Westfälische Wilhelms-Universität Münster Institut für Kernphysik

Diplomarbeit

Implementierung der Multipole Boundary Element Methode für das KATRIN-Experiment

Sebastian Vöcking

18. Januar 2008

Inhaltsverzeichnis

1 Einleitung

Seit ihrer Entdeckung im Jahre 1953 sind Neutrinos ein wichtiger Gegenstand der aktuellen Forschung. Sie sind von zentraler Bedeutung für verschiedene Forschungsgebiete auch über die Elementarteilchenphysik hinaus. Derzeit wird beispielsweise am Südpol mit Icecube eines der größten Neutrinoteleskope weltweit errichtet. Die Astronomen erhoffen sich hiervon vööllig neue Erkenntnisse über kosmologische Vorgänge.

Im Mittelpunkt des allgemeinen Interesses steht momentan jedoch die Frage nach der Masse des Neutrinos. Im Standardmodell wird es als masselose Elementarteilchen postuliert. Doch in den letzten Jahren mehrten sich die Evidenzen, dass es doch eine von null verschiedene Masse besitzt. Zum Beispiel kann die von mehreren Experimenten voneinander unabhängig beobachtete Neutrinooszillation nur mit einer endlichen Neutrinomasse erklärt werden. Zur Untersuchung dieser Frage gibt derzeit verschiedene Ansätze. Kosmologische Beobachtungen und des neutrinolosen Doppelbeta-Zerfalls liefern Hinweise auf die Größe der Masse des Neutrinos. Diese Untersuchungen stützen sich jedoch alle auf verschiedene theoretische Modellansätze. Ein rein kinematischer und damit modellunabhängiger Ansatz stellt die Vermessung des Energiespektrums des β-Zerfalls dar. Dessen exakte Form in der Nähe des Endpunkts hängt allein von der Masse des bei diesem Prozess entstehenden Neutrinos ab.

Das neueste Experiment, das sich dieser Messmethode bedient, ist das Kalsruher Tritium-Neutrinoexperiment (KATRIN). Es wird die Energieverteilung der Zerfallselektronen spektrometrisch mit einem MAC-E-Filter ermitteln und so neue Hinweise zur Größe der Neutrinomasse liefern. KATRIN besitzt eine Sensitivität hinreichend, um die Obergrenze für die Masse des Elektronneutrinos auf 0,2 eV zu senken oder die Masse mit einer Empfindlichkeit von 0,3 eV zu entdecken. Dabei ist der Einsatz einer speziellen Drahtelektrode zur Untergrundunterdrückung wichtig.

Bei der Integration dieser Drahtelektrode in das Spektrometer gilt es, die dadurch in den elektrischen Potentialen auftretenden Störungen zu minimieren. Dies geschieht mit aufwändigen Computersimulationen, für die eigens neue Programmen wntwickelt werden. Im Rahmen dieser Arbeit wurde dieses Softwarepaket verbessert und erweitert. Der Schwerpunkt liegt dabei auf der Entwicklung eines neuen Algorithmus zu Berechnung elektrischer Felder, der ermöglich noch detailgetreuere Simulationen durchzuführen.

Gliederung der Arbeit

Der Inhalt der vorliegenden Arbeit ist folgendermaßen gegliedert:

- Kapitel 2 gibt ein kurzen Überblick über den theoretischen Hintergrund dieser Arbeit. Die Eigenschaften des Neutrinos, sowie die Neutrinooszillation als Beleg für eine von null verschiedene Masse werden vorgestellt. Zudem wird das Verfahren der direkten Massenbestimmung erläutert.
- In Kapitel 3 wird der Aufbau und die Funktionsweise des KATRIN-Experiments beschrieben. Der Schwerpunkt liegt dabei auf den Anforderungen an das elektromagnetische Design.
- Kapitel 4 beschäftigt sich mit den zukünftigen Simulation zur globalen Bahnverfolgung und den dafür nötigen Vorbereitung, sowohl software- als auch hardwareseitig.
- Für realistische Simulationen des KATRIN-Experiments sind detailgetreue Computermodelle nötig. Im Rahmen dieser Arbeit wurden Programme zur Generierung und Visualisierung dieser Modelle entwickelt, welche in Kapitel 5 vorgestellt werden.
- Durch die immer komplexeren Simulationen geraten die verwendeten Computerprogramme an ihre Grenzen. In Kapitel 6 wird die Implementierung eines neue Algorithmus zur elektrischen Feldberechnung beschrieben, der die momentan verwendete Boundary Element Methode erweitert.
- Dieser Algorihmus wurde an verschiedenen Elektrodengeometrien getestet. Die Resultate dieser Tests werden in Kapitel 7 aufgeführt.
- Als Abschluss der Arbeit folgt in Kapitel 8 eine kurze Zusammenfassung.

2 Neutrinophysik

2.1 Die Rolle des Neutrinos in der Elementarteilchenphysik

Das Neutrino wurde 1930 von Wolfgang Pauli zur Erklärung des β-Zerfalls postuliert. Dieser besitzt im Gegensatz zum α- und γ-Zerfall ein kontinuierliches Energiespektrum, welches in einem Zwei-Körper-Zerfall nur unter Nichterhaltung der Energie möglich wäre. Ebenso schien die Drehimpulserhaltung verletzt, da aus einem Spin-$1/2$-Teilchen offenbar zwei Spin-$1/2$-Teilchen entstehen. Daher postulierte Pauli das Neutrino als masseloses, elektrisch neutrales Spin-$1/2$-Teilchen. Dies sollte beim β-Zerfall zusammen mit dem Elektron entstehen und die Energie- und Drehimpulsbilanz ausgleichen. Die experimentelle Bestätigung erfolgte erst 1956 durch Reines und Cowan. Ihnen gelang es erfolgreich, einen inversen β-Zerfall mit Reaktorneutrinos zu beobachten [Sch97].

Heutzutage findet man das Neutrino im theoretischen Rahmen des Standardmodells, welches die Elementarteilchen und ihre Wechselwirkungen erklärt. Nach dem Standardmodell ist die gesamte Materie aus Quarks und Leptonen aufgebaut. Zusätzlich existiert zu jedem Teilchen noch ein Antiteilchen, welches die gleiche Ruhemasse besitzt, aber umgekehrte Ladungen. Alle diese Teilchen unterliegen der elektromagnetischen, der schwache und die Quarks auch noch der starken Wechselwirkung. Dies sind die drei elementaren Wechselwirkungen. Die vierte Kraft, die Gravitation, kann nicht im Rahmen des Standardmodells erklärt werden.

Das Neutrino zählt zu den Leptonen. Es trägt keinerlei Ladung, daher kann es nur schwach wechselwirken. Dies erklärt den geringen Wirkungsquerschnitt bei allen Neutrinoreaktionen. Da bei der schwachen Wechselwirkung die Leptonenzahl erhalten bleibt, entsteht beim β^--Zerfall ein Antineutrino. Die genaue Reaktionsformel lautet:

$$\mathrm{n} \rightarrow \mathrm{p}^+ + \mathrm{e}^- + \bar{\nu}_e$$

Das Standardmodell ist eine äußerst erfolgreiche Theorie. Es wurde bereits durch viele Experimente bestätigt. Allerdings gibt es mittlerweile mehrere Hinweise auf eine Physik jenseits des Standardmodells. So kann es beispielsweise die Dunkle Materie, deren Existenz in der Theorie der Kosmologie gefordert wird, nicht erklären. Eine weitere Unzulänglichkeit des Standardmodells zeigt sich in der Frage der Neutrinomasse.

2.2 Neutrinooszillation

Lange ging man davon aus, dass die Neutrinomasse null beträgt, wie vom Standardmodell vorgegeben. Doch bis heute hat man durch verschiedene Experimente Hinweise erhalten, dass das Neutrino eine von null verschiedene Masse besitzen muss. So lag die gemessene Zählrate der solaren Neutrinos nur bei einem Drittel der theoretischen Vorhersage. Die fehlenden Neutrinos lassen sich dadurch erklären, dass sich ein Teil der Elektronneutrinos auf dem Weg von der Sonne zur Erde in einen anderen Flavour-Zustand umwandelt. Dieser Effekt wird als Neutrinooszillation bezeichnet. Er beruht darauf, dass sowohl die Flavour-Eigenzustände $\{|\nu_\alpha\rangle, \alpha = \mathrm{e}, \mu, \tau\}$ als auch die Massen-Eigenzustände $\{|\nu_i\rangle, i = 1, 2, 3\}$ eine Basis des Zustandsraums darstellen. Diese beiden Basen stimmen allerdings nicht überein. Das bedeutet es gibt eine unitäre Transformation U mit

$$|\nu_\alpha\rangle = \sum_i U_{\alpha i}|\nu_i\rangle. \tag{2.1}$$

Man kann somit die Flavour-Eigenzustände als eine Linearkombination der Massen-Eigenzustände schreiben. Da der Operator U unitär ist, was bedeutet, dass $U^{-1} = U^*$ ist, sind lediglich $(n-1)^2$ der n^2 Komponenten dieser Matrix unabhängig. Oft wählt man als unabhängige Parameter die $\frac{1}{2}n(n-1)$ Drehwinkel einer n-dimensionalen Rotationsmatrix und $\frac{1}{2}(n-1)(n-2)$ komplexe Phasen. Diese Wahl ist allerdings rein willkürlich und dient nur der Veranschaulichung. Für den Drei-Flavour-Fall wird daher die Matrix oft in dieser Form geschrieben

$$U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_2 & s_2 \\ 0 & -s_2 & c_2 \end{pmatrix} \cdot \begin{pmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & e^{i\delta} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_3 & s_3 \\ 0 & -s_3 & c_3 \end{pmatrix} \tag{2.2}$$

mit $s_i = \sin\vartheta_i$ und $c_i = \cos\vartheta_i$. Den Operator U bezeichnet man als Mischungsmatrix und die Winkel ϑ_i als Mischungswinkel. Die Schreibweise ist analog zu der CKM-Matrix in der Theorie der starken Wechselwirkung.

Die zeitliche Entwicklung eines Massen-Eigenzustands $|\nu_i\rangle$ mit der Energie E_i folgt aus der Schrödingergleichung ($\hbar = 1$ und $c = 1$)

$$|\nu_i(t)\rangle = e^{-iE_i t}|\nu_i\rangle \tag{2.3}$$

Im ultrarelativistischen Fall gilt ($c = 1$):

$$E_i = \sqrt{p_i^2 + m_i^2} \approx E + \frac{m_i^2}{2E}. \tag{2.4}$$

Angenommen, ein Neutrino befände sich zur Zeit $t = 0$ in einem der Flavour-Eigenzustände $|\nu(0)\rangle = |\nu_\alpha\rangle$, so würde für die Zeitentwicklung dessen Zustands gelten

$$|\nu(t)\rangle = \sum_i U_{\alpha i} e^{-iE_i t}|\nu_i\rangle = \sum_{i,\beta} U_{\alpha i} U^*_{\beta i} e^{-iE_i t}|\nu_\beta\rangle. \tag{2.5}$$

Der zu Beginn reine Zustand vermischt sich für $t > 0$ mit den anderen Flavour-Eigenzuständen. Die Wahrscheinlichkeit für einen Übergang beträgt:

$$\begin{aligned} P(\alpha \to \beta; t) = \langle \nu(t) | \nu_\beta \rangle^2 &= \left| \sum_i U_{\alpha i} U_{\beta i}^* e^{-iE_i t} \right|^2 \\ &= \sum_i |U_{\alpha i} U_{\beta i}^*|^2 + 2\,\mathrm{Re} \sum_{j>i} U_{\alpha i} U_{\alpha j}^* U_{\beta i}^* U_{\beta j} e^{-i(E_i - E_j)t}. \end{aligned} \tag{2.6}$$

Man kann einen oszillierenden Anteil an der Übergangswahrscheinlichkeit erkennen. Die Phase dieser Oszillation lässt sich auf die Flugstrecke $L = t$ umrechnen, da der ultrarelativistische Grenzfall betrachtet wird:

$$(E_i - E_j)t \approx \frac{m_i^2 - m_j^2}{2} \frac{L}{E} =: \frac{\Delta m_{ij}^2}{2} \frac{L}{E} \tag{2.7}$$

Aus der Frequenz dieser Oszillation lässt sich Δm_{ij}^2 bestimmen. Die Neutrinooszillation liefert somit Informationen über die Differenzen der Massenquadrate der Masseneigenzustände $|\nu_i\rangle$. Sie macht jedoch keine Aussage über den Absolutbetrag der Massenskala.

Für das Auftreten von Neutrinooszillation müssen zwei Bedingungen erfüllt sein. Einerseits muss wenigstens ein Massen-Eigenzustand einen endlichen Wert besitzen, andererseits darf die Mischungsmatrix U nicht gleich der Einheitsmatrix sein. Daher ist die Beobachtung der Neutrinooszillation eine klare Evidenz für eine von null verschiedene Neutrinomasse.

Aufgrund dieser großen Bedeutung wurde dieser Effekt bereits von mehreren Experimenten an verschiedenen Neutrinoquellen untersucht. So wurde 1998 vom Super-Kamiokande-Experiment in Japan die Neutrinooszillation bei atmosphärischen Neutrinos, die bei Reaktionen der kosmischen Strahlung mit der Atmosphäre entstehen, entdeckt. Der Experimentaufbau besteht in Wesentlichen aus einem Tank, der mit 50 kt ultrareinem Wasser gefüllt ist und sich 1 km unter der Erdoberfläche befindet. Dringt ein Neutrino in dieses Wasservolumen ein, kann es mit einem Nukleon wechselwirken:

$$\nu_\alpha + \mathrm{n} \to \alpha^- + \mathrm{p}^+ \quad \text{bzw.} \quad \bar{\nu}_\alpha + \mathrm{p}^+ \to \alpha^+ + \mathrm{n} \tag{2.8}$$

mit $\alpha = \mathrm{e}, \mu$. Die dabei entstehenden geladenen Leptonen erzeugen charakteristische Čerenkov-Strahlung, die von mehreren tausend Photomultipliern an den Wänden des Wassertanks detektiert wird. Dabei wurde ein zenitwinkelabhängiges Defizit an ν_μ festgestellt. Rechnet man die Winkelabhängigkeit auf die Flugstrecke in der Atmosphäre um, stimmt das beobachtete Defizit genau mit den Erwartungen aus der Theorie der Neutrinooszillation für eine bestimmte Kombination von Parametern überein. Demnach gelte $\Delta m_{12}^2 = 7,9 \cdot 10^{-5}\,\mathrm{eV}^2$, $\Delta m_{23}^2 = 2,6 \cdot 10^{-3}\,\mathrm{eV}^2$, $\vartheta_1 = 45°$, $\vartheta_2 = 32°$ und $\vartheta_3 = 13°$.

2.3 Direkte Bestimmung der Neutrinomasse

Die genaue Form des Energiespektrums des β-Zerfalls hängt von der Masse des Elektronneutrinos $m(\nu_e)$ ab. Da diese allerdings nicht exakt definiert ist, ist das Spektrum eine Überlagerung von den drei Einzelspektren zu den jeweiligen Masseneigenzuständen. Die Differenzen zwischen diesen Spektren sind so gering, dass sie mit den heutigen Messmethoden nicht aufgelöst werden können. Stattdessen wird ein Spektrum mit einer effektiven Neutrinomasse $m(\nu_e)$ beobachtet, für die gilt

$$m(\nu_e)^2 := \sum_{i=1}^{3} |U_{ei}|^2 m(\nu_i)^2. \tag{2.9}$$

Die Ruheenergie des Neutrinos steht den Zerfallselektronen nicht zur Verfügung; somit bestimmt sie den Endpunkt des Spektrums E_{max}. Wenn E_0 die beim Zerfall freiwerdende Energie ist, beträgt die maximale Energie der Zerfallselektronen

$$E_{max} = E_0 - m(\nu_e)c^2 \tag{2.10}$$

Somit würde eine präzise Bestimmung der maximalen Energie E_{max} den Betrag der Neutrinomasse liefern. Doch aufgrund der extrem geringen Zählrate und dem nur zu einem gewissen Grad unterdrückbaren Untergrund in diesem Energiebereich ist dies nicht mit ausreichender Genauigkeit möglich. Die ebenfalls zur Bestimmung benötigte Zerfallsenergie E_0 ist in der Regel auch nicht genau genug bekannt. Aus diesen Gründen ist die Methode zur Massenbestimmung ungeeignet. Stattdessen ist die genaue Form des β-Spektrums in der Nähe des Endpunkts zu untersuchen. Für das Spektrum gilt

$$\frac{\mathrm{d}\dot{N}}{\mathrm{d}E} = R(E)\sqrt{(E_0 - E)^2 - m(\nu_e)^2c^4}\,\Theta(E_0 - E - m(\nu_e)c^2) \tag{2.11}$$

mit

$$R(E) = \frac{G_F^2}{2\pi^3\hbar^7}\cos^2\theta_C |M|^2 F(Z,E) p(E + m_e c^2)(E_0 - E). \tag{2.12}$$

Dabei ist E die Energie und p der Impuls des Zerfallselektrons, Z die Ladung des Tochterions, G_F die Fermi-Konstante, θ_C der Cabibbo-Winkel und M das Kernmatrixelement für den Übergang. $F(Z,E)$ ist die Fermi-Funktion, welche die elektrostatische Wechselwirkung beschreibt. Vermisst man nun das Spektrum in der Nähe des Endpunkts sehr präzise, erhält man E_0 und $m(\nu_e)^2$ durch Anfitten des Spektrums an die Messdaten. Diese Methode ist modellunabhängig, da sie ausschließlich auf kinematischen Messmethoden basiert.

Dieses Verfahren wurde bereits in mehreren Experimenten erfolgreich angewendet. Bei diesen Experimenten wurde fast ausschließlich Tritium als radioaktives Material verwendet. Die dabei auftretende Zerfallsreaktion entspricht

$$^3\mathrm{H} \rightarrow {}^3\mathrm{He}^+ + \mathrm{e}^- + \bar{\nu}_e. \tag{2.13}$$

Aus folgenden Gründen ist diese Reaktion optimal für derartige Untersuchungen:

- Der Endpunkt des Energiespektrums E_0 liegt mit 18,6 keV sehr niedrig. So fallen relativ viele Zerfallsereignisse in das zu betrachtende Energieintervall.
- Die Halbwertszeit ist mit 12,3 Jahren relativ lang, bietet aber dennoch eine genügend große Aktivität bei einer eben noch handhabbaren an radioaktiven Material.
- Das Tritiummolekül T_2 hat eine sehr einfache elektronische Struktur, daher ist das elektronische Anregungsspektrum gut bekannt.

Die aktuelle Obergrenze für die Masse des Elektronneutrinos beträgt 2,3 eV mit 95 % und wurde durch das Mainz-Experiment ermittelt [Kra03]. Sie wurde mit dem hier vorgestellten Verfahren ermittelt.

3 Das KATRIN-Experiment

Das KATRIN-Experiment ist das neuste Experiment zur direkten Bestimmung der Neutrinomasse und der direkte Nachfolger der Experimente in Mainz und Troitsk. Wie bei den Vorgängerexperimenten erfolgt die Massenbestimmung durch eine genaue Vermessung der Energiespektrums des Tritiumzerfalls. Mit der angestrebten Sensitivität kann eine neue Obergrenze für die Masse des Elektronneutrinos von 0,2 eV mit einer Sicherheit von 90 % erreicht werden. Eine Masse von 0,35 eV könnte mit einer Genauigkeit von $5\,\sigma$ bestimmt werden, bei 0,3 eV wären es $3\,\sigma$.

3.1 Experimenteller Aufbau

Das KATRIN-Experiment wird am Forschungszentrum Karlsruhe aufgebaut. Dort existiert die optimale Infrastruktur für ein Experiment dieser Größenordnung. Das am Forschungszentrum ansässige Tritiumlabor Karlsruhe (TLK) stellt als einzige zivile Einrichtung weltweit die erforderliche Menge reinen Tritiums bereit. Daher wird zur Zeit das KATRIN-Experiment in unmittelbarer Nähe zum TLK errichtet. Der gesamte Aufbau hat eine Länge von etwa 70 m ist in Abbildung 3.1 gezeigt und besteht aus mehreren Komponenten, die im Folgenden vorgestellt werden.

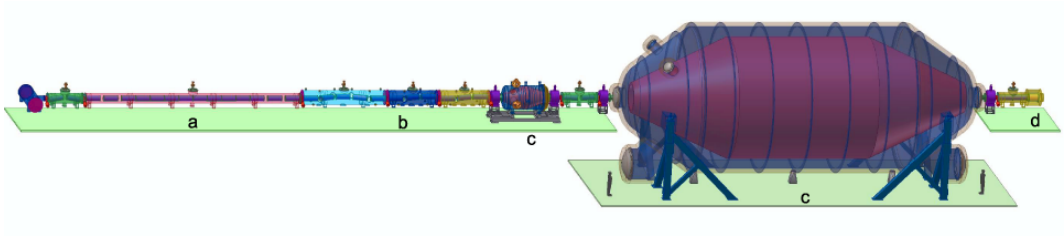

Abbildung 3.1: Gesamtaufbau des KATRIN-Experiments

Tritiumquelle

Für das KATRIN-Experiment wird eine fensterlose Quelle mit gasförmigem Tritium (Windowless Gaseous Tritium Source, WGTS) verwendet. Mit dieser Art von Quelle erreicht man eine hohe Zählrate ohne störende Festkörpereffekte. Der Quellaufbau besteht aus einem 10 m langen Rohr mit einem Durchmesser von 9 cm. Sie ist auf eine Temperatur von 27 K heruntergekühlt und befindet sich in einem Magnetfeld der Stärke 3,6 T, das von supraleitenden Magneten erzeugt wird und die Elektronen führen soll. Das Tritium strömt in der Mitte ein und wird an den Seiten wieder abgepumpt und gereinigt. Auf diese Weise ist es möglich, es in einem geschlossenen

Kreislauf zu halten und dadurch eine hohe Intensität mit einer geringen Menge Tritiums zu erreichen.

Transportstrecke

Die Transportstrecke schließt sich an die WGTS an. Sie besteht aus der differentiellen Pumpstrecke DPS2-F und der kryogenen Pumpstrecke CPS. Sie soll verhindern, dass molekulares Tritium aus der Quelle in den übrigen Experimentaufbau gelangt. Dies wird in der DPS2-F durch differentielles Pumpen und in der CPS durch Auffrieren erreicht. Zusätzlich ist die Strahlführung geknickt um zu verhindern, dass sich Tritium geradlinig durch diese Sektion bewegen kann. Dieser Effekt wird auch als Beaming-Effekt bezeichnet. Mehrere Solenoide erzeugen ein hohes Magnetfeld, das hingegen die Zerfallselektronen in das Vorspektrometer geführt.

Vorspektrometer

Das Vorspektrometer befindet sich zwischen Transportstrecke und Hauptspektrometer. Es hat eine Länge von 3,5 m und einen Durchmesser von 1,7 m. Seine Hauptaufgabe ist die grobe Vorfilterung des Elektronenflusses aus der Quelle. Es besitzt zwar nur ein Auflösungsvermögen von etwa 75 eV, jedoch genügt dies, um den Gesamtfluss an dieser Stelle bereits um einen Faktor 10^7 zu reduzieren. Durch diese Filterung trägt es zur Untergrundreduzierung bei. Eine Beschreibung dieses Spektrometertyps folgt in Kapitel 3.2.

Hauptspektrometer

Eine zentrale Komponente des KATRIN-Experiments ist das Hauptspektrometer. Es besteht aus einem Edelstahltank einer Länge von 23 m und einem Durchmesser von 10 m. Es wurde von der Firma MAN/DWE in Deggendorf gefertigt und wurde dann aufgrund der enormen Größe und des Gewichts von 200 t über den Seeweg über das Schwarze Meer, dem Mittelmeer und dem Atlantik nach Karlsruhe transportiert.

Die Aufgabe des Hauptspektrometers ist die exakte Vermessung des Tritium-β-Zerfallsspektrums. Dies geschieht ebenso wie beim Vorspektrometer nach dem Prinzip des MAC-E-Filters.

An der Innenseite des Hauptspektrometertanks soll eine zusätzliche Drahtelektrode zur Untergrundunterdrückung montiert werden. Eine Beschreibung ihres Aufbaus und ihrer Funktionenweise folgt in Kapitel 3.2. Im Vorspektrometer kommt eine ähnliche, aber wesentlich einfachere Drahtelektrode zum Einsatz.

Detektor

Das letzte Element des KATRIN-Experiments ist der Detektor. Die sensitive Fläche hat einen Durchmesser von etwa 9 cm und dient dazu, die Elektronen nachzuweisen, die das Retardierungspotential des Hauptspektrometers überwunden haben. Dazu ist im Wesentlichen eine hohe Nachweiseffizienz für Elektronen mit einer Energie von ungefähr 18,6 keV notwendig. Der Detektor besteht aus einem Array von PIN-Dioden und ist in 145 Pixel in einer Anordnung ähnlich einer Dartscheibe unterteilt. Damit kann zusätzlich zur Energieauflösung von ungefähr 600 eV eine Ortsauflösung erreicht werden, mit der das Strahlprofil untersucht werden kann. So können Inho-

mogenitäten im elektrischen Potential und dem Magnetfeld in der Analysierebene des Hauptspektrometers vermessen werden und herausgerechnet werden.

3.2 Elektromagnetisches Design

Das elektromagnetische Design des KATRIN-Experiments umfasst die geometrische Anordnung von Elektroden und Magnetspulen, sowie deren Potentialbelegung und Stromdichten. Die Entwicklung eines Designs, das den hohen Anforderungen von KATRIN genügt, stellt eine große Herausforderung dar. Die Energieauflösung der beiden Spektrometer, die den zentralen Bestandteil des KATRIN-Experiments bilden, hängt entscheidend vom elektromagnetischen Design ab. Sie funktionieren beide nach dem Prinzip des MAC-E-Filters. Diese Abkürzung steht für Magnetic Adiabatic Collimation and Electrostatic Filter. Ein derartiger Filter besteht aus zwei an den Enden des Spektrometers befindlichen Solenoid-Magneten, die ein stark inhomogenes Magnetfeld erzeugen. Die Feldstärke fällt zur Analysierebene hin um mehrere Größenordnungen ab. B_S sei das Magnetfeld am Eingang des Spektrometers und B_{min} das Feld in der Analysierebene, die genau in der Mitte zwischen den Spulen liegt. Der Aufbau ist skizziert in Abbildung 3.2.

Geladene Teilchen bewegen sich in einem Magnetfeld entlang der Magnetfeldlinien auf Zyklotronbahnen. Ihre Energie E lässt sich unterteilen in eine transversale $E_\perp$ und eine logitudinale Komponente $E_\parallel$ bezüglich des Magnetfeldes. Aufgrund der Zyklotronbewegung kann man dem Teilchen auch ein magnetisches Moment $\vec{\mu}$ zuordnen. Es gilt in nichtrelativistischer Näherung:

$$E_\perp = -\vec{\mu} \cdot \vec{B}. \tag{3.1}$$

Falls die adiabatische Näherung erfüllt ist, das bedeutet, dass die relative Änderung des Magnetfeldes $\Delta B/B$ pro Zyklotronumlauf klein ist, bleibt $\vec{\mu}$ konstant. In diesem Fall ist die Transversalenergie proportional zur Magnetfeldstärke B. Fliegt ein Teilchen also in das Spektrometer hinein, wird dessen Transversalenergie vom hohen zum schwachen Magnetfeld hin in Longitudinalenergie umgewandelt. Betrug sie im maximalen Magnetfeld noch $E_\perp^0$, so gilt in der Analysierebene

$$E_\perp^{min} = E_\perp^0 \frac{B_{min}}{B_S}. \tag{3.2}$$

Zusätzlich zum Magnetfeld liegt noch das elektrisches Retardierungspotential U an, dessen Maximum genau in der Analysierebene liegt. Dieses Potential kann nur von Elektronen überwunden werden, für die gilt $E_\parallel > eU$. Die transversale Komponente $E_\perp$ spielt dabei keine Rolle. Aus dem Grund muss sie möglichst vollständig in Longitudinalenergie umgewandelt werden, bevor die Elektronen die Analysierebene erreichen. Daher lässt sich aus (3.2) die Energieauflösung des MAC-E-Filters ΔE ableiten. Angenommen ein Elektron startet unter einem Winkel von 90° zu Magnetfeldlinie, so gilt zu Beginn $E_0 = E_\perp^0$. In der Analysierebene gilt dann

$$\Delta E = E_\perp^{min} = E_0 \frac{B_{min}}{B_S}. \tag{3.3}$$

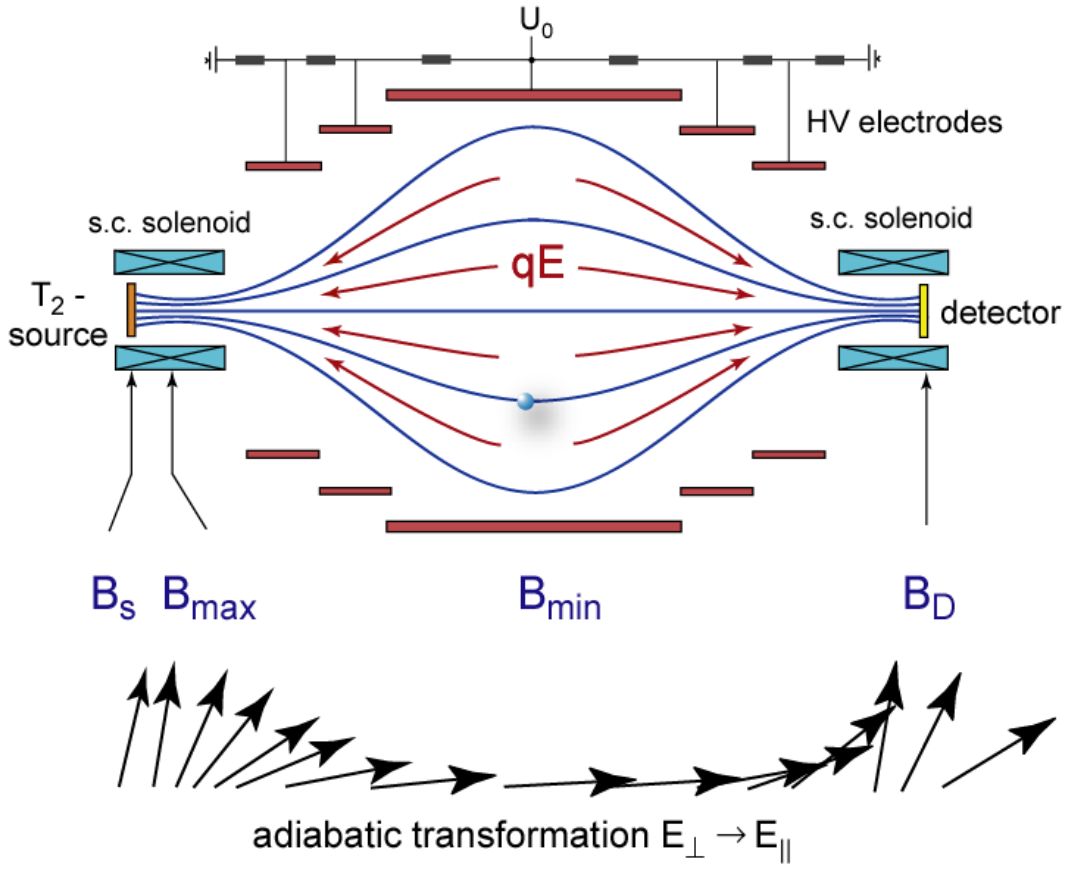

Abbildung 3.2: Funktionsweise eines MAC-E-Filters

Da diese restliche Transversalenergie durch die Analysierwirkung des Spektrometers nicht erfasst werden kann, entspricht sie der Energieauflösung ΔE. Man erkennt, dass ΔE nur vom Verhältnis der Magnetfeldstärken abhängt. Bei KATRIN wird ein Verhältnis von $B_S/B_{min} \approx 20000$ benötigt, um die angestrebte Sensitivität auf die Neutrinomasse im Sub-eV-Bereich zu erreichen. Die Energieauflösung am Endpunkt des Tritium-β-Spektrums wäre in diesem Fall $\Delta E \approx 1\,\mathrm{eV}$ und damit um einen Faktor 4–5 besser als bei den Vorgängerexperimenten. Das Verhältnis erklärt auch den großen Durchmesser des Hauptspektrometers, da sich der magnetische Flussschlauch in der Analysierebene entsprechend aufweitet. Die Länge des Spektrometers wird von der adiabatischen Näherung vorgegeben. Sie gilt nur solange die relative Magnetfeldänderung nicht zu groß wird. Daher wird bei diesen Magnetfeldstärken jeweils eine Strecke von über 10 m benötigt.

Aus dieser Funktionsweise ergeben sich mehrere Anforderungen an das elektromagnetischen Design. Einerseits muss sichergestellt werden, dass das Verhältnis der Magnetfeldstärken im Hauptspektrometer der Vorgabe entspricht. Gleichzeitig muss die Magnetfeldänderung im gesamten Hauptspektrometervolumen abiabatisch ver-

laufen. Die Magnetfeldstärke wird am Eingang des Hauptspektrometers 4,5 T betragen und dann zur Analysierebene hin auf etwa $3 \cdot 10^{-4}$ T = 3 G abfallen. Da das lokale Erdmagnetfeld bereits eine Feldstärke von $B \approx 0,5$ G aufweist, muss es entsprechend kompensiert werden. Zu diesem Zweck ist eine komplexe Anordnung von Luftspulen notwendig, die um das Spektrometer herum angeordnet sind und einen Durchmesser von 12 m besitzen. Zusätzlich zum Magnetfeld muss aber auch ein genau darauf abgestimmtes elektrisches Feld erzeugt werden. Das Maximum des Retardierungspotentials muss genau in der Analysierebene liegen, in der das Magnetfeld sein Minimum erreicht. Ist dies nicht der Fall, hat sich die Transversalenergie der Elektronen nicht vollständig in Longitudinalenergie umgewandelt und die angestrebte Energieauflösung kann nicht erreicht werden. Für eine optimale Retardierung der über den gesamten Flussschlauch verstreuten Zerfallselektronen sind insbesondere auch homogene Felder entlang der gesamten Analysierebene notwendig, dies gilt sowohl für das elektrische als auch das magnetische Feld.

Neben den Anforderungen an das elektromagnetische Design gibt die angestrebte Energieauflösung auch die Dimensionen des Hauptspektrometers vor, woraus die enorme Masse von 200 t Edelstahl folgt, die zur Konstruktion des Hauptspektrometers benötigt wurde. Demzufolge ist das Hauptspektrometer selbst eine nicht zu vernachlässigende Untergrundquelle. Einerseits führen instabile Isotope im Edelstahl zu radioaktiven Zerfällen in der Tankwand und anderseits treffen kosmische Myonen auf das Tankvolumen und erzeugen dort Sekundärelektronen. Beide Effekte führen zu unerwünschten Untergrundelektronen, die zwar zu einem großen Teil durch das Magnetfeld abgeschirmt werden, da es nur Elektronen innerhalb des magnetischen Flussschlauches zum Detektor führt. Ein kleiner Anteil der Sekundärelektronen gelangt dennoch zum Detektor. Der so entstehende Untergrund wäre immer noch zu groß. Daher muss im Tank eine zusätzlich Drahtelektrode installiert werden, die diesen Untergrund weiter reduziert. Sie besteht aus zwei Drahtlagen mit einem Abstand von 15 mm und 22 mm zur Tankwand. Während das elektrische Potential der Wand -18600 V beträgt, liegt die äußere Drahtlage bei -18500 V und die innere bei -18400 V. Niederenergetische Elektronen, die aus der Tankwand austreten, werden dadurch reflektiert. Diese Elektronen entstehen bei radioaktiven Zerfällen und durch kosmische Myonen, die auf die Tankwand treffen. Um zu die Anzahl der Elektronen zu minimieren, die aus den Drähten selbst austreten, haben diese lediglich einen Durchmesser von 0,3 mm in der äußeren Lage und 0,2 mm in der inneren Lage. Diese Zusatzelektrode stellt eine große Herausforderung bei der Entwicklung des elektromagnetischen Designs dar. Die Montage der Drahtelektrode erfolgt in 240 Modulen, die an speziellen Bolzen an der Tankwand befestigt werden. Ein Modul besteht aus zwei Kämmen zwischen denen die Drähte gespannt sind. Abbildung 3.3 zeigt einen Kamm aus dem zylindrischen Spektrometerteil. Diese Kämme stören allerdings die Homogenität des elektrischen Potentials im Tank, insbesondere führen sie zu Inhomogenitäten in der Analysierebene. Dieser Effekt muss durch zusätzliche Abschirmelektroden minimiert werden. Die Entwicklung dieser Elektroden und Haltestrukuren sowie die generelle Optimierung der Potentialbelegung im Hauptspektrometer erfolgen mittels eines eigens zu diesem Zweck entwickelten Softwarepakets. Nähere Informationen dazu

folgen in Kapitel 4.2.

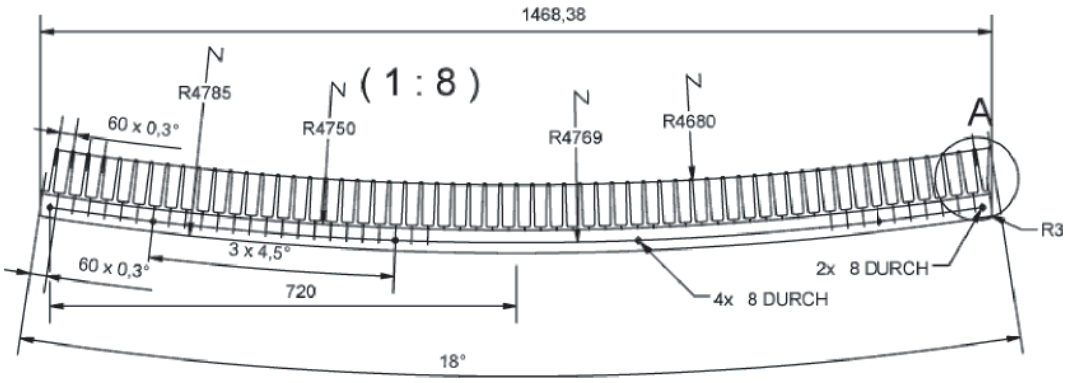

Abbildung 3.3: Kamm aus dem zylindrischen Spektrometerteil

Die abschirmende Wirkung einer Drahtelektrode wurde bereits erfolgreich am Mainzer Spektrometer nachgewiesen [Fla04]. Im Vorspektrometer kommt eine ähnliche, aber wesentlich einfachere Drahtelektrode zum Einsatz. Ein weitere wichtige Funktion dieser Elektrode ist der so genannte Dipolmodus. Dabei wird ein starkes elektrisches Feld erzeugt, welche gespeicherte Teilchen aus dem Hauptspektrometer entfernt.

4 Globale Bahnverfolgung

4.1 Bedeutung

Als globale Bahnverfolgung wird die numerische Berechnung der Flugbahn eines geladenen Teilchens durch den kompletten Aufbau des KATRIN-Experiments bezeichnet. Mit dieser Art von Simulation lässt sich ein elektromagnetisches Design unter realistischen Bedindungen testen. Dabei wird die mikroskopische Bewegung des Teilchens in den elektromagnetischen Feldern untersucht. Bisherige Simulationen in diesem Maßstab haben die Flugbahn lediglich in abiatischer Näherung bestimmt. Die Approximation besteht darin, dass nur die Bahn des Zentrums der Zyklotronbewegung des Teilchens bestimmt wird. Für nähere Informationen zu diesen Rechnungen sei auf [Thu02] verwiesen. Dadurch, dass hierbei jede Elektrode und jeder Magnet mit einbezogen werden, können auch Abweichungen gefunden werden, die durch das Zusammenspiel mehrerer verschiedener Komponenten entstehen. Solche Fehler können bei lokalen Simulationen nur sehr schwer entdeckt werden. Zusätzlich lassen sich nur auf diese Weise Toleranzrechnungen der gesamten Strahlführung durchführen.

Eine weitere wichtige Rolle wird die globale Bahnverfolgung bei der Datenanalyse spielen. Kombiniert mit einer Monte-Carlo-Simulation der β-Zerfälle und den Energieverlusten in der WGTS sowie des Elektronnachweises im Detektor, bildet sie die Grundlage für eine vollständige Simulation des KATRIN-Experiments. Mit einer solchen Simulation lassen sich die Auswirkungen verschiedener Störungen wie Potentialabweichungen oder radioaktive Zerfälle in der Tankwand auf die Detektorzählrate ermitteln. Man kann eine ortsaufgelöste Responsefunktion des Experiments ermitteln, welche Energieverluste, nicht-adiabatische Effekte sowie auftretende Synchrotronstrahlung beinhaltet. Für die Analyse bedeutet dies, dass diese Effekte korrigiert werden können und die Neutrinomasse mit einer höheren Genauigkeit gemessen werden kann.

Mit der bisher verfügbaren EMD-Software kann allerdings noch keine globale Bahnverfolgung durchgeführt werden. Daher wurde sie im Rahmen dieser Diplomarbeit optimiert und erweitert, so dass sie in Zukunft für diese Form der Simulation geeignet ist.

4.2 EMD-Software

Die Programme, welche bei der Entwicklung eines elektromagnetischen Design für KATRIN eingesetzt werden, wurden speziell für diesen Zweck geschrieben. Sie stam-

men hauptsächlich von Ferenc Glück (Universität Mainz, jetzt Forschungszentrum Karlsruhe) und sind in der Programmiersprache C geschrieben. Die Programme sind hoch optimiert und perfekt an die Problematik angepasst. Daher können sie auch nicht durch allgemeinere kommerzielle Pakete, wie *Simion* oder *CPO*, ersetzt werden.

Mit den EMD-Programmen ist es möglich die elektromagnetischen Felder, die aus einer Magnet- und Elektrodenkonfiguration resultieren, zu berechnen. Für die Magnetfeldbestimmung gibt es die Programme *Magfield2* und *Magfield3*. Sie sind sehr ähnlich aufgebaut, unterscheiden sich aber vor allem in der erlaubten Spulengeometrie. *Magfield2* kann nur Spulen verarbeiten, die auf der z-Achse liegen, insbesondere sind keine Neigungen erlaubt, während hingegen bei *Magfield3* die Spulenanordnung beliebig sein kann. Natürlich ist die allgemeinere Berechnung deutlich langsamer als bei *Magfield2*. Für eine Beschreibung des zugrunde liegenden Algorithmus sei auf [Hug08] verwiesen.

Die Bestimmung von elektrostatischen Potentialen erfolgt mit *Elcd3_2* und *Elcd3_3*. Beide Programme verwenden dazu die Boundary Element Methode, welche in Kapitel 6.2.1 beschrieben wird. Sie unterscheiden sich in den Elektrodentypen, die berechnet werden können. *Elcd3_2* unterstützt konusförmige Elektroden entlang der z-Achse und Drähte, die rotationssymmetrisch um die Achse angeordnet sind. *Elcd3_3* hingegen kann statt der Koni beliebig im Raum positionierte Rechtecke verarbeiten. Allerdings existiert ein Hilfsprogramm, das einen Konus in kleine Rechtecke zerlegt. Auf diese Weise lassen sich Geometrien von *Elcd3_2* nach *Elcd3_3* übertragen.

Die Routinen zur Feldberechnung werden von verschiedenen Hilfsprogrammen verwendet, die hauptsächlich bei der Entwicklung des EMD eingesetzt werden. Mit ihnen können magnetische Feldlinien bestimmt und das elektrische Potential entlang dieser ausgegeben werden. Diese Form der Berechnung ist essentiell für die Suche nach Teilchenfallen im Aufbau des KATRIN-Experiments. Das Finden und Entfernen ist eine der wichtigsten Aufgaben bei der Entwicklungen eines elektromagnetischen Designs. In solchen Fallen werden geladene Teilchen gespeichert, was wie im voherigen Kapitel beschrieben den Untergrund deutlich erhöhen kann.

Das wichtigste Anwendungsprogramm aber ist das Bahnverfolgungsprogramm, welches für globale Bahnverfolgung erweitert werden soll. Es benutzt die oben genannten Programme zur Feldberechnung und simuliert dann die Bewegung geladener Teilchen in diesen Feldern mit einem Runge-Kutta-Algorithmus. Bei diesem Berechnungsverfahren wird die gesamte Flugbahn stückweise in möglichst kleinen Schritten berechnet. Für jeden dieser Schritte wird jeweils die Lorenzkraft auf das Teilchen aus den elektromagnetischen Feldern bestimmt. Für eine genauere Beschreibung des Verfahrens sei auf [Val04] verwiesen.

4.3 Erweiterungen

Bevor die EMD-Software erweitert werden kann, müssen noch einige allgemeine Schwierigkeiten beseitigt werden. Ein wichtiger Punkt ist die Speicherverwaltung in allen Programmen. Bei modernen Computersystemem ist der Arbeitsspeicher in Stack- und Heap-Speicher unterteilt. Zwischen beiden gibt es keinen physikalischen Unterschied, lediglich die interne Adressierung ist unterschiedlich. Durch diese Aufteilung ist aber die Menge an verfügbarem Stack-Speicher begrenzt, auch wenn der Rechner eigentlich über ausreichend RAM verfügt. Daher sollte ein großer Speicherbedarf mit Heap-Speicher gedeckt werden. Bei der Programmierung in C kann man beide Arten von Speicher verwenden. Gewöhnliche Variablen wie

```
int a;
double b[10];
```

werden im Stack angelegt. Sie werden vom Compiler erzeugt und automatisch nach der Benutzung wieder gelöscht und der belegte Speicher freigegeben. Die Anforderung von Heap-Speicher ist ein wenig komplizierter. Sie erfolgt mit der *malloc()*-Funktion. Mit

```
double *b = (double*) malloc(sizeof(double) * 10);
```

legt man ein Array von double-Zahlen im Heap an. Leider verliert man dadurch auch die automatische Speicherverwaltung. So muss man nach der Verwendung den Speicherplatz manuell wieder freigeben:

```
free(b);
```

Bisher arbeiteten die EMD-Programme ausschließlich mit Stack-Speicher, daher müssen sie so angepasst werden, dass nun der Speicher vom Heap bezogen wird. Dieser Schritt macht die Programme wesentlich portabler. Ohne diese Änderung waren sie auf Rechnern, die nicht genug Stack-Speicher bereitstellen, nicht ohne Weiteres lauffähig. Zusätzlich tritt noch die Einschränkung auf, dass Arrays im Stack-Speicher immer eine feste Größe besitzen müssen. Dies führte dazu, dass beispielsweise die Anzahl der Elektroden nach oben durch einen im Programm versteckten Parameter begrenzt war. Andererseits wurden die Arrays immer in der maximalen Größe angelegt, auch wenn tatsächlich nicht so viel Platz benötigt wurde. Bei Arrays auf dem Heap kann die Größe dagegen dynamisch bestimmt werden. Wird nun vor der Speicheranforderung zunächst die erfordliche Größe bestimmt, muss nur so viel Speicherplatz belegt werden, wie auch wirklich benötigt wird. Da das Anfordern von Arbeitsspeicher auch immer Rechenzeit beansprucht, ergibt sich dadurch eine kleine Zeitersparnis.

Am deutlichsten musste das Bahnverfolgungsprogramm erweitert werden, da es die Grundlage für zukünftige globale Bahnfolgungen werden sollte. Daher mussten neue Funktionen implementiert werden, so dass die komplette KATRIN-Geometrie verarbeitet werden konnte. Dies war mit der ursprünglichen Version nicht möglich,

denn sie arbeitete mit den Feldberechnungsprogrammen *Magfield2* und *Elcd3_2*. Aus diesem Grund konnte weder die gekrümmte Magnetfeldstrecke in der DPS2-F und der CPS noch die detaillierte Drahtelektrode im Hauptspektrometer simuliert werden. Diese Einschränkung wurde behoben durch eine Neuimplementierung der Feldberechnung, die nun je nach Benutzerwahl auf *Magfield2* oder *Magfield3* bzw. *Elcd3_2* oder *Elcd3_3* zurückgreift. Dadurch treten viele neue Parameter auf, die spezifiziert werden müssen. Um dem Benutzer an dieser Stelle Arbeit abzunehmen, wurde eine Konfigurationsdatei hinzugefügt, in der diese und weitere Parameter angegeben werden können. Ebenfalls mit dieser Konfigurationsdatei kann eine weitere Erweiterung der Bahnverfolgung aktiviert werden. Sie kann nun die Streuung der Elektronen an Wasserstoffmolekülen, die im Wesentlichen das Restgas im Spektrometer darstellen, simulieren. Dabei werden elastische Streuungen, Anregungen und Ionisationproszesse berücksichtigt. Die Algorithmen für die Berechnung der Wirkungsquerschnitte dabei stammen von Ferenc Glück. Sie wurden bereits im Programm ADIPARK von Thomas Thümmler eingesetzt. Eine Beschreibung des Programms befindet sich in [Thu02].

Mit diesen Modifikationen können jetzt globale Bahnverfolgungen durchgeführt werden. Allerdings ist die Rechnung immer noch sehr zeitaufwändig. Da man aber für aussagekräftige Simulationen eine große Zahl an Berechnungen durchführen muss, wird auch eine entsprechende Computerkapazität benötigt. Dazu wurde in Münster der bereits existierende und für das ALICE-Experiment eingesetzte Rechencluster (Abbildung 4.1) um 9 Nodes mit jeweils zwei Dual-Opteron-Prozessoren und 4 GB Arbeitsspeicher erweitert. Für die anfallenden Daten wurde eigens ein 5 TB RAID-Server angeschafft. So existiert nun in Münster die technische Grundlage für globale Bahnverfolgungssimulationen.

Abbildung 4.1: Rechencluster in Münster

5 Hilfsmittel zum Geometriedesign

5.1 Aufgabenstellung

In der ersten Entwicklungsphase des elektromagnetischen Designs für die Drahtelektrode waren nur Simulationen in erster Näherung notwendig. Es galt die optimalen Potentiale und Anordnungen der Drahtlagen zu bestimmen. Dafür wurde die Geometrie des Hauptspektrometers durch rotationssymmetrische Strukturen angenähert. Für Detailrechnungen ist dies allerdings unzureichend. Sie sind notwendig, um Untergrundelektronen nahe der Tankwand oder den Kämmen zu simulieren oder um Penningfallen im Aufbau des Experiments zu finden. Für diese Art von Rechnung ist eine ganz neue dreidimensionale Elektrodengeometrie des Hauptspektrometers notwendig. In dieser Geometrie müssen die einzelnen Zähne der Kämme, sowie eventuelle Strukturen zur Korrektur der elektrischen Potentiale nachgebildet werden. Das Erstellen dieser Geometrie ist eine der grundlegenden Schwierigkeiten des elektromagnetischen Designs. Selbst in den Fällen, in denen es bereits CAD-Zeichnungen von Bauteilen gibt, kann man diese nicht direkt für elektrostatische Simulationen verwenden, denn für eine Elektrodengeometrie braucht man Informationen wie Potentialbelegung und Oberflächendiskretisierung, die man nicht aus technischen Zeichnungen gewinnen kann. Diese enthalten mechanische Materialeigenschaften, die für das elektrische Design irrelevant sind. Man muss also eine ganz neue Geometrie entwerfen, die sich mit dem CAD-Modell deckt und dieses noch erweitert. Für diesen Vorgang werden entsprechende Hilfsmittel benötigt.

Eine Elektrodengeometrie besteht aus einer großen Textdatei, in der zeilenweise die Elektroden definiert sind. Ein Beispiel findet sich in Abbildung 5.1. Solch eine Datei lässt sich für einfache Geometrien von Hand erstellen, doch bei komplizierten Geometrien wird dies unpraktikabel. Man müsste die Position und Lage jeder einzelnen Elektrode bestimmen. Wenn man danach grundlegende Parameter, wie den Abstand zwischen zwei Modulen der Drahtelektrode, variieren möchte, muss man jede Elektrode neu platzieren. Detailgetreue Simulationen sind auf diese Weise praktisch unmöglich. Daher wird ein Werkzeug benötigt, das verschiedene Parameter einliest und basierend darauf eine Geometrie generiert. Außerdem benötigt man eine Visualisierung der Elektroden zu Kontrollzwecken. Dies ist besonders wichtig, weil die Simulationsrechnungen sehr zeitaufwändig sind und eine vorherige Kontrolle sehr viel Zeit sparen kann.

```
-.13300000E+02 .10000000E+00 -.11767000E+02 .10000000E+00 .00000000E+00 15
-.11767000E+02 .10000000E+00 -.11555000E+02 .12430000E+00 .00000000E+00 3
-.11555000E+02 .12430000E+00 -.11439000E+02 .16200000E+00 .00000000E+00 3
-.11439000E+02 .16200000E+00 -.11396524E+02 .18536163E+00 .00000000E+00 3
-.11379000E+02 .19500000E+00 -.11378159E+02 .18739271E+00 .00000000E+00 1
-.11378159E+02 .18739271E+00 -.11382943E+02 .18141865E+00 .00000000E+00 1
-.11382943E+02 .18141865E+00 -.11390550E+02 .18057735E+00 .00000000E+00 1
-.11390550E+02 .18057735E+00 -.11396524E+02 .18536163E+00 .00000000E+00 1
-.11396524E+02 .18536163E+00 -.11397366E+02 .19296891E+00 .00000000E+00 1
-.11397366E+02 .19296891E+00 -.11392581E+02 .19894297E+00 .00000000E+00 1
-.11392581E+02 .19894297E+00 -.11384974E+02 .19978427E+00 .00000000E+00 1
-.11384974E+02 .19978427E+00 -.11379000E+02 .19500000E+00 .00000000E+00 1
-.11157089E+02 .49006613E+00 -.11155103E+02 .48679508E+00 -.18450000E+05 1
-.11155103E+02 .48679508E+00 -.11156012E+02 .48307766E+00 -.18450000E+05 1
-.11156012E+02 .48307766E+00 -.11159283E+02 .48109147E+00 -.18450000E+05 1
-.11159283E+02 .48109147E+00 -.11163000E+02 .48200000E+00 -.18450000E+05 1
-.11163000E+02 .48200000E+00 -.11164986E+02 .48527104E+00 -.18450000E+05 1
```

Abbildung 5.1: Auszug aus einer Geometriedatei

5.2 Der MainSpec-Designer

Das Generieren der Geometrie übernimmt ein selbst entwickeltes Programm namens *MainSpec*. Es hat die Aufgabe, ausgehend von einem Zahnmodell, der Tankgeometrie und vielen weiteren Parametern, eine vollständige Geometrie zu erstellen. Dazu gehören die Kämme, sowohl im Zylinder- als auch in den Konusteilen, Drähte, die zwischen diesen Kämmen angeordnet sind, und eine diskretisierte Tankgeometrie. So soll der Benutzer mit einem Schritt eine fertige Geometriedatei erhalten, die dann ohne weiteres von *Elcd3_3* verarbeitet werden kann. Mehr Informationen zur Spektrometergeometrie befinden sich in Kapitel 3.2.

Das Programm ist vollständig in Python [Pyt] geschrieben. Python ist eine objektorientierte Skriptsprache. Sie gewann in den letzten Jahren an Popularität und hat ältere Sprachen wie TCL abgelöst. Besonders in der Open-Source-Welt ist sie mittlerweile sehr weit verbreitet. Viele Linux-Programme sind heute schon in Python geschrieben. Die Sprache wird auch bereits von dem am CERN entwickelten ROOT-Framework unterstützt. Quelltext, der in Python geschrieben ist, ist in der Regel wesentlich kürzer als vergleichbare C++-Programme. Nach einer Änderung im Programm ist kein zeitaufwändiges Neukompilieren notwendig, weil Python im Gegensatz zu C++ eine interpretierte Sprache ist. Allerdings ist dadurch die Ausführungsgeschwindigkeit auch wesentlich geringer. Für komplexe Berechnungen ist Python daher ungeeignet. Da aber die Generierung von Elektrodengeometrien keine großen Rechnungen erfordert und der Zeitfaktor eine untergeordnete Rolle spielt, war Python die optimale Programmiersprache für *MainSpec*.

5.2.1 Programmoberfläche

MainSpec sollte dem Benutzer die Möglichkeit bieten die Geometrie durch viele Parameter zu variieren, um beispielsweise Toleranzrechnungen zu ermöglichen. Daher besitzt *MainSpec* eine grafische Oberfläche, über die der Benutzer bequem sämtliche Parameter einstellen kann. Abbildung 5.2.1 zeigt ein Bildschirmfoto des Programms. Für die Oberfläche werden die GTK-Bibliotheken [GTK] über deren PyGTK-Bindings [PyG] benutzt. Die GTK-Bibliotheken wurden ursprünglich für das bekannte Grafik-Programm Gimp entwickelt, sind jedoch mittlerweile zu einer Standardbibliothek unter Linux geworden. So setzt der bekannte Linux-Desktop GNOME auf GTK auf, ebenso wie viele bekannte Programme unter Linux. Als Beispiel seien Firefox und der RealPlayer genannt. Mittlerweile gibt es auch unter Windows Programme, die GTK benutzen.

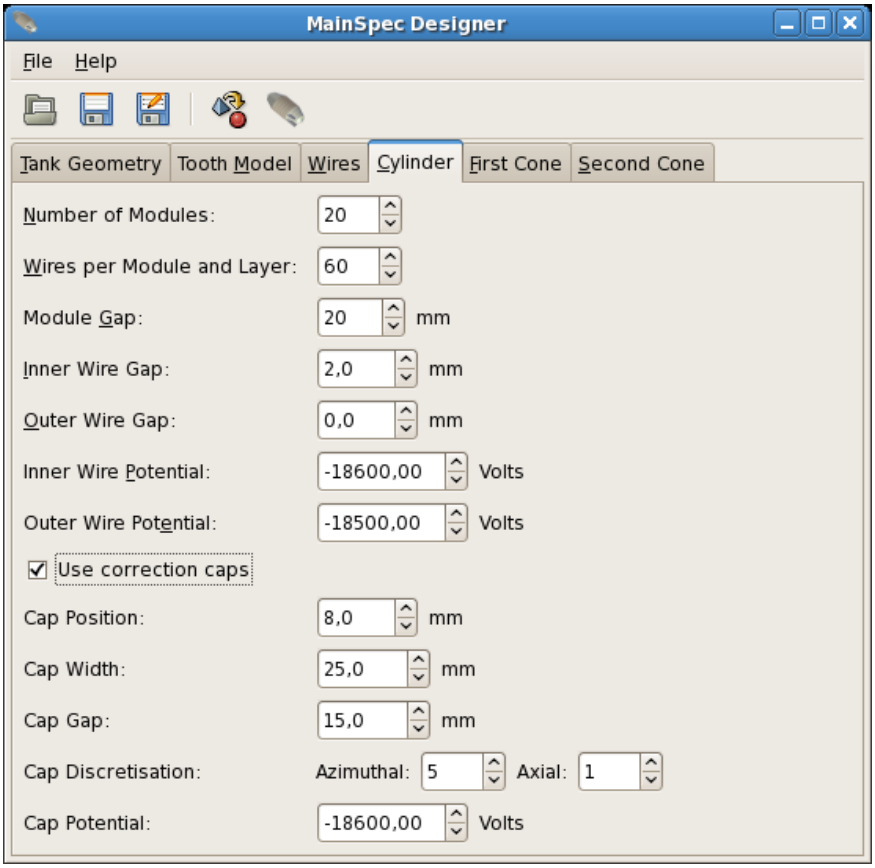

Abbildung 5.2: Bildschirmfoto von *MainSpec*

Die eingegeben Parameter lassen sich in einer XML-Datei sichern. So können Geometrien variiert werden, ohne dass alle Parameter erneut eingeben werden müssen.

5.2.2 Programmstruktur

Die Entwicklung des elektromagnetischen Designs verläuft meist nicht geradlinig. Oft werden neue Konzepte eingeführt, abgeändert und dann wieder verworfen. Da *MainSpec* als Hilfsmittel für einen solchen Entwicklungsprozess vorgesehen ist, muss dessen Struktur die notwendige Flexibilität aufweisen. Daher wurde es von Anfang an objektorientiert und modular entwickelt. Nur so können ohne großen Aufwand Programmteile hinzugefügt oder ersetzt werden. Dies wäre mit einem monolithischen Programm nur sehr schwer zu realisieren. Python erweist sich hier als ideale Programmiersprache. Sie bietet alle Mechanismen der objektorientierten Programmierung, und darüber hinaus lassen sich Python-Programme sehr einfach modularisieren.

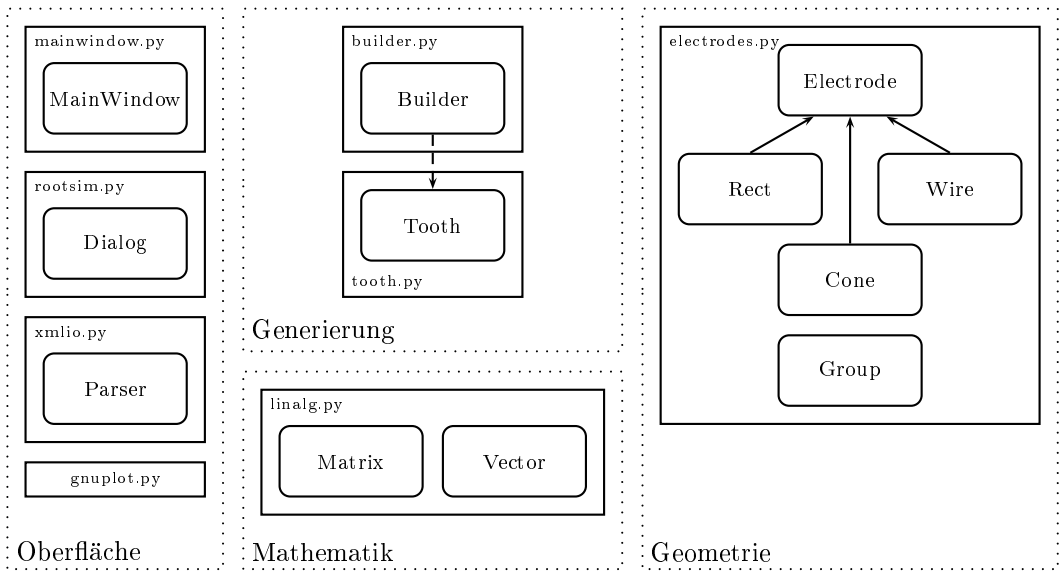

Abbildung 5.3: Module und Klassen von *MainSpec*

MainSpec zerfällt in vier nahezu unabhängige Module, wie in Abbildung 5.3 gezeigt. Dazu gehört zum einen die Oberfläche, die die gesamte Benutzerschnittstelle bereitstellt. Zur Oberflächen gehören auch Funktionen zur Darstellung der Geometrie mit Gnuplot und ROOTsim (siehe dazu Kapitel 5.2.3) und Speichern und Laden der Parameter. Getrennt davon ist die Generierung der Geometrie. Sie bekommt von der Oberfläche die Parameter übergeben, die der Benutzer eingegeben hat und erstellt daraus eine Geometrie. Die Klassen für die die mathematischen Operationen werden von einem Mathematik-Modul bereitgestellt. Dieses enthält für diesen Zweck eine Matrix- und eine Vektorklasse. Die konkrete Behandlung der Elektroden übernimmt das Geometrie-Modul. Es enthält mehrere hierarchisch angeordnete Klassen, deren gemeinsame Basis die abstrakte *Electrode*-Klasse ist. Diese bietet elementare Methoden zur Manipulation von Elektroden, wie Rotieren, Verschieben und

Spiegeln. Diese Funktionen werden von den Klassen *Cone*, *Rect* und *Wire* konkret für Konen, Rechtecke und Drähte implementiert. Zusätzlich gibt es noch die Klasse *Group*, welche Elektroden gruppiert und Operationen auf alle Unterelektroden gemeinsam ausführt. Sie kann auch eine Gruppe von Elektroden in eine Geometriedatei schreiben und wieder einlesen.

Da sowohl das Mathematik- als auch das Geometriemodul unabhängig vom Rest des Programms sind, bieten die beiden Module zusammen eine sehr mächtige Basis für weitere Werkzeuge zur Geometrieerstellung. Allein in Kombination mit dem Python-Interpreter kann man mit ihnen in kurzer Zeit recht komplizierte Geometrien erstellen. Die Module umfassen Routinen zum Lesen und Schreiben für die Geometrieformate von *Elcd3_2* und *Elcd3_3*. Drähte können daher mit nur wenigen Zeilen Python-Code von einem Format in das andere konvertiert werden. Da auch die Funktionen zur Elektrodendiskretisierung in den Modulen enthalten sind, kann man auch die Konen, die es nur bei *Elcd3_2* gibt, in Rechtecke umwandeln. Somit kann man die Geometriekonvertierungen, für die bisher jeweils spezielle C-Programme existierten, nun einfach und elegant in Python durchführen. Auf diese Weise kann man beispielsweise verschiedene Konvertierungen durch Skripte automatisieren.

5.2.3 Elektrodengenerierung

Wenn der Benutzer eine Geometrie erstellt, werden die Parameter von der Oberfläche an die *Builder*-Klasse übergeben. Im ersten Schritt werden dann die Geometrien für den Spektrometertank eingelesen. Daraus wird die genaue Form des Hauptspektrometers bestimmt. Dabei werden die Länge der einzelnen Spektrometersektionen und die genauen Konuswinkel berechnet. Diese sind notwendig, weil die Drähte der Drahtelektrode genau parallel in einem definierten Abstand zur Tankwand verlaufen sollen. Dieser Abstand ist bereits als 22 cm für die innere und 15 cm für die äußere Drahtlage fest definiert. Dennoch lässt er sich als Parameter vom Benutzer verändern. So kann der Benutzer ihn variieren und seinen Einfluss in Toleranzrechnungen untersuchen. Ähnliches gilt für viele weitere Benutzerparameter. Sie sind bereits auf KATRIN-Vorgabe eingestellt und sollten nur verändert werden um Toleranzen abzuschätzen.

Aus der Tankgeometrie werden die Positionen der Kämme und Drähte berechnet. Eine Hilfsklasse *Tooth* erstellt nun an diesen Positionen aus den Kammparametern Gruppen von Elektroden, die die Kämme darstellen. Sie liefert auch die korrekten Start- und Endpunkte für die Drähte, die nun erzeugt werden. Auf diese Weise wird die Drahtelektrode für die Zylindersektion und den ersten Konus erstellt. Im Zylinder werden bei Bedarf noch Korrekturelektroden an die Kämme angefügt. Diese werden für die Homogenität des Potentials benötigt, dazu sei auf [Val04] und [Hug08] verwiesen. Für den zweiten Konus und den Vollkonus wurde das Design abgewandelt. Statt eines Kammes gibt es hier zwei Reifen, die eine Drahtlage tragen, die zweite Drahtlage entfällt. Dies wird auch von MainSpec berücksichtigt. Dafür wird die Hilfsklasse *Ring* verwendet. Mit ihr werden zuerst die Reifen an den berechneten Positionen erstellt, danach wird die Drahtlage daran angepasst.

Wenn alle Teile der Geometrie erstellt sind, werden sie in einer großen Gruppe zusammengefügt. Die Tankgeometrie, die zu Beginn eingelesen wurde, wird in Rechtecke diskretisiert und ebenfalls hinzugefügt. Die langen Drähte in der Geometrie werden, falls der Benutzer das wünscht, vorher noch in kleine Drahtelemente diskretisiert. Die resultierende Elektrodengruppe wird in eine Geometriedatei im *Elcd3_3*-Format geschrieben. Diese Datei ist jetzt bereit für den Gebrauch durch den Benutzer.

5.3 Visualisierung

Will der Benutzer nun die erstellte Geometrie überprüfen, benötigt man ein Programm um diese sichtbar zu machen. Dabei reicht eine einfache zweidimensionale Darstellung nicht aus. Viele Details könnte man damit nicht erkennen. *ROOTsim* geht hier einen Schritt weiter. Es kann jedes Geometrieformat, das von den EMD-Programmen verwendet wird, einlesen und dreidimensional darstellen. So lässt sich auch die komplette KATRIN-Geometrie darstellen und detailliert betrachten. Der Benutzer kann sie im Ganzen analysieren, aber auch in kleine Details hereinzoomen und die ganze Struktur rotieren. Abbildung 5.4 zeigt das Programm. *ROOTsim* basiert auf einer Arbeit von Matthias Prall und Kathrin Valerius. Es ist in C++ geschrieben und verwendet den von ROOT bereitgestellten OpenGL-Viewer, der bereits alle 3D-Funktionen enthält. Das Einlesen der Geometrien geschieht über eine Klassenhierarchie, die die Elektroden und Magnete auf ROOTs *Geometry*-Klassen abbildet.

ROOTsim ist ein wichtiges und zuverlässiges Werkzeug zur Kontrolle von Geometrien geworden. Es ist auch bereits in *MainSpec* integriert. So lässt sich ohne Zwischenspeichern eine Geometrie in *ROOTsim* betrachten. Zusätzlich kann man alle vorgegebenen Komponenten wie Tank, Flansch oder Zähne, kontrollieren.

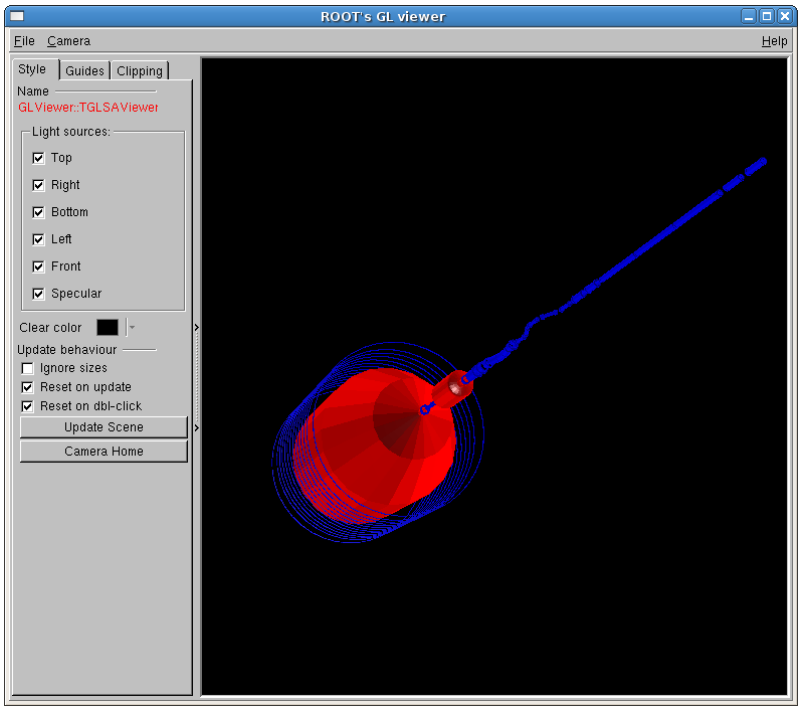

Abbildung 5.4: Die KATRIN-Geometrie mit Elektroden (rot) und Magnetspulen (blau) in *ROOTsim* und ROOT Version 5.17.02

6 Eine verbesserte Boundary Element Methode

In diesem Kapitel wird eine selbstentwickelte Alternative zu den bisherigen Programmen zur elektrischen Feldberechnung vorgestellt. Insbesondere soll es mit dem neuen Programm möglich sein, wesentlich mehr Oberflächenelemente als bisher zu verarbeiten. Dies würde wesentlich genauere Simulationen erlauben, sowohl bei der Bahnverfolgung als auch bei elektrostatischen Detailrechnungen. Zuerst wird auf die mathematischen Grundlagen eingegangen, die dem Algorithmus zugrunde liegen. Es folgt eine Beschreibung der einzelnen Aspekte des Programms.

6.1 Zielsetzung

Die momentan für das elektromagnetische Design eingesetzten Programme wurden speziell für diese Aufgabe entwickelt. Daher sind sie in Präzision und Flexibilität kommerziellen Produkten für den Einsatz beim KATRIN-Experiment deutlich überlegen. Jedoch bringt das verwendete Verfahren der Boundary Element Methode (BEM) eine große Einschränkung mit sich, denn der Speicherbedarf ist proportional zum Quadrat der Anzahl der Oberflächenelemente und die Rechenzeit sogar zu ihrer dritten Potenz. Das macht das Verfahren bei einer großen Elementanzahl sehr zeitaufwändig und begrenzt die Anzahl der unabhängigen Elemente auf ungefähr 10000 auf gewöhnlichen Arbeitsplatzrechnern mit etwa 1 GB Arbeitsspeicher, falls man keine Spiegel- oder Rotationssymmetrie der Geometrie ausnutzen kann.

Die Zielsetzung für das neue Programm ist nun die Umgehung dieser Einschränkung. Zum einen muss ein Weg gefunden werden, die Anzahl der Elemente zu erhöhen, denn für vollständige Simulationen des KATRIN-Aufbaus mit dem erforderlichen Grad an Detailgenauigkeit sind wesentlich mehr als 10000 unabhängige Elemente nötig. Daher sind mit den existierenden Programmen diese Rechnungen nicht möglich. Das andere wichtige Ziel bei der Entwicklung ist eine drastische Reduzierung der benötigten Rechenzeit. Diese erweist sich bereits jetzt als großes Hindernis. Denn nun können Hauptspektrometer-Geometrien mit diskretisierten Kämmen und Drähten erstellt werden, wodurch die Anzahl der Element deutlich gestiegen ist und somit auch die benötigte Rechenzeit. Da aber verschiedene Geometriekonzepte getestet und optimiert werden müssen, bremst dies die Entwicklung des elektromagnetischen Designs empfindlich.

Gleichzeitig soll aber die Kompatibilität zu den alten Programmen erhalten bleiben. Dabei muss es möglich sein, die alten Elektrodengeometrien mit dem neuen Programm zu berechnen. Nur so lassen sich realistische Vergleichsrechnungen

durchführen. Außerdem muss sich das neue Programm in die existierende Umgebung der EMD-Programme einfügen. Idealerweise könnte man es mit minimalen Anpassungen direkt in der bestehenden Bahnverfolgung einsetzen. Dies lässt sich am einfachsten durch eine Programmierung in C realisieren, denn in dieser Sprache sind bereits alle anderen Programme geschrieben und sie bringt der Vorteil der sehr guten Portabilität mit sich. Auch wenn in Münster praktisch nur Linux-Rechner verwendet werden, muss das Programm auch auf Windows- und Mac-Systemen lauffähig sein, denn diese kommen ebenfalls in der KATRIN-Kollaboration zum Einsatz.

Ein weiterer wichtiger Punkt bei der Planung betrifft die Struktur des neuen Programms. Die momentan existierenden Programme basieren noch auf alten Fortran-Programmen und nutzen kaum moderne Konzepte der C-Programmiersprache. Sie wurden mit der Zeit stückweise weiterentwickelt, was sich negativ auf die Übersichtlichkeit ausgewirkt hat. Durch diese Entwicklung ist der Code auch sehr unflexibel. Aus diesem Grund sollte ein neues Programm von Grund auf neu entwickelt werden und nicht eine Erweiterung von existierenden Programmen darstellen.

6.2 Mathematische Grundlagen

6.2.1 Boundary Element Method

Die Boundary Element Method (BEM) ist ein numerisches Verfahren zur Berechnung von elektrischen Potentialen. Sie benutzt die Tatsache, dass bei der Feldberechnung die Elektrode durch eine geladene Oberfläche ersetzt werden kann. Bei der BEM wird nun diese Oberflächenladung approximiert, indem man die gesamte Leiteroberfläche in kleine Elemente zerlegt und auf diesen jeweils eine konstante Ladungsdichte annimmt. Da die Ladungsdichte eine stetige Funktion ist, ist dieser Ansatz erlaubt und kann die reale Ladungsdichte beliebig genau approximieren.

Wendet man nun dieses Verfahren auf eine Elektrodenkonfiguration an, kann man aus den vorgegeben elektrischen Potentialen die Elementladungsdichten bestimmen. Dazu diskretisiert man die Oberfläche jeder Elektrode in kleine Elemente. Diese Oberflächenelemente liegen auf dem gleichen Potential wie die gesamte Elektrode, denn Leiteroberflächen sind Äquipotentialflächen. Da jetzt die geometrische Anordnung der Elemente untereinander sowie die elektrischen Potentiale an deren Mittelpunkten bekannt sind, kann man mit dieser Information ein lineares Gleichungssystem konstruieren. Dazu bestimmt man paarweise die Beiträge der Wechselwirkungen zwischen den Elementen. So ist ϕ_{ij} der Potentialbeitrag, der durch das Oberflächenelement j im Zentrum des Elements i erzeugt wird

$$\phi_{ij} = \sigma_j \frac{1}{4\pi\varepsilon_0} \int d^2 r_j \frac{1}{|\vec{r}_i - \vec{r}_j|}. \tag{6.1}$$

Integriert wird über das gesamte Element j. σ_j ist die Ladungsdichte auf diesem Element, da sie als konstant angenommen ist, kann man sie aus dem Integral herausziehen und das Integral hängt nur noch von der Geometrie der beiden Ober-

flächenelemente ab. Man kann den Ausdruck also vereinfachen zu

$$\phi_{ij} = \sigma_j k_{ij}. \tag{6.2}$$

Addieren wir nun die Potentialbeiträge von allen Elementen auf, so erhalten wir das Potential U_i im Zentrum des Elementes i:

$$U_i = \sum_{j=1}^{N} \phi_{ij} = \sum_{j=1}^{N} \sigma_j k_{ij}. \tag{6.3}$$

Dies entspricht einem linearen Gleichungssystem

$$K\vec{\sigma} = \vec{U}. \tag{6.4}$$

Die Matrix K erhält man aus der Geometrie, den Vektor $\vec{U}$ aus den Potentialen. Durch Lösen dieses Systems erhält man den Vektor $\vec{\sigma}$ mit den Ladungsdichten auf den Oberflächenelementen. Die Berechnung beinhaltet allerdings die Näherung, dass die Potentialbeiträge nur im Zentrum eines Elements aufaddiert werden. Dies ist aber kein großer Fehler, solange die Elemente klein genug sind.

Man ist aber normalerweise nicht an den Ladungsdichten interessiert, sondern an elektrischen Potentialen und Feldern, welche sich daraus berechnen lassen. Dazu summiert man über alle Oberflächenladungsdichten

$$U(\vec{r}) = \frac{1}{4\pi\varepsilon_0} \sum_{i=1}^{N} \sigma_i \int d^2 r_i \frac{1}{|\vec{r} - \vec{r}_i|}. \tag{6.5}$$

Das elektrische Feld für jeden beliebigen Punkt kann man nun durch numerisches Differenzieren des Potentials berechnen.

Die BEM eignet sich besonders gut für Probleme, bei denen ein großes Volumen betrachtet werden muss, das aber viele kleine Strukturen enthält. Dabei ist sie der Methode der finiten Differenzen (FDM) und der Methode der finiten Elemente (FEM), bei denen das gesamte Volumen diskretisiert wird, weit überlegen. Sowohl mit FDM und FEM kann man bei großen Volumina kleine Strukturen nur noch mit komplexen adaptiven Gittermethoden auflösen. Auf der anderen Seite stößt die BEM auch an ihre Grenzen, wenn man extrem detaillierte Oberflächen betrachtet und die Anzahl der Oberflächenelemente groß wird.

Das Verfahren der BEM wurde bereits 1963 entwickelt [Cru63]. Sie wurde bereits sehr erfolgreich bei der Berechnung elektrostatischer Linsen eingesetzt. In heutiger Zeit wird sie auch zur Untersuchung von Proteinmolekülen verwendet. Dabei betrachtet man das Molekül als geschlossenes Volumen in einem dielektrischen Medium. Dazu muss man die Methode etwas erweitern, weil nun geladene Atome im Volumen sind. Aber die BEM ist auch hier sehr erfolgreich und man versucht durch elektrostatische Berechnungen die äußerst komplexe Proteinfaltung zu verstehen.

Für KATRIN ist die BEM die richtige Methode, weil man hier ein sehr großes Volumen wie den Spektrometertank und sehr kleine Strukturen wie die Drahtelektrode zu implementieren hat. Der Tank hat eine Länge von 24 m, ein einzelner Zahn

eine Länge von 7 cm und die Drähte haben einen Durchmesser von nur 0,2 mm. Das Anwenden der Methode der finiten Differenzen auf so ein großes Volumen mit der benötigten Auflösung ist praktisch unmöglich, denn die benötigte Anzahl an Speicherzellen wäre in der Größenordnung 10^{15}.

6.2.2 Multipol-Entwicklung

Die Berechnung des von einer beliebigen Raumladungsdichte $\varrho(\vec{r})$ erzeugten Potentials $\varphi(\vec{r})$ am Punkt $\vec{r}$ erfolgt durch Berechnung des Coulomb-Integrals

$$\varphi(\vec{r}) = \frac{1}{4\pi\varepsilon_0} \int d^3r' \frac{\varrho(\vec{r}\,')}{|\vec{r} - \vec{r}\,'|}. \tag{6.6}$$

Bei komplizierten Ladungsverteilungen kann die Berechnung sehr schwierig werden, da dann das Integral nur noch numerisch gelöst werden kann. Oft benötigt man gar nicht das exakte Potential, besonders für den Fall $r \gg r'$. In diesem Fall kann man die Multipol-Entwicklung der Ladungsverteilung betrachten. Dazu entwickelt man den Ausdruck $|\vec{r} - \vec{r}\,'|^{-1}$ in eine Taylorreihe

$$\frac{1}{|\vec{r} - \vec{r}\,'|} = \sum_{n=0}^{\infty} \frac{(-\vec{r}\,' \cdot \vec{\nabla})^n}{n!} \frac{1}{r}. \tag{6.7}$$

Dabei ist $r := |\vec{r}|$ und im folgenden auch analog $r' := |\vec{r}\,'|$.

Setzen wir nun (6.7) in (6.6) ein, erhalten wir die Multipol-Entwicklung

$$\varphi(\vec{r}) = \frac{1}{4\pi\varepsilon_0} \sum_{n=0}^{\infty} \varphi_n(\vec{r}) = \frac{1}{4\pi\varepsilon_0} \int d^3r' \sum_{n=0}^{\infty} \frac{(-\vec{r}\,' \cdot \vec{\nabla})^n}{n!} \frac{\varrho(\vec{r}\,')}{r}. \tag{6.8}$$

Betrachten wir nun die ersten Terme dieser Entwicklung:

Monopol-Term

$$\varphi_0(\vec{r}) = \int d^3r' \frac{\varrho(\vec{r}\,')}{r} = \frac{q}{r} \tag{6.9}$$

Er enthält das Monopol-Moment q der Ladungsverteilung

$$q = \int d^3r' \varrho(\vec{r}\,'). \tag{6.10}$$

Es entspricht der Gesamtladung auf dem Oberflächenelement.

Dipol-Term

$$\varphi_1(\vec{r}) = \int d^3r' \varrho(\vec{r}\,')(-\vec{r}\,') \cdot \nabla \frac{1}{r} = \int d^3r' \varrho(\vec{r}\,') \frac{\vec{r}\,' \cdot \vec{r}}{r^3} = \frac{\vec{d} \cdot \vec{r}}{r^3} \tag{6.11}$$

Dabei ist $\vec{d}$ das Dipol-Moment der Ladungsverteilung

$$\vec{d} = \int d^3r' \varrho(\vec{r}\,') \vec{r}\,'. \tag{6.12}$$

Quadrupol-Term

$$\varphi_2(\vec{r}) = \int d^3r' \varrho(\vec{r}\,') \frac{(-\vec{r}\,' \cdot \nabla)^2}{2} \frac{1}{r} = \int d^3r' \varrho(\vec{r}\,') \frac{3(\vec{r}\,' \cdot \vec{r})^2 - r'^2 r^2}{2r^5} = \frac{\vec{r} \cdot \hat{Q} \cdot \vec{r}}{2r^5} \quad (6.13)$$

$\hat{Q}$ ist ein Tensor 2. Stufe und wird als Quadrupoltensor bezeichnet. Er ist symmetrisch und spurfrei, d. h. $Q_{11} + Q_{22} + Q_{33} = 0$. Daher sind von den 9 Komponenten nur 5 unabhängig.

$$Q_{ij} = \int d^3r' \varrho(\vec{r}\,')(3r'_i r'_j - r'^2 \delta_{ij}) \quad (6.14)$$

Insgesamt erhält man für die ersten drei Ordnungen:

$$\varphi(\vec{r}) = \frac{1}{4\pi\varepsilon_0} \left(\frac{q}{r} + \frac{\vec{d} \cdot \vec{r}}{r^3} + \frac{\vec{r} \cdot \hat{Q} \cdot \vec{r}}{r^5} \right) \quad (6.15)$$

Diese Näherung ist bereits für die meisten Anwendungen ausreichend.

Im Allgemeinen sind die Multipolmomente einer Ladungsverteilung abhängig vom Ursprung des Koordinatensystems. Eine Ausnahme bildet aber die erste nicht-verschwindene Ordnung, sie ist stets ursprungsunabhängig. Für eine ausführliche Diskussion dieses Zusammenhangs sei auf [Jac99] verwiesen.

6.3 Erweiterung der Boundary Element Methode

Die BEM ist die optimale Methode zur Berechnung der elektrischen Felder bei KATRIN. Dennoch ist sie aufgrund der oben genannten Einschränkung der Elementanzahl nicht ausreichend für fortgeschrittene Berechnungen. Sie muss in einer Form erweitert werden, die ihre Vorteile beibehält, aber gleichzeitig eine höhere Anzahl an Oberflächenelementen erlaubt. Im Rahmen dieser Arbeit wurde ein Ansatz entwickelt, der die Oberflächenelemente gruppiert und dann mit Multipolen approximiert. Diese Art der Erweiterung wurde erstmals in [GR87] beschrieben.

Der zentrale Gedanke bei diesem Konzept ist der, dass man die elektrostatische Wechselwirkung von zwei Ladungsdichten, die sehr weit von einander entfernt sind, nicht exakt kennen muss. Ihr Beitrag ist klein im Vergleich zur Wechselwirkung zwischen zwei Ladungsdichten mit wesentlich geringerem Abstand zueinander. Daher genügt es, wenn sie nur über eine Näherung in die Berechnung miteinfließt.

Für diese Näherung braucht man zuerst eine Definition der Begriffe „nah“ und „fern“. Dies geschieht mit einem dreidimensionalen Raumgitter. Man teilt das gesamte Volumen, welches die Elektrodengeometrie einnimmt, in kleine Untervolumina auf. Deren Form ist in erster Linie irrelevant. Es muss nur möglich sein, ein Element eindeutig einem der Volumina zuzuordnen. Dieses Raumgitter wird auch Boxsystem genannt und die Untervolumina sind die Boxen. Man kann nun dieses System benutzen, um die Elemente in nah und fern zu unterteilen. Dazu betrachtet man für jedes Paar von Elementen einzeln die Entfernung zwischen den Mittelpunkten ihrer jeweiligen Boxen. Ist dieser größer als ein vorgegebener Schwellwert, werden diese

Elemente als fern zueinander definiert. Anderfalls gelten sie als nah. Diese Unterteilung fließt in die Berechnung des linearen Gleichungssystems (6.4) ein. Hier wird die Matrix K aufgeteilt:

$$K = K_n + K_f \tag{6.16}$$

K_n enthält die Coulomb-Wechselwirkung zwischen nahen Elektrodenpaaren und K_f die zwischen fernen Elektronen. Setzt man nun (6.16) in das Gleichungssystem (6.4) ein, erhält man

$$K_n\vec{\sigma} = \vec{U} - K_f\vec{\sigma}. \tag{6.17}$$

Dieses neue Gleichungssystem hat den Vorteil, dass die Matrix nun sehr viele Nullen enthält. Jede Wechselwirkung ist 0, die nicht nah ist. Wenn man die Matrix intelligent speichert, belegen die Beträge dieser Wechselwirkungen auch keinen Arbeitsspeicher mehr. Dadurch ist der Speicherverbrauch nicht mehr proportional zum Quadrat der Elementanzahl, sondern proportional zu dem Produkt von Elementanzahl und der Anzahl der nahen Elemente.

Bei dieser Aufteilung entsteht jedoch das Problem, dass die rechte Seite des Gleichungssystems nun von den Ladungsdichten $\vec{\sigma}$ abhängt. Daher lässt es sich nicht mehr exakt lösen. Es muss ein neues numerisches Verfahren zu dessen Lösung gefunden werden. Statt der bisher verwendeten Gauß-Elimination, muss nun eine iterative Methode eingesetzt werden. Dies verbessert auch die Rechengeschwindigkeit deutlich, denn für große Gleichungssysteme sind iterative Ansätze in den meisten Fällen besser geeignet. Zwar bringen sie immer einen zusätzlichen Fehler in die Berechnung herein, aber auch ein nicht-iteratives Verfahren ist nur „exakt“ im Rahmen der numerischen Genauigkeit. Jede Fließkommaoperation am Computer enthält einen kleinen Rundungsfehler, daher gibt es keine wirklich exakte Methode ein Gleichungssystem numerisch zu lösen.

Leider lässt sich aufgrund der komplexeren Abhängigkeit von $\vec{\sigma}$ in diesem Fall keines der bekannten iterativen Verfahren ohne Modifikation anwenden. Zuvor muss $\vec{\sigma}$ aus der rechten Seite von (6.17) eliminiert werden. Dafür setzt man die Ladungsdichten auf geeignete Startwerte. Mit diesen Ladungsdichten wird die rechte Seite approximiert:

$$\vec{X} = \vec{U} - K_f\vec{\sigma}. \tag{6.18}$$

Die Approximation erfolgt über eine Multipolentwicklung. Man nähert das Potential, das alle Elemente einer Box induzieren, durch eine Multipolreihe. Diese erhält man, indem man den Mittelpunkt der Box in den Ursprung verschiebt und anschließend die Multipolmomente φ_j aus (6.15) der enthaltenen Elemente aufaddiert. Für das von der Box A induzierte Potential ϕ_A gilt somit:

$$\phi_A(\vec{r}) = \sum_{j\in A} \varphi_j(\vec{r}). \tag{6.19}$$

Da die Matrix K_f nur die fernen Wechselwirkungen enthält, kann man damit eine Formel zur Berechnung von $\vec{X}$ angeben:

$$X_i = U_i - \sum_{A\in F_i} \phi_A(\vec{r}_i). \tag{6.20}$$

F_i ist die Menge aller Boxen, die zum i-ten Element als „fern“ definiert sind und $\vec{r}_i$ der Mittelpunkt des i-ten Elements.

Mit dieser Näherung erhält man eine neues Gleichungssystem, das es gilt iterativ zu lösen. Es lässt sich schreiben als

$$K_n\vec{\sigma} = \vec{X}. \tag{6.21}$$

Der Vektor $\vec{X}$ wird nun für eine gewisse Anzahl an Iterationsschritten als Konstante betrachtet. Dadurch erhält man einen neuen Satz von Ladungsdichten $\vec{\sigma}$, mit denen danach $\vec{X}$ neu berechnet wird. Dies wiederholt man, bis die Ladungsdichten konvergieren.

6.4 Implementierung des neuen Programmes

Basierend auf den oben genannten Vorgaben wurde ein Programm mit dem Namen *Elcd4* entwickelt. Es ist in C geschrieben und benutzt nur Sprachkomponenten, die Teil des ANSI-Standards sind. So ist es möglich maximale Portabilität zu erzielen. Das Programm besteht aus einer Bibliothek, die den Programmkern mit allen Funktionen zur Feldberechnung enthält, sowie den Hilfsprogrammen *elmain4* und *elcheck4*. Diese greifen auf die Bibliotheksfunktionen zu und erlauben die Berechnung der Ladungsdichten bzw. der elektrischen Potentiale. Mehr Informationen zu den beiden Programmen befinden sich im Anhang B.1. Der Benutzer kann entweder diese Programme verwenden oder er greift direkt auf die Bibliothek zu. Zusätzlich gibt es noch das Konvertierungsskript *convert.py*. Es ist in Python geschrieben und dient zur Konvertierung von alten Geometriedateien. Es kann eine beliebige Anzahl an *Elcd3_2*- und *Elcd3_3*-Dateien in das neue, von *Elcd4* verwendete EDF-Format konvertieren.

Bei der Entwicklung von *Elcd4* wurde Wert auf eine objektorientierte Struktur gelegt, auch wenn die Programmiersprache C an sich nicht objektorientiert ist. Dadurch ist der Quelltext deutlich übersichtlicher und leichter zu warten. Außerdem soll *Elcd4* als Ausgangspunkt für weitere Programmentwicklungen dienen und muss daher leicht erweiterbar sein. Dazu eignet sich eine objektorientierte Struktur am besten. Ein weiterer Vorteil dieses Designs ist, dass sich so die *Elcd4*-Bibliothek leicht in eine C++-Klasse einbinden lässt. Diese kann man dann beispielsweise mit *ROOT* und *GEANT4* verwenden. Diese Einbindung war in der Vergangenheit mit den alten Programmen oft sehr schwierig.

Der Quelltext von *Elcd4* ist in mehrere logische Module gegliedert. Ein Modul besteht aus einer Quelldatei und einer zugehörigen Headerdatei. In der Quelldatei sind die Funktionen implementiert und sie hat die Dateiendung *.c*. In der Headerdatei befinden sich Definitionen der von diesem Modul bereitgestellten Funktionen. Sie werden von anderen Modulen eingebunden, die auf diese zugreifen. Die Headerdateien erkennt man an der Dateiendung *.h*. Sie sind auch im Anhang C zu finden.

Zentrale Funktionen

Im zentralen Modul *elcd4.c* wird der Variablentyp *Elcd* definiert. Ein Objekt dieses Typs enthält alle Informationen zur Ladungsberechung und der geladenen Elektrodengeometrie. Theoretisch könnte man auch mehrere Objekte erzeugen und so verschiedene Geometrien gleichzeitig berechnen. Die Objekte sind als C-Struct implementiert, dessen Felder sind privat und sollten nur durch Bibliotheksfunktionen selbst manipuliert werden. Dazu steht dem Anwender ein Satz von Funktionen zur Verfügung, mit denen er auf die komplette Elcd4-Bibliothek zugreifen kann.

Ein *Elcd*-Objekt wird erzeugt und initialisiert durch die Funktion

```
Elcd* elcd_new(int nr, int nphi, int nz)
```

Die Parameter beeinflussen die Form des bei der Berechnung verwendeten Boxsystems. Ihre genaue Bedeutung wird auf Seite 48 erklärt.

Das so erstellte Objekt enthält eine noch leere Geometrie. Daher müssen nun die Daten zu den Elektroden eingelesen werden. Dies geschieht mit der Funktion

```
void elcd_read_edf(Elcd* elcd, const char* filename).
```

Der Parameter *filename* ist der Name der einzulesenen Geometriedatei. Konnte die Geometrie mit den Daten aus dieser Datei erfolgreich angelegt werden, kann man mit der Funktion

```
void elcd_iteration_start(Elcd* elcd)
```

die iterative Berechnung der Ladungsdichten vorbereiten. Jetzt ist das *Elcd*-Objekt bereit für die eigentliche Ladungsdichtenberechnung. Diese erfolgt nun schrittweise mit der Funktion

```
double elcd_iteration_step(Elcd* elcd).
```

Jeder Aufruf dieser Funktion führt eine Iteration aus. Das heißt, der Approximationsvektor $\vec{X}$ wird neu berechnet und das iterative Lösungsverfahren wird mehrmals auf das Gleichungssystem angewendet. Danach wird die mittlere relative Änderung der Ladungsdichten $\Delta\sigma$ während des ganzen Schrittes zurückgegeben:

$$\Delta\sigma = \sum_{n=1}^{N} \left| \frac{\sigma_n^{\text{neu}} - \sigma_n^{\text{alt}}}{\sigma_n^{\text{alt}}} \right| \tag{6.22}$$

Man kann nun die Funktion solange aufrufen, bis $\Delta\sigma$ das Konvergenzkriterium erfüllt.

Die gesamte Funktionalität des *elcd4.c*-Moduls wird auch von einer C++-Klasse bereitgestellt, die in *elcd4mm.cpp* und *elcd4mm.h* definiert ist. Diese Klasse ruft die C-Funktionen auf und bietet so einen bequemen Zugriff von C++ aus.

Mathematische Funktionen

Im gesamten Programm werden häufig Funktionen aus der linearen Algebra verwendet. Zur Vereinfachung sind dazu im Modul *linalg.c* verschiedene Funktionen

zur Vektor- und Matrixrechnung definiert. Sie betrachten einen Vektor als statisches Array von 3 double-Zahlen, eine Matrix wird repräsentiert durch ein Array aus 9 double-Zahlen. So lässt sich zum Beispiel eine Vektoraddition durchführen:

```
vector_add(a, b, c)
```

Dies entspricht der Gleichung

$$\vec{a} + \vec{b} = \vec{c}.$$

Ähnliche Funktionen existieren auch für andere Operationen. Für Details sei auf den zugehörigen Header *linalg.h* auf Seite 86 verwiesen.

Zu Optimierungszwecken existieren auch Funktionen für Rechnungen mit Quadrupoltensoren. Diese werden nicht als gewöhnliche Matrix betrachtet, sondern als Array von 5 double-Werten. Damit wird die Tatsache ausgenutzt, dass nur 5 der 9 Komponenten wirklich unabhängig sind, weil der Quadrupoltensor symmetrisch und spurfrei ist.

Elektrodengeometrie

Die Behandlung der Elektrodengeometrie erfolgt in den Modulen *edf.c* und *electrode.c*. Dazu gehört das Einlesen der Geometriedateien, Diskretisierung der Geometrie, Berechnung von Multipolmomenten und Einzelpotentialen und einfache geometrische Operationen wie Rotieren, Verschieben und Spiegeln.

Das Einlesen der Geometriedateien geschieht im Modul *edf.c*. Der Name leitet sich ab von Electrode Definition File. Damit wird das Dateiformat bezeichnet, das eigens für *Elcd4* entworfen wurde. Es enthält die Definitionen von sämtlichen Elektroden und sollte die Dateiendung *.edf* besitzen. Die Dateien liegen im einem ASCII-Format vor, in dem zeilenweise die Elektroden definiert sind. Die genaue Form einer Zeile hängt vom Typ der definierten Elektrode ab. Momentan ist das Programm auf drei Elektrodentypen beschränkt:

Rechteck: Ein Rechteck wird über einen Eckpunkt $\vec{p}$, die Kanteneinheitsvektoren $\vec{n}_1$ und $\vec{n}_2$ und die Kantenlängen a und b definiert (Abbildung 6.1). Die Definition hat folgende Form:

$$1 \quad n_{rot} \quad p_x \quad p_y \quad p_z \quad n_{1x} \quad n_{1y} \quad n_{1z} \quad n_{2x} \quad n_{2y} \quad n_{2z} \quad a \quad b \quad U$$

Die Zahl 1 gibt den Typ der Elektrodenform an, in diesem Fall „Rechteck“. Der Parameter n_{rot} gibt die Rotationssymmetrie der Elektrode um die z-Achse an. Es werden insgesamt n_{rot} Elektroden erstellt, die jeweils um einen Winkel von $2\pi/n_{rot}$ rotiert sind. U ist das Potential der Elektrode. Falls $\vec{n}_1$ und $\vec{n}_2$ keine orthogonalen Einheitsvektoren sind, wird ein Fehler zurückgegeben.

Draht: Ein Draht wird über seine beiden Endpunkte $\vec{a}$, $\vec{b}$ und seinen Durchmesser d definiert (Abbildung 6.2). Die Definition sieht folgendermaßen aus:

$$2 \quad n_{rot} \quad a_x \quad a_y \quad a_z \quad b_x \quad b_y \quad b_z \quad d \quad U \quad [s \quad p]$$

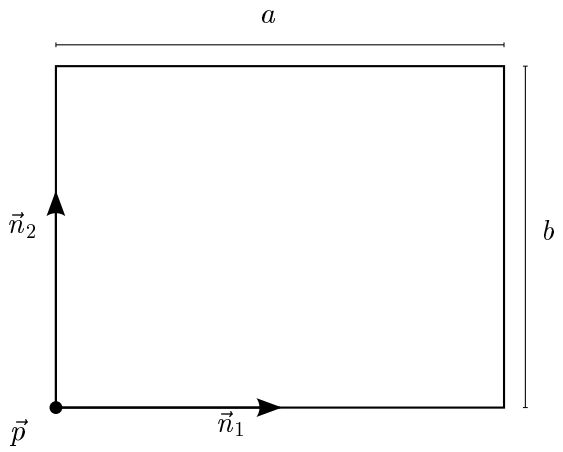

Abbildung 6.1: Parameter einer Rechteck-Elektrode

Zahl 2 gibt wieder den Typ an. Die Parameter n_{rot} und U haben dieselbe Bedeutung wie in der Definition des Rechtecks. Die zusätzlichen Parameter s und p sind optional. Werden sie angegeben, wird der Draht automatisch in kleine Drahtelemente diskretisiert. Eine detaillierte Beschreibung des Diskretisierungsverfahrens folgt auf Seite 45.

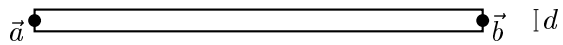

Abbildung 6.2: Parameter einer Draht-Elektrode

Konus: Eine Konuselektrode, die auf der z-Achse zentriert ist, kann durch die beiden z-Koordinaten z_{min}, z_{max} und die dazugehörigen Radien r_{min}, r_{max} definiert werden (Abbildung 6.3). So ergibt sich folgendes Zeilenformat:

$$3 \quad n_{rot} \quad z_{min} \quad r_{min} \quad z_{max} \quad r_{max} \quad U \quad [s \quad p]$$

Die Zahl 3 steht nun für den Konus-Typ. Die Diskretisierung ist beim Konus nicht optional, weil der Algorithmus nur mit Drähten und Rechtecken rechnen kann. Man definiert zwar eine konusförmige Elektrode, aber diese muss vor Beginn der Rechnung automatisch in einzelne Rechtecke zerlegt werden. Dafür wird nun auch der Parameter n_{rot} verwendet, welcher beim Konus eine neue Bedeutung erhält. Ein genaue Beschreibung folgt auf Seite 45.

Durch das modulare Design können bei Bedarf noch weitere Elektrodentypen hinzugefügt werden. So wären beispielsweise Dreiecke denkbar, mit denen man Konuselemente wesentlich genauer diskretisieren könnte.

Für das Einlesen der Geometriedatei wird das Hilfsmodul *edf.c* verwendet. Darin ist die Funktion

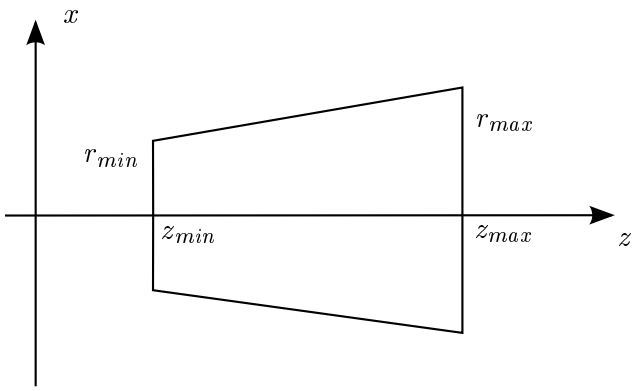

Abbildung 6.3: Parameter einer Konus-Elektrode

```
long edf_read(const char* filename, struct Electrode*** electrodes,
              unsigned short* nrot);
```

enthalten, die die Datei *filename* einliest. Sie gibt die Anzahl der gefundenen Elektroden zurück oder -1 im Falle eines Fehlers. Die Elektroden selbst werden in einem Array durch den Zeiger *electrodes* zurückgegeben. Sie werden durch C-Structs repräsentiert, die den Typ, die Geometrie, das Potential und die Ladungsdichte, welche zunächst mit null initialisiert sind, enthalten. Die von dieser Funktion gelieferten Elektroden sind bereits diskretisiert. Das Array enthält also nur Draht- und Rechteck-Elektroden, auch wenn in der Geometriedatei Konen existieren. Weiterhin gibt die Funktion noch die totale Rotationssymmetrie über den Zeiger *nrot* zurück. Dies ist der kleinste gemeinsame Teiler der n_{rot}-Parameter der Elektroden. Für die Rechnungen wird dann ausschließlich diese Symmetrie benutzt. Elektroden mit höherer Symmetrie werden dann passend kopiert und rotiert. Zu diesem Zweck bietet das Modul *electrode.c* verschiedene Funktionen, die die dafür notwendigen Operationen an den *Electrode*-Structs durchführen.

Potentialberechnung

Das elektrische Potential Φ einer Elektrodengeometrie ist die Summe der Einzelpotentiale der Oberflächenelemente Φ_n,

$$\Phi = \sum_n \Phi_n. \tag{6.23}$$

Die Einzelpotentiale erhalten wir durch das Flächenintegral über das Element

$$\Phi_n(\vec{r}) = \sigma_n \int d^2r' \frac{1}{|\vec{r} - \vec{r}\,'|}. \tag{6.24}$$

Für diese Aufgabe existieren im Modul *electrode.c* mehrere Funktionen. Sie berechnen die Potentiale von Draht- und Rechteck-Elektroden. Diese Funktionen sind der

einzige Teil von *Elcd4*, der – allerdings in stark modifizierter Form – von *Elcd3_3* übernommen wurde. Sie wurden angepasst, so dass sie die Geometrieinformationen aus einem *Electrode*-Struct beziehen und dann unter Verwendung der Funktionen aus *linalg.c* das Potential an jedem beliebigen Punkt $\vec{r}$ bestimmen können.

Multipolberechnung

Zusätzlich müssen für alle Elektrodentypen Funktionen zur Berechnung der Multipolmomente implementiert werden. Dazu muss man die Multipolberechnung aus Kapitel 6.2.2 durchführen. Dabei wird über die einzelnen Oberflächenelemente integriert. Die folgenden Formeln geben die Multipolmomente bezüglich des Koordinatenursprungs an. Einen anderen Referenzpunkt $\vec{c}$ erhält man durch eine geeignete Koordinatentransformation, z. B. im Rechteck-Fall $\vec{p} \to \vec{p} - \vec{c}$.

Führt man die Integration für rechteckige Elektroden aus, erhält man die folgenden Gleichungen.

Monopol:

$$q = \int d^3r' \varrho(\vec{r}\,') = \varrho ab \tag{6.25}$$

Mit ϱ ist die Ladungsdichte gemeint, die auf dem ganzen Element als konstant angenommen ist, a und b sind die Seitenlängen des Rechtecks.

Dipol:

$$\vec{d} = \int d^3r' \varrho(\vec{r}\,')\,\vec{r}\,' = q\left(\vec{p} + \frac{a}{2}\vec{n}_1 + \frac{b}{2}\vec{n}_2\right) \tag{6.26}$$

Quadrupol:

$$\begin{aligned} Qij = 3ab\varrho\Big(&p_i p_j + \tfrac{1}{2}a(p_i n_{1j} + p_j n_{1i}) + \tfrac{1}{2}b(p_i n_{2j} + p_j n_{2i}) \\ &+ \tfrac{1}{4}ab(n_{1i}n_{2j} + n_{1j}n_{2i}) + \tfrac{1}{3}a^2 n_{1i}n_{1j} + \tfrac{1}{3}b^2 n_{2i}n_{2j}\Big) \\ &- \Big(p^2 + a\vec{p}\cdot\vec{n}_1 + b\vec{p}\cdot\vec{n}_2 + \tfrac{1}{3}\left(a^2 + b^2\right)\Big)\delta_{ij} \end{aligned} \tag{6.27}$$

δ_{ij} ist das Kronecker-Symbol. Die übrige Notation ist identisch zu der oben verwendeten.

Bei drahtförmigen Elektroden erhält man folgende Multipolmomente:

Monopol:

$$q = 2\pi dl \tag{6.28}$$

Dipol:

$$\vec{d} = \tfrac{1}{2}q\left(\vec{a} + \vec{b}\right) \tag{6.29}$$

Quadrupol:

$$\begin{aligned} Q_{ij} = \tfrac{1}{4}q\Big(&\big(6p_i p_j + 3(p_i l_j + p_j l_i) + 2l_i l_j\big) - \\ &2\big(p^2 + \vec{p}\cdot\vec{l} + \tfrac{1}{3}l^2 + \tfrac{1}{4}d^2\big)\delta_{ij} + \tfrac{3}{4}d^2 A_{ij}\Big) \end{aligned} \tag{6.30}$$

Dabei wurden die folgenden Abkürzungen benutzt:

$$\vec{l} = \vec{b} - \vec{a} \tag{6.31}$$

$$A_{11} = \cos^2\varphi\cos^2\vartheta + \sin^2\varphi \tag{6.32}$$

$$A_{22} = \sin^2\varphi\cos^2\vartheta + \cos^2\varphi \tag{6.33}$$

$$A_{33} = \sin^2\vartheta \tag{6.34}$$

$$A_{12} = A_{21} = \sin\varphi\cos\varphi\sin^2\vartheta \tag{6.35}$$

$$A_{13} = A_{31} = -\cos\varphi\cos\vartheta\sin\vartheta \tag{6.36}$$

$$A_{23} = A_{32} = -\sin\varphi\cos\vartheta\sin\vartheta \tag{6.37}$$

(l, ϑ, φ) ist die Darstellung von $\vec{l}$ in sphärischen Koordinaten. Die vollständigen Rechnungen zu den Multipolentwicklungen befinden sich im Anhang A.1.

Im Modul *electrode.c* sind Funktionen für diese Berechnung implementiert.

Elektrodendiskretisierung

Bei der BEM wird auf einem Oberflächenelement jeweils eine konstante Ladungsdichte angenommen. Bei stark fluktuierenden Ladungsdichten wird die Methode daher ungenau. Um das zu vermeiden, muss man die Oberflächenelemente so klein wählen, dass die Abweichung vernachlässigbar ist. Das ist gerade bei Draht- und Konus-Elektroden wichtig, denn hier variieren die Ladungsdichten an den Enden der Elektrode sehr stark. Man benötigt daher einen Algorithmus, der einen Draht in kleinere Drahtelemente diskretisiert, die zu den Enden hin kürzer werden. Bei Konen kommt noch erschwerend dazu, dass das Programm eigentlich nicht mit Konen rechnen kann und diese daher durch kleine Rechtecke approximiert werden müssen.

Der verwendete Diskretisierungsalgorithmus stammt von Ferenc Glück. Er war ursprünglich für *Elcd3_3* in den zwei Zusatzprogrammen *input1* und *input2* implementiert. Diese lesen eine *Elcd3_2*-Geometriedatei, die Konen bzw. Drähte diskretisiert und die Geometrie im *Elcd3_3*-Format wieder ausgeben. Bei *Elcd4* ist dieser Algorithmus direkt integriert. So geschieht die Diskretisierung automatisch beim Einlesen der Geometrie. Das hat zwei entscheidene Vorteile, nämlich:

- Bei einer Änderung an der Geometrie entfällt der zusätzliche Aufruf des Diskretisierungsprogramms. Gerade dies hat sich in der Praxis als häufige Fehlerquelle herausgestellt, da zwar die Originalgeometrie aktualisiert, aber noch mit der alten diskretisierten Version gerechnet wurde.

- Die Geometriedateien werden kleiner. Je nach Diskretisierung kann sich die Dateigröße bis zum einem Faktor 100 oder mehr vergrößern. Neben der Speicherplatzersparnis bringt dies auch erheblich mehr Übersicht. Eine diskretisierte Geometrie kann nur noch sehr schwer kontrolliert oder visualisiert werden.

Man könnte argumentieren, dass die Verwendung von bereits diskretisierten Geometrien später eine Zeitersparnis bringen wird. Allerdings ist die Zeit, die zur Diskretisierung benötigt proportional zur Anzahl der der Elemente N, aber die gesamte

Rechenzeit proportional zu mehr als N^2. Daher können wir die Diskretisierungszeit gewöhnlich vernachlässigen.

Der bei der Elektrodendiskretisierung verwendete Algorithmus wurde bei der Entwicklung von *Elcd4* prinzipiell nicht modifiziert. Er wurde lediglich von Grund auf neu implementiert und dabei an die modernisierte Programmstruktur angepasst. Vergleichstests haben aber gezeigt, dass die Resultate in beiden Fällen identisch sind.

Das Diskretisierungsverfahren geschieht in mehreren Schritten. Die ersten sind bei Drähten und Konen analog. Das Intervall $[0, 1]$ wird zuerst in kleine Subintervalle zerlegt. Dabei werden die Parameter *scale* (s) und *power* (p) verwendet, s ist die Anzahl der Subintervalle und p bestimmt das Verhalten an den Enden der Elektrode. Bei $p = 1$ sind alle Subintervalle gleich groß, falls $p > 1$ sind die Randintervalle kleiner und bei $p < 1$ werden die Intervalle größer. Gewöhnlich wählt man $p = 2$. Das ist auch die Vorgabe, falls p in der Geometriedefinition weggelassen wird. Die Implementierung dazu findet sich im Modul *edf.c* in der Funktion

```
double* discretise_interval(int num, double power).
```

num entspricht hier s. Die Funktion gibt einen Zeiger auf ein neu angelegtes Array zurück, das später mit *free()* wieder freigegeben werden muss. In diesem Array befinden sich die Längen der einzelnen Subintervalle d_i.

Nun werden die Subintervalle benutzt um Drähte bzw. Konen aufzuteilen. Die Länge der neuen Teilstücke ist dabei die Länge des Subintervalls multipliziert mit der ursprünglichen Länge der nicht-diskretisierten Elektrode. Sind d_j die Längen der Subintervalle, dann gilt für die Endpunkte $\vec{a}_i$ und $\vec{b}_i$ der kleinen Drahtelemente

$$\vec{a}_i = \vec{a} + \vec{b}\sum_{j=1}^{i-1} d_j \qquad\qquad \vec{b}_i = \vec{a} + \vec{b}\sum_{j=1}^{i} d_j. \tag{6.38}$$

Die so neu entstandenen Elektroden sind mit demselben Potential belegt wie die

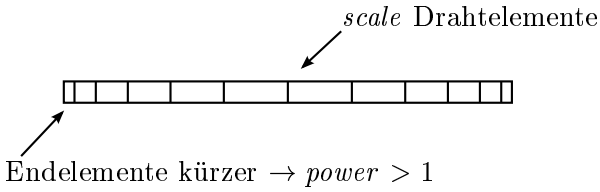

Abbildung 6.4: Diskretisierung einer Draht-Elektrode

Ursprungselektrode und besitzen auch dieselbe Rotationssymmetrie um die z-Achse. Die Diskretisierung eines Drahtes ist an dieser Stelle abgeschlossen (Abbildung 6.4). Für eine Konus-Elektrode ist dies nur der erste Schritt des Verfahrens. Die Subintervalle werden hier benutzt um die Elektrode entlang der z-Achse in kleinere Konen zu unterteilen (Abbildung 6.5). Für deren Anfangs- und Endkoordinaten z^i_{min}, z^i_{max},

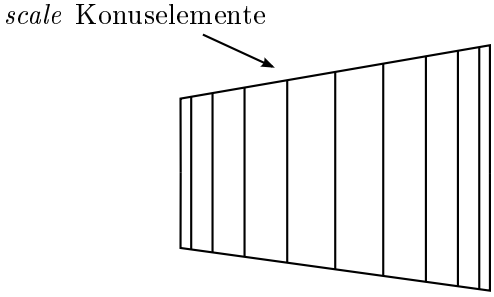

Abbildung 6.5: Diskretisierung einer Konus-Elektrode entlang der z-Achse

r^i_{min} und r^i_{max} gilt

$$z^i_{min} = z_{min} + z_{max} \sum_{j=1}^{i-1} d_j \qquad z^i_{max} = z_{min} + z_{max} \sum_{j=1}^{i} d_j \tag{6.39}$$

$$r^i_{min} = r_{min} + r_{max} \sum_{j=1}^{i-1} d_j \qquad r^i_{max} = r_{min} + r_{max} \sum_{j=1}^{i} d_j. \tag{6.40}$$

Wichtig für die weiteren Schritte dieses Verfahren ist, dass die Bedingung

$$z^i_{min} \leq z^i_{max} \quad \forall i \in [1, s] \tag{6.41}$$

erfüllt ist. Gegebenenfalls müssen dazu Variablen umbenannt werden.

Für die Ladungsdichtenberechnung ist dies aber noch nicht ausreichend. Die neuen Konen müssen noch in einem zweiten Schritt in Rechtecke zerlegt werden (Abbildung

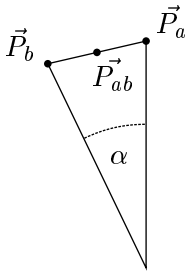

Abbildung 6.6: Diskretisierung einer Konus-Elektrode entlang des Umfangs

6.6). Hier wird ein dritter Parameter n_{rot} verwendet. Er gibt die Anzahl der Rechtecke an, in die die Konen entlang ihres Umfangs unterteilt werden. Genauer gesagt wird für jeden Konus nur ein Rechteck angelegt und diesen n_{rot} mal um die z-Achse rotiert. Die übrigen Geometrieparameter dieses Rechtecks werden aus der Geometrie des Konus bestimmt, wie Abbildung 6.6 demonstriert. Dabei werden verschiedene

Hilfspunkte definiert:

$$\vec{P}_a = (r_{min}, 0, z_{min}) \qquad \vec{P}_c = (r_{max}, 0, z_{max}). \tag{6.42}$$

Diese Punkte werden um den Winkel $\alpha = 2\pi/n_{rot}$ um die z-Achse rotiert, so dass $\vec{P}_a \to \vec{P}_b$ und $\vec{P}_c \to \vec{P}_d$. Damit werden weitere Punkte definiert:

$$\vec{P}_{ab} = \tfrac{1}{2}\left(\vec{P}_a + \vec{P}_b\right) \qquad \vec{P}_{cd} = \tfrac{1}{2}\left(\vec{P}_c + \vec{P}_d\right) \tag{6.43}$$

Aus diesen Hilfspunkten werden nun die Geometrieparameter der neu erstellten Rechteck-Elektroden berechnet.

$$\vec{p} = \vec{P}_a \qquad a = |\vec{P}_b - \vec{P}_a| \qquad b = |\vec{P}_{cd} - \vec{P}_{ab}| \tag{6.44}$$

$$\vec{n}_1 = \frac{\vec{P}_b - \vec{P}_a}{a} \qquad \vec{n}_2 = \frac{\vec{P}_{cd} - \vec{P}_{ab}}{b} \tag{6.45}$$

Diese Elektrode kann nun für die Ladungsberechnung verwendet werden. Diese Form der Diskretisierung ist sehr ungenau und nur anwendbar, falls n_{rot} und *scale* groß sind. Dabei ist auch sehr kritisch, dass allein aus Rechtecken keine konusförmige Fläche zusammengesetzt werden kann. Da aber beim Design von *Elcd4* auf Modularität und Erweiterbarkeit Wert gelegt wurde, wäre es denkbar in Zukunft an dieser Stelle Elektroden in Form eines Dreiecks hinzuzufügen. Damit wäre eine wesentlich bessere Approximation eines Konus möglich.

Elektrodengruppierung

Wie bereits in Kapitel 6.3 beschrieben, ist die grundlegende Erweiterung der BEM in *Elcd4* die Gruppierung von Oberflächenelementen. Dazu wird ein Boxsystem verwendet, das von einem dreidimensionalen Raumgitter erzeugt wird und zwei Aufgaben erfüllt. Zum einen zerlegt es das gesamte Volumen in einzelne Boxen und erlaubt eine eindeutige Zuordnung der Elementen zu einer Box. Andererseits bietet das Boxsystem auch eine Unterteilung der Elektroden in „nah“ und „fern“.

Da die Rotationssymmetrie um die z-Achse bereits in weiten Teilen des Programms eine zentrale Bedeutung hat, wird sie auch vom Raumgitter berücksichtigt. Das Raumgitter ist um die z-Achse angeordnet (Abbildung 6.7), so dass es die komplette Elektrodengeometrie enthält. Die begrenzenden Koordinatens des Gitters z_{min}, z_{max}, r_{min} und r_{max} werden nach dem Einlesen der Geometrie automatisch so gewählt, dass sie alle Elektroden umfassen. Die Anzahl der Boxen wird vom Benutzer über die Parameter n_r, n_z und n_φ bestimmt. In radialer Richtung ist das Boxsystem in n_r Boxen unterteilt, in z-Richtung entsprechend n_z. In Umfangsrichtung ist die Unterteilung etwas komplexer, weil das Boxsystem die gleiche Rotationssymmetrie aufweisen muss wie die gesamte Geometrie n_{rot}. Sie kann zusätzlich noch vom Anwender über den Parameter n_φ variiert werden. So erhält man eine Unterteilung in Umfangsrichtung von

$$n_{Gitter} = \begin{cases} n_{rot} n_\varphi & \text{falls } n_\varphi > 0 \\ -\frac{n_{rot}}{n_\varphi} & \text{falls } n_\varphi < 0 \end{cases} \tag{6.46}$$

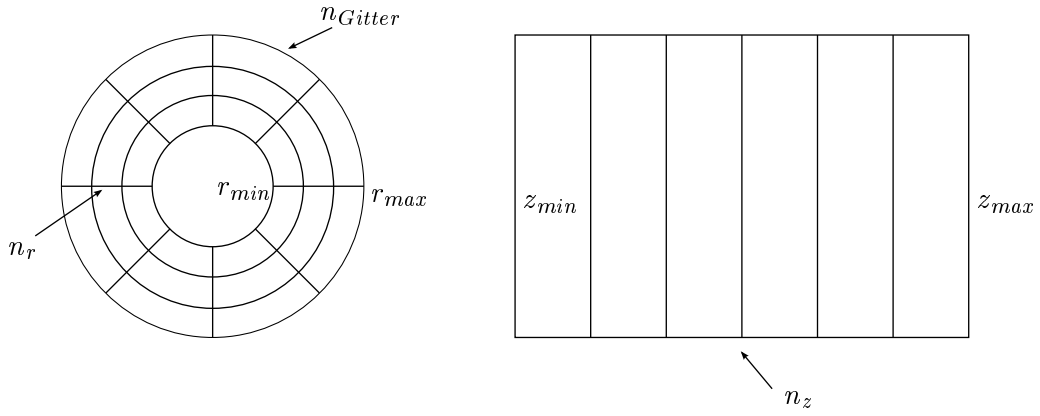

Abbildung 6.7: In Elcd4 verwendetes Raumgitter

Boxen. Falls $n_{Gitter} < n_{rot}$ werden Boxen bei der Berechnung zusammengefasst, was gerade bei hohen Rotationssymmetrien einen Geschwindigkeitsvorteil bringt.

Im Falle einer gespiegelten Geometrie umfasst das Boxsystem lediglich die ungespiegelte Geometrie. Bei der Berechnung der Multipolapproximation wird dann jede Box zusätzlich gespiegelt und doppelt ausgewertet.

Die Zuordnung eines Oberflächenelements zu einer Box geschieht nun über den Mittelpunkt des Elements. Es wird der Box zugeordnet, die seinen Mittelpunkt enthält. Die Mittelpunkte der Elemente lassen sich dazu einfach bestimmen. Für Rechtecke erhält man

$$\vec{c}_{\text{Rechteck}} = \vec{p} + \tfrac{a}{2}\vec{n_1} + \tfrac{b}{2}\vec{n_2} \tag{6.47}$$

und für Drähte

$$\vec{c}_{\text{Draht}} = \tfrac{1}{2}\big(\vec{a} + \vec{b}\big). \tag{6.48}$$

Die Einteilung der Elemente in nah und fern erfolgt boxweise. Dabei werden für ein Elementpaar die entsprechenden Boxen bestimmt und dann der Abstand der beiden Boxmittelpunkte berechnet. Ist dieser größer als R_n, gelten die beiden Elemente als fern zueinander. Der Parameter R_n wird vom Benutzer angegeben.

Iterative Ladungsdichtenberechnung

Nachdem mit Hilfe des Boxsystems das approximierte Gleichungssystem (6.21) aufgestellt wurde, muss man ein Lösungsverfahren darauf anwenden. Bei Elcd4 wird das Gauß-Seidel-Verfahren benutzt. Es ist eines der einfachsten iterativen Verfahren. Nähere Information zu diesem Verfahren finden sich in [Pla00]. Die Vorschrift für eine Iteration ist

$$\sigma_i^{n+1} = \frac{1}{k_{ii}}(x_i - k_{i1}\sigma_1^{n+1} + \cdots + k_{ii-1}\sigma_{i-1}^{n+1} + k_{ii+1}\sigma_{i+1}^{n} + \cdots + k_{iN}\sigma_N^{n}). \tag{6.49}$$

Nach dieser Vorschrift wird das Gleichungssystem mehrmals iteriert. Ein einzelner Iterationsschritt wird als kleine Iteration bezeichnet, im Gegensatz zur großen, die

alle Schritte umfasst und die anschließende Neuberechnung der Multipolmomente. Die Anzahl der Iterationschritte wird durch den Parameter n_{steps} vorgegeben. Für diesen Parameter wird ein Startwert vom Benutzer angegeben, danach wird er nach jeder großen Iteration angepasst. Dies ist nötig, weil zu viele Gauß-Seidel-Schritte an dieser Stelle ineffektiv sind. Sie führen nur dazu, dass die aktuellen Ladungsdichten sich der Lösung des aktuellen Gleichungsystems annähern. Da aber nach jeder großen Iteration $\vec{X}$ nach (6.20) neu berechnet wird, ändert sich das aktuelle Gleichungssystem ebenfalls. So lässt sich die höchste Konvergenzgeschwindigkeit erreichen, wenn das gesamte Gleichungssystem konvergiert. Das bedeutet, dass es für n_{steps} einen optimalen Bereich gibt. Wenn n_{steps} in diesem Bereich liegt, dann sollte die Änderung der Ladungsdichten bei dem ersten Schritt in derselben Größenordnung liegen wie die Änderung des letzten Schrittes der vorherigen Iteration.

Speichern der Ladungsdichten

Nachdem die Berechnung der Ladungsdichten abgeschlossen ist, können damit elektrische Potentiale bestimmt werden. Meist geschieht dies aber erst in einem zweiten Schritt, daher müssen die Ladungsdichten gespeichert werden. Zu diesem Zweck bietet Elcd4 die CDF-Datei (*Charge Density File*) an. Das ist eine Datei, in der in einem Binärformat alle Informationen zu den Oberflächenelementen und den dafür bestimmten Ladungsdichten gespeichert sind. Für eine spätere Potentialberechnung muss man diese Datei erneut einlesen und kann sofort aus den Ladungsdichten und Geometrie elektrische Potentiale für jeden beliebigen Punkt bestimmen.

Der Schreibvorgang geschieht durch den Aufruf der Funktion

```
void elcd_write_cdf(Elcd* elcd, const char* filename).
```

Sie schreibt die aktuellen Ladungsdichten in die Datei *filename*. Der Name sollte die Endung *.cdf* besitzen.

Statt die Daten in einem ASCII-Formet zu speichern wie die Vorgängerprogramme, wird ein Binärformat verwendet. Dadurch werden Rundungsfehler beim Schreiben von Fließkommazahlen vermieden. Bisher ging dadurch immer ein gewisses Maß an Genauigkeit verloren. In einer Binärdatei lassen sich mehr Informationen in einer kleineren Datei speichern.

Es ist auch möglich die Ladungsdichten aus einer CDF-Datei als Ausgangspunkt für eine neue Ladungsdichtenberechnung zu benutzen. Dann werden die Ladungsdichten bei Start der Iteration nicht mit Null initialisiert, sondern mit den Werten der vorherigen Iteration. Damit lässt sich bei minimalen Geometrievariationen sehr viel Zeit sparen, weil man nun bereits von einer Näherung für die Ladungsdichten startet. So sind nur wenige Schritte nötig um die Ladungsdichten zu aktualisieren. Gerade bei Toleranzrechnungen ist diese Funktion sehr nützlich. Nähere Informationen zu dieser Art von Rechnung befinden sind in [Hug08].

7 Tests

7.1 Programmparameter

Im Gegensatz zu den Vorgängerprogrammen gibt es bei *Elcd4* wesentlich mehr Laufzeitparameter. Der Grund dafür liegt in der größeren Komplexität des verwendeten Algorithmus. Diese Parameter müssen an die jeweilige Geometrie angepasst werden. Es gibt keine universellen Standardwerte. Stattdessen müssen sie vom Benutzer manuell vor dem Beginn der Berechnung gesetzt werden. Teilweise gibt es sinnvolle Vorgabewerte, allerdings oft auch nicht. In Tabelle 7.1 sind alle Laufzeitparameter von *Elcd4* aufgeführt.

Die große Anzahl von Parametern ist ein großer Nachteil für die Benutzerfreundlichkeit. Dieses Problem sollte in der Zukunft durch einen intelligenten Algorithmus gelöst werden, der die Geometrie analysiert und die Parameter entsprechend setzt. Für den Parameter n_{steps} existiert ein ähnlicher Ansatz bereits. Während der Iteration wird die Änderung der Ladungsdichten überwacht und dabei versucht, die Geschwindigkeit der Iteration über eine Variation von n_{steps} zu optimieren. Daher genügt es hier, dass der Benutzer einen Anfangswert für n_{steps} vorgibt. In der Regel hat sich ein Wert von 20 für n_{steps} in den Testläufen als sinnvoll erwiesen. Der Parameter C bestimmt, wie weit iteriert werden soll. Sobald die Änderung der Ladungsdichten in einem Iterationsschritt unter diesen Schwellwert fällt, wird die Iteration beendet. Mit ihm kann also die gewünschte Genauigkeit angegeben werden.

Die Parameter haben einen großen Einfluss auf das Verhalten des Programms. Doch kein anderer Parameter beeinflusst das Konvergenzverhalten des Algorithmus wie die Größe des Nahbereichs R_n. Wählt man sie zu klein, konvergiert der Algorithmus nicht mehr oder nur noch extrem langsam. Ist der Nahbereich zu groß, ist zwar die Konvergenzgeschwindigkeit am größten, allerdings steigt damit auch der Bedarf an Arbeitsspeicher. Der Grenzfall $R_n \to \infty$ entspricht der Situation in den bisherigen

Tabelle 7.1: Laufzeitparameter in *Elcd4*

n_z	Anzahl der Boxen in z-Richtung
n_φ	Anzahl der Boxen in Umfangsrichtung
n_r	Anzahl der Boxen in radialer Richtung
n_{steps}	Anzahl der Gauß-Seidel-Schritte innerhalb einer großen Iteration
R_n	Maximaler Abstand, mit dem zwei Boxen als nah gelten
C	Schwellwert der Ladungsdichtenänderung bei der Abbruchbedingung der Iteration

Programmen. Das Konvergenzverhalten wird zum Teil auch durch die Feinmaschigkeit des Raumgitters beeinflusst. Sie gibt die Anzahl der Multipole vor, die für die Approximation der Fernzone verwendet werden. Ist diese zu klein, geht Rechengenauigkeit verloren und die minimale Größe des Nahbereichs ist dementsprechend groß. Im folgenden Abschnitt wird die Abhängigkeit von diesen Parameter auch noch einmal quantitativ an Hand von Testgeometrien demonstriert.

7.2 Einfache Vergleichsrechnungen

Im Verlaufe der Entwicklung von *Elcd4* wurde das Programm an verschiedenen Elektrodengeometrien getestet, angefangen von sehr einfachen bis hin zu komplexen Strukturen, entsprechend dem Stand der Entwicklung. Die meisten Tests hatten zum Ziel Fehler zu finden und verschiedene Programmkomponenten zu optimieren. Ausführliche quantitative Test wurden mit einer Geometrie durchgeführt, die in Abbildung 7.1 gezeigt ist. Sie besteht aus einem 100 m langen Zylinder mit einem Radius von 1 m. Innerhalb dieses Zylinders befinden sich 11 rotationssymmetrisch angeordnete Drähte mit einem Abstand von der Zylinderachse von 0,8 m und einem Radius von 0,1 mm.

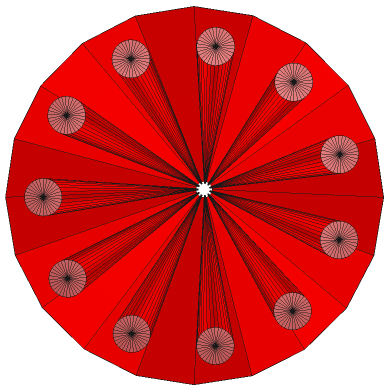

Abbildung 7.1: ROOTsim-Darstellung der Testgeometrie (Drähte 100fach vergrößert)

Diese Elektrodengeometrie erlaubt es, alle relevanten Programmteile zu testen. Dazu erstellt man die Testgeometrie zuerst als *Elcd3_2*-Geometrie. Diese wird nun einerseits mit den Diskretisierungsprogrammen *input1* und *input2* von Ferenc Glück in das *Elcd3_3*-Format konvertiert. Anderseits wird die Testgeometrie mit *convert.py* in eine EDF-Datei umgewandelt, dabei ist es wichtig, dass man dieselben Diskretisierungsparameter für beide Operationen wählt. Nun berechnet man mit *Elcd3_3* Referenzladungsdichten. Dafür wurde eine leicht abgewandelte Version von *Elcd3_3* verwendet, die mit doppelter Genauigkeit rechnet, mehr Nachkommastellen ausgibt und intern keinerlei Näherungen verwendet. Nur so war es möglich sinnvolle Ver-

gleiche durchzuführen. Danach kann man die Ladungsdichten von *Elcd4* berechnen lassen. Ein Python-Skript führt dann einen Vergleich der Ladungsdichten durch und gibt deren mittlere relative Abweichung aus. Wenn man während der Laufzeit von *Elcd4* die Laufzeit und den Speicherverbrauch misst, kann man durch Variation der Laufzeitparameter ein genaues Leistungsprofil der verbesserten Boundary Element Methode bestimmen. Zusätzlich zu den eben genannten Parametern lässt sich noch die Anzahl der Oberflächenelemente durch den Diskretisierungsparameter *scale* bestimmen. Ferner kann man die Rotationssymmetrie n_{rot} der Geometrie variieren. Dabei ist aber zu beachten, dass die Rotationssymmetrie in der Zylinderdiskretisierung mit der der Drähte übereinstimmt. Denn *Elcd3_3* behält die unterschiedlichen Rotationssymmetrien als Approximation bei. Bei *Elcd4* ist dies aufgrund des Boxsystems nicht mehr möglich. Hier werden die globale Rotationssymmetrie bestimmt und die Elektroden entsprechend vervielfacht. Daher würden bei unterschiedlichen Rotationssymmetrien in beiden Fällen unterschiedliche Geometrien berechnet werden und ein Vergleich der Ergebnisse wäre schwierig.

Der für diese und alle folgenden Rechnungen verwendete Computer verfügt über einen Intel Core 2 Duo 6300 Prozessor mit einer Taktfrequenz von 1,86 GHz und 2 GB Arbeitsspeicher. Das darauf installierte Betriebsystem ist Archlinux [Arc] mit dem Compiler gcc in der Version 4.2.2.

In der ersten Testrechnung wurde der Einfluss des Parameters R_n untersucht. Dazu wurde die Testgeometrie mit verschiedenen Werten für den Parameter n_z berechnet und dabei die Rechenzeit und der Speicherverbrauch unter Variation von R_n bestimmt. Im Folgenden sind die Resultate für $n_z = 80$ und $n_z = 100$ bei einer Diskretisierung mit $scale = 1000$ aufgeführt, alle übrigen Ergebnisse befinden sich in Tabelle D.1 im Anhang. Diese Diskretisierung erzeugt 2000 Oberflächenelemente. Für die übrigen Parameter galt bei diesen Rechnungen: $n_r = 1$, $n_\varphi = 1$, $n_{steps} = 20$ und $C = 10^{-10}$.

Abbildung 7.2 zeigt den Einfluss des Parameters R_n auf den Speicherverbrauch. Man erkennt einen nahezu linearen Zusammenhang. Dieser resultiert aus der besonderen Form der Testgeometrie. Der erforderliche Speicher ist proportional zur Anzahl der nahen Elemente. Aber da sich die Geometrie, und damit auch das Boxsystem, hauptsächlich in z-Richtung erstreckt, ist diese Anzahl auch proportional zu R_n. Wären die Elemente im Raum gleichverteilt, würde man einen kubischen Zusammenhang erwarten. Zusätzlich kann man der Grafik entnehmen, dass der Parameter n_z hier eine untergeordnete Rolle spielt. Dies entspricht jedoch genau den Erwartungen, da der Großteil des Speichers für die Informationen zur Nahzone benötigt wird.

Der komplette Programmablauf gliedert sich in zwei große Teile. Das sind zum einen die Konstruktion der Nahzone mit der Berechnung des Gleichungssystems für die BEM und zum anderen die anschließende Gauß-Seidel-Iteration. In Abbildung 7.3 ist die für den ersten Abschnitt benötigte Zeit gegen den Parameter R_n aufgetragen. Der erkennbare Zusammenhang entspricht dem bereits bei der Untersuchung des Speicherverbrauchs beobachteten, da die erforderliche Rechenzeit wiederum proportional zu der Anzahl der nahen Elemente ist.

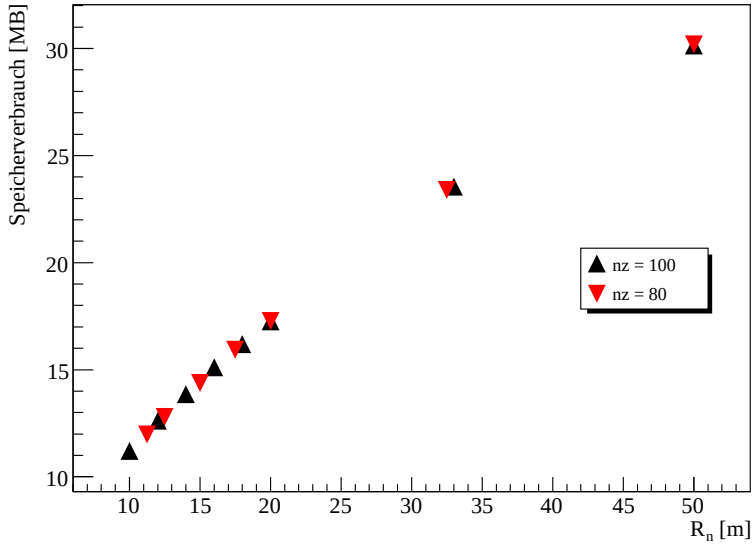

Abbildung 7.2: Speicherverbrauch in Abhängigkeit von R_n

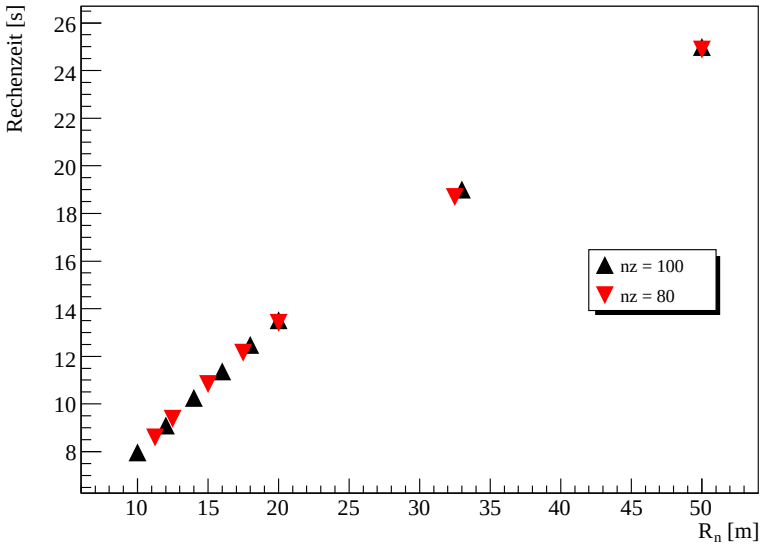

Abbildung 7.3: Rechenzeit zur Berechnung der Nahzone in Abhängigkeit von R_n

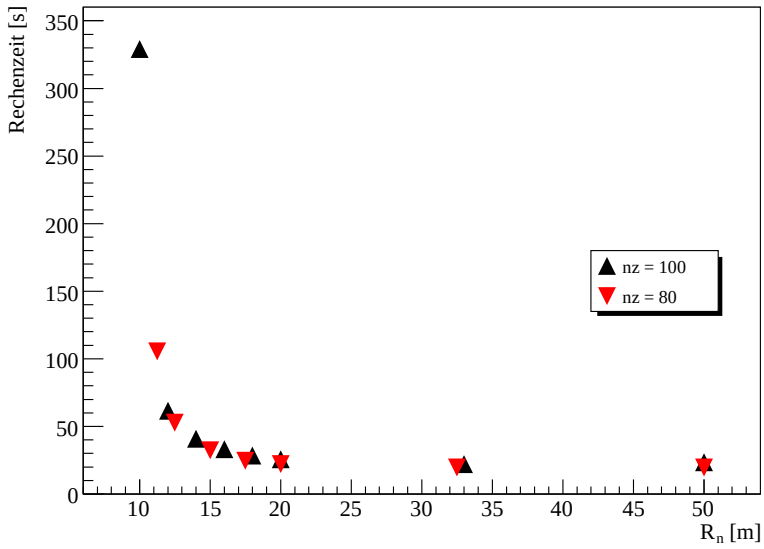

Abbildung 7.4: Dauer der Iteration der Abhängigkeit von R_n

In Abbildung 7.4 sieht man die Dauer der iterativen Ladungsdichtenberechnung in Abhängigkeit des Parameters R_n. Das Verhalten ist nun etwas komplexer aufgrund des großen Einflusses von R_n. Je kleiner die Nahzone ist, desto größer wird die Rechenzeit für die Iteration. Auch dies entspricht den Erwartungen, denn bei einer kleinen Nahzone braucht man mehr große Iterationsschritte bis das Gleichungssystem konvergiert. Allerdings kommen noch weitere Effekte dazu. So ist beispielsweise die Zeit, die für eine kleine Gauß-Seidel-Iteration benötigt wird, proportional zur Anzahl der nahen Elemente. Die Anzahl der Boxen wirkt sich stark auf die Dauer einer großen Iteration aus, bei der die Multipolmomente aus den Ladungsdichten aktualisiert werden. Bei einer großen Zahl an Boxen, zum Beispiel bei einer sehr hohen Gitterrotationssymmetrie n_{Gitter}, wird die Berechnung der Multipolapproximation sehr zeitaufwändig.

Da die Gesamtrechenzeit die Summe der beiden einzelnen Rechenzeiten ist, wird sie sowohl bei einer kleinen als auch bei einer großen Nahzone groß (Abbildung 7.5). Hier gilt es, einen optimalen Wert für R_n zu finden. Dieser ist aber stark von der zu berechnenden Geometrie abhängig. Für verschiedene Werte von n_z wurden die optimalen Werte manuell bestimmt. In Zukunft könnte basierend auf umfangreichen Tests ein intelligenter Algorithmus entwickelt werden, der anhand von Geometrieeigenschaften R_n automatisch korrekt setzt. Dies würde jedoch über den Rahmen dieser Arbeit hinausgehen.

Mit Hilfe der optimalen Nahzonengrößen lässt sich der Einfluss der Boxanzahl n_z auf die benötigte Rechenzeit untersuchen (Abbildung 7.6). Es zeigt sich ein nahezu lineares Verhalten, da die Berechnung der Multipolmomente einen Großteil der Zeit

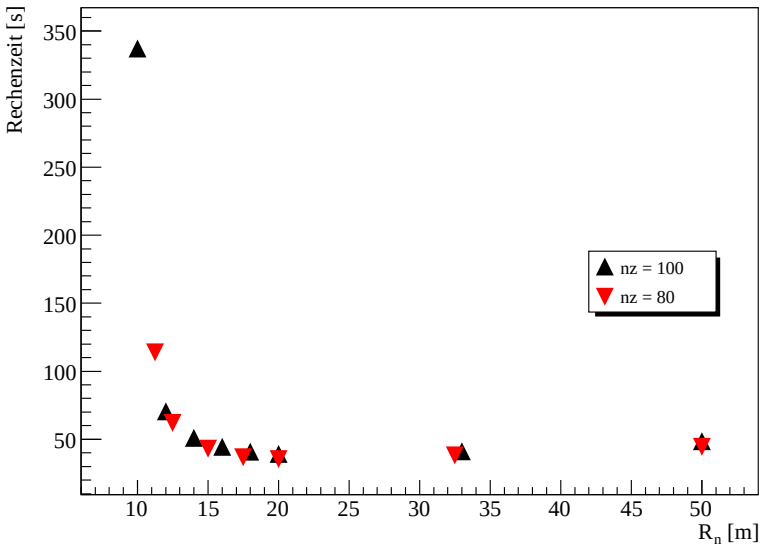

Abbildung 7.5: Gesamte Rechenzeit in Abhängigkeit von R_n

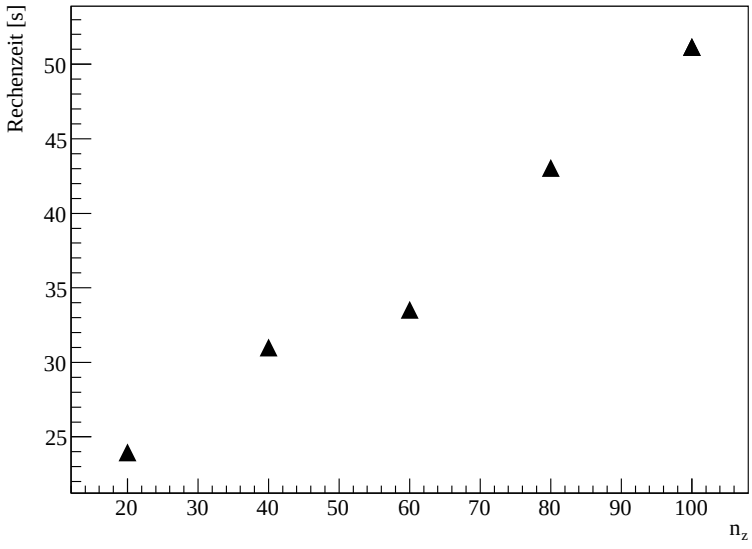

Abbildung 7.6: Optimale Rechenzeit für verschiedene Werte von n_z

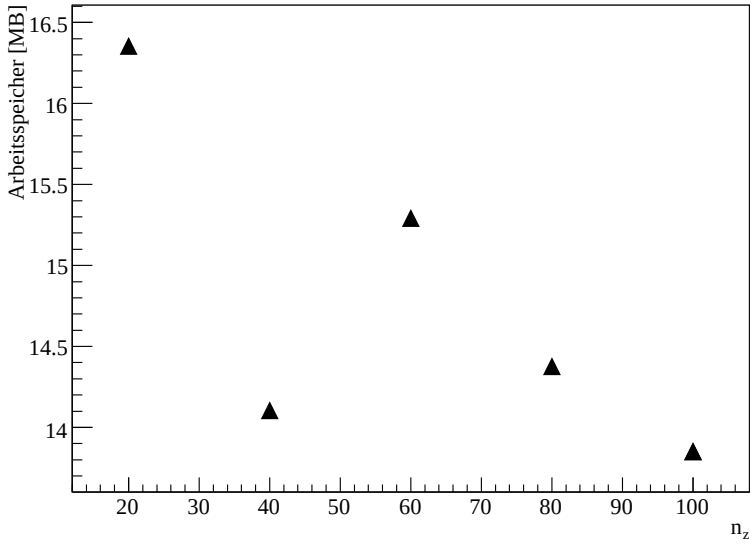

Abbildung 7.7: Optimaler Speicherverbrauch für verschiedene Werte von n_z

in Anspruch nimmt und die Anzahl der zu berechnenden Multipole proportional zu n_z ist. Dennoch weichen die Werte teilweise stark vom linearen Verhalten ab. Dieser Effekt tritt auf, weil sich nicht für jeden Wert von n_z die optimale Nahzonengröße einstellen lässt. Die Nahzone besteht immer aus einer ganzzahligen Menge von Boxen, daher sind gerade bei kleinen Werten für n_z größere Abweichungen zu erwarten.

In Abbildung 7.7 ist das Verhalten des Speicherverbrauchs bei einer optimalen Nahzonengröße dargestellt. Man kann erkennen, dass der Speicherverbrauch mit steigendem n_z sinkt. Dieser Effekt lässt sich mit der steigenden Anzahl an Multipolen erklären. Bei einer hohen Boxanzahl stehen mehr Multipole für die Approximation zur Verfügung, die dadurch genauer wird. Diese zusätzlich Genauigkeit kann nun verwendet werden, um die Nahzone weiter zu verkleinern, was wiederum den Speicherverbrauch reduziert.

Zusätzlich zum Speicherverbrauch und zur Rechenzeit ist die Genauigkeit ein wichtiges Vergleichskriterium. Dazu wurden die von *Elcd4* ermittelten Ladungsdichten mit denen von *Elcd3_3* verglichen. Abbildung 7.8 zeigt die relative Abweichung für verschiedene Werte von R_n. Das Ergebnis entspricht genau den Erwartungen. Bei einer kleinen Nahzone kann man den Effekt der Approximation beobachten. Dieser verschwindet, wenn sich die Nahzone über die komplette Geometrie erstreckt. In dem Fall bleibt lediglich eine Abweichung, die aus der iterativen Berechnung resultiert, zurück. Da die Anzahl der Boxen auch gleichzeitig die Anzahl der für die Approximation verwendeten Multipolreihen ist, wirkt sie sich deutlich auf die Abweichung aus. Die Approximation ist um so besser, je kleiner die einzelnen Boxen sind.

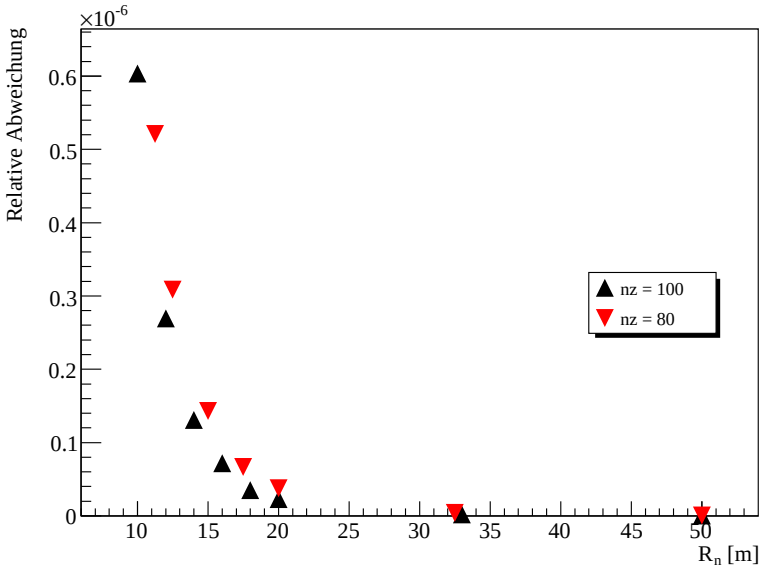

Abbildung 7.8: Mittlere relative Abweichung der Ladungsdichten in Abhängigkeit von R_n

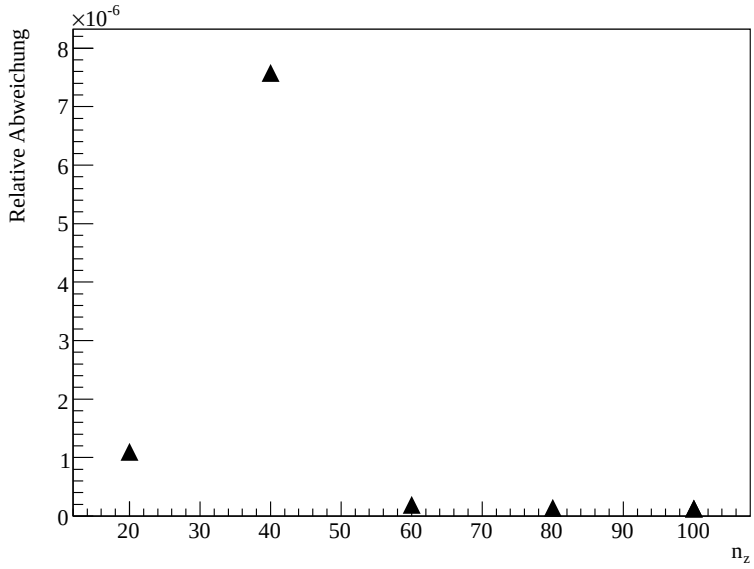

Abbildung 7.9: Mittlere relative Abweichung der Ladungsdichten in Abhängigkeit von n_z

Für die optimalen Nahzonengrößen ist eine Untersuchung der Abweichungen wenig aussagekräftig, da sie teilweise stark schwanken wegen den Schwierigkeiten beim Bestimmen des optimalen Wertes für R_n (Abbildung 7.9). Generell wird aber die obige Aussage bestätigt, dass die Abweichung für eine größere Anzahl an Boxen kleiner wird.

Tabelle 7.2: Vergleich von *Elcd3_3* und *Elcd4* mit einfacher Testgeometrie

	Elcd3_3		*Elcd4*		
scale	RAM [MB]	Zeit [s]	RAM [MB]	Zeit [s]	rel. Abweichung
1000	35,4	108	13,9	51	$1,30 \cdot 10^{-7}$
2000	130,1	651	53,8	134	$6,54 \cdot 10^{-8}$
3000	285,8	1953	117,7	270	$6,39 \cdot 10^{-7}$
4000	502,6	4347	206,9	459	$7,34 \cdot 10^{-8}$

Vergleicht man nun diese mit den bei *Elcd3_3* ermittelten Ergebnissen für die gleiche Geometrie, erhält man Tabelle 7.2. Dazu wurden die optimalen Ergebnisse für $n_z = 100$ und verschiedene Diskretisierungen verglichen. Die weiteren Ergebnisse befinden sich im Anhang in den Tabellen D.2, D.3 und D.4. Es zeigt sich, dass *Elcd4* für diese einfache Geometrie deutlich überlegen ist. Gerade bei einer hohen Elementanzahl bringt die iterative Berechnung einen deutlichen Zeitvorteil. Beim Speicherverbrauch erkennt man eine wesentliche Verbesserung aufgrund der Multipolapproximation. Man käme sogar mit noch weniger Speicher aus, wenn man die eingesparte Zeit dazu nutzen würde, Daten aus dem Speicher erst zu berechnen, wenn sie benötigt werden. Die Abweichung zwischen *Elcd4* und *Elcd3_3* liegt genau im angestrebtem Bereich.

Zusammenfassend kann man sagen, dass der neue Algorithmus funktioniert und bei der einfachen Testgeometrie deutlich bessere Leistungswerte erzielt als *Elcd3_3*. Die elektrischen Ladungsdichten lassen sich mit der erforderlichen Präzision in wesentlich kürzerer Zeit und mit deutlich weniger Speicherverbrauch berechnen. Die mittlere relative Abweichung der Ladungsdichten im Vergleich zu *Elcd3_3* hängt stark von den gewählten Parametern ab, aber der angestrebte Wert von 10^{-7} kann erreicht werden.

7.3 Tests mit KATRIN-Geometrien

Die Testrechnungen mit der oben genannten Testgeometrie haben gezeigt, dass *Elcd4* funktioniert und alle Programmkomponenten fehlerfrei zusammenarbeiten. Als nächstes muss überprüft werden, ob sich der neue Algorithmus auch auf realistische KATRIN-Geometrien anwenden lässt. Dazu wurden zunächst die Ladungsdichten für ein leeres Hauptspektrometer mit 1200facher Rotationssymmetrie ohne Drahtelektrode berechnet und mit den durch *Elcd3_3* bestimmten verglichen. Die Anzahl der Oberflächenelemente lässt sich bei dieser Elektrodenkonfiguration erneut über den *scale*-Paramater der Diskretisierung anpassen. Für die folgenden

Rechnungen wurde aber jeweils $scale = 5$ gesetzt. Das bedeutet, die gesamte Geometrie besteht aus 315 Oberflächenelementen. Es ist hierbei zu beachten, dass diese Elemente nur eine Hälfte des Spektrometers definieren. Bei der Berechnung wird die Spiegelsymmetrie an der Analysierebene ausgenutzt und jedes Element verdoppelt. Die Spiegelung wird während der Berechnung automatisch durchgeführt, so dass die Elemente nicht doppelt im Speicher vorliegen. Dieses Verfahren wird auch auf das Boxsystem angwendet. Jede Box existiert nur einmal im Speicher, wird aber zweifach ausgewertet.

Da diese Elektrodengeometrie mit $n_{rot} = 1200$ eine hohe Rotationssymmetrie besitzt, wird zuerst der Einfluss des Parameters n_φ und damit auch der Rotationssymmetrie des Boxsystems n_{Gitter} (6.46) untersucht. Dabei wird nur der Fall $n_\varphi < 0$ betrachtet, da die Geometrie ausschließlich aus diskretisierten Konen besteht. Der Fall $n_\varphi > 0$ würde nur zu weiteren leeren Boxen führen, die die Berechnung lediglich verlangsamen. Für die übrigen Parameter wurden $n_r = 5$, $n_z = 30$, $s = 20$, $C = 10^{-10}$ und $R_n = 9\,\mathrm{m}$ gewählt.

Tabelle 7.3: Auswirkung der Boxgruppierung

n_φ	Iterationszeit [s]	Speicherverbrauch [MB]	rel. Abweichung
-10	64,49	3,2	$1,33 \cdot 10^{-5}$
-20	34,23	3,1	$5,13 \cdot 10^{-5}$
-30	23,09	3,0	$1,05 \cdot 10^{-4}$
-40	16,33	3,0	$2,14 \cdot 10^{-4}$

Tabelle 7.3 zeigt die Resultate für verschiedene Werte für n_φ. Diese entsprechen genau den Erwartungen. Werden viele Boxen gruppiert, sinkt die Iterationszeit, da weniger Multipole berechnet werden müssen. Gleichzeitig wird aber auch die relative Abweichung der Ladungsdichten größer, da die Multipolapproximation schlechter wird. Allerdings ist die relative Abweichung allgemein sehr hoch. Bei den folgenden Rechnungen wurde mit $n_\varphi = -40$ gearbeitet. Im Anhang in Tabelle D.5 befinden sich auch Ergebnisse für andere Werte.

Tabelle 7.4: Einfluss der Boxanzahl

n_z	Iterationszeit [s]	Speicherverbrauch [MB]	rel. Abweichung
30	16,33	3,0	$2,14 \cdot 10^{-4}$
60	23,54	3,2	$3,43 \cdot 10^{-4}$
100	30,71	3,2	$1,43 \cdot 10^{-4}$

Da die Anzahl der Multipole offensichtlich einen sehr großen Einfluss hat, wird als nächstes die Auswirkung des Parameters n_z untersucht. Da er die Unterteilung des Boxsystems entlang der z-Achse vorgibt, ist er proportional zur Anzahl der verwendeten Multipole. Tabelle 7.4 zeigt die Ergebnisse unter Variation von n_z, alle übrigen Parameter wurden nicht verändert. Man kann erkennen, dass die Iterationszeit erneut mit der Anzahl der Multipole wächst. Aber gleichzeitig ist keine eindeutige

Verbesserung der Approximation zu erkennen, denn eine Erhöhung der Boxanzahl führt zum Teil nur zu leeren Boxen, die keinen Beitrag liefern. Daher wird im Folgenden weiterhin $n_z = 30$ gesetzt.

Tabelle 7.5: Einfluss der Nahzonengröße

	Rechenzeit [s]				
R_n [m]	Konstruktion	Iteration	Gesamt	RAM [MB]	rel. Abweichung
80,00	365,50	10,12	375,62	3,2	$3,34 \cdot 10^{-6}$
10,00	172,49	20,54	193,03	3,1	$7,21 \cdot 10^{-5}$
9,00	168,43	16,33	184,76	3,0	$2,14 \cdot 10^{-4}$
8,00	162,17	36,88	199,05	3,1	$3,12 \cdot 10^{-4}$
7,00	158,23	149,75	307,98	3,0	$5,75 \cdot 10^{-4}$

Der wichtigste Punkt, den es zu untersuchen gilt, ist die Auswirkung der Nahzonengröße. Diese wird durch den Parameter R_n bestimmt. Tabelle 7.5 listet die Resultate für verschiedenen Werte für R_n auf. Die Rechnung mit $R_n = 80\,\mathrm{m}$ dient zum Vergleich. Da die komplette Geometrie nach der Spiegelung nur eine Länge von 24 m besitzt, wird sie bei dieser Parameterwahl komplett als nah angesehen. Insgesamt lässt sich das gleiche Verhalten wie bei der Testgeometrie beobachten. Für die Rechenzeit gibt es einen optimalen Bereich bei $R_n = 9\,\mathrm{m}$ und die relative Abweichung ist umso geringer, je mehr von der Geometrie exakt berechnet wird. Beim Speicherverbrauch zeigt sich allerdings eine Schwierigkeit. Um ihn effektiv zu reduzieren, müsste die Geometrie mit einem deutlich kleineren Wert für R_n berechnet werden. Dann jedoch konvergiert das iterative Lösungsverfahren nicht mehr. Dies und die große relative Abweichung deuten darauf hin, dass die verwendete Multipolapproximation noch nicht ausreichend ist für diese Art von Geometrie. Auch durch eine Variation der Parameter n_r, n_φ und n_z kann sie nicht mehr verbessert werden. Da aber auch eine sehr große relative Abweichung für den Vergleichsfall $R_n = 80\,\mathrm{m}$ beobachtet wird, scheint es noch eine weitere Fehlerquelle zu geben. Es wäre möglich das die Fehler teilweise aus numerischen Artefakten der Fließkommaoperationen stammen können. Da an dieser Stelle nur die Ladungsdichten mit den von *Elcd3_3* ermittelten verglichen wurden, kann man auch nicht ausschließen, dass die Abweichung bereits in der Berechnung dort auftritt. Für eine klare Aussage über die Ursache sind somit noch weitere Untersuchungen nötig, die über diese Arbeit hinausgehen.

Somit ist es momentan noch nicht möglich, bei der Berechnung von realistischen KATRIN-Geometrien in vollem Unfang von den Neuerungen in *Elcd4* zu profitieren. Es ist aber ein Ausgangspunkt für zukünftige Entwicklungen und bietet eine optimale Struktur für verschiedene Erweiterungen. So kann mit wenig Aufwand die bisherige Multipolapproximation durch ein fortgeschritteneres Verfahren ersetzt werden. Mit dem hierarchischen Ansatz aus [Pur98] könnte beispielsweise die Näherung deutlich effizienter werden. Auch das bisher verwendete Boxsystem ist nicht optimal für KATRIN-Geometrien, da die starke Konzentration der Elektroden entlang der

Tankwand nicht ausgenutzt werden kann.

7.4 Eine verkleinerte Programmversion

Trotz des schlechten Konvergenzverhaltens bei realen KATRIN-Geometrien ist *Elcd4* dem bisherigen Feldberechnungsprogramm *Elcd3_3* dennoch in mehreren Bereichen überlegen. Wählt man daher die Parameter so, dass die komplette Elektrodengeometrie als nah angesehen wird, verliert man zwar jede Speicherersparnis, doch man umgeht so auch die Konvergenzprobleme. Es wird nun dieselbe Rechnung durchgeführt wie bei *Elcd3_3* mit dem Unterschied, dass die Ladungsdichten iterativ berechnet werden. Dies führt gerade bei einer großen Elementanzahl zu einer deutlichen Zeitersparnis. Insbesondere erhält man weiterhin die Möglichkeit, eine Elektrodengeometrie basierend auf bereits existierenden Ladungsdichten zu berechen. Dies sehr praktisch bei Designfragen und Toleranzrechnungen.

Dadurch, dass sich in diesem Fall die Nahzone über die vollständige Geometrie erstreckt, werden große Teile des Programms überflüssig. So wird kein Boxsystem benötigt, da man die komplette Geometrie als eine große Box auffassen kann. Ebenso werden die Routinen zur Multipolberechnung nicht mehr verwendet. Daher ist es sinnvoll, diese Teile aus dem Programm zu entfernen. Die obsoleten Komponenten verbrauchen so keinen Speicherplatz und keine Rechenzeit mehr. Dies ist in der verkleinerten Programmversion *Elcdlight* umgesetzt. Da die Unterstützung für das Boxsystem ein zentraler Bestandteil des gesamten Programms war, mussten bei der Vereinfachung große Teile neu strukturiert werden. Allerdings sind auch die zentralen Funktionen wesentlich einfacher und kürzer geworden.

Auch für den Anwender fallen so die meisten Parameter einfach weg. Von den oben genannten bleibt lediglich C bestehen, alle übrigen bezogen sich auf das nun entfernte Boxsystem.

Mit dieser verkleinerten Programmversion wurden verschiedene Testrechnungen zum Vergleich mit *Elcd3_3* durchgeführt. Dabei wurde wieder die exakte Version von *Elcd3_3* ohne Näherungen verwendet. Für sämtliche Testrechnungen wurde der Konvergenzparameter $C = 10^{-10}$ gesetzt.

Tabelle 7.6: Vergleich von *Elcd3_3* und *Elcdlight* mit einfacher Testgeometrie

	Elcd3_3		*Elcdlight*		
scale	RAM [MB]	Zeit [s]	RAM [MB]	Zeit [s]	rel. Abweichung
1000	35,4	108	32,8	48	$6,89 \cdot 10^{-10}$
2000	130,1	651	124,8	199	$2,19 \cdot 10^{-9}$
3000	285,8	1953	277,9	452	$5,99 \cdot 10^{-9}$
4000	502,6	4347	492,1	828	$8,04 \cdot 10^{-9}$

Zunächst wurden die Ergebnisse beider Programme anhand der oben genannten Testgeometrie aus einem langen Zylinder mit eingeschlossenen Drähten verglichen. Die Elementanzahl wurde wieder über den *scale*-Parameter variiert, sie beträgt für

diese Geometrie $N = 2 \cdot scale$. Die Resultate dieser Rechnung sind in Tabelle 7.6 aufgeführt. Man kann eine leichte Abnahme des Speicherverbrauchs erkennen. Dies ist zurückzuführen auf eine effizientere Speichernutzung der neuen Implementierung. Deutlicher ist der Unterschied in der Rechenzeit. Da die Elementanzahl in der Testgeometrie sehr hoch ist, ist das iterative Verfahren wesentlich besser für die Lösung des Gleichungssystems geeignet. Dabei liegt die Abweichung zwischen beiden Programmen genau im vorgebenen Rahmen.

Tabelle 7.7: Vergleich von *Elcd3_3* und *Elcdlight* mit leerem Spektrometertank

	Elcd3_3		*Elcdlight*		
scale	RAM [MB]	Zeit [s]	RAM [MB]	Zeit [s]	rel. Abweichung
5	89,1	516	2,5	364	$3,26 \cdot 10^{-6}$
10	178,0	2052	4,9	1455	$4,72 \cdot 10^{-5}$
15	268,5	4628	8,9	3277	$8,24 \cdot 10^{-5}$

Als nächstes wurden die Ladungsdichten für ein leeres Hauptspektrometer mit 1200facher Rotationssymmetrie ohne Drahtelektrode mit beiden Programmen berechnet. Dabei lässt sich erneut die Anzahl der Elemente über den *scale*-Paramater der Diskretisierung anpassen. Tabelle 7.7 zeigt die Resultate dieses Vergleichs. Am auffälligsten ist hier der deutliche Unterschied im Speicherverbrauch. Der Grund dafür liegt in der Rotationssymmetrie. In *Elcd3_3* werden die Elektroden nach dem Einlesen gemäß ihrer Rotationssymmetrie vervielfacht und entsprechend rotiert. Dieser Schritt kann bei *Elcdlight* wegfallen, denn aufgrund der neuen Programmstruktur sind Rotationen sehr einfach zu realisieren und ihre Berechnung sehr effizient. Die daraus resultierende zusätzliche Rechenzeit ist vernachlässigbar im Vergleich zur Zeit, die für eine einzelne Potentialberechnung benötigt wird. Daher lässt sich der für die Rechnung benötigte Speicher deutlich reduzieren. Dieser Effekt wird umso größer, je höher die Rotationssymmetrie der Geometrie ist. Er hat allerdings keinen Einfluss auf die Größe des Arbeitsspeichers, der vom Gleichungssystem der Boundary Element Methode belegt wird. Diese hängt nur von der Anzahl der Elemente ab und ist unabhängig von der Rotationssymmetrie. Neben der Speicherersparnis erkennt man auch eine leicht verkürzte Rechenzeit. Dies wurde erreicht durch eine Optimierung der Funktionen zur Potentialberechnung. Etwas störend wirkt allerdings die ungewöhnlich hohe relative Abweichung. Sie deckt sich aber mit der in Abschnitt 7.3 beobachteten mit derselben Geometrie. Diese deutet auf einen Fehler in Berechnung der Rechteckpotentiale hin, den es noch zu untersuchen gilt.

Abschließend folgte die wichtigste Vergleichsrechnung. Als Geometrie wurde hier eine von *MainSpec* generierte Datei verwendet. Das Besondere dabei ist, dass es sich in diesem Fall um eine Geometrie handelt, welche tatsächlich für die Entwicklung eines elektromagnetischen Designs berechnet werden muss. Als Parameter wurden bei *MainSpec* die Vorgabewerte benutzt, das bedeutet die Geometrie entspricht dem aktuellen Modell der realen KATRIN-Geometrie einschließlich diskretisierten Kämmen und Drähten der Drahtelektrode. Lediglich der *scale*-Parameter für die

Tabelle 7.8: Vergleich von *Elcd3_3* und *Elcdlight* mit vollständigem Spektrometer

scale		*Elcd3_3*		*Elcdlight*		
Drähte	Tank	RAM [MB]	Zeit [s]	RAM [MB]	Zeit [s]	rel. Abweichung
15	1	163,7	1512	5,5	1123	$2,09 \cdot 10^{-7}$
30	1	210,5	2044	8,0	1599	$1,35 \cdot 10^{-7}$
45	1	257,8	2609	11,4	2144	$1,27 \cdot 10^{-7}$
15	5	214,1	2671	7,6	1999	$1,63 \cdot 10^{-6}$
30	5	261,5	3408	10,8	2680	$3,00 \cdot 10^{-7}$
45	5	309,5	4192	14,6	3442	$2,37 \cdot 10^{-7}$

Draht- und Tankdiskretisierung wurde variiert, um verschiedene Elementanzahlen zu erhalten. Die Ergebnisse dieses Vergleichs sind in Tabelle 7.8 zusammengefasst. Man kann dieselben Effekte wie bei der Berechnung des leeren Tanks beobachten, da die Rotationssymmetrie der Geometrie ebenfalls 1200 beträgt. Da nun die relative Abweichung wird im angestrebten Bereich liegt, scheint der vorher beobachtete Fehler ausschließlich bei einem leeren Spektrometer aufzutreten.

Abschließend lässt sich sagen, dass *Elcdlight* einen Ersatz für *Elcd3_3* darstellt. Es arbeitet für sämtliche getesten Geometrien schneller und erfordert dabei weniger Arbeitsspeicher. Auch die relative Abweichung der Ladungsdichten liegt weitestgehend im erfordlichen Rahmen, wenn von dem Fall des leeren Spektrometers absieht. Man kann zwei Stärken von *Elcdlight* besonders hervorheben. Bei hohen Rotationssymmetrien wird der Speicherverbrauch deutlich reduziert durch die effizienteren Geometrieoperationen, und bei hohen Elementanzahlen wird die Rechenzeit durch das iterative Lösungsverfahren massiv verkürzt.

Negativ fällt dagegen die teilweise recht große Abweichnung in den Ladungsdichten auf. Sie lässt sich nicht durch eine Änderung des Parameters C reduzieren. Die Quelle des Fehlers scheint die Berechnung von Rechteckpotentialen zu sein. Man kann auch nicht ausschließen, dass sie aus der großen Zahl von Fließkomma-Operationen resultieren und bereits in *Elcd3_3* entstehen. Dies ließe sich durch Vergleiche mit analytisch berechenbaren Geometrien untersuchen. Eventuell sollte auch ein Einsatz des SSE-Befehlssatzes (Internet Streaming SIMD Extensions) für Fließkomma-Arithmetik erwogen werden, was mit neueren Prozessoren möglich ist. Solche Untersuchungen waren leider nicht mehr im Rahmen dieser Diplomarbeit machbar.

Der Einsatz von SSE würde auch weitergehende Optimierungen erlauben. Generell bezeichnet SSE eine von der Firma Intel entwickelte Befehlssatzerweiterung der x86er-Prozessoren. Sie wurde mit dem Pentium III erstmalig eingeführt. Sie erlaubt die parallele Ausführung von Fließkomma-Operationen, zunächst nur mit einfacher und ab dem Pentium 4 auch mit doppelter Genauigkeit (SSE2). Würde man die Programme entsprechend anpassen, könnte die erfordliche Rechenzeit nahezu halbiert werden. *Elcd4* und *Elcdlight* wurden während der Entwicklung bereits so struktiert, das solche Optimierungen nachträglich hinzugefügt werden könnten.

Andere sinnvolle Optimierungen wären noch in der Potentialberechnung möglich.

Elcd3_3 betrachtet dabei weit entfernte Elemente als Punktladungen, aber für die Vergleichsrechnungen wurde diese Näherung deaktiviert. Man könnte jedoch eine ähnliche Approximation in *Elcdlight* integrieren und so die Rechenzeit weiter reduzieren.

8 Zusammenfassung

Für eine präzise Auswertung der Messdaten des KATRIN-Experiments ist eine exakte Kenntnis der Flugbahnen geladener Teilchen im Aufbau des Experiments essentiell. Da dabei viele Komponenten zusammenspielen, lassen sich die Flugbahnen nur durch eine globale Bahnverfolgung untersuchen. Dabei wird die Bahn eines Teilchens durch den gesamten Experimentaufbau berechnet. Im Rahmen dieser Diplomarbeit wurden die existierenden Programme für die Entwicklung des elektromagnetischen Designs erweitert, so dass sie für diese Art von Simulationen verwendet werden können. Zusätzlich wurden in Münster am Institut für Kernphysik zusätzliche Computer aufgebaut, die in Zukunft für globale Bahnverfolgung genutzt werden können.

Zuvor muss noch ein möglichst detailgetreues Computermodell der elektromagnetischen Komponenten des KATRIN-Experiments erstellt werden. Zu diesem Zweck wurde ein grafisches Programm entwickelt, das basierend auf verschiedenen Parametern automatisch ein Modell der Drahtelektrode des Hauptspektrometers erstellt. In verschiedenen Simulationen werden diese Parameter nun optimiert. Auf diese Weise wird das elektromagnetische Design weiter verbessert mit dem Ziel die optimale Form der Drahtelektrode zu finden [Hug08]. Als Hilfsmittel für diese Untersuchungen wurde im Rahmen der vorliegenden Arbeit eine Software geschrieben, die eine Elektrodengeometrie dreidimensional darstellt. Damit können verschiedene Geometriemodelle betrachtet und eventuelle Schwachstellen ermittelt werden.

Durch die steigende Detailgenauigkeit der verwenden Geometrien stoßen die eingesetzten Feldberechnungsprogramme zunehmend an ihre Grenzen. Um die von allen Elektroden erzeugten elektrischen Potentiale zu berechnen, muss ein neuer Algorithmus benutzt werden. Zu diesem Zweck wurde die bisher verwendete Boundary Element Methode durch eine Multipolapproximation erweitert. Es wird nur noch ein kleiner Teil der gesamten Geometrie exakt berechnet, während alle übrigen Elektroden lediglich näherungsweise in die Berechnung einfließen. Dieses Verfahren hat sich für mehrere Testgeometrien als schneller und speichereffizienter erwiesen, während die Rechengenauigkeit im erforderlichen Bereich liegt. Für einen Einsatz bei der Berechnung von KATRIN-Geometrien muss noch die Multipolapproximation verbessert werden, da die bisherige Lösung hier zu relativ großen Abweichungen und einem schlechten Konvergenzverhalten führt. Verzichtet man auf diese Approximation, so lassen sich bereits jetzt für alle getesteten Geometrien bessere Resultate erzielen. Zusätzlich stellt die neu programmierte Software eine Basis für zukünftige Entwicklungen dar. Durch seine Struktur lässt es sich optimal in externe Pakete wie ROOT oder GEANT4 einbinden. Gleichzeitig wurde es aber auch so entworfen, dass die Kompatibilität zu der bisher verwendeten Software erhalten bleibt.

A Vollständige Berechnungen

A.1 Multipolmomente

Rechteckige Elektrode

Eine rechteckige Elektrode wird definiert durch den Eckpunkt $\vec{p}$, zwei orthogonalen Einheitsvektoren $\vec{n}_1$ und $\vec{n}_2$ und den beiden Seitenlängen a und b. Zusätzlich wird für die Multipolberechnung ein Referenzpunkt $\vec{c}$ benötigt, bezüglich dem die Multipolmomente berechnet werden sollen.

Zunächst nimmt man zur Vereinfachung eine Variablenumbennung vor:

$$\vec{p} - \vec{c} \to \vec{p}. \tag{A.1}$$

Dies verschiebt den Ursprung des Koordinatensystems in den Referenzpunkt.

Bei der Berechnung der Multipolmomente wird über die komplette Ladungsdichte integriert. Für diesen Fall kann man die Integrale umschreiben als Flächenintegral über die Rechteckfläche mit einer konstanten Ladungsdichte:

$$\int \mathrm{d}^3 r' \varrho(\vec{r}') \to \sigma \int_0^a \mathrm{d}a' \int_0^b \mathrm{d}b'. \tag{A.2}$$

Einen Punkt auf dem Rechteck $\vec{r}$ können wir schreiben als

$$\vec{r} = \vec{p} + a'\vec{n}_1 + b'\vec{n}_2. \tag{A.3}$$

Führt man diese Ersetzung bei der Monopolberechnung (6.10), erhält man

$$q = \sigma \int_0^a \mathrm{d}a' \int_0^b \mathrm{d}b' = \sigma ab. \tag{A.4}$$

Für das Dipolmoment (6.12) erhält man

$$\begin{aligned} \vec{d} &= \sigma \int_0^a \mathrm{d}a' \int_0^b \mathrm{d}b' \left(\vec{p} + a'\vec{n}_1 + b'\vec{n}_2\right) \\ &= \sigma ab \left(\vec{p} + \tfrac{1}{2}a\vec{n}_1 + \tfrac{1}{2}\vec{n}_2\right) \\ &= q \left(\vec{p} + \tfrac{1}{2}a\vec{n}_1 + \tfrac{1}{2}\vec{n}_2\right). \end{aligned} \tag{A.5}$$

Die Berechnung des Quadrupoltensors nach (6.14) ist sehr unübersichtlich, daher werden die beiden Terme einzeln berechnet:

$$Q_{ij} = \sigma \left(3Q^1_{ij} - \delta_{ij} Q^2_{ij}\right). \tag{A.6}$$

Für die beiden Terme gilt:

$$\begin{aligned} Q^1_{ij} &= \int_0^a \mathrm{d}a' \int_0^b \mathrm{d}b' (p_i + a' n_{1i} + b' n_{2i})(p_j + a' n_{1j} + b' n_{2j}) \\ &= \int_0^a \mathrm{d}a' \int_0^b \mathrm{d}b' \big(p_i p_j + a'(p_i n_{1j} + p_j n_{1i}) + b'(p_i n_{2j} + p_j n_{2i}) \\ &\qquad + a'b'(n_{1i} n_{2j} + n_{1j} n_{2i}) + a'^2 n_{1i} n_{1j} + b'^2 n_{2i} n_{2j}\big) \\ &= ab\Big(p_i p_j + \frac{1}{2}\big(a\,(p_i n_{1j} + p_j n_{1i}) + b\,(p_i n_{2j} + p_j n_{2i})\big) \\ &+ \frac{1}{4} ab\,(n_{1i} n_{2j} + n_{1j} n_{2i}) + \frac{1}{3}(a^2 n_{1i} n_{1j} + b^2 n_{2i} n_{2j})\Big) \end{aligned} \tag{A.7}$$

$$\begin{aligned} Q^2_{ij} &= \int_0^a \mathrm{d}a' \int_0^b \mathrm{d}b' (\vec{p} + a'\vec{n}_1 + b'\vec{n}_2)^2 \\ &= \int_0^a \mathrm{d}a' \int_0^b \mathrm{d}b' (p^2 + 2a'\vec{p}\cdot\vec{n}_1 + 2b'\vec{p}\cdot\vec{n}_2 + a' \underbrace{\vec{n}_1 \cdot \vec{n}_2}_{=0}) \\ &= ab\,(p^2 + a\,\vec{p}\cdot\vec{n}_1 + b\,\vec{p}\cdot\vec{n}_2). \end{aligned} \tag{A.8}$$

Somit gilt für den Quadrupoltensor $\hat{Q}$ einer rechteckigen Elektrode

$$\begin{aligned} Q_{ij} &= ab\Big(p_i p_j + \frac{1}{2}\big(a\,(p_i n_{1j} + p_j n_{1i}) + b\,(p_i n_{2j} + p_j n_{2i})\big) \\ &+ \frac{1}{4} ab\,(n_{1i} n_{2j} + n_{1j} n_{2i}) + \frac{1}{3}(a^2 n_{1i} n_{1j} + b^2 n_{2i} n_{2j}) \\ &- \delta_{ij}(p^2 + a\,\vec{p}\cdot\vec{n}_1 + b\,\vec{p}\cdot\vec{n}_2)\Big). \end{aligned} \tag{A.9}$$

Drahtelektrode

Eine Drahtelektrode wird über die Endpunkte $\vec{a}$, $\vec{b}$ und dem Durchmesser d definiert. Geometrisch betrachtet handelt es sich um einen Zylinder. Um die Multipolmomente zu berechnen, muss man über seine Oberfläche integrieren. Dazu wird zunächst der Vektor $\vec{s}$ definiert durch

$$\vec{s} = \vec{b} - \vec{a}. \tag{A.10}$$

Für die Integration wird ein neues Koordinationsystem definiert:

$$\vec{n}_1 = \begin{pmatrix} -\sin\varphi_s \\ \cos\varphi_s \\ 0 \end{pmatrix} \tag{A.11}$$

$$\vec{n}_2 = \begin{pmatrix} -\cos\vartheta_s\cos\varphi_s \\ -\cos\vartheta_s\sin\varphi_s \\ \sin\vartheta_s \end{pmatrix} \tag{A.12}$$

$$\vec{n}_3 = \frac{\vec{s}}{s}. \tag{A.13}$$

$(s, \vartheta_s, \varphi_s)$ ist hierbei die Darstellung von $\vec{s}$ in Kugelkoordinaten.

Die Stirnflächen des Zylinders werden bei dieser Rechnung ignoriert, da für einen Draht die Bedingung $s \gg d$ immer erfüllt ist. So kann jeder Punkt auf der Oberfläche des Drahtes $\vec{r}$ durch die Parameter x und φ beschrieben werden:

$$\vec{r} = \vec{a} + \frac{d}{2}\left(\vec{n}_1\cos\varphi + \vec{n}_2\sin\varphi\right) + \vec{n}_3 x. \tag{A.14}$$

Das Integral aus (6.8) lässt sich mit diesen Parametern umschreiben zu

$$\int d^3r'\varrho(\vec{r}\,') \to \sigma\frac{d}{2}\int_0^{2\pi} d\varphi \int_0^{s} dx. \tag{A.15}$$

σ ist die Oberflächenladungsdichte die auf dem kompletten Draht als konstant angenommen ist. Daher kann sie vor das Integral gezogen werden.

Setzt man dies in (6.10) ein, erhält man das Monopolmoment des Drahtes q:

$$q = \sigma\frac{d}{2}\int_0^{2\pi} \mathrm{d}\varphi \int_0^{s} \mathrm{d}x = \pi d s\sigma. \tag{A.16}$$

Einsetzen in (6.12) liefert das Dipolmoment

$$\begin{aligned} \vec{d} &= \sigma\frac{d}{2}\int_0^{2\pi}\mathrm{d}\varphi\int_0^{s}\mathrm{d}x\left(\vec{a} + \frac{d}{2}\left(\vec{n}_1\cos\varphi + \vec{n}_2\sin\varphi\right) + \vec{n}_3 x\right) \\ &= \pi d s\sigma\left(\vec{a} + \frac{1}{2}s\vec{n}_1\right) \\ &= \frac{1}{2}\left(\vec{a} + \vec{b}\right)q. \end{aligned} \tag{A.17}$$

Die Berechnung des Quadrupoltensors erfolgt wieder termweise:

$$Q_{ij} = \sigma\frac{d}{2}\left(3Q^1_{ij} - \delta_{ij}Q^2_{ij}\right). \tag{A.18}$$

Für die beiden Terme gilt:

$$
\begin{aligned}
Q_{ij}^{1} &= \int_0^{2\pi} \mathrm{d}\varphi \int_0^{s} \mathrm{d}x \; r_i r_j \\
&= \int_0^{2\pi} \mathrm{d}\varphi \int_0^{s} \mathrm{d}x \Bigg(a_i a_j + x(a_i n_{3j} + a_j n_{3i}) + x^2 n_{3i} n_{3j} \\
&\qquad + \left(\frac{d}{2}(a_i n_{1j} + a_j n_{1i}) + \frac{d^2}{4} x (n_{1j} n_{3i} + n_{1i} n_{3j}) \right) \cos\varphi \\
&\qquad + \left(\frac{d}{2}(a_i n_{2j} + a_j n_{2i}) + \frac{d^2}{4} x (n_{2i} n_{3j} + n_{2j} n_{3i}) \right) \sin\varphi \\
&\qquad + \frac{d^2}{4}(n_{1i} n_{2j} + n_{1j} n_{2i}) \sin\varphi \cos\varphi \\
&\qquad + \frac{d^2}{4} \left(n_{1i} n_{1j} \cos^2\varphi + n_{2i} n_{2j} \sin^2\varphi \right) \Bigg) \\
&= 2\pi \int_0^{s} \mathrm{d}x \left(a_i a_j + x(a_i n_{3j} + a_j n_{3i}) + x^2 n_{3i} n_{3j} + \frac{d^2}{8}(n_{1i} n_{1j} + n_{2i} n_{2j}) \right) \\
&= 2\pi s \left(a_i a_j + \frac{1}{2} s (a_i n_{3j} + a_j n_{3i}) + \frac{1}{3} s^2 n_{3i} n_{3j} + \frac{d^2}{8}(n_{1i} n_{1j} + n_{2i} n_{2j}) \right) \\
&= 2\pi s \left(a_i a_j + \frac{1}{2}(a_i s_j + a_j s_i) + \frac{1}{3} s_i s_j + \frac{d^2}{8}(n_{1i} n_{1j} + n_{2i} n_{2j}) \right)
\end{aligned}
\tag{A.19}
$$

$$
\begin{aligned}
Q_{ij}^2 &= \int_0^{2\pi} \mathrm{d}\varphi \int_0^s \mathrm{d}x\, r^2 \\
&= \int_0^{2\pi} \mathrm{d}\varphi \int_0^s \mathrm{d}x \left(\vec{a} + \frac{d}{2} \left(\vec{n}_1 \cos\varphi + \vec{n}_2 \sin\varphi \right) + \vec{n}_3 x \right)^2 \\
&= \int_0^{2\pi} \mathrm{d}\varphi \int_0^s \mathrm{d}x \big(a^2 + 2x\vec{a}\cdot\vec{n}_3 + x^2 \underbrace{n_3^2}_{=1} \\
&\qquad + d(\vec{a}\cdot\vec{n}_1 + x\underbrace{\vec{n}_1\cdot\vec{n}_3}_{=0})\cos\varphi \\
&\qquad + d(\vec{a}\cdot\vec{n}_2 + x\underbrace{\vec{n}_2\cdot\vec{n}_3}_{=0})\sin\varphi \\
&\qquad + \frac{d^2}{2}\underbrace{\vec{n}_1\cdot\vec{n}_2}_{=0}\sin\varphi\cos\varphi \\
&\qquad + \frac{d^2}{4}\underbrace{(\vec{n}_1^2\cos^2\varphi + \vec{n}_2^2\sin^2\varphi)}_{=1} \big) \\
&= \int_0^{2\pi} \mathrm{d}\varphi \int_0^s \mathrm{d}x \left(a^2 + 2x\vec{a}\cdot\vec{n}_3 + x^2 + d\vec{a}\cdot\vec{n}_1\cos\varphi + d\vec{a}\cdot\vec{n}_2\sin\varphi + \frac{d^2}{4} \right) \\
&= 2\pi \int_0^s \mathrm{d}x \left(a^2 + 2x\vec{a}\cdot\vec{n}_3 + x^2 + \frac{d^2}{4} \right) \\
&= 2\pi s \left(a^2 + s\vec{a}\cdot\vec{n}_3 + \frac{1}{3}s^2 + \frac{d^2}{4} \right) \\
&= 2\pi s \left(a^2 + \vec{a}\cdot\vec{s} + \frac{1}{3}s^2 + \frac{d^2}{4} \right).
\end{aligned}
\tag{A.20}
$$

Insgesamt gilt für das Quadrupolmoment eines Drahtes

$$
\begin{aligned}
Q_{ij} = 6\pi s &\left(a_i a_j + \frac{1}{2}(a_i s_j + a_j s_i) + \frac{1}{3} s_i s_j + \frac{d^2}{8} A_{ij} \right) \\
&- 2\pi s \left(a^2 + \vec{a}\cdot\vec{s} + \frac{1}{3}s^2 + \frac{d^2}{4} \right) \delta_{ij}.
\end{aligned}
\tag{A.21}
$$

Dabei wurde die folgende Abkürzung benutzt

$$A_{ij} = n_{1i}n_{1j} + n_{2i}n_{2j} \tag{A.22}$$

$$A_{11} = \sin^2 \varphi_s + \cos^2 \vartheta_s \cos^2 \varphi_s \tag{A.23}$$

$$A_{22} = \cos^2 \varphi_s + \cos^2 \vartheta_s \sin^2 \varphi_s \tag{A.24}$$

$$A_{33} = \sin^2 \vartheta_s \tag{A.25}$$

$$A_{12} = -\sin \varphi_s \cos \varphi_s + \cos^2 \vartheta_s \cos \varphi_s \sin \varphi_s = -\sin^2 \vartheta_s \cos \varphi_s \sin \varphi_s \tag{A.26}$$

$$A_{13} = -\cos \vartheta_s \sin \vartheta_s \cos \varphi_s \tag{A.27}$$

$$A_{23} = -\cos \vartheta_s \sin \vartheta_s \sin \varphi_s \tag{A.28}$$

B Anleitungen

B.1 Anleitung zu den Elcd4-Programmen

Elmain4

Elmain4 ist ein Kommandozeilenprogramm, mit dem man zu einer Elektrodengeometrie die Ladungsdichten bestimmen kann. Die Elektroden müssen in der EDF-Datei vorliegen und die Ladungsdichten werden in Form einer CDF-Datei ausgegeben. Die Syntax für den Programmaufruf ist

```
elmain4 [Optionen] <EDF-Datei>.
```

Dabei werden die folgenden Optionen erkannt:

--nr=N	Setze den Parameter n_r auf N
-z N, --nz=N	Setze den Parameter n_z auf N
-p N, --nphi=N	Setze den Parameter n_φ auf N
-r N, --range=N	Setze R_n auf N
-s N, --steps=N	Setze den Anfangswert für *steps* auf N
-v, --version	Gebe die aktuelle Programmversion aus
-h, --help	Gebe einen kurzen Hilfstext aus.

Die CDF-Datei erhält denselben Dateinamen wie die EDF-Datei, lediglich die Dateiendung wird in *.cdf* umgewandelt.

Elcheck4

Elcheck4 ist ebenfalls ein Kommandozeilenprogramm. Es dient dazu aus einer CDF-Datei elektrische Potentiale zu bestimmen. Die Syntax lautet:

```
elcheck4 <CDF-Datei> <Koordinaten-Datei>
```

Die Koordinaten-Datei ist eine ASCII-Datei, in der zeilenweise Koordinaten aufgeführt sind, an denen das Potential bestimmt werden soll. Die Potentiale werden dann auf der Standardausgabe ausgeben.

Elcdlight

Elcdlight ist sehr ähnlich wie *Elmain4*. Die Bedienung ist allerdings wesentlich einfacher geworden, da keine Paramter mehr angeben werden müssen. Die Syntax ist daher reduziert auf

```
elcdlight <EDF-Datei>.
```

C Header-Dateien

box.h

```
/* $Id: box.h 289 2007-12-25 09:53:59Z s_voec01 $ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
 *
 * Last changes:
 * $Author: s_voec01 $
 * $Date: 2007-12-25 10:53:59 +0100 (Di, 25 Dez 2007) $
 */

#ifndef BOX_H_
#define BOX_H_

typedef struct _Box Box;
typedef struct _BoxElement BoxElement;

#include "electrode.h"
#include "list.h"

struct _BoxElement
{
    Electrode* electrode;
    double monopol;
    double dipol[3];
#ifndef NO_QUADRUPOL
    double quadrupol[5];
#endif
#ifndef NO_OCTOPOLE
    double octopole[10];
#endif
};

struct _Box
{
    int num;
```

```
    int group;
    const double* matrix;
    BoxElement* elements;
    double center[3];
    double monopol;
    double dipol[3];
#ifndef NO_QUADRUPOL
    double quadrupol[5];
#endif
#ifndef NO_OCTOPOLE
    double octopole[10];
#endif
};

Box* box_new(double* center, int group, const double* matrix,
             List* electrodes);
void box_free(Box* box);
double box_potential(const Box* box, double* x);
void box_update_multipoles(Box* box);

#endif /*BOX_H_*/
```

Mit den Macros *NO_QUADRUPOL* und *NO_OCTOPOLE* können während der Kompilierung die verwendeten Multiordnungen gesteuert werden. Allerdings ist zu beachten das die Oktopol-Ordnung bisher nur für rechteckige Elemente implementiert ist. Als Vorgabe sind Oktopole deaktiviert.

cdf.h

```
/* $Id$ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
 *
 * Last changes:
 * $Author$
 * $Date$
 */

#ifndef CDF_H_
#define CDF_H_

#include "electrode.h"

enum CDFMode {
```

```
    MODE_READ,
    MODE_WRITE
};

typedef struct _CDF CDF;

CDF* cdf_new(const char* filename, enum CDFMode mode);
void cdf_close(CDF* cdf);

void cdf_add_electrode(CDF* cdf, const Electrode* electrode);
unsigned long cdf_get_electrodes(CDF* cdf, Electrode*** electrodes);

#endif /*CDF_H_*/
```

edf.h

```
/* $Id: edf.h 289 2007-12-25 09:53:59Z s_voec01 $ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
 *
 * Last changes:
 * $Author: s_voec01 $
 * $Date: 2007-12-25 10:53:59 +0100 (Di, 25 Dez 2007) $
 */

#ifndef EDF_H
#define EDF_H

#include "electrode.h"

long edf_read(const char* filename, Electrode*** electrodes,
              unsigned short* nrot);

#endif
```

elcd4.h

```
/* $Id: elcd4.h 272 2007-08-22 13:46:58Z s_voec01 $ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
 *
 * Last changes:
```

```
 * $Author: s_voec01 $
 * $Date: 2007-08-22 15:46:58 +0200 (Mi, 22 Aug 2007) $
 */

/*! @file
 * @brief Main module of elcd4
 *
 * This modules contains the function to use elcd4 from external code.
 */

#ifndef ELCD4_H
#define ELCD4_H

/*! @brief Opaque type containing the status of the calculation
 *
 * Use this type to do calculations with Elcd4. All members of this struct are
 * private. Any access should be performed with the following functions.
 */
typedef struct _Elcd Elcd;

/*! @brief Create a new Elcd object
 *
 * Return an empty Elcd object. You have to free it with elcd_free.
 * @param nr Number of grid boxes in r direction
 * @param nphi Number of grid boxes in phi direction
 * @param nz Number of grid boxes in z direction
 */
Elcd* elcd_new(int nr, int nphi, int nz);
/*! @brief Create a new Elcd object from a CDF
 *
 * Return an Elcd object containing the charge densities from a CDF. You have
 * to free it with elcd_free.
 * @param filename CDF which should be read
 */
Elcd* elcd_new_from_cdf(const char* filename);
/*! @brief Destroy an Elcd object
 *
 * Free an Elcd object and all the memory allocated to it.
 * @param elcd A valid Elcd object
 */
void elcd_free(Elcd* elcd);

/*! @brief Set near range
 *
```

```
 * Set the distance within electrodes are considered as "near".
 * @param elcd A valid Elcd object
 * @param near The near range
 */
void elcd_set_near(Elcd* elcd, double near);
/*! @brief Control geometry mirroring
 *
 * Control whether the geometry should be mirrored or not. And set the
 * mirroring plane.
 * @param elcd A valid Elcd object
 * @param mirror 1 to enable, 0 to disable mirroring
 * @param zmirror Z coordinate of the mirroring plane which will be used
 */
void elcd_set_mirror(Elcd* elcd, int mirror, double zmirror);

/*! @brief Read the geometry from an EDF
 *
 * Read the electrodes from an EDF. Existing electrodes will be replaced. The
 * near zones are automatically calculated with the current near range.
 * @param elcd A valid Elcd object
 * @param filename Name of the EDF which should be read
 */
long elcd_read_edf(Elcd* elcd, const char* filename);
/*! @brief Write charge densities into an CDF
 *
 * Write electrode data including the calculated charge densities into an
 * binary CDF.
 * @param elcd A valid Elcd object
 * @param filename Name of the CDF which should be written
 */
void elcd_write_cdf(Elcd* elcd, const char* filename);
void elcd_write_sigmas(Elcd* elcd, const char* filename);

/*! @brief Initialise the iteration
 *
 * Initialise the iteration and set the number of Gauss Seidel steps.
 * @param elcd A valid Elcd object
 * @param steps Number of Gauss Seidel steps
 */
void elcd_iteration_start(Elcd* elcd, int steps);
/*! @brief Perform one iteration step
 *
 * Perform exactly one iteration step and return the current convergence. Use
 * this function to monitor the progress of the iteration
```

```
 * @param elcd A valid Elcd object
 */
double elcd_iteration_step(Elcd* elcd);
/*! @brief Iterate to a specific convergence
 *
 * Iterate until the given convergence is reached
 * @param elcd A valid Elcd object
 * @param convergence Desired convergence
 */
void elcd_iteration(Elcd* elcd, double convergence);

/*! @brief Get the potential at a point
 *
 * Return the electric potential at a given point.
 * @param elcd A valid Elcd object
 * @param x Vector of the point for which the potential is needed
 */
double elcd_potential(Elcd* elcd, const double* x);
double elcd_get_last_factor(Elcd* elcd);

#endif
```

Die mit */*!* eingeleiteten Kommentare sind sogenannte *Doxygen*-Kommentare. Aus ihnen kann mithilfe des Programms *Doxygen* eine vollständige Dokumentation der Programmierschnittstelle (API) von *Elcd4* im HTML-Format generiert werden.

elcd4mm.h

```
/* $Id: elcd4mm.h 249 2007-08-15 10:08:34Z s_voec01 $ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
 *
 * Last changes:
 * $Author: s_voec01 $
 * $Date: 2007-08-15 12:08:34 +0200 (Mi, 15 Aug 2007) $
 */

#ifndef ELCD4MM_H_
#define ELCD4MM_H_

#include <string>

namespace C {
```

```
    typedef struct _Elcd Elcd;
}

class Elcd
{
    public:
        Elcd(int nr, int nphi, int nz);
        Elcd(const std::string& filename);
        ~Elcd();

        void set_near(int near);

        long read_edf(const std::string& filename);
        void write_cdf(const std::string& filename);
        void write_sigmas(const std::string& filename);

        void iteration_start(int steps);
        double iteration_step();
        void iteration(double convergence);

        double potential(const double* x);

        double get_last_factor();

    private:
        C::Elcd* m_elcd;
};

#endif /*ELCD4MM_H_*/
```

electrode.h

```
/* $Id$ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
 *
 * Last changes:
 * $Author$
 * $Date$
 */

#ifndef ELECTRODE_H
#define ELECTRODE_H
```

```
typedef struct _Electrode Electrode;

#include "box.h"
#include "macros.h"

#include <stdlib.h>

enum Type {
    TYPE_RECT = 1,
    TYPE_WIRE,
    TYPE_CONE,
    TYPE_UNKNOWN
};

struct ElectrodeData {
    enum Type type;
    unsigned short nrot;
    double p[MAX_TOKEN - 2];
    unsigned long scale;
    double power;
};

struct Rect {
    double p[3];
    double n1[3];
    double n2[3];
    double a;
    double b;
};

struct Wire {
    double a[3];
    double b[3];
    double d;
};

struct NearElement {
    Electrode *element;
    double coeff;
};

struct _Electrode {
    enum Type type;
```

```
    double U;
    double sigma;
    struct NearElement *near;
    size_t near_num;
    double self;
    union {
        struct Rect rect;
        struct Wire wire;
    } geo;
};

Electrode *electrode_new(struct ElectrodeData* data, unsigned short rot);
void electrode_free(Electrode* electrode);
void electrode_center(const Electrode* electrode, double *vec);
void electrode_rotate(Electrode* elec, double angle);
void electrode_mirror(Electrode* elec, double zmirror);

void electrode_add_near_box(Electrode* electrode, const Box* box, int start,
       const double* matrix, int mirror, double zmirror);
double electrode_monopol(const Electrode* electrode);
void electrode_dipol(const Electrode* electrode, double *c, double *d);
#ifndef NO_QUADRUPOL
void electrode_quadrupol(const Electrode* electrode, double *c, double *Q);
#endif
#ifndef NO_OCTOPOLE
void electrode_octopole(const Electrode* electrode, double* c, double* A);
#endif
double electrode_potential(const Electrode* electrode, double *r);
double electrode_gauss_seidel(const Electrode* electrode);

#endif /*ELECTRODE_H*/
```

grid.h

```
/* $Id: grid.h 266 2007-08-22 08:21:21Z s_voec01 $ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
 *
 * Last changes:
 * $Author: s_voec01 $
 * $Date: 2007-08-22 10:21:21 +0200 (Mi, 22 Aug 2007) $
 */
```

```
#ifndef GRID_H_
#define GRID_H_

#include "box.h"

typedef struct _Grid
{
    int nrot;
    int ngroup;
    int nr;
    int nphi;
    int nz;
    double zmin;
    double zmax;
    double rmin;
    double rmax;
    Box** boxes;
    int near_r;
    int near_phi;
    int near_z;
    double near_range;
    double* rotations;
    int mirror;
    double zmirror;
} Grid;

Grid* grid_new(int nrot, int nr, int nphi, int nz);
void grid_free(Grid* system);
void grid_set_group(Grid* grid, int group);
void grid_set_electrodes(Grid* system, unsigned long n,
                         Electrode** electrodes);
void grid_set_near(Grid* grid, int r, int phi, int z);
void grid_set_near_range(Grid* grid, double near);
double grid_multipole_potential(const Grid* grid, Electrode* electrode);
void grid_update_box_multipoles(const Grid* grid);

#endif /*GRID_H_*/
```

linalg.h

```
/* $Id$ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
```

```
 *
 * Last changes:
 * $Author$
 * $Date$
 */

#ifndef LINALG_H
#define LINALG_H

typedef double Vector[3];
typedef double Matrix[9];
#ifndef NO_QUADRUPOL
typedef double Quadrupole[5];
#endif
#ifndef NO_OCTOPOLE
typedef double Octopole[10];
#endif

int fuzzy_equal(double a, double b);

void vector_clear(Vector a);
void vector_copy(const Vector a, Vector b);
int vector_is_null(const Vector a);
int vector_equal(const Vector a, const Vector b);
int vector_equal_exact(const Vector a, const Vector b);
void vector_add(const Vector a, const Vector b, Vector c);
void vector_iadd(Vector a, const Vector b);
void vector_sub(const Vector a, const Vector b, Vector c);
void vector_mul(const Vector a, double b, Vector c);
double vector_dot(const Vector a, const Vector b);
void vector_cross(const Vector a, const Vector b, Vector c);
double vector_squared(const Vector a);
double vector_abs(const Vector a);
double vector_r(const Vector a);
double vector_theta(const Vector a);
double vector_phi(const Vector a);
void vector_norm(const Vector a, Vector b);
void vector_spheric(double r, double theta, double phi, Vector a);
void vector_polar(double r, double phi, double z, Vector a);
void vector_print(const Vector a);
double vector_angle(const Vector a, const Vector b);
void vector_mirror(const Vector a, double zmirror, double* b);
void vector_imirror(Vector a, double zmirror);
```

```
void matrix_clear(Matrix a);
void matrix_copy(const Matrix a, Matrix b);
void matrix_identity(Matrix a);
int matrix_equal(const Matrix a, const Matrix b);
int matrix_equal_exact(const Matrix a, const Matrix b);
void matrix_add(const Matrix a, const Matrix b, Matrix c);
void matrix_sub(const Matrix a, const Matrix b, Matrix c);
void matrix_mul(const Matrix a, const Matrix b, Matrix c);
void matrix_mul_vec(const Matrix a, const Vector b, Vector c);
void matrix_rotation_x(Matrix a, double phi);
void matrix_rotation_y(Matrix a, double phi);
void matrix_rotation_z(Matrix a, double phi);
void matrix_rotation_zy(Matrix a, double phi1, double phi2);
void matrix_transpose(const Matrix a, Matrix b);

#ifndef NO_QUADRUPOL
void quadrupol_clear(Quadrupole q);
void quadrupol_mul(const Quadrupole q, double a, Quadrupole b);
void quadrupol_mul_vec(const Quadrupole q, const Vector a, Vector b);
void quadrupol_iadd(Quadrupole q, const Quadrupole a);
#endif

#ifndef NO_OCTOPOLE
void octopole_clear(Octopole q);
double octopole_get(const Octopole q, int i, int j, int k);
void octopole_mul(const Octopole q, double a, Octopole b);
void octopole_mul_vec(const Octopole q, const Vector a, Matrix b);
void octopole_iadd(Octopole q, const Octopole a);
#endif

#endif /*LINALG_H*/
```

list.h

```
/* $Id$ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
 *
 * Last changes:
 * $Author$
 * $Date$
 */
```

```
#ifndef ARRAY_H_
#define ARRAY_H_

typedef struct _List List;

struct _List{
    void* data;
    List* next;
};

List* list_append(List* list, void* data);
List* list_prepend(List* list, void* data);
List* list_append_list(List* list, List* list2);
List* list_prepend_list(List* list, List* list2);
List* list_remove(List* list, List* node);
void* list_index(List* list, int index);
List* list_remove_index(List* list, int index);
List* list_find(List* list, void* data);
int list_count(List* list);
void list_free(List* list, int free_data);
void** list_to_array(List* list);

#endif /*ARRAY_H_*/
```

macros.h

```
/* $Id: macros.h 289 2007-12-25 09:53:59Z s_voec01 $ */

/*
 * Written by Sebastian Voecking <sebastian.voecking@uni-muenster.de>
 *
 * Last changes:
 * $Author: s_voec01 $
 * $Date: 2007-12-25 10:53:59 +0100 (Di, 25 Dez 2007) $
 */

#ifndef MACROS_H_
#define MACROS_H_

#include <stdio.h>
#include <stdlib.h>

#define MAX_TOKEN 14
#define TRUE 1
```

```
#define FALSE 0
#define PI 3.14159265358979323846
#define EPS0 8.854187817e-12

#endif /*MACROS_H_*/
```

D Testergebnisse

		Rechenzeit [s]				
n_z	R_n	Konstruktion	Iteration	Gesamt	RAM [MB]	rel. Abweichung
20	25,00	14,83	7,26	22,09	19,0	$3,32 \cdot 10^{-7}$
20	20,00	12,45	11,31	23,76	16,4	$1,10 \cdot 10^{-6}$
20	15,00	9,82	30,28	40,10	13,5	$4,31 \cdot 10^{-6}$
40	20,00	12,94	12,61	25,55	16,9	$1,85 \cdot 10^{-7}$
40	15,00	10,37	20,39	30,76	14,1	$7,58 \cdot 10^{-7}$
40	12,50	8,95	81,83	90,78	12,5	$1,70 \cdot 10^{-6}$
60	20,00	13,16	16,91	30,07	17,1	$8,01 \cdot 10^{-8}$
60	16,67	11,47	21,66	33,13	15,3	$1,92 \cdot 10^{-7}$
60	13,33	9,67	39,06	48,73	13,3	$5,25 \cdot 10^{-7}$
60	11,67	8,68	100,16	108,84	12,1	$8,94 \cdot 10^{-7}$
80	20,00	13,27	21,85	35,12	17,3	$3,75 \cdot 10^{-8}$
80	17,50	12,06	24,41	36,47	15,9	$6,65 \cdot 10^{-8}$
80	15,00	10,70	31,91	42,61	14,4	$1,43 \cdot 10^{-7}$
80	12,50	9,31	52,19	61,50	12,8	$3,09 \cdot 10^{-7}$
80	11,25	8,57	104,66	113,23	12,0	$5,21 \cdot 10^{-7}$
100	20,00	13,40	25,31	38,71	17,3	$2,28 \cdot 10^{-8}$
100	18,00	12,40	28,35	40,75	16,2	$3,53 \cdot 10^{-8}$
100	16,00	11,34	33,11	44,45	15,1	$7,20 \cdot 10^{-8}$
100	14,00	10,24	40,94	51,18	13,9	$1,31 \cdot 10^{-7}$
100	12,00	9,12	64,20	73,32	12,6	$2,69 \cdot 10^{-7}$
100	11,00	8,51	102,07	110,58	12,0	$3,72 \cdot 10^{-7}$

Tabelle D.1: Einfluss des Parameters R_n für die einfache Testgeometrie mit $scale = 1000$

		Rechenzeit [s]				
n_z	R_n	Konstruktion	Iteration	Gesamt	RAM [MB]	rel. Abweichung
20	25,00	60,37	23,20	83,57	69,7	$4,07 \cdot 10^{-7}$
20	20,00	49,80	41,96	91,76	59,1	$1,16 \cdot 10^{-6}$
40	20,00	51,86	35,21	87,07	61,3	$2,20 \cdot 10^{-7}$
40	15,00	41,51	106,78	148,29	49,8	$8,16 \cdot 10^{-7}$
60	20,00	52,89	45,27	98,16	62,0	$8,00 \cdot 10^{-8}$
60	16,67	46,01	65,21	111,22	54,6	$1,88 \cdot 10^{-7}$
60	13,33	38,83	218,80	257,63	46,5	$5,25 \cdot 10^{-7}$
80	20,00	53,06	54,93	107,99	62,5	$4,52 \cdot 10^{-8}$
80	17,50	48,23	65,50	113,73	56,9	$7,71 \cdot 10^{-8}$
80	15,00	42,96	95,95	138,91	51,1	$1,60 \cdot 10^{-7}$
80	12,50	37,36	319,46	356,82	44,9	$3,56 \cdot 10^{-7}$
100	20,00	53,43	61,86	115,29	62,8	$2,43 \cdot 10^{-8}$
100	18,00	49,48	71,90	121,38	58,4	$3,50 \cdot 10^{-8}$
100	16,00	45,18	89,06	134,24	53,8	$6,54 \cdot 10^{-8}$
100	14,00	40,97	128,72	169,69	48,9	$1,24 \cdot 10^{-7}$
100	12,00	36,22	346,44	382,66	43,8	$2,43 \cdot 10^{-7}$

Tabelle D.2: Einfluss des Parameters R_n für die einfache Testgeometrie mit $scale = 2000$

		Rechenzeit [s]				
n_z	R_n	Konstruktion	Iteration	Gesamt	RAM [MB]	rel. Abweichung
20	25,00	133,65	47,21	180,86	153,5	$9,49 \cdot 10^{-7}$
20	20,00	112,65	100,94	213,59	129,9	$1,71 \cdot 10^{-6}$
40	20,00	117,08	70,76	187,84	134,7	$7,53 \cdot 10^{-7}$
40	15,00	93,79	461,23	555,02	108,8	$1,34 \cdot 10^{-6}$
60	20,00	118,27	83,95	202,22	136,4	$6,49 \cdot 10^{-7}$
60	16,67	103,24	133,05	236,29	119,7	$7,33 \cdot 10^{-7}$
60	15,00	95,24	222,60	317,84	110,8	$8,37 \cdot 10^{-7}$
80	20,00	119,31	98,60	217,91	137,3	$6,18 \cdot 10^{-7}$
80	17,50	107,87	124,62	232,49	124,9	$6,45 \cdot 10^{-7}$
80	15,00	96,13	203,63	299,76	111,7	$7,04 \cdot 10^{-7}$
80	13,75	89,83	384,55	474,38	104,9	$7,78 \cdot 10^{-7}$
100	20,00	119,32	110,27	229,59	137,9	$6,07 \cdot 10^{-7}$
100	18,00	110,48	126,45	236,93	128,0	$6,19 \cdot 10^{-7}$
100	16,00	101,25	168,54	269,79	117,7	$6,39 \cdot 10^{-7}$
100	14,00	91,56	284,63	376,19	106,9	$6,86 \cdot 10^{-7}$
100	13,00	86,48	486,23	572,71	101,3	$7,26 \cdot 10^{-7}$

Tabelle D.3: Einfluss des Parameters R_n für die einfache Testgeometrie mit $scale = 3000$

		Rechenzeit [s]				
n_z	R_n	Konstruktion	Iteration	Gesamt	RAM [MB]	rel. Abweichung
20	25,00	237,64	80,79	318,43	270,6	$4,19 \cdot 10^{-7}$
20	20,00	200,08	202,27	402,35	228,7	$1,24 \cdot 10^{-6}$
40	20,00	207,51	114,81	322,32	237,3	$2,30 \cdot 10^{-7}$
40	17,50	187,14	173,26	360,40	214,9	$4,35 \cdot 10^{-7}$
60	20,00	209,93	133,30	343,23	240,3	$9,46 \cdot 10^{-8}$
60	16,67	183,78	230,04	413,82	210,5	$2,12 \cdot 10^{-7}$
60	15,00	168,73	460,84	629,57	194,6	$3,42 \cdot 10^{-7}$
80	20,00	211,15	155,19	366,34	241,9	$4,66 \cdot 10^{-8}$
80	17,50	191,27	198,06	389,33	219,7	$8,58 \cdot 10^{-8}$
80	15,00	170,47	379,55	550,02	196,4	$1,68 \cdot 10^{-7}$
80	13,75	159,31	1093,32	1252,63	184,1	$2,45 \cdot 10^{-7}$
100	20,00	211,99	174,66	386,65	242,8	$2,83 \cdot 10^{-8}$
100	18,00	196,44	203,09	399,53	225,3	$4,52 \cdot 10^{-8}$
100	16,00	182,30	276,81	459,11	206,9	$7,34 \cdot 10^{-8}$
100	14,00	162,58	583,60	746,18	187,6	$1,27 \cdot 10^{-7}$
100	13,00	153,67	1571,61	1725,28	177,7	$1,78 \cdot 10^{-7}$

Tabelle D.4: Einfluss des Parameters R_n für die einfache Testgeometrie mit $scale = 4000$

			Rechenzeit [s]				
n_φ	n_r	R_n [m]	Konstr.	Iteration	Gesamt	RAM [MB]	rel. Abweichung
-10	30	10,00	174,44	79,16	253,60	3,1	$8,75 \cdot 10^{-6}$
-10	30	9,00	170,40	64,49	234,89	3,2	$1,33 \cdot 10^{-5}$
-10	30	8,00	163,77	122,68	286,45	3,2	$2,11 \cdot 10^{-5}$
-10	30	7,00	158,99	591,10	750,09	3,1	$9,09 \cdot 10^{-5}$
-20	30	10,00	172,68	39,73	212,41	3,1	$1,96 \cdot 10^{-5}$
-20	30	9,00	168,75	34,23	202,98	3,1	$5,13 \cdot 10^{-5}$
-20	30	8,00	162,78	62,54	225,32	3,0	$7,77 \cdot 10^{-5}$
-20	30	7,00	158,23	274,24	432,47	3,0	$1,87 \cdot 10^{-4}$
-30	30	10,00	172,28	26,60	198,88	3,0	$4,19 \cdot 10^{-5}$
-30	30	9,00	168,37	23,09	191,46	3,0	$1,05 \cdot 10^{-4}$
-30	30	8,00	162,53	41,12	203,65	3,0	$1,71 \cdot 10^{-4}$
-30	30	7,00	158,31	189,14	347,45	2,9	$3,64 \cdot 10^{-4}$
-40	30	80,00	365,50	10,12	375,62	3,2	$3,34 \cdot 10^{-6}$
-40	30	10,00	172,49	20,54	193,03	3,1	$7,21 \cdot 10^{-5}$
-40	30	9,00	168,43	16,33	184,76	3,0	$2,14 \cdot 10^{-4}$
-40	30	8,00	162,17	36,88	199,05	3,1	$3,12 \cdot 10^{-4}$
-40	30	7,00	158,23	149,75	307,98	3,0	$5,75 \cdot 10^{-4}$
-40	60	80,00	366,33	15,14	381,47	3,4	$3,34 \cdot 10^{-6}$
-40	60	10,00	172,61	30,46	203,07	3,1	$1,02 \cdot 10^{-4}$
-40	60	9,00	168,59	23,54	192,13	3,2	$3,43 \cdot 10^{-4}$
-40	60	8,00	162,89	46,77	209,66	3,1	$5,21 \cdot 10^{-4}$
-40	60	7,00	158,50	215,02	373,52	3,1	$5,90 \cdot 10^{-4}$
-40	100	80,00	375,15	20,63	395,78	3,6	$3,34 \cdot 10^{-6}$
-40	100	10,00	173,87	39,26	213,13	3,3	$1,27 \cdot 10^{-4}$
-40	100	9,00	169,72	30,71	200,43	3,2	$1,43 \cdot 10^{-4}$
-40	100	8,00	163,64	61,88	225,52	3,2	$3,80 \cdot 10^{-4}$
-40	100	7,00	159,05	260,21	419,26	3,1	$6,16 \cdot 10^{-4}$

Tabelle D.5: Testrechnung mit *Elcd4* für ein leeres Hauptspektrometer

Literaturverzeichnis

[Arc] Archlinux. http://www.archlinux.org.

[Cru63] D. Cruise. A numerical method for the determination of an electric field about a complicated boundary. *Journal of Applied Physics*, 34(12):3477–3479, 1963.

[Fla04] B. Flatt. *Dissertation*. PhD thesis, Institut für Physik, Universität Mainz, 2004.

[GR87] L. Greengard and V. Rokhlin. A fast alogorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[GTK] Gtk+ - the gimp toolkit. http://www.gtk.org.

[Hug08] K. Hugenberg. *Diplomarbeit*. 2008.

[Jac99] J. Jackson. *Classical Electrodynamics*. Third edition, 1999.

[Kra03] C. Kraus. *Dissertation*. PhD thesis, Institut für Physik, Universität Mainz, 2003.

[Pla00] R. Plato. *Numerische Mathematik kompakt*. Vieweg, 2000.

[Pur98] E. Purisma. Fast summation boundary element method for calculating solvation free energies of macromolecules. *Journal of Computational Chemistry*, 19(13):1494–1504, 1998.

[PyG] Pygtk. http://www.pygtk.org.

[Pyt] Python programming language – official website. http://www.python.org.

[Sch97] N. Schmitz. *Neutrinophysik*. 1997.

[Thu02] T. Thuemmler. *Entwicklung von Methoden zur Untergrundreduzierung am Mainzer Tritium-β-Spektrometer*. Januar 2002.

[Val04] K. Valerius. *Elektromagnetisches Design für das Hauptspektrometer des KATRIN-Experiments*. 2004.

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die zur Entstehung dieser Arbeit beigetragen haben.
An erster Stelle bedanke ich mich bei Prof. Dr. Christian Weinheimer für die Vergabe dieses interessanten Themas und für die gewährte Unterstützung und Betreuung.
Kathrin Valerius möchte ich danken für die umfassende Unterstützung und Betreuung, die vielen Tipps und Hilfestellungen, sowie die Durchsicht dieser Arbeit.
Dr. Ferenc Glück danke ich für die Literatursuche und Hilfe bei meinen Verständnisfragen.
Prof. Dr. Johannes Wessels danke ich dafür, dass er das Korreferat für diese Diplomarbeit übernommen hat.
Ich danke Kim für die Durchsicht dieser Arbeit und die Korrektur der schlimmsten sprachlichen Patzer.

Ganz besonders möchte ich noch Fuzhou danken, die während der Vorbereitungen zu dieser Arbeit leider verstorben ist.
Besonderer Dank gilt ebenso meinen Eltern, die mich bei meinem Studium und auch bei meinem Aufenthalt am CERN stets unterstützt haben.

Eigenständigkeitserklärung

Ich versichere, dass ich die Arbeit selbständig verfasst
und keine anderen als die angegebenen Quellen und
Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Münster, 18. Januar 2008

Sebastian Vöcking