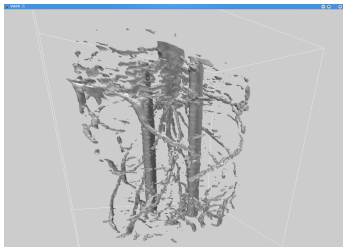


Abbildung 2.13: Rizinus

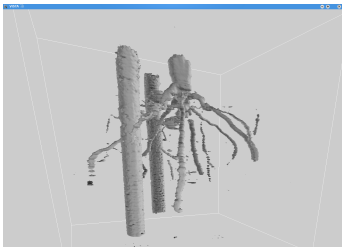


Abbildung 2.14: Mais

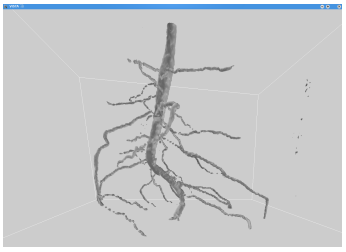


Abbildung 2.15: Lupine

2.3 Simulationen mit R-SWMS

Mit der Möglichkeit nicht-invasive dreidimensionale Messungen von Wurzelstrukturen durchführen zu können und mit neuen Fortschritten in der Pflanzenbiologie wurde der Weg für Untersuchungen der Wurzelwasseraufnahme in Bezug auf die dreidimensionale Wurzelarchitektur und Veränderungen im Boden geebnet [6]. Ein Modell, das die 3D Wurzeldaten und Wasserverteilungen zur Vorhersage der Bodenwasseraufnahme berücksichtigt, wurde 2005 vom ICG-4 entwickelt. Diese Software namens *R-SWMS* simuliert den Wasserfluss in Pflanzen und heterogenen, ungesättigten Böden [27] mit Hilfe einer Spannungsfunktion für die Wasseraufnahme. Diese Spannungsfunktion koppelt die Modelle von Somma et al. (1998) und Doussan et al. (1998) [6]. Das *R-SWMS*-Modell stellt den Boden und die Wurzelarchitektur als eine Reihe von im Raum miteinander verbundenen Knoten dar. Für jeden Wurzelknoten kann der entsprechende Wasserstrom anhand der Wasserpotentialdifferenzen bestimmt werden [27].
Eine Simulation der Wasseraufnahme der Lupine mit *R-SWMS* dient zur Plausibilitätsprüfung der Ergebnisse der im Rahmen dieser Arbeit entstandenen Software. Deswegen wird in den folgenden Unterkapiteln auf die Arbeitsweise des Programms *R-SWMS* eingegangen. Im Einzelnen sind das seine Modellierung der Boden-Pflanze-Wechselwirkung (Abschnitt 2.3.1), seine Eingabedateien (vergleiche Kapitel 2.3.2) und Routinen zur graphischen Darstellung seiner Simulationsergebnisse (siehe Abschnitt 2.3.3).

2.3.1 Modellierung der Boden-Pflanze-Wechselwirkung

Die Berechnungen des *R-SWMS*-Simulationsmodells basieren auf der dreidimensionalen Richards-Gleichung (2.2) unter Berücksichtigung eines Senkenterms S, der beispielsweise Niederschlag, Evaporation und vor allem die Wasseraufnahme durch die Wurzeln repräsentiert:

$$\frac{\partial \Theta}{\partial t} = \nabla \cdot [k \nabla (h - z)] - S, \tag{2.5}$$

wobei Θ der Volumenwassergehalt, t die Zeit, k der Tensor der hydraulischen Leitfähigkeit, h das Wasserpotential, S der Senkterm und z die vertikale Koordinate sind.
Das zugrunde liegende Modell von Doussan bestimmt die Druckhöhe und die Verteilung der Wurzelwasseraufnahme in dem Xylem einer Pflanze unter der Annahme bestimmter Rahmenbedingungen in Bezug auf den Wurzelhals und die Verteilung des Bodenwasserpotentials [27]. Der Wasserfluss zwischen Boden und Wurzel bzw. innerhalb der Wurzelrinde (radial) und des Wurzelxylems (axial) wird mittels der Diskretisierung des Wurzelsystems als Netzwerk von verbundenen Knoten gelöst. Dazu wird ein radialer Fluss J_r und ein axialer Fluss J_x definiert:

$$J_r = k_r s_r [h_s(z) - h_x(z)], \tag{2.6}$$

$$J_x = -k_x \left[\frac{\Delta h_x}{l} + \frac{\Delta z}{l} \right]. \tag{2.7}$$

Hier sind h_s und h_x die Wasserpotentiale an der Wurzeloberfläche und im Xylem. k_r und k_x sind die radialen bzw. axialen Leitfähigkeitsfaktoren. s_r beschreibt die Oberfläche und l ist die Länge des Wurzelsegments. Die Gleichungen (2.6) und (2.7) basieren auf der Annahme, dass das osmotische Potential vernachlässigbar ist, und können für jeden Wurzelknoten aufgestellt werden. Randbedingungen, wie ein vorgegebener Gesamtfluss in $\left[\frac{\mathrm{cm}^3}{\mathrm{Tag}}\right]$ am Wurzelhals der Pflanze, wirken sich auf den radialen und axialen Fluss aus. So kann mit ihrer Hilfe und durch Lösung des Systems von linearen Gleichungen, das Wasserpotential in den Wurzeln ermittelt werden.
Das System dieser Flussgleichungen ist mit dem Wasserfluss im Boden (3D-Richards-Gleichung (2.5)) über die radialen Boden-Wurzel-Strömungen gekoppelt. Die Aufnahme des Wassers von dem Boden in die Wurzel, die durch den Senkterm der 3D-Richards-Gleichung beschrieben wird, kann also aus der Summe der radialen Flüsse des im Bodenvoxel j gelegenen Wurzelsegments i berechnet werden [6]:

$$S_j = \frac{\sum_i^{n_i} J_{r,i}}{V_j}, \tag{2.8}$$

wobei V_j das Voxelvolumen und n_i die Anzahl der Wurzelsegmente im Voxel j ist.
Insgesamt wird das Wurzelmodell zuerst basierend auf den Anfangswerten für die Bodenwasserpotentiale (h_s in (2.6)) gelöst. Daraufhin wird der Senkterm (2.8) in jedem Voxel ermittelt und in die 3D-Richards-Gleichung (2.5) eingesetzt, um den Bodenwasserfluss zu erhalten. Die 3D-Richards-Gleichung wird mit Finite Elemente Methoden (FEM) gelöst, weswegen der Boden auch über Gitterpunkte definiert wurde. Die Iterationen werden so lange durchgeführt bis eine Änderung des Bodenwasserpotentials und des Wassergehalts gegen einen Schwellenwert konvergieren.

2.3.2 Parametrisierung durch Eingabedateien

Solche komplexen dreidimensionalen Modelle benötigen allerdings eine hohe Anzahl von Eingabeparametern. Diese werden bei *R-SWMS* über verschiedene Eingabedateien spezifiziert. Hierbei gibt es Dateien für generelle Parameter wie Einheitendefinition, Iterationskriterien oder Ausgabezeiten und Dateien, die Randbedingungen für den Boden und die

Wurzelstruktur definieren. Ebenfalls können über Eingabedateien die Geometrie der Gitterknoten, das heißt das Simulationsgebiet, und die Eigenschaften des Bodens festgelegt werden. Hier werden u.a. auch die Parameter für das Genuchten-Mualem-Modell ((2.3) und (2.4)) spezifiziert. In einer weiteren Eingabedatei werden die radialen und axialen Leitfähigkeiten der Wurzeln angegeben, die zur Berechnung der entsprechenden Wasserflüsse ((2.6) und (2.7)) benötigt werden.
Für diese Arbeit ist die Datei, in der die eigentliche Wurzelstruktur definiert wird, von besonderem Interesse. Diese Datei enthält die morphologischen Informationen, die der Benutzer mit Hilfe des im Rahmen dieser Arbeit entwickelten Programms aus dem visualisierten Messdatensatz extrahiert. Sie ist folglich das Bindeglied zwischen Experiment und Modell, das bisher nur durch Wachstumssimulationen, die auf Wachstumswahrscheinlichkeiten beruhen, modelliert werden konnte. Die Bestimmung der morphologischen Daten einer realen Wurzelstruktur ist eine Neuerung und Ziel dieser Arbeit.
Der erste Teil der Datei beinhaltet Werte, die das gesamte System beschreiben. Dazu gehören beispielsweise das Alter des Wurzelsystems und seine Anzahl an Ästen und Knoten. Danach folgt eine Liste von einzelnen Wurzelsegmenteinträgen. Für jedes Segment werden seine morphologischen Eigenschaften angegeben. Das sind die Raumkoordinaten des jeweiligen unteren Wurzelknotens des Segments und eine Referenz zum Vaterknoten, über die die hierarchische Struktur definiert wird. Zusätzlich werden seine Verzweigungsordnung, die Länge des Segments, seine Oberfläche, Masse und seine Wachstumszeit [7] spezifiziert. Einige dieser Angaben – wie die Segmentlänge und -oberfläche – gehen beispielsweise direkt in die Wasserflussberechnung innerhalb der Wurzeln ein (vergleiche (2.6) und (2.7)).

```
Time:
    0.00000

Number of seeds
    1

ID, X and Y coordinates of the seeds (one per line)
    0 0.00E+00 0.00E+00

[..]

Total # of branches, including axis(es):
    1

Total # of segment records:
    3

segID#        x           y           z prev or br#     length     surface       mass
origination time
  1  5.5839e-02 -3.9169e-02 -4.4504e-01    0  1   1 4.5024e-01 3.7313e-01 2.4607e-02
  0.0000e+00
  2  5.8379e-02 -1.2797e-01 -8.4575e-01    1  1   1 4.1043e-01 3.8306e-01 2.8375e-02
  0.0000e+00
  3  7.1439e-02 -1.9537e-01 -1.1426e+00    2  1   1 3.0468e-01 3.6606e-01 3.4364e-02
  0.0000e+00
```

Listing 2.1: Beispieldatei im R-SWMS-Format

In Listing 2.1 ist eine Beispieldatei mit einem solchen *R-SWMS*-Format zu sehen. Sie wurde zum Zeitpunkt „0“ erzeugt und das darin beschriebene Wurzelsystem entstand aus einem einzelnen Samen, der an der Position (0, 0, 0) liegt. Das Wurzelsystem besteht aus einem Hauptzweig, der durch vier Knoten, das heißt drei Wurzelsegmenten, aufgebaut wird. Es werden die Eigenschaften der drei Wurzelsegmente aufgelistet.

2.3.3 Visualisierung der Simulationsergebnisse

R-SWMS erstellt während der Simulation verschiedene Ausgabedateien, die die Eigenschaften der Wurzel- und Bodenknoten und die Wasserflüsse enthalten. Diese etwas unübersichtlichen Textdateien können mit Hilfe von einigen *Matlab*-Routinen in einem 3D Gitter an einem festen Zeitpunkt visualisiert werden.

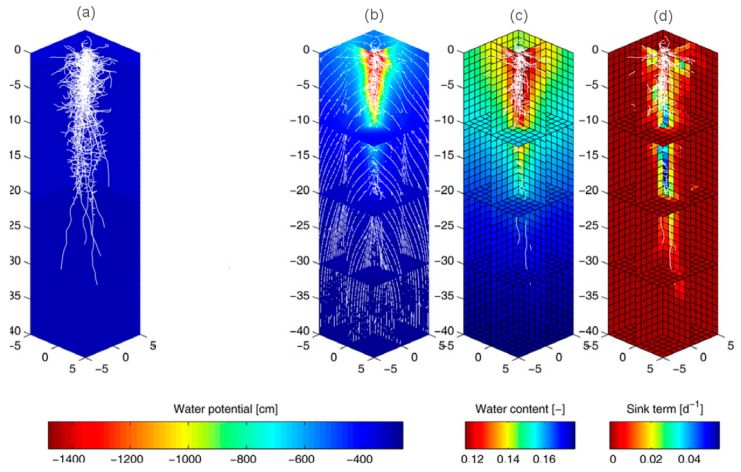


Abbildung 2.16: Simulationsergebnisse mit Matlab [6, modifiziert]

Zunächst kann, wie in Abbildung 2.16(a), das Wurzelsystem (in weiß) abgebildet werden. Diese Art der Darstellung basiert auf der mit der entwickelten Software erzeugten Datei, die die manuell rekonstruierte Wurzelstruktur im *R-SWMS*-Format enthält. Folglich kann so eine erste Prüfung des Datei-Formats und der Wurzelstrukturdaten geschehen. Abbildung 2.16(b) zeigt die Verteilung der Wasserpotentiale durch unterschiedlichen Farben nach simulierten fünf Tagen. Die Wasserflüsse werden hier als Streamlines dargestellt. Diese Abbildung verdeutlicht, dass der Boden ein höheres Wasserpotential als die Wurzeln besitzt und dass das Wasser von Schichten mit hohem Potential zu Schichten mit niedrigerem Potential fließt. Die entsprechenden Wassergehalte und Senkterme (vergleiche (2.5)) sind in den Abbildungen 2.16(c) und 2.16(d) zu sehen. Diese Darstellungen deuten darauf hin, dass Wasser vorzugsweise zuerst aus den oberflächennahen Bodenschichten aufgenommen wird. Dort ist das Bodenwasser leicht zugänglich und der Gradient zwischen dem Boden- und Xylem-Wasserpotential immernoch groß [6].
Mittels ähnlicher Abbildungen wird in Kapitel 5 auch die Anwendung der im Rahmen dieser Arbeit entstandenen Software bewertet.

Kapitel 3

Virtual Reality Techniken

Die Technologie der Virtual Reality oder zu deutsch der Virtuellen Realität (VR) reicht ca. 50 Jahre zurück. So stellte Morton Heilig, der oft als Vater der Virtuellen Realität bezeichnet wird, bereits 1962 einen multisensorischen Simulator – den so genannten *Sensorama* (Abbildung 3.1) – vor, der die Illusion von Realität durch einen 3D Film mit Gerüchen, Stereo-Sound, Vibrationen des Sitzes und Wind-Effekten erzeugte [31]. Angetrieben durch

Abbildung 3.1: Sensorama [31]

das Militär wurde in den 70er Jahren viel Forschungsarbeit in die VR-Entwicklung gesteckt um vor allem durch Software gesteuerte 3D Effekte in Flughelmen und -simulatoren zu erhalten. Auch die NASA förderte die frühe VR-Entwicklung durch Projekte wie zum Beispiel das VIVED-Projekt (Virtual Visual Environment Display), das 1981 gestartet wurde. Anfang der 90er Jahre fanden die ersten internationalen Konferenzen zu diesem Thema statt. Nachdem die VR-Technik bisher sehr kostspielig und dadurch der Markt klein war, führten die Verbesserungen in der Computer-Hardware (schnellere Prozessoren, schnellere Graphik-Beschleuniger, größere und bessere Displays) in den späten 90er die VR zu einem erneuten Aufleben [5]. Heute hat sie in vielen Bereichen wie beispielsweise der wissenschaftlichen Visualisierung, dem Militär, der Medizin, der Autoindustrie und der Unterhaltung Einzug erhalten.

Durch die vielseitige Entwicklung der Virtual Reality haben sich über die Zeit auch verschiedene Definitionen angesammelt, weswegen es schwierig ist **die** Definition hier zu geben. In [5, S. 3] wird Virtual Reality folgendermaßen definiert:

> Virtual reality is a high-end user-computer interface that involves real-time simulation and interactions through multiple sensorial channels. These sensorial modalities are visual, auditory, tactile, smell, and taste.

Laut dieser Definition kann die Virtuelle Realität als eine Simulation aufgefasst werden, die in der Computergrafik dazu benutzt wird eine realistisch aussehende Welt zu erzeugen. Desweiteren ist diese virtuelle Welt nicht statisch, sondern reagiert unmittelbar (in real-time) auf Eingaben des Benutzers (Gesten, verbale Kommandos, etc.). Die virtuelle Welt und die Interaktivität führen dazu, dass der Benutzer das Gefühl hat ein Teil des Geschehens auf dem Bildschirm zu sein und dieses zu erleben: Es entsteht der Eindruck der Immersion.
Neben der Interaktivität und der Immersion hat die virtuelle Realität noch eine dritte Haupteigenschaft: die Imagination. Denn gerade das Ausmaß, in dem VR-Anwendungen zu Problemlösungen beitragen, das heißt inwieweit eine Simulation gut ausgeführt wird, hängt von der menschlichen Fähigkeit ab virtuelle Dinge zu erkennen und zu verstehen. Zusammen werden diese drei Merkmale als die drei I's der VR oder auch I^3 bezeichnet [5].

Die klassische Architektur eines Virtual Reality Systems wird in Abbildung 3.2 dargestellt. Der Benutzer bedient sich eines VR-Systems um ein gegebenes Problem zu lösen.

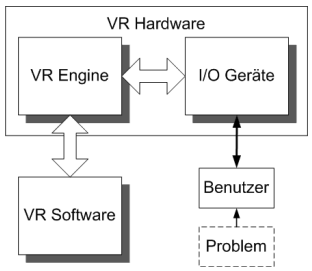


Abbildung 3.2: Architektur eines VR-Systems, nach [5]

Die VR Engine und die Ein-/Ausgabegeräte liefern ihm die Hardware-Basis für die Simulation. Der Begriff *VR Engine* ist eine Abstraktion für verschiedene physikalische Hardware Konfigurationen, die die Echtzeitinteraktionen des Benutzers unterstützen. Dabei ist die Engine für das Einlesen der Daten der Eingabegeräte, für das Zugreifen auf Datenbanken, das Durchführen der für die Aktualisierung der virtuellen Welt benötigten real-time Berechnungen und für das Darstellen der Ergebnisse auf dem Ausgabegerät zuständig. Diese Aufgaben werden hier von einem gewöhnlichen Desktop Computer in Zusammenarbeit mit dem Visualisierungssystem *PI-casso* übernommen, das in Unterkapitel 3.1.1 beschrieben wird. *PI-casso* enthält ebenfalls ein Display als Ausgabegerät und Tracking-Kameras als Eingabegeräte. Weitere wichtige Ein-/Ausgabegeräte sind die (getrackte) Stereobrille, der (getrackte) *Gyrostick* und der *Spacenavigator*, die in Abschnitt 3.1.2 beschrieben werden. Die *VR Software* dient zum Beispiel dazu die Ein-/Ausgabegeräte auf die Simulationsszene abzubilden. Desweiteren modelliert sie Objekte und ihre Attribute. In dieser Arbeit werden vor allem das VR-Toolkit *ViSTA*, *OpenSG* und *OpenGL* verwendet und deswegen in Kapitel 3.2 näher erläutert.

3.1 Hardware

Die im Rahmen dieser Arbeit verwendete spezielle VR-Hardware besteht aus dem Visualisierungssystem *PI-casso* und verschiedenen Eingabegeräten.

3.1.1 Visualisierungssystem PI-casso

Das Konzept des Visualisierungssystems *PI-casso* erweitert einen klassischen Desktop-Arbeitsplatz durch ein kompaktes Virtual Reality System, sodass Daten räumlich visualisiert und unmittelbar bewertet und modifiziert werden können [29]. Das *PI-casso*-System basiert auf der Personal-Immersion (PI)-Technologie des *Fraunhofer IAO* [21] und ermöglicht dem Benutzer in den Raum der Daten „einzudringen“. Als kommerzielles Produkt ist das System unter dem Namen *Flip* des Kooperationspartners *Imsys* erhältlich (Abbildung 3.3).

Abbildung 3.3: Visualisierungssystem der Firma Imsys [28]

Das Visualisierungssystem beinhaltet ein Display mit einer Bilschirmdiagonale von 150 cm. Durch eine Rückprojektion mit zwei Projektoren und polarisiertem Licht können darauf stereoskopische Bilder erzeugt werden. Zusätzlich wird die 3D Interaktion des Benutzers durch optisches Tracking von Kopfbewegungen und Eingabegeräten erlaubt.

Stereoskopie

Das Tiefenempfinden beim Sehen entsteht durch zwei versetzte Bildperspektiven der gleichen Szene (Abbildung 3.4): Die beiden Teilbilder – auch Stereopaar genannt – entstehen durch die Entfernung der Augen voneinander und verschmelzen im Gehirn zu einem einzigen Bild, das dreidimensional interpretiert wird. Anlehnend an die Definition aus [2] kann Stereoskopie so definiert werden:

> Die Stereoskopie ist eine leistungsfähige Methode zur Übermittlung von Tiefeninformationen, sodass eine raumtreue Abbildung entsteht.

Dadurch wird die Stereoskopie zu einer sehr wichtigen Methode zur Erzeugung von Immersion.

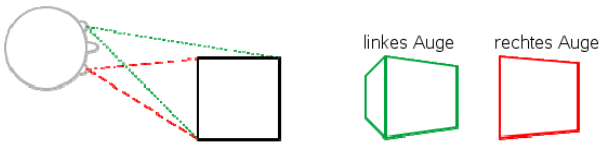


Abbildung 3.4: Stereopaar eines Würfels [20]

Neben der Stereopsis dienen aber auch okulomotorische Tiefenkriterien (Akkommodation, Konvergenz), monokulare Tiefenkriterien (Perspektive, Verdeckung, Beleuchtung, Detailreichtum) und bewegungsinduzierte Tiefenkriterien zur Wahrnehmung von Tiefe [24]. Diese werden aber größtenteils durch die VR Software, die im Kapitel 3.2 erläutert wird, realisiert.

Um die Stereoskopie in einer virtuellen Umgebung umzusetzen, muss das Projektionssystem je ein Bild pro Auge separiert anzeigen. Dieser Sachverhalt wird in Abbildung 3.5 verdeutlicht. Dabei wird bei den Projektionstechniken generell zwischen dem aktiven und dem passiven Multiplexing unterschieden.

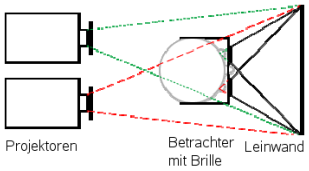


Abbildung 3.5: Stereoskopische Frontprojektion [20]

Beim aktiven oder auch zeitlichen Multiplexing werden die Bilder für das linke und rechte Auge auf einem einzigen Bildschirm schnell (50 - 60 Hz/Bild) hintereinander dargestellt. Der Benutzer muss eine elektrooptische Shutterbrille tragen, die unterschiedliche Shutter für jedes Auge hat und mit den wechselnden Bildschirmbildern synchronisiert ist. So entsteht durch die Speicherung und Mischung der Teilbilder im Gehirn der stereoskopische Effekt [10].
Das passive Multiplexing findet durch Darstellung des Stereopaars durch zwei Projektoren statt. Es kann entweder mit frequenzgetrenntem Licht oder mit polarisiertem Licht erfolgen. Das bekannteste stereoskopische Verfahren ist das Anaglyphen-Verfahren, das mit frequenzgetrenntem Licht funktioniert. Ausnahmsweise wird hierzu allerdings nur ein Projektor benötigt. Bei diesem Verfahren werden die Teilbilder transparent in unterschiedlichen Farben – meist Komplementärfarben (zum Beispiel rot-grün) – dargestellt, die beim Blick durch eine entsprechende farbgefilterte Brille zu einem einzigen monochromen Bild verschmelzen [10]. Eine Weiterentwicklung dieses Verfahrens ist die Infitec-Technik, bei der das für den Menschen sichtbare Farbspektrum unter Beachtung der drei Zapfenarten des Auges in drei mal zwei Bereiche unterteilt wird [20]. Eine entsprechende Infitec-Farbfilterbrille sorgt wieder für die Stereoskopie.
Das Multiplexing mit polarisiertem Licht ist weit verbreitet und findet auch im *PI-casso*-System Anwendung. Um das linke und rechte Bild zu separieren wird dort die lineare

Polarisationstechnik verwendet. Dazu werden orthogonal polarisierte Bilder mit Hilfe von zwei Projektoren, vor deren Objektive sich um 90° versetzte Polfilterfolien befinden, erzeugt. Abbildung 3.6 veranschaulicht die Wirkung dieser Filter. Die projektierten Bilder werden durch eine entsprechend polarisierte Brille betrachtet, sodass das linke Bild nur für das linke Auge sichtbar ist und für das rechte gesperrt ist und umgekehrt (Kanaltrennung). Der große Vorteil dieser Projektionstechnik liegt in der hohen Farbtreue der

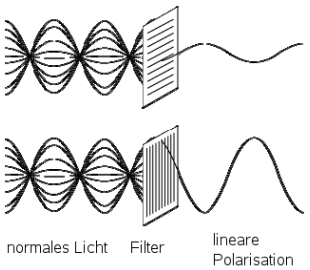


Abbildung 3.6: Lineare Polarisation [20]

dargestellten Bilder ohne den Synchronisationsaufwand betreiben zu müssen wie bei dem zeitlichen Multiplexing. Allerdings müssen dafür zwei Projektoren vorhanden sein und eine besondere (metallbeschichtete) Leinwand um die Polarisation aufrecht zu erhalten. Dies macht die Technik gerade bei kleinen Zuschauerzahlen teuer. Ein weiterer Nachteil ist, dass die Lichtintensität relativ gering ist, da ein Intensitätsverlust bis zu 70% verkommen kann [10]. Außerdem muss der Kopf gerade gehalten werden, sodass möglichst keine Drehungen zu den Polarisationsfiltern stattfinden.
Das im Rahmen der Arbeit verwendete *PI-casso*-System benutzt eine Rückprojektion, wobei zwei DLP-Projektoren (*Digital Light Processing*-Projektoren) mit einer Auflösung von 1400 x 1050 Pixeln [19] zur Erzeugung der stereoskopischen Bilder mit linear polarisiertem Licht dienen. Das Stereopaar kann auf eine Bildschirmfläche von 1,2 m x 0,9 m projektiert werden.

Tracking

Die Definition von Virtual Reality führt Immersion und unmittelbare Interaktivität als Schlüsseleigenschaften an. Folglich benötigt das VR-System bestimmte Hardware um die Bewegungen des Benutzers kontinuierlich zu erfassen und mit den erhaltenen Informationen die Szene der virtuellen Welt entsprechend der Benutzerperspektive anzupassen, wodurch ein Effekt der „physikalischen Immersion“ [15] entsteht. Die Tracker bestimmen dabei zumeist die Position und Orientierung des Kopfes des Benutzers. Beim *PI-casso*-System werden sie mit Hilfe der Stereobrille, auf der ein so genanntes Tracking-Target angebracht ist (näheres in Abschnitt 3.1.2), ermittelt. Ein bewegtes Objekt im dreidimensionalen Raum besitzt sechs Freitheitsgrade (engl.: degrees of freedom, DOF): drei Freiheitsgrade für die Translation (entlang der x-, y-, z-Achse eines Koordinatensystems) und drei für die Rotation (entsprechend der drei jeweiligen Koordinatenachsen), die in Abbildung 3.7 veranschaulicht werden.

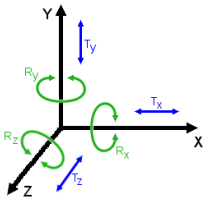


Abbildung 3.7: 6 Freiheitsgrade: Translationen T_x, T_y, T_z, Rotationen R_x, R_y, R_z

Desweiteren ermöglichen getrackte Eingabegeräte die real-time Interaktion mit der virtuellen Welt. So wird ein so genannter *Gyrostick* – eine spezielle 6D Maus (Abschnitt 3.1.2) – auf einen Zeiger im virtuellen Raum abgebildet und erlaubt die Manipulation der dargestellten Objekte. Zusammenfassend liefert [5, S. 17] die folgende Definition für die Tracking-Hardware:

> The special-purpose hardware used in VR to measure the real-time change in a 3D object position and orientation is called tracker.

In diesem Abschnitt wird auf das optische Tracking näher eingegangen. Zudem werden die Komponenten des *PI-casso*-Systems aufgeführt und ihr Funktionalitäten skizziert. Auch die Techniken zur Kalibrierung eines solchen Systems werden kurz aufgezeigt.

Es gibt verschiedene Typen von Tracking-Systemen, die jeweils ihre Vorteile und Restriktionen haben. Generell sollte bei der Wahl eines Trackers neben den Kosten vor allem auf die Messgenauigkeit und -geschwindigkeit, auf mögliche Störmedien (zum Beispiel Metalle oder opake Gegenstände) und auf mögliche Behinderungsgründe (wie zum Beispiel Kabel) geachtet werden. In der Anwendung finden mechanische Tracker, magnetische Tracker, Ultraschalltracker, optische Tracker, videometrische Tracker, Trägheitstracker, Tracker zur neuralen Positionserfassung und hybride Tracker ihren Einsatz.
Das verwendete Tracking-System sollte den Benutzer möglichst wenig in seiner Bewegungsfreiheit des Trackingprozesses einschränken. Deswegen eignen sich besonders gut berührungslose Tracker, das heißt Tracker ohne Kabel oder sonstige Fixierungen. Zu den berührungslosen Trackern gehört das optische Tracking, das vom Visualisierungssystem *PI-casso* eingesetzt wird. Optisches Tracking benutzt visuelle Informationen um die Position des Benutzers oder des Eingabegerätes zu bestimmen. Das hier installierte Verfahren verwendet dabei zwei fix angebrachte Videokameras, die sich wie elektronische Augen verhalten und die getrackte Person oder das getrackte Objekt erfassen [15]. Bei Verwendung von nur einer Kamera kann die beobachtete Position nur in 2D angegeben werden (Position in der Ebene, ohne Tiefeninformation). Bei zwei oder mehr Kameras kann die Tiefeninformation durch geometrische Triangulation berechnet werden. Wird zusätzlich ein Tracking-Target bestehend aus mehreren Markern verwendet, kann auch die Orientierung ermittelt werden und es werden 6 DOF berechnet. Optische Tracker können im Gegensatz zu magnetischen Tracking-Systemen nicht durch Interferenzen durch Metalle o.Ä. beeinflusst werden. Außerdem sind ihre Update-Raten viel höher und die Latenz wesentlich kleiner als die bei Ultraschalltrackern. Auch bieten sie einen relativ großen Arbeitsbereich bezüglich der Bewegungsfreiheit und sind nur durch den Sichtbereich der Kameras

eingeschränkt. Jedoch wird unbedingt eine freie Sichtlinie vom getrackten Objekt zu den Kameras benötigt.
Das optische Tracking-System des *PI-casso*-Systems wurde von der Firma *A.R.T.* entwickelt und besteht aus mehreren Komponenten: einem Tracking-PC, zwei Tracking-Kameras und mehreren Tracking-Targets [18]. Ein typisches *A.R.T.*-System ist in Abbildung 3.8 dargestellt. Der Tracking-PC ist die zentrale Einheit des Systems, denn er ist für die Berechnung der 6 DOF der Targets zuständig und synchronisiert die beiden Tracking Kameras. Die beiden Kameras sind je mit einem rauscharmen CCD-Chip (Charge-Coupled Device), einem FPGA (Field Programmable Gate Array) für die schnelle Analyse der Target-Daten und mit einem internen PC für die 2D Berechnung der Targets ausgestattet. Desweiteren beinhalten sie jeweils ein Infrarotlicht um die getrackten Objekte anzuleuchten. Getrackte Objekte müssen je mit einem Tracking-Target, das auch oft als „rigid body“ bezeichnet wird, ausstaffiert sein. Ein solches Tracking-Target besteht aus mehreren Markern, die meist durch kleine Kugeln repräsentiert werden und in bestimmter Art und Weise zueinander angeordnet sind. Generell ist zwischen aktiven und passiven Markern zu unterscheiden. Aktive Marker senden selbst Licht aus, um erkannt zu werden. Dagegen sind passive Marker Retroreflektoren und reflektieren folglich ankommende Infrarotstrahlung. Letztere werden beim *PI-casso*-System verwendet.

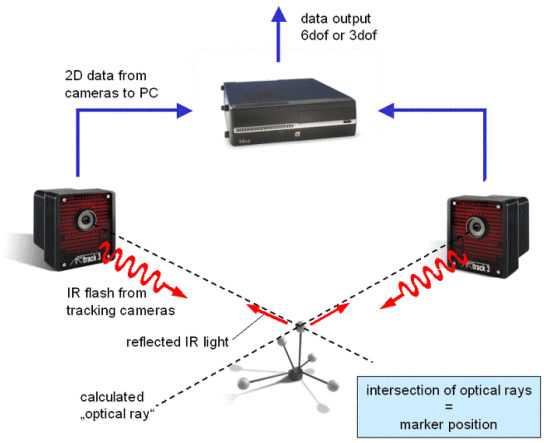


Abbildung 3.8: Typisches A.R.T. System [18]

Das Tracking-System des *PI-casso*-Systems arbeitet folgendermaßen: Die beiden Tracking-Kameras beleuchten mit Hilfe ihres Infrarotlichts das Messvolumen und erzeugen davon Bilder. Die internen Recheneinheiten der Kameras erkennen das Licht, das von den Markern zurückgestrahlt wird und berechnen daraus mit hoher Genauigkeit die Marker-Positionen in 2D Bildkoordinaten [18]. Diese 2D Marker-Koordinaten der beiden Tracking-Kameras werden über Ethernet zu dem zentralen Tracking-PC geschickt. Dort werden die 6 DOF Target-Positionen berechnet und zur Anwendungssoftware, die auf dem Desktop Computer läuft, weitergereicht.
Tracking-Kameras, die gut kalibriert sind, können bei diesem Verfahren eine hohe Genauigkeit, das heißt eine Ungenauigkeit von wenigen mm [39], erzielen. Dabei ist die Einstellung

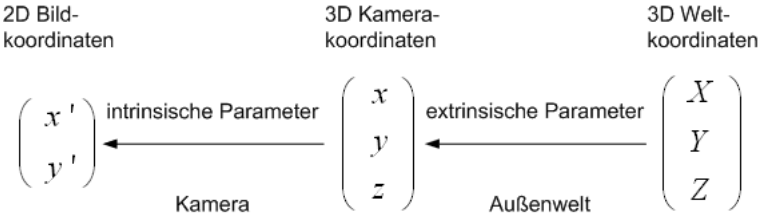


Abbildung 3.9: Parameter für die Kamerakalibrierung

der so genannten intrinsischen und extrinsischen Parameter der Kamera für eine genaue Abbildung der Freiheitsgrade ausschlaggebend (Abbildung 3.9). Die intrinsichen Parameter werden werksseitig eingestellt und definieren die Abbildung zwischen dem 3D Kamerakoordinatensystem und dem 2D Bildkoordinatensystem [39]. Sie beschreiben folglich die innere Geometrie der Kamera. Dabei muss die richtige Skalierung, mit der die unterschiedlichen Einheiten an Bild- und Kamerakoordinatenachsen angepasst werden, gewählt werden. Ebenso muss die Brennweite der Kamera, die Translation zwischen den Ursprüngen der beiden Koordinatensystemen und die Verzerrung des Bildes durch die Linsengeometrie berücksichtigt werden [40]. Desweiteren müssen die extrinsischen Parameter, das heißt die Parameter die die Orientierung der Kamera bezüglich des Weltkoordinatensystems beschreiben, bestimmt werden. So wird die Transformation der 3D Weltkoordinaten in die 3D Kamerakoordinaten ermittelt. Die Parameter werden vom Benutzer des *PI-casso*-Systems selbst durch eine Raumkalibrierung mit Hilfe eines Kalibrierungswinkels und eines Kalibrierungsstabs eingestellt. Außerdem kommt die Körperkalibrierung hinzu, bei der die Geometrie der Marker-Konfiguration kalibriert wird, um später die 3D Anordnungen der (verschiedenen) Marker identifizieren zu können [18].

3.1.2 Eingabegeräte

Neben dem eigentlichen Visualisierungssystem sind auch die Eingabegeräte wichtige Hardware-Komponenten der Virtual Reality. Sie bestimmen maßgeblich den Faktor der Interaktivität. Zu den verwendeten Eingabegeräten gehören die getrackte Stereobrille, der getrackte so genannte *Gyrostick* und der *Spacenavigator*, die in den folgenden Abschnitten beschrieben werden.

Getrackte Stereobrille

Die Stereobrille dient zwar hauptsächlich zum stereoskopischen Sehen, sie ist allerdings ebenfalls mit einem Tracking-Target ausgestattet, sodass die Kopfbewegungen des Benutzers erfasst werden können. Entsprechend diesen Bewegungen wird im verwendeten System die Darstellung des Bildes angepasst. Bewegt der Benutzer beispielsweise seinen Kopf zur Seite, werden auch die Objekte im Bild in ihrer Perspektive verändert, wodurch der Benutzer den Eindruck erhält, das Objekt von einer anderen Position aus zu betrachten und somit interaktiv mit den Objekten zu agieren. Abbildung 3.10 zeigt die Stereobrille mit dem beschriebenen Tracking-Target.

Abbildung 3.10: Getrackte Stereobrille

Getrackter Gyrostick

Das 3D-Eingabegerät der Firma *Gyration*, das im Folgenden als *Gyrostick* bezeichnet wird, ist eine optische Maus, in der ein kleines Gyroskop eingebettet ist. Der *Gyrostick* besitzt neben den gewöhnlichen Maustasten zusätzliche Tasten an der Seite und Unterseite des Körpers (Abbildung 3.11). Sein eingebettetes Gyroskop kann den Winkel und die Geschwindigkeit von Bewegungen messen [22]. Diese Eigenschaft kommt im Rahmen dieser Arbeit jedoch nicht zur Anwendung, weswegen hier nicht näher darauf eingegangen wird. Dagegen ist die Tatsache, dass der *Gyrostick* schnurlos verwendet und mit Hilfe eines Funksenders frei im Raum bewegt werden kann, von besonderem Interesse für die Interaktion.

Abbildung 3.11: Getrackter Gyrostick

Zusätzlich wurde ein Tracking-Target am Kopf der Maus angebracht. Dadurch können die Tracking-Kameras die Position und Orientierung des Gyrosticks ermitteln. Dies wird in dieser Arbeit dazu verwendet einen dreidimensionalen Cursor in Form einer Pfeilspitze an der entsprechenden Position und mit der entsprechenden Orientierung im virtuellen Raum zu zeichnen. Mit seiner Hilfe können dargestellte Objekte erstellt und manipuliert werden, sodass der Benutzer intuitiv interagieren kann.

Spacenavigator

Der *Spacenavigator* der Firma *3Dconnexion* [17] ist eine Art sechsdimensionale Maus. Mit seiner Hilfe kann der Benutzer komfortabel durch eine 3D Welt navigieren, indem der *Spacenavigator* beispielsweise Translationen und Rotationen auf Objekte ausführt. Durch Aktionen wie Bewegungen zur Seite, Schwenkungen nach oben/unten oder Vergrößern und Verkleinern können dementsprechend Objekte verschoben werden. Kippen, Drehen und Rollen symbolisieren dagegen die drei Rotationsmöglichkeiten um die x-, y- oder z-Achse (vergleiche Abbildung 3.12).

Abbildung 3.12: Spacenavigator mit 6 DOF [17]

3.2 Software

VR-Software unterstützt die Verwendung von Virtual Reality Techniken. Mit ihrer Hilfe können beispielsweise Objekte und ihre Attribute in 3D modelliert werden. Die entsprechenden Grafiksoftwaresysteme lassen sich wie viele Softwaresysteme in einem Schichtenmodell darstellen [2]. Dabei befindet sich in der obersten Schicht das Anwendungssys-

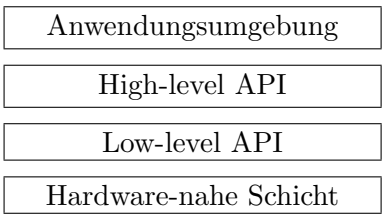


Abbildung 3.13: Schichtenmodell, nach [2]

tem, das hier durch die implementierte Applikation zur Visualisierung und Rekonstruktion der Pflanzenwurzeln vertreten ist. Als eine Art Zwischenschicht zwischen Applikation und der darunter folgenden High-level Application Programming Interfaces (APIs) kann das *ViSTA*-Toolkit aufgefasst werden, das im Abschnitt 3.2.3 erläutert wird. *ViSTA* kann verschiedene Szenengraphen verwenden und setzt in der aktuellen Version auf dem in Abschnitt 3.2.2 beschriebenden *OpenSG* auf. Das Szenengraphkonzept gehört zu den High-level-APIs und über diese Bibliotheken werden zahlreiche Funktionalitäten wie Szenenverwaltung und -darstellung zur Verfügung gestellt. Unterhalb dieser Ebene gibt es die Low-level-APIs, die sich bereits deutlich näher an der Grafikhardware befinden und somit weniger Overhead bezüglich Verwaltung von hierarchischen Szenen und Ähnlichem erzeugen. Im Rahmen der Arbeit wird dafür *OpenGL* verwendet, das sowohl für Windows- als auch für Linux-Systeme einsetzbar ist und in Abschnitt 3.2.1 beschrieben wird. Die unterste Schicht kommuniziert schließlich mit der Hardware. Auf sie wird hier nicht weiter eingegangen.

3.2.1 OpenGL

Die Low-level-Programmierschnittstelle *OpenGL (Open Graphics Library)* wird zur Entwicklung von interaktiver (2D- und) 3D-Grafik benutzt und findet im Toolkit *ViSTA* und vor allem in dem im Rahmen dieser Arbeit erstellten Programm Anwendung. Die *OpenGL*-Schnittstellen sind unabhängig vom konkreten Fenstersystem, in dem die Anwendung später laufen soll [2]. So unterstützen sie die von *ViSTA* geforderte Portabilität (vergleiche Abschnitt 3.2.3). Durch diese Unabhängigkeit müssten allerdings Fenster-Interaktionen mit Maus und Tastatur mit Hilfe des Fenster-Handlings des Betriebsystems eigenständig realisiert werden. Hier bietet jedoch die zusätzliche Bibliothek *GLUT (OpenGL Utitlity Toolkit)* eine normierte Schnittstelle für die Benutzerinteraktionen.
OpenGL ist eine „State-Machine“, das heißt, es werden alle Einstellungen zur Steuerung der Grafikhardware als Zustände verwaltet und gespeichert [2]. Über diese Zustände kann der Anwendungsprogrammierer die in seiner Grafikhardware implementierten Algorithmen parametrisieren.

Das Rendering einer 3D-Szene mit *OpenGL* beruht auf geometrischen Primitiven (wie Punkte, Linien und Polygone) oder Raster-Primitiven (wie Bitmaps oder Pixel-Rechtecke) und verfügt nicht über Wissen über die komplette Szene. Zur Manipulation einer Szene (Translation, Rotation, Skalierung, Scherung) werden Transformationsmatrizen auf die Primitive angewendet. Es existieren verschiedene Transformationsmatrizen zum Verändern der Koordinaten des Modelviews, der Projektion und der Textur. Die Modelview-Matrix bestimmt die Position der Kamera. Die Projektionsmatrix speichert die Projektionstransformation und die Textur-Matrix wird nur auf die in *OpenGL* definierten Texturkoordinaten angewendet. Grundsätzlich werden die Transformationsmatrizen in *OpenGL* auf die zuvor im entsprechenden Stack liegende Matrix aufmultipliziert, womit alle Transformationen in *OpenGL* relativ zu der Vorherigen sind.

3.2.2 OpenSG

OpenSG ist ein open source real-time Rendering-System, das auf einem auf *OpenGL* aufgesetzten Szenengraphen basiert [33].

Szenengraph

Ein Szenengraph speichert eine ganze Szene in einem Graphen, die meist in hierarchischer Struktur als Baum oder DAG (Directed Acyclic Graph) modelliert wird. Eine Szene enthält dabei häufig Objekte, die aus mehreren Teilen bestehen wie beispielsweise ein Fahrrad, das vereinfacht aus einem Rahmen und zwei Rädern zusammengesetzt wird. Wird der Fahrradrahmen nun bewegt, muss neben dieser Translation auch eine Rotation auf die Räder angewendet werden. Die Abbildung 3.14 verdeutlicht dieses Verhalten. Dort ist auch der Szenengraph des beschriebenen Fahrrads dargestellt.

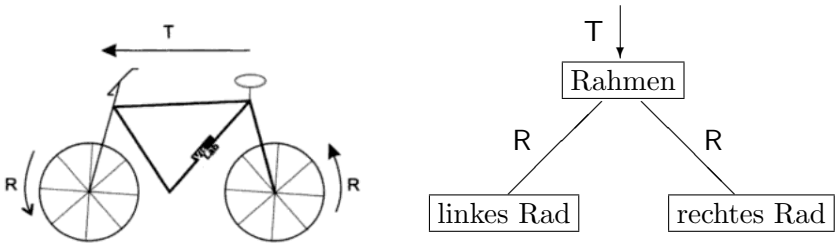


Abbildung 3.14: Fahrrad als ein hierarchisches Objekt mit Szenengraph [2], Bewegung des Fahrrads führt zur Translation T des Rahmens und Rotation R der Räder

In den Knoten des (Szenen-)Graphs wird die entsprechende Geometrie oder abstrakte Eigenschaften wie Textur oder Lichtquellen gespeichert. Die Transformationen, die auf die Geometrie angwendet werden sollen, werden sich entweder zusätzlich in dem Geometrieknoten oder in den Kanten des Szenengraphs gemerkt [2]. Diese Transformationen werden auf alle entsprechenden Kinderknoten rekursiv angewendet.
Wichtige Vorteile von Szenengraphen gegenüber Low-Level-APIs wie *OpenGL* bestehen in der Trennung der Anwendungslogik von den Grafikroutinen und dem Wissen welche

Objekte sich wo in der Szene befinden. Mit diesem Wissen ist es möglich nur für den Betrachter sichtbare Objekte zu zeichnen (zum Beispiel per Culling oder Bounding Volumes), was je nach Szene zu einer großen Performanzsteigerung führen kann [33].

OpenSG bietet folglich eine Bibliothek zur Entwicklung von rechenintensiven grafischen Anwendungen an ohne eine eigene Implementierung eines Szenengraphen durchführen zu müssen [33]. *OpenSG* ist ebenfalls verwendbar mit Multithreading und Clustering. Es arbeitet gut in heterogenen Netzwerken, sodass mehrere Computer mit verschiedenen Grafikkarten die selbe Applikation ausführen können. Diese Tatsache unterstützt die von *ViSTA* angestrebte Portabilität. In *ViSTA* wird beispielsweise die Navigation durch die Szene mit Hilfe des *Spacenavigators* durch angewendete Transformationen auf den Wurzelknoten des Szenengraphens realisiert. An diesen Szenengraphknoten können entsprechend auch die Orientierungen abgegriffen werden, die die im Rahmen dieser Arbeit entstandene Software zur gerichteten Darstellung des 3D-Cursors benötigt.

3.2.3 ViSTA

Das Virtual Reality Toolkit *ViSTA (Virtual Reality for Scientific Technical Applications)* wird seit ca. 10 Jahren von der VR Gruppe der RWTH Aachen entwickelt und ist als eine objektorientierte platformunabhängige Softwarebibliothek in *C++* implementiert. *ViSTA* bietet *C++*-Programmierern Software-Schnittstellen um VR Technologien und interaktive 3D Visualisierung in technische und naturwissenschaftliche Applikationen zu integrieren. Dabei setzt *ViSTA* konzeptionell auf dem Szenengraph auf, der durch *OpenSG* implementiert wird. Der VR Fokus an der RWTH Aachen liegt in der Visualisierung von numerischen Simulationen, virtuellem Prototyping, der Architektur, der Medizin und der Psychologie [14]. Die Heterogenität der Applikationen und besonders der Hardwareplattformen dieser Benutzer benötigen eine hohe Portabilität der Software, die durch Toolkits und Compiler, die auf den meisten Plattformen arbeiten, realisiert ist [14].
Die *ViSTA*-Implementierung verfolgt das Brücken-Konzept (engl.: Bridge-Pattern), das im Abschnitt über den *ViSTA* Kernel kurz erklärt wird. Das Brücken-Konzept bietet den großen Vorteil der Zugriffsabstraktion auf verschiedene externe Toolkits. Allerdings müssen die Anwendungsprogrammierer dadurch auch mit der *ViSTA* API vertraut sein um die entsprechenden Schnittstellen ansprechen zu können.
ViSTA wird als stetige Weiterentwicklung ständig konzeptionell angepasst und verbessert. So können neue Releases als eine Art Momentaufnahme des *ViSTA*-Systems aufgefasst werden, die auch dem Anwendungsprogrammierer die Möglichkeit bieten mit einer stabilen Softwareversion zu arbeiten. Die in dieser Arbeit für die stereoskopische Darstellung verwendete VR-Software *ViSTA* wird in dem momentan aktuellen Release vom 5.12.2008 eingesetzt. Im Folgenden wird ihre Architektur näher beschrieben.

ViSTA Basisbibliotheken

Die *ViSTA* Architektur basiert auf fünf Bibliotheken, die in Abbildung 3.15 veranschaulicht werden. Die Pfeile in der Abbildung verdeutlichen, dass die Applikation die Dienste

von allen Komponenten der Basisschicht direkt verwenden kann und diese nicht durch die Kernelschnittstellen versteckt werden. Zudem kann auf externe Toolkits wie *OpenSG* zugegriffen werden.

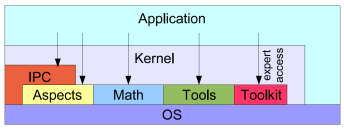


Abbildung 3.15: Architektur der ViSTA Basisbibliotheken [1]

Die *Aspects* Bibliothek beinhaltet abstrakte Schnittstellen zu sehr weitgefassten und unspezifischen Aufgaben des Systems oder der Applikation wie Serialisierung oder Call-Backs [34].
Die *Math* Bibliothek bietet Zugriff auf Implementierungen für mathematische Berechnungen wie Quaternionen oder Transformationsmatrizen.
Die *Tools* Bibliothek besteht aus betriebssystemabhängigen Diensten und Datenstrukturen wie Dateizugriffe oder Zeitstempel.
Die *Inter Process Communication (IPC)* Bibliothek liefert Tools zur Netzwerkkommunikation, Prozesssynchronisation, gemeinsamen Speicherzugriff und Ähnlichem.
Die *Kernel* Bibliothek ist die Schnittstelle für VR Applikationen. Sie wird im nächsten Unterkapitel erläutert.

ViSTA Kernel

Der Kern jeder *ViSTA* Applikation ist der *ViSTA* Kernel (Abbildung 3.16) und die durch ihn implementierte Infrastruktur [34]. Das eventbasierte Design des Kernels beinhaltet vor allem die Kapselung der Szenengraphen API und Abstraktionen für VR-Geräte. Der Kernel wird in vier Komponenten aufgeteilt: dem *Display System*, dem *Geometry System*, dem *Interaction System* und dem *Event System*. Diese Komponenten können größtenteils konfiguriert werden ohne den Quellcode verändern zu müssen, sodass eine hohe Flexibilität entsteht. Dies wird mit Hilfe von ini-Dateien realisiert, in denen viele der Parameter des Kernels spezifiziert werden können. Die vollständigen Konfigurationsdateien sind in Anhang B zu finden.
Das *Display System* definiert den Zugriff auf die physikalische VR-Umgebung als eine Sammlung von Display Systemen. Es steuert somit eine Anzahl von Display Systemen

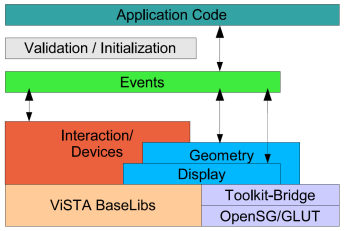


Abbildung 3.16: Layout des ViSTA Kernels [1]

einschließlich Fenstern, Viewports, Projektionen und Szenenüberlagerungen [1]. Jedem Display System ist ein Betrachter zugeordnet, dessen Position und Orientierung die Darstellung auf dem Display beeinflusst und von dem die Anfragen und Änderungen an Fenstern, Viewports und Projektionen abhängen (Abbildung 3.17). Infolgedessen kann nur ein Trackingverhalten pro Display System spezifiziert werden.

Abbildung 3.17: Display System und drei verschiedene Viewports [34]

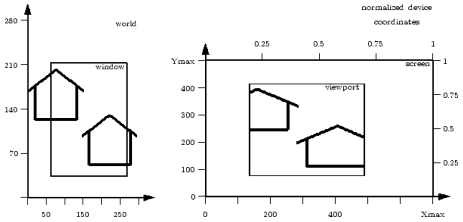


Abbildung 3.18: Fenster und Viewport [32]

Wie in der Abbildung 3.18 dargestellt, ist ein Fenster ein rechteckiger sichtbarer Bereich der Welt, angegeben in Weltkoordinaten. Ein Viewport ist ein Bereich auf dem Bildschirm, in dem der Inhalt des Fensters angezeigt wird und der in Bildkoordinaten definiert ist. Ein Viewport kann nur einem Fenster angehören, aber ein Fenster kann mehrere Viewports haben. Mit Hilfe von Projektionen wird die Fenstersicht auf die Viewports abgebildet, wobei jeder Viewport seine eigene Projektion haben kann.

```
[STEREO]
NAME                =   STEREO
VIEWPORTS           =   STEREO_VIEWPORT
VIEWER_POSITION     =   0, 1, 1
VIEWER_ORIENTATION  =   0, 0, 0, 1
LEFT_EYE_OFFSET     =   -0.03, 0, 0
RIGHT_EYE_OFFSET    =   0.03, 0, 0

[STEREO_VIEWPORT]
NAME            =   STEREO_VIEWPORT
PROJECTION      =   STEREO_PROJECTION
WINDOW          =   STEREO_WINDOW

[STEREO_PROJECTION]
NAME                =   STEREO_PROJECTION
PROJ_PLANE_MIDPOINT =   0, 0, 0
PROJ_PLANE_NORMAL   =   0, 0.258819, 0.965925
PROJ_PLANE_UP       =   0, 0.965925, -0.258819
PROJ_PLANE_EXTENTS  =   -0.6, 0.6, -0.45, 0.45
CLIPPING_RANGE      =   0.1, 65000
STEREO_MODE         =   FULL_STEREO

[STEREO_WINDOW]
NAME            =   STEREO_WINDOW
STEREO          =   true
SIZE            =   1400, 1050
DRAW_BORDER     =   false
FULL_SCREEN     =   true
```

Listing 3.1: Beispiel eines Display-System-Abschnitts

Das Display System wird im `[SYSTEM]` Abschnitt des ini-Files deklariert. Hier wird für diese Arbeit beispielsweise entweder ein monoskopisches (`MONO`) oder stereoskopisches (`STEREO`) Verhalten definiert. Die Konfiguration erfolgt unter dem entsprechenden Definitionsnamen. In Listing 3.1 ist die Konfiguration für das stereoskopische Display System des *PI-casso*-Systems zu sehen. Es werden Viewportnamen, die Position und Orientierung des Betrachters und der Abstand von seinem linken zu seinem rechtem Auge definiert. Der entsprechend passende Abschnitt zum Viewportnamen gibt die Verweise auf die Projektion und das Fenster. Es kann ebenfalls die Größe des Viewports festgelegt werden. Der Abschnitt über die Projektion legt zum Beispiel Normalen- und Upvektor fest und gibt den Clippingbereich an. Vor allem wird hier aber der Modus für die Projektion definiert, das heißt, ob zum Beispiel Offsets für eine Stereoprojektion berechnet werden sollen. Der Fenster-Abschnitt muss ebenfalls eine Stereoprojektion explizit festlegen. Zusätzlich können Größe, Position und andere Eigenschaften für das Fenster angegeben werden.
Das *Geometry System* managt alle Komponenten, die zur Verarbeitung von Geometrien, verwendet werden. Dazu zählen vor allem der Szenengraph und geometrische Strukturen und Primitiven. Das Geometry System setzt auf dem eingebundenen Szenengraphwerkzeug – hier *OpenSG* – auf, liefert aber dem Benutzer einheitlich definierte Schnittstellen. Dieses Design beruht auf dem in *ViSTA* verfolgten Brücken-Konzept [1], das das Bindungsglied zwischen der *ViSTA* Architektur und „externen“ Toolkits ist und das zwischen Spezifikation und Implementierung unterscheidet. Aufgrund des Brücken-Konzepts wird eine Abstraktionsebene eingefügt, die einen normierten Zugriff auf die verschiedenen externen Werkzeuge gewährleistet und ihre eigentliche Implementierung vor dem Benutzer verbirgt. Nach dem gleichen Konzept funktioniert beispielsweise das Display System bezüglich der *GLUT*.

```
Lights = LIGHT_D0, LIGHT_D1, LIGHT_P0

[LIGHT_D0]
Type          = DIRECTIONAL
AmbientColor  = 0.1, 0.1, 0.1
DiffuseColor  = 0.6, 0.6, 0.6
SpecularColor = 0.5, 0.5, 0.5
Direction     = 1.0, -1.0, 0.0
```

Listing 3.2: Beispiel einer Lichtquellen-Definition

Die Konfiguration des Geometry Systems in der ini-Datei wird durch Schlüsselwörter wie `Lights` oder `Root Orientation` vorgenommen. Die deklarierten Lichtquellen können in eigenen Abschnitten konfiguriert werden. Hier kann der Typ der Lichtquelle, zum Beispiel Punktlicht, ambientes Licht oder gerichtetes Licht, angegeben werden. Die jeweiligen Anteile von ambienten, diffusen und spekularem Licht beruhen auf dem lokalen Beleuchtungsmodell von Phong. Je nach Lichtquellentyp können noch weitere Eigenschaften spezifiziert werden. In Listing 3.2 wird eine Lichtquelle mit gerichtetem Licht definiert, bei der zusätzlich der Richtungsvektor des Lichteinfalls angegeben wird. Die Umgebung der Root Orientierung gibt eine mögliche Anfangsorientierung des Wurzelknotens des Szenengraphens an.
Das Modul des *Interaction Systems* verwaltet eine Anzahl von manuell angefertigten Gerätetreibern für gebräuchliche VR Geräte [1]. Ein weiteres Modul erlaubt zudem den

Zugriff auf externen Treibercode wie beispielsweise binäre API von kommerziellen Tracking Systemen. Das Interaction System unterscheidet zwischen den abstrakten Ebenen der Treiberdekodierungsschicht, der Transformationsphase und der Anwendungsebene [1]. Die Kommunikation zwischen den Ebenen findet über das in *ViSTA* integrierte Netzwerk von Datenflüssen (engl.: data flow network, DFN) statt. Die Treiberdekodierungsschicht bietet den Benutzern eine API zur Integration, Implementierung und Zugriff auf Low-level Geräte. Den Treibern werden Sensoren zugeordnet, welche die Treiberdaten des DFN aufbereiten, indem sie die eingehenden Daten des Treibers in einer zeitlichen Folge oder als „History“ sammeln. Dabei existieren zwei verschiedene Sampling Modi: „Lazy“ Sampling bedeutet, dass die Werte des Sensors ein Mal pro Frame abgefragt werden. Dagegen wird beim „hot“ Sampling der tatsächlich zuletzt aufgezeichnete Sensorwert zurückgeliefert. Letzteres ist jedoch nur bei hochfrequenten, nebenläufigen Sensorupdates sinnvoll und wird im Allgemeinen selten verwendet. Auf die erzeugte History von Samples kann gleichzeitig von Lesern der Transformations- oder Anwendungsschicht zugegriffen werden. Folglich können mit diesem Werkzeug Änderungen an Sensoren registriert, diese beispielsweise als enstprechendes Ereignis an den Event Bus weitergereicht und so auf Benutzereingaben reagiert werden. Praktisch müssen alle Gerätetreiber dem Programm zunächst bekannt gemacht werden. Dies geschieht in dem `DEVICEDRIVERS` Abschnitt der ini-Datei. Außerdem folgt eine Liste mit Verweisen auf die eigentlichen Interaktionsgeräte im Abschnitt `INTERACTIONCONTEXTS`, deren Transformationen mit Hilfe von *XML*-Dateien beschrieben werden.

```
[SPACENAVIGATOR]
TYPE    = SPACENAVIGATOR
SENSORS = SN_MAIN
HISTORY = 10
NAME    = 6DMOUSE

[SN_MAIN]
RAWID = 0
```

Listing 3.3: Treiber-Definition des Spacenavigators

Die Gerätetreiber des `DEVICEDRIVERS`-Abschnitts werden in eigenen Sektionen konfiguriert. Ein Beispielabschnitt für den *Spacenavigator* ist in Listing 3.3 zu sehen. In jeder Treiber-Sektion muss der `TYPE` spezifiziert werden, damit die korrekte Erstellungsmethode des Treibers für den Kernel ausgewählt wird und der Treiber richtig aktiviert wird. Optional können die Schlüsselwörter `HISTORY`, `NAME` und `SENSORS` angegeben werden. `HISTORY` benötigt eine Ganzzahl, die die Anzahl der History-Slots beschreibt. Wird keine History gebraucht, kann die Zahl zwischen eins und fünf gesetzt werden. Der `NAME` ist ein symbolischer Name, mit dessen Hilfe auf den Treiber innerhalb einer Applikation zugegriffen werden kann und auf den in einer *XML*-Interaktionsdatei referenziert werden kann. Das Schlüsselwort `SENSORS` nimmt eine Liste von Konfigurationsabschnitten auf, die für Sensoren des Treibers benötigt werden. Normalerweise wird diese Option für Geräte mit mehrere Sensoren benutzt. Die Senoren können dann in ihrem eigenen Abschnitt parametrisiert werden wie beispielsweise `HEAD` und `GYRO` für den `[DTRACK]`-Treiber. Jeder Sensor muss einen `RAWID`-Eintrag besitzen, der jedoch nur bei mehreren Sensoren als echte Identifikationsnummer dient [34].

```
[NAV_CAM]
ROLE  = CAMERA
GRAPH = configfiles/interaction/spacenavigator.xml
```

Listing 3.4: Verweis auf die XML-Konfigurationsdatei für den Spacenavigator

Die unter `INTERACTIONCONTEXTS` angegebenen Verhaltensweisen der Eingabegeräte werden jeweils in einem Abschnitt definiert, der auf eine *XML*-Interaktionsdatei verweist. In Listing 3.4 wird in der Sektion des *Spacenavigator* auf seine *XML*-Spezifikation referenziert.

```
<module>
  <graph>
    <node name="spacenavigator" type="Sensor">
      <param name="sensor" value="0" />
      <param name="driver" value="6DMOUSE" />
    </node>

    <node name="project_spacenav"
        type="HistoryProject">
      <param name="project">POSITION, AXIS, BUTTON_1,
          BUTTON_2</param>
    </node>

    <node name="nav_handler" type="actionnode">
          <param name="object" value="nav_handler"/>
    </node>
  </graph>
  <edges>
    <edge fromnode="spacenavigator"
        tonode="project_spacenav" fromport="history"
        toport="history" />
    <edge fromnode="project_spacenav"
        tonode="nav_handler" fromport="POSITION"
        toport="position" />
    <edge fromnode="project_spacenav"
        tonode="nav_handler" fromport="AXIS"
        toport="axis" />
    <edge fromnode="project_spacenav"
        tonode="nav_handler" fromport="BUTTON_1"
        toport="btn_1" />
    <edge fromnode="project_spacenav"
        tonode="nav_handler" fromport="BUTTON_2"
        toport="btn_2" />
  </edges>
</module>
```

Listing 3.5: Vereinfachte XML-Datei für den Spacenavigator

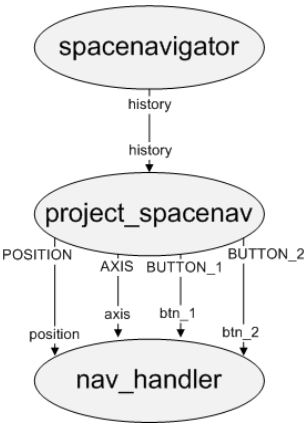


Abbildung 3.19: Graphdarstellung der XML-Datei

Eine vereinfachte *XML*-Interaktionsdatei, die den Aufbau des *Spacenavigators* beschreibt, ist in Listing 3.5 dargestellt. Eingeleitet werden diese *XML*-Interaktionsdateien mit einem `<module>`-Tag. Das entsprechende Modul wird als Graph mit Knoten und Kanten konstruiert, wie in Abbildung 3.19 veranschaulicht. Entsprechend gliedert sich die *XML*-Datei in die Bereiche `graph`, in dem die Knoten spezifiziert werden, und `edges`, die Aus- und Eingabeports der verschiedenen Knoten miteinander verknüpfen. Dabei existiert eine Art Hauptknoten, in dem der eigentliche Sensor, sein Typ und sein Treiber – letzteres anlehnend an den Devicedrivers-Abschnitt in der ini-Datei – angegeben werden. Mit Hilfe des „History-Knotens“ werden die Werte an den dort definierten Ports des Sensors ge-

lesen. Diese werden beispielsweise an einen Aktionsknoten weitergereicht, der wiederum das Auslesen der Werte in der Anwendungsschicht erlaubt. Neben diesem Grundgerüst können mit Hilfe der Graphenstruktur auch weitere Transformationen definiert werden. So können zum Beispiel ankommende Sensorwerte zunächst mit einem Skalar multipliziert werden bevor sie an das Objekt der Applikation übergeben werden. Aber auch komplexere Transformationen sind möglich. Falls der Benutzer auf Nicht-Standard-VR-Geräte zugreifen will oder ein besonderes Verhalten implementieren will, muss er in der Applikationsebene entsprechende Klassen von Aktionsobjekten einrichten, die mit Hilfe des Eventbusses auf Sensorwertänderung nach Belieben des Benutzers reagieren können. Ein Objekt der entsprechenden Klasse muss daraufhin definiert werden und der *ViSTA*-Objektregistry bekannt gemacht werden. Hierzu mehr in Kapitel 4.4.2.
Das bereits erwähnte *Event System* verwaltet alles, was mit Events zu tun hat. Es beinhaltet einen Event Bus, der von Komponenten benutzt werden kann um Events an anderen an dem Event Bus horchenden Komponenten zu senden oder von diesen zu erhalten. Dazu können entweder Observer, die ein Event nur registrieren, oder Handler, die das Event auch ändern können, verwendet werden. Ein Ereignis reflektiert einen bestimmten Zustand der Applikation und kann Auslöser von weiteren Ereignissen sein. *ViSTA* bietet dabei eine Reihe von Standardereignissen an, die sich auf das korrekte Verhalten des *ViSTA*-Systems beziehen [34]. Für den Benutzer besteht aber auch die Möglichkeit eigene Events einzuführen. Zu den wichtigen Ereignissen gehören u.a. die System Events und die Input Events. Die System Events spiegeln den internen Mechanismus des *ViSTA*-Systems wider und beinhalten beispielsweise Init- und Quit-Events, Pre- und Post-Draw-Events und Update-Interaction-Events. Letztere veranlassen den *Interaction Manager* zu überprüfen, ob am spezifierten Gerät eine Wertänderung stattgefunden hat. Diese Abfrage geschieht auf einer Frame-Basis und findet folglich häufiger statt als die Ausführung des entsprechenden Input Events, welches nur bei Wertänderung aufgerufen wird. Input Events teilen die Interaktionen in zwei Haupttypen auf: Bewegungen und Button Events. Bewegungen bedeuten eine Wertänderung am Sensor, die als eine Transformation im dreidimensionalen Raum interpretiert werden kann. Button Events beziehen sich auf das Drücken oder Loslassen eines Buttons, zum Beispiel der einer Maus.

Die Verwendung von *OpenGL*, *OpenSG* und *ViSTA* in der im Rahmen dieser Arbeit entwickelten Software wird im nächsten Kapitel beschrieben.

Kapitel 4

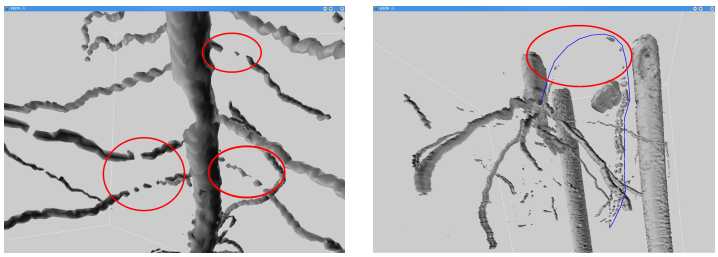
Software-Implementierung

Das Ziel dieser Arbeit ist die Entwicklung einer Software zur Bestimmung morphologischer Informationen von Wurzelsystemen mit Virtual Reality Techniken.
In den Kapiteln 2 und 3 wurden die Grundlagen der Wurzelsysteme, ihrer morphologischen Eigenschaften und des Einsatzes von Virtual Reality Techniken erklärt. Darauf aufbauend wird nun auf die eigentliche Software-Entwicklung und ihre Funktionalitäten eingegangen. Die im Rahmen dieser Arbeit entwickelte Software stellt dem Benutzer Funktionalitäten zur Verfügung, um einen 3D visualisierten MRT-Datensatz eines Wurzelsystems manuell und interaktiv zu rekonstruieren. Mit Hilfe dieser Rekonstruktion werden schließlich die morphologischen Eigenschaften der Wurzeln extrahiert.
In den folgenden Kapiteln wird zuerst die Zielsetzung und Problematik der Arbeit spezifiziert. Danach wird die Konfiguration der Hardwarekomponenten, sowie die Visualisierung der gemessenen Volumendatensätze betrachtet. In Kapitel 4.4.4 wird auf die Funktionalitäten der entwickelten Software eingegangen, das heißt auf die manuelle Erzeugung und Manipulation eines Wurzelsystems. Es folgt die Beschreibung der Extraktion der morphologischen Informationen aus der rekonstruierten Wurzelstruktur. Schließlich werden in Abschnitt 4.6 die aufgetretenen Einschränkungen der entwickelten Software erläutert.

4.1 Zielsetzung und Problematik

Die im Rahmen dieser Arbeit entwickelte Software soll ein Bindeglied zwischen den realen Volumendatensätzen von Wurzelsystemen aus MRT-Messungen und einem Modell zur Simulation der Bodenwasseraufnahme durch Wurzeln schaffen. Die Messdatensätze beinhalten indirekt alle benötigten Informationen in Form von skalaren Attributen für jedes Voxel. Allerdings können aus diesen Volumendatensätzen die Wurzeleigenschaften nicht logisch verknüpft und strukturiert extrahiert werden, da die skalaren Attribute keine zusammenhängende Baumstruktur aus Wurzelsegmenten beschreiben. Diese Extraktion der architektonischen und morphologischen Wurzeldaten ist das Ziel der entwickelten Software. Dazu bietet sie die Visualisierung der Messdatensätze, Methoden für die manuelle Rekonstruktion der Wurzelstruktur und dadurch schließlich die Bestimmung dieser morphologischen Informationen.

Für die Visualisierung der MRT-Volumendatensätze existieren eine Reihe von klassischen Volume Rendering Algorithmen, von denen hier ein indirektes und ein direktes Volume Rendering Verfahren eingesetzt wurde. Dadurch kann der Benutzer sich je nach Bedarf zwischen den Vor- und Nachteilen der beiden Methoden entscheiden.
Die Problematik liegt folglich nicht in der Darstellung der Datensätze, sondern in der Rekonstruktion des Wurzelsystems. Hierbei stellt sich zum einen die Schwierigkeit, dass die Messdatensätze oft verrauscht sind oder durch Messfehler große Lücken innerhalb eines Wurzelzweigs (Abbildung 4.1) entstehen. Diese Probleme kann der Benutzer nur mit Hilfe



(a) Kleinere Lücken (Lupine) (b) Wurzel nur erahnbar (Mais)

Abbildung 4.1: Unterbrochene Wurzelsegmente

seines Wissens über die Wurzel- und Bodenstruktur überwinden. Zum anderen ist es wichtig, dem Benutzer Funktionalitäten zur Verfügung zu stellen, mit denen er die Pflanzenwurzeln nachbilden kann. Eine besondere Rolle spielt dabei die Realisierung der Wurzeln in einer virtuellen Welt. Dazu gehört vor allem die Darstellung der Wurzelstruktur in 3D, die durch eine stereoskopische Projektion realisiert wird. Da die entwickelte Software zu Testzwecken auch mit einer „normalen“ monoskopischen Darstellung implementiert wurde (Abschnitt 4.2.1), konnten die verschiedenen Handhabungen direkt gegenüber gestellt werden. Dabei wurde deutlich, dass die monoskopische Version, die dieselben Funktionalitäten wie die stereoskopische Variante erlaubt, nur umständlich bedient werden kann und dadurch eine Wurzelnachbildung sehr aufwändig und zeitintensiv ist. Die Verwendung der stereoskopischen Abbildung bedeutet aber, dass für die Umsetzung der Stereoskopie für jedes Auge ein Bild mit entsprechend angepasster Perspektive erzeugt werden muss. Diese Aufgabe kann mit Hilfe der *ViSTA*-Programmierschnittstellen erledigt werden, indem der Anwendungsprogrammierer über *ViSTA*s Brücken-Konzept die VR-Hardware anspricht. Zusätzlich muss ein Mauszeiger in diesem dreidimensionalen Raum komfortabel bewegt werden können. Folglich müssen die von ihm gelieferten optischen Trackingdaten sinnvoll verarbeitet werden, wobei ebenfalls einige Funktionen von *ViSTA* behilflich sind. Da das *ViSTA*-Toolkit dem Anwendungsprogrammierer *C++*-Programmierschnittstellen zur Verfügung stellt, wurde auch die Software mit der Programmiersprache *C++* implementiert. Die Verwendung der implementierten Funktionalitäten zur Nachbildung der Wurzeln soll zudem intuitiv und schnell erlernbar sein und zügiges Arbeiten unterstützen. Die abschließende Extraktion der morphologischen Daten als Eingabe für das Bodenwasseraufnahmemodell gestaltet sich dann nach einer erfolgreichen Wurzelsystemrekonstruktion als verhältnismäßig unkompliziert.

4.2 Konfiguration der Hardwarekomponenten

Eine der Aufgaben dieser Arbeit ist die Visualisierung des Messdatensatzes und der rekonstruierten Wurzelstrukturen, sowie ihre interaktive Manipulation im dreidimensionalen Raum. Dafür wird ein stereoskopisches Visualisierungssystem benötigt, das in Kapitel 3.1.1 beschrieben wurde. Das verwendete *PI-casso*-System ist jedoch aufgrund seines Standorts oder seiner anderweitigen Benutzung manchmal nicht verfügbar. Für diesen Fall wird neben der stereoskopischen Darstellung auch eine monoskopische Darstellung implementiert, die zwar für eine komplette Wurzelrekonstruktion ungeeignet ist, aber zur Ansicht einer schon erstellten Wurzelstruktur und für kleine Manipulationen ausreichend ist. Die Umstellung zwischen Mono- und Stereo-Darstellung erfolgt durch Änderung der Spezifikation des Display Systems in der Konfigurationsdatei von *ViSTA*. Da während der Entwicklung jedoch ein ständiges Modifizieren der Datei Zeit kostet, wurde eine Konfigurationsdatei für das monoskopische System und eine für das stereoskopische System erzeugt. Die entsprechende Konfigurationsdatei lässt sich als Kommandozeilenparameter beim Aufruf der entwickelten Software angeben. Desweiteren verändert sich bei der Darstellung der Wurzeln auf einem gewöhnlichen Bildschirm die Aufgaben der Eingabegeräte, die durch zusätzliche *XML*-Interaktionsdateien realisiert werden. Der Quellcode bedarf jedoch nur wenig Änderung und die Hauptfunktionalitäten bleiben vollständig erhalten. In dieser Arbeit wird der Schwerpunkt auf die stereoskopische Darstellung gelegt und die monoskopische Variante nur angeschnitten.

Im Folgenden werden die Konfigurationen der Hardwarekomponenten beschrieben. Diese Konfigurationen müssen für *ViSTA* über *XML*-Interaktionsdateien (vergleiche Abschnitt 3.2.3) realisiert werden. Hier werden diese Konfigurationsdateien mit Hilfe ihrer Graphenstrukturen erläutert.
Das einzige Eingabegerät, dessen Nutzung bei monoskopischem und stereoskopischem Display System identisch ist, ist die Tastatur. Ihre Eigenschaften werden in der Datei `keycontrol.xml`, die im Anhang in Listing B.1 zu finden ist, spezifiziert.

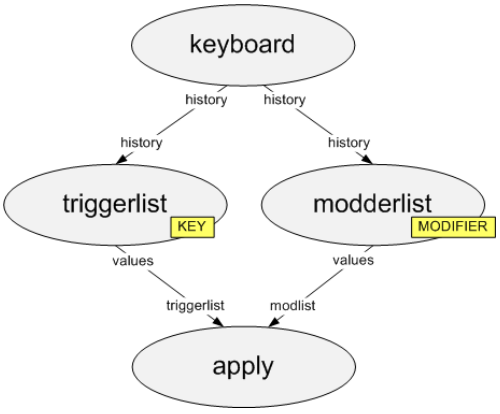


Abbildung 4.2: Graph der Keyboard-Konfiguration

Die entsprechende Graphenstruktur der Tastatur-Konfiguration ist in Abbildung 4.2 dargestellt. Der Hauptknoten `keyboard` liefert an die so genannten Aggregat-Knoten, die den üblichen History-Knoten ersetzen, die eingegangenen Werte des Sensors. Anstatt jedoch nur den neuesten Wert weiterzuleiten, nehmen sie alle neuen Werte seit dem letzten Frame auf und konstruieren daraus einen Vektor, der wiederum an den System-Control-Knoten `apply` weitergegeben wird. So wird verhindert, dass Tastendrücke verloren gehen, falls der Benutzer mehr als einmal pro Frame eine Taste betätigt.
Einer der Aggregat-Knoten (`triggerlist`) registriert die „normalen“ Tasten. Hierzu wird eine Ganzzahl an die Applikation weiter gereicht, die dem ASCII-Code des entsprechenden Buchstabens entspricht. Der andere Aggregat-Knoten (`modderlist`) erhält dagegen ein so genanntes Modifier-Tag, das später die Abfrage von Sondertasten wie zum Beispiel der Steuerung-Taste ermöglicht. Der `apply`-Knoten stellt die Daten schließlich der Applikation zur Verfügung. Die Art der Reaktion auf bestimmte Tastendrücke innerhalb der Applikation wird in Unterkapitel 4.4.2 beschrieben.

4.2.1 Monoskopische Darstellung

Bei der monoskopischen Darstellung wird eine gewöhnliche Maus als Navigationshilfsmittel verwendet. Da eine Maus jedoch nur 2 DOF besitzt, nämlich oben/ unten und rechts/ links, ist ein „freies“ Bewegen durch den Raum nicht möglich. Hier wird durch Drücken verschiedener Maustasten während der Mausbewegung sozusagen ein Freiheitgrad gewonnen. Somit wird eine Rotation des dargestellten Objekts um die x- bzw. y-Achse bei der Bewegung der Maus mit gedrückter linker Maustaste durchgeführt. Eine Translation in z-Richtung entsteht durch Führen der Maus nach oben/ unten mit gedrückter rechter Maustaste. Dieses Verhalten wird in der *XML*-Konfigurationsdatei `trackball_transform.xml` im Anhang in Listing B.2 spezifiziert. Es wird erreicht, indem dort je nach gedrückter Maustaste eine Rotationsmatrix aufmultipliziert wird oder ein Translationsvektor addiert wird.
Desweiteren wird eine Art Mauszeiger benötigt, der eine Position im Raum besitzt und mit dessen Hilfe der Benutzer Objekte erzeugen, anwählen und verändern kann. Dieser Mauszeiger kann in der monoskopischen Darstellung mittels des *Spacenavigators* im Raum bewegt werden. Obwohl der *Spacenavigator* 6 DOF besitzt, werden bei seiner monoskopischen Visualisierung einfachheitshalber nur die drei Translationen berücksichtigt. Seine Konfiguration wird im folgenden Abschnitt über die stereoskopische Darstellung erläutert.

4.2.2 Stereoskopische Darstellung

Bei der stereoskopischen Darstellung auf dem Visualisierungssystem *PI-casso* übernimmt der *Spacenavigator* die Navigation durch die abgebildete Szene, der *Gyrostick* die Manipulation der Objekte und das Kopftracking die interaktive Anpassung der Objekte an die Sicht des Benutzers.

Navigation mit dem Spacenavigator

Der *Spacenavigator* wird mit seinen sechs Freiheitsgraden zur Navigation durch den dreidimensionalen Raum verwendet. Seine Konfiguration wird in der Datei

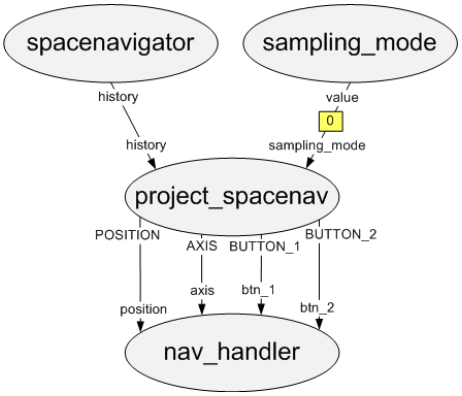


Abbildung 4.3: Graph der Spacenavigator-Konfiguration

`spacenavigator.xml` spezifiziert, die im Anhang in Listing B.3 dargestellt ist. Abbildung 4.3 zeigt die Graphenstruktur des *Spacenavigators* in der Transformationsschicht. Der Hauptknoten `spacenavigator` definiert den Sensor und den Treiber. Der History-Knoten `project_spacenav` liest die Position, die Orientierung und die Status der beiden Buttons des *Spacenavigators* aus. Die `POSITION` wird in einem dreidimensionalen Vektor gespeichert und gibt jeweils die relative Verschiebung zur vorherigen Lage an. Ein Quaternion spezifiziert die Orientierung (`AXIS`). Die Status der Buttons werden als boolsche Werte übermittelt: gedrückt/ nicht gedrückt. Das Auslesen aller Werte geschieht mit lazy Sampling, dass durch die Ganzzahl 0 des `sampling_mode`-Knotens spezifiziert wird. Die ausgelesenen Werte werden an den Aktionsknoten `nav_handler` weitergereicht, der sie dem Anwendungsprogrammierer zur Verfügung stellt.

Kopftracking

Die vom Benutzer getragene Brille ermöglicht das Kopftracking, wobei eine „User Centered Projection“ (UCP) verwendet wird, um das Bild an die Sicht des Benutzers anzupassen. Damit die Projektion korrekt berechnet werden kann, müssen die Versätze in Orientierung und Position des Tracking-Targets bezüglich des Weltkoordinatensystems mit einbezogen werden. Die entsprechenden Transformationen werden in der *XML*-Interaktionsdatei `ucp_picasso_stereo.xml` (Listing B.4 im Anhang) mit der Graphenstruktur aus Abbildung 4.4 definiert. Der History-Knoten `head` sammelt durch lazy Sampling die Werte der Position und der Orientierung des Tracking-Targets der Stereobrille. Der gelieferte Orientierungswert des Targets wird daraufhin mit Hilfe des `rotate_glasses`-Knoten und dem Knoten `rotate_y` an die Ausrichtung des Koordinatensystems angepasst. Zum Positionswert des Tracking-Targets, der sich auf das Weltkoordinatensystem bezieht, wird ein Offset mittels des `translate_pos_tracker`-Knotens addiert. Damit wird der Abstand der Targets zur Brille kompensiert. Zudem muss der Ursprung des lokalen Koordinatensystems der Brille, der standardmäßig in der unteren linken Ecke der Brille liegt, für die Projektion in den Punkt zwischen den Augen gelegt werden. Hierzu dient der `to_glasses_center`-Vektor, der die entsprechende Translation bezüglich der aktuellen Orientierung angibt.

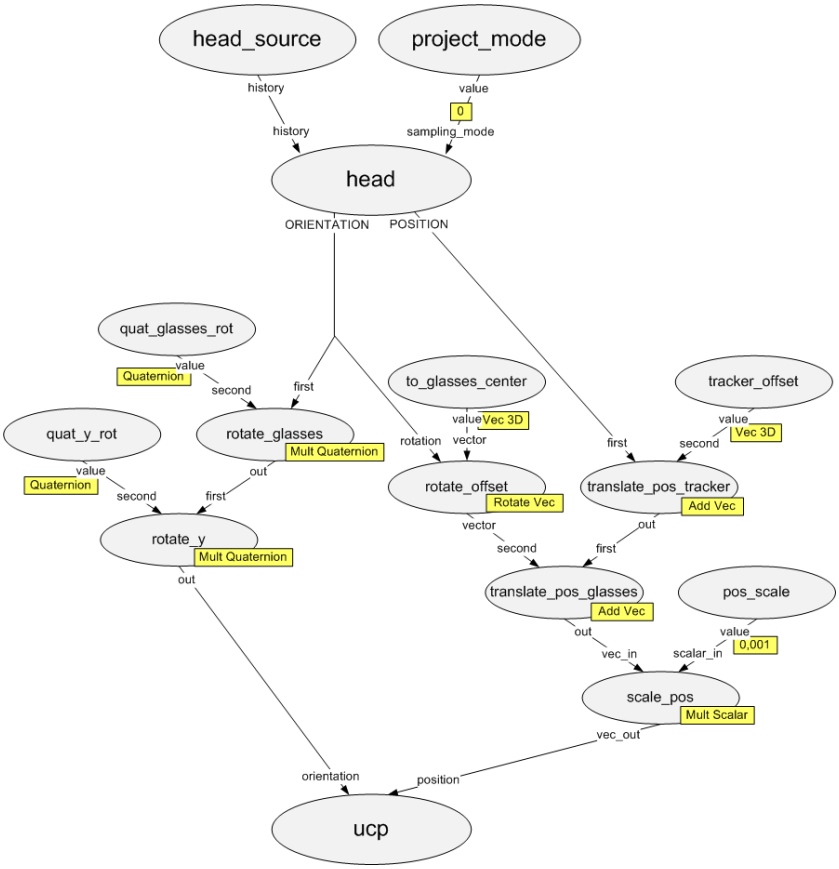


Abbildung 4.4: Graph der UCP-Konfiguration

Das Ergebnis der beiden Verschiebungen (`translate_pos_glasses`) wird danach durch die Multiplikation mit dem Wert 0,001 skaliert, um von der vorgegebenen Einheit m des Weltkoordinatensystems auf die Einheit mm zu wechseln. Die transformierte Orientierung und Position des Benutzers („viewers“) werden schließlich an den Viewersink-Knoten `ucp` weiter gegeben. Über den Viewersink-Knoten kann dann die Kamera des Display Systems verändert werden, das heißt die Projektion wird der Sicht des Benutzers angepasst.

Manipulation mit dem Gyrostick

Die Konfiguration des getrackten *Gyrosticks* aus der Datei `gyrostick.xml` (Listing B.5 im Anhang) ermöglicht zum einen ähnliche Anpassungen an das Koordinatensystem des Visualisierungssystems bezüglich Position und Orientierung wie beim Kopftracking und zum anderen die Eigenschaften einer Maus ähnlich den Buttons des *Spacenavigators*. Abbildung 4.5 verdeutlicht, dass das Orientierungsquaternion an die Ausrichtung des Koordinatensystems des Bildschirms angepasst wird. Zu den Positionskoordinaten, die sich auf das Weltkoordinatensystem beziehen, wird wie beim Kopftracking ein `tracker_offset` addiert. Die Translation `to_stick_center`, die den Ursprung des *Gyrosticks* definiert, be-

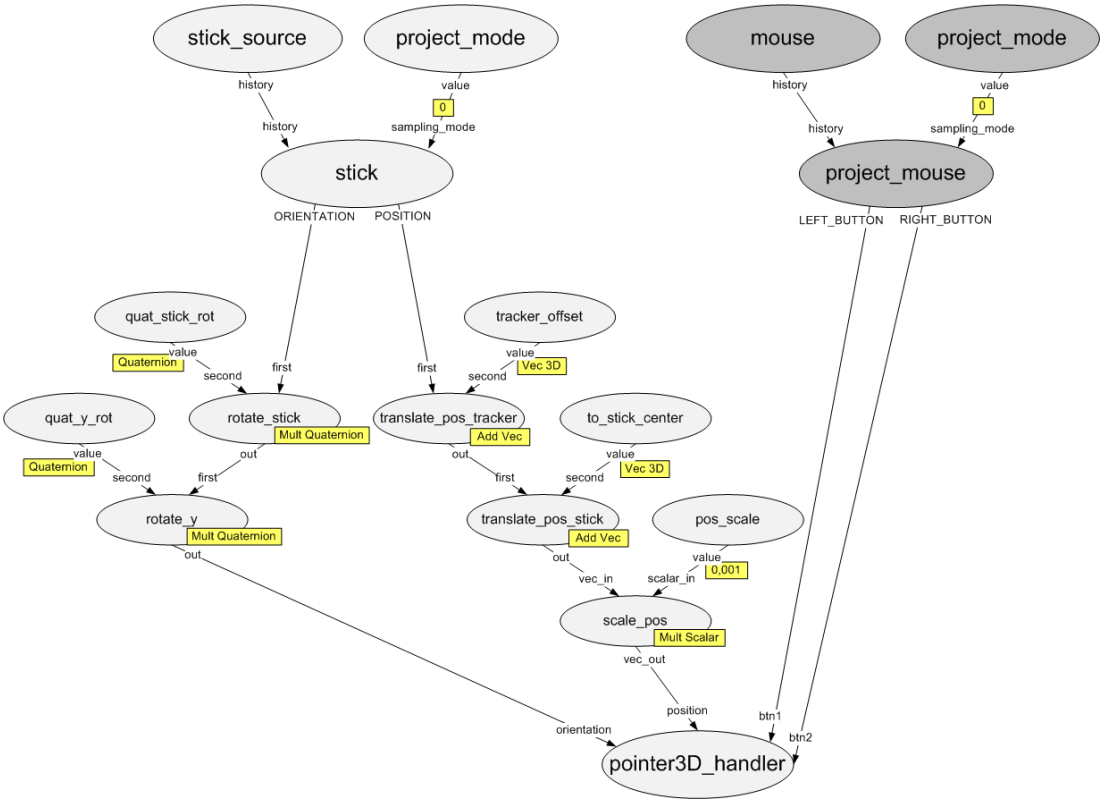


Abbildung 4.5: Graph der Gyrostick-Konfiguration

zieht hier allerdings nicht die aktuelle lokale Orientierung wie beim Kopftracking mit ein, sondern verschiebt pauschal den Ursprung einige Zentimeter in Richtung der negativen z-Achse des Weltkoordinatensystems. Dadurch erhält der Benutzer den Eindruck, dass sein Mauszeiger weiter in das Bild hinein reicht, sodass sich entfernte Objekte besser damit manipulieren lassen. Nach der Skalierung um den Faktor 0,001 läuft die Positionsangabe neben der Orientierung in den Aktionsknoten `pointer3D_handler`.
Die Knöpfe des *Gyrosticks* lassen sich über einen zweiten Hauptknoten, den `mouse`-Knoten, konfigurieren. Hier wird mit lazy Sampling die Status der linken und rechten Maustaste als boolsche Werte ausgelesen und ebenfalls dem Aktionsknoten `pointer3D_handler` zugeführt.

Konfiguration des Systems

Nachdem das Verhalten und die Eigenschaften der verschiedenen Eingabegeräte erläutert wurden, wird nun die Konfiguration des Gesamtsystems betrachtet. Teile davon wurden bereits in Kapitel 3.2.3 als Beispiele für die Konfiguration der verschiedenen *ViSTA* Kernel Einheiten gebraucht und werden hier nur noch kurz aufgegriffen. Die komplette Konfigurationsdatei `vista_picasso.ini` für das stereoskopische System ist in Anhang B.2 zu finden. Zunächst wird die Angabe der `DISPLAYSYSTEMS` auf `STEREO` gesetzt und damit das Fens-

ter, der Viewport und die Projektion definiert (vergleiche Listing 3.1). Danach werden die Treiber für die Eingabegeräte und die Verweise auf ihr Interaktionsverhalten gesetzt. In Listing 4.1 werden die Tastatur, der *Spacenavigator*, der *Gyrostick* und das Tracking definiert.

```
DEVICEDRIVERS       =  KEYBOARD, SPACENAVIGATOR, GYROMOUSE, DTRACK
INTERACTIONCONTEXTS =  KEYCONTROL, NAV_CAM, NAVIGATION_GYRO_CTX, UCP_CONTEXT
```

Listing 4.1: Definition der Eingabegeräte

Es folgen die Abschnitte, auf die in `INTERACTIONCONTEXTS` referenziert wird. Sie verweisen ihrerseits auf die *XML*-Interaktionsdateien wie in Listing 4.2 zu sehen ist.

```
[KEYCONTROL]
ROLE  = KEYCONTROL
GRAPH = configfiles/interaction/keycontrol.xml

[NAV_CAM]
ROLE          = CAMERA
GRAPH         = configfiles/interaction/spacenavigator.xml
RELOADTRIGGER = B

[NAVIGATION_GYRO_CTX]
ROLE  = POINTDEVICE
GRAPH = configfiles/interaction/gyrostick.xml

[UCP_CONTEXT]
ROLE  = UCP
GRAPH = configfiles/interaction/ucp_picasso_stereo.xml
```

Listing 4.2: Verweise auf die XML-Interaktionsdateien

In Listing 4.3 werden die Treiberinformationen gesetzt. Dies geschieht gemäß der Angaben in `DEVICEDRIVERS`. Dazu gehört vor allem der Typ des Geräts und seine Sensoren, die mittels der `RawID` identifiziert werden können. Der Treiber `DTRACK` für die Trackinggeräte nimmt eine Sonderrolle ein, da er für zwei Geräte – Stereobrille und *Gyrostick* – benötigt wird. Dies wird in der Sensorenliste mit `HEAD` und `GYRO` angegeben. Der `CONNECTIONS`-Eintrag liefert eine Liste von Sektionsverweisen, die zum Scannen und Erstellen einer Netzwerkverbindung für den Treiber benötigt werden. Die weiteren anzugebenen Eigenschaften hängen vom Verbindungstyp ab, sind aber im Allgemeinen selbsterklärend, wie auch durch die Sektionen `DTRACKCONTROL` und `DTRACKDATA` verdeutlicht wird.

```
[KEYBOARD]
TYPE          = KEYBOARD
DEFAULTWINDOW = TRUE
HISTORY       = 1
SENSORS       = KEYB_MAIN

[SPACENAVIGATOR]
TYPE    = SPACENAVIGATOR
SENSORS = SN_MAIN
HISTORY = 10
NAME    = 6DMOUSE

[SN_MAIN]
RAWID = 0
```

```
[GYROMOUSE]
TYPE          = MOUSE
HISTORY       = 10
SENSORS       = MOUSE_MAIN
DEFAULTWINDOW = TRUE

[MOUSE_MAIN]
RAWID = 0

[DTRACK]
TYPE        = DTRACK
NAME        = DTRACK
PROTOCOL    = DTRACK2
CONNECTIONS = DTRACKCONTROL, DTRACKDATA
SENSORS     = HEAD, GYRO
HISTORY     = 10
ATTACHONLY  = FALSE

[DTRACK2]
NAME = dtrack2

[HEAD]
TYPE = BODY
RAWID = 0

[GYRO]
TYPE = BODY
RAWID = 1

[DTRACKCONTROL]
TYPE       = TCP
DRIVERROLE = CONTROLCONNECTION
ADDRESS    = trackingpc
PORT       = 50105
DIRECTION  = OUTGOING

[DTRACKDATA]
TYPE       = UDP
DRIVERROLE = DATACONNECTION
ADDRESS    = renderpc
PORT       = 5001
DIRECTION  = INCOMING
```

Listing 4.3: Definition der Treiberinformationen

4.3 Visualisierung der Volumendatensätze

Der erste Schritt bei der Entwicklung der Software ist die Visualisierung des Volumendatensatzes, der aus den MRT-Messungen der Pflanzenwurzeln resultiert. Der Datensatz enthält die skalaren Attribute der Voxel in fortlaufender Reihenfolge. Das bedeutet jedoch, dass der Datensatz zunächst keine Informationen darüber enthält, welche Auflösung er in der x-, y- oder z-Dimension besitzt. Auch geht die Information über die Abmessungen des Mess-Pflanzencontainers, auf die sich die Samplingauflösung bezieht, verloren. Die Größe des Messcontainers und die Messpunktzahlen pro Dimension definieren zusammen die zur korrekt skalierten Darstellung erforderlichen Abstände zwischen den Voxeldaten, die auch als „Spacing“ bezeichnet werden.

Aus diesem Grund wird der Datensatz zunächst in das VTK-Legacy-Format umgewandelt, das alle benötigten Informationen in einem Datei-Header speichert. Der Aufbau einer solchen Datei und die entsprechende Umwandlung werden in Abschnitt 4.3.1 beschrieben. Danach kann der Datensatz grafisch dargestellt werden. Dies geschieht zum einen mit einem texturbasierten Verfahren und zum anderen mit einer Oberflächendarstellung mit Hilfe des Marching Cubes Algorithmus. Beide Volume Rendering Techniken werden in Abschnitt 4.3.2 erklärt.

4.3.1 Umwandlung ins VTK-Legacy-Format

Zum Abspeichern der Informationen über die Dimensionen und Spacings innerhalb des Datensatzes wurde für diese Arbeit das standardisierte VTK-Legacy-Format gewählt. Dieses ermöglicht die Darstellung des Datensatzes auch mit externen Betrachtungsprogrammen wie *Paraview*.

Aufbau einer VTK-Legacy-Datei

Das VTK-Legacy-Format ist ein Dateiformat der Klassenbibliothek *Visualization Toolkit (VTK)*, das sowohl per Computer als auch per Hand einfach zu lesen und zu schreiben ist [41].

```
1  # vtk DataFile Version 3.0
2  VTK dataset
3  BINARY
4  DATASET STRUCTURED_POINTS
5  DIMENSIONS 256 256 110
6  SPACING 0.039062 0.031250 0.050000
7  ORIGIN 0 0 0
8  POINT_DATA 7208960
9  SCALARS scalars unsigned_char 1
10 LOOKUP_TABLE default
```

Listing 4.4: Header einer VTK-Legacy-Datei

Eine VTK-Legacy-Datei besteht aus fünf Teilen. Zunächst wird die VTK-Bezeichnung und Versionsnummer angegeben, die in Zeile 1 des Listings 4.4 zu finden ist. In der 2. Zeile folgt der Titel, der zur Beschreibung der Daten verwendet werden kann. Der nächste Teil bezieht sich auf das Datenformat, das entweder ASCII oder binär sein kann. Für diese Arbeit werden die Original-ASCII-Datensätze in Binärdateien umgewandelt. Der vierte Teil beschreibt die Datenstruktur des Datensatzes. Dazu gehören die Geometrie der Datenstruktur und die Topologie. Die gemessenen Volumendaten liegen in einem anisotropen Gitter vor, das in *VTK* durch eine strukturierte Punktmenge beschrieben werden kann. Diese Datenstruktur wird durch die Dimensionen, die Spacings und den Ursprungspunkt definiert (Zeilen 4-7). Im Beispiellisting liegt ein Datensatz mit 256 x 256 x 110 Punkten vor. Dabei haben die Voxel einen Abstand von 0,039062 Längeneinheiten (LE) in x-Richtung, 0,031250 LE in y-Richtung und einen Abstand von 0,05 LE in z-Richtung. Der letzte Teil des VTK-Legacy-Headers besteht aus den Attributen des Datensatzes. Er wird mit der Anzahl der Punkte, die sich aus der Multiplikation der Dimensionszahlen ergibt, eingeleitet. Die Attribute `SCALARS` und `LOOKUP_TABLE` bieten die Möglichkeit eine

Transferfunktion anzugeben, die Skalarwerte mit dem Typ `unsigned_char` entsprechend farbig abbildet. Hinter diesem Header folgen die binären Werte der Volumendaten.

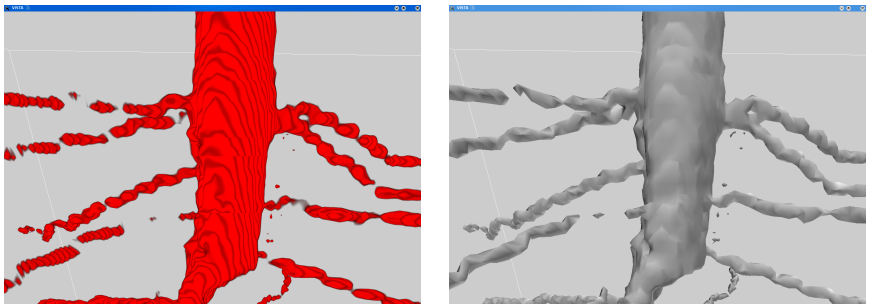
Umwandlung der ASCII-Datei

Die Messdatensätze müssen vor der ersten Verwendung einmalig in das VTK-Legacy-Format umgewandelt werden. Dazu wurde das kleine Tool *Ascii2bin* entwickelt, dass dem Anwender diese Arbeit abnimmt.
Der Benutzer muss dieses Kommandozeilenprogramm mit den entsprechenden Argumenten ausführen: Dazu zählen die Angaben der Dimensionen (Messpunktezahlen) in x-, y- und z-Richtung, sowie die Angabe der Abmessungen des Messvolumens in x-, y- und z-Richtung. Die Größeneinheiten müssen analog zu denen gewählt werden, die in der entwickelten Rekonstruktionssoftware verwendet werden. Optional kann angegeben werden, dass der Messdatensatz bereits in binärer Form vorliegt und nicht mehr in diese konvertiert werden muss. Es wird allerdings von einem ASCII-Datensatz ausgegangen. Außerdem kann der Name einer Ausgabedatei übergeben werden. Wenn nicht, erhält der Dateiname die Endung „.vtk“.
Das *Ascii2bin*-Programm schreibt mit Hilfe der angegebenen Informationen den VTK-Legacy-Header. Dazu werden die Dimensionen entsprechend gesetzt, die Spacings und die Anzahl der Punkte berechnet. Falls nicht die Option angegeben wurde, dass der Datensatz bereits binär vorliegt, werden zusätzlich die ASCII-Werte des Messdatensatzes in binäre Form umgewandelt.

4.3.2 Volume Rendering

Liegt der Volumendatensatz in dem korrekten Format vor und enthält somit alle wichtigen Informationen, kann er mit der im Rahmen dieser Arbeit entstandenen Software eingelesen und visualisiert werden.
Die Volumen-Visualisierung geschieht durch so genanntes Volume Rendering, das eine Reihe von Techniken zur Generierung von 2D Projektionen aus 3D diskret gesampelten Datenräumen umfasst. Der Messdatensatz liegt in einem solchen diskreten anisotropen Gitter vor und die Abtastpunkte werden durch Voxel mit einer skalaren Eigenschaft repräsentiert. Um eine kontinuierliche Darstellung zu erhalten, müssen diese skalaren Werte auch im Zwischenraum der Voxel ermittelt werden. Dies geschieht zumeist durch trilineare Interpolation.
Volume Rendering kann grob in die Bereiche des direkten und des indirekten Volume Rendering aufgeteilt werden [35]. Beim direkten Volume Rendering wird das 2D Bild direkt aus dem 3D Datensatz erzeugt und zwischen einer objekt-orientierten und bild-orientierten Vorgehensweise unterschieden. Indirektes Volume Rendering verwendet geometrische Primitive zur Darstellung. Hier wird zuerst der Datensatz bearbeitet, beispielsweise eine Iso-Oberfläche extrahiert, und dann die daraus gewonnenen Daten zur Visualisierung weiterverwendet [35].
Zur Volumen-Visualisierung wurde für diese Arbeit zunächst ein direktes Volume Rendering Verfahren mit 3D Texturen verwendet (Abbildung 4.6(a)). Dieses texturbasierte

Verfahren läuft für kleine Dimensionen wie bei 128^3 Volumenelementen sehr flüssig, allerdings ist die Echtzeit-Berechnung bei Transformationen eines Datensatzes mit 256^3 Punkten nicht mehr gewährleistet. Deswegen wurde zusätzlich eine indirekte Volume Rendering Technik mit Oberflächenextraktion durch den Marching Cubes Algorithmus eingesetzt (Abbildung 4.6(b)). Dieses Verfahren schränkt zwar die Wurzelstruktur auf ihre unstrukturierte Oberfläche ein, wodurch Details verloren gehen und keine Transparenz wie beim texturbasierten Verfahren möglich ist. Jedoch wird dadurch auch die Datenmenge auf ein wichtiges Teilstück reduziert, sodass – im Gegensatz zum texturbasierten Verfahren – eine ruckelfreie Visualisierung von großen Datensätzen möglich ist.



(a) 3D texturbasiertes Verfahren: Texturen auf Slices

(b) Iso-Oberflächendarstellung: Dreiecke bilden eine Oberfläche

Abbildung 4.6: Ergebnisse des Volume Renderings der Lupine

Beide Verfahren werden in den folgenden Abschnitten erläutert, wobei die Dimension des Pflanzencontainers bzw. des Datensatzes mit der größten Abmessung stets auf den Bereich $[0, 1]$ skaliert wird. Die anderen, kleineren Dimensionen werden entsprechend mit dem selben Skalierungsfaktor angeglichen. Die Skalierung wird vorgenommen, damit jedes Volumen auf den gleichen Bereich abgebildet wird und der Benutzer so eine einheitliche Sichtweise erhält. Außerdem wird während der Visualisierung des Datensatzes eine Art Würfel bzw. Konturbox, die die Wurzelstruktur umschließt und ebenfalls auf das Intervall $[0, 1]$ begrenzt ist, erzeugt. So kann sich der Benutzer leichter im Geschehen orientieren.

Texturbasiertes Verfahren

Die texturbasierten Verfahren gehören zu den direkten Volume Rendering Techniken und ermöglichen die Darstellung von Variationen und Details auf Oberflächen, die sich nicht durch die vorgegebene Geometrie darstellen lassen [3]. Durch die Verwendung von Transparenzwerten, den so genannten Alpha-Werten, kann sogar die innere Struktur des Volumendatensatzes visualisiert werden. Das Verfahren zum Aufbringen einer Textur auf eine Objektoberfläche im zwei- oder dreidimensionalen Raum wird auch als Texture-Mapping bezeichnet. Hier wurden 3D Texturen für das Texture-Mapping eingesetzt. Eine 3D Textur wird formal als eine Funktion beschrieben, die von drei Variablen (s, t, r) eines orthogonalen Texturkoordinatensystems abhängt, das auf den Einheitswürfel normiert wird [9]. Die Textur wird in dieser Arbeit durch den normierten Messdatensatz geliefert, der an jeder

Texturkoordinate (s, t, r) einen Wert im Bereich $[0, 255]$ definiert. Dieser skalare Wert wird später auf einen RGB-Farbwert mit Alpha-Kanal (RGBA) abgebildet, das heißt das Voxel erhält eine Farbe und eine bestimmte Transparenz.
Um die so definierte 3D Textur auf ein Objekt aufzuziehen, müssen die Volumendaten zunächst in Scheiben (engl.: slices), die dann die entsprechende Texturstruktur erhalten, geteilt werden (Abbildung 4.7). Die Slices bilden Polygone und werden so erzeugt, dass sie immer senkrecht zum Betrachter positioniert sind. Die Texturkoordinaten müssen der räumlichen Ausrichtung der Volumendaten angepasst werden [20]. Beim Rendering wird deswegen jedem Oberflächenpunkt des Slices eine Texturkoordinate zugewiesen. In *OpenGL* müssen jedoch nur die Eckpunkte der Textur Bildkoordinaten zugeordnet werden. Die restlichen Koordinaten eines Slices werden automatisch mittels Interpolation gesetzt.

Abbildung 4.7: 3D texturbasierter Ansatz: Slices sind zur Betrachterebene ausgerichtet [4]

Die Darstellung mit dem vorliegenden texturbasierten Verfahren verläuft dementsprechend folgendermaßen: Zunächst müssen die Eigenschaften des Texture-Mappings – wie die Aktivierung von Transparenz, die Festlegung der Art der Texturkoordinaten oder die Mischfunktion der transparenten Farbwerte – in *OpenGL* initialisiert werden. Zum Zeichnen der Slices wird zunächst der Würfel, der das Messvolumen einschließt, entsprechend zur Sicht des Betrachters transformiert. Danach wird der z-Bereich dieses Würfels objekt-orientiert von hinten nach vorne durchlaufen und der Schnitt der aktuellen xy-Ebene mit den Kanten des Würfels berechnet. Die Schnittpunkte werden gegen den Uhrzeigersinn sortiert und die Textur wird auf das dadurch gebildete Polygon abgebildet. Dabei wird ein so genanntes back-to-front Compositing angewendet. Folglich werden die skalaren Werte des Volumens, die zwischen den Scheiben interpoliert wurden, mit Hilfe einer Lookup-Tabelle auf die RGBA-Kanäle verteilt. Mit Hilfe des Alpha-Blendings wird dabei die wirkliche Farbe eines Pixels berechnet, indem die Farben der bereits überlagerten Schichten je nach Transparenzwert gewichtet mit in die Farbzuweisung einbezogen werden. Eine entsprechende eindimensionale Transferfunktion, die die RGBA-Werte für jeden Skalar angibt, muss vom Benutzer in einer kleinen *XML*-Datei definiert werden. Hierbei wird oft der Wertebereich der Volumendaten auf $[0, 1]$ skaliert.

```
<ColorMap name="TransferFuncRed" space="RGB">
<Point x="0" o="0" r="1" g="0" b="0"/>
<Point x="0.06" o="0" r="0" g="0" b="0"/>
<Point x="0.07" o="1" r="1" g="0" b="0"/>
<Point x="1" o="1" r="1" g="0" b="0"/>
</ColorMap>
```

Listing 4.5: Transferfunktion für die RGBA-Kanäle

Listing 4.5 legt beispielsweise eine Art Schwellenwert fest: Für alle Werte $x \leq 0,06$ wird der Alpha-Kanal auf 0, das heißt das Voxel auf vollkommen durchsichtig gesetzt. Die Wer-

te $x \geq 0,07$ beschreiben dagegen ein vollkommen solides, das heißt lichtundurchlässiges Element mit der Farbe rot. Auf den Werten dazwischen wird zwischen den Farbenwerten von schwarz und rot interpoliert. Dadurch entsteht eine Textur, die nur Voxel mit den skalierten Werten $x \geq 0,06$ bzw. den Byte-Werten $x \geq 0,06 \cdot 255 = 15,3$ darstellt, sodass gerade die Wurzeln rot mit einem dunklen Rand eingefärbt werden. Die Transferfunktion sollte vom Benutzer beim Aufruf der Software angegeben werden, da ansonsten eine Standard-Transferfunktion verwendet wird, die für die meisten Wurzelvisualisierungen nicht ausreichend ist.
Zur Reduktion von Aliasing-Effekten wird bei diesem 3D texturbasierten Verfahren eine so genannte Vorintegration (engl.: pre-integration) verwendet, die durch eine Vorberechnung der Alpha-Kanäle zwischen zwei Skalaren auf einem Strahl Peaks in der Transferfunktion besser darstellen kann.

Oberflächendarstellung mit Marching Cubes

Als indirektes Volume Rendering Verfahren wurde eine Iso-Oberflächenextraktion mit Hilfe des Marching Cubes Algorithmus eingesetzt. Dabei werden Punkte mit gleichwertigen skalaren Attributen – den Isowerten – zu Isoflächen mit dreidimensionalen Geometrien verbunden. Zur Herleitung des Marching Cubes Algorithmus wird zunächst der zweidimensionale Fall mit dem Marching Squares Algorithmus betrachtet.
Voraussetzung für die Anwendung dieser Algorithmen ist eine rechtwinklige Topologie und eine feine Datenauflösung, sodass eine lineare Interpolation zwischen den Attributen realitätsnah ist [2]. Mit dem Marching Squares Algorithmus wird eine Isolinie erzeugt, die Zellen mit dem gleichen Isowert verbindet. Abbildung 4.8 zeigt eine Isolinie mit dem Isolevel 5. Der Verlauf der Isolinie innerhalb einer Zelle wird durch lineare Interpolation

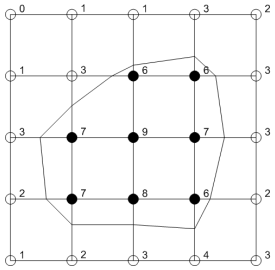


Abbildung 4.8: Isolinie durch lineare Interpolation zum Isowert 5, nach [2]

berechnet und durch einen Bitcode klassifiziert. Einem Gitterpunkt der Zelle wird eine 1 zugewiesen, falls sein Wert größer gleich dem gewählten Isowert ist, und eine 0, falls sein Skalar kleiner ist. Der Bitcode der oberen rechten Zelle in Abbildung 4.8 lautet, beginnend bei dem Wert 2, dementsprechend (0, 0, 1, 0). Da jede Zelle aus vier Gitterpunkten besteht, können so 2^4 mögliche Bitcodes entstehen. Die Bitcodes werden zusammen mit dem entsprechenden Linienverlauf in einer Lookup-Tabelle gespeichert. Ein Linienlauf muss jedoch nicht eindeutig sein, wie Abbildung 4.9 veranschaulicht. In solchen Fällen werden Kreuzungen verboten und aus den anderen Alternativen wird beispielsweise die Linienver-

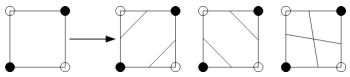


Abbildung 4.9: Mehrdeutigkeiten bei diagonal gegenüberliegenden Skalaren, nach [2]

laufsrichtung mit der kleinsten Steigungsrichtung ausgewählt [2]. Die gefundenden Linienverläufe werden unter Einbeziehung der Nachbarzellen schließlich zu zusammenhängenden Polygonzüge verbunden. Folglich baut sich der Marching Squares Algorithmus folgendermaßen auf: Zuerst wird eine Zelle ausgewählt und der Bitcode der Gitterpunkte der Zelle bestimmt. Der Verlauf der Linie innerhalb der Zelle wird mit Hilfe des Bitcodes und der Lookup-Tabelle ermittelt. Danach wird die Lage der Schnittpunkte der Isolinie mit den Zellkanten durch lineare Interpolation berechnet. Es wird mit der nächsten Zelle fortgefahren bis schließlich alle Zellen durchlaufen sind.
Der Marching Squares Algorithmus kann leicht auf den dreidimensionalen Fall verallgemeinert werden [2], wodurch der Marching Cubes Algorithmus entsteht. Die Klassifikation eines Voxels mit Hilfe des Bitcodes erzeugt $2^8 = 256$ Fälle, die sich durch Komplementbildung, Spiegelung und Rotation auf 15 Äquivalenzklassen abbilden lassen (siehe Abbildung 4.10). Durch den Bitcode wird eine passende Dreiecksmenge aus der Lookup-Tabelle

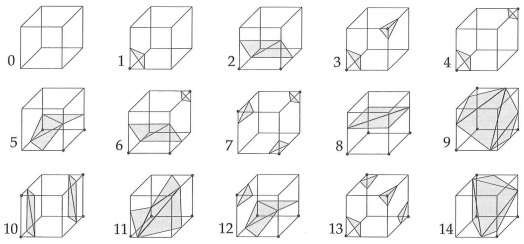


Abbildung 4.10: 15 Fälle im Marching Cubes Algorithmus [2]

gewählt und die Schnittpunkte, die die Eckpunkte der Dreiecke repräsentieren, durch lineare Interpolation berechnet. Mehrdeutigkeiten werden eingeschränkt, indem Löcher in Isoflächen ausgeschlossen werden und beispielsweise benachbarte Voxel identische Dreieckskanten besitzen müssen. Um lokale Beleuchtungsmodelle auf den Isoflächen anwenden zu können, müssen zusätzlich die Normalen auf den Dreiecken bestimmt werden. Zur Ermittlung der Normalen werden mit Hilfe von finiten Differenzen die Gradienten aus den Skalaren an den Gitterpunkten berechnet. Durch lineare Interpolation werden die Normalenvektoren an den Schnittpunkten des Dreiecknetzes mit den Gitterkanten bestimmt. Auf diese Weise werden alle Voxel des Datensatzes bearbeitet.
Zur Anwendung des Marching Cubes Algorithmus auf die Wurzelstruktur, muss der Benutzer den passenden Isowert beim Programmaufruf angeben. Ansonsten wird ein Standardwert genommen, der aber nur selten zu einer zufriedenstellenden Darstellung des Wurzelsystems führt. Die Angabe des Isowerts bedeutet jedoch, dass der Benutzer die skalaren Werte seines Datensatzes kennen muss. Anderenfalls muss er den Isowert durch Ausprobieren finden. Dabei muss die Oberfläche allerdings jedes Mal neu berechnet werden, wenn sich der Isowert ändert.

Da sowohl das texturbasierte Verfahren als auch die Iso-Oberflächendarstellung Vor- und Nachteile besitzen, wurden beide Techniken in der im Rahmen dieser Arbeit enstandenen Software eingesetzt. Durch Tastendruck auf der Tastatur kann der Benutzer die Ansicht zwischen den beiden Volumen-Visualisierungen wechseln um somit die bestmögliche Darstellung des gemessenen Wurzelsystems auswählen zu können.

4.4 Rekonstruktion der Wurzelstrukturen

Der nächste Schritt ist die eigentliche interaktive Rekonstruktion der visualisierten Wurzelstruktur um danach die morphologischen Informationen extrahieren zu können. Diese Rekonstruktion wird manuell durchgeführt, indem die Wurzelsegmente mit Hilfe des *Gyrosticks*, der auf der Arbeitsumgebung durch einen Zeiger repräsentiert wird, „nachgezeichnet“ werden. Eine typische Arbeitsumgebung ist dabei in Abbildung 4.11 dargestellt. Durch die Rekonstruktion wird eine interne Datenstruktur aufgebaut, deren Objekte die

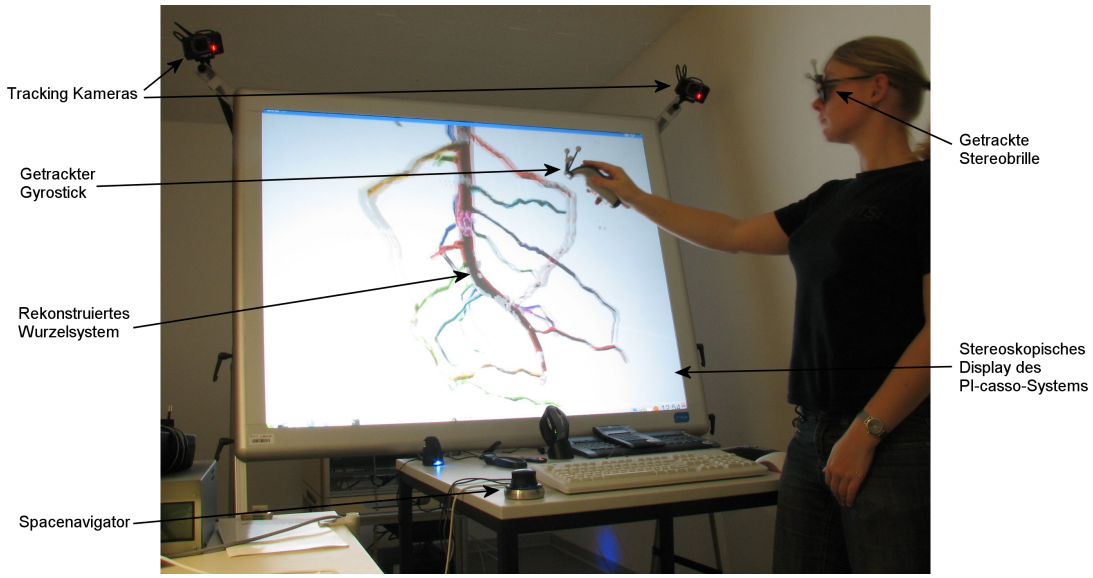


Abbildung 4.11: Arbeitsumgebung für die Wurzel-Rekonstruktion am PI-casso-System

Attribute, die zur Bestimmung der Morphologie benötigt werden, speichern. Auf diese Datenstruktur wird in Abschnitt 4.4.1 näher eingegangen. Um in der Software die Benutzerinteraktionen korrekt umzusetzen, müssen die Informationen der Eingabegräte gesammelt und entsprechend verarbeitet werden, was in Unterkapitel 4.4.2 erläutert wird. Ebenfalls müssen die Benutzerinteraktionen auf dem Bildschirm sichtbar werden, dass durch verschiedene *OpenGL*-Zeichenroutinen (Abschnitt 4.4.3) erreicht wird. Zur Erzeugung des Wurzelsystems und zur nachträglichen Manipulation stehen dem Benutzer eine Reihe von Funktionalitäten zur Verfügung. Sie werden in Abschnitt 4.4.4 beschrieben. Wurde die Nachbildung der Wurzelstruktur erfolgreich fertig gestellt, können schließlich die benötigten morphologischen Informationen bestimmt und in eine Ausgabedatei ge-

schrieben werden (Kapitel 4.5).
Alle Manipulationsaktionen und Meldungen über Fehler bei der Benutzung der Software werden für den Anwender auf der Konsole protokolliert.

4.4.1 Datenstrukturen und Attribute

Ein Wurzelsystem entwickelt sich aus einem Samen. Je nach Pflanzenart entsteht ein allorhizes oder homorhizes Wurzelsystem. Beide Wurzelsysteme bilden jedoch vom Samen aus gesehen eine hierarchische Baumstruktur. Aus diesem Grund wurde auch ein hierarchischer Baum als Datenstruktur zur Nachbildung der Wurzeln gewählt. Der Benutzer ist beim Nachbauen des Wurzelsystems nicht an die chronologische Wachstumsreihenfolge der Wurzelsegmente gebunden, da diese bei den bisherigen Bodenwasseraufnahme-Simulationen nicht von Interesse ist. Deswegen wird dem Anwender die Möglichkeit gegeben zunächst mehrere kleine Bäume aufzubauen und diese dann später zu einem kompletten Wurzelsystem zu verbinden. Dies wird mit Hilfe einer „Wald"-Datenstruktur realisiert, die eine Liste von Bäumen enthält. Ein Baum besteht klassischerweise aus „Knoten" und wird über den Wurzelknoten („root") definiert. Jeder Knoten speichert wiederum eine Liste von Kinderknoten und besitzt zusätzlich eine Referenz auf seinen eindeutigen Vaterknoten. Dadurch kann ein Baum rekursiv vorwärts und rückwärts durchlaufen werden. Eine vereinfachte Darstellung dieser Datenstruktur ist in Abbildung 4.12 zu sehen.

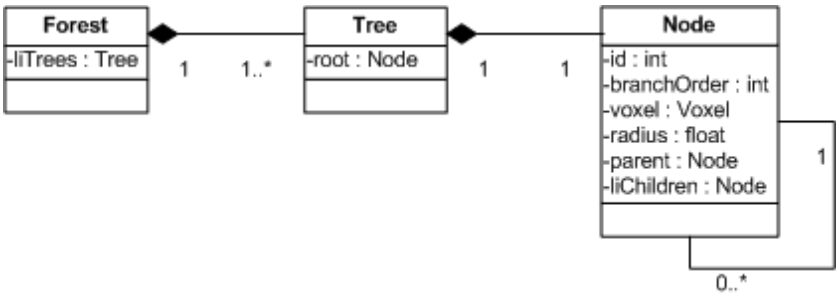


Abbildung 4.12: Vereinfachte Darstellung der Datenstruktur als UML-Diagramm

Die Knoten sind der Teil der Datenstruktur, die vom Benutzer explizit und interaktiv angelegt und manipuliert werden. Dabei bestimmt die Reihenfolge des Setzens den hierarchischen Aufbau. Die Strecke zwischen zwei Knoten definiert ein Wurzelsegment. Pro gesetztem Knoten wird symbolisch eine Kugel in der Darstellung gezeichnet und jedes Wurzelsegment wird durch einen Kegelstumpf repräsentiert. Der Anwender kann die Knoten an eine beliebige Stelle im Raum platzieren, sollte sie aber sinnvollerweise zumindest an jede Verzweigung setzen. Jeder Knoten speichert zudem eine Anzahl von Attributen, die zur Bestimmung der morphologischen Informationen herangezogen werden. Jedem Knoten wird eine Identifizierungsnummer zugewiesen. Seine Lage im Raum wird durch Voxelkoordinaten definiert und der Radius eines Knotens bestimmt die Größe der darstellenden Kugel. Die Zweigordnung gibt an, in welchem Haupt- oder Teilbaum sich der Knoten befindet. Weitere Attribute dienen der internen Verarbeitung. Die Funktionalitäten, die auf diese Datenstrukturen angewendet werden können, werden in Abschnitt 4.4.4 erläutert.

4.4.2 Verarbeitung der eingehenden Sensorwerte

Die Verwendungsart der Eingabegeräte hängt von dem in der System-Konfigurationsdatei spezifizierten Display System ab. Durch Auslesen dieses Display Modus kann im entwickelten Programm entsprechend reagiert werden. Für beide Darstellungsweisen gilt, dass zur Sammlung der ankommenden Informationen der Sensoren der Eingabegeräte jeweils eine entsprechende Klasse erstellt werden muss, die diese Informationen zur Weiterverarbeitung zur Verfügung stellt. Das Objekt der Klasse muss zunächst mit Hilfe einer „Object Registry“ dem Netzwerk für die *ViSTA*-Datenflüsse (DFN) bekannt gemacht werden. Dadurch werden die Daten der in den *XML*-Interaktionsdateien konfigurierten Sensoren erst an das entsprechend registrierte Objekt weitergeleitet.
Bei der monoskopischen Darstellung übernimmt die gewöhnliche Maus die Navigation der Sichtweise auf die Wurzelstruktur. Die entsprechenden Transformationen werden auf einen so genannten „TransformNode“ des Szenengraphen angewendet. Dem TransformNode unterliegen alle Knoten bzw. (Wurzel-)Objektteile, sodass diese gleichermaßen mittransformiert werden. Die Bewegung des Zeigers geschieht mit Hilfe des *Spacenavigators*. Das entsprechend registrierte Objekt liefert hier die zu seiner vorherigen Lage relativen Positionskoordinaten und den Status, ob ein Button des *Spacenavigators* gedrückt wurde. Bei der stereoskopischen Darstellung übernimmt der *Gyrostick* die Bewegung des Zeigers und die Reaktion auf Knopfdrücke. Innerhalb der Software liefert das dafür registrierte Objekt neben der Position des Sticks im Weltkoordinatensystem auch seine Orientierung. Dadurch kann der dreidimensionale Zeiger beispielsweise durch ein Kippen im Handgelenk mit der Spitze nach oben oder unten zeigen. Zur Veränderung der Lage der abgebildeten Wurzelstruktur werden die 6 DOF des *Spacenavigators* eingesetzt. Die daraus gelieferten Positions- und Orientierungsangaben werden dazu verwendet die Transformationsmatrizen des TransformNodes zu berechnen. Bei der Berechnung der Matrizen wird ein vom Anwendungsprogrammierer festgelegtes „Bewegungsverhalten“ berücksichtigt, das bestimmte Translationen und Rotationen einschränken oder erlauben kann. Zum Beispiel könnte darüber ein Spaziergang simuliert werden, bei dem die Verschiebung in y-Richtung und die unnatürliche Körperdrehung um die z-Achse gesperrt sind. Bei dem hier definierten Bewegungsverhalten hält der Benutzer das Objekt sozusagen in der Hand. Das Objekt bewegt sich in die selbe Richtung wie der *Spacenavigator*. Dabei wird angenommen das das Zentrum der Rotation im Zentrum des Objekts liegt. Dies kann ggf. mit Hilfe einer anfänglichen Translation des Objekts erreicht werden. Bei diesem eingesetzten Bewegungsverhalten sind folglich alle Translationen und Rotationen erlaubt. Während des Testens dieses Bewegungsmodus fiel auf, dass besonders die Translation in z-Richtung, das heißt das Zoomen, nicht immer zu einer angenehmen Bedienung führte. Deswegen wurde zusätzlich ein Bewegungsverhalten implementiert, dass statt einer Translation in z-Richtung eine Skalierung des Objekts durchführt. Der Benutzer kann dann zwischen diesen beiden Navigationsmodi während der interaktiven Wurzelmanipulation je nach Belieben wechseln. Zusätzlich zum Navigationsverhalten wird ein Event Handler implementiert, der den TransformNode des Szenengraphen mit der aktualisierten Transformationsmatrix belegt sobald eine Änderung in den ankommenden Daten stattgefunden hat.
Die eingehenden Daten der Tastatur werden in einer Klasse verarbeitet, die sowohl für die

monoskopische als auch für die stereoskopische Darstellung verwendet wird. Hier können die gedrückten Tasten bzw. Modifizierer abgefragt und darauf in unterschiedlicher Weise reagiert werden. Sie werden dazu verwendet einen bestimmten Modus für die Manipulation der rekonstruierten Wurzelstruktur zu setzen oder Manipulationsaktionen direkt auszuführen. Ein Übersicht über alle Tastenbelegungen befindet sich in Anhang A.2.

4.4.3 Zeichenroutinen

Das Zeichnen wird in der entwickelten Software mit *OpenGL*-Routinen realisiert, die vom *ViSTA*-System einmal pro Frame aufgerufen werden. Bei jedem Darstellungsaufruf werden vier Elemente unter Beachtung der vom Spacing abhängigen Skalierung auf den Bildschirm gezeichnet: der Messdatensatz, eine Konturbox um den Datensatz zur Orientierung des Benutzers im Raum, das manuell rekonstruierte Wurzelsystem und der mit einem eigenen kleinen Koordinatensystem ausgestattete Mauszeiger (Abbildung 4.13).

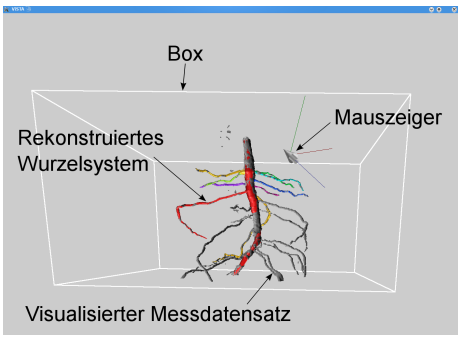
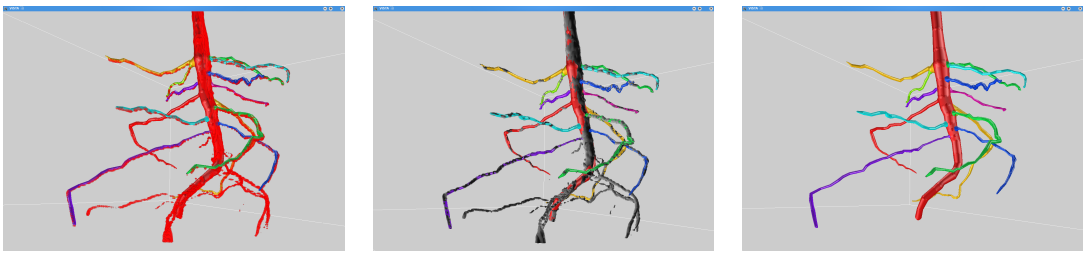


Abbildung 4.13: Elemente auf dem Display

Mit Hilfe von *OpenGL*-Routinen findet die Visualisierung des Messdatensatzes entweder per texturbasiertem oder Marching Cubes Verfahren statt. Dadurch können die Vorteile der beiden Methoden nach Belieben genutzt werden. In die visualisierte Struktur wird das rekonstruierte Wurzelsystem gezeichnet. Durch Tastendruck kann der Benutzer zwischen den Ansichten der beiden Verfahren wechseln. Zudem kann er die Wurzelstruktur des Messdatensatzes auch vollständig ausblenden und sich so nur die eigene rekonstruierte Version anzeigen lassen. Die möglichen Ansichten sind in Abbildung 4.14 aufgezeigt.
Zum Zeichnen der nachgebildeten Wurzelstruktur wird die Wald-Baum-Knoten-Datenstruktur rekursiv durchlaufen. Für jeden Knoten wird die Farbe anhand der Verzweigungsordnung gesetzt und eine Kugel, deren Mittelpunkt durch die Voxelkoordinaten des Knotens definiert wird, mit dem als Attribut gespeicherten Radius gezeichnet. Für alle Nicht-Wurzelknoten wird zusätzlich ein Kegelstumpf zum Vaterknoten mit den entsprechenden Radien gezeichnet. Diese Kegelstümpfe symbolisieren die Wurzelsegmente.
Alle gezeichneten Objekte unterliegen dem TransformNode des in *ViSTA* verwendeten Szenengraph. Deswegen würden alle Transformationen, die durch Navigation mit dem *Spacenavigator* durch den Raum entstehen, auch auf den dreidimensionale Zeiger angewendet werden. Diese Positionsänderung des Mauszeigers ist jedoch unerwünscht, da sich die Zeigerlage nur durch Bewegung des *Gyrosticks* verändern soll. Das heißt jedoch, dass



(a) Ansicht: texturbasiertes Verfahren (b) Ansicht: Iso-Oberflächendarstellung (c) Ansicht: rekonstruiertes Wurzelsystem

Abbildung 4.14: Wechsel zwischen drei Ansichten

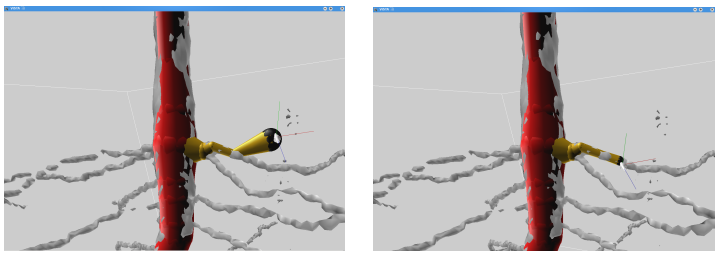
zum einen die ankommenden Positionsdaten des *Gyrosticks*, die im Weltkoordinatensystem liegen, so in das transformierte Bildkoordinatensystem abgebildet werden müssen, dass ein neu erzeugter Knoten an der Stelle des Zeigers korrekt bezüglich der 3D Wurzelstruktur eingeordnet werden kann. Zum anderen muss der Zeiger aber an der von der Hardware gelieferten Position mit der entsprechenden Orientierung im Weltkoordinatensystem gezeichnet werden. Um beide Bereiche zu berücksichtigen, werden die ankommenden Weltkoordinaten der Zeigerposition zunächst mit Hilfe der im Szenengraph gespeicherten Transformation angepasst, sodass Manipulationen der Knoten innerhalb des Wurzelsystems korrekt ausgeführt werden. Vor dem Zeichnen des Zeigers wird dann zunächst die Transformation, die durch die Bewegung des *Spacenavigators* entstanden ist, für das Zeiger-Objekt rückgängig gemacht bzw. invertiert. Dadurch stimmt für die Zeit der Zeigerdarstellung die Lage und Orientierung des Bildkoordinatensystems – veränderbar mit dem *Spacenavigator* – mit der Lage und Orientierung des Weltkoordinatensystem, das der *Gyrostick* verwendet, überein. Nun kann der Zeiger mit seiner Orientierung an den ursprünglich gelieferten Weltkoordinaten dargestellt werden.

4.4.4 Erzeugung und Manipulation der Wurzelstruktur

Zur interaktiven Erzeugung und Manipulation der Wurzelstruktur stehen die Tastatur und der *Gyrostick* zur Verfügung. Zunächst wird mittels Tastatur der Manipulationsmodus gesetzt, der bis zu einer Modusänderung aktiv bleibt. Es besteht die Möglichkeit einen Knoten zu erzeugen, einen vorhandenen Knoten auszuwählen oder zwei vorhandene Knoten auszuwählen. Durch Drücken des linken Buttons des *Gyrosticks* wird die Aktion, die sich hinter dem gesetzten Manipulationsmodus verbirgt, ausgelöst. Desweiteren werden bestimmte Aktionen direkt auf Tastendruck der Tastatur durchgeführt, teilweise auch unabhängig vom Manipulationsmodus. Dazu zählen beispielsweise das Deselektieren aller Knoten, das Umschalten der Ansicht des Wurzelsystems oder das Speichern der Wurzelstruktur. Die Tabelle in Anhang A.2 liefert eine Übersicht über alle möglichen Manipulationsaktionen, die auch in den folgenden Abschnitten erläutert werden.

Erzeugung von Knoten

Wurde der Erzeugungsmodus vor dem Drücken des linken Buttons des *Gyrosticks* gewählt, entsteht an der Zeigerspitze ein neuer Knoten, der durch eine schwarze Kugel dargestellt wird. Solange die Maustaste gedrückt bleibt, kann der Knoten beliebig im Raum bewegt werden. In Abbildung 4.15 ist die Verschiebung eines erzeugten Knotens verdeutlicht.
Wurde zuvor kein anderer Knoten ausgewählt, wird durch den Druck des Buttons nicht nur ein neuer Knoten, sondern auch ein neuer Baum angelegt. Der erstellte Knoten repräsentiert den Wurzelknoten des Baums. In jedem anderen Fall wird der erzeugte Knoten als Kindknoten des vorher ausgewählten Knotens aufgefasst. Dadurch entsteht ein Wurzelsegment, dass durch eine Verbindungslinie zwischen den Knoten dargestellt wird. Folglich ist die Reihenfolge des Setzens der Knoten von Bedeutung. Diese sollte analog zu der Hierarchie des Wurzelsystems geschehen. Andere Funktionalitäten helfen jedoch, falls die Vater-Kind-Beziehung später geändert werden muss. Nach dem Loslassen der Maustaste, wird der neu erzeugte Knoten zum ausgewählten Knoten. Da der Erzeugungsmodus allerdings weiterhin aktiv bleibt, wird dieser Knoten bei einem weiteren Buttondruck zum Vaterknoten und ein weiterer Kindknoten erzeugt.
Die Größe der Kugel eines Kindknotens hängt von seinem Radius ab. Zunächst wird der Knoten mit einem Standardradius kreiert. Durch Betätigung des rechten Buttons wird der Radius der Knotenkugel interaktiv angepasst. Hierbei erzeugt eine Bewegung in Richtung der positiven x-Achse die Vergrößerung des Radius und eine Bewegung in die entgegengesetzte Richtung die Verkleinerung der Kugel. Der Radius ist auf einen Maximal- bzw. Minimalwert beschränkt.



(a) Erzeugung eines neuen Kindknotens

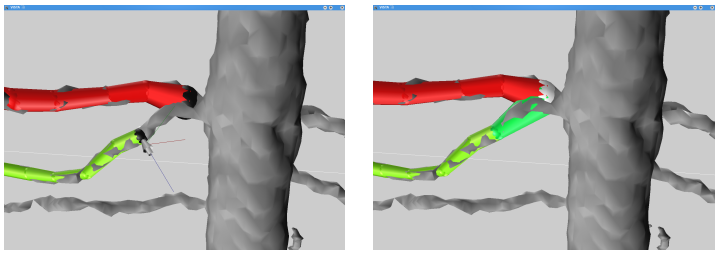
(b) Veränderung von Position und Radiusgröße

Abbildung 4.15: Manipulation eines neu erzeugten Knotens

Bei der Positionswahl des neuen Knotens und seiner Radiusveränderung können Restriktionen eintreten. So wird eine Erzeugung von Knoten innerhalb anderer Knoten nicht zugelassen und auch die Vergrößerung eines Radius über den Mittelpunkt eines anderen Knotens hinweg kann nicht ausgeführt werden. Eigentlich wäre auch die Erzeugung von Knoten innerhalb von Wurzelsegmenten oder die Durchdringung von zwei Wurzelsegmenten nicht sinnvoll. Allerdings bedarf diese Einschränkung komplexe Berechnungen mit Schnitten im dreidimensionalen Raum, weswegen hier darauf verzichtet wurde und die korrekte Handhabung in diesem Fall dem Anwender überlassen wird.

Auswahl von Knoten

Soll ein weiterer Verzweigungsast erstellt werden, muss zunächst der bereits vorhandene Vaterknoten ausgewählt werden. Dies geschieht, indem der Benutzer mit Hilfe der Tastatur den Auswahlmodus setzt. Bei Druck der linken Maustaste des *Gyrosticks* wird nun ermittelt, welcher der im Raum bereits erzeugten Knoten den Koordinaten der Zeigerspitze am nächsten liegt und dieser Knoten wird selektiert. Nun kann durch Wechsel in den Erzeugungsmodus ein neuer Ast erstellt werden.
Ebenfalls bietet der Auswahlmodus dem Benutzer die Möglichkeit die Eigenschaften des ausgewählten Knotens nachträglich zu ändern. Bleibt die Maustaste bei Auswahl eines Knotens gedrückt, kann die Position des Knotens im Raum interaktiv und relativ zu der Bewegung des Zeigers verändert werden. Auch die Größe der Knotenkugel kann nachträglich – wie im vorherigen Abschnitt beschrieben – verändert werden. Hier gilt ebenfalls die Einschränkung, dass keine Knoten in andere Kugeln geschoben werden können.
Neben der Selektion eines einzelnen Knotens existiert auch die Möglichkeit zwei Knoten gleichzeitig auszuwählen. Dazu wird der erste Knoten wie beschrieben markiert. Vor Auswahl des zweiten Knotens muss zunächst der Manipulationsmodus auf „Auswahl zweiter Knoten“ gesetzt werden und danach die entsprechende Kugel durch Druck des linken Mausbuttons selektiert werden (Abbildung 4.16(a)). Sind nun die beiden gewünschten Knoten markiert, kann durch Drücken der Enter-Taste eine von zwei möglichen Aktionen ausgeführt werden: Existiert zwischen den beiden Kugeln keine direkte Verbindung, entsteht zwischen ihnen ein Wurzelsegment, wobei der zuerst gewählte Knoten den Vaterknoten und der als zweites gewählte Knoten den Kindknoten repräsentiert (Abbildung 4.16(b)). Sollte der neue Kindknoten selber Kinderknoten besitzen, bleibt er gleichzeitig Vater dieser Kinder. War dieser Knoten zuvor einem anderen Vaterknoten zugeordnet, wird diese Verbindung gelöscht. Falls beide ausgewählte Knoten jedoch im gleichen Baum liegen und der zweitgewählte in der Verzweigungshierarchie höher liegt, wird die Erstellung des Wurzelsegments zwischen den beiden Knoten nicht zugelassen, da dadurch ein Kreis entstehen würde und die Baumhierarchie nicht mehr gewährleistet sein würde. Falls zwischen den beiden selektierten Knoten jedoch bereits eine direkte Verbindung existiert, verursacht das Drücken der Enter-Taste die Löschung dieses Wurzelsegments. Dadurch wird der vorherige Kindknoten zu einem root-Knoten eines neuen Baums. Seine Kinder bleiben ihm erhalten. Auch der vorherige Vaterknoten behält seine restlichen Kinder.

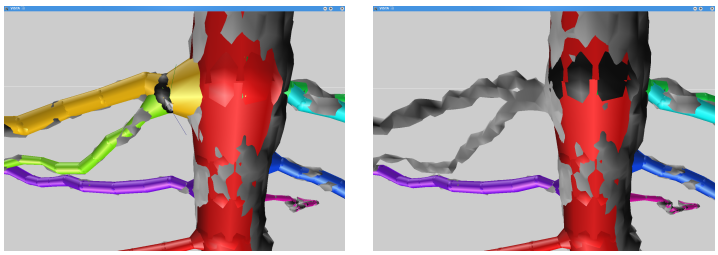


(a) Auswahl des zweiten Knotens (b) Segment hinzufügen

Abbildung 4.16: Hinzufügen eines Segments

Löschen von Knoten

Beim Löschen von Knoten wird zwischen dem Löschen eines einzelnen Knotens und dem Löschen des Knotens inklusive seines Teilbaums unterschieden. Zum Löschen eines Knotens muss zuvor kein spezieller Modus gesetzt sein. Allerdings muss der zu löschende Knoten ausgewählt sein.
Durch Drücken der Entfernen-Taste wird der einzelne Knoten gelöscht. Die Kinderknoten dieses Knotens gehen dabei jedoch nicht verloren, sondern werden zu direkten Kinderknoten des Vaters des zu löschenden Knotens. Ein root-Knoten mit Teilbäumen kann jedoch auf diese Weise nicht gelöscht werden, da die Kinderknoten keinem Vater zugeordnet werden können. Hier muss die Aktion „Löschen mit Teilbaum“ durchgeführt werden, die durch Drücken der Entfernen-Taste + Shift ausgelöst wird und sich wie in Abbildung 4.17 auswirkt. Auch Nicht-Wurzelknoten mit Teilbäumen können so gelöscht werden.

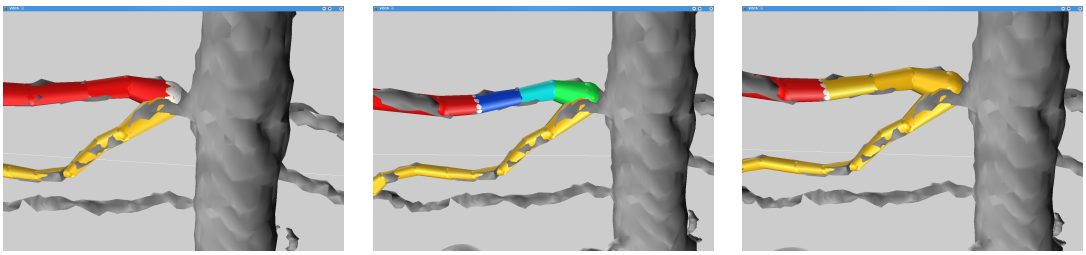


(a) Auswahl eines Knotens (b) Wurzelstruktur nach Löschen mit Teilbaum

Abbildung 4.17: Löschen des ausgewählten Knotens mit Teilbaum

Weitere Knotenmanipulationen

Um einen neuen Baum zu beginnen, darf beim Wechsel in den Erzeugungsmodus kein Vaterknoten ausgewählt sein. Dies wird erreicht, indem zuvor alle Knoten mit der entsprechenden Taste deselektiert werden.
Eine weitere Manipulationsmöglichkeit bezieht sich auf die Reihenfolge des Knotensetzens. Wurde beim Anlegen der Knoten die korrekte hierarchische Struktur der Wurzel missachtet, kann diese nachträglich korrigiert werden: Ein Knoten der in der Hierarchie nicht ganz oben steht, kann hinterher zum root-Knoten gemacht werden. Dazu muss dieser Knoten ausgewählt werden und die entsprechende Aktion mit dem Keyboard ausgelöst werden (Ergebnis siehe Abbildung 4.18(b)). Die Kinderknoten dieses Knotens bleiben erhalten, jedoch werden zusätzlich die Knoten mit Verbindungen in aufsteigender Hierarchie ebenfalls zu Kindern gemacht. Dadurch gehen keine Wurzelsegmente verloren.
Die Platzierung innerhalb der Baumhierarchie gibt zudem die Zweigordnung des Knotens an. Diese bestimmt die Farbe des Knotens und des entsprechenden Segments, die dem Benutzer hilft, den Überblick über ein weitreichendes Wurzelsystem zu behalten. Standardmäßig werden nacheinander acht verschiedene Farben des HSV-Farbraums auf die Baumäste verteilt. Die Farbaufteilung und die Anzahl verschiedener Farben kann mittels



(a) Anfängliche Wurzelstruktur (weißer Knoten = Wurzelknoten) (b) Konvertierung des mittleren Knotens zum Wurzelknoten (c) Anpassung der Verzweigungsordnung → Zweigfarbänderung

Abbildung 4.18: Nachträgliche Veränderung der Knotenhierarchie

Eingabeparameter gesteuert werden.
Während der Manipulationen kann durch Löschen oder Änderung der Knotenhierarchie die Zweigordnung verändert werden. Diese kann durch einen Tastendruck des Keyboards korrigiert werden (vergleiche Abbildung 4.18(c)). Diese Anpassung wurde nur als Option implementiert, da die mögliche Farbänderung der Wurzelsegmente auch die Übersicht des Benutzers negativ beeinflussen kann.

4.4.5 Steuerung durch Aufrufargumente

Das Verhalten der Visualisierung und anderer Eigenschaften kann von „außen“ gesteuert werden. Hierzu können eine Reihe von Argumenten beim Aufruf der entwickelten Software angegeben werden.
Als obligatorischer Parameter muss die Datei im VTK-Legacy-Format angegeben werden, die die Daten der Messungen enthält. Ebenfalls sollten die Transferfunktion für die Darstellung des Messdatensatzes mit Texturslices, sowie ein Isowert für die Visualisierung mit dem Marching Cubes Algorithmus angegeben werden. Anderenfalls werden Standardwerte für diese Eigenschaften verwendet, die nur selten ein gutes Ergebnis liefern. Desweiteren kann der durch eine Zahl repräsentierte Farbmodus und die Anzahl der verschiedenen Farben von außen geändert werden. So könnten beispielsweise die Wurzelsegmente in 16 unterschiedlichen Farben, die sich nur im Farbbereich von Gelb bis Rot befinden, eingefärbt werden.
Weitere Parameter beziehen sich auf die Erzeugung der morphologischen Informationen, die in Unterkapitel 4.5 erläutert wird. Bei der Extraktion dieser Daten dürfen Wurzelsegmente beispielsweise standardmäßig nicht länger als 1 cm sein, da sonst das Simulationsprogramm *R-SWMS* nicht korrekt funktioniert. Soll diese maximale Segmentlänge geändert werden, kann der entsprechende Wert durch einen Parameter beim Aufruf angegeben werden. Ebenso kann die Dichte der Wurzeln, die für die Berechnung der Wurzelmasse benötigt wird, von außen angepasst werden. Bei der Bestimmung der morphologischen Eigenschaften ist auch die Zuordnung der Koordinatenachsen von Bedeutung, da beispielsweise die Gravitationskraft nur in Richtung der negativen z-Achse wirkt. Durch einen optional angebbaren Faktor können die Werte auf der Koordinatenachse nachträglich

korrekt skaliert werden.
Ein letzter optionaler Parameter nimmt die Spezifikation einer *XML*-Datei auf, die ein bereits rekonstruiertes Wurzelsystem enthält. Mit Hilfe dieses Arguments wird die angegebene Wurzelstruktur eingelesen und kann danach weiter manipuliert werden (vergleiche Abschnitt 4.4.6).
Eine tabellarische Übersicht aller Aufrufargumente ist in Anhang A.1 aufgelistet.

4.4.6 Speichern und Laden

Die entwickelte Software stellt dem Benutzer die Möglichkeit zur Verfügung ein rekonstruiertes Wurzelsystem zu speichern und später wieder zu laden. Beim Schreiben des nachgebildeten Wurzelsystems bietet das entwickelte Programm zwei Möglichkeiten. Zum einen kann die Wurzelstruktur in einer Datei im *R-SWMS*-Format gespeichert werden, die für nachfolgende Simulationen zur Vorhersage der Bodenwasseraufnahme benötigt wird. Zum anderen können die Informationen der Wurzelstruktur während oder nach der Rekonstruktion in einer *XML*-Datei gesichert werden.
Das Problem bei der Erzeugung der Ausgabedatei im *R-SWMS*-Format ist, dass wichtige Informationen eines Wurzelknotens dabei verloren gehen bzw. nur indirekt aufgenommen werden. Beispielsweise fließt der Radius einer Wurzelkugel, der zwingend für die korrekte Darstellung der Wurzelstruktur benötigt wird, nur in Form der Oberflächenangabe eines Wurzelsegments ein und wird nicht explizit angegeben. Dies führt dazu, dass beim Auslesen so einer Datei mit der entwickelten Software nicht alle Attribute rekonstruiert werden könnten. Auch würde das Auslesen einer Datei im *R-SWMS*-Format speziell auf das Format angepasste Methoden erfordern, was einen Mehraufwand bei der Implementierung bedeutet. Ein weiteres Problem des *R-SWMS*-Formats besteht darin, dass es keine Darstellungsmöglichkeit gibt, mehrere Bäume zu speichern, wodurch der Benutzer selbst bei Zwischenspeicherungen einer Wurzelnachbildung gezwungen wäre stets vorher seine Strukturen auf einen einzigen Baum zu reduzieren. Aufgrund dieser Einschränkungen, die das fest vorgegebene *R-SWMS*-Format mit sich bringt, wurde die Entscheidung getroffen, ein zweites Ausgabeformat für die Wurzelstrukturen anzubieten, das für ihre interne Verarbeitung genutzt wird.
Zu der internen Beschreibung der gespeicherten Wurzeldaten wird *XML* verwendet, da diese Auszeichnungssprache zur Darstellung von hierarchisch strukturierten Daten in Form von Textdaten besonders gut geeignet ist. Desweiteren ist diese Art der Abbildung der Daten auch für den Menschen lesbar, sodass er ggf. kleine Veränderungen an den Werten vornehmen könnte. Größter Vorteil der *XML*-Spezifikation ist jedoch, dass ein standardisierter Zugriff auf Elemente und Attribute existiert und dementsprechend für verschiedene Programmiersprachen mehrere Toolkits zum Parsen der Datenstrukturen vorhanden sind. Im Rahmen dieser Arbeit wurde das Tool *Tiny XML* [37] für C++ verwendet, da dieses bereits in der *ViSTA*-Architektur eingebettet ist, sodass keine neuen Bibliotheken hinzugefügt werden müssen. *Tiny XML* erzeugt aus dem *XML*-Dokument ein Document Object Model (DOM), das heißt einen hierarchischen Baum, dessen Eigenschaften gelesen, verändert und wieder abgespeichert werden können. *Tiny XML* ist jedoch kein validierender *XML*-Parser. Dadurch kann die Wohlgeformtheit einer *XML*-Datei, die mit Hilfe von

XML-Schemas (XSD) spezifiziert werden kann, nicht geprüft werden. Da hier allerdings davon ausgegangen wird, dass die *XML*-Dateien nur durch das Programm erzeugt werden und normalerweise nicht manuell vom Benutzer verändert werden müssen, spielt die nicht vorhandene Fehlerprüfung nur eine geringfügige Rolle.

```
<?xml version="1.0" ?>
<Forest id="9" bo="5" cm="1023" cn="8" d="1.000000" dlx="6.000000" dly="6.500000"
    dlz="10.000000" sl="1.000000" zFac="1.000000" iso="16.000000"
    dataset="data/test.vtk" transferFunc="data/test.xml">
 <Tree id="1">
  <Node id="0" bo="1" rad="0.013999" x="0.006773" y="-0.024925" z="-0.498762">
   <Node id="1" bo="1" rad="0.015601" x="0.014065" y="-0.030727" z="-0.401572">
    <Node id="2" bo="1" rad="0.018821" x="0.012611" y="-0.037531" z="-0.345622">
     <Node id="3" bo="1" rad="0.019668" x="0.013073" y="-0.046178" z="-0.300999" />
     <Node id="4" bo="2" rad="0.019299" x="0.013558" y="-0.050667" z="-0.282027" />
    </Node>
   </Node>
  </Node>
 </Tree>
 <Tree id="2">
  <Node id="5" bo="3" rad="0.020000" x="0.015870" y="-0.065867" z="-0.152657">
   <Node id="6" bo="3" rad="0.017759" x="0.023878" y="-0.070524" z="-0.119282" />
   <Node id="7" bo="4" rad="0.016746" x="0.026674" y="-0.076750" z="-0.084658" />
   <Node id="8" bo="5" rad="0.017809" x="0.028325" y="-0.080281" z="-0.042092" />
  </Node>
 </Tree>
</Forest>
```

Listing 4.6: XML-Datei einer vereinfachten Wurzelstruktur

Beim Speichern der Wurzelnachbildung im *XML*-Format bildet – analog zur Datenstruktur – das `Forest`-Tag das umschließende Modul, das als Attribute alle wichtigen Eigenschaften und Informationen aufnimmt, die größtenteils den Aufrufargumenten der Software ähneln. Alle Bäume werden durch `Tree`-Tags eingeleitet, die übersichtshalber mit einer ID nummeriert werden. Das erste `Node`-Tag in jedem Baum repräsentiert den entsprechenden Wurzelknoten. Jeder Knoten kann beliebig viele Kinderknoten besitzen, die analog zum Aufbau der Wurzelstruktur hierarchisch in der *XML*-Datei angeordnet werden. Jedes `Node`-Tag speichert die wichtigen Knoten-Attribute: `id` beschreibt die eindeutige Identifikationsnummer eines Knotens und `bo` seine „branch order“ (Verzweigungsordnung). Durch `rad` wird der Radius der Knotenkugel angegeben und `x`, `y`, `z` definieren die Voxeldaten des Mittelpunkts der Knotenkugel. Die Werte für die Voxelkoordinaten befinden sich im Bereich $[-0,5;\ 0,5]$, da die Größenordnung der internen Skalierung auf $[0,1]$ verwendet wird. Die Radiusgröße wird entsprechend dieser Skalierung angepasst. Die globalen Attribute des Waldes enthalten ebenfalls eine ID und eine Verzweigungsordnung, die sich jedoch auf das gesamte System beziehen und somit die Anzahlen der insgesamt im Wald enthaltenden Knoten bzw. Zweige angeben. `cm` und `cn` geben den „colour mode“ (Farbmodus) und die Anzahl der verschiedenen Farben an. Im zum Beispiellisting 4.6 gehörenden Wurzelsystem wurden acht Farben aus dem kompletten HSV-Farbraum (`cm=1023`) verwendet. Die Dichte der Wurzeln, angegeben durch `d`, liegt hier bei 1 $\left[\frac{g}{cm^3}\right]$. Die Dimensionen des Messcontainers waren 6 cm x 6,5 cm x 10 cm und die maximale Länge eines Wurzelsegments, abgekürzt `sl`, darf später 1 cm nicht überschreiten. Auch wird der Dateiname des Datensatzes angegeben, auf den sich die Wurzelnachbildung bezieht. Mit `zFac` wird angegeben

mit welchem Faktor die z-Koordinaten der Knoten multipliziert werden bevor sie in das *R-SWMS*-Format umgewandelt werden. `iso` und `transferFunc` geben den Isolevel und die Datei mit der Transferfunktion für das Volume Rendering an.
Die in einer *XML*-Datei gespeicherte Wurzelstruktur kann bei Aufruf des im Rahmen dieser Arbeit entwickelten Programms wieder eingelesen und zum Beispiel weiter manipuliert werden. Beim Laden der *XML*-Datei werden neben der Wiederherstellung der Datenstrukturen auch alle dort spezifizierten Werte der Knotenattribute für die Knoten-Objekte der Software übernommen. Die Attribute des Waldes werden jedoch im Programm nur teilweise gesetzt. Übernommen werden die globale ID und Verzweigungsordnung, sowie der Farbmodus und die Anzahl der verschiedenen Farben. Auch die maximale Länge eines Wurzelsegments und die Dichte werden wieder eingelesen. Der Faktor für die Werte der Koordinatenachse und der Isowert werden ebenfalls im Programm gespeichert. Dagegen dient die Angabe des Dateinamens des Originaldatensatzes und die Angabe der Transferfunktion nur zur Information. Diese Daten werden nicht im Programm gesetzt, sondern nur auf der Konsole zu Informationszwecken ausgegeben, weswegen sie beim Programmstart als Kommandozeilenparameter aufgeführt werden sollten. Die Dimensionslängen des Messcontainers (`dlx`, `dly`, `dlz`) werden aus den Headerdaten des VTK-Legacy-Datensatzes berechnet. Sie dienen ebenfalls nur zur Information.
Wird eine *XML*-Datei mit einem rekonstruierten Wurzelsystem beim Aufruf der entwickelten Software spezifiziert, können dennoch weitere der in Abschnitt 4.4.5 erläuterten Aufrufargumente angegeben werden. Wird dabei ein Parameter spezifiziert, der auch beim Einlesen der *XML*-Wurzeldatei gesetzt werden würde, hat der Wert des Kommandozeilenarguments eine höhere Priorität und überschreibt den der *XML*-Datei. So kann das Verhalten und Aussehen auch von geladenen Wurzelstrukturen weiterhin von außen gesteuert werden und es ist kein Eingriff in den Code der *XML*-Datei nötig.

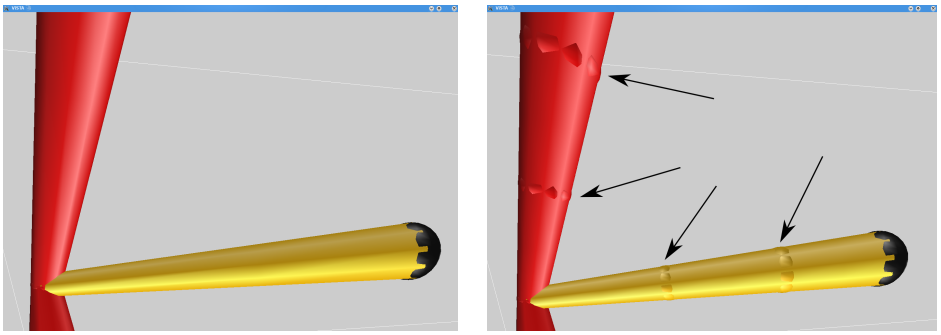
4.5 Extraktion der morphologischen Informationen

Das *R-SWMS*-Format der Ausgabedatei enthält alle wichtigen morphologischen Informationen, die für die Simulationen zur Bodenwasseraufnahme benötigt werden. Bevor diese Datei jedoch erzeugt werden kann, muss das rekonstruierte Wurzelsystem einige Voraussetzungen erfüllen, die in Unterkapitel 4.5.1 behandelt werden. Beim Rausschreiben der Datei müssen dann die gespeicherten Informationen auf das reale System skaliert werden und die fehlenden morphologischen Eigenschaften berechnet werden (Abschnitt 4.5.2).

4.5.1 Voraussetzungen an die Wurzelnachbildung

Grundvoraussetzung für die Erzeugung der Ausgabedatei im *R-SWMS*-Format ist das Vorliegen der Wurzelrekonstruktion als ein einzelner Baum. Da der Benutzer während der Erzeugung seiner Wurzelnachbildung mehrere Bäume anlegen kann, wird ihre Anzahl vor dem Rausschreiben der morphologischen Informationen geprüft. Der Benutzer muss dann ggf. alle Bäume miteinander verknüpfen, sodass die hierarchische Wurzelstruktur der Pflanze gewährleistet wird.

In der Ausgabedatei werden die Knoten bzw. die Wurzelsegmente, die durch zwei benachbarte Knoten definiert sind, durch Zuweisung von IDs bzw. Verweis auf die ID des Vaterknotens verbunden und so die Wurzelhierarchie abgebildet. Da jedoch durch Löschen und Neuerzeugung von Knoten während der Rekonstruktionsphase Lücken in der ID-Nummerierung auftreten können, werden mit einem Baum-Durchlauf vor der Informationsextraktion die IDs durch fortlaufende Zahlen korrigiert. Die gleiche Ursache oder das Abändern des hierarchischen Aufbaus von Teilbäumen kann eine falsche Durchnummerierung bei der Verzweigungsordnung der Knoten bewirken, die ebenfalls angepasst wird.
Eine Besonderheit des *R-SWMS*-Programms ist die Tatsache, dass für die Simulationen die Wurzelsegmente eine Länge von 1 cm nicht überschreiten dürfen. Aus diesem Grund wird vor dem Schreiben der Datei die Länge aller Wurzelsegmente geprüft und ggf. eine minimal benötigte Anzahl von Knoten zusätzlich eingefügt. Diese Hilfsknoten werden gleichmäßig auf der Strecke des Wurzelsegments verteilt und ihre Radien werden mit Hilfe des Strahlensatzes so angepasst, dass das ursprüngliche Wurzelsegment seine Kegelstumpf-Form beibehält. Abbildung 4.19 verdeutlicht diese Erzeugung von Hilfsknoten auf langen Wurzelsegmenten. Die zusätzlichen Knoten können nach dem Schreiben der Datei im *R-SWMS*-Format verworfen werden oder zu einer detaillierten Anpassung der Wurzelnachbildung weiter verarbeitet werden.



(a) Ausgangswurzelstruktur (b) Wurzelstruktur mit zusätzlichen Knoten

Abbildung 4.19: Wurzelsystemveränderung durch Schreiben der R-SWMS-Datei

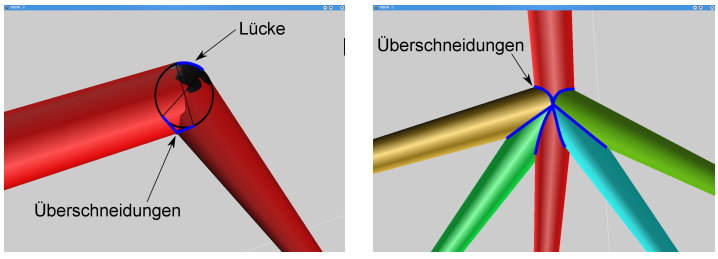
4.5.2 Ausgabedatei im R-SWMS-Format

Wenn alle Voraussetzungen des *R-SWMS*-Formats erfüllt sind, können die morphologischen Eigenschaften schließlich in eine Textdatei geschrieben werden. Da intern die Dimensionen auf einen Bereich von je -0,5 bis 0,5 skaliert wurden, müssen die räumlichen Koordinaten mit Hilfe des Spacings zunächst auf die Größenordnung des realen Systems transformiert werden. Desweiteren wird aus Konvention angenommen, dass der Wurzelknoten, das heißt der Samen der Pflanze, im Ursprung des Koordinatensystem liegt. Deswegen muss zu jedem Voxel ein Offset gerechnet werden. Auch wird bei angegebenem z-Faktor die Skalierung bzw. Richtung der z-Achse angeglichen.
Neben den Koordinaten für die x-, y-, z-Achse, den IDs und der Verzweigungsordnung,

wird die Oberfläche, das heißt die Mantelfläche jedes Wurzelsegments berechnet. Diese morphologische Eigenschaft ist besonders wichtig, da sie die Menge der Wasseraufnahme beeinflusst. Weitere Informationen sind die Länge des Wurzelsegments und dessen Masse. Für diese Berechnungen müssen sich alle (Kommandozeilen-)Angaben wie Größe des Pflanzencontainers und Dichte der Wurzel auf die gleiche Größeneinheit beziehen.
Zusätzlich zu den morphologischen Eigenschaften der Wurzelsegmente werden eine Reihe von zusammenfassenden Informationen in den Header der Datei geschrieben. Hierzu zählt beispielsweise die Anzahl der aufgelisteten Wurzelsegmente. Die *R-SWMS*-Datei des Wurzelsystems der Lupine, die in Kapitel 5 betrachtet wird, ist in Anhang C.3 zu finden.

4.6 Einschränkungen

Bei der Entwicklung der Software für diese Arbeit wurden einige Einschränkungen erkannt, die bei der Verwendung der Software zu beachten sind.
Bei der Nachbildung des Wurzelsystems wird der Benutzer daran gehindert Knotenkugeln innerhalb anderer Knotenkugeln zu erzeugen, um somit unsinnige Oberflächenberechnungen für die *R-SWMS*-Datei zu vermeiden. Allerdings können trotzdem Knotenkugeln innerhalb der zylindrischen Wurzelsegmente erstellt werden. Um dies zu verhindern, müssen Schnittberechnungen im dreidimensionalen Raum mit den Kegelstümpfen der Wurzelsegmenten gemacht werden. Da diese Berechnungen sehr aufwändig sind und angenommen wird, dass der Anwender die Wurzelstruktur „mit Verstand“ rekonstruiert, wurde auf ihre Implementierung verzichtet. Die gleiche Problematik gilt für die Berechnungen der Durchdringung von zwei Wurzelsegmenten.
Während der Wurzelnachbildung kann der Benutzer die Knotenkugeln beliebig setzen. Dadurch werden die Wurzelsegmente jedoch normalerweise nicht in einer geraden Reihe miteinander verknüpft. Folglich entsteht bei der Berechnung der Oberflächen für die *R-SWMS*-Datei ein Fehler, da sich die Mantelflächen von Wurzelsegmenten zu einem gemeinsamen Knoten zum Teil überschneiden oder Lücken bilden (Abbildung 4.20). Diese Ungenauigkeit wird aber als vernachlässigbar klein angesehen.



(a) Zweig: Lücke und Überschneidung könnten sich ausgleichen

(b) Verzweigung: mehrere Überschneidungen

Abbildung 4.20: Mögliche Fehler bei der Oberflächenberechnung der Wurzelsegmente

Eine weitere Restriktion bezieht sich auf die Handhabung bei fälschlicher Knotenerzeu-

gung, -verschiebung oder -löschung. Diese muss der Benutzer manuell rückgängig machen, da keine implementierte „Undo“-Funktion existiert. Der Anwender sollte deswegen sicherheitshalber die Wurzelstruktur gelegentlich mit Hilfe des *XML*-Formats (Abschnitt 4.4.6) speichern.
Neben diesen Software-Beschränkungen existiert ebenfalls eine kleine Einschränkung bezüglich der VR-Hardware, die die Verwendung des *Gyrosticks* betrifft. Mit Hilfe der *ViSTA*-Treiberschicht konnten nur die gewöhnlichen Maustasten, die sich auf der Oberseite des *Gyrosticks* befinden, zur Benutzung freigegeben werden. Hier wäre eine Verwendung der speziellen Tasten, die auf der Unterseite des *Gyrosticks* liegen und für die Handhabung des *Gyrosticks* in der Luft konzipiert wurden, komfortabler. Allerdings ließen sich diese speziellen Maustasten über *ViSTA* nicht ansprechen, sodass der Benutzer auf ihren Einsatz verzichten muss.

Kapitel 5

Evaluation am Anwendungsbeispiel Lupine

Die im Rahmen dieser Arbeit entwickelte Software wird mit Hilfe eines Anwendungsbeispiels evaluiert und die entsprechenden Maßnahmen in diesem Kapitel dokumentiert. Für die Prüfung der Software wird das Wurzelsystem der Lupine, die in der deutschen Landwirtschaft oft als Gründünger verwendet wird (Kapitel 5.1), betrachtet. Der MRT-Datensatz der Lupine wird im dreidimensionalen Raum visualisiert und manuell rekonstruiert. Diese Vorgänge werden in Kapitel 5.2 schrittweise beschrieben. Die Resultate der Nachbildung, das heißt die morphologischen Informationen des Lupinen-Wurzelsystems im *R-SWMS*-Format, werden mit dem Simulationsprogramm *R-SWMS* verarbeitet. Eine Auflistung der für diese Simulation wichtigen, verwendeten Parameter ist in Kapitel 5.3 zu finden. Die mit *R-SWMS* modellierte Bodenwasseraufnahme der Wurzeln wird mittels *Matlab*-Routinen visuell dargestellt und anhand dieser Abbildungen erläutert (Abschnitt 5.4). Mit Hilfe des daraus abgeleiteten Wasseraufnahmeverhalten kann eine Bewertung der Software vorgenommen werden.

5.1 Lupinen in der Landwirtschaft

Lupinen gehören zu der Familie der Hülsenfrüchte und bilden ein allorhizes Wurzelsystem aus. Es existieren weltweit etwa 200 Lupinen-Arten, wovon vor allem die Gelben (Lupinus luteus), Weißen (Lupinus albus) und Blauen oder Schmalblättrigen Lupinen (Lupinus angustifolius) landwirtschaftlich genutzt werden. Das ICG-4 betrachtet hauptsächlich die Weiße und die Blaue Lupine.
Lupinen werden – vor allem im Mittelmeergebiet – als Futterpflanzen angebaut. Ihre Samen dienen als hochwertiger pflanzlicher Eiweißlieferant für Ernährungsprodukte. In Deutschland werden hauptsächlich Blaue Lupinen angepflanzt. Als Anwendungsbeispiel wird deswegen ebenfalls die Wurzelstruktur einer Blauen Lupine betrachtet. Lupinen fallen meist durch ihre farbenprächtigen Blütenstände, die wie engständige Trauben wachsen, auf. In Abbildung 5.1 ist eine Blaue Lupine dargestellt. Die Blaue Lupine kann bis zu 80 cm hoch werden und bildet eine sehr tiefgehende Pfahlwurzel aus. Damit ist die

Abbildung 5.1: Blaue Lupine [30]

Pflanze relativ unempfindlich gegen Trockenheit [30]. Wie alle Lupinen geht die Blaue Lupine eine Symbiose mit so genannten Knöllchenbakterien ein. Dadurch wird Stickstoff in Knöllchen an den Wurzeln gebunden und der Boden mit Stickstoff angereichert. Da diese Bodenverbesserung in der Landwirtschaft erwünscht ist, wird die Lupine oft als Gründünger verwendet und während des Zwischenfruchtbaus angepflanzt. Die einjährige Lupine-Pflanze wird dann vor der nächsten Fruchtfolge untergepflügt.

5.2 Rekonstruktion der Lupine

Mit Hilfe des Magnetresonanztomographen des ICG wurde ein Volumendatensatz einer ca. 3 Wochen alten Blauen Lupine erstellt. Diese Lupine wurde in einem mit Sandboden gefüllten 10 cm x 8 cm x 5,5 cm großen Pflanzencontainer angepflanzt und mit 256 x 256 x 110 Messpunkten gesampelt.

```
-dataset data/lupine.vtk -transferFunc data/lup_red.xml -iso 16.0 -zFac -1.0
-vistaini configfiles/vista_picasso.ini
```

Listing 5.1: Kommandozeilenargumente für die Rekonstruktion der Lupine

Der resultierende Messvolumendatensatz wird zunächst ins VTK-Legacy-Format umgewandelt und die entstandene Datei der im Rahmen dieser Arbeit entwickelten Software (vergleiche Anhang C.1) beim Aufruf als Parameter übergeben. Alle verwendeten Aufrufargumente sind in Listing 5.1 zu sehen. Da die Lupine in positive z-Richtung bezüglich der MRT-Messung gewachsen ist, das Simulationsprogramm *R-SWMS* jedoch einen Wachstum in negative z-Richtung voraussetzt, werden mit Hilfe des angegebenen z-Faktor-Parameters alle z-Werte mit „-1“ multipliziert. Desweiteren werden zur stereoskopischen Darstellung der Wurzelstruktur mit dem *PI-casso*-System – festgelegt durch die Konfigurationsdatei `vista_picasso.ini` (Anhang B.2) – eine Transferfunktion und ein Isowert für die Volume Rendering Verfahren angegeben. Die spezifizierte Transferfunktion bewirkt eine Rotfärbung aller Wurzelvoxel. Der Isowert wird auf den Wert 16 von den möglichen skalaren Werten 0 - 255 festgelegt. In Abbildung 5.2 ist die Isoflächen-Visualisierung dieses Messdatensatzes zu sehen. Die Wurzelstruktur ist hier gut sichtbar und wenig verrauscht. Da die Pflanze bei der Aufnahme noch relativ jung ist, sind die Protein-Knöllchen an den

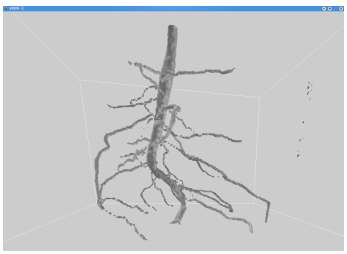
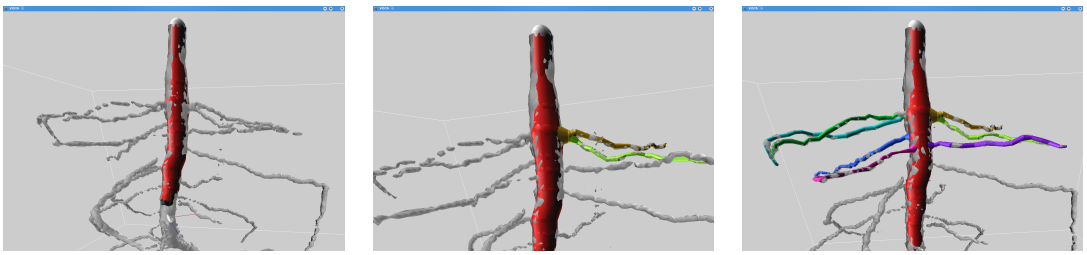


Abbildung 5.2: Visualisierung des Lupinen-Volumendatensatzes (Iso-Oberflächen)

Wurzeln noch nicht ausgebildet und deswegen nicht erkennbar.
Diese Wurzelstruktur wird nun beginnend mit der Hauptwurzel Schritt für Schritt manuell rekonstruiert. Hierbei zeigt Abbildung 5.3(a) den ersten Teil der Nachbildung der Hauptwurzel. In Abbildung 5.3(b) wurden bereits zwei Seitenwurzeln nachgezeichnet. Die bei der Rekonstruktion je nach Verzweigungsordnung zugewiesenen verschiedenen Farben sind in Abbildung 5.3(c), die die ersten acht Wurzeln zeigt, dargestellt. Ein vollständig nachgebildetes Wurzelsystem der Lupine ist in Abbildung 5.4 erkennbar. Zur Anfertigung dieser detaillierten Struktur wurden etwa 1,5 Stunden benötigt.



(a) Teil der Hauptwurzel (b) Zwei Seitenwurzeln (c) Acht Zweige

Abbildung 5.3: Schrittweise Rekonstruktion der Lupinen-Wurzel

Die komplett rekonstruierte Wurzelstruktur der Lupine wird zur internen Verarbeitung im *XML*-Format gesichert (Anhang C.2). Im letzten Schritt werden aus der Wurzelrekonstruktion die morphologischen Informationen extrahiert und in eine Datei im *R-SWMS*-Format geschrieben. Das Resultat (Listing C.3 im Anhang) zeigt auf, dass insgesamt 26 Zweige und 518 Knoten erstellt wurden.

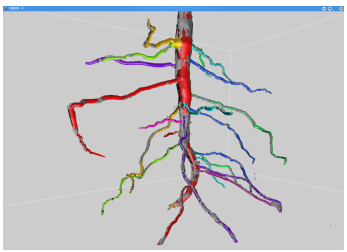


Abbildung 5.4: Vollständige Rekonstruktion der Lupinen-Wurzeln

5.3 Simulationsparameter

Die Plausibilitätsprüfung der Ergebnisse der entwickelten Software erfolgt mit Hilfe des Simulationsprogramms *R-SWMS*. Zur Simulation der Bodenwasseraufnahme des Lupinen-Wurzelsystems müssen neben den ermittelten morphologischen Informationen (wie Segmentlänge und -oberfläche) auch weitere Simulationsparameter, die den Boden, die Wurzel und ihre Wechselwirkungen beschreiben, vom Anwender spezifiziert werden.
Die Bodenwasseraufnahme wird hier über eine Zeitraum von 4 Tagen simuliert. Dabei liegt die axiale Wurzel-Leitfähigkeit k_x, die zur Berechnung des axialen Flusses J_x (vergleiche Gleichung (2.7)) benötigt wird, bei 0.0432 $\frac{\text{cm}}{\text{Tag}}$. Die radiale Leitfähigkeit k_r für den radialen Wasserfluss (Gleichung (2.6)) ist mit 0.0001728 $\frac{\text{cm}}{\text{Tag}}$ gegeben.
Als Boden wird ein homogener Sandboden simuliert. Dieser Boden ist ein reiner Modellboden. Da er das Wasser durch seine großen Poren nicht sehr lange halten kann, wird er durch eine steile Wasserspannungskurve ($h \mapsto \Theta$) charakterisiert, die durch einen großen Wert von n (hier 6.0) in der van Genuchten Gleichung (2.3) angegeben wird. Weitere Parameterwerte für das Genuchten-Mualem-Modell ((2.3) und (2.4)) sind der gesättigte Wassergehalt $\Theta_s = 0,36\ \frac{\text{cm}^3}{\text{cm}^3}$, der residuale Wassergehalt $\Theta_r = 0,04\ \frac{\text{cm}^3}{\text{cm}^3}$ und die gesättigte hydraulische Leitfähigkeit $k_s = 12\ \frac{\text{cm}}{\text{Tag}}$. Die entsprechenden empirischen Parameter sind durch $\alpha = 0,026\ \frac{1}{\text{cm}}$ und $m = 1 - \frac{1}{n} = -0,66$ gegeben.

5.4 Simulationsergebnisse

Die mit *R-SWMS* erzeugten Simulationsergebnisse werden in Ausgabedateien geschrieben und mit Hilfe von *Matlab*-Routinen (vergleiche Abschnitt 2.3.3) visuell dargestellt.

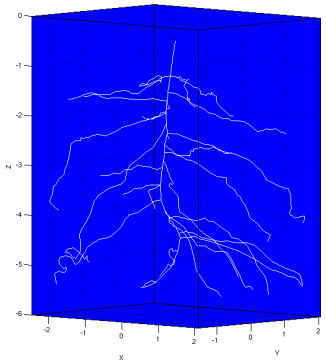


Abbildung 5.5: Wurzelsystem der Lupine in Matlab

Eine erste *Matlab*-Routine stellt auf Basis der mit der entwickelten Software erzeugten Datei im *R-SWMS*-Format das rekonstruierte Wurzelsystem der Lupine (Abbildung 5.5) dar. Aufgrund der Möglichkeit das rekonstruierte Wurzelsystem fehlerfrei mit dieser Routine darzustellen, kann geschlossen werden, dass das Format der erzeugten Datei korrekt ist und die Umskalierung in das reale Koordinatensystem funktioniert hat. Ebenfalls wurde die hierarchische Struktur korrekt in die Datei abgebildet.

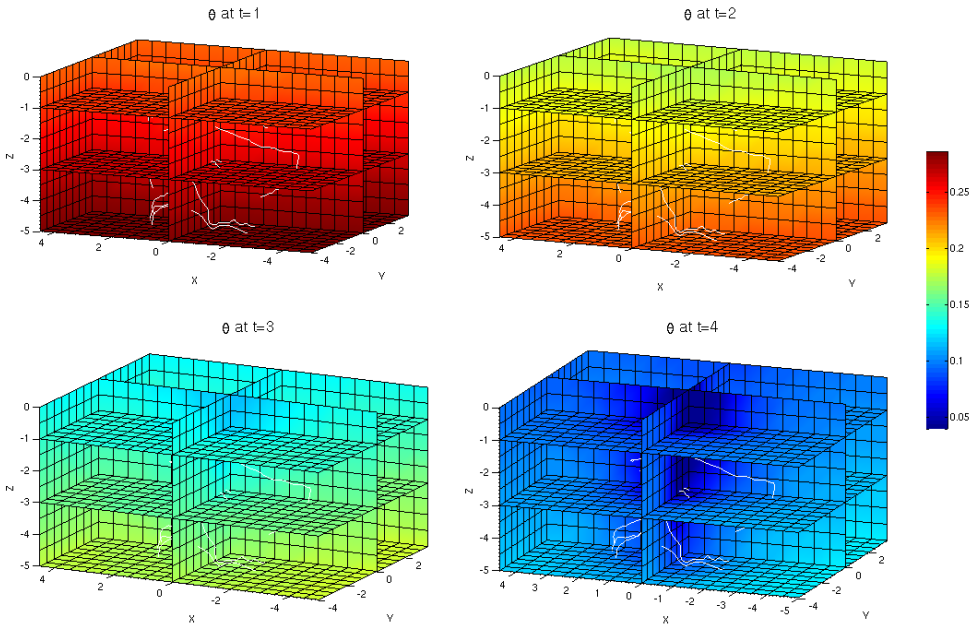


Abbildung 5.6: Verteilung des Wassergehalts Θ an den Tagen 1-4

Die extrahierten morphologischen Attribute Segmentlänge und -oberfläche werden für die Berechnungen der Bodenwasseraufnahme benötigt, wofür zunächst Wassergehalte und Wasserpotentiale ermittelt werden müssen. Der berechnete Wassergehalt des Bodens wird hier an den Tagen 1-4 betrachtet. Die Abbildung 5.6 verdeutlicht, dass der Wassergehalt in den unteren Schichten meist höher als in den oberflächennahen Bodenschichten ist. An Tag 1 ist der Boden noch relativ homogen bewässert mit einem Wassergehalt um 25 %. In den darauf folgenden Tagen nimmt der Wassergehalt stetig ab, sodass an Tag 4 der Boden nur noch einen Wassergehalt von ca. 10 % besitzt. Die relativ homogene Austrocknung des Bodens ist auf den modellierten Sandboden zurückzuführen. Dieser besitzt eine hohe Leitfähigkeit, wodurch das Wasser mittels der Saugspannung auch aus etwas entfernteren Bereichen gezogen werden kann. Ab Tag 3 ist jedoch zusätzlich eine verstärkte Bodenaustrocknung im Bereich der Hauptwurzel, die in den Abbildungen in der xy-Ebene zentriert ist, festzustellen. Diese ist besonders an Tag 4 durch den dunkelblauen „Schatten“ deutlich erkennbar. Da bei dieser Simulation weder ein „freier Abfluss“ (engl.: free drainage) nach unten, noch eine Transpirationsrate angegeben wurde, findet im Simulationssystem nur die gravitative Equilibrierung statt, sodass die beschriebene Bodenaustrocknung ausschließlich auf der Wasseraufnahme der Wurzeln beruht.
Die zeitliche Änderung des Wassergehalts ist vom Wasserpotential und vom Senkterm abhängig (vergleiche (2.5)). Dabei beschreibt der Senkterm die eigentliche Bodenwasseraufnahme der Wurzel, das heißt die Änderung des Wassergehalts pro Tag. Die Abbildungen 5.7 und 5.8 zeigen die Wasserpotentiale (in cmWS) bzw. die Senkterme der Wurzelstruktur der Lupine an Tag 4. Die Wasserpotentiale im Boden ändern sich kaum. Dagegen kann eine Erniedrigung des Wasserpotentials bis auf -1800 cmWS an der Hauptwurzel

festgestellt werden. Die Streamlines des Wassers (in grau dargestellt) verdeutlichen in der Abbildung, dass das Wasser vor allem zu diesen Bereichen der Wurzel fließt. Die Abbildung der Senkterme unterstützt diese Aussage, da im Bereich der Hauptwurzel die größten Senken entstanden sind. Kleinere Senken haben sich an den Seitenwurzeln gebildet, aber die von den Wurzeln weiter entfernten Bodenbereichen sind nach 4 Tagen noch senkenlos.

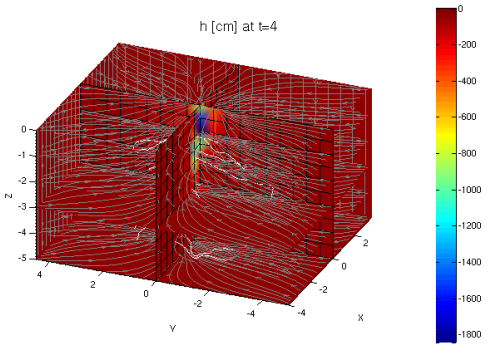


Abbildung 5.7: Wasserpotentiale h und Flusslinien an Tag 4

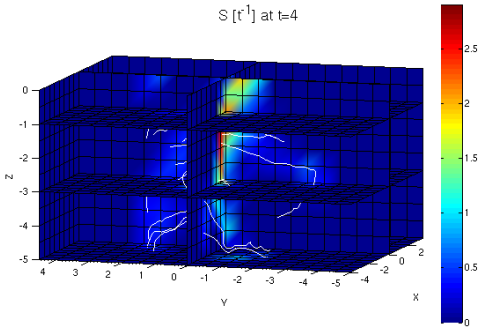


Abbildung 5.8: Senkterme S (Bodenwasseraufnahme) an Tag 4

Die Abbildungen lassen auf ein „normales“ Wasseraufnahmeverhalten des Wurzelsystems schließen: Das Wasser wird aufgrund des (bekannten) Potentialgefälles im SPAC zum Wurzelsystem der Lupine transportiert (Abbildung 5.7). Dort wird es von den Wurzeln aufgenommen, wohingegen der weiter entfernte Boden folglich senkenlos bleibt (Abbildung 5.8). Die Größenordnungen der berechneten Wasserpotentiale und Senkenterme zeigen dabei, dass auch die mit der Software ermittelten Wurzelsegmentlängen und -oberflächen im richtigen Größenbereich liegen. Diese Tatsachen weisen gemeinsam auf die Korrektheit der entwickelten Software hin.
Um das Simulationsergebnis jedoch vollständig evaluieren zu können, müsste eine experimentelle Messung der Bodenwasseraufname der Lupinen-Wurzeln als Vergleich gemacht werden. Diese lag für diese Arbeit aber leider noch nicht vor. Aus dem Vergleich der simulierten und gemessenen Daten lassen sich vor allem Rückschlüsse auf die Qualität des Simulationsmodells ziehen.

Kapitel 6

Zusammenfassung und Ausblick

Die Untersuchung der Bodenwasseraufnahme durch Pflanzenwurzeln wird durch eine ressourcenschonende Nutzung des Agrarökosystems, insbesondere der Wasser- und Bodennutzung, motiviert. Da Pflanzenwurzeln durch ihre versteckte Lage innerhalb des Bodens die „hidden half“ des Ökosystems sind, war die Analyse der Boden-Wurzel-Wechselwirkungen jedoch lange Zeit schwierig. Hier trieb die Verwendung von nicht-invasiven dreidimensionalen Messtechniken wie Magnetresonanztomographie die Forschung voran. Zudem wurden kürzlich Simulationsmodelle für die Vorhersage der Bodenwasseraufnahme durch Wurzeln entwickelt, die auf hierarchisch strukturierten Daten einer dreidimensionalen Wurzelstruktur und ihrer morphologischen Eigenschaften basieren.
Das Ziel dieser Arbeit ist die Verknüpfung dieser beiden Komponenten: Das Bindeglied wird durch eine simulationsfähige Beschreibung der Wurzelstruktur und ihrer morphologischen Informationen geschaffen, die aus den mit einem MR-Tomographen experimentell erzeugten Volumendatensätzen extrahiert wird. Diese Extraktion wird mittels einer manuellen interaktiven Rekonstruktion des gemessenen Wurzelsystems realisiert. Die Rekonstruktion beinhaltet das Aufbauen einer baumartigen Datenstruktur, indem der Benutzer Knoten, die ihre morphologischen Eigenschaften als Attribute speichern, und ihre hierarchischen Verbindungen definiert. Als Grundlage für diese Nachbildung dient der visualisierte Messdatensatz.
Die Visualisierung des Messvolumendatensatzes wurde in dieser Arbeit mit einem texturbasierten Verfahren und einem Iso-Oberflächen-Algorithmus umgesetzt. Die Darstellung des Messvolumendatensatzes kann jedoch Rauschanteile oder unterbrochene Wurzelsegmente enthalten. Diese Unstetigkeiten kann der Benutzer im Gegensatz zu einem automatischen Algorithmus leicht erkennen. Deswegen wurde hier eine manuelle Rekonstruktionsweise des Wurzelsystems gewählt.
Eine Nachbildung der Wurzeln per Hand erfordert jedoch einen hohen Grad an Interaktionsmöglichkeiten mit dem (Wurzel-)System. Sie werden in dieser Arbeit mit Hilfe von Virtual Reality Techniken umgesetzt. Die Immersion des Benutzers in der virtuellen Welt wird hier durch Stereoskopie und optisches Tracking erzeugt. Softwaretechnisch wird dieses Verhalten durch das VR-Toolkit *ViSTA* unterstützt. Zur Nutzung der VR-Komponenten wird innerhalb der im Rahmen dieser Arbeit entstandenen Software auf die Schnittstellen

von *ViSTA* zugegriffen.
Die implementierte Software stellt dem Benutzer eine Reihe von Funktionalitäten zur Verfügung, mit denen er das Wurzelsystem interaktiv nachbilden kann. Mit dem *Gyrostick* können die Knoten der Datenstruktur erzeugt und manipuliert werden. Manipulationen können auf den Durchmesser des Wurzelsegments bzw. den Radius des Knotens, auf die Position des Knotens und auf den hierarchischen Verknüpfungen zwischen den Knoten ausgeführt werden. Ein teilweises oder komplett rekonstruiertes Wurzelsystem kann gesichert und zu einer Weiterverarbeitung wieder geladen werden. Hierfür wird die standardisierte Auszeichnungssprache *XML* verwendet, da sie eine hierarchische Struktur direkt abbilden kann. Ebenfalls ist das Wurzelsystem in einem speziellen Format speicherbar, das für die Simulationen zur Vorhersage der Bodenwasseraufnahme mit dem Programm *R-SWMS* benötigt wird. Diese Datei erlaubt eine Abbildung des hierarchischen Wurzelsystems nur über sequentiell angeordnete Spezifizierungen der einzelnen Wurzelsegmente. Sie beinhaltet die zur Berechnung der Wasseraufnahme benötigten morphologischen Informationen der Wurzelsegmente.
Die korrekte Erzeugung der Datei wurde anhand eines Anwendungsbeispiels gezeigt. Dieses Beispiel behandelt das Wurzelsystem der Nutzpflanze Lupine. Beginnend mit dem MRT-Datensatz dokumentiert es alle durchgeführten Schritte der Rekonstruktion bis hin zu einer Wasseraufnahme-Simulation und der Bewertung ihrer Ergebnisse als plausibel.

Nachdem nun die praktische Anwendbarkeit der Resultate der entwickelten Software dargelegt wurde, sollte in naher Zukunft die auf realen Wurzelmessungen basierenden Simulationsergebnisse mit den experimentell gemessenen Bodenwasseraufnahmen verglichen werden. Dadurch können die physikalischen Annahmen, die dem *R-SWMS*-Simulationsmodell zugrunde liegen, bewertet und ggf. weiterentwickelt werden.
Eine künftige Verbesserung der entwickelten Software könnte Einschränkungen, die während der Implementierung und Anwendung aufgetreten sind (Kapitel 4.6), beheben. So sollten beispielsweise mögliche Durchdringungen von Wurzelsegmenten softwaretechnisch verhindert und die kleinen Ungenauigkeiten bei der Berechnung der Oberfläche eines Wurzelsegments eliminiert werden. Um die Bedienung der Software benutzerfreundlicher zu gestalten, sollte eine „Undo-Funktion“ implementiert werden. Zusätzlich könnte die Software um eine grafische Oberfläche ergänzt werden, die zum Beispiel eine einfache Auswahl der Manipulationsaktionen erlaubt.
Bei der Durchführung der manuellen Rekonstruktionen fällt außerdem auf, dass sie oft zeitaufwändig sind und je nach Komplexität des Wurzelsystems bis zu 2 Stunden dauern. Eine mögliche Erweiterung der Software könnte sich deswegen mit der Automatisierung der Wurzelrekonstruktion befassen. Wie bereits festgestellt wurde, existieren bei der automatischen Rekonstruktion Schwierigkeiten bei der Unterscheidung von Bodenrauschanteilen und Wurzelstruktur, sowie bei unterbrochenen Wurzelsegmenten. Deswegen wäre die Entwicklung eines halb-automatischen Systems sinnvoll, bei dem der Benutzer gelegentlich manuell eingreift und so Uneindeutigkeiten auflöst.
Bei einer möglichen Umsetzung dieser Erweiterung könnte der Benutzer zunächst alle Blattknoten und den Wurzelknoten per Hand spezifizieren, sodass das Bodenrauschen

nicht betrachtet wird. Mit einer Art dreidimensionalem Konturverfolgungsalgorithmus müsste dann je eine Linie von den definierten Blattknoten zum Wurzelknoten automatisch aufgebaut werden. Dazu würden die skalaren Werte der benachbarten Voxel untersucht werden und das Voxel mit dem ähnlichsten Wert ausgewählt werden. Durch eine sequentielle Fortsetzung entstände dadurch eine räumliche Kontur. Einschränkungen, wie ein Mindestbetrag für die Differenz zwischen zwei Skalaren oder ein Schwellenwert für die maximale Richtungsänderung der Linie, verursachen dabei das Stoppen an Segmentlücken bzw. verhindern unnatürlich scharfe Kurven der Kontur. Vom Algorithmus registrierte Wurzelunterbrechungen müsste der Benutzer per Hand schließen, indem er den nachfolgenden Punkt hinter der Lücke definiert. Bei zwei Blattknoten, die von der gleichen Verzweigung ausgehen, könnte es jedoch passieren, dass keine gemeinsame Kontur hinter der Verzweigung gefunden wird. Auch hier müsste der Benutzer manuell Konturlinien manipulieren. In einem letzten Schritt müssten dann die ermittelten Wurzelkonturen in ihrer Dicke den Durchmessern der Wurzelsegmente angepasst werden.

Literaturverzeichnis

Bücher und Paper

[1] Assenmacher, I., Kuhlen, T.; *The ViSTA Virtual Reality Toolkit*; SEARIS Workshop auf der IEEE VR 2008, Reno; 2008.

[2] Bender, M., Brill, M.; *Computergrafik – Ein anwendungsorientiertes Lehrbuch*; 2. Auflage; München, Wien; Carl Hanser Verlag; 2006.

[3] Birn, J.; *Lighting und Rendering*; München; Markt+Technik Verlag; 2001.

[4] Engel, K., Ertl, T.; *Interactive High-Quality Volume Rendering with Flexible Xonsumer Graphics Hardware*; Eurographics 2002; Saarbrücken; 2002.

[5] Grigore, C.B., Philippe, C.; *Virtual Reality Technology – Second Edition*; Hoboken, New Jersey, USA; Wiley-Interscience; 2003.

[6] Javaux, M., Schröder, T., Vanderborght, J., Vereecken, H.; *Use of a Three-Dimensional Detailed Modeling Approach for Predicting Root Water Uptake*; Vadose Zone Journal Vol. 7, No. 3, Seiten 1079-1088; 2008.

[7] Javaux, M., Schroeder, T., Vanderborght, J.; *R-SWMS: three-dimensional, simultaneous modelling of root growth, transient soil water flow, and solute transport and uptake*; Release 3.2.2; Forschungszentrum Jülich; 2009.

[8] Lösch, R.; *Wasserhaushalt der Pflanzen*; Wiebelsheim; Quelle & Meyer Verlag; 2001.

[9] Nischwitz, A., Fischer, M., Haberäcker, P.;*Computergrafik und Bildverarbeitung*; 2. Auflage; Wiesbaden; Vieweg Verlag; 2007.

[10] Ostnes, R., Abbott, V., Lavender, S.; *Visualition Techniques: An Overview – Part 2*; The Hydrographic Journal No. 114, October 2004, Seiten.3-9; 2004.

[11] Pohlmeier, A., Oros-Peusquens, M., Javaux, M., Menzel, M.I., Vanderborght, J., Kaffanke, J., Romanzetti, S., Lindenmair, J., Vereecken, H., Shah, N.J.; *Changes in Soil Water Content Resulting from Ricinus Root Uptake Monitored by Magnetic Resonance Imaging*; Vadose Zone Journal Vol. 7, No. 3, Seiten 1010-1017; 2008.

[12] Pohlmeier, A., Vergeldt, F., Gerkema, E., van As, H., van Dusschoten, D., Vereecken, H.; *MRI in Soils: Determination of water content changes due to root water uptake by means of a Multi-Slice-Multi-Echo sequence (MSME)*

[13] Pohlmeier, A., van Dusschoten, D., Blümler, P.; *Magnetic Resonance Imaging Methods in Soil Science*; Geophysical Research Abstracts, Vol. 11, EGU2009-5723, 2009; EGU General Assembly 2009; Wien; 2009.

[14] van Reimersdahl, T., Kuhlen, T., Gerndt, A., Henrichs, J., Bischof, C.; *ViSTA: a multimodal, platform-independent VR-Toolkit based on WTK, VTK, and MPI*; 4. International Immersive Projection Technology Workshop (IPT2000); Ames, Iowa; 2000.

[15] Sherman, W.R., Craig, A.B.; *Understanding Virtual Reality: Interface, Application, and Design*; San Francisco, USA; Morgan Kaufmann Publishers; 2003.

[16] Wild, A.; *Umweltorientierte Bodenkunde: Eine Einführung*; Heidelberg, Berlin, Oxford; Akademischer Verlag; 1995.

Internetquellen

[17] Homepage von 3DConnexion; *SpaceNavigator – Die universelle 3D-Maus*; http://www.3dconnexion.de/3dmouse/spacenavigator.php; Abruf 20.06.2009.

[18] Homepage von Advanced Realtime Tracking (A.R.T.); http://www.ar-tracking.de; Abruf 09.06.2009.

[19] Forschungszentrum Jülich; JSC News No. 167, September 2008; *PI-casso Workplace for JARA*; http://www.fz-juelich.de/jsc/docs/newsletter/2008/jscnews-167; Abruf 07.06.2009.

[20] Forschungszentrum Jülich, Vorlesung Wissenschaftliche Visualisierung; Zilken, H., Schumacher, H., Boltes, M.; WS 2008/2009; http://www.fz-juelich.de/jsc/cv/vislab/vorlesung/; Abruf 09.06.2009.

[21] VIEW of the future; Fraunhofer IAO Stuttgart; *PI-casso: Concept of an integrated VR and Desktop Workplace*; http://www.view.iao.fraunhofer.de/products.html; Abruf 09.06.2009.

[22] Homepage von Gyration; http://www.gyration.com/?l=ge; Abruf 05.07.2009.

[23] Fachdidaktische Seiten der botanischen Institute der Heinrich Heine Universität Düsseldorf; Gruchow, S.; *Wasser, das Lebenselixier der Pflanzen*; http://www.uni-duesseldorf.de/MathNat/Biologie/Didaktik/Wasserhaushalt/dateien/1_start/dateien/start.html; Abruf 09.07.2009.

[24] HTW Berlin, Vorlesung Entwicklung von Mediensoftware I; Jung, T.; *Stereo*; 10.05.2006; http://www.f4.fhtw-berlin.de/ tj/meso/stereo.pdf; Abruf 08.06.2009.

[25] Hydro Skript; Schöniger, M., Dietrich, J.; *Bodenwasser*; 25.08.2008; http://www.hydroskript.de/html/_index.html?page=/html/_info.html; Abruf 05.08.2009.

[26] Homepage des ICG-4 des Forschungszentrums Jülich; http://www.fz-juelich.de/icg/icg-4; Abruf 09.07.2009.

[27] Homepage des ICG-4 des Forschungszentrums Jülich; Javaux, M.; *Dreidimensionale, gleichzeitige Modellierung von Wurzelwachstum, Wurzelwasseraufnahme und vorübergehenden Wasserfluss im Boden und innerhalb der Wurzel*; http://www.fz-juelich.de/icg/icg-4/index.php?index=668; Abruf 10.07.2009.

[28] Hompage von imsys (immersive systems); *flip - VR günstig, schnell und flexibel*; http://www.imsys-vr.com/128.0.html; Abruf 09.07.2009.

[29] Innovations Report – Forum für Wissenschaft, Industrie und Wirtschaft; *Virtual Reality am Arbeitsplatz*; 22.06.2004; http://www.innovations-report.de/html/berichte/informationstechnologie/bericht-30412.html; Abruf 09.06.2009.

[30] Julius Kühn-Institut – Bundesforschungsinstitut für Kulturpflanzen; Ruge-Wehling, B., Wohlers, W.; *Lupinus angustifolius – Blaue oder Schmalblättrige Lupine*; Februar 2009; http://www.jki.bund.de/nn_1192396/DE/Home/kulturpflanzen/lupine/lupine__node.html__nnn=true; Abruf 31.7.2009.

[31] Homepage von Morton Heilig; *Inventor in the field of Virtual Reality*; http://www.mortonheilig.com/InventorVR.html; Abruf 09.06.2009.

[32] Ohio State University, Introduction to Computer Graphics; Gurari, E.; *Coordinates, Windows, and Viewports*; Herbst 1993; http://www.cse.ohio-state.edu/g̃urari/course/cis681/cis681Ch7.html; Abruf 01.08.2009.

[33] Homepage von OpenSG; Tutorial; http://opensg.vrsource.org/trac/wiki/Tutorial/OpenSG1/Introduction; Abruf 05.07.2009.

[34] RWTH Aachen, VR Group Wiki; *ViSTA*; http://www1.rz.rwth-aachen.de/vr/wiki/dokuwiki/doku.php/start; Abruf 05.07.2009.

[35] RWTH Aachen, Seminar Medizinische Datenverarbeitung; Erfurt; R.; SS 2007; *Vorintegriertes Volume Rendering: Slicing vs. Raycasting*; http://phobos.imib.rwth-aachen.de/lehmann/seminare/bv_2007-09.pdf; Abruf 05.07.2009.

[36] Grimm, J. (Zentrum für Molekulare Bildgebung, Radiologische Abteilung, Massachusetts General Hospital und Harvard Medical School, Boston, Massachusetts, USA), Schmitt, F. (Siemens Medical Solutions, Abteilung MREF, Erlangen); *MR-Tomographie (MRT) bei 7 Tesla*; Medical Solutions März 2006;

http://www.medical.siemens.com/siemens/de_DE/rg_marcom_FBAs/files/brochures/magazine1_2006/Science_7_Tesla_Maerz2006_d.pdf; Abruf 25.07.2009.

[37] Thomason, L., Berquin, Y., Ellerton, A.; *TinyXml Documentation*; 18.08.2006; http://www.grinninglizard.com/tinyxml/; Abruf 05.07.2009.

[38] Institut für Geoökologie (IGÖ), TU Braunschweig; 17.10.2007; http://www.igoe.tu-bs.de/soil/pics_download/downloads/pubs/vortraege/2007-10-17.UMS%20Weihenstefan.ppt.pdf; Abruf 05.08.2009.

[39] TU München, Hauptseminar Augmented Reality; Dimitrova, I.; *Kamera Kalibrierung*; http://campar.in.tum.de/twiki/pub/Chair/TeachingWs04Tracking/05CameraCalibration.pdf; Abruf 18.06.2009.

[40] Uni Magdeburg, Vorlesung Grundlagen Computer Vision; Tönnis; K.; SS 2009; *Photogrammetrie*; http://wwwisg.cs.uni-magdeburg.de/bv/skript/gcv/GCV03_2009.pdf; Abruf 25.07.2009.

[41] Homepage von VTK; *VTK File Formats*; http://www.vtk.org/VTK/img/file-formats.pdf; Abruf 05.07.2009.

Anhang A

Softwarebedienung

A.1 Aufrufargumente

Argument	**Bedeutung**
-dataset <dateiname>	Zur Visualisierung des Messdatensatzes im VTK-Legacy-Format, Pflichtangabe
-transferFunc <dateiname>	Stellt den angegebenen Datensatz entsprechend der angegebenen Transferfunktion (xml-Datei) dar, Default: Regenbogen-Transferfunktion
-iso <isowert>	Float-Isowert für die Oberflächendarstellung des Datensatzes mit dem Marching Cubes Algorithmus, Default-Wert: 10,0
-load <dateiname>	<dateiname> spezifiziert eine XML-Datei, dadurch wird eine zuvor gespeicherte rekonstruierte Wurzelstruktur geladen
-cm <colourmode>	Ganzzahl, die aus einer Farb-Map einen bestimmten Farbmodus auswählt und entsprechend die verschiedenen Zweige der Wurzelrekonstruktion einfärbt (0: schwarz bis weiß, 1: blau bis rot, 2: gelb bis rot, 3: cyan bis rot rückwärts, 4: grün bis blau, 1023: alle Farben des HSV-Raums, 1024: alle Farben von rot bis rot), Default-Wert: 1023
-cn <farbanzahl>	Ganzzahl, die die maximale Anzahl an verschiedenen Farben bzgl. des Farbmodus angibt, Default-Wert: 8
-dens <dichte>	Float-Wert, der die Dichte der Wurzeln in $\left[\frac{g}{cm^3}\right]$ angibt, Default-Wert: 1,0
-segLen <segmentlänge>	Float-Wert, der die maximale Länge eines Wurzelsegments in cm angibt, Default-Wert: 1,0
-zFac <faktor>	Float-Wert $\neq 0$, der einen Faktor für die z-Werte der rekonstruierten Wurzel beim Rausschreiben der Daten im R-SWMS-Format beschreibt, Default-Wert: 1,0

A.2 Manipulationsaktionen

Taste	Modus	Aktion
c	Create	Modus zum Erzeugen eines neuen Knotens.
a	Auswahl	Modus zum Selektieren des nächstliegenden Knotens.
t	Auswahl zweiter Knoten	Modus zum Selektieren eines zweiten Knotens zum Hinzufügen oder Löschen eines Segments.
Enter	Change Segment	Nach Auswahl von zwei Knoten wird ein Segment hinzugefügt oder gelöscht.
d	Deselect	Deselektiert alle Knoten, z.B. zur Erzeugung eines neuen Baums.
Entf	Entfernen	Löscht den ausgewählten Knoten. Kinderknoten bleiben erhalten und werden dem Vaterknoten zugordnet.
Shift + Entf	Entfernen mit Teilbaum	Löscht den ausgewählten Knoten und alle seine Kinderknoten.
r	Convert to Root	Wandelt den ausgewählten Knoten in den Wurzelknoten des Baums um. Alle Knoten bleiben erhalten.
b	Branch order	Korrigiert die Verzweigungsordnung aller Knoten, was zu einer Farbänderung der Wurzelsegmente führen kann. Sinnvoll nach dem Löschen von Knoten.
i	View	Wechselt zwischen den (Volume Rendering-) Ansichten: Textur - Oberfläche - Ohne.
z	Scale/Zoom	Wechselt zwischen den Navigationsmodi Skalierung und Zoom. Modus wirkt sich nur auf Bewegungen mit dem Spacenavigators in z-Richtung aus.
Strg + s	Speichern unter (XML)	Speichert die rekonstruierte Wurzelstruktur in der angegebenen XML-Datei, die später wieder geladen werden kann.
s	Speichern (XML)	Speichert die rekonstruierte Wurzelstruktur in einer zuvor spezifizierten XML-Datei (siehe Speichern unter). Eingabe des Dateinamens (außer beim aller ersten Speichern) nicht möglich.
Strg + w	Write unter (R-SWMS)	Speichert die rekonstruierte Wurzelstruktur im R-SWMS-Format in einer angegebenen Datei.
w	Write (R-SWMS)	Speichert die rekonstruierte Wurzelstruktur im R-SWMS-Format in einer zuvor spezifizierten Datei (siehe Write unter). Eingabe des Dateinamens (außer beim aller ersten Speichern) nicht möglich.
q	Quit	Schließt das Programm und sollte immer verwendet werden, um ein sauberes Beenden zu gewährleisten.
linke Maustaste		Drücken zur Erzeugung und Auswahl von Knoten. Gedrückt halten zur Positionsveränderung des Knotens.
rechte Maustaste		Gedrückt halten zum Vergrößern/Verkleinern des Radius des ausgewählten Knotens. Es wirken sich nur Bewegungen in x-Richtung aus.

Anhang B

Konfiguration der VR-Hardware

B.1 XML-Interaktionsdateien

Konfiguration der Tastatur

```
<module>
  <graph>
    <!-- data source: a keyboard -->
    <node name="keyboard" type="Sensor">
      <param name="sensor" value="0"/>
      <param name="driver" value="KEYBOARD"/>
    </node>

    <node name="triggerlist" type="Aggregate[int]">
      <param name="in" value="KEY"/>
    </node>
    <node name="modderlist" type="Aggregate[int]">
      <param name="in" value="MODIFIER"/>
    </node>

    <node name="apply" type="systemcontrol"/>
  </graph>
  <edges>
    <!-- connect hid -> overlay -->
    <edge fromnode="keyboard" tonode="triggerlist" fromport="history"
        toport="history"/>
    <edge fromnode="keyboard" tonode="modderlist" fromport="history"
        toport="history"/>
    <edge fromnode="triggerlist" tonode="apply" fromport="values"
        toport="triggerlist"/>
    <edge fromnode="modderlist" tonode="apply" fromport="values" toport="modlist"/>
  </edges>
</module>
```

Listing B.1: keycontrol.xml

Konfiguration der Maus

```
<module>
  <graph>
    <!-- data source: a mouse -->
    <node name="mouse" type="Sensor">
      <param name="sensor" value="0"/>
      <param name="driver" value="MOUSE"/>
    </node>
    <!-- project the necessary information from the mouse history -->
    <node name="project_mouse" type="HistoryProject">
      <param name="project">POSITION, LEFT_BUTTON, RIGHT_BUTTON</param>
    </node>
    <!-- sampling mode: lazy sampling -->
    <node name="project_mode" type="Constant[int]">
      <param name="mode" value="0"/>
    </node>

    <!-- get normalize source -->
    <node name="normwindow" type="windownode">
      <param name="value" value="MONO_WINDOW"/>
    </node>
    <!-- the window spits out ints as attribs, we need floats for the normalize
        node -->
    <node name="conv_w" type="Convert[int,float]"/>
    <node name="conv_h" type="Convert[int,float]"/>

    <node name="normalize" type="3DNormalize"/>

    <!-- we will re-use the constant node more than once, so we will simply call it
         value-2 as it represents a value of 2 -->
    <node name="value-2" type="Constant[float]">
      <param name="value" value="2.0"/>
    </node>
    <!-- same here, but for a 0-value -->
    <node name="value-0" type="Constant[float]">
      <param name="value" value="0.0"/>
    </node>

    <!-- we want to normalize between [-1:1] for x and y -->
    <node name="min_value" type="Constant[float]">
      <param name="value" value="-1.0"/>
    </node>

    <!-- setup the operations of the graph -->
    <node name="position_to_norm" type="MultVector"/>

    <!-- the final coords to give into the trackball need a flipped
         y axis for the mouse -->
    <node name="flip_axes" type="MultVector"/>

    <node name="y_sign_invert" type="Constant[CVistaTransformMatrix]">
      <param name="value"> 1,  0,  0,  0,
                           0, -1,  0,  0,
                           0,  0,  1,  0,
                           0,  0,  0,  1 </param>
    </node>

    <!-- now get the transform to modify, using a trackball. We will try to claim
        an object that is called "camera". Note that we have to set this within the
         application by using the VdfnObjectRegistry from the VistaSystem -->
    <node name="get_transform" type="GetTransform">
```

```
    <param name="object" value="model"/>
  </node>

  <!-- the result is a transform that has to be applied to an object -->
  <node name="apply_transform" type="ApplyTransform">
    <param name="object" value="model"/>
  </node>

  <!-- state control, attach left/right button to a change detect node -->
  <node name="dt_cg_rotate" type="ChangeDetect[bool]"/>
  <node name="dt_cg_zoom" type="ChangeDetect[bool]"/>

  <node name="trackball_center_transform" type="GetTransform">
    <param name="object" value="model"/>
  </node>
  <node name="vec_zero" type="Constant[CVistaVector3D]">
    <param name="value"> 0,  0,  0,  1 </param>
  </node>
  <node name="trackball_center" type="MultVector"/>

  <!-- almost finally, the trackball transformation itself -->
  <node name="trackball" type="trackball">
    <param name="invert" value="false" />
    <param name="displaysystem" value="MONO" />
  </node>
</graph>
<edges>
  <!-- connect mouse -> project -->
  <edge fromnode="mouse" tonode="project_mouse" fromport="history"
      toport="history"/>
  <edge fromnode="project_mode" tonode="project_mouse" fromport="value"
      toport="sampling_mode"/>

  <!-- setup the normalize node -->
  <!-- add typechangers -->
  <edge fromnode="normwindow" tonode="conv_w" fromport="win_w" toport="in"/>
  <edge fromnode="normwindow" tonode="conv_h" fromport="win_h" toport="in"/>
  <edge fromnode="conv_w" tonode="normalize" fromport="out" toport="source_w"/>
  <edge fromnode="conv_h" tonode="normalize" fromport="out" toport="source_h"/>

  <!-- connect the "0" to the depth value here -->
  <edge fromnode="value-0" tonode="normalize" fromport="value"
      toport="source_d"/>
  <!-- connect the same "0" to the min_z for the normalizer -->
  <edge fromnode="value-0" tonode="normalize" fromport="value" toport="min_z"/>
  <!-- connect the min value -1 to both, x and y -->
  <edge fromnode="min_value" tonode="normalize" fromport="value" toport="min_x"/>
  <edge fromnode="min_value" tonode="normalize" fromport="value" toport="min_y"/>

  <!-- connect the target width from value-2 -->
  <edge fromnode="value-2" tonode="normalize" fromport="value"
      toport="target_w"/>
  <edge fromnode="value-2" tonode="normalize" fromport="value"
      toport="target_h"/>
  <edge fromnode="value-0" tonode="normalize" fromport="value"
      toport="target_d"/>

  <!-- setup the operations / transforms of this graph this operation will scale
      the position vector to the target frame -->
  <edge fromnode="project_mouse" tonode="position_to_norm" fromport="POSITION"
      toport="vec_in"/>
```

```
    <edge fromnode="normalize"     tonode="position_to_norm" fromport="transform"
        toport="mat_in"/>

    <!-- set constant sign-mirror matrix -->
    <edge fromnode="y_sign_invert" tonode="flip_axes" fromport="value"
        toport="mat_in"/>

    <!-- apply the flip axes -->
    <edge fromnode="position_to_norm" tonode="flip_axes" fromport="vec_out"
        toport="vec_in"/>

    <!-- up to now, we have normalized mouse coordinates (as a vec 3D) with a
        correct y-sign, so push those into the trackball -->
    <edge fromnode="flip_axes" tonode="trackball" fromport="vec_out"
        toport="coords"/>

    <!-- get the base transform to modify -->
    <edge fromnode="get_transform" tonode="trackball" fromport="transform"
        toport="transform"/>

    <!-- set the rotation center -->
    <edge fromnode="vec_zero" tonode="trackball_center" fromport="value"
        toport="vec_in"/>
    <edge fromnode="trackball_center_transform" tonode="trackball_center"
        fromport="transform" toport="mat_in"/>
    <edge fromnode="trackball_center" tonode="trackball" fromport="vec_out"
        toport="center"/>

    <!-- link state control for left and right button accordingly -->

    <!-- first, connect the change detect node to the values from the project node
        in order to get the latest values -->
    <edge fromnode="project_mouse" tonode="dt_cg_rotate" fromport="LEFT_BUTTON"
        toport="val_in"/>
    <edge fromnode="project_mouse" tonode="dt_cg_zoom" fromport="RIGHT_BUTTON"
        toport="val_in"/>

    <!-- now link the change detect to the trackball rotate / zoom input port -->
    <edge fromnode="dt_cg_rotate" tonode="trackball" fromport="val_out"
        toport="mode_rotate"/>
    <edge fromnode="dt_cg_zoom" tonode="trackball" fromport="val_out"
        toport="mode_zoom"/>

    <!-- apply output of trackball_trans to the apply node -->
    <edge fromnode="trackball" tonode="apply_transform" fromport="transform"
        toport="transform"/>
  </edges>
</module>
```

Listing B.2: trackball_transform.xml

Konfiguration des Spacenavigators

```
<module>
  <graph>
    <!-- data source: spacenavigator -->
    <node name="spacenavigator" type="Sensor">
      <param name="sensor" value="0" />
      <param name="driver" value="6DMOUSE" />
    </node>

    <!-- history project node -->
    <node name="project_spacenav" type="HistoryProject">
      <param name="project">POSITION, AXIS, BUTTON_1, BUTTON_2</param>
    </node>

    <!-- timer node -->
    <node name="timer" type="TimerNode">
      <param name="reset_on_activate" value="true" />
    </node>

    <!-- this will give the sampling mode to the history project node, a value of
        "0" means "LAZY" while a value of "1" means "HOT". There is no other
        sampling value currently -->
    <node name="sampling_mode" type="Constant[int]">
      <param name="mode" value="0"/>
    </node>

    <node name="nav_handler" type="actionnode">
      <param name="object" value="nav_handler"/>
    </node>
  </graph>
  <edges>
    <!-- connect spacenavi -|-> mode -|-> project -->
    <edge fromnode="spacenavigator" tonode="project_spacenav" fromport="history"
        toport="history" />
    <edge fromnode="sampling_mode" tonode="project_spacenav" fromport="value"
        toport="sampling_mode" />

    <edge fromnode="project_spacenav" tonode="nav_handler" fromport="POSITION"
        toport="position" />
    <edge fromnode="project_spacenav" tonode="nav_handler" fromport="AXIS"
        toport="axis" />
    <edge fromnode="project_spacenav" tonode="nav_handler" fromport="BUTTON_1"
        toport="btn_1" />
    <edge fromnode="project_spacenav" tonode="nav_handler" fromport="BUTTON_2"
        toport="btn_2" />
    <edge fromnode="timer" tonode="nav_handler" fromport="dt" toport="delta_time"/>
  </edges>
</module>
```

Listing B.3: spacenavigator.xml

Konfiguration des Kopftrackings

```
<module>
  <graph>
    <node name="head_source" type="Sensor">
      <param name="sensor" value="0"/>
      <param name="type" value="BODY"/>
      <param name="driver" value="DTRACK"/>
    </node>
    <node name="head" type="HistoryProject">
      <param name="project">POSITION, ORIENTATION</param>
    </node>
    <!-- this will give the sampling mode to the history project -->
    <node name="project_mode" type="Constant[int]">
      <param name="mode" value="0"/>
    </node>

    <node name="scale_pos" type="MultScalar" />
    <node name="translate_pos_tracker" type="add[CVistaVector3D]" />
    <node name="translate_pos_glasses" type="add[CVistaVector3D]" />

    <node name="tracker_offset" type="Constant[CVistaVector3D]">
      <param name="value" value="0, -1395, 1093" />
    </node>

    <node name="pos_scale" type="Constant[float]">
      <param name="value" value=".001" />
    </node>

    <node name="quat_glasses_rot" type="Constant[CVistaQuaternion]">
      <param name="value" value="-0.000000, -0.000000, -0.216440, 0.976296" />
    </node>

    <node name="quat_y_rot" type="Constant[CVistaQuaternion]">
      <param name="value" value="0.707, 0, 0, 0.707" />
    </node>

    <node name="rotate_y" type="mult[CVistaQuaternion]" />

    <!-- this is the offset, in mm, from the zero-marker to the glasses center -->
    <node name="to_glasses_center" type="Constant[CVistaVector3D]">
      <param name="value" value="78, -29, 0" />
    </node>

    <node name="rotate_glasses" type="mult[CVistaQuaternion]" />

    <node name="rotate_offset" type="rotate_vector" />

    <!-- SINK NODE -->
    <node name="ucp" type="viewersink">
      <param name="displaysystem" value="STEREO"/>
    </node>
  </graph>
  <edges>
    <!-- history projection -->
    <edge fromnode="project_mode" tonode="head" fromport="value"
        toport="sampling_mode"/>
    <edge fromnode="head_source" tonode="head" fromport="history"
        toport="history"/>

    <!-- position transformation -->
```

```
    <edge fromnode="head"              tonode="rotate_offset"
        fromport="ORIENTATION" toport="rotation" />
    <edge fromnode="to_glasses_center" tonode="rotate_offset" fromport="value"
             toport="vector" />

    <edge fromnode="head"           tonode="translate_pos_tracker"
        fromport="POSITION" toport="first" />
    <edge fromnode="tracker_offset" tonode="translate_pos_tracker"
        fromport="value" toport="second" />

    <edge fromnode="translate_pos_tracker" tonode="translate_pos_glasses"
        fromport="out"  toport="first" />
    <edge fromnode="rotate_offset"         tonode="translate_pos_glasses"
        fromport="vector" toport="second" />

    <edge fromnode="translate_pos_glasses" tonode="scale_pos" fromport="out"
        toport="vec_in" />
    <edge fromnode="pos_scale"             tonode="scale_pos" fromport="value"
        toport="scalar_in" />

    <!-- orientation transformation -->
    <edge fromnode="head"              tonode="rotate_glasses"
        fromport="ORIENTATION" toport="first"/>
    <edge fromnode="quat_glasses_rot" tonode="rotate_glasses" fromport="value"
             toport="second"/>

    <edge fromnode="rotate_glasses" tonode="rotate_y"  fromport="out"
        toport="first"/>
    <edge fromnode="quat_y_rot"     tonode="rotate_y"  fromport="value"
        toport="second"/>

    <!-- connect to the viewer sink -->
    <edge fromnode="scale_pos"     tonode="ucp"  fromport="vec_out"
        toport="position"/>
    <edge fromnode="rotate_y"      tonode="ucp"  fromport="out"
        toport="orientation"/>
  </edges>
</module>
```

Listing B.4: ucp_picasso_stereo.xml

Konfiguration des Gyrosticks

```
<module>
  <graph>
    <node name="stick_source" type="Sensor">
      <param name="sensor" value="1"/>
      <param name="type" value="BODY"/>
      <param name="driver" value="DTRACK"/>
    </node>
    <node name="stick" type="HistoryProject">
      <param name="project">POSITION, ORIENTATION, DSCALAR</param>
    </node>
    <node name="pointer3D_handler" type="actionnode">
      <param name="object" value="pointer3D_handler"/>
    </node>

    <!-- this will give the sampling mode to the history project -->
    <node name="project_mode" type="Constant[int]">
      <param name="mode" value="0"/>
    </node>

    <node name="scale_pos" type="MultScalar" />
    <node name="translate_pos_tracker" type="add[CVistaVector3D]" />
    <node name="translate_pos_stick" type="add[CVistaVector3D]" />

    <node name="tracker_offset" type="Constant[CVistaVector3D]">
      <param name="value" value="0, -1395, 1093" />
    </node>

    <node name="pos_scale" type="Constant[float]">
      <param name="value" value=".001" />
    </node>

    <node name="quat_stick_rot" type="Constant[CVistaQuaternion]">
      <param name="value" value="-0.000000, -0.000000, 0, 1" />
    </node>

    <node name="quat_y_rot" type="Constant[CVistaQuaternion]">
      <param name="value" value="0, 0, 1, 0" />
    </node>

    <node name="rotate_y" type="mult[CVistaQuaternion]" />

    <!-- this is the offset, in mm, from the zero-marker to the stick center -->
    <node name="to_stick_center" type="Constant[CVistaVector3D]">
    <!-- coordinate system of hardware device is rotated by z-axis. Therefore x-
        and y-axes have to direct to the opposite direction: minus. However this is
         not true anymore if we dont use the rotate_offset node then you have to
        use "normal" values -->
      <param name="value" value="-50, 100, -250" />
    </node>

    <node name="rotate_stick" type="mult[CVistaQuaternion]" />

    <node name="rotate_offset" type="rotate_vector" />

    <!-- nodes for usage of mouse buttons -->
    <!-- data source: a mouse -->
    <node name="mouse" type="Sensor">
      <param name="sensor" value="0"/>
      <param name="driver" value="GYROMOUSE"/>
    </node>
```

```
    <node name="project_mouse" type="HistoryProject">
      <param name="project">LEFT_BUTTON, RIGHT_BUTTON</param>
    </node>
    <node name="project_mode" type="Constant[int]">
      <param name="mode" value="0"/>
    </node>
  </graph>
  <edges>
    <!-- history projection -->
    <edge fromnode="project_mode" tonode="stick" fromport="value"
        toport="sampling_mode"/>
    <edge fromnode="stick_source" tonode="stick" fromport="history"
        toport="history"/>

    <!-- position transformation -->
    <edge fromnode="to_stick_center" tonode="translate_pos_glasses"
        fromport="value"       toport="second" />

    <edge fromnode="stick"           tonode="translate_pos_tracker"
        fromport="POSITION" toport="first" />
    <edge fromnode="tracker_offset" tonode="translate_pos_tracker"
        fromport="value"    toport="second" />

    <edge fromnode="translate_pos_tracker" tonode="translate_pos_glasses"
        fromport="out"  toport="first" />

    <edge fromnode="translate_pos_glasses" tonode="scale_pos" fromport="out"
        toport="vec_in" />
    <edge fromnode="pos_scale"             tonode="scale_pos" fromport="value"
        toport="scalar_in" />

    <!-- orientation transformation -->
    <edge fromnode="stick"             tonode="rotate_glasses"
        fromport="ORIENTATION" toport="first"/>
    <edge fromnode="quat_stick_rot" tonode="rotate_glasses" fromport="value"
        toport="second"/>

    <edge fromnode="rotate_glasses" tonode="rotate_y"  fromport="out"
        toport="first"/>
    <edge fromnode="quat_y_rot"     tonode="rotate_y"  fromport="value"
        toport="second"/>

    <!-- connect to the pointer 3d handler -->
    <edge fromnode="scale_pos"     tonode="pointer3D_handler"  fromport="vec_out"
          toport="position"/>
    <edge fromnode="rotate_y"      tonode="pointer3D_handler"  fromport="out"
        toport="orientation"/>

    <!-- edges for mouse -->
    <edge fromnode="mouse" tonode="project_mouse" fromport="history"
        toport="history"/>
    <edge fromnode="project_mode" tonode="project_mouse" fromport="value"
        toport="sampling_mode"/>
    <edge fromnode="project_mouse" tonode="pointer3D_handler"
        fromport="LEFT_BUTTON" toport="btn1"/>
    <edge fromnode="project_mouse" tonode="pointer3D_handler"
        fromport="RIGHT_BUTTON" toport="btn2"/>
  </edges>
</module>
```

Listing B.5: gyrostick.xml

B.2 System-Konfiguration

```
##########################################################################
#        V I S T A     I N I T I A L I Z A T I O N     F I L E        #
##########################################################################
[SYSTEM]
MSGPORT = TRUE
MSGPORTIP = localhost
MSGPORTPORT = 66666
MOUSEVIEWPORT = MONO_VIEWPORT

DRIVERPLUGINS= /home/wienke/Vista/Vista/lib/LINUX/

NEWINTERACTION = true

DUMPGRAPHS = TRUE

# ==============
# CLUSTERSECTION
# ==============
CLUSTERSECTION = VISTACLUSTER

# ===============
# GRAPHICSSECTION
# ===============
GRAPHICSSECTION = GRAPHICS_NEW

# =============
# DISPLAYSYSTEM
# =============
DISPLAYSYSTEMS = STEREO

#################################################################
LOADPLUGINS = 3DCSpaceNavigatorPlugin
 xx
DEVICEDRIVERS = GYROMOUSE, KEYBOARD, DTRACK, SPACENAVIGATOR

INTERACTIONCONTEXTS = KEYCONTROL, UCP_CONTEXT,NAV_CAM,NAVIGATION_GYRO_CTX

SYSTEMCONTROLDEVICE = KEYCONTROL_DEVICE

[KEYCONTROL]
ROLE = KEYCONTROL
GRAPH = configfiles/interaction/keycontrol.xml

[NAV_CAM]
ROLE = CAMERA
GRAPH = configfiles/interaction/spacenavigator.xml
RELOADTRIGGER = B

[NAVIGATION_GYRO_CTX]
ROLE = POINTDEVICE
GRAPH = configfiles/interaction/gyrostick.xml

# ucp = user centered projection
[UCP_CONTEXT]
ROLE = UCP
GRAPH = configfiles/interaction/ucp_picasso_stereo.xml

[KEYBOARD]
TYPE = KEYBOARD
```

```
DEFAULTWINDOW = TRUE
HISTORY = 1
SENSORS = KEYB_MAIN

[KEYB_MAIN]
RAWID = 0

[KEYCONTROL_DEVICE]
SENSORIDX = 0
TRANSFORM =
DRIVER = KEYBOARD

[SPACENAVIGATOR]
TYPE    = SPACENAVIGATOR
SENSORS = SN_MAIN
HISTORY = 10
NAME    = 6DMOUSE

[SN_MAIN]
RAWID = 0

[GYROMOUSE]
TYPE = MOUSE
HISTORY = 10
SENSORS = MOUSE_MAIN
#WINDOWS = MAIN_WINDOW
DEFAULTWINDOW = TRUE

[MOUSE]
TYPE = MOUSE
HISTORY = 5
SENSORS = MOUSE_MAIN
DEFAULTWINDOW = TRUE

[MOUSE_MAIN]
RAWID = 0

[TRACKBALL_TRANS]
ROLE = TRACKBALL
GRAPH = configfiles/interaction/trackball_transform.xml

##########################################################################

[GRAPHICS_NEW]
# a temporally used flag
# Choose GraphicsSystem: NEW, OLD
GraphicsManager = NEW

#Transform matrix for root node
ROOT_ORIENTATION11 = 1.0
ROOT_ORIENTATION12 = 0.0
ROOT_ORIENTATION13 = 0.0
ROOT_ORIENTATION14 = 0.0

ROOT_ORIENTATION21 = 0.0
ROOT_ORIENTATION22 = 1.0
ROOT_ORIENTATION23 = 0.0
ROOT_ORIENTATION24 = 0.0

ROOT_ORIENTATION31 = 0.0
ROOT_ORIENTATION32 = 0.0
ROOT_ORIENTATION33 = 1.0
```

```
ROOT_ORIENTATION34 = 0.0

Lights = LIGHT_A0, LIGHT_D0, LIGHT_D1, LIGHT_P0

[LIGHT_A0]
Type = AMBIENT
AmbientColor = 0.2, 0.2, 0.2
Intensity = 1.0

[LIGHT_D0]
Type = DIRECTIONAL
AmbientColor = 0.1, 0.1, 0.1
DiffuseColor = 0.6, 0.6, 0.6
SpecularColor = 0.5, 0.5, 0.5
Direction = 1.0, -1.0, 0.0

[LIGHT_D1]
Type = DIRECTIONAL
AmbientColor = 0.1, 0.1, 0.1
DiffuseColor = 0.6, 0.6, 0.6
SpecularColor = 0.5, 0.5, 0.5
Direction = 0.0, 0.0, -1.0

[LIGHT_D2]
Type = DIRECTIONAL
AmbientColor = 0.1, 0.1, 0.1
DiffuseColor = 0.6, 0.6, 0.6
SpecularColor = 0.5, 0.5, 0.5
Direction = -1.0, 1.0, 1.0

[LIGHT_P0]
Type = POINT xx
AmbientColor = 0.1, 0.1, 0.1
DiffuseColor = 0.9, 0.9, 0.9
SpecularColor = 0.5, 0.5, 0.5
Position = 0.0, 1.0, 1.0
Attenuation = 1.0, 0.0, 0.0

[LIGHT_S0]
Type = SPOT
AmbientColor = 0.1, 0.1, 0.1
DiffuseColor = 0.5, 0.5, 0.5
SpecularColor = 0.5, 0.5, 0.5
Direction = 0.0, 0.0, -1.0
Position = 0.0, 0.0, 0.0
Attenuation = 1.0, 0.0, 0.0

#########################################################################
# DISPLAY-SETTINGS
#########################################################################

### mono window ###################################
[MONO]
NAME      = MONO
VIEWPORTS     =   MONO_VIEWPORT
VIEWER_POSITION   = 0, 0, 2
VIEWER_ORIENTATION  = 0, 0, 0, 1
LEFT_EYE_OFFSET   = -0.01, 0, 0
RIGHT_EYE_OFFSET  = 0.01, 0, 0

REFERENCE_FRAME   = MONO_REF_FRAME
```

```
[MONO_REF_FRAME]
NAME       = Mono Reference Frame
TRANSLATION    = 0, 0, -0.25

[MONO_VIEWPORT]
NAME     = MONO_VIEWPORT
PROJECTION   = MONO_PROJECTION
WINDOW     = MONO_WINDOW
#POSITION = 0, 0
#SIZE    = 1280, 1024

[MONO_PROJECTION]
NAME       = MONO_PROJECTION
PROJ_PLANE_MIDPOINT = 0, 0, 0
PROJ_PLANE_NORMAL = 0, 0, 1
PROJ_PLANE_UP    = 0, 1, 0
PROJ_PLANE_EXTENTS   = -1, 1, -0.75, 0.75
CLIPPING_RANGE     = 0.1, 65000
STEREO_MODE    = MONO

[MONO_WINDOW]
NAME       = MONO_WINDOW
STEREO       = false
POSITION     = 40, 20
SIZE       = 640, 480
DRAW_BORDER    = true

### stereo window ################################
[STEREO]
NAME       = STEREO
VIEWPORTS      =    STEREO_VIEWPORT
VIEWER_POSITION    = 0, 1, 1
VIEWER_ORIENTATION   = 0, 0, 0, 1
LEFT_EYE_OFFSET    = -0.03, 0, 0
RIGHT_EYE_OFFSET   = 0.03, 0, 0

[STEREO_VIEWPORT]
NAME     = STEREO_VIEWPORT
PROJECTION   = STEREO_PROJECTION
WINDOW     = STEREO_WINDOW
#POSITION = 0, 0
#SIZE    = 1280, 1024

[STEREO_PROJECTION]
NAME       = STEREO_PROJECTION
PROJ_PLANE_MIDPOINT = 0, 0, 0
PROJ_PLANE_NORMAL = 0, 0.258819, 0.965925
PROJ_PLANE_UP    = 0, 0.965925, -0.258819
PROJ_PLANE_EXTENTS   = -0.6, 0.6, -0.45, 0.45
CLIPPING_RANGE     = 0.1, 65000
STEREO_MODE    = FULL_STEREO

[STEREO_WINDOW]
NAME       = STEREO_WINDOW
STEREO       = true
SIZE       = 1400, 1050
DRAW_BORDER    = false
FULL_SCREEN    = true

#####################################################################
###                            DTRACK
#####################################################################
```

```
[DTRACK]
TYPE = DTRACK
NAME = DTRACK
PROTOCOL = DTRACK2
CONNECTIONS = DTRACKCONTROL, DTRACKDATA
SENSORS = HEAD, GYRO
HISTORY = 10
ATTACHONLY = FALSE

[DTRACK2]
NAME = dtrack2

[HEAD]
TYPE = BODY
RAWID = 0

[GYRO]
TYPE = BODY
RAWID = 1

[DTRACKCONTROL]
TYPE = TCP
DRIVERROLE = CONTROLCONNECTION
#ADDRESS = 192.168.0.1
ADDRESS = trackingpc
PORT    = 50105
DIRECTION = OUTGOING

[DTRACKDATA]
TYPE = UDP
DRIVERROLE = DATACONNECTION
#ADDRESS = 192.168.0.5
ADDRESS = renderpc
PORT    = 5001
DIRECTION = INCOMING
```

Listing B.6: vista_picasso.ini

Anhang C

Rekonstruktion der Lupine

C.1 Datensatz im VTK-Legacy-Format

```
# vtk DataFile Version 3.0
VTK dataset
BINARY
DATASET STRUCTURED_POINTS
DIMENSIONS 256 256 110
SPACING 0.039062 0.031250 0.050000
ORIGIN 0 0 0
POINT_DATA 7208960
SCALARS scalars unsigned_char 1
LOOKUP_TABLE default
```

Listing C.1: Header der VTK-Legacy-Datei

C.2 Wurzelsystem im internen XML-Format

```
<?xml version="1.0" ?>
<Forest id="588" bo="26" cm="1023" cn="8" d="1.000000" dlx="9.999872"
    dly="8.000000" dlz="5.500000" sl="1.000000" zFac="1.000000" iso="16.000000"
    dataset="data/lupi.vtk" transferFunc="data/lup_red.xml">
  <Tree id="1">
    <Node id="0" bo="1" rad="0.013334" x="0.007318" y="-0.022008" z="-0.272744">
      <Node id="1" bo="1" rad="0.013045" x="0.012902" y="-0.025925" z="-0.228239">
        <Node id="2" bo="1" rad="0.016555" x="0.013156" y="-0.034805"
            z="-0.188168">
          <Node id="3" bo="1" rad="0.021244" x="0.014462" y="-0.041545"
              z="-0.158483">
            <Node id="4" bo="1" rad="0.018098" x="0.015071" y="-0.049518"
                z="-0.127584">
              <Node id="6" bo="1" rad="0.017547" x="0.015888" y="-0.052815"
                  z="-0.109456">
                <Node id="9" bo="1" rad="0.017582" x="0.016142" y="-0.056379"
                    z="-0.086444">
[..]
  </Tree>
</Forest>
```

Listing C.2: Teilstück der XML-Datei

C.3 Wurzelsystem im R-SMWS-Format

```
Time:
  0.00000

Number of seeds
  1

ID, X and Y coordinates of the seeds (one per line)
  0 0.00E+00 0.00E+00

Root DM, shoot DM, leaf area:
  0.000000  0.000000  0.000000

Average soil strength and solute concentration experienced by root system:
  0.000000  0.000000

Total # of axes:
  1

Total # of branches, including axis(es):
  26

Total # of segment records:
  518

segID#             x             y             z  prev  or   br#         length
          surface          mass
origination time
    1    5.5839e-02   -3.9169e-02   -4.4504e-01     0   1     1    4.5024e-01
        3.7313e-01    2.4607e-02
   0.0000e+00
    2    5.8379e-02   -1.2797e-01   -8.4575e-01     1   1     1    4.1043e-01
        3.8306e-01    2.8375e-02
   0.0000e+00
    3    7.1439e-02   -1.9537e-01   -1.1426e+00     2   1     1    3.0468e-01
        3.6606e-01    3.4364e-02
   0.0000e+00
    4    7.7529e-02   -2.7510e-01   -1.4516e+00     3   1     1    3.1916e-01
        3.9638e-01    3.8880e-02
   0.0000e+00
    5    8.5699e-02   -3.0807e-01   -1.6329e+00     4   1     1    1.8443e-01
        2.0662e-01    1.8406e-02
   0.0000e+00
    6    8.8239e-02   -3.4371e-01   -1.8630e+00     5   1     1    2.3287e-01
        2.5700e-01    2.2570e-02
   0.0000e+00
    7    1.4792e-01   -3.9325e-01   -2.0928e+00     6   1     1    2.4253e-01
        2.6991e-01    2.3901e-02
   0.0000e+00
    8    2.1965e-01   -4.5265e-01   -2.2658e+00     7   1     1    1.9647e-01
        2.1974e-01    1.9556e-02
   0.0000e+00
    9    2.2043e-01   -5.1199e-01   -2.4851e+00     8   1     1    2.2726e-01
        2.4678e-01    2.1290e-02
   0.0000e+00
[..]
```

Listing C.3: Ein Teilstück des Wurzelsystems im R-SWMS-Format

Jül-4307
September 2009
ISSN 0944-2952