Kopplung von statistischen „Monte-Carlo“ Solvern mit numerischen Fluidcodes

Diplomarbeit von
Petra Börner
p.boerner@fz-juelich.de

eingereicht bei

Prof. Dr. Jörg Keller

Fachbereich Informatik
Fernuniversität in Hagen

Jülich / Düren, Juli 2005

Inhaltsverzeichnis

1 Einleitung

Durch immer leistungsfähigere Rechnersysteme wird es in zunehmendem Maße möglich, das dynamische Verhalten von physikalischen Systemen auf makroskopischer Skala (z.B. Kontinuumsmechanik) durch Kopplung an mikroskopische (kinetische) Untermodelle, die auf molekularer Skala definiert sind, auch in Grenzbereichen der Gültigkeit der Strömungsmechanik noch genau zu simulieren.

Zahlreiche Anwendungen in Wissenschaft und Technik resultieren aus diesen neuen Möglichkeiten, etwa

a) bei der Beschreibung von viskoelastischen Flüssigkeiten,
b) bei der Kopplung von mikroskopischen Turbulenzmodellen und strömungsmechanischen Beschreibungen turbulenter Bereiche in Gasströmungen, etwa in Grenzschichten an Tragflächen im Flugzeugbau,
c) bei (kinetischen) Korrekturen an Strömungsgleichungen im Übergangsbereich zu eigentlich kinetisch zu beschreibenden Phänomenen (das Problem des sinnvollen Abschlusses (engl.: „closure“) der makroskopischen Gleichungen), z.B.: Überschallflug, Re-entry-Probleme in der Raumfahrt, etc...
d) bei der konsistenten Beschreibung komplexer chemischer Vorgänge in Strömungen (z. B. die Simulation von Verbrennungsprozessen in Motoren).

In all diesen Anwendungen wird einerseits das mikroskopische Verhalten

in a) die molekulare Konfiguration der Polymere in der Lösung,
in b) die mikroskopische Turbulenz,
in c) die individuellen elastischen Stossprozesse der Gasmoleküle
in d) die einzelnen chemischen Reaktionen

durch die makroskopischen Parameter (Dichte, Temperatur, etc.) der Strömung bestimmt, und andererseits beeinflusst dieses mikroskopische Verhalten die Strömung.

Obwohl diese „Mikro-Makro“ Techniken bezüglich der benötigten Computerressourcen weitaus höhere Anforderungen stellen als etwa konventionelle Strömungs- oder Kontinuumsrechnungen, erlauben sie die direkte Nutzung von Modellen der kinetischen Theorie und vermeiden so potentielle Ungenauigkeiten insbesondere in Grenzbereichen der Anwendbarkeit rein makroskopischer Modelle.

Aus diesem Grund sind numerische Verfahren für sogenannte „Mikro-Makro” Probleme, und deren effiziente Umsetzung auf Rechenanlagen, ein zentrales Forschungsgebiet geworden, das interdisziplinär ist und zwischen der „Computational Physics“ und der Informatik liegt.

Einige der wichtigen, aber besonders komplexen Anwendungsbereiche liegen in der technischen Plasmaphysik, und auch in der Fusionsforschung. In beiden Fällen liegt in der Regel kein vollständiges thermodynamisches Gleichgewicht vor, und es handelt sich um ein typisches „Multispezies-Problem“, also eine Mischung von vielen sich in ihrem dynamischen Verhalten signifikant unterscheidenden, aber dennoch gegenseitig stark beeinflussenden Komponenten. Selbst in den einfachsten Plasmen hat man immer mindestens eine Mischung aus Elektronen, Ionen, neutralen Teilchen und Photonen (Strahlungsfeld).
Einige der Plasmaeigenschaften lassen sich durch thermodynamische Gleichgewichtsansätze formulieren, andere durch Kontinuumsmechanik, und wieder andere nur durch mikroskopische, kinetische Beschreibungen.

Die relativ kalten, technischen Plasmen (d.h. kälter als 20000 K) sind oft schwach ionisiert, hier verhält sich die neutrale Komponente häufig wie eine makroskopische Strömung. Ein makroskopischer Diffusionsansatz wäre also möglich. Ionen und/oder Elektronen müssen

aber gleichzeitig oft kinetisch beschrieben werden. Das Strahlungsfeld kann entweder entkoppelt sein (im optisch dünnen Fall), oder über Reabsorptionsprozesse selbst signifikant auf das Plasma zurückwirken.
Bei sehr hohen Plasmadrucken kann das Verhalten von Ionen und Elektronen in den Gültigkeitsbereich der Kontinuumsmechanik fallen, aber selbst dann muss oft noch das Strahlungsfeld „kinetisch", d.h. mikroskopisch, beschrieben werden. Eine solche Situation liegt zum Beispiel in Plasma-Hochdrucklampen, die etwa bei Kaufhausbeleuchtungen oder in Autoscheinwerfern verwendet werden, vor:

Zusammensetzung:	chemisches Gleichgewicht
Elektronen, Ionen, Neutralgas:	(makroskopische) Kontinuumsmechanik
Strahlung:	kinetisches (mikroskopisches) Verhalten.

Die sich unter informatischen Gesichtspunkten bei „Mikro-Makro" Problemen ergebenden Aufgaben werden in der vorliegenden Arbeit anhand dieses letztgenannten Beispiels diskutiert. Es handelte sich hierbei um Untersuchungen im Rahmen eines vom BmBF geförderten Verbundprojektes mit industriellen und öffentlich geförderten Partnern.

Im Institut für Plasmaphysik des Forschungszentrum Jülich werden schon seit ca. 2 Jahrzehnten analoge „Mikro-Makro" Probleme im Zusammenhang mit der Kernfusionsforschung bearbeitet. Im wandnahen Bereich der heißen Wasserstoffplasmen, wie sie derzeit schon mit Temperaturen bis zu 100 Mill. Grad erzeugt und magnetisch eingeschlossen werden, verhalten sich die Hauptkomponenten des Plasmas (Wasserstoffionen und -elektronen) makroskopisch, werden also durch Methoden der CFD (Computational Fluid Dynamics) beschrieben. Das Neutralgas (Wasserstoffatome und -moleküle) sowie die Strahlung bestimmen in der Nähe der am stärksten belasteten Bauteile der Brennkammer zwar das Strömungsverhalten, müssen aber kinetisch behandelt werden, da die freien Weglängen für die Elementarprozesse im Vergleich zu typischen Systemabmessungen nicht zu vernachlässigen sind.
Hier, wie auch beim in der Arbeit konkret untersuchten Anwendungsfall aus der Lichttechnik, ist der mikroskopische Teil des Systems geometrisch und physikalisch so komplex, dass für diesen Teil (statistische) Monte Carlo Verfahren eingesetzt werden müssen. (Dies gilt analog auch für die oben genannten Probleme bei viskoelastischen Strömungen, und vermutlich sogar generell bei „Mikro-Makro" Problemen, wenn der mikroskopische Teil sehr detailliert behandelt werden muss.)
Einerseits erleichtert dies die Parallelisierbarkeit dieses Programmteils, andererseits treten bei der iterativen Kopplung von statistischen mit deterministischen Algorithmen für die unterschiedlichen Komponenten des Gesamtsystems eine ganze Reihe von neuen, prinzipiellen Schwierigkeiten auf (z.B.: Gitterharmonisierung, Speicherplatzverwaltung, Implementierungsaspekte, load balancing, etc.), die offenbar für diese Art von numerisch/statistischen Aufgaben in der Literatur nur wenig untersucht sind. Auch diese werden in der vorliegenden Arbeit diskutiert und anhand des oben genannten Beispiels konkret erläutert.

Die Arbeit ist wie folgt gegliedert:
Das nachfolgende Kapitel 2 diskutiert die allgemeine Problemstellung bei „Mikro-Makro"-Simulationen, und zwar den wichtigen Spezialfall des Einsatzes von statistischen Algorithmen für den mikroskopischen Teil. Die sich unter informatischen Aspekten ergebenden Konsequenzen werden herausgestellt.

Dagegen wird dort und im Folgenden vorausgesetzt, dass die Algorithmen und die Software für den makroskopische Teil in ausreichendem Umfang verfügbar ist und verweisen hier auf die im großen Umfang vorhandene Standardliteratur /1/ , /2/ ,/3/ .
Diese Arbeit betrachtet also keine Probleme des Algorithmus-Engineering für den Makro-Teil des gekoppelten Programmsystems. Diese Annahme ist angesichts der historisch schon sehr lange andauernden und umfangreichen, auch kommerziellen Forschung, Entwicklung und Anwendung auf diesem Gebiet im Allgemeinen sicherlich gut begründet, wenn es auch speziell bei der Simulation des Randgebietes von magnetisch eingeschlossenen Plasmen

der Fusionsforschung noch viele auch algorithmisch ungelöste Teilaspekte gibt /4/ . Diese werden in der vorliegenden Arbeit aber nicht behandelt.

Anders scheint die Situation beim Mikro-Teil des Systems zu sein. Monte Carlo Prozeduren werden oft rein intuitiv begründet: Begriffe wie „Teilchensimulation“, „Auswürfeln von Elementarprozessen“ usw. werden oft verwendet. Dies ist ausreichend bei Problemen, die nur den Einsatz des Monte Carlo Verfahrens erfordern. Bei Vernetzung des MC-Codes mit einem deterministischen numerischen CFD Code ist es aber notwendig, dieses mathematisch zu präzisieren, um die zwischen den beiden beteiligten Teilmodulen auszutauschende Information auch explizit angeben, und notfalls die Monte Carlo Prozeduren den speziellen algorithmischen Eigenschaften des CFD Codes anpassen zu können.

Aus diesem Grund werden die für die weiteren Diskussionen benötigten konzeptionellen Grundlagen einer Monte Carlo Simulation von linearen Transportproblemen im Kapitel 3 kurz zusammengestellt. Dabei wird im Wesentlichen Bezug genommen auf das Online-Manual /5/ des seit ca. 25 Jahren im Institut für Plasmaphysik der Forschungszentrum Jülich GmbH für die Fusionsforschung entwickelten und angewendeten 3D Monte Carlo Code EIRENE /6/ .

Anhand des kommerziell verfügbaren 3D Finite-Elemente Codes FIDAP /7/ (Makro-Code) für technische Plasmen und seiner Verkopplung an den in der Fusionsforschung weit verbreiteten 3D Monte Carlo Code EIRENE (Mikro-Code) wird dann in den folgenden Kapiteln 4-6 konkret gezeigt welche Aufgaben konkret anfallen, wie sie im Einzelnen gelöst werden können, und welche verbliebenen Probleme noch in der Zukunft behandelt werden müssen.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick.

Als wesentlichste Ergebnisse der vorliegenden Arbeit konnte eine weitestgehende Konsistenz zwischen einem 3D Finite-Elemente Code und einem ebenfalls 3D Monte Carlo Code erzielt werden. Es wurde gezeigt, wie die gegensätzlichen Algorithmen über einer Gitterharmonisierung geometrisch kompatibel gemacht werden können, aber auch welche Grenzen nach wie vor bestehen.
Hinsichtlich Datenaustausch, Speicherplatzverwaltung, Visualisierung und Optimierung fallen bei „Mikro-Makro“ Modellen spezifische Aufgaben an, die anhand eines konkreten Anwendungsbeispiels beschrieben und teilweise durch konkret vorgeschlagene Algorithmen gelöst werden.

2 Das generelle Konzept der Code-Kopplung bei „Mikro-Makro“-Poblemen

Ausgangspunkt der Betrachtungen sind zwei zunächst völlig separierte Programmpakete A für den makroskopischen Teil und B für den mikroskopischen Teil. Der sicherlich am häufigsten auftretende Fall ist ein CFD (computational fluid dynamics) Algorithmus in A, und ein Monte Carlo Solver für ein lineares Transportproblem (z.B eine lineare kinetische Gleichung vom Boltzmann- oder Fokker-Planck Typ) in B.

Die generische Gleichung für Modul A /2/ für eine Größe Φ lautet dann:

$$\frac{\partial}{\partial t}(\rho\Phi) + div(\rho\,\vec{u}\,\Phi) = div(\kappa\ grad\,\Phi) + S \qquad (1)$$

κ ist der Diffusionskoeffizient, *S* ist der Quellterm, Φ ist die abhängige Variable (z.B. Temperatur, Geschwindigkeit, Teilchendichte) und ρ die Massendichte. Der erste Term links beschreibt die explizite Zeitabhängigkeit, der zweite ist der Konvektionsterm (mit Geschwindingkeit $\vec{u}$). Der erste Term rechts ist der diffusive Anteil.
Alles was in einer speziellen physikalischen oder technischen Anwendung formal nicht in einen dieser Terme passt, wird in den „Quellterm“ *S* gesteckt.

Einzelne Gleichungen diesen Typs (etwa: die Navier-Stokes Gleichung in der Fluidmechanik) oder gekoppelte Systeme solcher Gleichungen (z.B. in der Plasmaphysik, oder bei Problemen der Wärmeübertragung) jeweils mit geeigneten Randbedingungen, werden mit Standardmethoden der numerischen Mathematik (Differenzenkalkül) behandelt. Unabhängige Variablen sind der Ort und die Zeit $(\vec{r},t)$.

Die generische Gleichung für Programmpaket B ist typischerweise eine integro-differential Gleichung in einem höherdimensionalen Phasenraum $(\vec{r},t,\vec{v},\ldots)$.

$$\begin{aligned}\left[\frac{\partial}{\partial t} + \underline{v}\cdot\nabla_{\underline{r}} + \frac{\underline{F}(\underline{r},\underline{v},t)}{m}\cdot\nabla_{\underline{v}}\right] f(\underline{r},\underline{v},t) \;&=\; \iiint \sigma(\underline{v}',\underline{V}';\underline{v},\underline{V})\,|\underline{v}'-\underline{V}'|\, f(\underline{v}')f_b(\underline{V}') - \\ &-\; \iiint \sigma(\underline{v},\underline{V};\underline{v}',\underline{V}')\,|\underline{v}-\underline{V}|\, f(\underline{v})f_b(\underline{V})\end{aligned} \qquad (2)$$

Diese generische mikroskopische Gleichung ist in Kapitel 3 noch einmal erläutert. Sie beschreibt einen Teilaspekt des Gesamtproblems detaillierter als Gleichung (1), in einem höherdimensionalen Phasenraum. Bei Übergabe von Ergebnissen von B nach A müssen die Daten also über die in B gegenüber A zusätzlichen „Dimensionen“ integriert werden. Nur Monte Carlo Codes leisten dies „automatisch“, sie integrieren über Teile des Phasenraums von *f*, ohne f zuvor erst im ganzen Phasenraum bestimmen zu müssen (siehe Kapitel 3). Umgekehrt muss bei Datenübernahme von A nach B der Phasenraum „aufgeblasen“ werden. D.h. die implizit in dem „Makro“-Ansatz gemachten Gleichgewichtsannahmen (z. B. Maxwellverteilung für Atome) müssen wieder explizit in Modul B verwendet werden.

Weder der mathematische Typ der Gleichungen noch der Phasenraum sind also bei A und B identisch. Aus diesem Grund ist davon auszugehen, dass in A und B auch prinzipiell unterschiedliche Algorithmen zum Einsatz kommen.
Bei großen, integrierten Programmsystemen, wie sie in den hier diskutierten Beispielen vorliegen, ist sogar davon auszugehen, dass A und B von unterschiedlichen Arbeitsgruppen, mit komplementärer numerischer und physikalischer Expertise, entwickelt werden. Aus diesem Grund werden auch im gekoppelten Gesamtpaket die beiden einzelnen Module A und B

weitgehend separiert bleiben müssen, damit die jeweiligen Arbeitsgruppen ihren Modul weiter warten, ausbauen und sicher bedienen können.

Andererseits muss ein regelmäßiger Austausch großer Datenmengen zwischen A und B sicher und effektiv erfolgen. Ferner müssen aus Gründen der inneren Konsistenz Teile des Gesamtpakets von A und B gemeinsam genutzt werden können, etwa atomare oder molekulare Daten, die sowohl in die Auswertung von Termen in Gleichung (1) als auch in Gleichung (2) eingehen.

Um den Anforderungen nach einerseits weitgehender Separiertheit und andererseits nach größtmöglicher Nähe zum effizienten Datentransfer und gemeinsamer Nutzung von Programmteilen gerecht zu werden, ist es sinnvoll, zwischen den beiden Modulen ein Kopplungsplugin vorzusehen. Dieses kann alle kopplungsspezifischen Programmkomponenten aufnehmen und beiden Modulen zur Verfügung stellen. Durch Definition geeigneter standardisierter Schnittstellen mit den beiden Modulen A und B eröffnet dieses Vorgehen auch die Möglichkeit, die Einzelmodule mit unterschiedlichen Programmpaketen zu verbinden. Auf diese Weise ist zum Beispiel der im speziellen Anwendungsfall verwendete Mikro-Code EIRENE bereits seit langem mit dem CFD-basierten 2D Plasmatransportcode B2 gekoppelt /4/ /6/

2.1 Notwendigkeit der Gitterharmonisierung

In wissenschaftlichen Anwendungen sind die beiden oben beschriebenen Typen von Gleichungen in der Regel Erhaltungsgleichungen. Die Differentialgleichungen der CFD sind Erhaltungsgleichungen z.B. für Masse, Impuls und Energie eines gedachten infinitesimalen Flüssigkeitsvolumens. Die kinetischen Gleichungen sind als Bilanzgleichungen der mikroskopisch behandelten Spezies in einem höherdimensionalen Phasenraum (r,v) anzusehen. Beim gekoppelten Paket müssen diese konservativen Eigenschaften erhalten bleiben. Mit anderen Worten: durch die gemeinsame und konsistente Lösung von Gleichung (1) und (2) dürfen durch Interpolation, oder sonstige Ungenauigkeiten beim Datentransfer, keine zusätzlichen, künstlichen „numerischen“ Terme in Gleichung (1) entstehen. Umgekehrt müssen bei Mittelung über die zusätzlichen Phasenraumvariablen v in Gleichung (2) im Modul B an entsprechenden Stellen exakt die Terme entstehen, die sich auch innerhalb des Moduls A ergeben.

Zum Beispiel kann die Emission von Photonen im Modul A aus makroskopischen Größen berechnet werden. Andererseits liefert genau diese Reaktion im Modul B die Photonenquelle. Die in B mit Zufallszahlen „erwürfelte“ Quelle muss zur Vermeidung von numerischen Inkonsistenzen also genau die in A berechnete Größe ergeben.
Der Modul B liefert an A oft die Differenz zweier Größen SA-SB (z.B. Quelle – Senke = „Nettoquelle“ für Φ in Gleichung (1)). Die Differenz der beiden Terme ist manchmal klein gegenüber den Einzelanteilen SA und SB, aber sie ist groß im Vergleich mit anderen Termen, die in die Gleichung (1) eingehen, die Modul A löst. Es ist also darauf zu achten, dass bei der Übertragung von A nach B und von B nach A keine Übertragungsfehler entstehen, die in die Größenordnung von SA-SB kommen.
Aus diesem Grund ergibt sich eine erste Notwendigkeit:
Die Diskretisierung des gemeinsamen Teils des Phasenraums (hier z. B.: Ortsraum, Koordinate r) sollte in A und B möglichst identisch sein: d.h. gleiche Rechengitter, also Gitterharmonisierung.

Da die Monte Carlo Methode intrinsisch ein Integrationsverfahren ist, siehe Kapitel 3, also die von B nach A gelieferte Information in Form von Integralen übergeben wird, ist es sinnvoll auf CFD Seite Finite-Elemente- /3/ oder „Finite Volume“-Verfahren /1/ zu verwenden, da diese Prozeduren auch auf integralen Erhaltungssätzen basieren.

Weniger günstig sind aus diesem Grund „Finite Differenzen“-Verfahren, weil dann zusätzliche Interpolation bei der Kommunikation zwischen A und B nötig wird, die in der Regel bei komplexen geometrischen Anordnungen nicht mehr konservativ, also nicht mehr streng den Erhaltungsgesetzen gehorchend, ausgelegt werden kann.

In Kapitel 4 wird am konkreten Beispiel FIDAP-EIRENE die Gitterharmonisierung und die speziell dafür notwendigen Algorithmen im Modul B beschrieben.
Allgemein lässt sich feststellen:
Wegen der unterschiedlichen algorithmischen Konzepte in A und B hat der gemeinsame geometrische Block auch unterschiedliche Aufgaben zu erfüllen. Die Gitter im Teil A dienen der Diskretisierung der Gleichungen und der Berechnung der Metrik (metrischer Tensor, lokale Basiseinheitsvektoren, Christoffelsymbole, etc. /8/), im Monte Carlo Modul B fallen andere Aufgaben an. Es müssen dort sehr schnell und genau Schnittpunkte von Geraden mit den Gitterwänden berechnet werden /9/ (der Hauptteil der Rechenzeit bei Modul B besteht nur darin), und es müssen beliebige Punkte im Rechengebiet sehr effizient einer Gitterzelle zugeordnet werden können, dies auch bei unstrukturierten Gittern.

Dies bedeutet, dass exakte algebraische Ansätze verwendet werden müssen. Bei numerischer Schnittpunktberechnung treten bei Teilchensimulationen auf Grund von Rundungsungenauigkeiten immer wieder Fehler z.B. bei der Zuordnung einer Teilchenposition zu einer Gitterzellnummer auf. Dies macht den Algorithmus überaus unübersichtlich und fehleranfällig. Algebraische Schnittpunktberechnungen schränken aber die Form der Grenzflächen ein.

Mit Hilfe der Cardano’schen Formeln /10/ ist es prinzipiell möglich Gleichungen bis zu maximal 4ter Ordnung zu lösen. Gleichungen höherer Ordnung können nicht mehr algebraisch gelöst werden /11/ . Schnittpunktberechnungen von Geraden mit Tori zum Beispiel führen zu Gleichungen 4ter Ordnung, Schnittpunktberechnungen von Geraden mit Zylindern ergeben Gleichungen 2ter Ordnung, und Schnittpunktberechnungen von Geraden mit ebenen Gitterflächen führen zu einem linearen Problem.
Falls der Code A (wie z.B. der in Kapitel. 4 beschriebene FIDAP Code) aber in kartesischen Koordinaten nicht zu solchen Gleichungen für die Gitterflächen reduzierbar ist, müssen Kompromisse zwischen der Gitterharmonisierung und der Rechenzeit zur Schnittpunktberechnung gefunden werden. Eine Möglichkeit ist die weitere Zerlegung der Gitterzellen des Moduls A in feinere, aber geometrisch einfachere Unterzellen im Modul B. Dieses Konzept wird weiter unten genauer ausgeführt.

2.2 Datentransfer und Speicherplatzverwaltung

Wenn die Gitterharmonisierung erfolgt ist, so ist der Datentransfer von A nach B relativ einfach, da eine eins zu eins Beziehung zwischen den Werten der Strömungsfelder im Modul A und B besteht. Auch der umgekehrte Weg, von B nach A, ist nach erfolgter Gitterharmonisierung prinzipiell trivial, wenn Algorithmus A auf finiten Volumen (nicht: finiten Differenzen) aufbaut ist, und Integrale über Raumzellen direkt in Gleichung (1) eingesetzt werden können.

Wie der Datenaustausch zwischen den beiden beteiligten Modulen realisiert wird, ist stark problemabhängig. Der erste Ansatz wird immer sein, die beiden Module als vollständig separierte Codes zu betreiben und die benötigten Daten per Filetransfer auszutauschen. Dieses Vorgehen kann für spezielle Anwendungen ausreichend sein, wenn sich das Gesamtsystem hinreichend robust verhält und eine gute Konvergenzrate aufweist, sodass nur wenige Iterationen notwendig sind. In anderen Anwendungen kann es sich als notwendig erweisen, die Kopplung enger zu gestalten und z.B. Modul B als Prozedur innerhalb von Modul A zu betreiben. Dies kann zum Beispiel der Fall sein, wenn erhebliche Datenmengen ausge-

tauscht werden müssen, oder die Anzahl der Iteration des gekoppelten Systems sehr hoch ist, so dass der Rechenzeitaufwand für Filetransfer (Schreiben und Lesen von Dateien ist langsam) und/oder Programminitialisierung (Aufbereiten der geometrischen Informationen, Bereitstellen des Hintergrundmediums, etc) nicht mehr tolerabel ist. In diesem Fall kann der Datentransfer mit Hilfe von internen Feldern geschehen.

An dieser Stelle ergeben sich Aufgaben der Speicherplatzverwaltung und Optimierung. In der Regel sind CFD-Codes alleine immer schon so umfangreich, dass sie die verfügbaren Kapazitäten der Rechner voll ausschöpfen. Die mikroskopische Komponente hat einen höherdimensionalen Phasenraum und, siehe Kapitel 4, bereits im Orts (-Unter-) raum die gleiche Auflösung. Sie stellt also mindestens genauso große Speicherplatzanforderungen wie A. Kapitel 5 diskutiert, wie diese Probleme durch dynamische Speicherplatzverwaltung entschärft werden können.
Hierzu ist es nötig, dass die Module A und B jeweils Speicher freigeben müssen, wenn gerade das andere Modul aktiv ist. Dem sind unter Umständen Grenzen gesetzt, wenn nicht in beide Module eingegriffen werden kann („open source“ Problematik, siehe Kapitel 5.5).

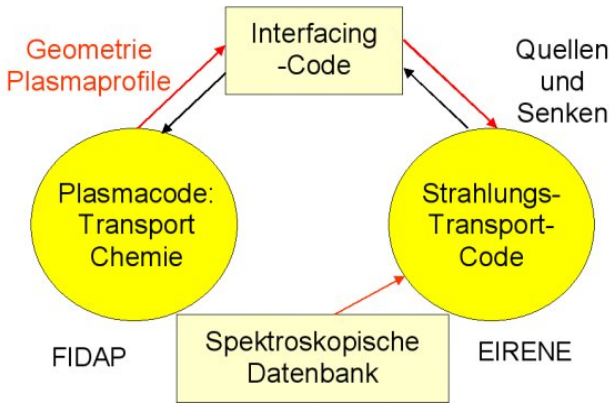


Abbildung 1 die allgemeine Struktur des Gesamtcodes

Wenn einzelne Elementarprozesse sowohl in Modul A als auch in Modul B auftreten, dann bietet sich ein gemeinsamer Code-Teil an. Im konkreten Anwendungsfall ist dies zumindest eine gemeinsame atomare bzw. spektroskopische Datenbank (siehe Kapitel 5). Aus dieser Datenbank werden in Modul A Teile des Quellterms (Gleichung (1), letzter Term rechte Seite) berechnet, und in Modul B die Volumenquelle S (Gleichung (7) unten und Kapitel 4.3.3) definiert und simuliert. Die allgemeine Struktur des gekoppelten Codes ist dargestellt in Abbildung 1.

Auch die Konvergenz des Gesamtpakets ist in dem Fall weder durch Modul A ($\sim O\left(\Delta t^{\alpha}\right)$) noch durch Modul B ($\sim O\left(\frac{1}{\sqrt{CPU}}\right) \sim O\left(\frac{1}{\sqrt{N}}\right)$) bestimmt, sondern durch eine Kombination davon /6/ , siehe Abbildung 2.

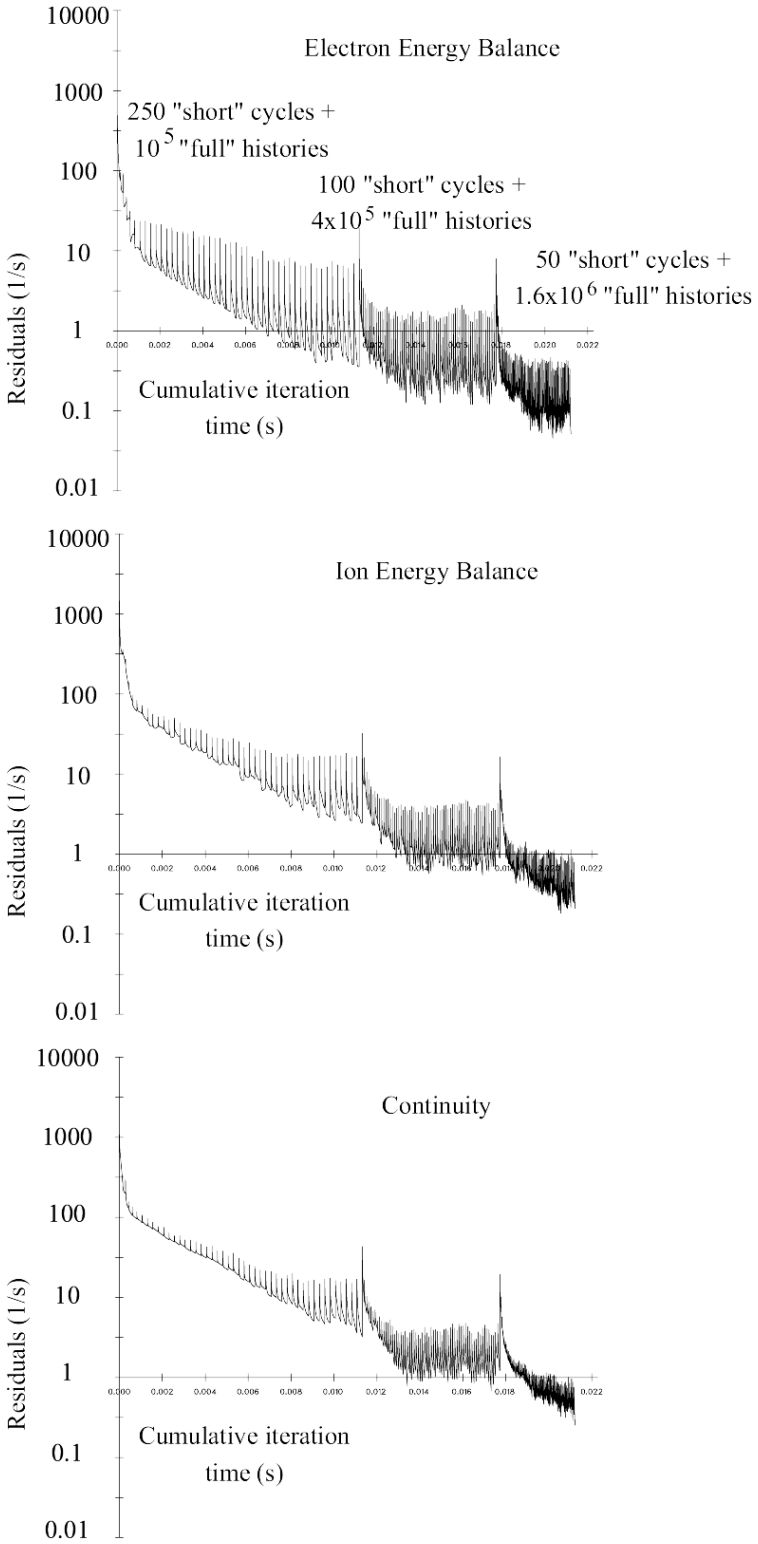


Abbildung 2 Konvergenz des B2-EIRENE-Codes mit dem typischen „saturated residual" Verhalten aufgetragen gegen die akkumulierte Iterationszeit und die numerischen und statistischen Kontrollparameter n_s und N_{MC} für die Elektronenenergie-, Ionenenergie- und Kontinuitätsgleichung. Die Residuen (1/s) sind hier auf einer logarithmischen Skala dargestellt. Das Inverse der gezeigten Residuen kann man als typische physikalische Zeitkonstante (s) ansehen, in der die Temperaturen und Dichten sich in weiteren Iterationen ändern.

Abbildung 2 zeigt die Abnahme der Residuen (normalisierte Fehler in den Bilanzgleichungen) des gekoppelten B2-EIRENE Codes mit steigender Anzahl der Iterationen. Wenn nach einer gewissen Anzahl von Iterationszeitschritten eine Sättigung eintritt, dann ist das (normalisierte) Rauschen im Monte Carlo Quellterm vergleichbar geworden zum numerischen Fehler in der Bilanz. Eine weitere Verringerung der Residuen kann nur durch Erhöhen der Anzahl der Monte Carlo Historien N_{MC}, und durch Reduktion des Zeitschritts Δt bzw. von n_s, der Anzahl der internen B2 Iterationen zwischen zwei vollen EIRENE Aufrufen erreicht werden.
Dieses Verhalten ist typisch für Mikro-Makro Anwendungen, wenn der Mikro-Teil ein Monte Carlo Verfahren ist, nicht nur für B2-EIRENE wie z.B. in /6/ . Die Konvergenz zeigt also das Verhalten sogenannter gesättigter Residuen. Für jedes feste N ist ein bestimmtes Niveau des Iterationsfehlers (= „Residuum“) nicht zu unterschreiten, sondern durch Rauschen in Modul B bestimmt. Dann muss die Rechenzeit (oder Prozesssorenzahl) in B erhöht werden bis erneut Sättigung eintritt. Um festzustellen, ob sie Sättigung aus der Varianz in Modul B kommt, oder aber Ursachen in A (z.B. Wahl von Δt , Δx) hat, muss dort neben den „Tallies“ (Momenten $R = \langle g, \Psi \rangle$, siehe Kapitel 3) auch deren Varianz σ_R^2 in Modul B berechnet werden.

Bedingt durch die Monte Carlo Prozedur im Modul B ist die Konvergenzrate dort nicht abhängig von Diskretierungsparametern δt und/oder δx , und auch nicht durch Übergang von expliziten zu stärker impliziten Algorithmen veränderbar. Wegen des „Naturgesetzes“ der großen Zahlen bleibt im Modul B die Konvergenzrate immer proportional zu $\sqrt{1/CPU}$. Durch Optimierung des Algorithmus, Parallelisierung, etc. ist hieran nichts zu ändern. Allerdings kann natürlich der Proportionalitätsfaktor für jeden konkreten Anwendungsfall stark beeinflusst werden.
Man kann bei Monte Carlo Verfahren durch eine Vielzahl von sogenannten varianzreduzierenden Maßnahmen bei gleicher Rechenzeit kleinere statistische Varianzen erzielen.
Hierzu gibt es eine Fülle von Literatur /12/ , /13/ , /14/ , /15/ . Allerdings ist allgemein zu sagen, dass diese Methoden stark problemspezifisch sind. Eine Optimierung des Monte Carlo Teils kann nicht „hard wired“ in einen Code, der allgemein einsetzbar sein soll, implementiert werden, da die gleiche Optimierung bei anderen Anwendungen zu einer Verschlechterung anstatt zu einer Verbesserung der Performance führen kann
Die oben erwähnten varianzreduzierenden Maßnahmen laufen in der Regel auf ein Biassing plus Gewichtskorrektur hinaus, also auf eine vorsätzliche Verfälschung des physikalischen Modells, welches statistisch simuliert wird, mit einer nachfolgenden analytischen Korrektur der Ergebnisse derart, dass das Verfahren doch wieder gegen die korrekte Gleichung konvergiert.
Ferner ist bei starkem Biassing (und damit starken Schwankungen in den statistischen Korrekturgewichten) eine Fehlerabschätzung der statistischen Schätzungen durch sog. Konfidenzintervalle zunehmend unsicherer.
Aus diesem Grund werden bei den in dieser Arbeit diskutierten Algorithmen Optimierungen des Monte Carlo Moduls durch „weighted sampling“ weitgehend vermieden. Allerdings sind Monte Carlo Prozeduren in der Regel sehr effizient parallelisierbar, mit nahezu linearem Speedup. Aus diesem Grund werden in Kapitel 6 Aspekte der Parallelisierbarkeit des Moduls B besprochen.

Neben der Parallelisierung ist eine weitere Möglichkeit der Verbesserung der Effizienz von Modul B durch die Wahl der sogenannten Schätzfunktion (engl. „estimator“) gegeben. Drei mögliche Schätzfunktionen, die je nach den durch Modul A gelieferten makroskopischen Parametern des „Hintergrundmediums“ unterschiedliche (gegensätzliche) Performance haben, sind in Kapitel 3 angegeben und können mit dem in Kapitel 4 beschriebenen geometrischen Block wahlweise und je nach Bedarf verwendet werden. Die Frage der automatischen optimalen Kombination von Stoßschätzer und Weglängenschätzer ist vor allem dann wichtig, wenn die im Modul B behandelte Spezies selbst in „Gleichgewichtsnähe“ kommt, also fast

schon selbst durch einen Makro-Algorithmus gut beschreibbar ist. Solche gleichgewichtsnahen Prozesse sind mit Monte Carlo Verfahren intrinsisch ungünstig behandelbar. „Gleichgewichtsnähe“ bedeutet ja bei Transportproblemen, dass ein Elementarprozess mit seinem Umkehrprozess nahezu exakt bilanziert ist. Es müssen Differenzen zweier großer Zahlen („Quelle – Senke“) statistisch genau erfasst werden. Einige mögliche Verfahren werden in Kapitel 5 im Zusammenhang mit dem Datentransfer von Modul B nach Modul A vorgeschlagen. Häufig ist es der verfügbare Speicherplatz in B, der hier Grenzen setzt (siehe „correlation sampling“, abspeichern und wiederverwenden von Trajektorien in Kapitel 5 und „Ausblick“). In Kapitel 5 wird die Emission und, als Umkehrprozess, die Absorption als konkretes Beispiel verwendet. Im „optisch dicken“, d.h. gleichgewichtsnahen, Fall werden beide nahezu gleich („Schwarzer Körper“), aber auch diesen Grenzfall muss Modul B effizient behandeln können.

Ein letzter Aspekt, der sich immer wieder bei der praktischen Arbeit bei der Vernetzung ergab, liegt in der Visualisierung der Ergebnisse, siehe Kapitel 7. Schon bei zweidimensionalen Problemstellungen ist die graphische Darstellung der Ergebnisse eine Notwendigkeit. Diese Notwendigkeit ist noch viel stärker beim Umgang mit dreidimensionalen unstrukturierten Rechengebieten. Wenn die Zuordnung eines Resultats zu einer Zelle des Rechengebiet keinerlei Information mehr darüber bietet, wo im Rechenvolumen sich diese Zelle befindet und wo somit der Resultatwert angenommen wird, ist es um so wichtiger, geeignete Werkzeuge zur Visualisierung der Resultate und ebenso zur Diagnostik zur Verfügung zu haben.

3 Die mikroskopische Transportgleichung; Monte-Carlo Terminologie

In diesem Kapitel wird die Terminologie für den Modul B, den kinetischen „Mikro"-Teil des Algorithmus, eingeführt. Dies geschieht in Anlehnung an die Definitionen und Begriffsbildungen aus der Theorie von Monte Carlo Verfahren für lineare Transportprobleme und hier insbesondere an die in der Neutronentransporttheorie üblichen Bezeichnungsweisen, siehe z.B. /12/ . Diese konkrete Anwendung ist die historisch älteste, und zudem am anschaulichsten. Man kann sich hier vorstellen, dass die Bewegung und inneren Eigenschaften von Objekten (Teilchen, Testteilchen, oder „Historien") stochastisch formuliert und simuliert wird. Im Ortsraum (Koordinate r) bewegen sich diese Objekte im Rechengebiet des „Makro" Moduls A. Allerdings sind sie durch weitere mikroskopische Eigenschaften ausgezeichnet. Hierfür nimmt man im Folgenden exemplarisch die individuelle Geschwindigkeit v des „Teilchens", die also von der makroskopisch beobachtbaren Strömungsgeschwindigkeit V(r) (z.B. dem Wind) des Mediums zu unterscheiden ist. Aber eine andere Interpretation der zusätzlichen mikroskopischen Dimension (etwa die Frequenz einer turbulenten Fluktuation, oder Freiheitsgrade die den inneren chemischen Zustand kennzeichnen) ist immer möglich und in Anwendungen inzwischen ebenso häufig.

Man gibt zunächst die bekannte lineare (Boltzmann-) Transportgleichung an, die dann r und v als unabhängige Variablen hat.

Danach (siehe Kapitel 3.2) wird die Monte Carlo Prozedur zur Lösung solcher Gleichungen skizziert, und die bekanntesten Monte Carlo Schätzverfahren dafür eingeführt: „Weglängen und Stossschätzer" (engl.: tracklength und collision estimators). Kapitel 3.3 erläutert die geometrischen Aspekte bei der Implementierung dieser Verfahren, da diese die zentrale Rolle bei der Verkopplung des Monte Carlo Teils mit dem makroskopischen Code und die sich ergebenden informatischen und algorithmischen Aspekte spielt.

3.1 Die lineare Boltzmann Gleichung für die Verteilungsfunktion *f*

Der „Phasenraum" μ-Raum der unabhängigen Variablen des Mikro-Moduls ist in dieser Arbeit also exemplarisch der Phasenraum eines einzelnen individuellen Teilchens (Objekts). Die abhängige gesuchte Lösung des Mikro-Teils ist dann die Ein-Teilchenverteilungsfunktion f

$$f(\underline{r},\underline{v}) \quad \text{oder} \quad f(\underline{r},\nu,\underline{\Omega}) \quad \text{oder} \quad f(x), \tag{3}$$

wobei der Zustand x des μ-Raumes durch die Position r, den Geschwindigkeitsvektor v, (oder die Frequenz ν, die Richtung Ω, etc..) und gegebenenfalls weitere, hier nicht angegebene Variablen wie etwa einen Speziesindex i (i steht z.B. für die chemische Spezies H, D, T, D_2, DT, He, CH_n, oder für eine bestimmte scharfe Linie eines Strahlungsübergangs (siehe Kapitel 5)) und der Zeit t bei instationären Problemen beschrieben wird.

Zum Beispiel lautet dann die Teilchendichte $n_i(\underline{r})$ für Spezies i:

$$n_i(\underline{r},t) = \int d^3v\, f(\underline{r},\underline{v},i,t) \tag{4}$$

Die Boltzmanngleichung für *f* erhält man, wenn man annimmt, dass sich die Variablen v (und i, etc.) diskontinuierlich, bei sogenannten „Stößen" ändern. Im Falle von Neutronen sind dies

wirklich die Stossprozesse. r (und t) ändern sich stetig während des Fluges der Objekte von einem Stossort zum nächsten. Damit ergibt sich eine Gleichung, die differentiellen Charakter im Ortsraum hat (wie auch die Gleichungen des Moduls A) und zusätzlich einen Integraloperator im Geschwindigkeitsraum besitzt.

Die kinetische Boltzmann Gleichung für die Verteilungsfunktion f lautet dann

$$\left[\frac{\partial}{\partial t}+\underline{v}\cdot\nabla_{\underline{r}}+\frac{\underline{F}(\underline{r},\underline{v},t)}{m}\cdot\nabla_{\underline{v}}\right]f(\underline{r},\underline{v},t) = \iiint\sigma(\underline{v}',\underline{V}';\underline{v},\underline{V})|\underline{v}'-\underline{V}'|\,f(\underline{v}')f_b(\underline{V}')- - \iiint\sigma(\underline{v},\underline{V};\underline{v}',\underline{V}')|\underline{v}-\underline{V}|\,f(\underline{v})f_b(\underline{V}) \quad (5)$$

Die Integration wird über die Geschwindigkeiten $\underline{v}'$, $\underline{V}$ und $\underline{V}'$ erstreckt. $\sigma(\underline{v}',\underline{V}';\underline{v},\underline{V})$ ist der "Wirkungsquerschnitt" (ein Maß für die Wahrscheinlichkeit dieses Ereignisses).

Durch diesen Integralausdruck wird ein Prozess beschrieben, bei dem sich die gestrichenen Geschwindigkeiten (inneren Variablen) der beiden Stosspartner in die ungestrichenen verwandeln. Der Index b markiert die Spezies des zweiten Stosspartners, die entweder in Modul A makroskopisch beschrieben wird (dann ist Gleichung (5) linear), oder aber ebenfalls in Modul B auftritt. Im letzteren Fall ist die Gleichung (5) nichtlinear und muss iterativ durch sukzessive Linearisierung gelöst werden. Da aber Modul A und B ohnehin iterativ miteinander gekoppelt werden, bringt dies nichts grundsätzlich Neues. Im Rahmen dieser Arbeit wird daher angenommen, dass die Verteilung der Hintergrundspezies b aus dem Makro-Teil bekannt ist. f_b wird dann z.B. eine gegebene Gleichgewichtsverteilung (z.B. Maxwellverteilung) in den Parametern $n(r)$, $V(r)$ und $T(r)$ des Makro-Moduls sein. Die durch Gleichung (5) beschriebene mikroskopische Spezies bewegt sich also in einem Bad von anderen Spezies, welches durch makroskopische Parameter gegeben ist (z.B. ein Hintergrundplasma, in dem Photonen oder neutrale Atome „laufen", siehe Kapitel 5).
Weiter bedeutet *m* die Masse der Testteilchen und $\underline{F}(\underline{r},\underline{v},t)$ eine Kraft (etwa die Gravitation, die elektrische Kraft bei geladenen Testteilchen).
Der linke Teil der Gleichung beschreibt die Bewegung der mikroskopischen Objekte zwischen zwei Stössen (Ereignissen), die rechte Seite ist das sogenannte Stossintegral $\left.\frac{\partial f}{\partial t}\right|_b$ mit der Hintergrundspezies b.
Ohne die Kraft F bewegen sich die Teilchen einfach auf geraden Flugbahnen. Mit der Kraft F ist die Flugbahn nur noch numerisch zu berechnen, und kann dann näherungsweise als Polygonzug (z.B. durch Runge Kutta Integration der Bewegungsgleichungen) betrachtet werden. Diese Elementarschritte sind also auch dann wieder kleine gerade Flugstücke. Damit bringt also auch die Kraft F für den informatischen und algorithmischen Gesichtspunkt nichts prinzipiell Neues, wird ab jetzt also fortgelassen.

Im Rahmen dieser Arbeit werden nur noch Objekte betrachtet, die sich kraftfrei auf geraden Flugbahnen bewegen, bis zum nächsten Stossort. Neutronen in Reaktoren, Neutrinos, kosmische Strahlung, Photonen in astrophysikalischen und plasmaphysikalischen Anwendungen, Atome und Moleküle in hochverdünnten Gasströmungen, all dies sind Beispiele, wo diese Annahmen sogar in der Natur nahezu exakt erfüllt sind. In allen weiteren Fällen sei auf die immer noch bestehende mathematische Analogie der Prozesse und generischen mikroskopischen Gleichung verwiesen, um die Diskussion in Kapitel 4 bis 6 auf andere Anwendungen übertragen zu können.

Nach einigen formalen Manipulationen, und mit diesen eben geschilderten Vereinfachungen, entsteht die generische Gleichung für lineare Transportprobleme in Wissenschaft und Technik:

$$\left[\frac{\partial}{\partial t}+\underline{v}\cdot\nabla_{\underline{r}}\right]f(\underline{r},\underline{v},t)+\Sigma_t(\underline{r},\underline{v})|\underline{v}|f(\underline{v})=\int d^3v'\,C(\underline{v}'\to\underline{v})|\underline{v}'|f(\underline{v}') \qquad (6)$$

$\Sigma_t(\underline{r},\underline{v})$ (Dimension: 1/Länge) ist hier der Absorptionskoeffizient (engl.: extinction coefficient), der, anschaulich gesehen, beschreibt, auf welcher Länge sich ein Strahl aus identisch präparierten Testteilchen der hier beschriebenen Spezies auf Grund der ablaufenden Prozesse exponentiell verdünnt. In C sind Stoßprozesse mit Folgeteilchen nach dem Stoß in einer sehr kompakten Form zusammengefasst /12/ .

Oft ist die charakteristische Zeit für Transportphänomene im Modul B sehr viel kürzer als die charakteristischen Zeiten im Makro-Teil A (etwa: atomare Bewegung verglichen mit makroskopischer Bewegung einer Strömung. Man kann dann die explizite Zeitabhängigkeit in dieser Gleichung sogar auch noch vernachlässigen, und sich auf den Standpunkt stellen, dass das Mikro-System sich immer instantan im Gleichgewicht befindet, also während der zeitlichen Entwicklung des Systems A eine Folge von Gleichgewichtszuständen durchläuft. Ist sogar System A zeitunabhängig, wie bei den in dieser Arbeit diskutierten Anwendungen (Kapitel 4-6), dann tritt diese Frage erst gar nicht auf.

Aber selbst wenn eine explizite Zeitabhängigkeit im Mikro-System B zu berücksichtigen wäre, wie etwa bei einigen Problemen der Wärmeübertragung durch Strahlungsprozesse, so ist lediglich die Dimension des gemeinsamen Teils des Phasenraumes der Teile A und B um eins zu erhöhen (also: (r, t) anstatt (r)), für das generelle Konzept und die informatischen Aspekte bedeutet dies erneut nichts Neues. Fortan werden also alle Zeitabhängigkeiten, in A und in B, vernachlässigt.

Durch weitere formale Manipulationen (Integration entlang der Charakteristiken von Gleichung (6) ergibt sich die gleiche Transportgleichung, nun in der kompakten Form

$$\Psi(x)=S(x)+\int dx'\,\Psi(x')\cdot K(x'\to x). \qquad (7)$$

In Gleichung (7) sind x' und x die Zustände des Testteilchens (Objekts) an zwei auf einander folgenden Stößen. Das Integral $\int dx$ ist ein Integral über den Ortsraum, über den Geschwindigkeitsraum und eine Summation über alle Speziesindizes (allgemein: diskrete Phasenraumvariablen). Der "Übergangskern" K ist zerlegbar in einen Stoss- und einen Transportkern, d.h., C and T, wobei

$$K(\underline{r}',\underline{v}',i'\to\underline{r},\underline{v},i)=C(\underline{r}';\underline{v}',i'\to\underline{v},i)\cdot T(\underline{v},i;\underline{r}'\to\underline{r}). \qquad (8)$$

Der Kern C liefert die neuen Koordinaten $(\underline{v},i)$ wenn ein Teilchen der Spezies i' mit der Geschwindigkeit $\underline{v}'$ einen Stoss am Ort $\underline{r}'$ „erlebt“.
Der Kern T beschreibt die Bewegung der Testteilchen zwischen den Stossereignissen. D.h., T kann als Verteilung der Fluglänge l von einem Stoss zum nächsten interpretiert werden, da man, siehe oben, ohne wesentliche Beschränkung der Allgemeinheit von geraden Flugbahnen ausgehen kann.
Die Inhomogenität S in Gleichung (1) ist als Startverteilung (Anfangsverteilung), oder, in physikalischer Interpretation, als Quelle der Testteilchen zu interpretieren.

Diese Gleichung hat genau die Form der generischen Gleichung für Markov'sche Sprungprozesse. Damit ist klar, dass durch Simulation eines Sprungprozesses die lineare Transportgleichung (5) in der Tat gelöst werden kann. Mathematische Beweise der Exaktheit des Monte Carlo Kalküls für Boltzmanngleichungen beruhen auf diesem Zusammenhang. Aber schon eine direkte, intuitive Interpretation dieser Fredholm'schen Integralgleichung (7) reicht aus, um die folgenden Diskussionen in Kapitel 4-6 verstehen zu können.

Speziell in Mikro-Makro-Anwendungen, so wie in dieser Arbeit behandelt, ist eine vollständige Kenntnis der Lösung *f* (Gleichung (6)) oder Ψ (Gleichung (7)) nicht notwendig, ja sogar: nicht einmal verwertbar. Vom Prozess B können nur makroskopische Mittelwerte in Prozess A eingehen (man kann keinen Fingerhandschuh über einem Fausthandschuh anziehen). Deshalb reicht es aus, nur sogenannte Responses R (Momente) der kinetischen Lösung zu berechnen und von B nach A zu übergeben:

$$R = \langle \Psi | g_c \rangle = \int dx\, \Psi(x) \cdot g_c(x) \quad \left(= \langle f | g_t \rangle = \int dx\, f(x) \cdot g_t(x) \right) \tag{9}$$

wobei $g_c(x), g_t(x)$ beliebig vorgebbare „detector functions" (Gewichtsfunktionen) sind, deren explizite Form davon abhängt, welche Größen genau in Module A benötigt werden, aber auch davon, welche Schätzfunktionen (siehe Kapitel 3.2) im Modul B verwendet werden.

Relevant für diese Arbeit ist nur, das der Modul B in der Lage sein muss, beliebige Momente der Lösung *f* oder Ψ der kinetischen Gleichung zu berechnen, und zwar bei Momentbildung (Integration) über alle die unabhängigen Variablen (v,i), die im mikroskopischen Modell zusätzlich zu den makroskopischen unabhängigen Variablen (hier z.B. (r,t)) auftreten.

Da eine jede statistische Mittelung, also auch die, die im Monte Carlo Kalkül über viele Testteilchenhistorien auftritt, immer als Integral (Erwartungswert) angesehen werden kann /12/ , /9/ , /13/ , liefert ein Monte Carlo Algorithmus solche Momente in besonders natürlicher Weise: die Integration über die unabhängige Funktion f (oder Ψ) wird quasi automatisch ausgeführt, ohne das zuvor die Funktion f selbst bestimmt werden müsste. Allerdings erstreckt sich die Integration immer auch über einen Teil des Ortsraumes, etwa über die Raumzelle aus der Diskretisierung der Gleichungen des Moduls A. Deshalb sind die Monte Carlo Lösungen, die von B nach A übergeben werden, immer auch Integrale (Mittelwerte) über die Zellen des Ortsgitters, nicht etwa Punktschätzungen an den Gitterknoten.

3.2 Monte Carlo Lösung der Gleichung (1)

Eine statistische Lösung der Gleichung (5) bzw. (7) ist besonders natürlich und intuitiv einfach, denn insbesondere in der Form (7) ist die Gleichung ja bereits in wahrscheinlichkeitstheoretischer Sprache formuliert, oder zumindest so interpretierbar.

Eine zu Gleichung (7) gehörige diskrete Markovkette (ein stochastischer Sprungprozess) ist definiert, indem man *S* als Anfangsverteilung und $L = T \cdot C$ (Reihenfolge von *C* und *T* vertauscht gegenüber *K* in Gleichung (8)) als Übergangswahrscheinlichkeit wählt. Historien, auch "Pfade", oder engl.: "random walks" ω^n dieses Zufallsprozesses lassen sich dann am Rechner mittels Zufallszahlen erzeugen, gemäß $\omega^n = (x_0, x_1, x_2, \ldots, x_n)$, (wobei $x_j = x_a$ für alle $j \geq n$ und $x_i \neq x_a$ für alle $i < n$), und wobei x_n der erste Zustand x_a nach einem die Teilchentrajektorie beendenden Prozess (Absorption) war. x_0 ist der Anfangszustand der Kette, gesammelt aus der Startverteilung *S*. Mit anderen Worten: die echte Länge *n* der Kette ω^n ist selbst eine Zufallsvariable. Ein Algorithmus zum Erzeugen solcher Ketten am Rechner besteht im Konvertieren von gleichverteilten Zufallszahlen (siehe: Pseudozufalls-

zahlgeneratoren, /24/) $\xi_{i_1}, \xi_{i_2}, \ldots$ in Zufallszahlen mit den Verteilungen *S*, *T* und *C*. Nachdem *N* (in der Regel: mehrere Tausend bis Millionen) Ketten ω , $i = 1,2,\ldots,N$, erzeugt wurden, können die oben beschriebenen „Responsefunktionen“ (Momente der Lösung *f* oder Ψ als arithmetisches Mittel geeigneter Statistiken („Schätzfunktionen“) $\mathrm{X}(\omega)$, berechnet werden

$$R \approx \tilde{R} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{X}(\omega_i) \tag{10}$$

Eine mögliche Wahl eines solchen Schätzers $\mathrm{X}(\omega)$ ist der so genannte Stossschätzer („collision estimator“) X_c ,

$$\mathrm{X}_c(\omega_i^n) = \sum_{l=1}^{n} g_c(x_l) \cdot \prod_{j=1}^{l-1} \frac{c(x_j)}{(1 - p_a(x_j))} \tag{11}$$

Diese Schätzfunktion hat als statistischen Erwartungswert genau das gesuchte Moment *R* der Lösung (siehe Gleichung (10))

$$R = E(\mathrm{X}_c) = \int d(\omega)\mathrm{X}_c(\omega)h(\omega) \tag{12}$$

wobei hier $h(\omega)$ die Wahrscheinlichkeitsdichte für das Sammeln genau der Kette *ω* aus dem oben definierten Markovprozess ist. Mit anderen Worten: X_c ist ein erwartungstreuer (“unbiased”) Schätzer für *R*. In diesem präzisen mathematischen Sinne lösen Monte Carlo Codes also in der Tat eine lineare Transportgleichung, allgemeiner: eine Fredholm’sche Integralgleichung 2-ter Art.

Es gibt viele andere “Schätzer” (z. B. die "Weglängenschätzfunktion"), die ebenfalls erwartungstreu sind, deren höhere Momente (statistische Varianz) aber von X_c abweichen können. Anstatt einer Auswertung der Gewichtsfunktion $g_c(x)$ an den Stossorten, x_1 (wie es bei X_c der Fall war), gehen hier Linienintegrale einer anderen Gewichtsfunktion $g_t(x)$ längs der Trajektorie ein:

$$\mathrm{X}_t(\omega_i^n) = \sum_{l=0}^{n-1} \left\{ \int_{x_l}^{x_{l+1}} ds\, g_t(s) \right\} \cdot \prod_{j=1}^{l-1} \frac{c(x_j)}{(1 - p_a(x_j))} \tag{13}$$

aber wieder mit $R = E(\mathrm{X}_t) = E(\mathrm{X}_c)$.

Ein dritter Schätzer X_e ist besonders vorteilhaft, um auch sehr seltene Ereignisse noch statistisch präzise zu schätzen, also z.B. ein sehr tiefes Eindringen von den Objekten in ein stark absorbierendes Medium (so wie in Kapitel 4-6 in der Tat der Fall, oder bei Simulationsaufgaben zum Strahlenschutz).
Er entsteht aus X_t, indem man die Linienintegration, die in X_t auf das Wegstück von x_l nach x_{l+1} zwischen zwei Ereignissen beschränkt war, nun auf das ganze Liniensegment von x_l bis nach x_{end} ausdehnt. Hierbei ist x_{end} der naheste Punkt längs der bei x_l startenden Flugbahn, der auf dem Rand des Rechengebietes liegt. Dieser Schätzer heißt „conditional expectation estimator“ (kein vernünftiger deutscher Name scheint zu existieren), weil er auch noch erwartete Beiträge (analytisch, oder numerisch) mit hinzurechnet, die in der eigentlichen Monte Carlo Simulation gar nicht wirklich eingetreten sind.

$$\mathrm{X}_e\left(\omega_i^n\right)=\sum_{l=0}^{n-1}\left\{\int_{x_l}^{x_{end}} ds\, g_t(s)\cdot\exp\left(-\int_0^s ds'\,\Sigma_t(s')\right)\right\}\cdot\prod_{j=1}^{l-1}\frac{c\left(x_j\right)}{\left(1-p_a\left(x_j\right)\right)}, \qquad (14)$$

Mit diesem Schätzer wird aus einem reinen Monte Carlo Code eine Mischform zwischen statistischem und numerischen Algorithmus, wobei im Prinzip hier die Grenzen noch je nach Bedarf (und konkretem Anwendungsfall) zur einen oder anderen Seite verschiebbar sind.

Kapitel 4 beschreibt anhand eines konkreten Beispiels einen geometrischen Modul für einen 3D Monte Carlo Code innerhalb eines Mikro-Makro-Modells, der die notwendigen Informationen

- zur Erzeugung der Trajektorien und
- zum Auswerten all dieser drei Schätzfunktionen

erzeugt.
Damit kann man im Rechencode dann je nach Bedarf den einen oder anderen dieser Monte Carlo Algorithmen verwenden, ohne im geometrischen Kern des Codes etwas ändern zu müssen.

3.2.1 Effizienz (FOM)

Die Effizienz eines Monte Carlo Codes ist die Inverse der sog. “figure of merit" (FOM) des Algorithmus, definiert als

$$\mathrm{FOM}=\text{statistische Varianz}\cdot\left(\text{CPU - Kosten}\right). \qquad (15)$$

Man beachte, dass die FOM approximativ unabhängig von der Laufzeit ist, denn die Anzahl der erzeugten Teilchentrajektorien ist, wenn man den Overhead vernachlässigt, proportional zur CPU Zeit und umgekehrt proportional zur statistischen Varianz. Die FOM misst also die Effizienz des Algorithmus, bei jeder fest gegebenen Rechnerplattform. Häufig wird in Anwendungen jedoch anstatt FOM (aus Gleichung (15)) die Anzahl der simulierten „Testteilchen“ angegeben, manchmal zumindest noch bezogen auf die Rechenzeit und spezielle Rechneranlagen. Da jedoch die Varianz/Teilchen ($\sigma_N^2/N\equiv\sigma_1^2$) des Algorithmus im Prinzip je nach verwendeter Monte Carlo Prozedur irgendwo zwischen 0 und ∞ liegen kann, haben diese Angaben keinerlei Informationsgehalt bezüglich des Algorithmus und dienen ausschließlich psychologischen Zwecken, um beim Hörer/Leser nicht näher begründbaren Eindruck zu erzielen.

3.2.2 Sampling, Non-analog sampling

In dieser Arbeit werden die Prozeduren zum Erzeugen von Zufallszahlen nach gegebener Verteilung als bekannt /16/ vorausgesetzt, wenngleich im konkreten Anwendungsfall an dieser Stelle häufig erst geeignete Algorithmen entwickelt werden, weil die Standardprozeduren, die immer funktionieren, im Einzelfall extrem uneffektiv sein können. Im Rahmen dieser Arbeit geht man also davon aus, dass es möglich ist, die Startkoordinaten der Testobjekte aus der Inhomogenität S zufällig zu sammeln, ebenso die Fluglängen aus *T* und die Richtungsänderung aus *C*. Lediglich die sich speziell aus der Ankopplung des Monte Carlo Solvers an einen Modul A, der *S*, *T* und *C* aus einem Finite Elemente Algorithmus bereitstellt, ergebenden Prozeduren werden in Kapitel 4 diskutiert.

3.2.3 Stratified Source Sampling

Der in Kapitel 4-6 angewendete EIRENE Monte Carlo Code beruht auf einer „stratified source sampling“ Technik. Dieses Monte Carlo Konzept hat spezielle Auswirkung auf die Parallelisierung (Kapitel 6) und soll deshalb hier in allgemeiner Form erläutert werden.
Die „Methode der geschichteten Stichprobe" (also: stratified sampling) bedeutet bei Anwendung auf lineare Transportprobleme, dass die Quellverteilung (siehe Gleichung (7)) zerlegt werden kann in eine Summe von unabhängigen Teilquellen: $S = \sum_i S_i$. Man kann sich zum Beispiel vorstellen, dass zwei Gruppen von Testteilchen an zwei verschiedenen Rändern des Rechengebiets in das Volumen starten (siehe Kapitel 4.3.2). Die Lösung der Gleichung ergibt sich wegen der Linearität dann als lineare Überlagerung der partiellen Lösungen der Gleichung für jede Teilquelle („stratum“) S_i .
Wie bereits beschrieben, liefert der Monte Carlo Kalkül Schätzungen für lineare Funktionale ("Momente", "Responses")

$$\left\langle g, \Psi_S \right\rangle = \left\langle S, \Psi_g^* \right\rangle \tag{16}$$

Oben wurde bereits geschildert, dass *g* die problemspezifische Gewichtsfunktion ist („weighting function“), Ψ_S ist die Lösung der kinetischen Transportgleichung (Boltzmann Gleichung), in der *S* als inhomogener Quellterm auftrat. Ψ_g^* ist die Lösung der zugehörigen adjungierten Gleichung, mit *g* als Quellterm. In der obigen Gleichung sind also beim Übergang zum adjungierten Problem die Rollen von *g* und *S* vertauscht worden: die Teilchen laufen vom Detektor *g* zurück zur Quelle *S* (Fredholm'sche Alternative).

Dieses formale Argument zeigt: Monte Carlo Lösungen von Transportproblemen kann man auch ansehen als Spezialfall einer Monte Carlo Schätzung eines (hochdimensionalen) Integrals einer Funktion über eine Wahrscheinlichkeitsverteilung *S*.

Dann ist es offenkundig, dass man dieses Integral aufteilen kann in eine Summe von Integralen über Teilbereiche des Integrationsgebietes, d.h. durch eine Zerlegung der Quelle *S* in „strata”, Teilquellen $S_i(x)$ mit

$$\mathrm{S(x)} = \sum_i \mathrm{S_i}(x).$$

Diese Zerlegung in Teilbereiche beim stochastischen Integrieren ist das allgemeine Konzept der geschichteten Stichprobe und in der Literatur ausführlich beschrieben. Durch geschickte Aufteilung der CPU Ressourcen auf die Teilquellen lassen sich Monte Carlo Algorithmen relativ robust optimieren. Dies ist also anders als beim Einführen von Gewichten und dem Sammeln aus „nichtanalogen“ Verteilungen ohne das Risiko behaftet, dass das Verfahren in einzelnen Fällen auch verschlechtert werden könnte.

Man kann zum Beispiel allgemein zeigen, dass bei einer Zuordnung der CPU-Zeit zu den Sub-Quellen, die proportional zu den relativen Quellstärken $\int dx\, S_i$ geschieht, das Verfahren auf jeden Fall nicht schlechter werden kann als das Verfahren ohne diese Schichtung der Stichprobe, jedoch, im Einzelfall und je nach Aufteilung der Quelle in Unterquellen, durchaus effizienter. Die in Kapitel 6 beschriebenen Arbeiten zur Parallelisierung beruhen auf dem Versuch, diesen Aspekt auszunutzen.

3.3 Beschreibung der Geometrie in Monte Carlo Codes

Die vollständige geometrische Information für das Verfolgen von Testteilchen in einem Rechengitter und für das Auswerten der Schätzfunktionen in einem beliebigen Monte Carlo Transport Code lässt sich aus folgenden Elementaroperationen gewinnen:

Man betrachte ein dreidimensionales Rechengebiet, und eine Diskretisierung dieses Volumens durch ein Netz aus n Flächen S_i, $i = 1,2,...,n$. Jede einzelne dieser Flächen (Label *i*) möge definiert sein als Nullstellenmenge (Lösung) einer Menge von Λ_i Gleichungen:

$$f_\lambda^i(x,y,z) = 0, \qquad \lambda = 1,2,...,\Lambda_i,$$

und, optional, für jedes *λ* eine Untermenge von Π_λ Ungleichungen:

$$g_{\lambda,\pi}^i(x,y,z) \le 0, \qquad \pi = 1,2,...,\Pi_\lambda,$$

die sozusagen den Gültigkeitsbereich einer Fläche λ^i einschränken, z. B.
$x^2 + y^2 + z^2 - R^2 = 0, \quad x \le 0$ ist eine Halbkugeloberfläche.

Betrachte ein Testteilchen an der Position $\underline{r}_0 = (X0, Y0, Z0)$, das sich in die Richtung $\underline{\Omega}_0 = (VELX, VELY, VELZ)$ bewegt. Der geometrische Modul muss nun den nahesten Schnittpunkt $\underline{r}_1$ der geraden Linie

$$G(t): \underline{r}_0 + t \cdot \underline{\Omega}_0$$

unter allen möglichen Schnittpunkten mit den Flächen S_i identifizieren. Man versetzt dann das „Teilchen“ von $\underline{r}_0$ nach $\underline{r}_1$ und wiederholt den Block so lange, bis die akkumulierte Fluglänge die aus dem Transportkern *T* erwürfelte freie Weglänge übersteigt. Auf diese Weise können die Bahnen berechnet werden, und gleichzeitig sind die Bahnstücke innerhalb der einzelnen Zellen des Rechengebietes zum Auswerten der Schätzfunktionen bekannt. Ferner ist an jedem Bahnpunkt auch der Index der zugehörigen Zelle, sowie der eines bestimmten Stossereignisses, bekannt.
In EIRENE (Kapitel 4) sind *f* und *g* allgemeine algebraische Gleichungen der Form

$$h(x,y,z) = a_0 + a_1 \cdot x + a_2 \cdot y + a_3 \cdot z + a_4 \cdot x^2 + a_5 \cdot y^2 + a_6 \cdot z^2 + a_7 \cdot xy + a_8 \cdot xz + a_9 \cdot yz \qquad (17)$$

mit linearen (ebenen) Begrenzungsflächen als Spezialfall. In diesem Fall muss der geometrische Modul also nichts weiter als quadratische Gleichungen

$$t^2 + p \cdot t + q = 0$$

lösen, dies allerdings exakt und so effizient wie nur irgend möglich. Es sei hier darauf verwiesen, dass es prinzipiell exakte algebraische Algorithmen gibt bis zu Gleichungen 4ter Ordnung (Cardano'sche Formeln, /10/), aber keine mehr für algebraische Gleichungen höherer Ordnung (Satz von Abel, /11/).

Eine absolut konsistente Diskretisierung des Rechengebietes der Module A und B in Mikro-Makro Algorithmen müsste dem Rechnung tragen, d.h. es sollten auch bei Verwendung krummliniger Koordinatensysteme, wie in der Fluidmechanik üblich, nur solche Diskretisierungen verwendet werden, für die geeignete Algorithmen zur (exakten, algebraischen) Berechnung von den Schnittpunkten von Flugbahnen der mikroskopisch behandelten Objekte existieren.

Beim konkreten Beispiel FIDAP-EIRENE in Kapitel 4 war dies nicht der Fall. Jedenfalls scheinen hier keine algebraischen, geschlossenen Ausdrücke bis maximal 4ter Ordnung zu existieren. Aus diesem Grund mussten einerseits genaue, andererseits rechenzeiteffiziente Algorithmen speziell für dieses Mikro-Makro Problem entwickelt werden. Da aber die hier vorkommende Form der Diskretisierung auch allgemeiner eine grundsätzlich sehr häufig gewählte Form in Finite-Elemente Anwendungen ist, sind auch die in Kapitel 4 beschriebenen Algorithmen von allgemeiner Bedeutung.

4 Spezifische Algorithmen für die Gitterharmonisierung

In diesem und in den folgenden Kapiteln werden Algorithmen vorgestellt, die für die beispielhaft dargestellte Kopplung des makroskopischen FEM-Codes FIDAP und des mikroskopischen MC-Code EIRENE benötigt werden. Die Verfahren werden ausführlich erläutert. Es handelt sich um eine wichtige Konkretisierung des in Kapitel 3.3 beschriebenen allgemeinen geometrischen Moduls.
Der FORTRAN Quellcode mit den entsprechenden programmtechnischen Umsetzungen befindet sich im Anhang.

4.1 Die Geometrie

Eines der ersten Probleme, mit denen man sich bei Mikro-Makro-Problemen konfrontiert sieht, ist die unterschiedliche Beschreibung des Rechengebietes. Unterschiedliche Verfahren benötigen häufig verschiedene Ansätze zur Diskretisierung ihres Rechenbereichs.

So verwenden Finite-Elemente und auch Finite-Volumen Verfahren (/3/ , /1/) im allgemeinen Basiselemente verschiedenster Art zur Definition der Problemgeometrie. Solche Basiselemente können zum Beispiel in 2 dimensionalen Anwendungen Dreiecke, Vierecke mit geraden oder krummlinigen Begrenzungen, bzw. Tetraeder oder Hexaeder in 3D sein.
Die FEM-Verfahren arbeiten intern mit Standardelementen, die ein eigenes lokales Koordinatensystem besitzen, und mit Shapefunktionen, die die Abbildung zwischen der realen (kartesischen) Geometrie und den lokalen Koordinaten der entsprechenden Basiselemente festlegen.
Dabei ist zu beachten, dass offenbar in der Regel nur die Abbildung von den krummlinigen lokalen Koordinaten in ein globales, z.B. kartesisches System in expliziter Form angegeben ist, während die inverse Abbildung nur implizit bekannt ist. Für die Kopplung eines mikroskopischen Moduls B an solche FEM Codes erweist sich das ein signifikantes technisches Hindernis und erfordert die Entwicklung eigener spezielle Algorithmen in Modul B für diesen Anwendungsfall.

Mit Hilfe der gleichen Shapefunktion, die auch die geometrische Abbildung beschreibt, werden auch Profilinformationen der abhängigen Variablen (z. B. $T(\underline{r})$, $n(\underline{r})$, $V(\underline{r})$) innerhalb der Basiselemente codiert, siehe Kapitel 5.

Im konkreten Fall werden zur Beschreibung der Rechengeometrien im FIDAP-Code triquadratische Hexaeder, im folgenden auch Bricks genannt, verwendet. Diese Bricks sind verallgemeinerte Quader, deren Begrenzungsflächen durch quadratische Gleichungen in zwei der drei krummlinigen lokalen Koordinaten beschrieben werden können. Werden in einem konkreten Anwendungsfall einfachere Basiselemente verwendet, dann lassen sich die folgenden Ausführungen auch, in vereinfachter Form, übertragen.

Der allgemeine triquadratische Brick ist definiert durch 27 Punkte in den natürlichen, lokalen Koordinaten r, s, t mit $-1 \leq r, s, t \leq 1$, siehe Abbildung 3.
Der Koordinatenursprung befindet sich im Zentrum des Bricks. Die anderen 26 Punkte sind die acht Eckpunkte, die zwölf Kantenmittenpunkte und die sechs Seitenmittelpunkte des Quaders. Es befinden sich also neun Punkte auf jeder Seitenfläche des Bricks und drei Punkte auf jeder Kante.

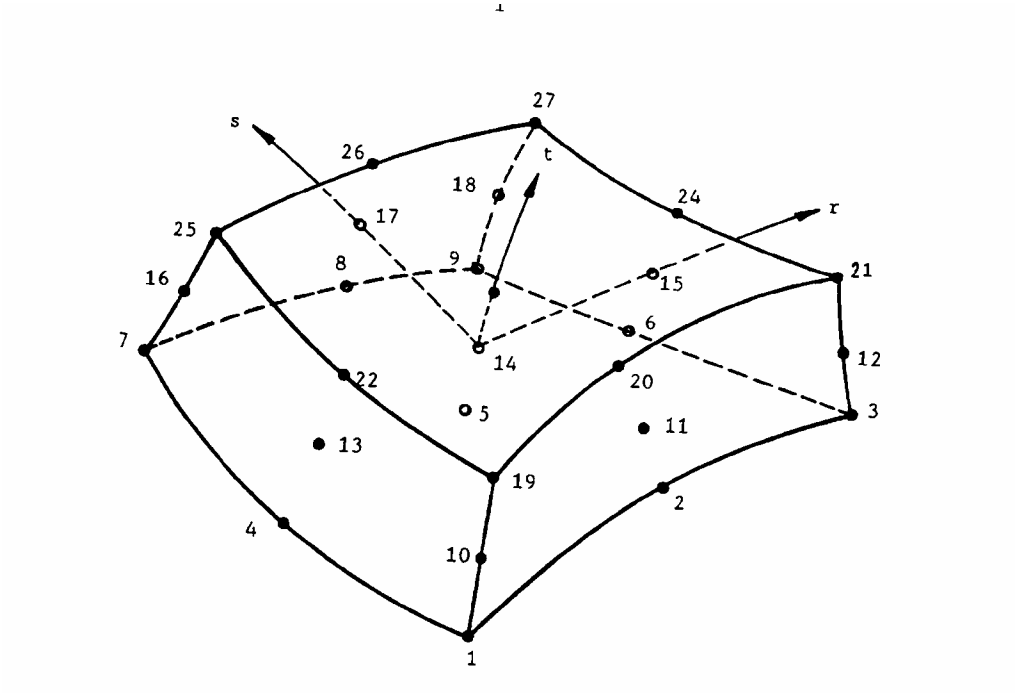


Abbildung 3 Darstellung eines triquadratischen Hexaeders

Die Beziehung zwischen den physikalischen Koordinaten (x, y, z) und den natürlichen Koordinaten (r, s, t) kann mit Hilfe der parametrisierten Transformation

$$x = \mathbf{N}^T \vec{x}\,, \quad y = \mathbf{N}^T \vec{y}\,, \quad z = \mathbf{N}^T \vec{z}\ ; \quad \mathbf{N}^T = \mathbf{N}^T(r, s, t)$$

definiert werden. Dabei sind $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{27} \end{pmatrix}, \vec{y} = \begin{pmatrix} y \\ \vdots \\ y_{27} \end{pmatrix}, \vec{z} = \begin{pmatrix} z \\ \vdots \\ z_{27} \end{pmatrix}$ Vektoren der Eckkoordinaten des Bricks und **N** ist ein 27er Vektor von Interpolationsfunktionen, die das Element in den natürlichen Koordinaten beschreiben. Diese Koordinatentransformation ist ein-eindeutig. Das heißt, es existiert eine Umkehrabbildung. Allerdings ist diese nicht einfach zugänglich.
Es gilt **N** = φ mit

$$\varphi = \begin{pmatrix} -\frac{1}{8}rst(1-r)(1-s)(1-t) \\ \frac{1}{4}st(1-r^2)(1-s)(1-t) \\ \frac{1}{8}rst(1+r)(1-s)(1-t) \\ \frac{1}{4}rt(1-r)(1-s^2)(1-t) \\ -\frac{1}{2}t(1-r^2)(1-s^2)(1-t) \\ -\frac{1}{4}rt(1+r)(1-s^2)(1-t) \\ \frac{1}{8}rst(1-r)(1+s)(1-t) \\ -\frac{1}{4}st(1-r^2)(1+s)(1-t) \\ -\frac{1}{8}rst(1+r)(1+s)(1-t) \\ \frac{1}{4}rs(1-r)(1-s)(1-t^2) \\ -\frac{1}{2}s(1-r^2)(1-s)(1-t^2) \\ -\frac{1}{4}rs(1+r)(1-s)(1-t^2) \\ -\frac{1}{2}r(1-r)(1-s^2)(1-t^2) \\ (1-r^2)(1-s^2)(1-t^2) \\ \frac{1}{2}r(1+r)(1-s^2)(1-t^2) \\ -\frac{1}{4}rs(1-r)(1+s)(1-t^2) \\ \frac{1}{2}s(1-r^2)(1+s)(1-t^2) \\ \frac{1}{4}rs(1+r)(1+s)(1-t^2) \\ \frac{1}{8}rst(1-r)(1-s)(1+t) \\ -\frac{1}{4}st(1-r^2)(1-s)(1+t) \\ -\frac{1}{8}rst(1+r)(1-s)(1+t) \\ -\frac{1}{4}rt(1-r)(1-s^2)(1+t) \\ \frac{1}{2}t(1-r^2)(1-s^2)(1+t) \\ \frac{1}{4}rt(1+r)(1-s^2)(1+t) \\ -\frac{1}{8}rst(1-r)(1+s)(1+t) \\ \frac{1}{4}st(1-r^2)(1+s)(1+t) \\ \frac{1}{8}rst(1+r)(1+s)(1+t) \end{pmatrix} \tag{18}$$

Die kartesischen Koordinaten (x, y, z) lassen sich somit als Funktionen der natürlichen, lokalen Koordinaten r, s und t schreiben:

$$x = X(r,s,t), \qquad y = Y(r,s,t), \qquad z = Z(r,s,t).$$

Dabei enthalten die Funktionen X,Y,Z algebraische Terme bis zu 6ter Ordnung, aber nur maximal quadratisch in jeder der Koordinaten r, s und t. Begrenzungsflächen, z.B. gegeben durch $t=t_0$ in krummlinigen Koordinaten, beinhalten algebraische Terme bis zu 4ter Ordnung in r und s, höchstens aber wieder nur quadratisch in jeder einzelnen dieser beiden Variablen.

Wie man sich leicht vorstellen kann, sind solche triquadratischen Bricks keine der Standard-gitteroptionen des EIRENE-Codes. Sie sind auch für die Teilchenverfolgung im EIRENE-Code denkbar ungeeignet, da bei der Verfolgung der Teilchen fortlaufend Schnittpunkte der geraden Teilchenbahnen mit den Rändern der Gitterzellen berechnet werden müssen. Hierfür müssten die Seiten der Bricks in kartesischen Koordinaten beschrieben werden, was nicht ohne weiteres möglich ist, da die Umkehrabbildung φ^{-1} von φ nicht einfach zugänglich ist (wenn sie überhaupt algebraisch geschlossen angebbar ist). Zudem würde das Aufstellen der Umkehrabbildung vermutlich ebenfalls zu Gleichungen höherer Ordnung führen.
Schnittpunkte von geraden Flugbahnen $\vec{x}(time) = \vec{x}_0 + time \cdot \vec{v}$ mit diesen Flächen lassen sich algebraisch exakt vermutlich nur in Ausnahmefällen berechnen. Die Existenz effizienter Algorithmen für solche Probleme wären für Mikro-Makro Codes generell sehr nützlich.
Numerische Verfahren zur Nullstellenbestimmung sind andererseits in diesem Zusammenhang zu langsam und zu ungenau.

Aus diesen Gründen erscheint es sinnvoll, die triquadratischen Bricks im EIRENE-Code durch lineare Geometrieelemente anzunähern. Da es sich um dreidimensionale Geometrien handelt, bieten sich Tetraeder als geometrische Grundform an. Diese wurden zwar bis dahin auch nicht von EIRENE unterstützt, konnten aber mit deutlich geringerem Aufwand implementiert werden.
Dabei ist ein Tetraeder durch die kartesischen Koordinaten seiner vier Eckpunkte definiert. Die ersten drei Punkte bilden die Grundfläche des Tetraeders, der vierte Punkt befindet sich oberhalb der Grundfläche.

4.1.1 Überlegungen zur Zerlegung von triquadratischen Bricks in Tetraeder

Wie kann man nun einen triquadratischen Brick in Tetraeder zerlegen?
Vernachlässigt man die Seiten- und Kantenkrümmung des Bricks, so erhält man einen einfachen Quader. Dieser lässt sich auf verschiedene Arten in 6 Tetraeder zerlegen. Eine mögliche Zerlegung ist in Abbildung 4 dargestellt:

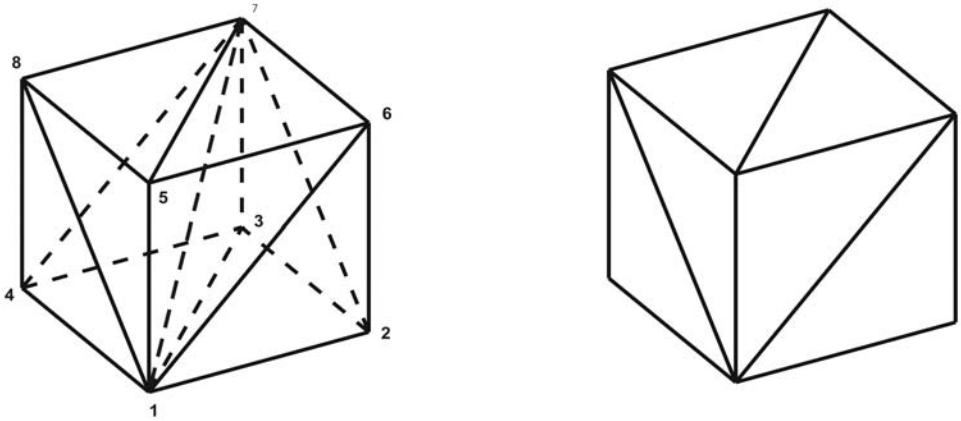


Abbildung 4 eine von mehreren möglichen Zerlegungen eines Quaders in sechs Tetraeder

Alle diese Zerlegungen haben aber einen gravierenden Nachteil: die Seiten des Quaders werden unterschiedlich geteilt. So wird in Abbildung 4 die Seite 1-2-6-5 durch die Diagonale vom linken unteren Eckpunkt zum rechten oberen Punkt in zwei Dreieck geteilt, wohingegen die Seite 4-1-5-8 durch die Diagonale vom oberen linken Eckpunkt zum unteren rechten Eckpunkt zerlegt wird. Zur Erleichterung der Teilchenverfolgung wird aber im EIRENE-Code erwartet, dass eine 1 zu 1 Beziehung zwischen benachbarten Zellen besteht, d.h. eine Seite einer Zelle muss genau an eine Seite der Nachbarzelle grenzen. Diese 1 zu 1 Zuordnung der Nachbarseiten ist vom Prinzip her nicht zwingend notwendig, aber das Auffinden der Nachbarzelle, in die ein Teilchen wechselt, ist bedeutend einfacher und somit auch schneller, wenn nur eine Nachbarzelle in Frage kommt.

Unter dieser Voraussetzung muss die oben vorgestellt Zerlegung in einem unstrukturierten Gitter, in dem a priori unbekannt ist, welche Bricks mit welchen Seiten aneinandergrenzen, zu Problemen führen.

Zur Umgehung dieser Probleme ist es notwendig, die Zerlegung der Bricks so zu wählen, dass alle Seitenflächen des Bricks auf gleiche Art in Dreiecke unterteilt werden. Zieht man zudem in Betracht, dass die Zerlegung die Krümmung des Bricks bestmöglich nachbilden soll, so ergeben sich folgende mögliche Zerlegungen der Bricks in Tetraeder:

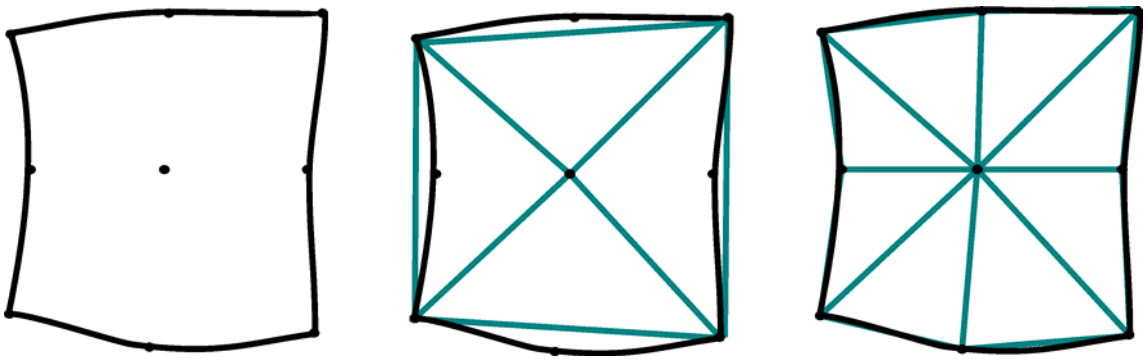


Abbildung 5 mögliche Zerlegungen einer Brickseite in vier bzw. acht Dreiecke

Man zerlegt jede Seite des Bricks mit Hilfe der neun gegebenen Punkte auf der Fläche in entweder 4 oder 8 Dreiecke. Jedes Dreieck bildet die Grundfläche eines Tetraeders. Als jeweils vierten Punkt des Tetraeders oberhalb der Grundfläche wählt man den Punkt im Zentrum des Bricks. Man erhält eine aus 24 bzw. 48 Tetraedern bestehende Zerlegung, die jeweils nur die 27 gegebenen Eckpunkte des Bricks zur Definition der Tetraeder verwendet.

Wie man an Abbildung 5 leicht ablesen kann, ist die Zerlegung einer gekrümmten Quaderseite in 4 Dreiecke deutlich ungenauer als die Zerlegung in 8 Dreiecke. Die Information über die ursprüngliche Krümmung der Quaderseite geht fast vollständig verloren. Daher verbleibt als einzig sinnvolle Alternative die Zerlegung der triquadratischen Bricks in jeweils mindestens n_t = 48 Tetraeder. Da einerseits dies im konkreten Anwendungsfall bereits zu Problemen mit der Speicherplatzverwaltung führt (siehe Kapitel 5.5), und andererseits eine weitere Zerlegung von Tetraedern in „Untertetraeder“ algorithmisch trivial ist, wird nur genau dieser Fall mit n_t=48 weiter behandelt.

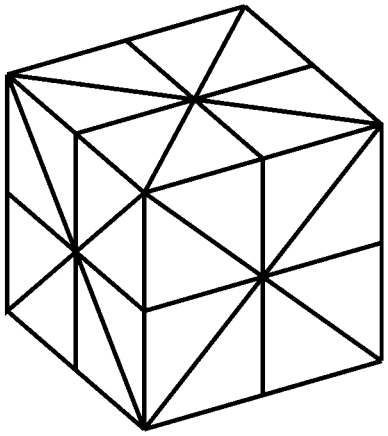


Abbildung 6 Blick auf einen in 48 Tetraeder zerlegten Quader. Jede Seite ist in 8 Dreiecke zerlegt. Alle 48 Tetraeder treffen sich im Mittelpunkt des Bricks.

Durch diese Art der Zerlegung ist sichergestellt, dass die Seiten zweier benachbarter Bricks die gleiche Zerlegung besitzen und die daraus resultierenden Dreiecke deckungsgleich sind. Gleichzeitig wird die von FIDAP verwendete Geometrie bestmöglich angenähert.

Nachdem jetzt die Art der Zerlegung festgelegt ist, kann mit der Umsetzung in EIRENE begonnen werden. Dazu muss berücksichtigt werden, wie EIRENE eine Tetraedergeometrie definiert. Zur Darstellung einer aus Tetraedern zusammengesetzten Geometrie verwendet der EIRENE-Code folgende Informationen:

- eine Liste aller Koordinaten des Rechengebiets. Diese werden in den Vektoren XTETRA, YTETRA, ZTETRA gespeichert.
- die Zuordnung der Koordinaten zu den Tetraedern, d.h. die Nummern der Koordinaten der 4 Eckpunkte werden im Feld NTECK abgelegt.
- Informationen über die Nachbarschaftsbeziehungen der Tetraeder untereinander. Das Feld NTBAR erwartet zu jeder Seite eines Tetraeders die Nummer seines Nachbartetraeders und NTSEITE liefert die zugehörige Seitennummer des Nachbartetraeders.
- Informationen über spezielle Randbedingungen (z. B. „äußere Berandung des Rechengebiets“) auf den Tetraederflächen werden in INMTIT abgelegt.

Diese Felder müssen aus den vom FIDAP-Code gelieferten Informationen aufgebaut werden.

4.1.2 Vom FIDAP-Code gelieferte Geometriedaten

Der Makro-Code (FIDAP) erzeugt eine Datei mit den vom Mikro-Code EIRENE benötigten Geometriedaten. Es handelt sich dabei um den sogenannten „FIDAP Neutral File.
Der „FIDAP Neutral File“ enthält die vollständige Beschreibung der verwendeten Geometrie und besitzt folgenden Aufbau (wichtige Schlüsselwort wurden gelb unterlegt):

```
   NO. OF NODES    NO. ELEMENTS NO. ELT GROUPS          NDFCD          NDFVL
          23523            3876              31              3              3
                                      .
                                      .
NODAL COORDINATES                     .
         1   3.9500000000e+000   2.0000000000e-001   0.0000000000e+000
         2   3.9500000000e+000  -2.0000000000e-001   2.4492127076e-017
         3   3.9500000000e+000   1.8477590650e-001   7.6536686473e-002
                                      .
                                      .
                                      .
                                      .
     23521   5.8652381395e+000  -1.3757402239e+000   5.6800330049e-001
     23522   5.8659534488e+000  -1.2142803417e+000   8.0894752120e-001
     23523   5.8646670086e+000  -1.4327453934e+000   2.8400165025e-001
BOUNDARY CONDITIONS
         0         0         0     0.0
ELEMENT GROUPS
GROUP:       1 ELEMENTS:    884 NODES:      27 GEOMETRY:   3 TYPE:   3
ENTITY NAME:   lte
       1     49     65      2    107    109     17     66    108      9
            630    634    300   2711   2713   1485   2707   2710   1483
            418    633    299   2709   2712   1484   2051   2708   1371
       2    418    633    299   2709   2712   1484   2051   2708   1371
            677    681    301   2718   2720   1508   2714   2717   1506
            419    680    298   2716   2719   1507   2092   2715   1372
                                      .
                                      .
                                      .
                                      .
    2628  12681  23503  12715  23073  23521  23013  12651  18958  12652
          20811  23507  20968  23077  23523  23017  18911  18965  18964
          10758  11416  10792  11390  11418  11386  10728  11148  10729
GROUP:       3 ELEMENTS:     52 NODES:       9 GEOMETRY:   1 TYPE:  17
ENTITY NAME:   anodtip
    2629      2     65     49    107     66    108      9     17    109
    2630     49     64     48    110     67    111     66    107    112
    2631     48     63     47    113     68    114     67    110    115
    2632     47     62     46    116     69    117     68    113    118
                                      .
                                      .
                                      .
                                      .
    3831   9728   9773   9730   9789   9636   9644   9637   9787   9790
    3832   9730   9775   8288   8290   8283   9643   9636   9789   9791
GROUP:      21 ELEMENTS:      8 NODES:       3 GEOMETRY:  0 TYPE:  17
ENTITY NAME:   azlinw
    3833    8207    9135    9128
    3834    9128    9136    9129
    3835    9129    9137    9130
```

Nach einigen Headerzeilen findet man in der Datei die Information über die Anzahl der Punkte und der Elemente, die in der Geometrie vorhanden sind. Nach der Überschrift „NODAL COORDINATES“ folgen dann die kartesischen Koordinaten der Knotenpunkte hier z.B. in mm. Der EIRENE-Code arbeitet allerdings nur in relativen geometrischen Einheiten, hier könnten also genauso nm, km, Lichtjahre, etc. stehen.
Anschließend werden die Elemente, eingeteilt in Elementgruppen, beschrieben. Jede Elementgruppe beginnt mit einer Headerzeile, die Informationen über die Anzahl der Elemente in dieser Gruppe und über die Anzahl der Knoten pro Element enthält. Ferner wird jeder Elementgruppe ein Name zugeordnet. Dann folgt die Beschreibung der Elemente. Jedes Ele-

ment wird durch eine fortlaufende Nummer gekennzeichnet und durch die Koordinatennummern der zu diesem Element gehörigen Knoten beschrieben.

Bei den hier verwendeten konkreten Anwendungsfällen finden sich im „FIDAP Neutral File“ drei unterschiedliche Arten von Elementen. Es gibt Elemente mit 27 Knoten, dabei handelt es sich um die oben beschriebenen triquadratischen Bricks. Dann gibt es Elemente mit 9 Knoten. Diese beschreiben die Außenflächen der Geometrie. Die Elemente sind identisch mit den Seiten der Bricks, die die Außenwand des Rechengebiet darstellen. Schließlich gibt es noch Elemente mit nur 3 Knotenpunkten. Dabei handelt es sich um äußere Kanten. Diese Kantenelemente werden im Folgenden nicht verwendet. Die Wahrscheinlichkeit, dass ein Testteilchen in 3D mit statistisch erwürfelten Flugbahnen genau ein Element der Dimension 1 (Linie) oder 0 (Punkt) trifft ist 0. Es gibt im „Mikro“-Modul also keine punkt- oder liniengemittelten Schätzer.

4.1.3 Das Erzeugen der Tetraedergeometrie

Die Anzahl und die Liste der kartesischen Koordinaten liefert FIDAP im „FIDAP Neutral File“. Sie können einfach eingelesen und von mm- in cm-Angaben umgerechnet werden.
Alle weiteren Informationen hingegen müssen aus den Beschreibungen der triquadratischen Bricks gewonnen werden.

Zum Erzeugen der Elementzuordnungen wurde zunächst ein Papiermodell eines in Tetraeder zerlegten Quaders (siehe Abbildung 7) erstellt, das sich im Folgenden als sehr nützlich erwies.

Abbildung 7 Papiermodell eines in 48 Tetraeder zerlegten Quaders

Durch Nummerieren der Eckpunkte des Papierquaders analog zur Nummerierung der Eckpunkte der Bricks in FIDAP (vergleiche Abbildung 3) kann die Zerlegung des Bricks in Tetraeder direkt vom Modell abgelesen werden. Dabei wird jede Quaderseite, beginnend mit dem Dreieck links unten unterhalb der Diagonalen, entgegen dem Uhrzeigersinn durchlaufen. Jedes Dreieck wird zur Grundfläche eines Tetraeders. Die Eckpunkte des Dreiecks werden ebenfalls entgegen dem Uhrzeigersinn durchlaufen. Der vierte Punkt eines jeden Tetraeders ist der Mittelpunkt des Bricks.
Sei also NTET die Anzahl der schon vorhandenen Tetraeder und INDCO(1:27) enthalte die Nummern der Koordinaten des aktuell zu bearbeitenden Bricks, dann beschreibt

```
NTECK(1,NTET+1) = INDCO(1)
NTECK(2,NTET+1) = INDCO(2)
NTECK(3,NTET+1) = INDCO(11)
NTECK(4,NTET+1) = INDCO(14)
```

die Elementzuordnung des ersten Tetraeders, der aus dem zu zerlegenden Brick entsteht. Dabei ist INDCO(14) die Nummer des Punktes im Zentrums des Bricks. Analog verfährt man für die restlichen 47 Tetraeder dieses Bricks.

Da dieser Vorgang keine weiteren Erkenntnisse bietet, wurde auf die weitere Ausführung an dieser Stelle verzichtet. Der vollständige Programmcode für die Zerlegung eines Quaders in Tetraeder befindet sich in der PROZEDUR MAKE_TETRA_48, die im Anhang wiedergegeben ist.

Ein in Tetraeder zerlegtes Rechengitter einer Hochdruckplasmalampe aus der konkreten Anwendung zeigt Abbildung 8. Man erkennt gut die beiden Symmetrieebenen der Lampe und die Bereiche mit unterschiedlich feiner Diskretisierung.

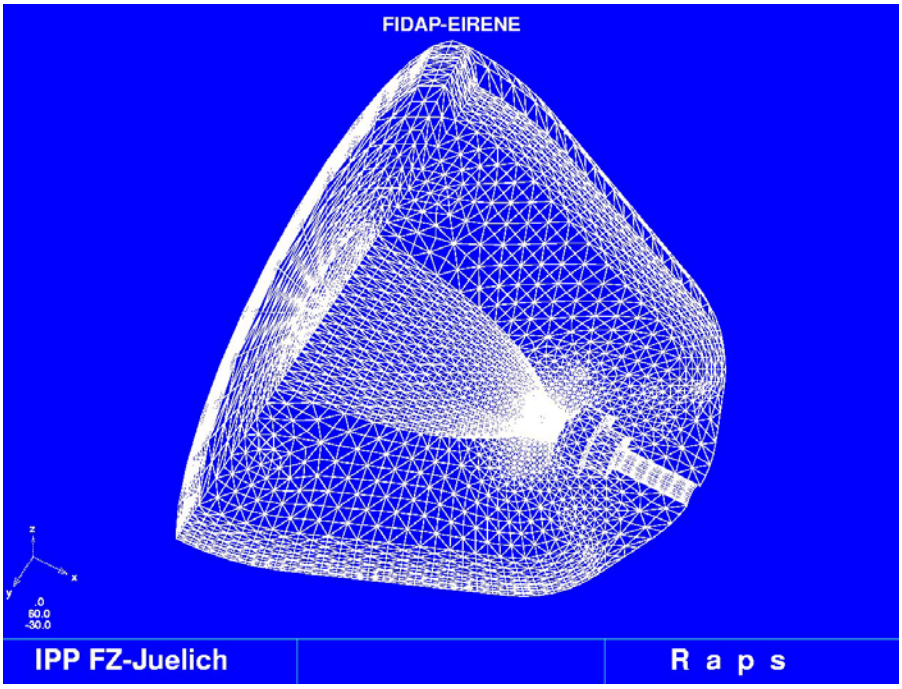


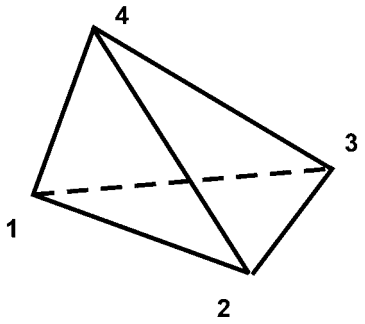
Abbildung 8 Rechengitter für eine Hochdruckplasmalampe

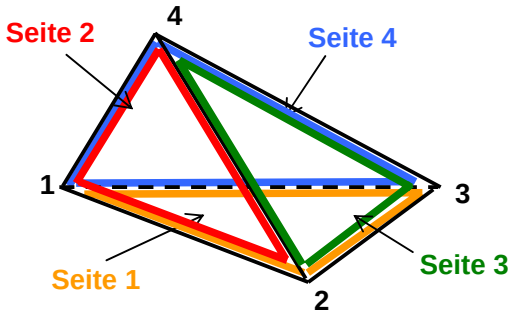
4.1.4 Aufbau der Nachbarschaftsbeziehungen

Zur Bereitstellung der übrigen benötigten Informationen müssen erst noch einige Festlegungen getroffen werden:

Ein Tetraeder ist definiert durch seine vier Eckpunkte.

Abbildung 9 Definition eines Tetraeders





Seine Seiten werden durch Dreiecke beschrieben. Seite 1 ist gegeben durch das Dreieck 1-2-3. Seite 2 ist das Dreieck 1-4-2, Seite 3 ist gegeben durch 2-4-3 und das Dreieck 3-4-1 ist Seite 4.

Jede Seite eines Tetraeders kann einen Nachbartetraeder besitzen. Ebenso kann für jede Seite eines Tetraeders eine Randbedingung definiert sein. Besitzt eine Tetraederseite aber **keinen** Nachbarn, so **muss** eine Randbedingung für diese Seite existieren, da diese Tetraederseite zur äußeren Berandung des Rechengitters gehört.

Ein Teil der benötigten Nachbarschaftsbeziehungen können ebenfalls direkt aus dem Papiermodell abgelesen werden.
Die Beziehungen der Tetraeder innerhalb eines Bricks sind durch die Zerlegung und die beschriebenen Seitendefinitionen festgelegt. D.h. die Nachbarn der Seiten 2 bis 4 eines Tetraeders können direkt bestimmt werden.
Dabei grenzt die Seite 2 eines Tetraeders I immer an die Seite 2 des benachbarten Tetraeders J1. Seite 3 des Tetraeders I stößt immer an die Seite 4 seines Nachbarn J2 und Seite 4 von I ist benachbart zur Seite 3 von Tetraeder J3. Aus dem Modell kann man nun die Nummern der benachbarten Tetraeder einfach ablesen. So ist zum Beispiel Tetraeder 1 benachbart zu den Tetraedern 34, 2 und 8.
Die Nachbarschaftsbeziehungen der Tetraeder innerhalb eines Bricks werden zusammen mit der Zerlegung selbst in PROZEDUR MAKE_TETRA_48 gesetzt. Dabei wird auch gleich geprüft, ob der Tetraeder zu einer Ebene oder einer Gerade degeneriert ist. Dies ist der Fall, wenn mindestens zwei Eckpunkte des Tetraeders gleich sind. Degenerierte Tetraeder haben keine Nachbarn. Für sie werden NTBAR und NTSEITE auf -1 gesetzt.
Zusätzlich wird zu jedem Eckpunkt eines Tetraeders eine Liste COORTET aufgebaut, die die Nummern aller Tetraeder enthält, die an diesen Eckpunkt grenzen.
PROZEDUR MAKE_TETRA_48 ist im Anhang aufgeführt.

Die Nachbarschaftsbeziehungen zu den Tetraedern außerhalb des Bricks können erst ermittelt werden, nachdem sämtliche Bricks in Tetraeder zerlegt wurden.

Dazu werden in der PROZEDUR SUCHE_NACHBARN alle Seiten IS aller Tetraeder ITET darauf überprüft, ob ihr Nachbar unbekannt ist (NTBAR(IS,ITET) .EQ. 0). Ist dies der Fall, so werden alle Tetraeder auf eine mögliche Nachbarschaft überprüft, die an den ersten Eckpunkt der Seite angrenzen. Auch bei diesen Tetraedern werden nur jene Seiten getestet, deren Nachbar noch nicht bekannt ist. Ist also ein Tetraeder JTET gefunden, der an den ersten Eckpunkt der Seite IS des Tetraeders ITET angrenzt und dessen Seite JS noch keinen Nachbarn hat, so müssen die Eckpunkte der beiden Dreiecke auf Gleichheit überprüft werden. Nur wenn die beiden Dreiecke die gleichen Eckpunkte besitzen, sind die zu einander gehörigen Nachbarn gefunden.

Dazu müssen alle möglichen Kombinationen der Eckpunkte verglichen werden, denn die Reihenfolge der Punktnummern kann für beide Dreiecke unterschiedlich sein. Für den Test werden die Nummern der Eckpunkte der Dreiecke in zwei Vektoren IP und JP gespeichert.

Anschließend kann in einer Doppelschleife verglichen werden, ob alle Eckpunkte des ersten Dreiecks im zweiten Dreieck vorhanden sind. Sobald ein Eckpunkt des ersten Dreiecks nicht im zweiten Dreieck gefunden werden kann, wird die Untersuchung dieser beiden Dreiecke abgebrochen und mit dem nächsten möglichen Dreieck fortgefahren. Stimmen hingegen alle Eckpunkte überein, so können die Nachbarschaftsbeziehungen für beide Dreiecke gesetzt werden. Es ist

NTBAR(IS,ITET) = JTET
NTSEITE(IS,ITET) = JS
NTBAR(JS,JTET) = ITET
NTSEITE(JS,JTET) = IS .

Anschließend wird analog mit der Seite IS+1 des Tetraeders ITET verfahren.

Der Programmtext der Prozedur SUCHE_NACHBARN kann im Anhang nachgelesen werden.

4.1.5 Setzen der Randbedingungen

Der letzte wichtige Punkt im Geometrieteil der Kopplung der beiden Codes FIDAP und EIRENE ist das Auffinden der Randflächen bzw. das Setzen von Randbedingungen auf den Geometrierändern. Während der Teilchenverfolgung muss zu jedem Zeitpunkt bekannt sein, ob sich das Teilchen im Rechenvolumen befindet, oder ob der Rand des Rechengebiets erreicht wurde. Daher kann jedem Gitterstück ein Flag INMTIT zugeordnet werden, das über die Eigenschaften des Gitterstücks Auskunft gibt. Für transparente Gitterstücke im Innern der Geometrie ist INMTIT=0. Aber auf den Randflächen des Rechengebiets muss INMTIT ≠ 0 gelten. Daher ist es notwendig, die Randflächen des Gitters zu identifizieren und mit Randbedingungen zu verknüpfen. An den Rändern müssen die makroskopischen Randbedingungen für Gleichung (1) in konsistente mikroskopische Randbedingungen für Gleichung (2) übersetzt werden. Randbedingung in Gleichung (2) bedeutet bei einer Monte Carlo Lösung, anschaulich, dass man den Testteilchen mitteilen muss, wie sie sich verhalten sollen, wenn sie auf solche Ränder treffen (z.B. Absorption, Spiegelreflexion oder komplexere „Reflexionsmodelle“).

Wie bereits in Kapitel 4.1.2 erwähnt, enthält der „FIDAP Neutral File“ mehrere Elementgruppen, die unterschiedliche Elementarten beschreiben. Die Elemente, die durch 27 Punkte gegeben sind, beschreiben die triquadratischen Bricks. Andere Elementgruppen bestehen aus Elementen mit neun Punkten. Dabei handelt es sich um Flächenelemente, genauer um Vierecke mit quadratisch gekrümmten Rändern. Dies sind die Flächen, die die Ränder der FIDAP-Geometrie beschreiben. Sie sind identisch mit den Außenflächen der am Rand befindlichen triquadratischen Bricks.
Zusammengehörige Randbereiche sind in der gleichen Elementgruppe zusammengefasst. Die verschiedenen Elementgruppen können durch ihre Namen identifiziert werden. Im EIRENE-Inputfile werden diese Namen mit den entsprechenden Reflektionsmodellen, die die zugehörigen Randbedingungen beschreiben, verknüpft.

Um die Randflächen der Tetraedergeometrie zu bestimmen, muss während der Verarbeitung der Bricks geprüft werden, ob sich eine oder mehrere Seiten des Bricks auf dem Rand des Rechenvolumens befinden. Dazu ist festzustellen, ob es ein Flächenelement gibt, das mit einer Brickseite identisch ist.

Man kann sich nun überlegen, dass sich zwei benachbarte Flächenelemente die drei Punkte auf der gemeinsamen Seitenwand teilen. D.h. die Randpunkte eines Flächenelements können daher in jeweils zwei Flächenelementen und in mindestens zwei Bricks vorkommen. Der

Mittelpunkt eines Flächenelements hingegen ist eindeutig. Er kommt nur in diesem einen Flächenelement und in dem zugehörigen Brick vor.
Zur Identifikation von Randseiten der Bricks genügt es also, die sechs Seitenmittelpunkte der Quaderseiten mit den Mittelpunkten der Flächenelemente zu vergleichen. Ist eine Übereinstimmung gefunden, so gehört die entsprechende Quaderseite zum Rand. Die acht Tetraeder, die aus dieser Seite gebildet werden, haben an ihrer Seite 1 keinen Nachbarn und erhalten die Kennung des Reflektionsmodells, das mit dem Flächenelement verknüpft ist.

Nachdem alle Nachbarschaftsbeziehungen und alle Reflexionsmodelle gesetzt sind, kann ein einfacher Konsistenztest durchgeführt werden. Dabei wird geprüft, ob alle Seiten der Tetraeder entweder einen Nachbarn oder ein Reflektionsmodell besitzen. Ist dies nicht der Fall, so liegt definitiv ein Fehler vor.

4.2 Teilchenverfolgung auf Tetraedergittern

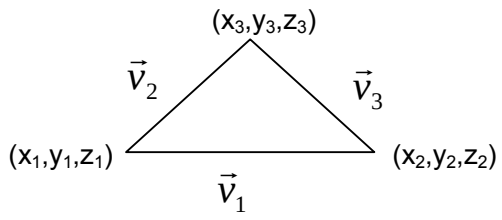
4.2.1 Schnittpunktberechnung

Herzstück der Teilchenverfolgung ist die Berechnung der Schnittpunkte der Teilchenbahnen mit den umgebenden Zellwänden. Diese muss sehr häufig durchgeführt werden, typisch ca. 50 Millionen mal in einer FIDAP-EIRENE Rechnung, und sollte daher möglichst effizient, aber auch möglichst genau sein. Hier werden die allgemeineren Ausführungen aus Kapitel 3.3 für ein Netz aus „linearen“ Tetraedern spezialisiert.

Die Ausgangssituation ist folgende:
Ein Teilchen befindet sich an seinem Entstehungsort oder am Ort des letzten Stoßes, den das Teilchen erlebt hat. Dem Teilchen wird eine Geschwindigkeit und eine Flugrichtung zugeordnet. In diese Richtung fliegt das Teilchen bis es eine Wand erreicht oder mit einem anderen Teilchen zusammenstößt. Dabei muss zu jeder Zeit bekannt sein, in welcher Zelle des Gitters sich das Teilchen gerade befindet. Die Bewegung des Teilchens kann man mit Hilfe der Geradengleichung

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + t \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{19}$$

beschreiben, wobei $(x_0, y_0, z_0)^T$ der Aufenthaltsort des Teilchens und $(v_x, v_y, v_z)^T$ die Flugrichtung ist. Außerdem gilt $t > 0$, denn das Teilchen kann sich nur in positiver Flugrichtung bewegen.
In einer Tetraedergeometrie sind die Wände der umgebenden Zelle die Dreiecksseiten der Tetraeder, also schiefliegende Dreiecke im $\Re^3$. Sie können durch ihre Ebenengleichung dargestellt werden. Ein beliebiges Dreieck



kann durch die Gleichung

$$\begin{pmatrix} x \\ y \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mu \cdot \begin{pmatrix} v_{1x} \\ v_{1y} \\ v_{1z} \end{pmatrix} + \lambda \cdot \begin{pmatrix} v_{2x} \\ v_{2y} \\ v_{2z} \end{pmatrix} \tag{20}$$

beschrieben werden. Dabei gilt innerhalb des Dreiecks $0 \le \mu \le 1$, $0 \le \lambda \le 1$ und $0 \le \mu + \lambda \le 1$. Um den Schnittpunkt einer Teilchenbahn mit der Seite eines Tetraeders zu bestimmen, muss also folgendes Gleichungssystem gelöst werden:

$$\begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + t \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mu \cdot \begin{pmatrix} v_{1x} \\ v_{1y} \\ v_{1z} \end{pmatrix} + \lambda \cdot \begin{pmatrix} v_{2x} \\ v_{2y} \\ v_{2z} \end{pmatrix} \tag{21}$$

oder in Matrixschreibweise

$$A \cdot \vec{x} = \vec{b} \tag{22}$$

mit

$$A = \begin{pmatrix} v_{1x} & v_{2x} & -v_x \\ v_{1y} & v_{2y} & -v_y \\ v_{1z} & v_{2z} & -v_z \end{pmatrix}, \quad \vec{x} = \begin{pmatrix} \mu \\ \lambda \\ t \end{pmatrix} \quad \text{und} \quad \vec{b} = \begin{pmatrix} x_0 - x_1 \\ y_0 - y_1 \\ z_0 - z_1 \end{pmatrix}.$$

Die Matrix- und Vektorelemente sind Abstände. Sie sind per Default in cm gegeben. EIRENE-Geometrien können aber sehr unterschiedlich in der Größe sein.
Die Autolampen, die mit FIDAP-EIRENE simuliert werden sollen, sind ca. 2-3 cm lang und haben einen Durchmesser von 1 cm. Ein Viertel einer solchen Lampe wird in etwa 100000 Tetraeder zerlegt. Fusionsexperimente hingegen sind mehrere Meter groß und werden mit etwa gleich vielen Tetraedern diskretisiert. (Abbildung 10 vermittelt einen Eindruck von den möglichen Geometrieabmessungen.)

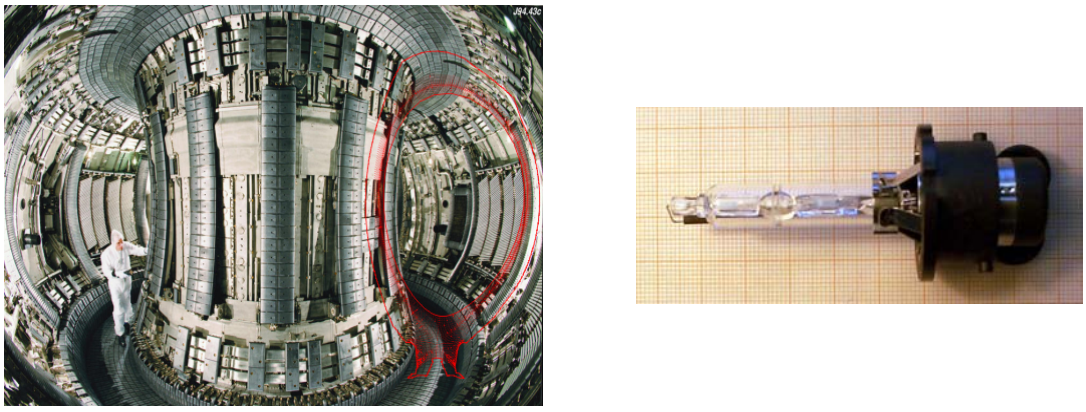


Abbildung 10 Zwei Experimentgeometrien, die gegenwärtig mit dem MC-Code EIRENE simuliert werden. Das linke Bild zeigt den JET Joint European Torus, das rechte Bild eine HID Autolampe. Im linken Bild (rot) ist ein 2D Raumgitter, 2D wegen der hier vorliegenden Symmetrie in toroidaler Richtung, zu sehen.

Um für alle möglichen Geometrien ein verlässliches Maß für die Genauigkeit eines Schnittpunkts zu haben, ist es notwendig Gleichung (22) zu normieren, das heißt mit dimensionslo-

sen Größen zu rechnen. Da es sich in Gleichung (22) um Abstände innerhalb des Dreiecks handelt, ist der Radius des Innenkreises r_{incrc} des Dreiecks eine gute Normierungsgröße. Es gilt

$$r_{incrc} = \sqrt{\frac{\left(s-\left|\vec{v}_1\right|\right)\left(s-\left|\vec{v}_2\right|\right)\left(s-\left|\vec{v}_3\right|\right)}{s}} \qquad mit \qquad s = \frac{\left|\vec{v}_1\right|+\left|\vec{v}_2\right|+\left|\vec{v}_3\right|}{2}. \qquad (23)$$

Man löst also das Gleichungssystem

$$\frac{1}{r_{incrc}} A \cdot \vec{x} = \frac{1}{r_{incrc}} \vec{b}\,. \qquad (24)$$

mit den Nebenbedingungen $t > 0$, $0 \le \mu \le 1$, $0 \le \lambda \le 1$ und $0 \le \mu + \lambda \le 1$. Gleichung (24) hat die gleichen Lösungen wie Gleichung (22). Man erhält die Lösungen auf einfache Weise mit Hilfe der Cramerschen Regel. Es ist

$$\mu = \frac{\left|(A_1 b)'\right|}{\left|A'\right|},\ \lambda = \frac{\left|(A_2 b)'\right|}{\left|A'\right|},\ t = \frac{\left|(A_3 b)'\right|}{\left|A'\right|} \quad \text{mit} \quad A' = \frac{1}{r_{incrc}} A, \quad b' = \frac{1}{r_{incrc}} \vec{b}, \quad \left|A'\right| \neq 0\,,$$

ferner bezeichne $(A_i b)'$ die Matrix, die entsteht, wenn man die i-te Spalte von A' durch $\vec{b}'$ ersetzt.

Die Größen $\vec{v}_1$, $\vec{v}_2$, $\vec{v}_3$ und $1/r_{incrc}$ sind unabhängig von den jeweils zu verfolgenden Teilchen. Sie können zur Zeitersparnis vorab einmalig berechnet werden. Die Determinanten der 3x3-Matrizen werden mit Hilfe der Regel von Sarrus berechnet.

Die programmtechnische Umsetzung findet man im Anhang unter Prozedur TIMER.

4.2.2 Suchen in Tetraedergeometrien

Während der Verfolgung eines Teilchen muss zu jedem Zeitpunkt bekannt sein, in welcher Zelle sich das Teilchen gerade befinden. Dies gilt insbesondere auch an seinem Geburtsort, da die Informationen über das Hintergrundplasma, wie z.B. Temperatur und Dichte, an die Zelle gekoppelt sind. Diese Informationen werden aber benötigt, um die Startenergie und Geschwindigkeit des Teilchens zu bestimmen, sowie dessen freie Weglänge $l_{mfp} = 1/\Sigma_t$, siehe Gleichung (6). Σ_t ist eine Funktion der Plasmaparameter in der Zelle. Es ist also notwendig, die Nummer der aktuellen Zelle effizient bestimmen zu können.

Dazu ist zunächst zu klären, wann sich ein Punkt innerhalb eines bestimmten Tetraeders befindet. Dies kann man mit Hilfe von Volumina ermitteln.

Sei T ein Tetraeder mit den Eckpunkten P_1, P_2, P_3, P_4 und sei P ein Punkt innerhalb des Tetraeders. Verbindet man P mit den Eckpunkten des Tetraeders so erhält man eine Zerlegung von T in vier neue Tetraeder T_1, T_2, T_3, T_4.

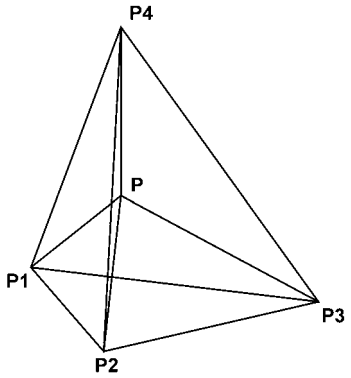


Abbildung 11 Tetraeder mit innerhalb liegendem Punkt P. Der Tetraeder lässt sich dadurch in vier Teiltetraeder zerlegen.

Für die Volumina der Tetraeder muss gelten

$$V_T = V_{T_1} + V_{T_2} + V_{T_3} + V_{T_4} \tag{25}$$

mit

$$V_T = \frac{1}{6}\begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} = \frac{1}{6}\begin{vmatrix} x_1 - x_2 & y_1 - y_2 & z_1 - z_2 \\ x_1 - x_3 & y_1 - y_3 & z_1 - z_3 \\ x_1 - x_4 & y_1 - y_4 & z_1 - z_4 \end{vmatrix}. \tag{26}$$

Es ist $V_T > 0$, wenn die Vektoren $\overrightarrow{P_1P_2}$, $\overrightarrow{P_1P_3}$ und $\overrightarrow{P_1P_4}$ ein Rechtssystem bilden. Ebenso gilt $V_{T_1}, V_{T_2}, V_{T_3}, V_{T_4} > 0$, wenn ($\overrightarrow{P_1P_2}$, $\overrightarrow{P_1P_3}$, $\overrightarrow{P_1P}$), ($\overrightarrow{P_3P_2}$, $\overrightarrow{P_3P_4}$, $\overrightarrow{P_3P}$), ($\overrightarrow{P_1P_3}$, $\overrightarrow{P_1P_4}$, $\overrightarrow{P_1P}$) und ($\overrightarrow{P_1P_4}$, $\overrightarrow{P_1P_2}$, $\overrightarrow{P_1P}$) Rechtssysteme bilden. Liegt P außerhalb des Tetraeders, so ist mindestens ein Teilvolumen negativ.

Um nun den Tetraeder zu finden, der einen vorgegebenen Punkt $P=(x_0, y_0, z_0)^T$ enthält, kann man natürlich in einer Schleife alle Tetraeder mit dem oben beschriebenen Kriterium prüfen. Dieses Verfahren wird sicherlich zum Erfolg führen, ist aber bei großen Zellzahlen recht zeitaufwändig. Man beachte, dass in einer Monte Carlo Simulation diese Elementaraufgabe bis zu vielen Millionen mal anfallen kann. Daher wurde eine verbesserte Variante im EIRENE-Code implementiert.

Zum einfacheren Verständnis wird das verwendete Verfahren zunächst für den zweidimensionalen Fall beschrieben.

Das Analogon zu einem 3D Tetraedergitter ist im $\Re^2$ ein Dreiecksgitter. Sei D ein Dreiecksgitter und P ein Punkt innerhalb des Rechengebietes. Es ist die Nummer des Dreiecks ge sucht, in dem sich P befindet.

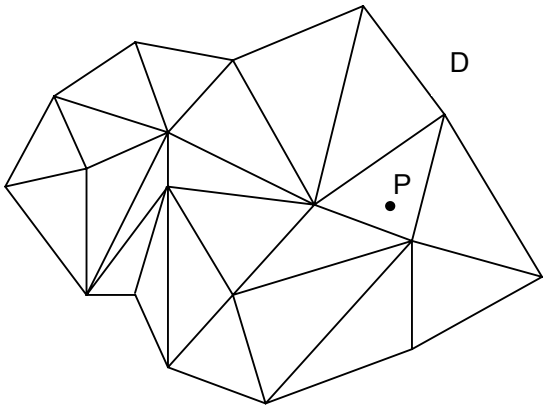


Abbildung 12 Beispiel für ein Dreiecksgitter

Zunächst bestimmt man ein Rechteck R das D vollständig enthält und definiert ein äquidistantes Gitter auf R mit Gitterweiten *dx* bzw. *dy*.

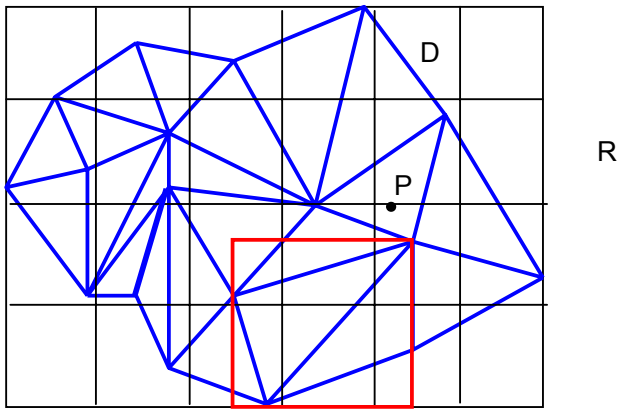


Abbildung 13 Dreiecksgitter mit Rechtecküberdeckung, rot markiert ist die Box um ein Dreieck

Anschließend wird für jede Zelle des Rechteckgitters ein Liste aufgebaut mit den Nummern der Dreiecke, die die Zelle schneiden können. Dazu bestimmt man für jedes Dreiecke eine Box, die das Dreieck beinhaltet. Jeder Zelle des Rechteckgitters die von dieser Box erfasst wird, wird die Nummer des gerade bearbeiteten Dreiecks zugeordnet.
Will man dann den Punkt P im Gitter lokalisieren, so bestimmt man die Nummer der Rechteckzelle, die P enthält - dafür benötig man nur zwei Multiplikationen mit $dx_i = 1/dx$, $dy_i = 1/dy$, ($ix = P_x*dx_i+1$, $iy = P_y*dy_i+1$) – und muss dann nur noch die Dreiecke überprüfen, die mit der Rechteckzelle (ix, iy) assoziiert sind.

Da sich im Laufe einer Rechnung das Gitter nicht ändert, muss das Aufbauen des Rechteckgitters und der zugehörigen Listen mit den Dreiecksnummern nur einmalig zu Beginn der Rechnung durchgeführt werden, gehört also nur zum Overhead der Monte Carlo Simulation.
Die Anzahl der dabei zu testenden Dreiecke ändert sich mit der Feinheit des Rechtecknetzes. Je feiner das Netz, desto weniger Dreiecke müssen getestet werden. Allerdings erhöht

sich in gleichem Maße der Speicherplatz, der für den Aufbau der Listen benötigt wird. Man gewinnt also Rechenzeit auf Kosten des Speicherplatzes.

Im dreidimensionalen Fall eines Tetraedergitter verfährt man analog zum oben vorgestellten Verfahren. Dazu bestimmt man zunächst einen Quader, der das gesamte Rechengebiet umschließt, indem man die Minima und Maxima der x-, y- und z-Koordinaten der Tetraedereckpunkte berechnet. Der Quader Q mit dem unteren linken Eckpunkt (xmin, ymin, zmin) und dem oberen rechten Eckpunkt (xmax, ymax, zmax) ist der kleinste Quader, der das komplette Rechengitter enthält. Im nächsten Schritt unterteilt man Q in n_q^3 gleichgroße Teilquader Q_t. Dazu teilt man jede Dimension äquidistant in n_q Teilstücke. Für jeden dieser kleineren Quader Q_t erzeugt man nun eine Liste L_{Tet} mit den Nummern der Tetraeder, die ganz oder teilweise in Q_t enthalten sind. Zu jedem Tetraeder T berechnet man den minimalen Quader Q_m, der T enthält, indem man wiederum die Minima und Maxima der Koordinaten der Eckpunkte von T bestimmt. Anschließend werden die Quader Q_t ermittelt, die von Q_m geschnitten werden. Ihnen wird der Tetraeder T zugeordnet, da sie einen Teil von T enthalten können.

Nach diesen Vorarbeiten, die wie bereits erwähnt einmalig zu Beginn einer Rechnung durchgeführt werden, kann die Bestimmung des Tetraeders, der einen vorgegebenen Punkt P enthält, folgendermaßen durchgeführt werden:
Man bestimmt der Teilquader Qt, der P enthält und überprüft dann nur noch die Tetraeder, die in der zugehörigen Liste L_{Tet} enthalten sind. Bei geschickter Wahl von n_q müssen dabei nur vergleichsweise wenige Tetraeder überprüft werden. Allerdings wächst der Speicherbedarf kubisch mit n_q.

Der vollständige Programmtext zu diesem Algorithmus findet sich in FUNCTION LEARCT im Anhang.

4.3 Die Definition der Teilchenquelle

Der EIRENE-Code kennt verschiedene Möglichkeiten, Teilchenquellen zu beschreiben („Sammeln aus Anfangsverteilung S“ in Gleichung (7)). Hier werden die unterschiedlichen Arten von Teilchenquellen in Tetraedergittern vorgestellt und ihre Implementation beschrieben. Einführend sei kurz daran erinnert, dass eine Zufallszahl ξ aus einer Verteilungsfunktion $f(x)$ erzeugt werden kann, indem man setzt:

$$r = \int_{-\infty}^{\xi} f(x)dx =: F(\xi)$$

(r eine gleichverteilte Zufallszahl)
und dann die (kumulative) Verteilung $F(\xi)$ invertiert, also $r = F(\xi)$ nach ξ auflöst.

4.3.1 Die Punktquelle

Die Punktquelle ist die einfachste Form einer Teilchenquelle. Sie wird durch die Vorgabe der kartesischen Koordinaten des Startpunkts (Geburtsort) x_0 der Teilchen und durch Angabe eines Richtungsvektors im EIRENE-Inputfile definiert (S entspricht einer δ-Funktion im Raum). Der Richtungsvektor beschreibt dabei die äußere Normale einer virtuellen Fläche, an der das startende Teilchen zur Bestimmung der Flugrichtung reflektiert wird (nur nötig, falls die Emission bei x_0 nicht isotrop im V-Raum). Da die Koordinaten des Startpunkts bekannt sind, ist nur noch die Nummer des Tetraeders zu ermitteln, der diesen Punkt enthält. Dazu wird das in Kapitel 4.2.2 vorgestellte Verfahren verwendet.

4.3.2 Die Flächenquelle

Die nächst kompliziertere Teilchenquelle ist die Flächenquelle. Hierbei möchte man die Teilchen räumlich verteilt auf einer vorgegebenen Fläche starten. S ist eine Funktion von nur zwei unabhängigen Koordinaten auf der Oberfläche,

$$S = S(u,v)$$

mit *u=u(x,y,z), v=v(x,y,z).*

Die Aufgabe des Sammelns aus einer Flächenquelle fällt an, wenn Randbedingungen auf den Rändern des Rechengebiets zu einer Umwandlung von Objekten des „Makro"-Moduls A in solche des „Mikro"-Moduls B beschreiben.
Beispiel: Plasmen in einem Gefäß (Modul A) strömen gegen eine Fläche; dort tritt Neutralisation ein und neutrale Atome oder Moleküle (Modul B) strömen ins Plasma zurück und wechselwirken dort mit dem Plasma.

Die Verteilung der Teilchen für Modul B kann dabei gleichverteilt auf der Fläche sein, oder vom Teilchenfluß aus Modul A auf die Fläche abhängen.

In einer Tetraedergeometrie besteht jede Fläche aus einer oder mehreren Tetraederseiten. Handelte es sich um eine zusammengesetzte Fläche, so müssen zunächst alle Tetraederseiten, also alle Dreiecke, die zu der Fläche gehören, gefunden werden. Um dies zu erleichtern, ordnet man jeder Startfläche ein eigenes Reflektionsmodell zu. Wie bereits in Kapitel 4.1.5 erwähnt, wird diese Information für jede Tetraederseite im Feld INMTIT abgespeichert. Man kann also alle zu einer Fläche gehörigen Dreiecke auffinden, indem man alle Tetraederseiten durchläuft und auf die Nummer des entsprechenden Reflektionsmodells überprüft.

Die gefundenen Dreiecke sammelt man in einer „Stufenfunktion". Dabei bildet die Summe der Dreiecksflächen die Abszisse RRSTEP und die Teilchenflußdichte auf das Dreieck die Ordinate FLSTEP.
Die Fläche eines Dreiecks $P_1P_2P_3$ im Raum berechnet sich nach

$$A = \sqrt{A_1^2 + A_2^2 + A_3^2} \tag{27}$$

mit

$$A_1 = \frac{1}{2}\begin{vmatrix} y_1 & z_1 & 1 \\ y_2 & z_2 & 1 \\ y_3 & z_3 & 1 \end{vmatrix}, \quad A_2 = \frac{1}{2}\begin{vmatrix} z_1 & x_1 & 1 \\ z_2 & x_2 & 1 \\ z_3 & x_3 & 1 \end{vmatrix}, \quad A_3 = \frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}.$$

Die Teilchenflußdichte ist z.B. im Fall von Plasmen in Modul A gegeben durch

$$\Phi = \frac{1}{2} cs \cdot n_i \tag{28}$$

mit der Schallgeschwindigkeit *cs* (Funktion von $T(\underline{r})$) und der Ionendichte $n_i = n_i(\underline{r})$. Zusätzlich werden zu jedem Dreieck die Nummer des Tetraeders und die Nummer der Tetraederseite, die Elektronen- und die Ionentemperatur, die Ionendichte und die Driftgeschwindigkeiten im Tetraeder gespeichert.
In der Funktion STEP wird die Flußdichte der Teilchen über RRSTEP integriert und auf Eins normiert. Man erhält so eine kumulative Auswahlfunktion, aus der die Startpunkte der Teilchen gezogen werden. Analog geht man im allgemeinen Fall vor, in dem Flussverteilungen auf Flächen durch Modul A als Quellterm nach Modul B übergeben werden.

Zur Berechnung des Startpunkts zieht man eine gleichverteilte Zufallszahl *r* aus dem Intervall [0,1] und bestimmt zunächst den Index *i* mit FLSTEP(*i*-1) ≤ *r* ≤ FLSTEP(*i*). Mit Hilfe von i findet man also die Nummer des Tetraeders ITET und die Seite IS, die den Startpunkt enthal-

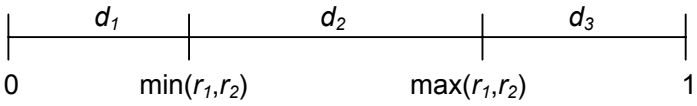
ten sollen. Nun ist noch der Startpunkt innerhalb der Tetraederseite festzulegen. Dabei bedient man sich der Dreiecks- oder baryzentrischen Koordinaten /17/ .

Faßt man die Punkte einer Ebene als Vektoren auf, so lässt sich jeder Punkt *P* der Ebene darstellen als Linearkombination der Dreiecksecken *A*, *B* und *C*

$$P = a \cdot A + b \cdot B + c \cdot C \quad \text{mit} \quad a + b + c = 1 \tag{29}$$

Man nennt *a*, *b* und *c* die baryzentrischen Koordinaten. Gilt $0 \le a,b,c \le 1$, so bewegt man sich innerhalb des Dreiecks.

Für die Berechnung des Teilchenstartpunkts /16/ bedeutet das Folgendes:
Man zieht zwei gleichverteilte Zufallszahlen r_1 und r_2 aus dem Intervall [0,1]. Damit zerlegt man das Intervall [0,1] in drei Teile



Es ist

$$d_1 = \min(r_1, r_2), \quad d_2 = \max(r_1, r_2) - \min(r_1, r_2), \quad d_3 = 1 - \max(r_1, r_2)$$

und

$$d_1 + d_2 + d_3 = 1.$$

Sind $(x_1, y_1, z_1)^T$, $(x_2, y_2, z_2)^T$, $(x_3, y_3, z_3)^T$ die Eckpunkte der Tetraederseite, so ist

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = d_1 \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + d_2 \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + d_3 \cdot \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix}$$

der gesuchte Startpunkt des Teilchens.

4.3.3 Die Volumenquelle

Die Volumenquelle wird im Zusammenhang mit Strahlungstransportrechnungen am häufigsten verwendet. $S(\underline{r})$ (Gleichung (7)) ist hier gegeben als $S(\underline{r}) = n_2(\underline{r}) A_{21}$, wobei $n_2(\underline{r})$ (aus Modul A) die Dichte der Emitter von Strahlung ist, A_{21} der Einsteinsche Koeffizient für spontane Emission einer Linie $2 \rightarrow 1$. Jede Linie wird sinnvollerweise als eigene Spezies im Modul B behandelt. Zu Beginn der Rechnung wird berechnet, welche Zellen des Rechengebiets aufgrund der Plasma- und Gasparameter durch einen spezifischen Prozess zu dieser Quelle beitragen können.

Mit Hilfe dieser Information wird wieder eine kumulative, normierte Auswahlfunktion („Verteilungsfunktion“) aufgestellt. Aus dieser Auswahlfunktion wird mittels einer gleichverteilten Zufallszahl die Startzelle bestimmt. Dazu muss ermittelt werden, in welcher Zelle die Auswahlfunktion den Wert der Zufallszahl annimmt. Da dies sehr häufig ausgeführt wird und die Auswahlfunktion aufsteigend sortiert ist, wird dazu ein binäres Suchverfahren eingesetzt.

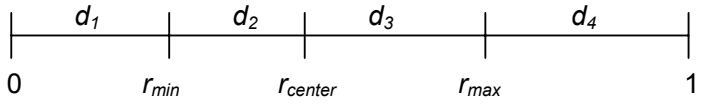
Nachdem die Startzelle ermittelt ist, wird analog zum Verfahren bei der Flächenquelle der Startpunkt des Teilchens in der Zelle berechnet. Man verwendet dazu Tetraederkoordinaten, die das Analogon zu den Dreieckskoordinaten oder baryzentrischen Koordinaten im $\Re^2$ darstellen.

Sind $(x_1, y_1, z_1)^T$, $(x_2, y_2, z_2)^T$, $(x_3,y_3,z_3)^T$, $(x_4,y_4,z_4)^T$ die Eckpunkte eines Tetraeders, so lässt sich jeder Punkt des $\Re^3$ schreiben als /17/

$$P = a \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + b \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + c \cdot \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} + d \cdot \begin{pmatrix} x_4 \\ y_4 \\ z_4 \end{pmatrix} \quad \text{mit} \quad a + b + c + d = 1 \tag{30}$$

Gilt $0 \leq a,b,c,d \leq 1$, so bewegt man sich innerhalb des Tetraeders.

Zur Bestimmung des Startpunkts /16/ zieht man drei Zufallszahlen r_1, r_2 und r_3 aus dem Intervall [0,1] und erzeugt sich mit ihrer Hilfe eine Zerlegung des Intervalls [0,1].



Es ist

$$r_{\min} = \min(r_1, r_2, r_3), \quad r_{\max} = \max(r_1, r_2, r_3), \quad r_{center} = r_1 + r_2 + r_3 - r_{\min} - r_{\max}$$

und

$$d_1 = r_{\min}, \quad d_2 = r_{center} - r_{\min}, \quad d_3 = r_{\max} - r_{center}, \quad d_4 = 1 - r_{\max}.$$

Die Koordinaten des Startpunkts erhält man durch Einsetzen in Gleichung (30)

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = d_1 \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + d_2 \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + d_3 \cdot \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} + d_4 \cdot \begin{pmatrix} x_4 \\ y_4 \\ z_4 \end{pmatrix}.$$

5 Spezifische Algorithmen für Datentransfer und Speicherplatzverwaltung

Wenn man sich damit beschäftigt, zwei Programme miteinander koppeln zu wollen, hat man im Allgemeinen ein klares Ziel vor Augen. Man will die Programme A und B miteinander in Kontakt bringen, da A Daten benötigt, die B erzeugen kann und umgekehrt. Andererseits ist die Berechnung der Daten zu komplex und bei „Mikro-Makro"-Problemen auch algorithmisch zu unterschiedlich, als dass die Neuprogrammierung von B innerhalb von A eine sinnvolle Alternative darstellt. A und B bleiben also weitgehend separierte Codes.

Im konkreten Anwendungsfall dieser Arbeit wird der kommerzielle Finite-Elemente-Code FIDAP /7/ zur Simulation von Hochdruckplasmalampen verwendet. Dazu löst er die dreidimensionale, stationäre Energiebilanzgleichung, die durch folgende Gleichungen beschrieben wird:

$$
\begin{aligned}
\text{Energiebilanz}: &\quad \sigma E^2 = U_{rad} - \vec{\nabla}\cdot\left(\kappa\vec{\nabla}T\right) + \rho c_p \vec{v}\vec{\nabla}T \\
\text{Impulsbilanz}: &\quad \rho\left(\vec{v}\cdot\vec{\nabla}\right)\vec{v} = \rho\vec{g} - \vec{\nabla}p_{total} + \vec{\nabla}\cdot\underline{\underline{\tau}} \\
\text{Viskositätstensor}: &\quad \tau_{ij} = \eta\left(\frac{\partial v_j}{\partial x_i} + \frac{\partial v_i}{\partial x_j} - \frac{2}{3}\delta_{ij}\left(\vec{\nabla}\cdot\vec{v}\right)\right) \\
\text{Ohm'sches Gesetz}: &\quad \vec{j} = \sigma\vec{E} \\
\text{Ladungserhaltung}: &\quad \vec{\nabla}\cdot\vec{j} = 0 \\
\text{Massenerhaltung}: &\quad \vec{\nabla}\cdot\left(\rho\vec{v}\right) = 0 \\
\text{Zustandsgleichung}: &\quad p_{total} = \frac{\rho}{M_{carrier}}RT
\end{aligned}
\tag{31}
$$

Dabei beschreibt die erste Gleichung die Umwandlung der Eingangsstromdichte ($\vec{E}$: elektrische Feldstärke, σ: elektrische Leitfähigkeit des Plasmas) in die totale Nettostrahlungsdichte U_{rad} und thermische und konvektive Verluste (zweiter und dritter Term auf der rechten Seite der Gleichung). κ ist die Wärmeleitung, c_p ist die Wärmeaufnahme bei konstantem Druck, $\vec{v}$ ist die Gasgeschwindigkeit. Die anderen Größen sind: *T*: Plasmatemperatur, ρ: totale Massendichte, p_{total}: totaler Plasmadruck, $M_{carrier}$: geometrisch gemittelte molare Masse, τ: Viskosität, x_i (i=1,2,3): räumliche Koordinaten, δ_{ij}: Kronecker Symbol und *R*: molare Gaskonstante.

Ein Bestandteil der dabei modellierten Gleichungen ist der Strahlungstransportterm. Er beschreibt die Energiequellen und –senken im Plasma, die durch den Strahlungstransport von Photonen verursacht werden. Im Lauf der Modellierungsarbeiten zeigte sich, dass die bislang in der industriellen Forschung verwendeten eindimensionalen zylindersymmetrischen Näherungen für den Strahlungstransport zu ungenau sind. So war man auf der Suche nach einem Verfahren zur dreidimensionalen Beschreibung des Strahlungstransports von Photonen in einem Hochdruckplasma.

Dieses 3D-Verfahren zur Beschreibung des Strahlungstransports von Photonen in einem Hintergrundplasma bietet der Monte-Carlo Code EIRENE. EIRENE war ursprünglich ein Programm zur Beschreibung des Transports von neutralen Teilchen in einem Fusionsplasma. EIRENE „löst" also Gleichung (6) bzw. Gleichung (7) in 3D-Konfigurationen. Da aber das

Verhalten von Photonen dem von Neutralteilchen stark ähnelt - beide reagieren weder auf elektrische Felder noch auf Magnetfelder, sondern fliegen von ihrem Geburtsort einfach geradeaus bis ein Zusammenstoß mit einem anderen Teilchen oder der umgebenden Wand erfolgt – konnte der EIRENE-Code im Rahmen einer Diplomarbeit in Zusammenarbeit mit dem Institut für Laserphysik der Universität Düsseldorf /19/ zu einem Photonen-Strahlungstransport-Code ausgebaut werden.

Aufgrund der genannten Problematik wurde beschlossen, den makroskopischen FEM-Code FIDAP und den mikroskopischen MC-Code EIRENE miteinander zu koppeln. Neben der bereits diskutierten Gitterharmonisierung sind dabei eine Reihe von Problemen zu lösen.

Dazu ist als Erstes zu untersuchen, welche Daten die beiden Programme austauschen müssen. Aus der oben beschriebenen Problemstellung ergibt sich automatisch, dass Daten über die räumlich Verteilung der Energiequellen und –senken von EIRENE an FIDAP geschickt werden müssen. Auf der anderen Seite benötigt der EIRENE-Code zur Berechnung des Strahlungstransports Informationen über das Rechengebiet, das heißt hier speziell über die zugrundeliegende Lampengeometrie und ihre Diskretisierung, und über das Hintergrundplasma und Gas.

Dabei stellt sich die Frage nach der Zusammensetzung des Gases in der Lampe:
- welche Teilchensorten sind vorhanden,
- wie ist die räumliche Verteilung ihrer Dichten,
- wie ist die Temperaturverteilung in der Lampe.

Diese Daten müssen vom FIDAP-Code an den EIRENE-Code geliefert werden. Dies ist möglich, da es ausschließlich makroskopische Größen sind: $n_i(\underline{r}), T(\underline{r})$ und $v(\underline{r})$.
Der Datenaustausch kann im ersten Schritt der Kopplungsentwicklung mit Hilfe von Dateien geschehen. Wenn es sich als notwendig erweisen sollte, EIRENE als Unterprogrammpaket in den FIDAP-Code zu integrieren, können die Daten später über interne Felder ausgetauscht werden.

5.1 Der Plasmahintergrund

Neben den Geometriedaten wird also die Beschreibung des Plasma- und Gashintergrundes benötigt, in dem sich die Testteilchen bewegen. Dabei handelt es sich um die Temperaturen der Elektronen und der verschiedenen, im Plasma befindlichen Ionen, um die Dichteverteilung der Ionen und der neutralen Gaskomponenten und ihre Driftgeschwindigkeiten. Wegen der sehr hohen Drucke (20 – 100 Bar) herrscht abgesehen von der Strahlung weitgehend „Gleichgewicht“, sog. „lokales thermodynamisches Gleichgewicht“, engl. „LTE“). Diese Daten müssen für jede Zelle des Rechengebiets zur Verfügung stehen, bevor das erste Teilchen seine Bahn beginnen kann.

Die benötigten Daten stellt der FIDAP-Code in einem Plasmafile zur Verfügung. Der Plasmafile liefert zu jedem Punkt des Rechengebietes die Gasdaten in tabellarischer Form. Jede Datenspalte ist dabei mit einer eindeutigen Kennung überschrieben.
Es folgen zur Verdeutlichung einige Zeilen aus einer Beispieldatei. Die einzelnen Zeilen wurden aufgrund des Papierformats umgebrochen.

```
  node  x             y             z             TEMP          -1
Ar            Hg            W             Ar2           Hg2
Ar+1          Hg+1          W+1           W-1
     1  0.395000E+01  0.200000E+00  0.000000E+00  0.290175E+04
0.231568E+18  0.727584E+24  0.469346E+26  0.431622E+17  0.401363E+19
0.785806E+24  0.513429E+07  0.231567E+18  0.218828E+13  0.514752E+09
```

```
     2  0.395000E+01 -0.200000E+00  0.244921E-16  0.290175E+04
0.231568E+18  0.727584E+24  0.469346E+26  0.431622E+17  0.401363E+19
0.785806E+24  0.513429E+07  0.231567E+18  0.218828E+13  0.514752E+09
     3  0.395000E+01  0.184776E+00  0.765367E-01  0.290175E+04
0.231568E+18  0.727584E+24  0.469346E+26  0.431622E+17  0.401363E+19
0.785806E+24  0.513429E+07  0.231567E+18  0.218828E+13  0.514752E+09
     4  0.395000E+01  0.141421E+00  0.141421E+00  0.290175E+04
0.231568E+18  0.727584E+24  0.469346E+26  0.431622E+17  0.401363E+19
0.785806E+24  0.513429E+07  0.231567E+18  0.218828E+13  0.514752E+09
                                    .
                                    .
                                    .
                                    .
                                    .
```

Man sieht, jeder Punkt des Rechengebietes erhält eine fortlaufende Nummer und wird mit seinen kartesischen Koordinaten aufgeführt. Anschließend werden ihm eine Temperatur in Kelvin, die Elektronendichte in #/m^3 sowie die Teilchendichten für Argon, Quecksilber, Wolfram, Ar_2, Hg_2, Ar^+ ,Hg^+, W^+ und W^-, ebenfalls in #/m^3, zugeordnet.

Aus den an den Punkten des Rechengebiets gegebenen Werte können im Makro-Code mit Hilfe der in Gleichung (18) definierten Shapefunktion die Plasmaprofile in jedem Punkt des Rechengebiets bestimmt werden. Dies ist Teil des Finite-Elemente Algorithmus in FIDAP. Dabei wird die gleiche parametrisierte Transformation verwendet wie für die Geometrie (siehe Kapitel 4.1).

Der Mikro-Code EIRENE hingegen erwartet konstante Werte in den Zellen, von denen man annimmt, dass sie am Schwerpunkt der Zelle angenommen werden. D.h., man muß die von FIDAP gelieferten Daten entsprechend interpolieren.

Dazu ließe sich natürlich die Shapefunktion φ aus Gleichung (18) verwenden. Dies setzt aber voraus, dass man die Koordinaten der Schwerpunkte der Tetraeder im natürlichen Koordinatensystem der Bricks kennt. Das ist leider nicht der Fall, und die Umkehrfunktion φ^{-1} kann offenbar nicht auf einfache Weise gewonnen werden. Jedenfalls sind dem Author keine FEM Prozeduren bekannt, die dieses für „Mikro-Makro"-Probleme wichtige Programmelement enthalten.

Allerdings kann man sich auf zweierlei Arten helfen:

1. Es handelt sich um lineare Tetraeder. Der Schwerpunkt kann durch arithmetische Mittelung der Koordinaten der Eckpunkte exakt berechnet werden. Die Plasma- und Gasparameter im Schwerpunkt des Tetraeders erhält man durch eine einfache, gewichtete Interpolation auf dem Tetraeder. Sei $\vec{x}_0$ der Schwerpunkt des Tetraeders, seien weiterhin $\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4$ die Eckpunkte des Tetraeders (diese sind nach den Konstruktionen in Kapitel 4 bekannt) und F_1, F_2, F_3 und F_4 die Funktionswerte auf den Ecken, so ist

$$F_{center} = \frac{\sum_{i=1}^{4} \frac{1}{dist(\vec{x}_i, \vec{x}_0)} \cdot F_i}{\sum_{i=1}^{4} \frac{1}{dist(\vec{x}_i, \vec{x}_0)}} \qquad (32)$$

   mit $dist(\vec{x}_i, \vec{x}_0)$ =euklidischer Abstand der Punkte $\vec{x}_i$ und $\vec{x}_0$.

2. Da die Eckpunkte der Tetraeder gleichzeitig Eckpunkte von Bricks sind, kennt man die natürlichen Koordinaten (r_i, s_i, t_i) der Ecken jedes Tetraeders in seinem zugehörigen Brick. Aus diesen natürlichen Koordinaten kann man die natürlichen Koordinaten (r_0, s_0, t_0) des Schwerpunkts im krummlinigen Koordinatensystem bestimmen. Unter Zuhilfenahme der

Shapefunktion φ können dann die Werte der makroskopischen Hintergrundparameter am Punkt (r_0, s_0, t_0) bestimmt werden.

Beide Verfahren wurden exemplarisch zur Bestimmung der Elektronentemperatur durchgeführt und die relativen Abweichungen bestimmt. Die mittlere relative Abweichung betrug 0.466%. Allerdings ist in einigen Zellen die Abweichung deutlich größer. Die maximale relative Abweichung beträgt 23.9%.
Die relative geringe mittlere Abweichung von weniger als einem halben Prozent lässt hoffen, dass der tatsächliche Fehler, den man bei Verwendung einer der beiden vorgestellten Varianten macht, vernachlässigbar ist. Allerdings sollte man die maximale Abweichung von fast 24% als Fingerzeig darauf werten, genauer zu untersuchen, unter welchen Bedingungen diese großen Abweichungen auftreten (z.B. geometrische Deformation des Bricks, Übergang in der Gittergröße, schlecht an Gradienten angepasstes Gitter), bzw. die Anstrengungen bei der Suche nach der Umkehrfunktion von φ zu verstärken.

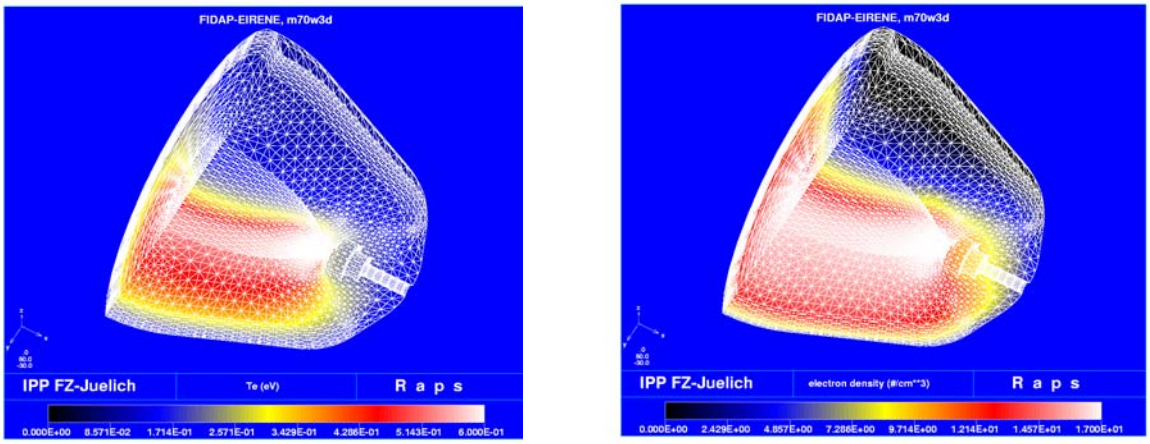


Abbildung 14 Elektronentemperatur und Elektronendichte in einer Beispiellampe, berechnet mit Verfahren 1

Abbildung 14 zeigt die Verteilung von Elektronentemperatur und –dichte in einer Hochdruckplasmalampe. Beide Pofile wurden mit Hilfe des ersten Verfahrens berechnet.

Zusätzlich zu den genannten primären Hintergrundparametern werden im Rahmen der Mikro-Makro-Kopplung auch noch abgeleitete Größen wie Gradienten und Laplace-Operatoren, insbesondere ∇T_e und ΔT_e, benötigt (siehe Kapitel 5.2). Diese Größen sind ebenso wie die primären Hintergrundparameter während der Monte-Carlo-Simulation konstant und können somit zu Beginn der Rechnung einmalig berechnet werden, tragen also nur zum Offset (Overhead), nicht zur Performance des Monte Carlo Teils selbst, bei.

5.1.1 Berechnung des Gradienten

Die primären Hintergrundparameter sind im Mikro-Code EIRENE in den Zellen konstant, und werden nach dem Mittelwertsatz der Differentialrechnung irgendwo in der Zelle angenommen. Daraus lässt sich kein Gradient dieser Parameter in der Zelle berechnen. Andererseits sind aus dem Makro-Code die Plasma- und Gasparameter an den Eckpunkten der Bricks bekannt. Da man außerdem die Shapefunktion φ (siehe Gleichung (18)) im Brick kennt, kann man daraus die partiellen Ableitungen der Hintergrundparameter an jeder Stelle des Bricks berechnen, sofern man diesen Ort in den natürlichen Koordinaten *r,s,t* ausdrücken kann. Die

dazu benötigten partiellen Ableitungen $\partial\varphi/\partial r, \partial\varphi/\partial s, \partial\varphi/\partial t$ der Shapefunktion φ nach den natürlichen Koordinaten sind leicht zu berechnen und können im Source-Code der Prozedur DFTRIQUA im Anhang eingesehen werden.

Im Folgenden wird exemplarisch für die Elektronentemperatur T_e die Berechnung des Gradienten $\nabla T_e = \left[\partial T_e/\partial x,\ \partial T_e/\partial y,\ \partial T_e/\partial z\right]$ vorgestellt /17/ .
Glücklicherweise wird der Temperaturgradient nur in den Mittelpunkten der Bricks benötigt. Dort kennt man die natürlichen Koordinaten. Es ist r=0, s=0 und t=0.

Wie bereits in Kapitel 4.1 erwähnt, gilt

$$\mathrm{x} = \mathbf{N}^{\mathrm{T}}\vec{x}\,,\quad \mathrm{y} = \mathbf{N}^{\mathrm{T}}\vec{y}\,,\quad \mathrm{z} = \mathbf{N}^{\mathrm{T}}\vec{z}\ ;\quad \mathbf{N}^{\mathrm{T}} = \mathbf{N}^{\mathrm{T}}(\mathrm{r,\ s,\ t})$$

mit $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{27} \end{pmatrix}, \vec{y} = \begin{pmatrix} y \\ \vdots \\ y_{27} \end{pmatrix}, \vec{z} = \begin{pmatrix} z \\ \vdots \\ z_{27} \end{pmatrix}$ Vektoren der Eckkoordinaten des Bricks und **N**=φ.
Gleichermaßen kann T_e berechnet werden als

$$T_e = \mathbf{N}^T \vec{T}_e$$

Dabei ist $\vec{T}_e = \begin{pmatrix} T_{e,1} \\ \vdots \\ T_{e,27} \end{pmatrix}$ ein Vektor mit den Werten der Elektronentemperatur an den Eckpunkten des Bricks.

Wenn man die Summen ausschreibt, so hat man

$$x = \sum_{i=1}^{27} x_i N_i(r,s,t), \quad y = \sum_{i=1}^{27} y_i N_i(r,s,t), \quad z = \sum_{i=1}^{27} z_i N_i(r,s,t) \quad \text{und} \quad \mathrm{T_e} = \sum_{i=1}^{27} T_{e,i} N_i(r,s,t).$$

Die partiellen Ableitungen von T_e sind dann /17/

$$\frac{\partial T_e}{\partial x} = \sum_{i=1}^{27} T_{e,i} \frac{\partial N_i(r,s,t)}{\partial x}, \quad \frac{\partial T_e}{\partial y} = \sum_{i=1}^{27} T_{e,i} \frac{\partial N_i(r,s,t)}{\partial y}, \quad \frac{\partial T_e}{\partial z} = \sum_{i=1}^{27} T_{e,i} \frac{\partial N_i(r,s,t)}{\partial z}. \qquad (33)$$

Die Ableitungen der Shapefunktion nach den kartesischen Koordinaten x, y und z erhält man mit Hilfe der Kettenregel:

$$\begin{aligned} \frac{\partial N}{\partial x} &= \frac{\partial N}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial N}{\partial s}\frac{\partial s}{\partial x} + \frac{\partial N}{\partial t}\frac{\partial t}{\partial x}, \\ \frac{\partial N}{\partial y} &= \frac{\partial N}{\partial r}\frac{\partial r}{\partial y} + \frac{\partial N}{\partial s}\frac{\partial s}{\partial y} + \frac{\partial N}{\partial t}\frac{\partial t}{\partial y} \\ \frac{\partial N}{\partial z} &= \frac{\partial N}{\partial r}\frac{\partial r}{\partial z} + \frac{\partial N}{\partial s}\frac{\partial s}{\partial z} + \frac{\partial N}{\partial t}\frac{\partial t}{\partial z} \end{aligned} \qquad (34)$$

In Matrixform

$$\begin{pmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \\ \frac{\partial N}{\partial z} \end{pmatrix} = \begin{pmatrix} \frac{\partial r}{\partial x} & \frac{\partial s}{\partial x} & \frac{\partial t}{\partial x} \\ \frac{\partial r}{\partial y} & \frac{\partial s}{\partial y} & \frac{\partial t}{\partial y} \\ \frac{\partial r}{\partial z} & \frac{\partial s}{\partial z} & \frac{\partial t}{\partial z} \end{pmatrix} \begin{pmatrix} \frac{\partial N}{\partial r} \\ \frac{\partial N}{\partial s} \\ \frac{\partial N}{\partial t} \end{pmatrix} \tag{35}$$

Die 3x3 Matrix aus Gleichung (35) ist die Inverse der Matrix

$$J = \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{27} x_i \frac{\partial N_i}{\partial r} & \sum_{i=1}^{27} y_i \frac{\partial N_i}{\partial r} & \sum_{i=1}^{27} z_i \frac{\partial N_i}{\partial r} \\ \sum_{i=1}^{27} x_i \frac{\partial N_i}{\partial s} & \sum_{i=1}^{27} y_i \frac{\partial N_i}{\partial s} & \sum_{i=1}^{27} z_i \frac{\partial N_i}{\partial s} \\ \sum_{i=1}^{27} x_i \frac{\partial N_i}{\partial t} & \sum_{i=1}^{27} y_i \frac{\partial N_i}{\partial t} & \sum_{i=1}^{27} z_i \frac{\partial N_i}{\partial t} \end{pmatrix} \tag{36}$$

Um an einem Punkt des Bricks den Gradienten zu berechnen, muss man also zunächst die Matrix J aufstellen und invertieren. Anschließend kann man die partiellen Ableitungen der Shapefunktion nach den Koordinaten x, y und z berechnen und in Gleichung (33) einsetzen.

Die Inverse einer 3x3 Matrix kann man z.B. mit Hilfe der Adjunkten bilden. Es ist

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \qquad \left(A^{-1}\right)^T = \frac{1}{|A|} \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix},$$

mit

$$\begin{aligned} A_{11} &= a_{22}a_{33} - a_{23}a_{32}\,, & A_{22} &= a_{33}a_{11} - a_{31}a_{13}\,, \\ A_{33} &= a_{11}a_{22} - a_{12}a_{21}\,, & A_{12} &= a_{23}a_{31} - a_{21}a_{33}\,, \\ A_{23} &= a_{31}a_{12} - a_{32}a_{11}\,, & A_{31} &= a_{12}a_{23} - a_{13}a_{22}\,, \\ A_{21} &= a_{32}a_{13} - a_{12}a_{33}\,, & A_{32} &= a_{13}a_{21} - a_{23}a_{11}\,, \\ A_{13} &= a_{21}a_{32} - a_{31}a_{22}\,, & |A| &= a_{11}A_{11} + a_{12}A_{12} + a_{13}A_{13}\,. \end{aligned} \tag{37}$$

5.1.2 Berechnung des Laplace Operators

Zumindest zu Diagnostikzwecken in Modul B, ev. Aber auch zum Transfer von bereits speziell aufbereiteten Größen von B nach A kann es sinnvoll oder notwendig sein, weitere abgeleitete Größen aus Modul A zunächst in B zu berechnen. Zum Beispiel können als weitere Größen auch Laplace Operatoren der primären Hintergrundparameter *F,* mit *F* z.B. n_i oder T_e,

$$\Delta F = \frac{\partial^2 F}{\partial x^2} + \frac{\partial^2 F}{\partial y^2} + \frac{\partial^2 F}{\partial z^2}\,, \tag{38}$$

schon zu Beginn der Simulation zur Verfügung gestellt werden. Allerdings ist die Berechnung von ΔF relativ aufwändig, denn F ist mittels der Shapefunktion φ (Gleichung (18)) nur auf krummlinigen Koordinaten gegeben. Um hier die Berechnung darlegen zu können, müssen zunächst einige Notationen und Konventionen /19/ eingeführt werden.

Seien **i**, **j**, **k** die Einheitsvektoren im $\Re^3$, $x^1 = r$, $x^2 = s$, $x^3 = t$ die krummlinigen Koordinaten im Brick. Dann kann man lokale Basisvektoren $\mathbf{e_i}$ entlang der Koordinatenlinien im krummlinigen Koordinatensystem definieren durch

$$\boldsymbol{e_i} = \frac{\partial x}{\partial x^i}\boldsymbol{i} + \frac{\partial y}{\partial x^i}\boldsymbol{j} + \frac{\partial z}{\partial x^i}\boldsymbol{k}\,.$$

Analog dazu sind

$$\boldsymbol{e^1}(x^1,x^2,x^3) = \frac{\boldsymbol{e_2}\times\boldsymbol{e_3}}{\left[\boldsymbol{e_1}\boldsymbol{e_2}\boldsymbol{e_3}\right]} \quad , \quad \boldsymbol{e^2}(x^1,x^2,x^3) = \frac{\boldsymbol{e_3}\times\boldsymbol{e_1}}{\left[\boldsymbol{e_1}\boldsymbol{e_2}\boldsymbol{e_3}\right]} \quad , \quad \boldsymbol{e^3}(x^1,x^2,x^3) = \frac{\boldsymbol{e_1}\times\boldsymbol{e_3}}{\left[\boldsymbol{e_1}\boldsymbol{e_2}\boldsymbol{e_3}\right]}$$

die lokalen Basisvektoren, die senkrecht auf den Koordinatenflächen stehen. Dabei ist $\left[\boldsymbol{e_1}\boldsymbol{e_2}\boldsymbol{e_3}\right]$ die Determinante der Matrix, die aus den Spaltenvektoren $\boldsymbol{e_i}$ gebildet werden kann.

Ferner werden hier Summationen folgendermaßen abgekürzt:

$$\sum_{k=1}^{n} A^{ik}B_k = c^i \quad (i = 1,2,\ldots,n) \quad \text{durch} \quad A^{ik}B_k = c^i$$

$$\sum_{i=1}^{n}\sum_{k=1}^{n} A^i_{kj}B^k C_{ih} = D_{jh} \quad (j,h = 1,2,\ldots,n) \quad \text{durch} \quad A^i_{kj}B^k C_{ih} = D_{jh}$$

und ebenso

$$\sum_{k=1}^{n} a^k e_k = a \quad \text{durch} \quad a^k e_k = a.$$

Es gelten folgende Konventionen:

1. **Summenkonvention**: Eine Summation von 1 bis n wird für jeden Dummyindex ausgeführt, der einmal als Superskript und einmal als Subskript auftritt.
2. **Bereichskonvention**: Alle freien Indizes, die nur als Superskripts oder als Subskripts auftreten, laufen von 1 bis n.
3. In Ableitungen wie z.B. $\partial a^i / \partial x^k$, wird k als Subskript aufgefasst.

Ferner seien die Christoffel Drei-Index Symbole der ersten und zweiten Art /21/ definiert durch

$$\left[ij;k\right] = \frac{1}{2}\left(\frac{\partial g_{ik}}{\partial x^j} + \frac{\partial g_{jk}}{\partial x^i} - \frac{\partial g_{ij}}{\partial x^k}\right) \qquad \left\{ \begin{matrix} k \\ i \quad j \end{matrix} \right\} = g^{kh}\left[ij;h\right]$$

mit

$$g_{ik}\left(x^1,x^2,x^3\right) = \left[\frac{\partial x}{\partial x^i}\frac{\partial x}{\partial x^k} + \frac{\partial y}{\partial x^i}\frac{\partial y}{\partial x^k} + \frac{\partial z}{\partial x^i}\frac{\partial z}{\partial x^k}\right]_{x^1,x^2,x^3} = g_{ki}\left(x^1,x^2,x^3\right) \quad (i,k = 1,2,3)$$

und

$$g^{ik}(x^1, x^2, x^3) = \boldsymbol{e}^i \cdot \boldsymbol{e}^k .$$

Nach /20/ gilt dann

$$\Delta Te = \nabla^2 Te = g^{ik} \frac{D}{\partial x^i} \frac{\partial Te}{\partial x^k} \equiv g^{ik} \left(\frac{\partial^2 Te}{\partial x^i \partial x^k} - \left\{ \begin{matrix} j \\ i \quad k \end{matrix} \right\} \frac{\partial Te}{\partial x^j} \right).$$

Oder ausgeschrieben:

$$\Delta F = \sum_{i=1}^{3} \sum_{k=1}^{3} g^{ik} \frac{\partial^2 F}{\partial x^i \partial x^k} - \frac{1}{2} \sum_{i=1}^{3} \sum_{k=1}^{3} \sum_{j=1}^{3} \sum_{h=1}^{3} g^{ik} g^{jh} \frac{\partial F}{\partial x^j} \cdot \left(\frac{\partial g_{ih}}{\partial x^k} + \frac{\partial g_{kh}}{\partial x^i} - \frac{\partial g_{ik}}{\partial x^h} \right).$$

Beim Testen der Implementation von ΔF muss berücksichtigt werden, dass die auf den Bricks definierte Shapefunktion φ nur maximal quadratische Funktionen in r, s und t exakt beschreiben kann.
Es zeigte sich, dass der hier angegebene Algorithmus exakt das analytische Resultat für ΔF liefert, wenn *F(r)* höchstens quadratische Terme in x, y, und z enthielt.

Wählt man eine Testfunktion höherer Ordnung, so ergeben sich Fehler durch die unvollständige Annäherung der Funktion durch φ. Aus diesem Grunde wurde zum Test auch die Potentialfunktion

$$f(x,y,z) = \frac{1}{R} = \frac{1}{\sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2}}$$

zu einer Punktquelle (Einheitsquellstärke) am Aufpunkt (x_0, y_0, z_0) verwendet. Befindet sich (x_0, y_0, z_0) außerhalb des Rechengebiets, so gilt analytisch $\Delta f \equiv 0$ überall im Rechengebiet.

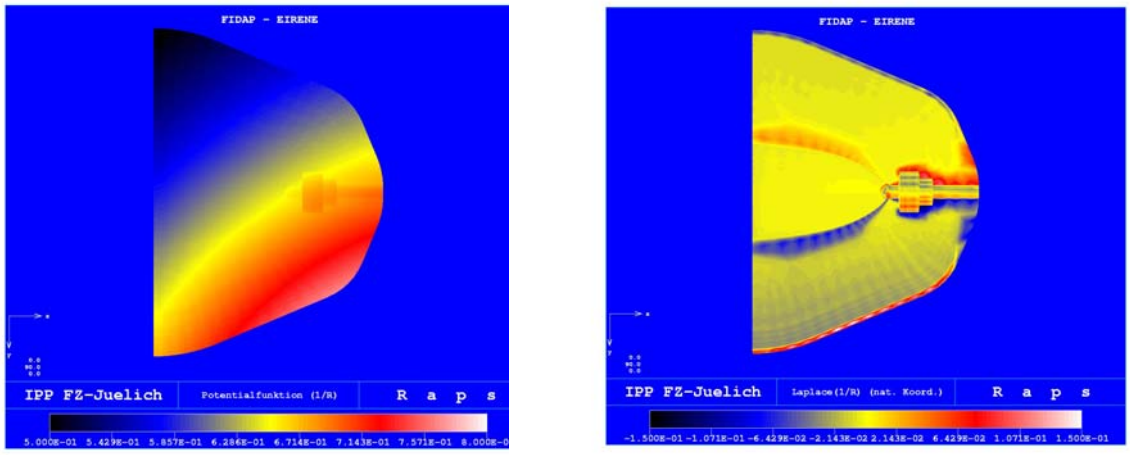


Abbildung 15 links: Potentialfunktion *f=1/R* mit Aufpunkt (1,1,1), rechts: *Laplace(f)*

In Abbildung 15 ist im linken Bild beispielhaft die Potentialfunktion mit dem Aufpunkt (1,1,1) abgebildet. Zum besseren Verständnis der Graphik ist nur der Blick auf eine der Symmetrieebenen des Rechengebiets dargestellt. Der Aufpunkt der Potentialfunktion befindet sich rechts unten, außerhalb des Bildes. Wie zu erwarten, breitet sich das Potential kreisförmig von seinem Aufpunkt aus.

In der rechten Graphik wird der zugehörige Laplace-Operator Δf gezeigt. Es ist die gleiche Symmetrieebene dargestellt wie im linken Bild. Da sich der Aufpunkt außerhalb des Rechengebiets befindet, sollte überall im Rechengebiet $\Delta f \equiv 0$ gelten. Wie man sieht, ist dies nicht überall der Fall. In den Bereichen des Rechengitters, in denen Zellen unterschiedlicher Größe aneinander stoßen, ist Δf deutlich größer als in anderen Bereichen. Hier sind offenbar die Grenzen der Diskretisierung bei diesem Rechengitter erreicht. Die Funktion *f* wird offensichtlich durch die Zellgröße nicht mehr fein genug aufgelöst, um zweite Ableitungen korrekt darstellen zu können.
Dies zeigt, dass es numerisch notwendig ist, starke diskontinuierliche Änderungen der Zellgröße zu vermeiden. Generell wäre es wünschenswert, lokale Gradienten mit Hilfe von adaptiver Gitterverfeinerung besser auflösen zu können.
Auch wenn der Einbau eines Verfahrens zur adaptiven Gitterverfeinerung im Makro-Code problematisch ist, so soll doch die Gitterharmonisierung auf Seiten des Mikro-Codes in Zukunft dahingehen optimiert werden, dass mit Hilfe der Tetraedrisierung des Rechengebiets die Zellgröße in Bereichen mit starken Änderungen der Hintergrundparameter besser an die physikalischen Bedingungen angepasst wird.

5.2 Strahlungsquellen und -senken für FIDAP

Nun behandeln wir die Übergabe von Ausgabedaten des Mikro-Codes an den Makro-Code im konkreten Fall des FIDAP-EIRENE Pakets. Im vorliegenden Fall sind dies die Nettoquellen durch Strahlung in der Energiebilanz von FIDAP, wobei die Strahlung mikroskopisch in EIRENE simuliert wurde.

An diesem konkreten Spezialfall lassen sich die wichtigsten Problemfelder bezüglich des Datentransfers bei der Kopplung von stochastischen Mikro- und numerischen Makro-Modellen verdeutlichen: Das Gesamtsystem besteht aus mehreren Komponenten. Einige davon werden dem Makro-Modul, die anderen dem Mikro-Modul zugeordnet. Dabei kann es Komponenten geben, die im Grenzbereich liegen, mit anderen Worten, die noch mikroskopisch beschrieben werden, aber die sich zumindest an einigen Stellen des Rechengebietes doch recht gleichgewichtsnah (also makroskopisch beschreibbar) verhalten. An dieser Stelle treten speziell bei Monte Carlo Verfahren für den Mikro-Modul Probleme auf, denn die an den Makro-Modul zu übergebenden Daten setzen sich als Differenz von zwei großen Zahlen zusammen. Beispiel: das Strahlungsfeld: in der Nähe des Gleichgewichts („Schwarzer Körper“) sind die Absorptions- und Emissionsraten nahezu gleich.
In der Rechnung wird aber die Differenz (Nettoemission) benötigt. Differenzen zweier ähnlicher Größen sind mit Monte Carlo Verfahren nur sehr schwierig zu schätzen, weil dann der relative statistische Fehler sehr groß werden kann. Ebenso ist es prinzipiell schwierig, mit Monte Carlo Verfahren den Übergangsbereich vom kinetischen (durch die Transportgleichung (7) beschriebenen Regime) in ein diffusives Regime (Makro-Code, Gleichung (2)) zu erfassen. Im Letzteren geht die freie Weglänge gegen Null, die Testteilchen „bleiben quasi stecken“, was sich durch nahezu unendliche Schleifen im Monte Carlo Module (lange Abfolge von CTCTCTCT… Kernen) ausdrückt, siehe Kapitel 3.2.

Durch Emission von Strahlung an einer (heißen) Stelle im Medium, und deren Absorption an einer anderen (kälteren) Stelle, wird Energie von der heißeren zur kälteren Stelle übertragen. Dies ist eine Form des Wärmetransports, zusätzlich zur Konvektion und Wärmeleitung. Die-

ser Wärmefluss kann also in die Energiebilanz als lokale Quelle/Senke U_{rad}, und/oder als Wärmefluss, dann ein Beitrag zu κ, eingeführt werden.

Ziel der Kopplung der beiden Programmpakete FIDAP und EIRENE ist die Berechnung der Strahlungsverluste U_{rad} mit

$$U_{rad}(r) = A_{21}n_2 - \int d\nu \int d\Omega\, n_1 B_{12} I_\nu(r,\Omega,\nu). \qquad (39)$$

U_{rad} setzt sich aus zwei Anteilen zusammen: aus der Strahlungsquelle *Emis* (erster Summand rechts) und aus der Strahlungssenke *Absorp* (zweiter, negativer Summand rechts).
Die Strahlungsquelle beschreibt die Strahlung, die emittiert wird, wenn ein Plasma- oder Gasteilchen aus einem oberen Energieniveau in ein unteres Energieniveau wechselt. Bei diesem Übergang entsteht ein Photon, d.h. es wird aus innerer Anregungsenergie des Gases Strahlungsenergie freigesetzt.

Das aus der Verteilung *Absorb* (*S*, Gleichung (7)) entstandene Photon bewegt sich dann durch das Plasma/Gas, (Transportkern T) bis es entweder den Rand der Geometrie erreicht hat und die Lampe verlässt, oder bis es im Plasma absorbiert wird (Stosskern *C*). Dabei reagiert das Photon mit einem Plasma- oder Gasteilchen aus einem unteren Energieniveau, das dadurch in ein höheres Energieniveau übergeht. Dieser Übergang verzehrt Strahlungsenergie, die Energie wird an das Medium zurückgegeben. Diesen Vorgang beschreibt die Strahlungssenke.

Emis entspricht der Quelle in der Strahlungstransportgleichung, und zwar einer Volumenquelle (siehe Kapitel 4.3.3). *Emis* kann sowohl von FIDAP als auch von EIRENE alleine aus den makroskopischen Parametern (n_2) und atomaren Daten (A_{21}, gemeinsamer Teil beider Codes) berechnet werden.
Wenn beides bekannt ist, so kann die Emission analytisch berechnet werden mittels

$$Emis = A_{21} \cdot n_2. \qquad (40)$$

Dabei ist A_{21} der Einsteinkoeffizient für die Emission von Photonen, n_2 ist die Dichte des oberen Energieniveaus.
Was sich allerdings in der Regel der analytischen oder numerischen Berechnung entzieht, ist die räumliche Verteilung der Strahlungssenke durch Absorption der Photonen. Es gilt

$$Absorp = \int d\nu \int d\Omega\, n_1 B_{12} I_\nu(r,\Omega,\nu). \qquad (41)$$

Die Absorption berechnet man mit Hilfe der Monte Carlo Simulation.
Dabei bildet man die Entstehung der Photonen (Emission), ihre Flugbahn durch das Plasma und ihre Vernichtung (Absorption bzw. Verlassen des Plasmas durch das Außenglas) im Rechner nach. Dadurch erhält man die spezifische Strahlungsintensität I_ν als Lösung der Strahlungstransportgleichung

$$\frac{1}{c}\frac{\partial I(\nu,\vec{\Omega},t)}{\partial t} + \vec{\Omega}\cdot\nabla I(\nu,\vec{\Omega},t) + \sigma_a(\nu) I(\nu,\vec{\Omega},t) = q(\nu). \qquad (42)$$

Diese Gleichung ist mathematisch analog zu Gleichung (7). Nach den Ausführungen in Kapitel 3 löst der Monte Carlo Mikro Code also auch diese Gleichung (42) (f, Ψ in (7) bzw. (5) entspricht hier: I_ν). Im Lauf der Rechnung gewinnt man also die vollständige Verteilung des Strahlungsfeldes, aufgelöst in Ort, Winkel und Frequenz. Durch Monte Carlo Integration über Winkel und Frequenz erhält man die räumliche Verteilung der Absorption.
Absorb ist also eines der „Tallies“ (Momente, oder „Response“, siehe Kapitel 3), die der Monte Carlo Algorithmus durch Photonensimulation liefert:

$$R = Absorb = \langle g, I \rangle, \quad mit\ g = B_{12}(v) \cdot n_1 \tag{43}$$

Da die komplette Information über das Strahlungsfeld aufgrund der Simulation automatisch vorliegt, erhält man gleichermaßen auch die Emission der Photonen als Monte Carlo Resultat: *Emis_gesammelt.* Wenn die Volumenquelle korrekt implementiert wurde, dann gilt:

$$Emis_{gesammelt}(N) \to Emis_{gerechnet} \quad für \quad N \to \infty$$

wobei N die Anzahl der Stichproben (Testteilchen) ist.

Diese doppelt vorliegende Information kann unter Umständen nützlich sein. Dazu betrachtet man die Varianz (statistischer Fehler) bei der Schätzung der Nettoquelle.

Bei Verwendung der analytisch berechneten Emission $Emis_{gerechnet}$, gilt für die statistische Varianz der Nettostrahlungsquelle

$$\sigma^2(Emis_{gerechnet} - Absorp) = \sigma^2(Absorp)$$

denn $Emis_{gerechnet}$ ist nur ein konstanter Summand, der also die Varianz von Schätzern nicht beeinflusst. Aber mit der zufällig gesammelten Emission (gemäß Kapitel 4.3.3, Volumenquelle), gilt

$$\sigma^2(Emis_{gesammelt} - Absorp) = \sigma^2(Emis_{gesammelt}) + \sigma^2(Absorp) - 2\,\mathrm{cov}(Emis_{gesammelt}, Absorp)$$

.

(44)

cov ist die Kovarianz der beiden Schätzer. Wenn die Emission und die Absorption in unmittelbarer Nähe voneinander stattfinden (sehr kleine Fluglänge), dann ist zu erwarten, dass diese beiden Schätzer positiv korreliert sind, d.h. cov>0. Hier sei bemerkt, dass ein Monte Carlo Kalkül neben Momenten R auch deren Varianz σ_R^2 liefert (mit g^2 anstatt g in den Formeln für die Schätzfunktion 1, 2, 3 in Kapitel 3.2). Ebenso kann auch die „empirische" Kovarianz cov(R1,R2) mittels $g_1 \cdot g_2$ (anstatt g) in den Schätzfunktionen erhalten werden. Auch dies ist eine Standardoption im EIRENE-Code (/5/ ,Manual, Block 10)
Bei positiver Korrelation der beiden Zufallsvariablen kann also die zweite Variante, bei der sowohl die Emission als auch die Absorption gesammelt werden, den kleineren statistischen Fehler besitzen.
Dies scheint der Intuition zu widersprechen, denn es wurde bei der zweiten Variante ja ein exakt bekannter Anteil durch eine statistische Schätzung ersetzt. Bei Gleichgewichtsnähe (kleine Weglängen) kompensiert diese allerdings teilweise das statistische Rauschen des zweiten Anteils in Gleichung (44).
Daher ist es sinnvoll beide Alternativen zur Verfügung zu haben. In Zukunft soll hier ein Verfahren entstehen, das anhand der berechneten Varianzen eine Entscheidung trifft, welche Alternative verwendet wird.

CFD Codes, wie auch FIDAP, behandeln in der Regel einige konvektive und diffusive Flüsse, Wärmeflüsse (Gleichung (1)), etc. implizit, während Quellen und Senken in den Gleichungen explizit beschrieben werden.
Deshalb ist zu erwarten, dass die numerische Stabilität des Gesamtpakets verbessert wird, wenn Quellen und Senken (etwa die aus dem Mikro-Modul) möglichst klein bleiben. Dies kann erreicht werden, indem an einen Teil dieser Quellterme formal als Divergenz eines Flusses formuliert und dann nicht in den expliziten Teil, sondern in den impliziten Teil des FIDAP Codes übergibt.

Innerhalb der FIDAP-Rechnung wird dann also nicht direkt der gesamte Strahlungsverlustterm U_{rad} verwendet, sondern FIDAP benutzt zur Stabilitätsverbesserung der Iteration eine Source-Term-Decomposition-Technik. Dabei wird U_{rad} in zwei Anteile zerlegt:

$$U_{rad} = U^1_{rad} + U^2_{rad}$$

U^1_{rad} beschreibt den Anteil der optisch dünnen Photonen am Strahlungstransportterm. Ein Objekt des Moduls B wird als „optisch dünn" bezeichnet, wenn es eine große freie Weglänge besitzt. D.h. es kann weit fliegen, bevor es absorbiert wird. Optisch dünne Photonen haben somit eine gute Chance den Rand der Geometrie zu erreichen, ohne absorbiert zu werden. U^1_{rad} geht als direkter Verlustterm in die Energiebilanz ein, und dieser Anteil kann wegen seiner einfachen Abhängigkeit von den Parametern des makroskopischen Plasmamodells sogar recht gut implizit korrigiert werden U^1_{rad} ist der Anteil, der besonders stark „mikroskopisches" Verhalten zeigt.
In diesem Anteil finden sich speziell bei Problemen des Strahlungstransports Beiträge von Linien, deren unteres Niveau nur relativ schwach besetzt ist, aber auch Anteile von Resonanzlinien (unteres Niveau = Grundzustand) aus den Linienflügeln.

U^2_{rad} hingegen beschreibt den Anteil der nicht „optisch dünnen" Objekte am Transportterm. Diese Objekte haben eine mittlere bis kleine freie Weglänge, und es besteht eine hohe Wahrscheinlichkeit, dass sie vom „Plasma" (Synonym für: Medium berechnet im Makro Code A) absorbiert werden. Explizite Abhängigkeiten dieses nichtlokalen Teils gibt es nicht, außer wieder im Grenzfall sehr kleiner Weglängen, in dem U^2_{rad} dann doch wieder nur von lokalen Parametern bestimmt ist. Im Allgemeinen jedoch hängt $U^2_{rad}(r_1)$ von der gesamten Lösung $I_v(\underline{r})$ der kinetischen Gleichung an allen Punkten $\underline{r}$, nicht nur an r_1, ab.

Um den impliziten Charakter der FIDAP Prozedur trotzdem nutzen zu können, wird nicht U^2_{rad} direkt in die Energiebilanz eingesetzt, sondern es wird mittels der Beziehung

$$U^2_{rad} = -\kappa_{rad} \cdot \Delta T_e$$

ein zusätzlicher Anteil des Diffusionskoeffizienten κ (hier: Wärmeleitkoeffizient) definiert, und so in den FIDAP Code übergeben. Dieser Ansatz liefert generell numerische Vorteile und trägt damit zur Stabilität der Mikro-Makro Iteration bei.

Wenn der Strahlungstransportterm mit Hilfe des Mikro-Codes B berechnet werden soll, muss naturgemäß auch die Zerlegung in die beiden Terme U^1_{rad} und κ_{rad} auf dieser Seite erfolgen, denn nur dort steht während der Simulation die dazu benötigte Information über die Weglängen zur Verfügung. Nachträglich kann nicht mehr entschieden werden, welcher Anteil des Strahlungsterms von „optisch dünnen" Objekten verursacht wurde. Es muss also während der Teilchenverfolgung getrennt nach „dünnen" und „dicken" Objekten gesammelt werden.

Wie unterscheidet man nun, wann ein Monte Carlo Objekt als dünn oder dick gezählt wird?
Um diese Entscheidung treffen zu können, berechnet man die freie Weglänge z_{mfp} des Testteilchens und vergleicht sie mit der Länge des Gradienten des relevanten Parameters z.B. T_e in Flugrichtung des Testteilchens. Die freie Weglänge z_{mfp} zwischen zwei Stößen wird (siehe Kapitel 3, Gleichung (7)) aus T erwürfelt, wobei T im Wesentlichen bis auf unwesentliche Vorfaktoren eine exponentielle Verteilung ist

$$T(z_{mfp}) \approx \exp(-\Sigma_t \cdot z_{mfp})$$

(Σ_t aus Gleichung (7)).

Diese Weglänge vergleicht man mit einer charakteristischen Länge des Hintergrundmediums, z.B. mit der Gradientenlänge in Flugrichtung $\left[\left(\vec{\nabla} T_e \cdot \frac{\vec{v}}{|\vec{v}|}\right) \Big/ T_e\right]^{-1}$.

Gilt nun

$$z_{mfp} < \alpha \cdot \left[\left(\vec{\nabla} T_e \cdot \frac{\vec{v}}{|\vec{v}|}\right) \Big/ T_e\right]^{-1}, \; mit \quad 0 < \alpha < 1,$$

α Inputparameter für Modul B, so wird das Objekt als dick betrachtet. Diese Entscheidung wird für jedes Wegstück getroffen, das ein Testteilchen zurücklegt. Entlang der Flugbahn des Teilchens wird für jede durchflogene Zelle berechnet, ob das Teilchen in dieser Zelle als dünn oder als dick betrachtet wird. Es kann also vorkommen, dass ein Monte Carlo Objekt in einer Zelle als dünn gilt, beim Durchflug durch die Nachbarzelle aufgrund der dort vorhandenen „Plasmaparameter" als dick gezählt wird, und in einer nachfolgenden Zelle wieder zu den dünnen Objekten gerechnet wird, weil sich die „Plasmaparameter" dort wieder verändert haben.

Beim Sammeln der Strahlungsverluste während der Monte Carlo Simulation sollte noch ein weiterer Punkt Beachtung finden: bei der Mikro-Makro Kopplung treffen zwei unterschiedliche Rechenprinzipien aufeinander. Der Makro-Code arbeitet mit analytischen und numerischen Verfahren. Der Mikro-Code hingegen verwendet eine Monte-Carlo Simulation zur Berechnung seiner Ergebnisse. Diese sind dadurch naturgemäß mit statistischem Rauschen belegt. Durch die Übernahme der Monte-Carlo-Ergennisse in den Makro-Code findet dieses statistische Rauschen Eingang in die numerische Rechnung. Dadurch können die verwendeten Verfahren an Stabilität verlieren bzw. die so gewonnenen Ergebnisse verfälscht werden. Aus diesem Grund sollte man versuchen, während der Monte-Carlo-Simulation die auftretenden statistischen Fehler möglichst zu minimieren.
Einen Beitrag dazu leistet die Verwendung unterschiedlicher Schätzfunktionen während der Teilchenverfolgung. Im EIRENE-Code sind drei verschiedene Schätzer implementiert (siehe Kapitel 3):

- Der sogenannte „collisional estimator"

$$\mathrm{X}_c\left(\omega_i^n\right) = \sum_{l=1}^{n} g_c\left(x_l\right) \cdot \prod_{j=1}^{l-1} \frac{c\left(x_j\right)}{\left(1 - p_a\left(x_j\right)\right)}$$

  zählt nur die eingetretenen Ereignisse, also z.B. die Anzahl der absorbierten Photonen pro Zelle. g_c ist die Gewichtsfunktion, siehe Gleichung (9) in Kapitel 3.2. p_a ist die Absorptionswahrscheinlichkeit am Stossort x_j und $c(x_j)$ ist die mittlere Anzahl an „überlebenden" Folgeteilchen nach dem Stoss. Wählt man als p_a die dem physikalischen Prozess wirklich zugrunde liegende Absorptionswahrscheinlichkeit, und kann, so wie bei den hier betrachteten Photonen, immer nur höchstens ein neues Photon aus einem Stoss hervorgehen, dann ist das Produkt der Gewichte im obigen Ausdruck identisch Eins.

- Der „tracklength estimator"

$$\mathrm{X}_t\left(\omega_i^n\right) = \sum_{l=0}^{n-1} \left\{ \int_{x_l}^{x_{l+1}} ds \; g_t\left(s\right) \right\} \cdot \prod_{j'1}^{l-1} \frac{c\left(x_j\right)}{\left(1 - p_a\left(x_j\right)\right)}$$

  berechnet entlang der Flugbahn des Photons in jeder Zelle das Wegintegral der Detektorfunktion g_t multipliziert mit der Gewichtskorrektur für Teilchenmultiplikations- und Absorptionsprozesse (siehe oben).

- Der „conditional expectation estimator“

$$X_e\left(\omega_i^n\right) = \sum_{l=0}^{n-1} \left\{ \int_{x_l}^{x_{end}} ds\; g_t(s) \cdot \exp\left(- \int_0^s ds' \Sigma_t(s') \right) \right\} \cdot \prod_{j=1}^{l-1} \frac{c(x_j)}{(1 - p_a(x_j))}$$

  erweitert den „tracklength estimator“ dahingehend, dass die Flugbahn nicht nur bis zum Eintritt einer Absorption verfolgt wird, sondern die Flugbahn wird über den Punkt der Absorption hinaus bis an den Rand des Rechengebietes verfolgt. Gleichzeitig erfolgt eine Gewichtskorrektur der gezählten Zufallsgröße.

Wie man an den Definitionen der Schätzer direkt sehen kann, erhöht sich jeweils der Aufwand, der für die Berechnung des Schätzers notwendig ist. Andererseits vermindert sich aber auch das statistische Rauschen pro Teilchen. Aus diesem Grund wird in der Simulation von Hochdruckplasmalampen standardmäßig der „conditional expectation estimator“ verwendet. Allerdings wird auf die Verwendung des „collision estimators“ umgeschaltet, wenn ein Photon direkt nach seiner Entstehung in der gleichen Zelle wieder absorbiert wird, ohne das es die Zelle zwischenzeitlich verlassen hätte. In diesem Fall muss im Nettoverlustterm eine 0 gesammelt werden. Um hier nicht durch Rundungsfehler zusätzliches Rauschen zu produzieren und ebenso um Rechenzeit zu sparen, werden in diesem Fall nur die beiden Ereignisse Emission und erfolgte Absorption gezählt und die Verfolgung des Photons beendet.

5.3 Datentransfer vom Mikro- zum Makro-Code

In Kapitel 5.1 wurde bereits diskutiert, wie der Makro-Code die Hintergrunddaten für den Datentransfer vom Makro- zum Mikro-Code zur Verfügung stellt, und wie sie auf die Erfordernisse des Mikro-Codes umgerechnet werden. In ähnlicher Weise ist dies auch für den Datentransfer in umgekehrter Richtung, also vom Mikro- zum Makro-Modul, notwendig. In Kapitel 3.1 wurde darauf hingewiesen, dass die vom Monte Carlo Algorithmus berechneten Ergebnisse Integrale (Mittelwerte) über die Zellen des Ortsgitters darstellen. Der Makro-Code hingegen benötigt die Quellen und Senken an den Gitterknoten, da die zur Berechnung Funktionen und Integralen verwendete Shapefunktion φ (siehe Gleichung (18)) auf den Eckpunkten der Basiselemente definiert ist. Es ist also, zumindest bei Verwendung von FEM Prozeduren in Modul A, eine Umrechnung der Resultate auf die Gitterknoten notwendig.

Im konkreten Anwendungsfall sind die Resultatwerte nicht nur Integrale über die Zellen des Ortsraum, sie sind auch schon volumengemittelt. Das heißt, die Größen liegen bezogen auf Volumeneinheiten vor, z.B. als #/cm^3.
Für die Interpolation auf die Gitterknoten wird unterstellt, die Werte werden nicht nur an einem Punkt der Zelle angenommen, sondern gelten überall in der Zelle. Dann gilt der einer Ergebniszelle (einem Brick) zugeordnete Wert gleichermaßen in allen zugehörigen Tetraedern. Zum Resultat an einem Gitterpunkt tragen schließlich alle Tetraeder bei, die diesen Gitterpunkt berühren. Dabei wird der Beitrag jedes Tetraeders gewichtet mit dem Inversen des Abstands zwischen Gitterpunkt und Schwerpunkt des Tetraeders.
Sei also $G = \vec{x}_G$ ein Gitterpunkt, I beschreibe die Menge der Tetraeder, die an G grenzen. Für alle $i \in I$ sei F_i der im Tetraeder i angenommene Wert und $\vec{x}_{0,i}$ sei der Schwerpunkt des Tetraeders i. Dann ist

$$F_G = \frac{\sum_{i \in I} \frac{1}{dist(\vec{x}_{0,i}, \vec{x}_G)} \cdot F_i}{\sum_{i \in I} \frac{1}{dist(\vec{x}_{0,i}, \vec{x}_G)}}$$

mit $dist(\vec{x}_{0,i}, \vec{x}_G)$=euklidischer Abstand der Punkte $\vec{x}_{0,i}$ und $\vec{x}_G$.

Die so gewonnenen Daten werden analog zum Plasmafile in tabellarischer Form in eine Datei geschrieben und an den Makro-Modul übergeben.

5.4 Gemeinsamer Codeteil: Datenbank

Das Gesamtsystem aus einem makroskopischen Modul A und einem mikroskopischen Modul B soll gemeinsam konsistent ein physikalisches Problem beschreiben. Dabei liegt eine starke Betonung auf dem Wort konsistent. D.h. zur Verwendung gleicher physikalischer Prozesse sollten beide Komponenten des Systems nach Möglichkeit gleichartige Beschreibungen dieser Prozesse verwenden. Wenn es machbar ist, sollten sogar gemeinsame Programmteile verwendet werden, um die Konsistenz zu erhöhen.
Im konkreten Anwendungsfall können die beteiligten Codes zwar keine gemeinsamen Programmteile verwenden, dennoch gibt es gemeinsam genutzte Komponenten.
Beide Codes benötigen zur Beschreibung der physikalischen Vorgänge die Kenngrößen von spektroskopischen Linienübergängen wie z.B. die Energieniveaus der Linien, den Einstein-Koeffizienten, die Verbreiterungskonstanten, etc. Diese Informationen werden im Rahmen des Verbundprojekts vom Institut für Laserphysik der Heinrich-Heine-Universität in Düsseldorf in einer spektroskopischen Datenbank zur Verfügung gestellt /32/ .

Diese Datenbank hat folgenden Aufbau:

Element	Wellenlänge	Zähl	Übergang	A_ul [1/s]	f_lu	g_l	g_u	l_l	l_u	E_l [1/cm]	E_u [1/cm]	C_2	C_3 (theo.)
Hg I	404.656	b		2.07E+07		1	3	1	0	37645.08	62350.456		0.00E+00
Hg I	365.015	b		1.29E+08		5	7	1	2	44042.977	71431.311		0.00E+00
Hg I	435.835	a		5.57E+07		3	3	1	0	39412.3	62350.456		4.05E-15
Hg I	546.074	a		4.87E+07		5	3	1	0	44042.977	62350.456		0.00E+00
Hg I	576.959	b		4.63E+07		3	5	1	2	54068.781	71396.22		7.10E-15
Hg I	579.065	b		4.58E+07		3	5	1	2	54068.781	71333.182		1.00E-14
Hg I	184.9492	b		6.77E+08		1	3	0	1	0	54068.781		5.35E-15
Hg I	296.728	b		4.50E+07		1	3	1	2	37645.08	71336.164		0.00E+00
Hg I	253.652	a		8.00E+06		1	3	0	1	0	39142.3		1.98E-16
Zn I	213.8573	a	s4p 1P - 4s2 1S	7.04E+08		1	3	0	1	0	46745.413		1.53E-14
Zn I	275.6449	a	s5d 3D - s4p 3P	3.42E+07		1	3	1	2	32311.35	68579.19		0.00E+00

C_4	C_6 (theo)	C_6qs (mess)	C_6 (Ar)	C_3 (mess)	Wellenlänge	Quellen/Kommentare
3.00E-22	1.63E-42				404.656	
3.00E-22	1.35E-41				365.015	
3.00E-22	1.60E-42				435.835	
6.00E-22	1.60E-42				546.074	
9.65E-20	1.33E-41				576.959	
9.55E-20	1.30E-41				579.065	
1.81E-22	4.20E-43				184.9492	
3.00E-22	1.34E-41				296.728	
1.81E-22	6.80E-44				253.652	
	2.04E-43				213.9247	
	4.46E-42			0.00E+00	275.7264	

Abbildung 16 spektroskopische Datenbank des ILPP, Universität Düsseldorf

Die Art der Verwendung der Datenbank durch den Makro-Code FIDAP ist nicht Bestandteil dieser Arbeit. Im Weiteren wird nur der Zugang des Mikro-Codes EIRENE zu dieser Datenbank diskutiert.
Der EIRENE-Code erhält in seiner Eingabedatei Informationen über die in der Rechnung zu verwendenden photonischen Reaktionen. Diese sind einzelnen spektroskopischen Linienübergängen zugeordnet. Die Zuordnung erfolgt über das Material, die Wellenlänge und einen Zähler, um unterschiedliche Varianten der Liniendefinition unterscheiden zu können. Diese Informationen entsprechen den ersten drei Felder der Datenbank.
Anhand dieser Eingaben findet der EIRENE-Code die entsprechenden Reaktionsdaten in der Datenbank. Die in der Datenbank enthaltene c_6-Konstante zur Berechnung von Verbreiterungskonstanten bezieht sich ausschließlich auf Argon. Will man die Verbreiterungskonstanten für andere Elemente verwenden, so müssen die entsprechenden c_6-Konstanten mit Hilfe einer Polarisationstabelle aus der c_6-Konstanten für Argon berechnet werden. Es gilt

$$c_6(\text{Element}) = c_6(\text{Argon}) * \text{Polarisationsfaktor(Element)}.$$

Mit diesen Informationen sind Emission (Quelle S) und Absorption (Kerne *C* und *T*) aus Gleichung (7) eindeutig für jede Linie („Monte Carlo Objekt“) festgelegt.
Andererseits kann auch Modul A z.B. den Term U^1_{rad} in der Energiebilanz (Gleichung (31)) daraus berechnen.

5.5 Speicherplatzverwaltung

Bei der Kopplung zweier Programmsysteme ist die Frage des benötigten Speicherplatzes immer von großer Bedeutung. Typischerweise schöpft bereits ein Makro-Code wie z.B. der FIDAP-Code die Speicherplatzressourcen des verwendeten Computers voll aus. Und auch ein Mikro-Code wie der EIRENE-Code stellt erhebliche Speicherplatzanforderungen.

In der ersten Phase der Code-Kopplung, in der beide Codes noch vollständig separiert laufen, stellt der Speicherplatz noch kein größeres Problem dar. Die Programme laufen nacheinander und haben jeweils den vollen Speicherplatz des Rechners zur Verfügung.

Das ändert sich, sobald die beiden Codes als ein Programmpaket ausgeführt werden müssen. Der Modul B also als Unterprogramm des Moduls A fungiert. In diesem Moment müssen beide Codes gleichzeitig in den Speicher des verwendeten Rechners passen.

Es stellt sich dann die Frage: sind die Codes „Open Source“ Programme, d.h. liegt von beiden Programmen der Sourcecode vor und kann somit in das Speichermanagement eingegriffen werden? Oder ist einer der Codes ein kommerzielles Produkt, das nur in kompilierter Form vorliegt und nicht verändert werden kann. Bei solchen Programmen können häufig zusätzliche Benutzerprogramme angefügt werden, man kann aber nicht in den eigentlichen Programmkern eingreifen.

Liegen hingegen beide Programmteile als Source vor, dann wäre es sinnvoll, in Modul A vor dem Aufruf von Modul B aktuell nicht benötigten Speicherplatz freizugeben. Analog sollte auch Modul B nach Beendigung eines Programmdurchlaufs den belegten Speicherplatz weitestgehend wieder zur Verfügung stellen.
Alle Informationen, die von primären Hintergrundparametern abhängen, sind beim nächsten Aufruf von Modul B durch zwischenzeitliche Änderungen in Modul A nicht mehr aktuell und müssen auf jeden Fall neu berechnet werden. Der Speicherplatz, der in Modul B für diese Informationen angelegt wurde, kann also bei Verlassen des Moduls zur Speicherplatzreduktion freigegeben werden. Bei Daten, die sich innerhalb einer gekoppelten Rechnung nicht ändern, wie z.B. die Beschreibung des Rechengebiets, muss eine Abwägung zwischen dem benötigten Speicherplatzes und der notwendigen Rechenzeit für eine wiederholte Berechnung stattfinden.

Prinzipiell sollte in beiden Codes zur Speicherreduzierung auf die Verwendung von automatischen, statisch angelegten Variablen und Felder verzichtet und die dynamische Speicherverwaltung bevorzugt werden.

Im konkreten Fall ist der makroskopische Finite-Elemente Code FIDAP ein kommerzielles Programm, das keine Eingriffe in den eigentlichen Programmkern und somit das Speichermanagement erlaubt. Es besteht daher nur die Möglichkeit im Mikro-Code Speicherplatzreduktionen vorzusehen.

Aus diesem Grund wurde im Mikro-Code das Konzept fest allozierter Felder zu Gunsten von dynamisch angelegten Array fallen gelassen. Zusätzlich wurden der Speicherplatz für die Ergebnistallies als ein großer zusammenhängender Speicherbereich angelegt. Die Teilbereiche für die einzelnen Ergebnisse werden über Pointer angesprochen. Dadurch hat man die Möglichkeit Speicherplatz nur für „erwünschte“ Ergebnisse anzulegen. Die Pointer für nicht erwünschte Tallies zeigen auf einen speziellen Speicherbereich, der als „Datenfriedhof“ dient. Alle Ergebnistallies wurden einzeln an- bzw. abschaltbar gemacht. Das ermöglicht es, in der Rechnung nur benötigte Tallies zu berücksichtigen. Zusätzlich bietet es die Möglichkeit für einen einfachen Check. Enthält der Datenfriedhof am Ende der Rechnung Werte ungleich Null, so ist irgendwo im Programm ein massiver Fehler aufgetreten.

Aber auch ohne vollständige Codekopplung, bei der der Mikro-Code EIRENE als Unterprogramm im FIDAP-Code laufen würde, bereitet die Zerlegung der triquadratischen Bricks in jeweils 48 Tetraeder (siehe Kapitel 4) Speicherplatzprobleme im EIRENE-Code. Die typischen FIDAP-Geometrien, die nur ein Viertel einer Lampe beschreiben, besitzen schon bei grober Zerlegung ungefähr 2500 Bricks. Nach der Zerlegung in Tetraeder hat man dann ca. 120000 Zellen. Bei 15 Eingabe- und bis zu 82 automatisch berechneten Ausgabegrößen (/5/ , Manual, Beschreibung der Ein-/Ausgabetallies), die räumlich aufgelöste Verteilungen enthalten, hat man bereits bei nur einer Spezies in Modul A einen Speicherplatzbedarf von

(15 + 82) * 120000 Zellen * 8 Byte = 93 MByte.

Weiterhin skaliert der Speicherplatzbedarf ungefähr mit ½ * Anzahl der Spezies in Modul A. So waren also Platzprobleme vorherzusehen. Daher wurde beschlossen, die Eingabegrößen vorerst in der feinen räumlichen Auflösung zu belassen, um keine Ungenauigkeiten in der Übernahme der Plasma- und Gasparameter hervorzurufen, die Auflösung der Ausgabegrößen aber zu vergröbern.

Aus diesem Grund wurde eine indirekte Adressierung der Ausgabetallies eingeführt, die es erlaubt, mehrere Zellen des Rechengebiets zu einem Volumen zusammenzufassen. Dazu wurde ein Indexarray implementiert, der für jede Zelle des Rechengebiets auf die zugehörige Zelle der Ausgabegröße verweist.
Standardmäßig haben die Ausgabetallies genauso viele Zellen wie das Rechengebiet. Durch Setzen des Indexarrays, z.B. in der benutzerspezifischen Prozedur GEOUSR oder wie in diesem Fall in der entsprechenden Kopplungsroutine, können die Zellen des Rechengebiets vom Benutzer beliebig zusammengefasst werden. Im speziellen Anwendungsfall der Kopplung an den FIDAP-Code wurden die 48 Tetraeders eines Bricks jeweils zu einer Ergebniszelle vereinigt. Dazu wird bereits bei der Zerlegung der Bricks in Tetraeder jedem Tetraeder die Nummer seines Ursprungsbricks zugeordnet.

Dieses Vorgehen bietet zusätzliche Möglichkeiten für künftige Optimierungen im Mikro-Code. In Anbetracht der Probleme, die bei der Berechnung von ΔTe an den Grenzen zwischen unterschiedlich fein diskretisierten Bereichen beobachtet wurden (siehe Kapitel 5.1.2), kann man überlegen, ob eine Art adaptive Gitterverfeinerung mit Hilfe dieses Indexarrays machbar ist. Man könnte in Bereichen mit steilen Gradienten die Tallies auf den fein aufgelösten Tetraedern zur Verfügung stellen, während in anderen, weniger kritischen Bereichen die Bricks oder sogar Volumina aus mehreren Bricks als Ergebniszellen in Betracht kommen. Die sich hier eröffnenden Möglichkeiten sollen in Zukunft untersucht werden.

6 Parallelisierung des Monte Carlo Teils

Vorab sei noch einmal daran erinnert, das wegen $FOM \sim \sqrt{CPU}$ im Monte Carlo Code nicht die Testteilchenzahl sondern die zu verbrauchende Rechenzeit die wesentliche primäre Inputgröße ist. Der Monte Carlo Code verfolgt so lange Testobjekte, bis diese Zeit verbraucht ist.

Bereits in einem frühen Stadium des Projekts wurde erkannt, dass die Berechnung des Strahlungstransports mit Hilfe des Monte Carlo Codes EIRENE recht rechenzeitaufwendig werden würde, da bei der Beschreibung der Lampenplasmen eine große Zahl von verschiedenen Testteilchen auftreten, deren Reaktionen mit einer hinreichend guten Statistik beschrieben werden müssen. Aus diesem Grund wurde beschlossen, den EIRENE-Code zu parallelisieren, um durch Ausnutzen der vorhandenen Möglichkeiten zu parallelem Rechnen zu kürzeren Gesamtrechenzeiten zu kommen.

Das Monte Carlo Verfahren zum Neutralteilchen- bzw. Photonentransport ist inhärent parallelisierbar /22/ /23/ . Die Teilchenbahnen sind unabhängig voneinander, da die Teilchen nicht untereinander reagieren, und können daher parallel berechnet werden.
Es würde sich also anbieten, nach einer Initialisierungsphase, in der die Geometrie und die Hintergrunddaten aufbereitet werden, die zur Teilchenverfolgung benötigten Informationen auf die beteiligten Prozessoren zu verteilen. Dann die Teilchenbahnen parallel zu berechnen und am Ende der Rechnung die Ergebnisse zu akkumulieren.
Das implementierte Verfahren ist eine auf den EIRENE-Code zugeschnittene Variante dieses Algorithmus.

In einer Rechnung des Mikro-Codes EIRENE gibt es fast immer eine Vielzahl unterschiedlicher Orte, an denen Testteilchen auf durchaus unterschiedliche Arten entstehen können. Dies in einem geschlossenen Ausdruck zu beschreiben, ist nicht trivial, wenn nicht unmöglich. Daher wurde bereits vor langer Zeit das Verfahren des „stratified sampling“ (siehe Kapitel 3.2) in EIRENE implementiert. Dieses Verfahren erlaubt es, die gesamte Teilchenquelle in mehrere Teile (Strata) zu zerlegen, die einfacher zu beschreiben sind. Die Teilquellen werden unabhängig von einander bearbeitet und die Ergebnisse am Ende der Rechnung superponiert.

Aus der historischen Speicherplatznot heraus ist ferner im EIRENE-Code zu jedem Zeitpunkt nur Speicherplatz für die Ergebnistallies eines Stratums und das Gesamtergebnis, die Summe über die Strata, vorhanden. Als der Gedanke entstand, EIRENE für den Einsatz auf Parallelrechnern mit verteiltem Speicher zu parallelisieren, stellte sich die Frage, ob sich durch das „stratified sampling“ und der zusätzlich vorhandene Speicher zusätzliche Optimierungsmöglichkeiten bieten.

Im Unterprogramm MCARLO des EIRENE-Codes, das die Durchführung des Monte-Carlo-Verfahrens steuert, hat man zwei geschachtelte Schleifen, die den Kern des MC-Verfahrens darstellen:

```
do istra=1, Anzahl der Teilquellen (NSTRA)
  do iprt=1, Anzahl der Teilchen (NPTS)
      verfolge ein Teilchen
      aktualisiere die Ergebnistallies
  end do
  skaliere und integriere die Ergebnisse der Teilquelle
  bilde das Gesamtergebnis
end do
```

Parallelisiert man die Teilchenschleife, so müssen am Ende der Teilchenschleife die Ergebnisse der verschiedenen Prozessoren akkumuliert werden, bevor die Ergebnisse der Teilchenquelle skaliert und integriert werden können. Dies bedeutet zusätzliche Kommunikation und zusätzlichen Rechenaufwand.
Parallelisiert man hingegen die Schleife über die Teilquellen, so bieten sich andere Möglichkeiten:

1. Fall: die Anzahl der Prozessoren NPES ist kleiner oder gleich NSTRA

Jeder Prozessor berechnet die Teilchen für NSTRA/NPES Quellen. Am Ende der Teilchenschleife ist keine zusätzlich Kommunikation zur Akkumulierung der Rechenergebnisse notwendig. Die Ergebnisse aller Teilquellen, die ein Prozessor berechnet hat, werden lokal im Prozessor zum teilweisen Gesamtergebnis akkumuliert. Erst am Ende der Rechnung, nach Abarbeitung aller Teilquellen, werden die Gesamtergebnisse der Prozessoren zusammengeführt.
Die Rechenzeit pro Quelle beträgt

CPUTIME / NSTRA * NPES,

wobei CPUTIME die vorgegebene Rechenzeit ist.

Bei dieser Art der Rechnung ist darauf zu achten, dass NSTRA ein ganzzahliges Vielfaches von NPES sein sollte, da bislang nur dann ein optimales Loadbalancing erreicht wird. Hier ist noch Raum für künftige Verbesserungen, die zum Beispiel unterschiedliche Quellstärken bei der Zuteilung von Prozessoren zu Quellen und Rechenzeiten besser berücksichtigen.

2. Fall: die Anzahl der Prozessoren NPES ist größer als NSTRA

Die Zuordnung der Prozessoren zu den Teilquellen erfolgt folgendermaßen: Jeder Teilquelle wird mindestens ein Prozessor zugeordnet. Jede Quelle und jeder Prozessor erhalten die maximal erlaubte Rechenzeit. Die überzähligen Prozessoren werden den Teilquellen proportional zu ihrer Quellstärke zusätzlich zugeordnet. Auf diese Weise erhalten alle Quellen Rechenleistung, die wichtigeren Quellen mit den höheren Quellstärken bekommen aber mehr Rechenleistung, was zur Verbesserung der Statistik beiträgt.

In diesem Fall muss am Ende der Teilchenschleife Kommunikation zur Akkumulation der Ergebnisse der Teilquellen eingeschoben werden. Jeweils der erste Prozessor einer Gruppe von Prozessoren, die gemeinsam eine Quelle bearbeitet haben, sammelt die gemeinsamen Ergebnisse und gibt sie bei der Akkumulation des Gesamtergebnisses weiter.

In diesem Fall ist zu beachten, dass Prozessoren, die die gleiche Teilquelle bearbeiten, unterschiedliche Startpunkte für die Berechnung ihrer Zufallsfolgen /24/ , /25/ erhalten. Andernfalls würden mehrere Prozessoren die gleiche Zufallsfolge durchlaufen und dadurch die gleichen Teilchen berechnen. Das würde die Rechenzeit vervielfachen, aber keinen Gewinn für die Statistik bringen. Weiterhin sollte man im Auge behalten, dass der aktuell verwendete Zufallszahlengenerator eine Periode von 2^{31} hat. Wenn man pro Teilchen ca. 10 Zufallszahlen benötigt, so wiederholt sich die Folge nach ca. $2*10^8$ Teilchen. Bei intensiver Nutzung von massiv parallelen Rechnersystemen müssen an dieser Stelle Verbesserungen vorgenommen werden.

Die Implementation der notwendigen Programmteile zum Verteilen der Startup-Informationen und zur Akkumulation der Ergebnisse wurde mit Hilfe des „Message Passing Interfaces“ (MPI, /27/) realisiert. Zur Evaluation der Parallelisierung des Monte Carlo Teils des Mikro-Makro Systems wurden verschiedene Test auf dem „JUelicher Multi Processor“ (JUMP) des Forschungszentrum Jülich durchgeführt. Dabei handelt es sich um ein massiv paralleles IBM Regatta p690+ e-server Cluster mit 1312 Prozessoren.

Im Rahmen der Tests wurden verschiedene Szenarien der Rechenzeitzuteilung auf Teilquellen und Prozessoren berücksichtigt. Einige Ergebnisse dieser Tests sollen im Folgenden kurz dargestellt werden.

Der Monte Carlo Code EIRENE bietet dem Benutzer verschiedene Arten an, Rechenzeit auf die einzelnen Teilquelle zu verteilen. Zum einen ist es möglich für jede Teilquelle die maximale Anzahl der zu berechnenden Testteilchen vorzugeben. Auf der anderen Seite wird auch die vom Programm maximal zu nutzende CPU-Zeit vorab festgelegt. Schließlich kann die Zuteilung der CPU-Zeit auf die einzelnen Teilquellen ausschließlich nach der Anzahl der zu berechnenden Teilchen erfolgen, oder es werden zusätzlich die Quellstärken der verschiedenen Teilquellen dabei berücksichtigt.

Die zum Test verwendete Lampensimulation berechnet den Strahlungstransport von 29 verschiedenen Zinklinien. Da jede Linie mit Hilfe einer eigenen Quelle berechnet wird, hat man 29 Teilquellen in dieser Rechnung.

Im ersten Szenario wird die maximal erlaubte Rechenzeit auf unendlich gesetzt, und jeder Teilquelle eine maximale Teilchenzahl von 100000 Testteilchen vorgeschrieben.

Diese Rechnung wurde dann mit unterschiedlicher Anzahl von Prozessoren mehrfach durchgeführt und die benötigte Rechenzeit pro Prozessor gemessen. In der Theorie würde man erwarten, dass mit steigender Anzahl an Prozessoren NPES die Rechenzeit pro Prozessor mit 1/NPES sinkt. Die Rechnung wurde mit 1, 2, 4, 8, 15 und 29 Prozessoren durchgeführt. Die Ergebnisse sind in Abbildung 17 dargestellt.

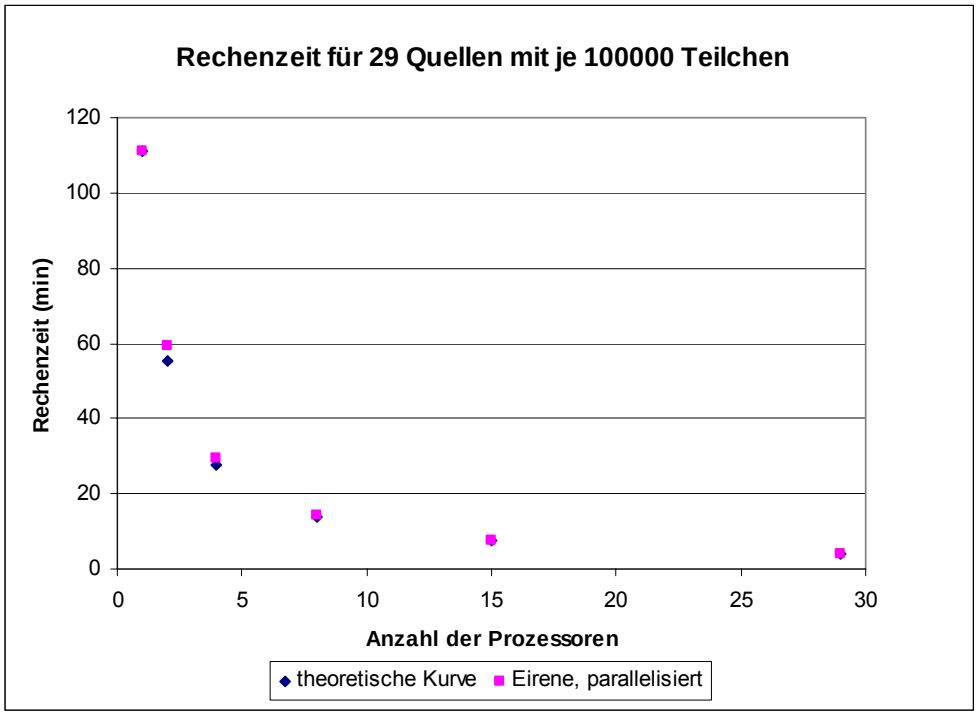


Abbildung 17 Rechenzeitverhalten bei fester Anzahl an Quellen und Teilchen und steigender Zahl an Prozessoren

Wie man sieht, folgen die gemessenen Rechenzeiten der theoretischen Kurve recht gut. Die Abweichungen resultieren aus der Tatsache, dass mit 29 Teilquellen ein optimales Loadbalancing nur bei einem und bei 29 Prozessoren erreicht werden kann, da 29 eine Primzahl ist.

Andererseits ist es sinnlos, eine höhere Anzahl an Prozessoren zu wählen, da in diesem Fall die Art und Weise der Parallelisierung ein weiteres Absinken der Rechenzeiten verhindern würde. Bei mehr als 29 Prozessoren würden nämlich einigen Teilquellen mehrere Prozessoren zugeordnet, die dann jeweils 100000 Teilchen rechnen würde. Die Rechenzeit pro Prozessor ändert sich dadurch nicht, nur die Statistik wäre für einige Quellen besser, da mehr Teilchen für sie gerechnet würden.

Das zweite Testszenario, das hier vorgestellt werden soll, begrenzt die maximale Rechenzeit auf 1000 Sekunden. Gleichzeitig wird die erlaubte Teilchenzahl pro Teilquelle auf unendlich gesetzt und die Quellstärke der verschiedenen Teilquellen bei der Rechenzeitzuteilung berücksichtigt.
Dies führt zu einer sehr unterschiedlichen Rechenzeitzuteilung für die verschiedenen Teilquellen. Die erste zu berechnende Linie, Zink bei 213 nm, ist eine Resonanzlinie und als solche sehr viel „dicker“ als die anderen 28 Linien in dieser Rechnung. Aufgrund dieser Tatsache erhält diese Linie in der sequentiellen Rechnung mit einem Prozessor 22% der Rechenzeit. Zusätzlich benötigt jedes einzelne Teilchen, das aus dieser Linie gerechnet wird, sehr viel weniger Rechenzeit als die Teilchen aus den anderen, dünneren Linien, denn die „dicken“ Teilchen werden nahezu direkt nach ihrem Entstehen vom Gas reabsorbiert. Die „dünnen“ Photonen hingegen habe eine reelle Chance, die ganze Geometrie zu durchqueren und die Lampe zu verlassen. Sie laufen also deutlich länger.

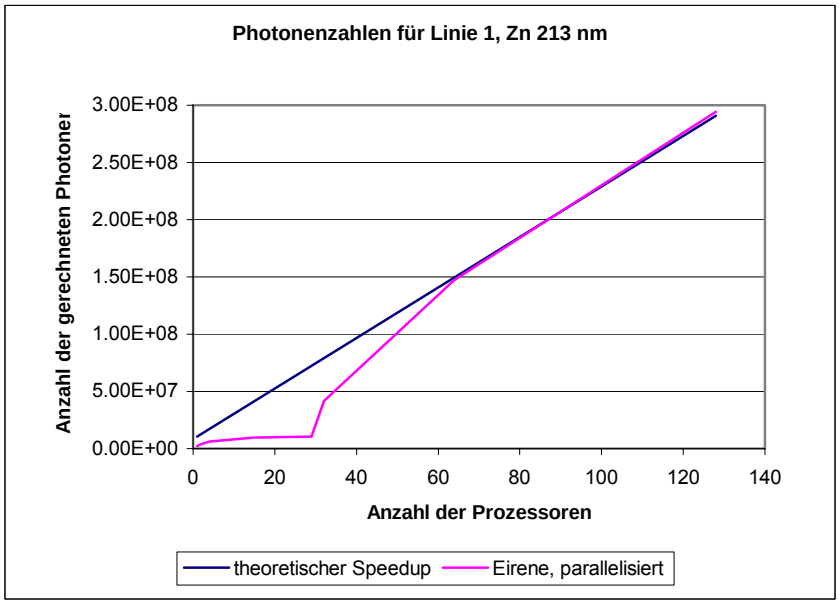


Abbildung 18 Entwicklung der Photonenzahl bei steigender Anzahl der Prozessoren

Abbildung 18 zeigt die Entwicklung der Anzahl der gerechneten Photonen für die erste Teilquelle, Zink 213 nm, bei steigenden Prozessorzahlen. Bei großen Prozessorzahlen erreichen die Teilchenzahlen die theoretisch erwarteten Werte. Die Zuteilung der Rechenzeit nach der Quellstärke der Teilquelle bleibt also erhalten.
Im Bereich unterhalb von 29 Prozessoren sieht man allerdings deutlich, dass das bislang implementierte Verfahren für die Verteilung der Prozessoren in diesem Fall noch nicht optimal ist. Ist die Anzahl der Prozessoren NPES kleiner als die Anzahl der Quellen NSTRA, so muss jeder Prozessor NSTRA/NPES Quellen bearbeiten. Dadurch erhält Linie 1 weniger Rechenleistung als ihr nach ihrer Quellstärke zusteht. Hier sollte das Zuteilungsverfahren für die Rechenzeit verbessert werden. Dazu muss man sich von dem Konzept trennen, dass im Falle NSTRA>NPES alle Prozessoren gleich viele Quellen bearbeiten. Wahrscheinlich wird es auf eine Mischform hinauslaufen, bei der Quellen mit hoher Quellstärke von mehreren

Prozessoren bearbeitet werden, während sich Quellen mit geringer Quellstärke einen Prozessor teilen müssen.

Das folgende Bild (Abbildung 19) zeigt die „Figure of Merit“ (Gleichung (15)) für den zweiten Testfall. Wie man sieht bleibt die Kenngröße sowohl für die Einzelquelle als auch für das Gesamtergebnis aus allen Quellen bei wachsenden Prozessorzahlen im Wesentlichen konstant. Dies ist ein erfreuliches Ergebnis, denn daraus kann man schließen, dass die Berechnung der statistischen Größen wie zum Beispiel die Standardabweichung im Rahmen der Parallelisierung korrekt umgesetzt wurde.
Bei der Berechnung der Standardabweichung

$$\sigma = \sqrt{\sigma^2}, \quad \sigma^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right]$$

muss die Reihenfolge der Berechnungen beachtet werden. Zunächst werden bei der Akkumulation der Ergebnisse einer Teilquelle, die von mehreren Prozessoren bearbeitet wurde, die Anteile der einzelnen Prozessoren zu $\sum_{i=1}^{n} x_i^2$ und zu $\sum_{i=1}^{n} x_i$ zusammengeführt. Erst danach kann der Ausdruck für σ berechnet werden. Entsprechendes gilt für die Summe über alle Quellen, das Gesamtergebnis: es müssen erst die Beiträge von allen Teilquellen vorliegen, bevor die Standardabweichung für das Gesamtergebnis berechnet werden kann.

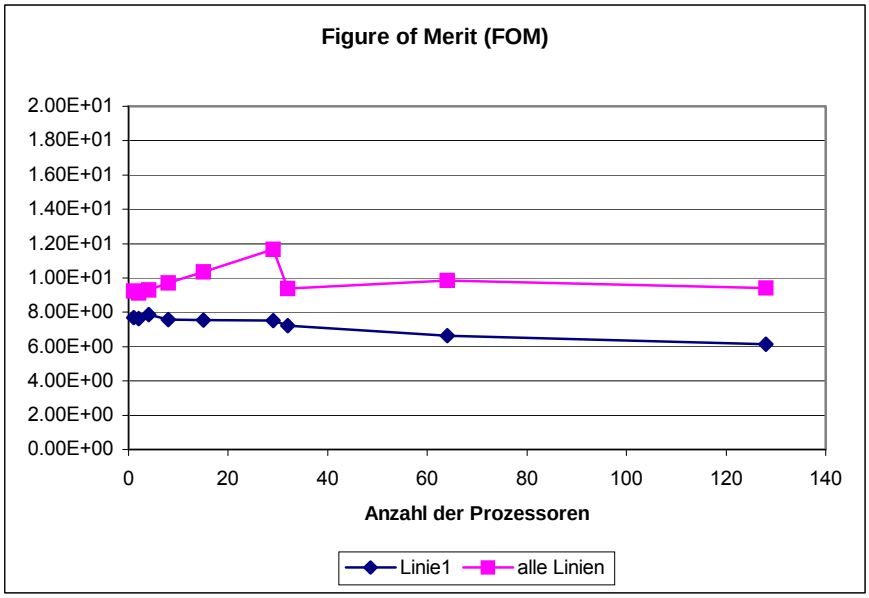


Abbildung 19 Figure of Merit für den zweiten Testfall

In Abbildung 20 wird die Standardabweichung der Energiequelle sowohl für Linie 1 als auch für das Gesamtergebnis dargestellt. Beide Kurven zeigen das gleiche Verhalten, da Linie 1 das Gesamtergebnis dominiert. Im Bereich bis zu 29 Prozessoren fällt die Kurve ab und scheint eine Sättigung zu erreichen. Die ist der Fall, da Linie 1 in diesem Bereich nur wenig zusätzlich Rechenzeit erhält. Im Bereich des Bildes mit mehr als 29 Prozessoren fällt die Kurve dann deutlich nach unten ab, da Linie 1 jetzt mehr Rechenleistung erhält und damit die Varianz und die Standardabweichung kleiner werden.

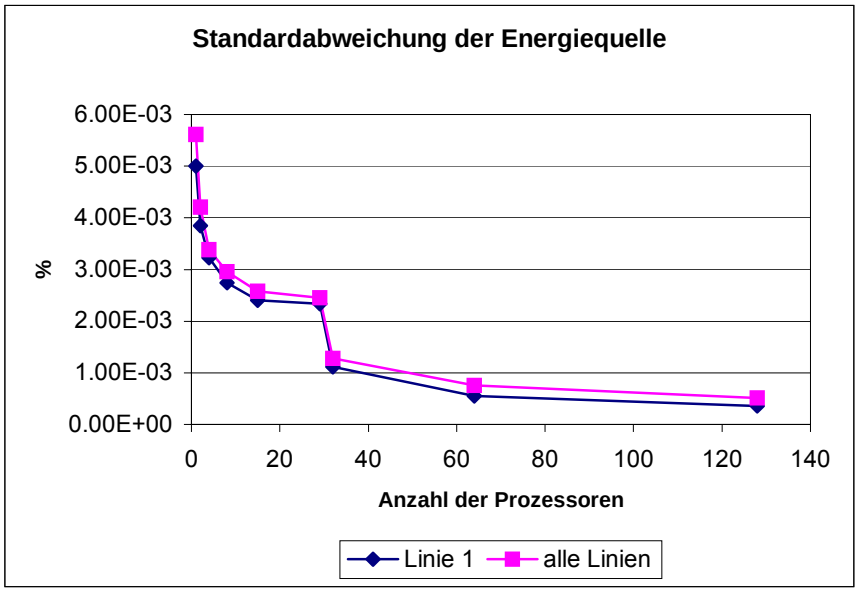


Abbildung 20 Standardabweichung der Energiequelle

7 Ergebnisdarstellung, Visualisierung in 3D

Ein weiteres Problem im Mikro-Modul B, das zwar nicht ausschließlich aus der Kopplung an den Makro-Code resultiert, aber dennoch gelöst werden muss, ist die Frage der Ergebnisdarstellung. Spätestens mit der Nutzung von dreidimensionalen Rechengittern stellt sich dieses Problem, denn räumlich verteilte Ergebnisse sind als Zahlen auf Papier nicht besonders anschaulich. Man benötigt demzufolge eine grafische Ausgabe.

Da es sich bei den verwendeten Tetraedern (Kapitel 4) um Grundformen von Finite-Elemente Codes handelt, bietet es sich an, ein Programm zur Darstellung von Ergebnissen von Finite-Elemente Rechnungen, zu verwenden. In dieser Arbeit wurde dazu RAPS /26/ gewählt.
RAPS erwartet als Input drei Dateien. Die erste Datei enthält die Koordinaten der darzustellenden Geometrie. Die zweite Datei beschreibt die aus den Koordinaten zu bildenden Elemente durch Zuordnung der Koordinatennummern zu den Elementen. Der dritte File schließlich enthält für jeden Punkt der Geometrie die Werte der darzustellenden Größen.
Das Erzeugen dieser Dateien, stellt kein Problem dar, da die benötigten Geometrieinformationen bereits im Programm vorliegen. Bei Erstellen der dritten Datei mit den darzustellenden Werten muss allerdings wiederum eine Interpolation der zu darzustellenden Werte von den Zellmitten auf die Knoten erfolgen. Dies geschieht in gleicher Weise wie bei der Interpolation der Quellen und Senken für FIDAP, siehe Kapitel 5.3.

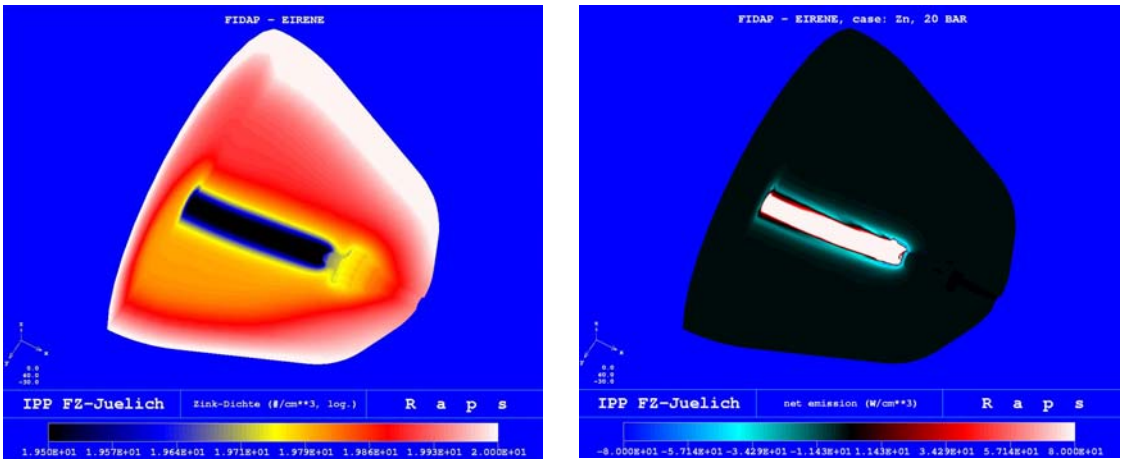


Abbildung 21 Ergebnisdarstellung mit RAPS-Graphik. Links ist die Verteilung der Zinkdichte, logarithmisch skaliert, dargestellt, rechts die Nettoemission

Abbildung 21 zeigt beispielhaft, die Darstellung von Eingabegrößen aus dem Makro-Code und Ergebnissen des Mikro-Codes mit Hilfe von RAPS-Graphik. Das linke Bild zeigt die Verteilung der Zink-Dichte in der Lampe mit logarithmischer Skala. Wie man sieht, ist Zink im Hintergrundmedium nahezu gleichmäßig verteilt vorhanden. Nur im Bereich des Lampenbogens, ist die Zinkdichte etwas herabgesetzt. Das rechte Bild zeigt die Verteilung der Nettoemission. Im Zentrum des Lampenbogens überwiegt die Emission, am Rand des Bogens (der blaue Saum) sieht man Bereiche, in denen die Absorption dominiert.

Wie Abbildung 21 zeigt, ist diese Art der Darstellung für die in dieser Arbeit behandelten Lampen ausreichend. Diese haben zwei Symmetrieebenen, so dass die Berechnung eines Viertels der Originallampe genügt. Durch Drehen der Geometrie und Betrachtung der Symmetrieflächen hat man einen guten Blick ins Innere des Rechengebiets. Sollte es allerdings notwendig werden, vollständige Lampen zu rechnen, hat man erneut ein Darstellungsproblem, denn RAPS stellt immer das ganze Rechenvolumen dar. Dann schaut man von Außen auf die Lampe und sieht nicht, was im Inneren passiert. Man kann zwar Teile des Rechen-

gebietes ausschneiden, die Schnittkanten sind aber gegebenenfalls nicht glatt, da RAPS immer ganze Basiselemente darstellt.
Ebenso ist es nicht möglich, einen Schnitt durch die Geometrie zu legen und diesen Querschnitt darzustellen. Man kann nur von den Rändern des Rechengebiets ausgehend Schichten von Basiselementen abtragen. Allerdings ist bei räumlich unstrukturierten Gittern die Definition einer Schicht eher vage.

Um diesem Problem zu begegnen, wurde ein Verfahren entwickelt, das es erlaubt, eine beliebige Ebene durch ein gegebenes Tetraedergitter zu legen und den Schnitt graphisch darzustellen
.
Dazu überlegt man sich als Erstes, welche Formen der Schnitt zwischen einer Ebene und einem Tetraeder annehmen können. Es können vier mögliche Schnittgebilde entstehen:

1. Die Ebene schneidet den Tetraeder in einem Eckpunkt.
2. Eine Kante des Tetraeders liegt in der Ebene.
3. Die Ebene schneidet drei Kanten des Tetraeders. Die Schnittfläche ist ein Dreieck. Die Eckpunkte des Dreiecks sind die Schnittpunkte der Tetraederkanten mit der Ebene.
4. Die Ebene schneidet vier Kanten des Tetraeders. Die Schnittfläche ist ein ebenes Viereck, dessen Eckpunkte durch die Schnittpunkte der Tetraederkanten mit der Ebene gegeben sind.

Die beiden ersten Möglichkeiten sind für die Darstellung des Schnitts irrelevant und können vernachlässigt werden. Bei den beiden anderen Möglichkeiten fällt auf, dass jeweils die Schnittpunkte der Ebene mit den Kanten des Tetraeders von Interesse sind. Außerdem lassen sich beide Schnittgebilde leicht durch Dreiecke darstellen. Es bietet sich daher an, den Schnitt durch das Tetraedergitter in Dreieck zu zerlegen und die so gewonnene Triangulierung des Schnitts wieder mit RAPS darzustellen.

Um den Schnitt der Ebene mit dem Rechengebiet zu bestimmen, ist es notwendig, alle Tetraeder mit der Ebene zu schneiden. Wie bereits erwähnt, werden die Schnittgebilde durch die Schnittpunkte der Tetraederkanten mit der Ebene bestimmt.

Seien $P_1 = (x_1, y_1, z_1)^T$ und $P_2 = (x_2, y_2, z_2)^T$ Eckpunkte eines Tetraeders und somit Endpunkte einer Tetraederkante $\vec{k}$. Dann lässt sich $\vec{k}$ schreiben als

$$\vec{k} = P_1 + t \cdot \vec{v}, \text{ mit } \vec{v} = P_2 - P_1. \tag{45}$$

Sei weiterhin

$$a_0 + a_1 x + a_2 y + a_3 z = a_0 + \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0 \tag{46}$$

die Gleichung der Schnittebene. Einsetzen von $\vec{k}$ in Gleichung (46) führt

$$t = \frac{-a_0 - \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}}{\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}}. \tag{47}$$

Verschwindet das Skalarprodukt im Nenner von Gleichung (47), so ist die Kante $\vec{k}$ parallel zur Schnittebene. $\vec{k}$ muss in diesem Falle nicht weiter berücksichtigt werden, denn wenn $\vec{k}$ in der Ebene liegt, haben zwei andere Kanten des Tetraeders echte Schnittpunkte mit der Ebene.
Gilt $0 \leq t \leq 1$, so schneidet die Ebene die Tetraederkante.

Leider ist es aber nicht ausreichend, einfach die Schnittpunkte aller Tetraeder mit der Eben zu berechnen und die entstehenden Dreiecke zu sammeln. Dadurch entsteht keine sinnvolle Triangulierung der Schnittebene, sondern nur ein Flickenteppich aus nichtzusammenhängenden Dreiecken. Um eine sinnvolle Triangulierung zu erhalten, müssen sich benachbarte Dreiecke jeweils die beiden Eckpunkte der gemeinsamen Dreiecksseite teilen. Dies ist aber nicht automatisch der Fall. Dazu müsste beim Sammeln der Dreiecke jeweils überprüft werden, ob die Eckpunkte des neuen Dreiecks bereits bekannt sind.
Bei der Entscheidung, wann zwei Punkte gleich sind, muss man unterscheiden, ob ein Tetraeder nahe einer Ecke geschnitten wird und wirklich drei dicht beieinanderliegende Schnittpunkte vorhanden sind, oder ob eine Kante mehrfach geschnitten wurde, da sie zu mehreren Tetraedern gehört und dabei Rundungsfehler den Schnittpunkt verfälschen. Um diese Probleme zu umgehen, wurde anderer Ansatz gewählt.

Ausgangspunkt ist der Gedanke, dass sich das Tetraedergitter auch als Menge von Kanten auffassen läßt. Jede Kante gehört dabei zu einem oder mehreren Tetraedern und ist eindeutig durch ihre Endpunkte bestimmt. Jede Kante hat entweder keinen Schnittpunkt mit der Ebene oder genau einen. Dieser kann eindeutig bestimmt werden.

Aufgrund dieser Überlegungen kommt man zu folgender Vorgehensweise:

1. bestimme die Menge der Kanten,
2. berechne die Schnittpunkte der Kanten mit der Schnittebene,
3. führe eine Koordinatentransformation durch, um Koordinaten in der Ebene zu erhalten
4. bilde die Triangulierung aus den Schnittpunkten
5. ordne den Dreiecken Werte zu.

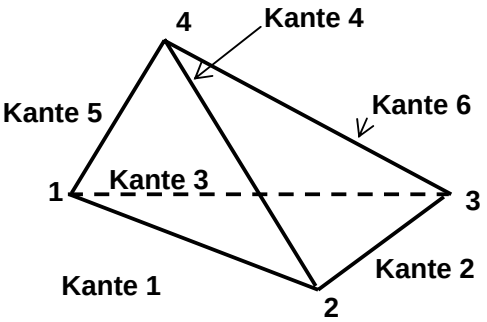
Wie diese einzelnen Punkte realisiert wurden, wird im Folgenden beschrieben.

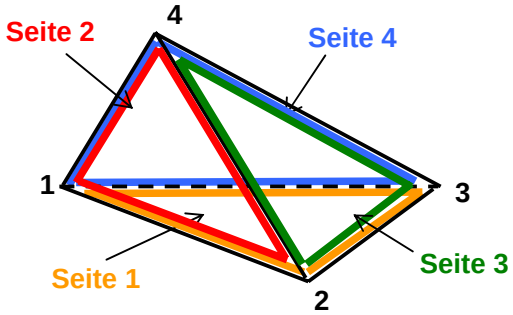
1. bestimmen der Kantenmenge

Vor der Bestimmung der Menge der Kanten in einem Tetraedergitter überlegt man sich welche Eigenschaften eine Kante besitzt.

- Eine Kante ist definiert durch ihre beiden Endpunkte.
- Jeder Tetraeder besitzt sechs Kanten.
- Jede Kante trennt zwei Tetraederseiten voneinander.

Diese Eigenschaften kann man sich zu Nutze machen.





Dazu nummeriert man die Kanten eines Tetraeders folgendermaßen durch:

Nummer der Kante	linker Endpunkt der Kante	rechter Endpunkt der Kante
1	1	2
2	1	3
3	1	4
4	2	3
5	2	4
6	3	4

Aufgrund der Nummerierung ist zudem bekannt:

Kante	trennt die Seiten
1	1 und 2
2	1 und 4
3	2 und 4
4	1 und 3
5	2 und 3
6	3 und 4

Ferner kennt man zu jeder Tetraederseite die Nummer des Nachbartetraeders und die Seitennummer im Nachbartetraeder.
Mithilfe dieser Informationen ergibt sich folgendes Verfahren:

Sei *tetra_kanten*(6,ntet) ein Feld, das zu jeder Kante eines Tetraeder die Kantennummer enthält. Sei ferner *kanten*(2,3*ntet) ein Feld, das zu jeder Kante die Nummer der Endpunkte enthält, und *nkanten* die Anzahl der gefundenen Kanten.

Zu Beginn sind *tetra_kanten = kanten = nkanten = 0*
Man durchläuft nun alle Tetraeder *itet* und für jeden Tetraeder alle sechs Kanten *ik*. Für jede Kante prüft man, ob *tetra_kanten(ik,itet)* bereits bekannt ($\neq$0) ist. Ist dies der Fall, so muß nichts getan werden, und man kann mit der nächsten Kante fortfahren. Andernfalls ist die Kante neu. Jetzt wird die Anzahl der Kanten *nkanten* erhöht und die Nummern *ic* und *jc* der Endpunkte der Kante bestimmt. Man setzt

kanten(1,nkanten) = ic
kanten(2,nkanten) = jc
tetra_kanten(ik,itet) = nkanten.

Dann bestimmt man die Nummer der ersten an die Kante *ik* angrenzenden Tetraederseite *no_side*. Nun durchläuft man alle Nachbarn des Tetraeders, die sich die Kante *ik* teilen. Dazu bestimmt man zunächst die Nummer des Tetraeders *nxt_tet*, der an die Seite *no_side* angrenzt. Ist *nxt_tet* = 0, so hat man den Rand des Rechengebiets erreicht. Man kehrt dann zum Tetraeder *itet* zurück und setzt das Verfahren mit der zweiten an die Kante *ik* angrenzenden Tetraederseite fort, indem man wieder *no_side* und *nxt_tet* bestimmt. Zu *nxt_tet* bestimmt man dann *nxt_side* die Nummer der Seite, die an *no_side* grenzt. Dann ermittelt man die Nummer der Kante *nxt_edge* im Tetraeder *nxt_tet*. Auch diese Kante wird mit der Kantennummer *nkanten* markiert.

tetra_kanten(nxt_edge, nxt_tet) = nkanten.

Anschließend ermittelt man die Seitennummer der zweiten an *nxt_edge* angrenzenden Tetraederseite durch

no_side = angrenzende_seiten(1,nxt_edge) +
angrenzende_seiten(2,nxt_edge) - nxt_side

und setzt das Verfahren mit dem Nachbarn von nxt_tet fort. Das Verfahren endet, wenn *nxt_tet = itet* ist, oder wenn zweimal der Rand des Rechengebiets erreicht wurde.

2. Berechnung der Schnittpunkte der Kanten mit der Schnittebene

Zur Berechnung der Schnittpunkte der Kanten mit der Schnittebene werden alle unter 1. gefundenen Kanten durchlaufen und Gleichung (47) für jede Kante ausgewertet. Für alle Kanten mit gültigem t, $0 \le t \le 1$, werden die Koordinaten der Schnittpunkte berechnet. Um doppelt auftretende Schnittpunkte zu erkennen und zu eliminieren, werden alle gefundenen Schnittpunkte in einen balancierten, binären Sortierbaum eingefügt. Zwar besitzt jeder Kante maximal einen Schnittpunkt, dennoch können Schnittpunkte doppelt auftreten, wenn Zum Beispiel zwei Kanten von der Ebene in ihrem gemeinsamen Endpunkt geschnitten werden. Der Sortierbaum enthält in seinen Knoten die Koordinaten der Schnittpunkte und eine laufende Nummer. Die Schnittpunkte werden zuerst nach ihren x-Koordinaten sortiert eingefügt. Sind die x-Werte zweier Punkte gleich, so wird nach den y-Koordinaten entschieden. Sind auch diese gleich, so werden schließlich die z-Koordinaten zu Rate gezogen. Sind zwei Punkte gleich, so wird der zweite Punkt nicht in den Baum eingefügt, sondern die laufende Nummer des bereits gespeicherten Punkts zurückgemeldet.
Man merkt sich zu jeder Kante das gefundene t, die Nummer der Schnittpunktkoordinaten und zu jedem Schnittpunkt die Nummer der ersten Kante, zu der der Punkt gehört.

3. Berechnung der Koordinatentransformation

Nachdem die Schnittpunkte der Tetraederkanten mit der Schnittebene bekannt sind, müssen sie noch in ebene Koordinaten umgerechnet werden. Dazu benötigt man eine Orthonormalbasis in der Schnittebene. Aus Gleichung (46) erhält man $\vec{a} = (a_1, a_2, a_3)^T$, den Normalenvektor der Ebene. Ein Vektor $\vec{b}$, der senkrecht auf $\vec{a}$ steht, liegt demzufolge in der Ebene. Man berechnet $\vec{b}$ leicht aus

$$a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 = 0 \tag{48}$$

Einen Vektor $\vec{c}$, der auf $\vec{a}$ und $\vec{b}$ senkrecht steht, erhält man aus dem Kreuzprodukt von $\vec{a}$ und $\vec{b}$

$$\vec{c} = \vec{a} \times \vec{b} \tag{49}$$

Die Vektoren $\vec{b}$ und $\vec{c}$ liegen in der Schnittebene und sind orthogonal.
Mit Hilfe des Spatprodukts $(\vec{a} \times \vec{b})\vec{c}$ kann man prüfen, ob, $\vec{a}$, $\vec{b}$ und $\vec{c}$ ein Rechtssystem bilden. Ist die nicht der Fall, so ist $\vec{c} = -\vec{c}$ zu setzen. Durch Normieren der beiden Vektoren $\vec{b}$ und $\vec{c}$ erhält man schließlich die gesucht Orthonormalbasis der Schnittebene. Als Ursprung $\vec{o}$ des neuen Koordinatensystems wählt man zum Beispiel den ersten gefunden Schnittpunkt einer Tetraederkante mit der Schnittebene.

Um die Koordinatentransformation durchzuführen muss für jeden Schnittpunkt $(x, y, z)^T$ folgendes Gleichungssystem gelöst werden:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix} + x_{tri} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} + y_{tri} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \quad \Leftrightarrow \quad \begin{pmatrix} b_1 & c_1 \\ b_2 & c_2 \\ b_3 & c_2 \end{pmatrix} \begin{pmatrix} x_{tri} \\ y_{tri} \end{pmatrix} = \begin{pmatrix} x - o_x \\ y - o_y \\ z - o_z \end{pmatrix} \tag{50}$$

Durch Multiplikation mit

$$\begin{pmatrix} b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix}$$

erhält man das reduzierte Gleichungssystem

$$\begin{pmatrix} \vec{b}^2 & \vec{b}\vec{c} \\ \vec{b}\vec{c} & \vec{c}^2 \end{pmatrix} \begin{pmatrix} x_{tri} \\ y_{tri} \end{pmatrix} = \begin{pmatrix} \vec{b}(\vec{P}-\vec{o}) \\ \vec{c}(\vec{P}-\vec{o}) \end{pmatrix},$$

wenn $\vec{P}$ den zu transformierenden Schnittpunkt bezeichnet.

Da sich für die verschiedenen Schnittpunkte nur die rechte Seite des Gleichungssystems ändert, muss die Matrixinversion nur einmal ausgeführt werden. Es ist

$$\begin{pmatrix} \vec{b}^2 & \vec{b}\vec{c} \\ \vec{b}\vec{c} & \vec{c}^2 \end{pmatrix}^{-1} = \frac{1}{\vec{b}^2\vec{c}^2 - (\vec{b}\vec{c})^2} \begin{pmatrix} \vec{c}^2 & -\vec{b}\vec{c} \\ -\vec{b}\vec{c} & \vec{b}^2 \end{pmatrix} .$$

Damit erhält man

$$\begin{pmatrix} x_{tri} \\ y_{tri} \end{pmatrix} = \frac{1}{\vec{b}^2\vec{c}^2 - (\vec{b}\vec{c})^2} \begin{pmatrix} \vec{c}^2 & -\vec{b}\vec{c} \\ -\vec{b}\vec{c} & \vec{b}^2 \end{pmatrix} \begin{pmatrix} \vec{b}(\vec{P}-\vec{o}) \\ \vec{c}(\vec{P}-\vec{o}) \end{pmatrix}.$$

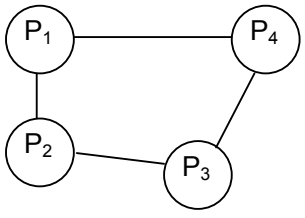
4. Aufstellen der Triangulierung

Die Triangulierung gewinnt man, indem alle Tetraeder durchlaufen und daraufhin untersucht werden, ob ihre Kanten drei oder vier unterschiedliche Schnittpunkte aufweisen. Hat ein Tetraeder drei Schnittpunkte, so hat man direkt ein Dreieck gefunden. Die Nummern der Schnittpunkte werden in den Array für die Elementzuordnung übernommen, nachdem die Orientierung der Punkte getestet wurde. Raps erwartet die Eckpunkte der Dreiecke im Gegenuhrzeigersinn. Die Orientierung kann bestimmt werden, indem man die gerichtete Fläche des Dreiecks berechnet. Es ist

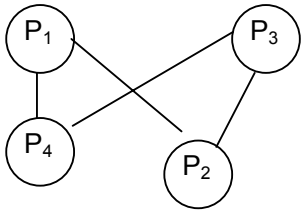
$$A = \frac{1}{2}\left[x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)\right] . \qquad (51)$$

A ist größer 0, wenn die Eckpunkte des Dreiecks im Gegenuhrzeigersinn durchlaufen werden. Ist *A*<0, so müssen zwei Punkte vertauscht werden.

Besitzt ein Tetraeder vier Schnittpunkte, so muss zunächst überprüft werden, ob sich aus den Schnittpunkten in der Reihenfolge, wie sie gefunden wurden, ein reguläres Viereck bilden lässt, oder ob sich die Verbindungslinien der Punkte schneiden würden.



richtig



falsch

Ist dies der Fall, so müssen solange Punkte vertauscht werden, bis ein reguläres Viereck entstanden ist. Dieses Viereck wird dann in zwei Dreiecke zerlegt und deren Orientierung überprüft.

Zu jedem Dreieck merkt man sich die Nummer des Tetraeders, aus dem es entstanden ist. Ferner wird zu jedem Dreieck auch der Schwerpunkt des Dreiecks berechnet, da er für die Interpolation der Werte vom Innern der Dreiecke zu den Knotenpunkten benötigt wird.

5. Zuordnen der Werte zu Dreiecken

Bei der Aufstellen der Triangulierung wurde zu jedem Dreieck vermerkt, aus welchem Tetraeder es entstanden ist. Diese Beziehung wird nun benutzt, um den Dreiecken Werte zuzuordnen. In einer Schleife über die Dreiecke wird jedem Dreieck der Wert zugewiesen, der dem zugehörigen Tetraeder entspricht.

Das vollständige Programm RPSCUT ist im Anhang aufgeführt.

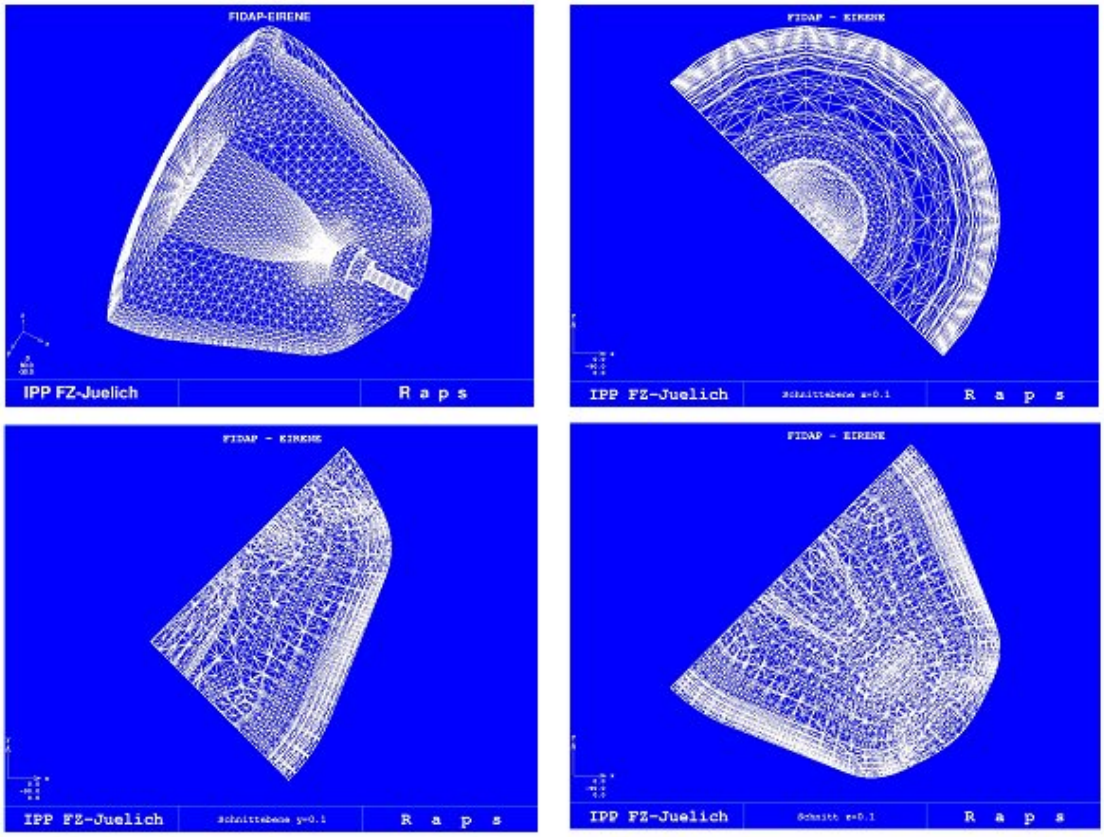


Abbildung 22 ein 3D Rechengitter und Schnitte in jeder Ebene

Das Bild in Abbildung 22, oben links zeigt ein dreidimensionales Rechengitter für eine Hochdruckplasmalampe wie sie in der konkreten Anwendung verwendet wird. Die übrigen drei Teilbilder stellen Schnitte durch das Rechenvolumen. Die Ausdehnung des Lampenviertels beträgt ungefähr 1 cm in x-Richtung ($0 < x < 1$), 1 cm in y-Richtung ($-0.5 < y < 0.5$) und 0.5 cm in z-Richtung ($0 < z < 0.5$). Der Schnitt im Bild oben rechts befindet sich bei x=0.1, das Bild unten links zeigt y=0.1 und im Bild unten rechts ist z=0.1 dargestellt.

Auch in Abbildung 23 ist diese Anordnung beibehalten worden. Es ist exemplarisch die Elektronendichte dargestellt. In allen vier Teilbildern wurde die gleiche logarithmische Skalierung verwendet. In den Schnittdarstellungen sieht man, dass den Bereichen, wo Zellen unterschiedlicher Größe aneinander grenzen, das Interpolationsverfahren Schwierigkeiten hat. In den „ausgefransten“ Säumen kann man deutlich die Zellgrößen erkennen.

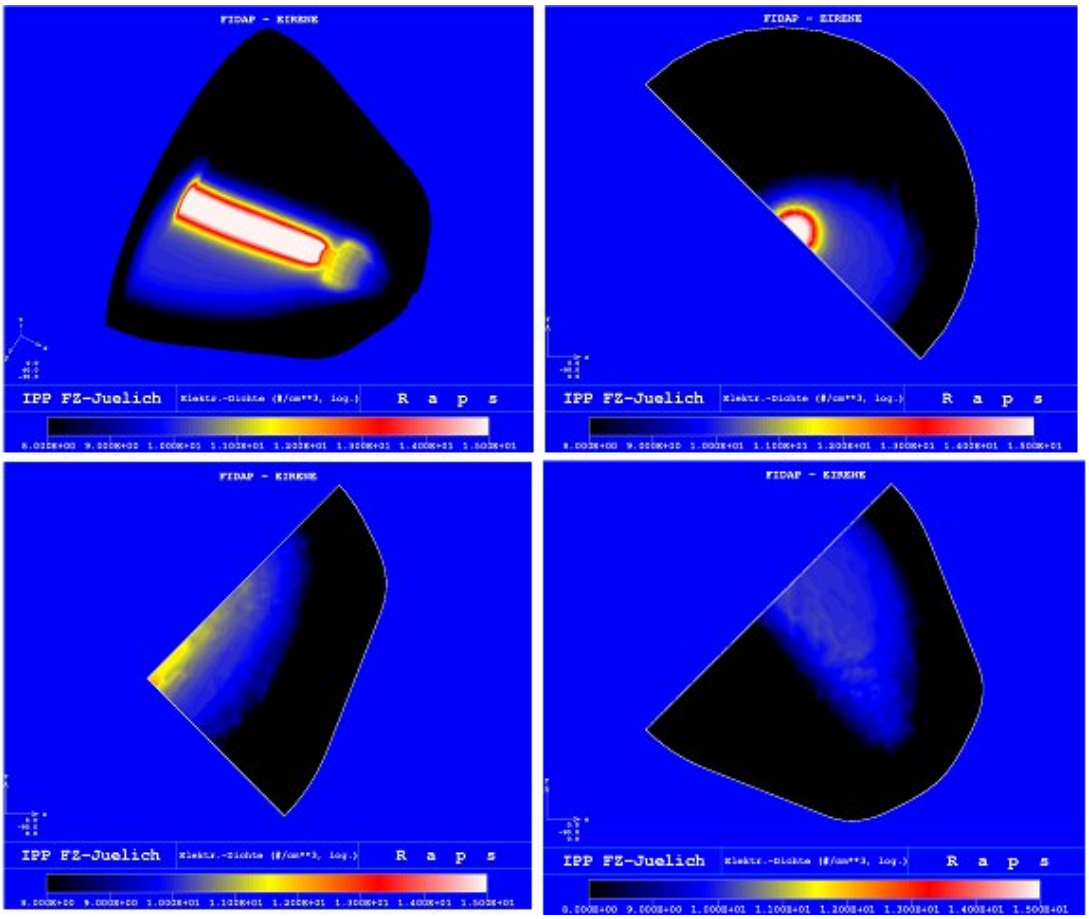


Abbildung 23 die Elektronendichte, exemplarisch dargestellt in 3D und in den Schnittebenen x=0.1, y=0.1, z=0.1

8 Zusammenfassung, Ausblick

Zusammenfassung

Seit 25 Jahren werden im Institut für Plasmaphysik des Forschungszentrum Jülich Monte Carlo Codes für Transportprobleme aus der Kernfusionsforschung entwickelt. Diese simulieren physikalische Vorgänge auf mikroskopischem (hier atomarem, molekularem) Niveau. Ab ca. 1990 gelang es, diese mit makroskopischen Strömungscodes (CFD für magnetische Plasmen) zu koppeln und so konsistente Gesamtsimulationen des komplexen Systems „Plasmarandschicht eines Fusionsreaktors“ zu erhalten. Im Rahmen eines vom Bundesministerium für Bildung und Forschung (BmBF) finanzierten Verbundprojektes zwischen industriellen und öffentlich geförderten Partnern wurden diese spezifischen Programmentwicklungen verallgemeinert zu allgemeinen Algorithmen und Implementierungen für sogenannte Mikro-Makro Probleme in Wissenschaft und Technik. Insbesondere wurde der fusionsspezifische Makro-Modul (CFD) durch einen allgemein „standardisierten“ und kommerziell verfügbaren „Finite-Elemente Code“ ersetzt. Die sich bei Vernetzung und iterativer Kopplung von FEM- und MC-Codes ergebenden Fragen und notwendigen Algorithmen sind Gegenstand der vorliegenden Arbeit. In allgemeinen Diskussionen wurde erläutert, dass diese Vernetzung der komplementären, separierten Codes aus Gründen der inneren Konsistenz des Gesamtsystems eine weitestgehende Gitterharmonisierung erfordert. Hierzu sind im Wesentlichen algebraische Aufgaben effizient und genau zu lösen (Bestimmung von Nullstellenmengen von Systemen algebraischer Gleichungen).
Des weiteren ergeben sich spezifische Aufgaben bei der Speicherplatzverwaltung und dem Datentransfer zwischen beiden Modulen.
Anhand zweier spezieller, völlig unabhängig voneinander entwickelter Codes: FIDAP (FEM-Makro-Modul) und EIRENE (MC-Mikro-Modul) werden diese Aufgaben diskutiert und, teilweise approximativ, gelöst. Bei den hier beschriebenen Prozeduren wird oft das makroskopische Medium mit „Plasma“ bezeichnet und die mikroskopischen Objekte mit „Photonen“, weil dies bei der konkreten Aufgabe im BmBF Projekt die vorgesehene Anwendung (Hochdruckplasmalampen) war. Die Ausführungen in dieser Arbeit haben aber generelle Bedeutung für „Mikro-Makro“ Probleme. Mit den insbesondere in Kapitel 4-7 beschriebenen Verfahren kann in der Tat nun ein Höchstmaß an Konsistenz zwischen den beiden konzeptuell völlig gegensätzlichen Codes erzeugt werden.

Ausblick

Weitere Entwicklungen auf diesem Arbeitsgebiet dürften zum einen bei der Parallelisierung des Monte Carlo Teils innerhalb des Gesamtcodes liegen, da dies, wegen der grundsätzlich nicht zu beeinflussenden Konvergenzrate (proportional $1/\sqrt{CPU}$), die einzige Möglichkeit ist, „turn-around“ Zeiten zu reduzieren (derzeit bei Fusionsanwendungen: typisch bis zu 3 Monten auf einem PC).
Insbesondere das „load balancing“ bei „stratified sampling“ scheint noch verbesserungsfähig. Die Verteilung der CPU-Zeit auf die Teilquellen proportional zum Fluss, der den verschiedene Strata zugeordnet ist, unter gleichzeitiger Berücksichtigung unterschiedlich langer mittlerer Lebensdauern der Testteilchen ist ein Aspekt, der verstärkt Beachtung verdient.

Zum zweiten scheinen vor allem schnelle, algebraisch exakte (d.h. keine numerische Iteration) Solver für Systeme algebraischer Gleichungen bis zu der Ordnung, in der Flächen zwischen Basiselementen in FEM definiert sind, erforderlich zu sein, z.B. als schnelle „intrinsische Funktion“ auf den Rechnerplattformen.

Auch auf dem Gebiet der Gitterharmonisierung gibt es noch weiteren Entwicklungsbedarf. Wie in Kapitel 5.2 beschrieben, stoßen die aktuell verwendeten Diskretisierungen der Rechengebiete an ihre Grenzen. Bei sich verändernden Gasparametern werden lokale Gra-

dienten durch feste, unveränderliche Rechengitter nicht mehr optimal aufgelöst. Hier wird man Verfahren zur Gitteradaption entwickeln müssen, die die Zellgrößen der Rechengitter wenigstens im Mikro-Code an die lokalen Gradienten anpassen.

Im Lauf der Iterationen des gekoppelten Gesamtsystems gewinnt ein weiteres Problem, das bislang noch nicht diskutiert wurde, an Bedeutung. Je besser das gekoppelte System konvergiert ist, desto mehr Einfluss auf die Konvergenz gewinnen die Regionen des Rechengebiets, die aufgrund ihrer Lage oder der dort herrschenden Plasmaparametern von weniger Teilchen besucht werden. Hier ist die Statistik im Allgemeinen nicht sehr gut. Wenn die Rechnung in den wichtigeren Region konvergiert ist, reagiert das gekoppelte System im Wesentlichen nur noch auf das statistische Rauschen in den weniger wichtigen Regionen des Rechengebiets. Ein Teil dieses Rauschens resultiert daraus, dass aufgrund der geänderten Hintergrundparameter im Modul A die Testteilchen im Modul B in jeder Iteration entlang anderer Wege laufen. Das führt dazu, dass im Laufe der Iterationen einige Zellen einmal häufiger und einmal weniger häufig besucht werden.
Um diesem Problem zu begegnen, wurde schon vor einiger Zeit das sogenannte Correlationsampling im Mikro-Code EIRENE implementiert. Dieses Verfahren sorgt dafür, dass die erste Zufallszahl eines jeden Teilchens über die verschiedenen Iterationen hinweg gleich bleibt. Dadurch versucht man die Teilchentrajektorien in den verschiedenen Iterationen möglichst ähnlich zu gestalten. Dieses Verfahren ist sicherlich eine hilfreiche Option, beseitigt das oben genannte Problem aber nicht vollständig.

Daher wird momentan darüber nachgedacht, das Problem auf andere Art zu lösen. Das durch die unterschiedlichen Teilchenbahnen verursachte statistische Rauschen würde vollständig verschwinden, wenn man garantieren könnte, dass die Testteilchen in jeder Iteration den gleichen Weg nehmen. Dies könnte man erzwingen, indem man in der ersten Iteration die Teilchenbahnen abspeichert. In allen weiteren Iterationen wären dann nur noch diese Wege abzulaufen und die Ergebnistallies entsprechend der geänderten Plasma- und Gasbedingungen neu zu berechnet. Das Verfahren wäre vom Rechenzeitaufwand sicherlich ebenfalls interessant. Allerdings ist der Aufwand an Speicherplatz nicht zu vernachlässigen.
Man benötigt etwa eine Million Trajektorien für eine brauchbare Statistik. Die aktuellen Rechengitter haben etwas mehr als 100000 Zellen. Angenommen jedes Teilchen durchläuft 1% der Zellen, dann benötigt man pro abgespeicherter Information

1000000 Teilchen * 1000 Zellen * 8 Byte = 8000000000 Byte = 8 GByte.

Die benötigten Informationen beinhalteten mindestens die Nummer der durchlaufenen Zelle, die in der Zelle zurückgelegte Flugstrecke, die Energie und das Gewicht des Teilchens. Damit ist man bei mindestens 32 GByte zusätzlich benötigtem Speicherplatz.
Diese Datenmenge kann nicht im Speicher gehalten werden, sondern die Daten müssen notwendigerweise in Dateien ausgelagert werden. Dann hat man aber den langsamen I/O zu berücksichtigen, der den erwarteten Rechenzeitgewinn mehr als auffressen wird. Dies ist auf alle Fälle ein Szenario, dass noch einige Überlegungen verlangt.

9 Literaturverzeichnis

/1/ H.K. Versteeg, W. Malalasekera, “An Introduction to Computational Fluid Dynamics, The Finite Volume Method”, Longman Scientific & Technical (1995)

/2/ S. V. Patankar, “Numerical Heat transfer and fluid flow”, Hemisphere Publ. Corporation, Taylor & Francis Group, New York (1980)

/3/ O.C. Zienkiewicz, R.L. Taylor, “The finite element method” McGraw-Hill, New York (1991)

/4/ D. Reiter, “Progress in 2-dimensional plasma edge modelling”, J. Nucl. Mat. 196-198, (1992) 80-89

/5/ D. Reiter et al., „EIRENE – A Monte Carlo Linear Transport Solver“, verfügbar im Internet unter http://www.eirene.de (2002 - …).

/6/ D.Reiter, M. Baelmans, P Börner, Fus. Sci. Eng., **47**, (2005) p172

/7/ FIDAP Theory Manual, FLuent Inc., Lebanon, NH, 1998

/8/ J.F. Thompson, Z.U. Warsi, C.W. Mastin, “Numerical grid generation” North-Holland, New York (1985)

/9/ C.J. Everett and E.D. Cashwell, “A Monte Carlo Sampler”, Pergamon Press (1959)

/10/ M.P. Cohn, „Algebra, Second Edition, Volume 2“, John Wiley & Sons (1989)

/11/ J. Naas, H.L. Schmid, „Mathematisches Wörterbuch“, B. G. Teubner Verlagsgesellschaft, Stuttgart, (1961) (Satz von Abel)

/12/ J. Spanier, E.M. Gelbard, „Monte Carlo Principles and Neutron Transport Problems“, Addison Wesley, Reading, Massachusetts, (1969)

/13/ J. M. Hammersley, D.C. Handscomb, “Monte Carlo Methods”, Methuen & Co LTD (1964)

/14/ M.H. Kalos, P.A. Whitlock, “Monte Carlo Methods, Volume 1: Basics”, John Wiley & Sons (1986)

/15/ R.Y. Rubinstein, “Simulation and the Monte Carlo Method”, John Wiley & Sons (1981)

/16/ L. Devroye, “Non-Uniform Random Variate Generation”, Springer Verlag (1986)

/17/ E. Peschl, „Analytische Geometrie“, B-I-Hochschultaschenbücher Band 15/15a, Bibliographisches Institut Mannheim, (1961)

/18/ „Advanced Finite Element Methods (ASEN-537) - Spring 2003“, Aerospace Engineering Sciences, Univ. of Colorado, Boulder(AFEM), verfügbar im Internet unter www.colorado.edu/engineering/AEROSPACE/CAS/courses.d/AFEM.d

/19/ S. Wiesen, „Simulation von Photonentransport in Plasmen“, Diplomarbeit an der Heinrich-Heine-Universität Düsseldorf (2002)

/20/ G.A. Korn, T.M. Korn, „Mathematical Handbook for Scientists and Engineers“, McGraw-Hill, New York, (1961)

/21/ E.W. Weisstein, “CRC Concise Encyclopaedia of Mathematics”, CRC Press, (1999)

/22/ B. Wilkinson, M. Allen, “Parallel Programming”, 2nd ed., Pearson Education Inc, Upper Saddle River, NJ (2005)

/23/ W.R. Martin, T.-Ch Wan, T.S. Abdel-Rahmann, and T.N. Mudge, The International Journal of Supercomputer Applications, Vol. **1**, No. 3, (1987) p57-74

/24/ D.E. Knuth, “The Art of Computer Programming, Volume 2: Seminumerical Algorithms”, Addison-Wesley, Reading, Massachusetts, (1981)

/25/ W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, “Numerical Recipes in C”, 2nd ed. Cambridge University Press (1999)

/26/ http://www.fz-juelich.de/vislab/software/raps/raps_g.html

/27/ Message Passing Interface Forum, “A Message-Passing Interface Standard”, University of Tennesse, Knoxville (1995), verfügbar im Internet unter http://www.mpi-forum.org

/28/ A.N. Gorban, I.V. Karlin, „Invariant Manifolds for Physical and Chemical Kinetics“, Springer, Heidelberg, (2005)

/29/ R. Keunings, „Rheology Reviews 2004“, D.M. Binding, K. Walters, (Eds.), British Society of Rheology

/30/ A.I. Shestakov, R.H. Cohen, J.A. Crotinger et al., J. Comp. Physics **185,** (2003), p399

/31/ P. Börner, D. Reiter, M. Born, H. Giese, S. Wiesen and M. Baeva, Light Sources 2004, IoP Conf. Series **182,** 407 (2004)

/32/ M. Baeva, M. Born, S. Meier, D. Reiter, H. Schubert, M. Weiß, and S. Wiesen, Proc. 26th Int. Conf. on Phenomena in Ionised Gases (ICPIG XXVI), Greifswald, Germany, **2**, 63 (2003)

A. FORTRAN Source-Code

A.1. MAKE_TETRA_48

```
      SUBROUTINE MAKE_TETRA_48 (INDCO)

      USE PRECISION
      USE PARMMOD
      USE CTETRA
      USE CLGIN

      IMPLICIT NONE

      INTEGER,INTENT(IN) :: INDCO(27)
      INTEGER :: ITET, JTET, IS, JS

      IF (NTET+48 > NTETRA) THEN
        WRITE (6,*) ' ALLOWED NUMBER OF TETRAHEDRONS EXCEEDED '
        WRITE (6,*) ' INCREASE NTETRA '
        CALL EXIT_OWN(1)
      END IF


!  tetrahedron 1
      NTECK(1,NTET+1) = INDCO(1)
      NTECK(2,NTET+1) = INDCO(2)
      NTECK(3,NTET+1) = INDCO(11)
      NTECK(4,NTET+1) = INDCO(14)
      CALL EINFUEGEN (INDCO(1),NTET+1)
      CALL EINFUEGEN (INDCO(2),NTET+1)
      CALL EINFUEGEN (INDCO(11),NTET+1)
      CALL EINFUEGEN (INDCO(14),NTET+1)

      IF (COORD_TEST(INDCO(1),INDCO(2),INDCO(11),INDCO(14))) THEN
        NTBAR(2,NTET+1) = NTET+34
        NTSEITE(2,NTET+1) = 2
        NTBAR(3,NTET+1) = NTET+2
        NTSEITE(3,NTET+1) = 4
        NTBAR(4,NTET+1) = NTET+8
        NTSEITE(4,NTET+1) = 3
      ELSE
        NTBAR(1:4,NTET+1) = -1
        NTSEITE(1:4,NTET+1) = -1
      END IF

!  tetrahedron 2
      NTECK(1,NTET+2) = INDCO(2)
      NTECK(2,NTET+2) = INDCO(3)
      NTECK(3,NTET+2) = INDCO(11)
      NTECK(4,NTET+2) = INDCO(14)
      CALL EINFUEGEN (INDCO(2),NTET+2)
      CALL EINFUEGEN (INDCO(3),NTET+2)
      CALL EINFUEGEN (INDCO(11),NTET+2)
      CALL EINFUEGEN (INDCO(14),NTET+2)

      IF (COORD_TEST(INDCO(2),INDCO(3),INDCO(11),INDCO(14))) THEN
        NTBAR(2,NTET+2) = NTET+33
        NTSEITE(2,NTET+2) = 2
        NTBAR(3,NTET+2) = NTET+3
        NTSEITE(3,NTET+2) = 4
```

```
        NTBAR(4,NTET+2) = NTET+1
        NTSEITE(4,NTET+2) = 3
      ELSE
        NTBAR(1:4,NTET+2) = -1
        NTSEITE(1:4,NTET+2) = -1
      END IF


!  tetrahedron 3
      NTECK(1,NTET+3) = INDCO(3)
      NTECK(2,NTET+3) = INDCO(12)
      NTECK(3,NTET+3) = INDCO(11)
      NTECK(4,NTET+3) = INDCO(14)
      CALL EINFUEGEN (INDCO(3),NTET+3)
      CALL EINFUEGEN (INDCO(12),NTET+3)
      CALL EINFUEGEN (INDCO(11),NTET+3)
      CALL EINFUEGEN (INDCO(14),NTET+3)

      IF (COORD_TEST(INDCO(3),INDCO(12),INDCO(11),INDCO(14))) THEN
        NTBAR(2,NTET+3) = NTET+16
        NTSEITE(2,NTET+3) = 2
        NTBAR(3,NTET+3) = NTET+4
        NTSEITE(3,NTET+3) = 4
        NTBAR(4,NTET+3) = NTET+2
        NTSEITE(4,NTET+3) = 3
      ELSE
        NTBAR(1:4,NTET+3) = -1
        NTSEITE(1:4,NTET+3) = -1
      END IF


!  tetrahedron 4
      NTECK(1,NTET+4) = INDCO(12)
      NTECK(2,NTET+4) = INDCO(21)
      NTECK(3,NTET+4) = INDCO(11)
      NTECK(4,NTET+4) = INDCO(14)
      CALL EINFUEGEN (INDCO(12),NTET+4)
      CALL EINFUEGEN (INDCO(21),NTET+4)
      CALL EINFUEGEN (INDCO(11),NTET+4)
      CALL EINFUEGEN (INDCO(14),NTET+4)

      IF (COORD_TEST(INDCO(12),INDCO(21),INDCO(11),INDCO(14))) THEN
        NTBAR(2,NTET+4) = NTET+15
        NTSEITE(2,NTET+4) = 2
        NTBAR(3,NTET+4) = NTET+5
        NTSEITE(3,NTET+4) = 4
        NTBAR(4,NTET+4) = NTET+3
        NTSEITE(4,NTET+4) = 3
      ELSE
        NTBAR(1:4,NTET+4) = -1
        NTSEITE(1:4,NTET+4) = -1
      END IF


!  tetrahedron 5
      NTECK(1,NTET+5) = INDCO(21)
      NTECK(2,NTET+5) = INDCO(20)
      NTECK(3,NTET+5) = INDCO(11)
      NTECK(4,NTET+5) = INDCO(14)
      CALL EINFUEGEN (INDCO(21),NTET+5)
      CALL EINFUEGEN (INDCO(20),NTET+5)
      CALL EINFUEGEN (INDCO(11),NTET+5)
      CALL EINFUEGEN (INDCO(14),NTET+5)
```

```
      IF (COORD_TEST(INDCO(21),INDCO(20),INDCO(11),INDCO(14))) THEN
        NTBAR(2,NTET+5) = NTET+42
        NTSEITE(2,NTET+5) = 2
        NTBAR(3,NTET+5) = NTET+6
        NTSEITE(3,NTET+5) = 4
        NTBAR(4,NTET+5) = NTET+4
        NTSEITE(4,NTET+5) = 3
      ELSE
        NTBAR(1:4,NTET+5) = -1
        NTSEITE(1:4,NTET+5) = -1
      END IF


!  tetrahedron 6
      NTECK(1,NTET+6) = INDCO(20)
      NTECK(2,NTET+6) = INDCO(19)
      NTECK(3,NTET+6) = INDCO(11)
      NTECK(4,NTET+6) = INDCO(14)
      CALL EINFUEGEN (INDCO(20),NTET+6)
      CALL EINFUEGEN (INDCO(19),NTET+6)
      CALL EINFUEGEN (INDCO(11),NTET+6)
      CALL EINFUEGEN (INDCO(14),NTET+6)

      IF (COORD_TEST(INDCO(20),INDCO(19),INDCO(11),INDCO(14))) THEN
        NTBAR(2,NTET+6) = NTET+41
        NTSEITE(2,NTET+6) = 2
        NTBAR(3,NTET+6) = NTET+7
        NTSEITE(3,NTET+6) = 4
        NTBAR(4,NTET+6) = NTET+5
        NTSEITE(4,NTET+6) = 3
      ELSE
        NTBAR(1:4,NTET+6) = -1
        NTSEITE(1:4,NTET+6) = -1
      END IF


!  tetrahedron 7
      NTECK(1,NTET+7) = INDCO(19)
      NTECK(2,NTET+7) = INDCO(10)
      NTECK(3,NTET+7) = INDCO(11)
      NTECK(4,NTET+7) = INDCO(14)
      CALL EINFUEGEN (INDCO(19),NTET+7)
      CALL EINFUEGEN (INDCO(10),NTET+7)
      CALL EINFUEGEN (INDCO(11),NTET+7)
      CALL EINFUEGEN (INDCO(14),NTET+7)

      IF (COORD_TEST(INDCO(19),INDCO(10),INDCO(11),INDCO(14))) THEN
        NTBAR(2,NTET+7) = NTET+28
        NTSEITE(2,NTET+7) = 2
        NTBAR(3,NTET+7) = NTET+8
        NTSEITE(3,NTET+7) = 4
        NTBAR(4,NTET+7) = NTET+6
        NTSEITE(4,NTET+7) = 3
      ELSE
        NTBAR(1:4,NTET+7) = -1
        NTSEITE(1:4,NTET+7) = -1
      END IF


!  tetrahedron 8
      NTECK(1,NTET+8) = INDCO(10)
```

```
      NTECK(2,NTET+8) = INDCO(1)
      NTECK(3,NTET+8) = INDCO(11)
      NTECK(4,NTET+8) = INDCO(14)
      CALL EINFUEGEN (INDCO(10),NTET+8)
      CALL EINFUEGEN (INDCO(1),NTET+8)
      CALL EINFUEGEN (INDCO(11),NTET+8)
      CALL EINFUEGEN (INDCO(14),NTET+8)

      IF (COORD_TEST(INDCO(10),INDCO(1),INDCO(11),INDCO(14))) THEN
        NTBAR(2,NTET+8) = NTET+27
        NTSEITE(2,NTET+8) = 2
        NTBAR(3,NTET+8) = NTET+1
        NTSEITE(3,NTET+8) = 4
        NTBAR(4,NTET+8) = NTET+7
        NTSEITE(4,NTET+8) = 3
      ELSE
        NTBAR(1:4,NTET+8) = -1
        NTSEITE(1:4,NTET+8) = -1
      END IF


!  tetrahedron 9
      NTECK(1,NTET+9) = INDCO(3)
      NTECK(2,NTET+9) = INDCO(6)
      NTECK(3,NTET+9) = INDCO(15)
      NTECK(4,NTET+9) = INDCO(14)
      CALL EINFUEGEN (INDCO(3),NTET+9)
      CALL EINFUEGEN (INDCO(6),NTET+9)
      CALL EINFUEGEN (INDCO(15),NTET+9)
      CALL EINFUEGEN (INDCO(14),NTET+9)

      IF (COORD_TEST(INDCO(3),INDCO(6),INDCO(15),INDCO(14))) THEN
        NTBAR(2,NTET+9) = NTET+40
        NTSEITE(2,NTET+9) = 2
        NTBAR(3,NTET+9) = NTET+10
        NTSEITE(3,NTET+9) = 4
        NTBAR(4,NTET+9) = NTET+16
        NTSEITE(4,NTET+9) = 3
      ELSE
        NTBAR(1:4,NTET+9) = -1
        NTSEITE(1:4,NTET+9) = -1
      END IF


!  tetrahedron 10
      NTECK(1,NTET+10) = INDCO(6)
      NTECK(2,NTET+10) = INDCO(9)
      NTECK(3,NTET+10) = INDCO(15)
      NTECK(4,NTET+10) = INDCO(14)
      CALL EINFUEGEN (INDCO(6),NTET+10)
      CALL EINFUEGEN (INDCO(9),NTET+10)
      CALL EINFUEGEN (INDCO(15),NTET+10)
      CALL EINFUEGEN (INDCO(14),NTET+10)

      IF (COORD_TEST(INDCO(6),INDCO(9),INDCO(15),INDCO(14))) THEN
        NTBAR(2,NTET+10) = NTET+39
        NTSEITE(2,NTET+10) = 2
        NTBAR(3,NTET+10) = NTET+11
        NTSEITE(3,NTET+10) = 4
        NTBAR(4,NTET+10) = NTET+9
        NTSEITE(4,NTET+10) = 3
      ELSE
        NTBAR(1:4,NTET+10) = -1
```

```
        NTSEITE(1:4,NTET+10) = -1
      END IF


!  tetrahedron 11
      NTECK(1,NTET+11) = INDCO(9)
      NTECK(2,NTET+11) = INDCO(18)
      NTECK(3,NTET+11) = INDCO(15)
      NTECK(4,NTET+11) = INDCO(14)
      CALL EINFUEGEN (INDCO(9),NTET+11)
      CALL EINFUEGEN (INDCO(18),NTET+11)
      CALL EINFUEGEN (INDCO(15),NTET+11)
      CALL EINFUEGEN (INDCO(14),NTET+11)

      IF (COORD_TEST(INDCO(9),INDCO(18),INDCO(15),INDCO(14))) THEN
        NTBAR(2,NTET+11) = NTET+24
        NTSEITE(2,NTET+11) = 2
        NTBAR(3,NTET+11) = NTET+12
        NTSEITE(3,NTET+11) = 4
        NTBAR(4,NTET+11) = NTET+10
        NTSEITE(4,NTET+11) = 3
      ELSE
        NTBAR(1:4,NTET+11) = -1
        NTSEITE(1:4,NTET+11) = -1
      END IF


!  tetrahedron 12
      NTECK(1,NTET+12) = INDCO(18)
      NTECK(2,NTET+12) = INDCO(27)
      NTECK(3,NTET+12) = INDCO(15)
      NTECK(4,NTET+12) = INDCO(14)
      CALL EINFUEGEN (INDCO(18),NTET+12)
      CALL EINFUEGEN (INDCO(27),NTET+12)
      CALL EINFUEGEN (INDCO(15),NTET+12)
      CALL EINFUEGEN (INDCO(14),NTET+12)

      IF (COORD_TEST(INDCO(18),INDCO(27),INDCO(15),INDCO(14))) THEN
        NTBAR(2,NTET+12) = NTET+23
        NTSEITE(2,NTET+12) = 2
        NTBAR(3,NTET+12) = NTET+13
        NTSEITE(3,NTET+12) = 4
        NTBAR(4,NTET+12) = NTET+11
        NTSEITE(4,NTET+12) = 3
      ELSE
        NTBAR(1:4,NTET+12) = -1
        NTSEITE(1:4,NTET+12) = -1
      END IF


!  tetrahedron 13
      NTECK(1,NTET+13) = INDCO(27)
      NTECK(2,NTET+13) = INDCO(24)
      NTECK(3,NTET+13) = INDCO(15)
      NTECK(4,NTET+13) = INDCO(14)
      CALL EINFUEGEN (INDCO(27),NTET+13)
      CALL EINFUEGEN (INDCO(24),NTET+13)
      CALL EINFUEGEN (INDCO(15),NTET+13)
      CALL EINFUEGEN (INDCO(14),NTET+13)

      IF (COORD_TEST(INDCO(27),INDCO(24),INDCO(15),INDCO(14))) THEN
        NTBAR(2,NTET+13) = NTET+44
```

```
        NTSEITE(2,NTET+13) = 2
        NTBAR(3,NTET+13) = NTET+14
        NTSEITE(3,NTET+13) = 4
        NTBAR(4,NTET+13) = NTET+12
        NTSEITE(4,NTET+13) = 3
      ELSE
        NTBAR(1:4,NTET+13) = -1
        NTSEITE(1:4,NTET+13) = -1
      END IF


!  tetrahedron 14
      NTECK(1,NTET+14) = INDCO(24)
      NTECK(2,NTET+14) = INDCO(21)
      NTECK(3,NTET+14) = INDCO(15)
      NTECK(4,NTET+14) = INDCO(14)
      CALL EINFUEGEN (INDCO(24),NTET+14)
      CALL EINFUEGEN (INDCO(21),NTET+14)
      CALL EINFUEGEN (INDCO(15),NTET+14)
      CALL EINFUEGEN (INDCO(14),NTET+14)

      IF (COORD_TEST(INDCO(24),INDCO(21),INDCO(15),INDCO(14))) THEN
        NTBAR(2,NTET+14) = NTET+43
        NTSEITE(2,NTET+14) = 2
        NTBAR(3,NTET+14) = NTET+15
        NTSEITE(3,NTET+14) = 4
        NTBAR(4,NTET+14) = NTET+13
        NTSEITE(4,NTET+14) = 3
      ELSE
        NTBAR(1:4,NTET+14) = -1
        NTSEITE(1:4,NTET+14) = -1
      END IF


!  tetrahedron 15
      NTECK(1,NTET+15) = INDCO(21)
      NTECK(2,NTET+15) = INDCO(12)
      NTECK(3,NTET+15) = INDCO(15)
      NTECK(4,NTET+15) = INDCO(14)
      CALL EINFUEGEN (INDCO(21),NTET+15)
      CALL EINFUEGEN (INDCO(12),NTET+15)
      CALL EINFUEGEN (INDCO(15),NTET+15)
      CALL EINFUEGEN (INDCO(14),NTET+15)

      IF (COORD_TEST(INDCO(21),INDCO(12),INDCO(15),INDCO(14))) THEN
        NTBAR(2,NTET+15) = NTET+4
        NTSEITE(2,NTET+15) = 2
        NTBAR(3,NTET+15) = NTET+16
        NTSEITE(3,NTET+15) = 4
        NTBAR(4,NTET+15) = NTET+14
        NTSEITE(4,NTET+15) = 3
      ELSE
        NTBAR(1:4,NTET+15) = -1
        NTSEITE(1:4,NTET+15) = -1
      END IF


!  tetrahedron 16
      NTECK(1,NTET+16) = INDCO(12)
      NTECK(2,NTET+16) = INDCO(3)
      NTECK(3,NTET+16) = INDCO(15)
      NTECK(4,NTET+16) = INDCO(14)
      CALL EINFUEGEN (INDCO(12),NTET+16)
```

```
      CALL EINFUEGEN (INDCO(3),NTET+16)
      CALL EINFUEGEN (INDCO(15),NTET+16)
      CALL EINFUEGEN (INDCO(14),NTET+16)

      IF (COORD_TEST(INDCO(12),INDCO(3),INDCO(15),INDCO(14))) THEN
        NTBAR(2,NTET+16) = NTET+3
        NTSEITE(2,NTET+16) = 2
        NTBAR(3,NTET+16) = NTET+9
        NTSEITE(3,NTET+16) = 4
        NTBAR(4,NTET+16) = NTET+15
        NTSEITE(4,NTET+16) = 3
      ELSE
        NTBAR(1:4,NTET+16) = -1
        NTSEITE(1:4,NTET+16) = -1
      END IF


!  tetrahedron 17
      NTECK(1,NTET+17) = INDCO(9)
      NTECK(2,NTET+17) = INDCO(8)
      NTECK(3,NTET+17) = INDCO(17)
      NTECK(4,NTET+17) = INDCO(14)
      CALL EINFUEGEN (INDCO(9),NTET+17)
      CALL EINFUEGEN (INDCO(8),NTET+17)
      CALL EINFUEGEN (INDCO(17),NTET+17)
      CALL EINFUEGEN (INDCO(14),NTET+17)

      IF (COORD_TEST(INDCO(9),INDCO(8),INDCO(17),INDCO(14))) THEN
        NTBAR(2,NTET+17) = NTET+38
        NTSEITE(2,NTET+17) = 2
        NTBAR(3,NTET+17) = NTET+18
        NTSEITE(3,NTET+17) = 4
        NTBAR(4,NTET+17) = NTET+24
        NTSEITE(4,NTET+17) = 3
      ELSE
        NTBAR(1:4,NTET+17) = -1
        NTSEITE(1:4,NTET+17) = -1
      END IF


!  tetrahedron 18
      NTECK(1,NTET+18) = INDCO(8)
      NTECK(2,NTET+18) = INDCO(7)
      NTECK(3,NTET+18) = INDCO(17)
      NTECK(4,NTET+18) = INDCO(14)
      CALL EINFUEGEN (INDCO(8),NTET+18)
      CALL EINFUEGEN (INDCO(7),NTET+18)
      CALL EINFUEGEN (INDCO(17),NTET+18)
      CALL EINFUEGEN (INDCO(14),NTET+18)

      IF (COORD_TEST(INDCO(8),INDCO(7),INDCO(17),INDCO(14))) THEN
        NTBAR(2,NTET+18) = NTET+37
        NTSEITE(2,NTET+18) = 2
        NTBAR(3,NTET+18) = NTET+19
        NTSEITE(3,NTET+18) = 4
        NTBAR(4,NTET+18) = NTET+17
        NTSEITE(4,NTET+18) = 3
      ELSE
        NTBAR(1:4,NTET+18) = -1
        NTSEITE(1:4,NTET+18) = -1
      END IF
```

```
!  tetrahedron 19
      NTECK(1,NTET+19) = INDCO(7)
      NTECK(2,NTET+19) = INDCO(16)
      NTECK(3,NTET+19) = INDCO(17)
      NTECK(4,NTET+19) = INDCO(14)
      CALL EINFUEGEN (INDCO(7),NTET+19)
      CALL EINFUEGEN (INDCO(16),NTET+19)
      CALL EINFUEGEN (INDCO(17),NTET+19)
      CALL EINFUEGEN (INDCO(14),NTET+19)

      IF (COORD_TEST(INDCO(7),INDCO(16),INDCO(17),INDCO(14))) THEN
        NTBAR(2,NTET+19) = NTET+32
        NTSEITE(2,NTET+19) = 2
        NTBAR(3,NTET+19) = NTET+20
        NTSEITE(3,NTET+19) = 4
        NTBAR(4,NTET+19) = NTET+18
        NTSEITE(4,NTET+19) = 3
      ELSE
        NTBAR(1:4,NTET+19) = -1
        NTSEITE(1:4,NTET+19) = -1
      END IF


!  tetrahedron 20
      NTECK(1,NTET+20) = INDCO(16)
      NTECK(2,NTET+20) = INDCO(25)
      NTECK(3,NTET+20) = INDCO(17)
      NTECK(4,NTET+20) = INDCO(14)
      CALL EINFUEGEN (INDCO(16),NTET+20)
      CALL EINFUEGEN (INDCO(25),NTET+20)
      CALL EINFUEGEN (INDCO(17),NTET+20)
      CALL EINFUEGEN (INDCO(14),NTET+20)

      IF (COORD_TEST(INDCO(16),INDCO(25),INDCO(17),INDCO(14))) THEN
        NTBAR(2,NTET+20) = NTET+31
        NTSEITE(2,NTET+20) = 2
        NTBAR(3,NTET+20) = NTET+21
        NTSEITE(3,NTET+20) = 4
        NTBAR(4,NTET+20) = NTET+19
        NTSEITE(4,NTET+20) = 3
      ELSE
        NTBAR(1:4,NTET+20) = -1
        NTSEITE(1:4,NTET+20) = -1
      END IF


!  tetrahedron 21
      NTECK(1,NTET+21) = INDCO(25)
      NTECK(2,NTET+21) = INDCO(26)
      NTECK(3,NTET+21) = INDCO(17)
      NTECK(4,NTET+21) = INDCO(14)
      CALL EINFUEGEN (INDCO(25),NTET+21)
      CALL EINFUEGEN (INDCO(26),NTET+21)
      CALL EINFUEGEN (INDCO(17),NTET+21)
      CALL EINFUEGEN (INDCO(14),NTET+21)

      IF (COORD_TEST(INDCO(25),INDCO(26),INDCO(17),INDCO(14))) THEN
        NTBAR(2,NTET+21) = NTET+46
        NTSEITE(2,NTET+21) = 2
        NTBAR(3,NTET+21) = NTET+22
        NTSEITE(3,NTET+21) = 4
        NTBAR(4,NTET+21) = NTET+20
```

```
        NTSEITE(4,NTET+21) = 3
      ELSE
        NTBAR(1:4,NTET+21) = -1
        NTSEITE(1:4,NTET+21) = -1
      END IF


!  tetrahedron 22
      NTECK(1,NTET+22) = INDCO(26)
      NTECK(2,NTET+22) = INDCO(27)
      NTECK(3,NTET+22) = INDCO(17)
      NTECK(4,NTET+22) = INDCO(14)
      CALL EINFUEGEN (INDCO(26),NTET+22)
      CALL EINFUEGEN (INDCO(27),NTET+22)
      CALL EINFUEGEN (INDCO(17),NTET+22)
      CALL EINFUEGEN (INDCO(14),NTET+22)

      IF (COORD_TEST(INDCO(26),INDCO(27),INDCO(17),INDCO(14))) THEN
        NTBAR(2,NTET+22) = NTET+45
        NTSEITE(2,NTET+22) = 2
        NTBAR(3,NTET+22) = NTET+23
        NTSEITE(3,NTET+22) = 4
        NTBAR(4,NTET+22) = NTET+21
        NTSEITE(4,NTET+22) = 3
      ELSE
        NTBAR(1:4,NTET+22) = -1
        NTSEITE(1:4,NTET+22) = -1
      END IF


!  tetrahedron 23
      NTECK(1,NTET+23) = INDCO(27)
      NTECK(2,NTET+23) = INDCO(18)
      NTECK(3,NTET+23) = INDCO(17)
      NTECK(4,NTET+23) = INDCO(14)
      CALL EINFUEGEN (INDCO(27),NTET+23)
      CALL EINFUEGEN (INDCO(18),NTET+23)
      CALL EINFUEGEN (INDCO(17),NTET+23)
      CALL EINFUEGEN (INDCO(14),NTET+23)

      IF (COORD_TEST(INDCO(27),INDCO(18),INDCO(17),INDCO(14))) THEN
        NTBAR(2,NTET+23) = NTET+12
        NTSEITE(2,NTET+23) = 2
        NTBAR(3,NTET+23) = NTET+24
        NTSEITE(3,NTET+23) = 4
        NTBAR(4,NTET+23) = NTET+22
        NTSEITE(4,NTET+23) = 3
      ELSE
        NTBAR(1:4,NTET+23) = -1
        NTSEITE(1:4,NTET+23) = -1
      END IF


!  tetrahedron 24
      NTECK(1,NTET+24) = INDCO(18)
      NTECK(2,NTET+24) = INDCO(9)
      NTECK(3,NTET+24) = INDCO(17)
      NTECK(4,NTET+24) = INDCO(14)
      CALL EINFUEGEN (INDCO(18),NTET+24)
      CALL EINFUEGEN (INDCO(9),NTET+24)
      CALL EINFUEGEN (INDCO(17),NTET+24)
      CALL EINFUEGEN (INDCO(14),NTET+24)
```

```
      IF (COORD_TEST(INDCO(18),INDCO(9),INDCO(17),INDCO(14))) THEN
        NTBAR(2,NTET+24) = NTET+11
        NTSEITE(2,NTET+24) = 2
        NTBAR(3,NTET+24) = NTET+17
        NTSEITE(3,NTET+24) = 4
        NTBAR(4,NTET+24) = NTET+23
        NTSEITE(4,NTET+24) = 3
      ELSE
        NTBAR(1:4,NTET+24) = -1
        NTSEITE(1:4,NTET+24) = -1
      END IF


!  tetrahedron 25
      NTECK(1,NTET+25) = INDCO(7)
      NTECK(2,NTET+25) = INDCO(4)
      NTECK(3,NTET+25) = INDCO(13)
      NTECK(4,NTET+25) = INDCO(14)
      CALL EINFUEGEN (INDCO(7),NTET+25)
      CALL EINFUEGEN (INDCO(4),NTET+25)
      CALL EINFUEGEN (INDCO(13),NTET+25)
      CALL EINFUEGEN (INDCO(14),NTET+25)

      IF (COORD_TEST(INDCO(7),INDCO(4),INDCO(13),INDCO(14))) THEN
        NTBAR(2,NTET+25) = NTET+36
        NTSEITE(2,NTET+25) = 2
        NTBAR(3,NTET+25) = NTET+26
        NTSEITE(3,NTET+25) = 4
        NTBAR(4,NTET+25) = NTET+32
        NTSEITE(4,NTET+25) = 3
      ELSE
        NTBAR(1:4,NTET+25) = -1
        NTSEITE(1:4,NTET+25) = -1
      END IF


!  tetrahedron 26
      NTECK(1,NTET+26) = INDCO(4)
      NTECK(2,NTET+26) = INDCO(1)
      NTECK(3,NTET+26) = INDCO(13)
      NTECK(4,NTET+26) = INDCO(14)
      CALL EINFUEGEN (INDCO(4),NTET+26)
      CALL EINFUEGEN (INDCO(1),NTET+26)
      CALL EINFUEGEN (INDCO(13),NTET+26)
      CALL EINFUEGEN (INDCO(14),NTET+26)

      IF (COORD_TEST(INDCO(4),INDCO(1),INDCO(13),INDCO(14))) THEN
        NTBAR(2,NTET+26) = NTET+35
        NTSEITE(2,NTET+26) = 2
        NTBAR(3,NTET+26) = NTET+27
        NTSEITE(3,NTET+26) = 4
        NTBAR(4,NTET+26) = NTET+25
        NTSEITE(4,NTET+26) = 3
      ELSE
        NTBAR(1:4,NTET+26) = -1
        NTSEITE(1:4,NTET+26) = -1
      END IF


!  tetrahedron 27
      NTECK(1,NTET+27) = INDCO(1)
      NTECK(2,NTET+27) = INDCO(10)
```

```
      NTECK(3,NTET+27) = INDCO(13)
      NTECK(4,NTET+27) = INDCO(14)
      CALL EINFUEGEN (INDCO(1),NTET+27)
      CALL EINFUEGEN (INDCO(10),NTET+27)
      CALL EINFUEGEN (INDCO(13),NTET+27)
      CALL EINFUEGEN (INDCO(14),NTET+27)

      IF (COORD_TEST(INDCO(1),INDCO(10),INDCO(13),INDCO(14))) THEN
        NTBAR(2,NTET+27) = NTET+8
        NTSEITE(2,NTET+27) = 2
        NTBAR(3,NTET+27) = NTET+28
        NTSEITE(3,NTET+27) = 4
        NTBAR(4,NTET+27) = NTET+26
        NTSEITE(4,NTET+27) = 3
      ELSE
        NTBAR(1:4,NTET+27) = -1
        NTSEITE(1:4,NTET+27) = -1
      END IF


!  tetrahedron 28
      NTECK(1,NTET+28) = INDCO(10)
      NTECK(2,NTET+28) = INDCO(19)
      NTECK(3,NTET+28) = INDCO(13)
      NTECK(4,NTET+28) = INDCO(14)
      CALL EINFUEGEN (INDCO(10),NTET+28)
      CALL EINFUEGEN (INDCO(19),NTET+28)
      CALL EINFUEGEN (INDCO(13),NTET+28)
      CALL EINFUEGEN (INDCO(14),NTET+28)

      IF (COORD_TEST(INDCO(10),INDCO(19),INDCO(13),INDCO(14))) THEN
        NTBAR(2,NTET+28) = NTET+7
        NTSEITE(2,NTET+28) = 2
        NTBAR(3,NTET+28) = NTET+29
        NTSEITE(3,NTET+28) = 4
        NTBAR(4,NTET+28) = NTET+27
        NTSEITE(4,NTET+28) = 3
      ELSE
        NTBAR(1:4,NTET+28) = -1
        NTSEITE(1:4,NTET+28) = -1
      END IF


!  tetrahedron 29
      NTECK(1,NTET+29) = INDCO(19)
      NTECK(2,NTET+29) = INDCO(22)
      NTECK(3,NTET+29) = INDCO(13)
      NTECK(4,NTET+29) = INDCO(14)
      CALL EINFUEGEN (INDCO(19),NTET+29)
      CALL EINFUEGEN (INDCO(22),NTET+29)
      CALL EINFUEGEN (INDCO(13),NTET+29)
      CALL EINFUEGEN (INDCO(14),NTET+29)

      IF (COORD_TEST(INDCO(19),INDCO(22),INDCO(13),INDCO(14))) THEN
        NTBAR(2,NTET+29) = NTET+48
        NTSEITE(2,NTET+29) = 2
        NTBAR(3,NTET+29) = NTET+30
        NTSEITE(3,NTET+29) = 4
        NTBAR(4,NTET+29) = NTET+28
        NTSEITE(4,NTET+29) = 3
      ELSE
        NTBAR(1:4,NTET+29) = -1
```

```
        NTSEITE(1:4,NTET+29) = -1
      END IF


!  tetrahedron 30
      NTECK(1,NTET+30) = INDCO(22)
      NTECK(2,NTET+30) = INDCO(25)
      NTECK(3,NTET+30) = INDCO(13)
      NTECK(4,NTET+30) = INDCO(14)
      CALL EINFUEGEN (INDCO(22),NTET+30)
      CALL EINFUEGEN (INDCO(25),NTET+30)
      CALL EINFUEGEN (INDCO(13),NTET+30)
      CALL EINFUEGEN (INDCO(14),NTET+30)

      IF (COORD_TEST(INDCO(22),INDCO(25),INDCO(13),INDCO(14))) THEN
        NTBAR(2,NTET+30) = NTET+47
        NTSEITE(2,NTET+30) = 2
        NTBAR(3,NTET+30) = NTET+31
        NTSEITE(3,NTET+30) = 4
        NTBAR(4,NTET+30) = NTET+29
        NTSEITE(4,NTET+30) = 3
      ELSE
        NTBAR(1:4,NTET+30) = -1
        NTSEITE(1:4,NTET+30) = -1
      END IF


!  tetrahedron 31
      NTECK(1,NTET+31) = INDCO(25)
      NTECK(2,NTET+31) = INDCO(16)
      NTECK(3,NTET+31) = INDCO(13)
      NTECK(4,NTET+31) = INDCO(14)
      CALL EINFUEGEN (INDCO(25),NTET+31)
      CALL EINFUEGEN (INDCO(16),NTET+31)
      CALL EINFUEGEN (INDCO(13),NTET+31)
      CALL EINFUEGEN (INDCO(14),NTET+31)

      IF (COORD_TEST(INDCO(25),INDCO(16),INDCO(13),INDCO(14))) THEN
        NTBAR(2,NTET+31) = NTET+20
        NTSEITE(2,NTET+31) = 2
        NTBAR(3,NTET+31) = NTET+32
        NTSEITE(3,NTET+31) = 4
        NTBAR(4,NTET+31) = NTET+30
        NTSEITE(4,NTET+31) = 3
      ELSE
        NTBAR(1:4,NTET+31) = -1
        NTSEITE(1:4,NTET+31) = -1
      END IF


!  tetrahedron 32
      NTECK(1,NTET+32) = INDCO(16)
      NTECK(2,NTET+32) = INDCO(7)
      NTECK(3,NTET+32) = INDCO(13)
      NTECK(4,NTET+32) = INDCO(14)
      CALL EINFUEGEN (INDCO(16),NTET+32)
      CALL EINFUEGEN (INDCO(7),NTET+32)
      CALL EINFUEGEN (INDCO(13),NTET+32)
      CALL EINFUEGEN (INDCO(14),NTET+32)

      IF (COORD_TEST(INDCO(16),INDCO(7),INDCO(13),INDCO(14))) THEN
        NTBAR(2,NTET+32) = NTET+19
        NTSEITE(2,NTET+32) = 2
```

```
        NTBAR(3,NTET+32) = NTET+25
        NTSEITE(3,NTET+32) = 4
        NTBAR(4,NTET+32) = NTET+31
        NTSEITE(4,NTET+32) = 3
      ELSE
        NTBAR(1:4,NTET+32) = -1
        NTSEITE(1:4,NTET+32) = -1
      END IF


!  tetrahedron 33
      NTECK(1,NTET+33) = INDCO(3)
      NTECK(2,NTET+33) = INDCO(2)
      NTECK(3,NTET+33) = INDCO(5)
      NTECK(4,NTET+33) = INDCO(14)
      CALL EINFUEGEN (INDCO(3),NTET+33)
      CALL EINFUEGEN (INDCO(2),NTET+33)
      CALL EINFUEGEN (INDCO(5),NTET+33)
      CALL EINFUEGEN (INDCO(14),NTET+33)

      IF (COORD_TEST(INDCO(3),INDCO(2),INDCO(5),INDCO(14))) THEN
        NTBAR(2,NTET+33) = NTET+2
        NTSEITE(2,NTET+33) = 2
        NTBAR(3,NTET+33) = NTET+34
        NTSEITE(3,NTET+33) = 4
        NTBAR(4,NTET+33) = NTET+40
        NTSEITE(4,NTET+33) = 3
      ELSE
        NTBAR(1:4,NTET+33) = -1
        NTSEITE(1:4,NTET+33) = -1
      END IF


!  tetrahedron 34
      NTECK(1,NTET+34) = INDCO(2)
      NTECK(2,NTET+34) = INDCO(1)
      NTECK(3,NTET+34) = INDCO(5)
      NTECK(4,NTET+34) = INDCO(14)
      CALL EINFUEGEN (INDCO(2),NTET+34)
      CALL EINFUEGEN (INDCO(1),NTET+34)
      CALL EINFUEGEN (INDCO(5),NTET+34)
      CALL EINFUEGEN (INDCO(14),NTET+34)

      IF (COORD_TEST(INDCO(2),INDCO(1),INDCO(5),INDCO(14))) THEN
        NTBAR(2,NTET+34) = NTET+1
        NTSEITE(2,NTET+34) = 2
        NTBAR(3,NTET+34) = NTET+35
        NTSEITE(3,NTET+34) = 4
        NTBAR(4,NTET+34) = NTET+33
        NTSEITE(4,NTET+34) = 3
      ELSE
        NTBAR(1:4,NTET+34) = -1
        NTSEITE(1:4,NTET+34) = -1
      END IF


!  tetrahedron 35
      NTECK(1,NTET+35) = INDCO(1)
      NTECK(2,NTET+35) = INDCO(4)
      NTECK(3,NTET+35) = INDCO(5)
      NTECK(4,NTET+35) = INDCO(14)
      CALL EINFUEGEN (INDCO(1),NTET+35)
```

```
      CALL EINFUEGEN (INDCO(4),NTET+35)
      CALL EINFUEGEN (INDCO(5),NTET+35)
      CALL EINFUEGEN (INDCO(14),NTET+35)

      IF (COORD_TEST(INDCO(1),INDCO(4),INDCO(5),INDCO(14))) THEN
        NTBAR(2,NTET+35) = NTET+26
        NTSEITE(2,NTET+35) = 2
        NTBAR(3,NTET+35) = NTET+36
        NTSEITE(3,NTET+35) = 4
        NTBAR(4,NTET+35) = NTET+34
        NTSEITE(4,NTET+35) = 3
      ELSE
        NTBAR(1:4,NTET+35) = -1
        NTSEITE(1:4,NTET+35) = -1
      END IF


!  tetrahedron 36
      NTECK(1,NTET+36) = INDCO(4)
      NTECK(2,NTET+36) = INDCO(7)
      NTECK(3,NTET+36) = INDCO(5)
      NTECK(4,NTET+36) = INDCO(14)
      CALL EINFUEGEN (INDCO(4),NTET+36)
      CALL EINFUEGEN (INDCO(7),NTET+36)
      CALL EINFUEGEN (INDCO(5),NTET+36)
      CALL EINFUEGEN (INDCO(14),NTET+36)

      IF (COORD_TEST(INDCO(4),INDCO(7),INDCO(5),INDCO(14))) THEN
        NTBAR(2,NTET+36) = NTET+25
        NTSEITE(2,NTET+36) = 2
        NTBAR(3,NTET+36) = NTET+37
        NTSEITE(3,NTET+36) = 4
        NTBAR(4,NTET+36) = NTET+35
        NTSEITE(4,NTET+36) = 3
      ELSE
        NTBAR(1:4,NTET+36) = -1
        NTSEITE(1:4,NTET+36) = -1
      END IF


!  tetrahedron 37
      NTECK(1,NTET+37) = INDCO(7)
      NTECK(2,NTET+37) = INDCO(8)
      NTECK(3,NTET+37) = INDCO(5)
      NTECK(4,NTET+37) = INDCO(14)
      CALL EINFUEGEN (INDCO(7),NTET+37)
      CALL EINFUEGEN (INDCO(8),NTET+37)
      CALL EINFUEGEN (INDCO(5),NTET+37)
      CALL EINFUEGEN (INDCO(14),NTET+37)

      IF (COORD_TEST(INDCO(7),INDCO(8),INDCO(5),INDCO(14))) THEN
        NTBAR(2,NTET+37) = NTET+18
        NTSEITE(2,NTET+37) = 2
        NTBAR(3,NTET+37) = NTET+38
        NTSEITE(3,NTET+37) = 4
        NTBAR(4,NTET+37) = NTET+36
        NTSEITE(4,NTET+37) = 3
      ELSE
        NTBAR(1:4,NTET+37) = -1
        NTSEITE(1:4,NTET+37) = -1
      END IF
```

```
!  tetrahedron 38
      NTECK(1,NTET+38) = INDCO(8)
      NTECK(2,NTET+38) = INDCO(9)
      NTECK(3,NTET+38) = INDCO(5)
      NTECK(4,NTET+38) = INDCO(14)
      CALL EINFUEGEN (INDCO(8),NTET+38)
      CALL EINFUEGEN (INDCO(9),NTET+38)
      CALL EINFUEGEN (INDCO(5),NTET+38)
      CALL EINFUEGEN (INDCO(14),NTET+38)

      IF (COORD_TEST(INDCO(8),INDCO(9),INDCO(5),INDCO(14))) THEN
        NTBAR(2,NTET+38) = NTET+17
        NTSEITE(2,NTET+38) = 2
        NTBAR(3,NTET+38) = NTET+39
        NTSEITE(3,NTET+38) = 4
        NTBAR(4,NTET+38) = NTET+37
        NTSEITE(4,NTET+38) = 3
      ELSE
        NTBAR(1:4,NTET+38) = -1
        NTSEITE(1:4,NTET+38) = -1
      END IF


!  tetrahedron 39
      NTECK(1,NTET+39) = INDCO(9)
      NTECK(2,NTET+39) = INDCO(6)
      NTECK(3,NTET+39) = INDCO(5)
      NTECK(4,NTET+39) = INDCO(14)
      CALL EINFUEGEN (INDCO(9),NTET+39)
      CALL EINFUEGEN (INDCO(6),NTET+39)
      CALL EINFUEGEN (INDCO(5),NTET+39)
      CALL EINFUEGEN (INDCO(14),NTET+39)

      IF (COORD_TEST(INDCO(9),INDCO(6),INDCO(5),INDCO(14))) THEN
        NTBAR(2,NTET+39) = NTET+10
        NTSEITE(2,NTET+39) = 2
        NTBAR(3,NTET+39) = NTET+40
        NTSEITE(3,NTET+39) = 4
        NTBAR(4,NTET+39) = NTET+38
        NTSEITE(4,NTET+39) = 3
      ELSE
        NTBAR(1:4,NTET+39) = -1
        NTSEITE(1:4,NTET+39) = -1
      END IF


!  tetrahedron 40
      NTECK(1,NTET+40) = INDCO(6)
      NTECK(2,NTET+40) = INDCO(3)
      NTECK(3,NTET+40) = INDCO(5)
      NTECK(4,NTET+40) = INDCO(14)
      CALL EINFUEGEN (INDCO(6),NTET+40)
      CALL EINFUEGEN (INDCO(3),NTET+40)
      CALL EINFUEGEN (INDCO(5),NTET+40)
      CALL EINFUEGEN (INDCO(14),NTET+40)

      IF (COORD_TEST(INDCO(6),INDCO(3),INDCO(5),INDCO(14))) THEN
        NTBAR(2,NTET+40) = NTET+9
        NTSEITE(2,NTET+40) = 2
        NTBAR(3,NTET+40) = NTET+33
        NTSEITE(3,NTET+40) = 4
        NTBAR(4,NTET+40) = NTET+39
```

```
        NTSEITE(4,NTET+40) = 3
      ELSE
        NTBAR(1:4,NTET+40) = -1
        NTSEITE(1:4,NTET+40) = -1
      END IF


!  tetrahedron 41
      NTECK(1,NTET+41) = INDCO(19)
      NTECK(2,NTET+41) = INDCO(20)
      NTECK(3,NTET+41) = INDCO(23)
      NTECK(4,NTET+41) = INDCO(14)
      CALL EINFUEGEN (INDCO(19),NTET+41)
      CALL EINFUEGEN (INDCO(20),NTET+41)
      CALL EINFUEGEN (INDCO(23),NTET+41)
      CALL EINFUEGEN (INDCO(14),NTET+41)

      IF (COORD_TEST(INDCO(19),INDCO(20),INDCO(23),INDCO(14))) THEN
        NTBAR(2,NTET+41) = NTET+6
        NTSEITE(2,NTET+41) = 2
        NTBAR(3,NTET+41) = NTET+42
        NTSEITE(3,NTET+41) = 4
        NTBAR(4,NTET+41) = NTET+48
        NTSEITE(4,NTET+41) = 3
      ELSE
        NTBAR(1:4,NTET+41) = -1
        NTSEITE(1:4,NTET+41) = -1
      END IF


!  tetrahedron 42
      NTECK(1,NTET+42) = INDCO(20)
      NTECK(2,NTET+42) = INDCO(21)
      NTECK(3,NTET+42) = INDCO(23)
      NTECK(4,NTET+42) = INDCO(14)
      CALL EINFUEGEN (INDCO(20),NTET+42)
      CALL EINFUEGEN (INDCO(21),NTET+42)
      CALL EINFUEGEN (INDCO(23),NTET+42)
      CALL EINFUEGEN (INDCO(14),NTET+42)

      IF (COORD_TEST(INDCO(20),INDCO(21),INDCO(23),INDCO(14))) THEN
        NTBAR(2,NTET+42) = NTET+5
        NTSEITE(2,NTET+42) = 2
        NTBAR(3,NTET+42) = NTET+43
        NTSEITE(3,NTET+42) = 4
        NTBAR(4,NTET+42) = NTET+41
        NTSEITE(4,NTET+42) = 3
      ELSE
        NTBAR(1:4,NTET+42) = -1
        NTSEITE(1:4,NTET+42) = -1
      END IF


!  tetrahedron 43
      NTECK(1,NTET+43) = INDCO(21)
      NTECK(2,NTET+43) = INDCO(24)
      NTECK(3,NTET+43) = INDCO(23)
      NTECK(4,NTET+43) = INDCO(14)
      CALL EINFUEGEN (INDCO(21),NTET+43)
      CALL EINFUEGEN (INDCO(24),NTET+43)
      CALL EINFUEGEN (INDCO(23),NTET+43)
      CALL EINFUEGEN (INDCO(14),NTET+43)
```

```
      IF (COORD_TEST(INDCO(21),INDCO(24),INDCO(23),INDCO(14))) THEN
        NTBAR(2,NTET+43) = NTET+14
        NTSEITE(2,NTET+43) = 2
        NTBAR(3,NTET+43) = NTET+44
        NTSEITE(3,NTET+43) = 4
        NTBAR(4,NTET+43) = NTET+42
        NTSEITE(4,NTET+43) = 3
      ELSE
        NTBAR(1:4,NTET+43) = -1
        NTSEITE(1:4,NTET+43) = -1
      END IF


!  tetrahedron 44
      NTECK(1,NTET+44) = INDCO(24)
      NTECK(2,NTET+44) = INDCO(27)
      NTECK(3,NTET+44) = INDCO(23)
      NTECK(4,NTET+44) = INDCO(14)
      CALL EINFUEGEN (INDCO(24),NTET+44)
      CALL EINFUEGEN (INDCO(27),NTET+44)
      CALL EINFUEGEN (INDCO(23),NTET+44)
      CALL EINFUEGEN (INDCO(14),NTET+44)

      IF (COORD_TEST(INDCO(24),INDCO(27),INDCO(23),INDCO(14))) THEN
        NTBAR(2,NTET+44) = NTET+13
        NTSEITE(2,NTET+44) = 2
        NTBAR(3,NTET+44) = NTET+45
        NTSEITE(3,NTET+44) = 4
        NTBAR(4,NTET+44) = NTET+43
        NTSEITE(4,NTET+44) = 3
      ELSE
        NTBAR(1:4,NTET+44) = -1
        NTSEITE(1:4,NTET+44) = -1
      END IF


!  tetrahedron 45
      NTECK(1,NTET+45) = INDCO(27)
      NTECK(2,NTET+45) = INDCO(26)
      NTECK(3,NTET+45) = INDCO(23)
      NTECK(4,NTET+45) = INDCO(14)
      CALL EINFUEGEN (INDCO(27),NTET+45)
      CALL EINFUEGEN (INDCO(26),NTET+45)
      CALL EINFUEGEN (INDCO(23),NTET+45)
      CALL EINFUEGEN (INDCO(14),NTET+45)

      IF (COORD_TEST(INDCO(27),INDCO(26),INDCO(23),INDCO(14))) THEN
        NTBAR(2,NTET+45) = NTET+22
        NTSEITE(2,NTET+45) = 2
        NTBAR(3,NTET+45) = NTET+46
        NTSEITE(3,NTET+45) = 4
        NTBAR(4,NTET+45) = NTET+44
        NTSEITE(4,NTET+45) = 3
      ELSE
        NTBAR(1:4,NTET+45) = -1
        NTSEITE(1:4,NTET+45) = -1
      END IF


!  tetrahedron 46
      NTECK(1,NTET+46) = INDCO(26)
      NTECK(2,NTET+46) = INDCO(25)
```

```
      NTECK(3,NTET+46) = INDCO(23)
      NTECK(4,NTET+46) = INDCO(14)
      CALL EINFUEGEN (INDCO(26),NTET+46)
      CALL EINFUEGEN (INDCO(25),NTET+46)
      CALL EINFUEGEN (INDCO(23),NTET+46)
      CALL EINFUEGEN (INDCO(14),NTET+46)

      IF (COORD_TEST(INDCO(26),INDCO(25),INDCO(23),INDCO(14))) THEN
        NTBAR(2,NTET+46) = NTET+21
        NTSEITE(2,NTET+46) = 2
        NTBAR(3,NTET+46) = NTET+47
        NTSEITE(3,NTET+46) = 4
        NTBAR(4,NTET+46) = NTET+45
        NTSEITE(4,NTET+46) = 3
      ELSE
        NTBAR(1:4,NTET+46) = -1
        NTSEITE(1:4,NTET+46) = -1
      END IF


!  tetrahedron 47
      NTECK(1,NTET+47) = INDCO(25)
      NTECK(2,NTET+47) = INDCO(22)
      NTECK(3,NTET+47) = INDCO(23)
      NTECK(4,NTET+47) = INDCO(14)
      CALL EINFUEGEN (INDCO(25),NTET+47)
      CALL EINFUEGEN (INDCO(22),NTET+47)
      CALL EINFUEGEN (INDCO(23),NTET+47)
      CALL EINFUEGEN (INDCO(14),NTET+47)

      IF (COORD_TEST(INDCO(25),INDCO(22),INDCO(23),INDCO(14))) THEN
        NTBAR(2,NTET+47) = NTET+30
        NTSEITE(2,NTET+47) = 2
        NTBAR(3,NTET+47) = NTET+48
        NTSEITE(3,NTET+47) = 4
        NTBAR(4,NTET+47) = NTET+46
        NTSEITE(4,NTET+47) = 3
      ELSE
        NTBAR(1:4,NTET+47) = -1
        NTSEITE(1:4,NTET+47) = -1
      END IF


!  tetrahedron 48
      NTECK(1,NTET+48) = INDCO(22)
      NTECK(2,NTET+48) = INDCO(19)
      NTECK(3,NTET+48) = INDCO(23)
      NTECK(4,NTET+48) = INDCO(14)
      CALL EINFUEGEN (INDCO(22),NTET+48)
      CALL EINFUEGEN (INDCO(19),NTET+48)
      CALL EINFUEGEN (INDCO(23),NTET+48)
      CALL EINFUEGEN (INDCO(14),NTET+48)

      IF (COORD_TEST(INDCO(22),INDCO(19),INDCO(23),INDCO(14))) THEN
        NTBAR(2,NTET+48) = NTET+29
        NTSEITE(2,NTET+48) = 2
        NTBAR(3,NTET+48) = NTET+41
        NTSEITE(3,NTET+48) = 4
        NTBAR(4,NTET+48) = NTET+47
        NTSEITE(4,NTET+48) = 3
      ELSE
        NTBAR(1:4,NTET+48) = -1
        NTSEITE(1:4,NTET+48) = -1
```

```
      END IF


      DO ITET = NTET+1, NTET+48
        DO IS = 1,4
          JTET = NTBAR(IS,ITET)         ! NUMBER OF NEIGHBORING TETRAHEDRON
          IF (JTET > 0) THEN
            JS = NTSEITE(IS,ITET)
            IF (NTBAR(JS,JTET) < 0) THEN  ! SIDE POINTS TO A COLLAPSED
              NTBAR(IS,ITET) = 0          ! TETRAHEDRON
              NTSEITE(IS,ITET) = 0
            END IF
          END IF
        END DO
      END DO

      NTET = NTET+48

      RETURN

      CONTAINS

      SUBROUTINE EINFUEGEN (IC,ITET)
!  inserts tetrahedron ITET into the tetrahedra list of point IC
        INTEGER, INTENT(IN) :: IC, ITET
        TYPE(TET_ELEM), POINTER :: CUR

        ALLOCATE (CUR)
        CUR%NOTET = ITET
        CUR%NEXT_TET => COORTET(IC)%PTET
        COORTET(IC)%PTET => CUR
        MCLSTR = MCLSTR+1
      END SUBROUTINE EINFUEGEN


      FUNCTION COORD_TEST (I1,I2,I3,I4)
!  tests for collapsed tetrahedron
        INTEGER, INTENT(IN) :: I1,I2,I3,I4
        LOGICAL COORD_TEST
        LOGICAL LTEST

        LTEST= (I1==I2) .OR. (I1==I3) .OR. (I1==I4) .OR.
     .         (I2==I3) .OR. (I2==I4) .OR. (I3==I4)
        COORD_TEST = .NOT. LTEST
        RETURN
      END FUNCTION COORD_TEST

      END
```

A.2. SUCHE_NACHBARN

```
      SUBROUTINE SUCHE_NACHBARN

      USE CTETRA
      IMPLICIT NONE

      TYPE(TET_ELEM), POINTER :: CUR, CUR2
      INTEGER :: ITET,IS,JTET,JS,MINIS,MAXIS,MITIS,MINJS,MAXJS,MITJS,
     .           IP1,IP2,IP3,JP1,JP2,JP3, IC, i, j
      INTEGER :: JP(3),ip(3)
      INTEGER :: ITSIDE(3,4)
      DATA ITSIDE /1,2,3,
     .             1,4,2,
     .             2,4,3,
     .             3,4,1/

      DO ITET=1,NTET      ! FOR ALL TETRAHEDRONS
        DO IS=1,4         ! AND FOR ALL SIDES OF EACH TETRAHEDRON
          IF (NTBAR(IS,ITET) == 0) THEN   ! IF IT HAS NO NEIGHBOR JET
            IP(1)=NTECK(ITSIDE(1,IS),ITET)
            IP(2)=NTECK(ITSIDE(2,IS),ITET)
            IP(3)=NTECK(ITSIDE(3,IS),ITET)

            CUR => COORTET(IP(1))%PTET
            WHLOOP:DO WHILE (ASSOCIATED(CUR))
              JTET = CUR%NOTET
              IF (JTET /= ITET) THEN  ! OMIT TETRAHEDRON ITET
                JSLOOP:DO JS=1,4                      ! CHECK ALL SIDES
                  IF (NTBAR(JS,JTET) == 0) THEN
                    JP(1)=NTECK(ITSIDE(1,JS),JTET)
                    JP(2)=NTECK(ITSIDE(2,JS),JTET)
                    JP(3)=NTECK(ITSIDE(3,JS),JTET)
                     iloop:do i=1,3
                      jloop:do j=i,3
                        if (ip(i) == jp(j)) then
                          ip1=jp(j)
                          jp(j)=jp(i)
                          jp(i)=ip1
                          cycle iloop
                        endif
                      end do jloop
                      cycle jsloop
                      end do iloop
                          NTBAR(IS,ITET) = JTET ! NEIGHBOR FOUND
                          NTSEITE(IS,ITET) = JS
                          NTBAR(JS,JTET) = ITET
                          NTSEITE(JS,JTET) = IS
                          EXIT WHLOOP
                  END IF
                END DO JSLOOP ! JS
              END IF
              CUR => CUR%NEXT_TET
            END DO WHLOOP ! WHILE
          END IF
        END DO
      END DO

      RETURN
      END
```

A.3. TIMER

```
C
C  FULL EIRENE GEOMETRY BLOCK  (GEO3D)
C
C
      SUBROUTINE TIMER (PT)
C
C  THIS SUBROUTINE CALCULATES INTERSECTION TIMES IN THE STANDARD
C  MESH (X- OR RADIAL DIRECTION)
C
C  INPUT:
C       NRCELL = CELL NUMBER FOR WHICH NEXT INTERSECTION
C                TIME IS TO BE CALCULATED (I.E. NOT NECESSARLY THE
C                CELL WHICH CONTAINS THE STARTING POINT X0,Y0,Z0)
C                NRCELL=0 IF PARTICLE OUTSIDE STANDARD MESH
C       NJUMP = 0 MEANS: THIS IS THE FIRST CALL OF TIMER FOR THIS TRACK
C                        IN THIS CASE , NLSRFX MUST BE KNOWN
C           NLSRFX = .TRUE. :PARTICLE ON A SURFACE, IN THIS CASE
C                            THE SUBROUTINE NEEDS 'MRSURF'
C                            MRSURF = NUMBER OF THIS SURFACE
C
C           NLSRFX = .FALSE.:PARTICLE NOT ON A SURFACE
C
C           X0,Y0,Z0 = STARTING POINT OF THIS TRACK
C           VELX,VELY,VELZ = VELOCITY OF PARTICLE
C       NJUMP = 1  X0,Y0,Z0,VELX,VELY,VELZ ARE THE SAME
C                  AS IN THE PREVIOUS CALL
C       NJUMP = 2  ONLY VELX,VELY,VELZ ARE THE SAME
C                  AS IN THE PREVIOUS CALL, I.E. PARTICLE HAS
C                  BEEN MOVED BUT VELOCITY HAS NOT BEEN CHANGED
C                  (TO BE WRITTEN)
C
C  OUTPUT :
C       NJUMP = 1
C       MRSURF = INDEX OF NEXT SURFACE ALONG TRACK
C              = 0 IF NO NEXT SURFACE IS FOUND
C       PT = TIME TO REACH THIS SURFACE
C          = 1.D30 IF NO NEXT SURFACE IS FOUND
C       TIMINT(MRSURF) NE 0 INDICATES FURTHER INTERSECTION TIMES FOUND
C                      IN THIS CALL, WHICH MAY BE USED IN A LATER CALL
C       NINCX = INDEX FOR DIRECTION IN GRID "RSURF": +1 OR -1
C             = 0 IF NO NEXT SURFACE FOUND
C       NRCELL:  NUMBER OF FINAL RADIAL CELL (IE. NOT MODIFIED)
C       IRCELL: NOT NEEDED ON RADIAL SURFACE. SET IRCELL=NRCELL
C
C
      USE PRECISION
      USE PARMMOD
      USE COMUSR
      USE CTSURF
      USE CADGEO
      USE CCONA
      USE CLOGAU
      USE CUPD
      USE CPOLYG
      USE CGRID
      USE CGEOM
      USE CTETRA
      USE COMPRT
      USE CLGIN
      USE CTRIG
```

```
      IMPLICIT NONE
        .
        .
        .
      DATA ITSIDE /1,2,3,
     .             1,4,2,
     .             2,4,3,
     .             3,4,1/
      DATA IRICH / 1, -3,
     .             4,  1,
     .             5,  2,
     .             6,  3 /
      DATA ICALL /0/
      DATA ITFRST /0/
      SAVE
C
        .
        .
        .

      IF (LEVGEO > 5) GOTO 12000
C
C     TETRAHEDRONS
C
      TIMTET = 1.E30
      NTIMT = 0
      IF (NRCELL .NE. 0) THEN
        IF (NJUMP.EQ.0) THEN
          IZELL = NRCELL
          IPOLGO=0
          IF (NLSRFX) THEN
            IPOLGO=IPOLG
          ENDIF
C       ELSEIF (NJUMP.EQ.1) THEN
        ENDIF
        IF (NLTRC) THEN
          WRITE (6,*) ' IZELL,IPOLGO ',IZELL,IPOLGO
          WRITE (6,*) ' X0,Y0,Z0 ',X0,Y0,Z0
          WRITE (6,*) ' VELX,VELY,VELZ ',VELX,VELY,VELZ
        ENDIF

        XX = X0
        YY = Y0
        ZZ = Z0
        TM=0.
        IF (IZELL.EQ.0) THEN
          WRITE (6,*) ' IZELL = 0 IN TIMER '
          CALL EXIT_OWN(1)

        END IF
C
C  CHECK TETRAHEDRON IZELL FOR INTERSECTION WITH TRAJECTORY
C
11020   CONTINUE
        DO J=1,4
          IF (J.NE.IPOLGO) THEN
            SIG1=SIGN(1,IRICH(1,J))
            I1=ABS(IRICH(1,J))
            SIG2=SIGN(1,IRICH(2,J))
            I2=ABS(IRICH(2,J))
            PNORMI=RINCRC(J,IZELL)
            A(1:3,1) = (/ VTETX(I1,IZELL), VTETY(I1,IZELL),
```

```
     .                    VTETZ(I1,IZELL)/) * SIG1 * PNORMI
            A(1:3,2) = (/ VTETX(I2,IZELL), VTETY(I2,IZELL),
     .                    VTETZ(I2,IZELL)/) * SIG2 * PNORMI
            A(1:3,3) = (/ -VELX, -VELY, -VELZ /) * PNORMI
            B(1:3) = (/ XX-XTETRA(NTECK(ITSIDE(1,J),IZELL)),
     .                  YY-YTETRA(NTECK(ITSIDE(1,J),IZELL)),
     .                  ZZ-ZTETRA(NTECK(ITSIDE(1,J),IZELL)) /) * PNORMI
            DET = A(1,1) * A(2,2) * A(3,3)
     .          + A(1,2) * A(2,3) * A(3,1)
     .          + A(1,3) * A(2,1) * A(3,2)
     .          - A(3,1) * A(2,2) * A(1,3)
     .          - A(3,2) * A(2,3) * A(1,1)
     .          - A(3,3) * A(2,1) * A(1,2)
            IF (ABS(DET) < 1.D-5) CYCLE
            AB = A
            AB(:,1) = B
            XMU = ( AB(1,1) * AB(2,2) * AB(3,3)
     .            + AB(1,2) * AB(2,3) * AB(3,1)
     .            + AB(1,3) * AB(2,1) * AB(3,2)
     .            - AB(3,1) * AB(2,2) * AB(1,3)
     .            - AB(3,2) * AB(2,3) * AB(1,1)
     .            - AB(3,3) * AB(2,1) * AB(1,2) ) / DET
            IF (XMU .GE.0.D0 .AND. XMU .LE.1.D0) THEN
              AB = A
              AB(:,2) = B
              XETA = ( AB(1,1) * AB(2,2) * AB(3,3)
     .                 + AB(1,2) * AB(2,3) * AB(3,1)
     .                 + AB(1,3) * AB(2,1) * AB(3,2)
     .                 - AB(3,1) * AB(2,2) * AB(1,3)
     .                 - AB(3,2) * AB(2,3) * AB(1,1)
     .                 - AB(3,3) * AB(2,1) * AB(1,2) ) / DET
              IF ((XETA.GE.0.D0 .AND. XETA.LE.1.D0) .AND.
     .            (XMU+XETA <= 1.D0)) THEN
                AB = A
                AB(:,3) = B
                T = ( AB(1,1) * AB(2,2) * AB(3,3)
     .                + AB(1,2) * AB(2,3) * AB(3,1)
     .                + AB(1,3) * AB(2,1) * AB(3,2)
     .                - AB(3,1) * AB(2,2) * AB(1,3)
     .                - AB(3,2) * AB(2,3) * AB(1,1)
     .                - AB(3,3) * AB(2,1) * AB(1,2) ) / DET
                IF (T .GT. 0.) THEN
                  XX = XX + T * VELX
                  YY = YY + T * VELY
                  ZZ = ZZ + T * VELZ
                  TM = TM + T
                  TIMTET = TM
C  NUMMER DER GESCHNITTENEN SEITE DES ALTEN DREIECKS
                  NTIMT = J
                  IF (NLTRC) THEN
                    WRITE (6,*) 'IZELL,XX,YY,ZZ,J ',IZELL,XX,YY,ZZ,J
                  ENDIF
C  ZELLENNUMMER DES NEUEN DREIECKS
                  IZELLO=IZELL
                  IPOLGOO=IPOLGO
                  IZELL = NTBAR(J,IZELLO)
                  PT = TM
                  IF (IZELL .EQ. 0) GOTO 11050
C  SEITENNUMMER DES NEUEN DREIECKS
                  IPOLGO = NTSEITE(J,IZELLO)
                  IF (NLTRC) THEN
                    WRITE(6,*) 'GEHE IN ZELLE ',IZELL,' TM= ',TM
```

```
                  WRITE(6,*) 'DORT AUF SEITE IPOLGO ',IPOLGO
                ENDIF
                GOTO 11050
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      END DO
C
      IF (NLTRC) WRITE (6,*) ' NO INTERSECTION FOUND IN TETRAHEDRON '
      PT=1.D30
      NINCX=0
C  DURCHSUCHE NACHBARDREIECK
      IF (NLSRFX) THEN
        IZELL=NTBAR(IPOLG,NRCELL)
        IPOLGO=NTSEITE(IPOLG,NRCELL)
        NRCELL=IZELL
        NLSRFX=.FALSE.
        GOTO 11020
      ENDIF
C  KEIN SCHNITTPUNKT GEFUNDEN UND NLSRFX=.FALSE.
      RETURN
C
C
C   THIS IS DONE FOR NJUMP=0 AND NJUMP=1
C
11050   CONTINUE
      NLSRFX=.FALSE.
      NJUMP=1
      MRSURF=NRCELL
      PT=TIMTET
      IPOLGN=NTIMT
      ISTS=ABS(INMTIT(IPOLGN,NRCELL))
      IF (ISTS.EQ.0) THEN
        NINCX=NTBAR(IPOLGN,NRCELL)-NRCELL
      ELSEIF (ISTS.GT.0.AND.
     .          ISTS.LE.NLIM+NSTSI) THEN
C  ON NON DEFAULT SURFACE (ADD. OR STD.) ISTS=INMTI(IPOLGN,NRCELL)
        IF (ILIIN(ISTS) == 0) THEN
          NINCX=NTBAR(IPOLGN,NRCELL)-NRCELL
        ELSE
          NINCX=SIGN(1,INMTIT(IPOLGN,NRCELL))
        END IF
      ELSE
        GOTO 9999
      ENDIF
      IF (NLTRC) WRITE (6,*) ' NRCELL,MRSURF,PT,NINCX,IPOLGN ',
     .                         NRCELL,MRSURF,PT,NINCX,IPOLGN

      RETURN

C
9999  CONTINUE
      WRITE (6,*) 'ERROR IN TIMER, EXIT CALLED AT 9999'
      WRITE (6,*) 'NPANU ', NPANU
      CALL EXIT_OWN(1)
      END
```

A.4. LEARCT

```
      INTEGER FUNCTION LEARCT (X,Y,Z)

      USE PRECISION
      USE COMUSR
      USE CCONA
      USE CTETRA

      IMPLICIT NONE

      INTEGER, PARAMETER :: NCL=30

      REAL(DP), INTENT(IN) :: X,Y,Z
      REAL(DP) :: PC1(3), PC2(3), PC3(3), PC4(3), P(3)
      REAL(DP) :: V1, V2, V3, V4, CAL_VOL
      REAL(DP), SAVE :: XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX,
     .                  DISTX, DISTY, DISTZ,
     .                  EPDX, EPDY, EPDZ
      REAL(DP) :: DELTAX, DELTAY, DELTAZ,
     .            XTRMIN, YTRMIN, ZTRMIN, XTRMAX, YTRMAX, ZTRMAX
      INTEGER :: ITET, IFIRST, I, J, K, IX, IY, IZ, IHEADX1, IHEADX2,
     .           IHEADY1, IHEADY2, IHEADZ1, IHEADZ2
      LOGICAL :: LG(NTET)

      TYPE :: CELL
        INTEGER :: TETNR
        TYPE(CELL),POINTER :: NEXT
      END TYPE CELL

      TYPE :: POIFELD
        TYPE (CELL),POINTER :: P
      END TYPE POIFELD

      TYPE (POIFELD) :: HELPCUR(8)
      TYPE (POIFELD),ALLOCATABLE,SAVE :: HEADS(:,:,:)
      TYPE (CELL),POINTER :: CUR

      DATA IFIRST /0/

      IF (IFIRST.EQ.0) THEN
        IFIRST = 1
!  initialize quadrangular grid
        ALLOCATE(HEADS(NCL,NCL,NCL))
        DO I=1,NCL
          DO J=1,NCL
            DO K=1,NCL
              NULLIFY(HEADS(I,J,K)%P)
            ENDDO
          ENDDO
        ENDDO
!  find box for quadrangular grid
        XMIN=MINVAL(XTETRA(1:NCOOR))
        YMIN=MINVAL(YTETRA(1:NCOOR))
        ZMIN=MINVAL(ZTETRA(1:NCOOR))
        XMAX=MAXVAL(XTETRA(1:NCOOR))
        YMAX=MAXVAL(YTETRA(1:NCOOR))
        ZMAX=MAXVAL(ZTETRA(1:NCOOR))
        XMIN = XMIN *(1.-EPS5)
        XMAX = XMAX *(1.+EPS5)
        YMIN = YMIN *(1.-EPS5)
```

```
        YMAX = YMAX *(1.+EPS5)
        ZMIN = ZMIN *(1.-EPS5)
        ZMAX = ZMAX *(1.+EPS5)
        DISTX=(XMAX-XMIN)/REAL(NCL,KIND=DP)
        DISTY=(YMAX-YMIN)/REAL(NCL,KIND=DP)
        DISTZ=(ZMAX-ZMIN)/REAL(NCL,KIND=DP)
        EPDX=DISTX*EPS5
        EPDY=DISTY*EPS5
        EPDZ=DISTZ*EPS5
        DO I=1,NTET
!  find box for tetrahedron
          IF (VOL(I) < EPS30) CYCLE
          XTRMIN = MIN(XTETRA(NTECK(1,I)),XTETRA(NTECK(2,I)),
     .                 XTETRA(NTECK(3,I)),XTETRA(NTECK(4,I)))
          XTRMAX = MAX(XTETRA(NTECK(1,I)),XTETRA(NTECK(2,I)),
     .                 XTETRA(NTECK(3,I)),XTETRA(NTECK(4,I)))
          YTRMIN = MIN(YTETRA(NTECK(1,I)),YTETRA(NTECK(2,I)),
     .                 YTETRA(NTECK(3,I)),YTETRA(NTECK(4,I)))
          YTRMAX = MAX(YTETRA(NTECK(1,I)),YTETRA(NTECK(2,I)),
     .                 YTETRA(NTECK(3,I)),YTETRA(NTECK(4,I)))
          ZTRMIN = MIN(ZTETRA(NTECK(1,I)),ZTETRA(NTECK(2,I)),
     .                 ZTETRA(NTECK(3,I)),ZTETRA(NTECK(4,I)))
          ZTRMAX = MAX(ZTETRA(NTECK(1,I)),ZTETRA(NTECK(2,I)),
     .                 ZTETRA(NTECK(3,I)),ZTETRA(NTECK(4,I)))
          DELTAX=XTRMIN-XMIN
          IHEADX1=INT(DELTAX/DISTX)+1
          DELTAX=XTRMAX-XMIN
          IHEADX2=MAX(IHEADX1,INT(DELTAX/DISTX)+1)
          DELTAY=YTRMIN-YMIN
          IHEADY1=INT(DELTAY/DISTY)+1
          DELTAY=YTRMAX-YMIN
          IHEADY2=MAX(IHEADY1,INT(DELTAY/DISTY)+1)
          DELTAZ=ZTRMIN-ZMIN
          IHEADZ1=INT(DELTAZ/DISTZ)+1
          DELTAZ=ZTRMAX-ZMIN
          IHEADZ2=MAX(IHEADZ1,INT(DELTAZ/DISTZ)+1)
!  associate tetrahedron with all quadrangular cell it can cut
          DO IX=IHEADX1,IHEADX2
            DO IY=IHEADY1,IHEADY2
              DO IZ=IHEADZ1,IHEADZ2
                ALLOCATE(CUR)
                CUR%TETNR = I
                CUR%NEXT => HEADS(IX,IY,IZ)%P
                HEADS(IX,IY,IZ)%P => CUR
              ENDDO
            ENDDO
          ENDDO
        ENDDO
      ENDIF

      DELTAX=X-XMIN
      IHEADX2 = 0
      IF (ABS(MOD(DELTAX,DISTX)) .LT. EPDX) THEN
        IHEADX2=INT(DELTAX/DISTX)
      ENDIF
      IHEADX1=INT(DELTAX/DISTX)+1

      DELTAY=Y-YMIN
      IHEADY2 = 0
      IF (ABS(MOD(DELTAY,DISTY)) .LT. EPDY) THEN
        IHEADY2=INT(DELTAY/DISTY)
      ENDIF
      IHEADY1=INT(DELTAY/DISTY)+1
```

```
      DELTAZ=Z-ZMIN
      IHEADZ2 = 0
      IF (ABS(MOD(DELTAZ,DISTZ)) .LT. EPDZ) THEN
        IHEADZ2=INT(DELTAZ/DISTZ)
      ENDIF
      IHEADZ1=INT(DELTAZ/DISTZ)+1

      HELPCUR(1)%P => HEADS(IHEADX1,IHEADY1,IHEADZ1)%P
      IF (IHEADX2 .GT. 0) THEN
        HELPCUR(2)%P => HEADS(IHEADX2,IHEADY1,IHEADZ1)%P
      ELSE
        NULLIFY(HELPCUR(2)%P)
      ENDIF
      IF (IHEADY2 .GT. 0) THEN
        HELPCUR(3)%P => HEADS(IHEADX1,IHEADY2,IHEADZ1)%P
      ELSE
        NULLIFY(HELPCUR(3)%P)
      ENDIF
      IF ((IHEADX2 .GT. 0) .AND. (IHEADY2 .GT. 0)) THEN
        HELPCUR(4)%P => HEADS(IHEADX2,IHEADY2,IHEADZ1)%P
      ELSE
        NULLIFY(HELPCUR(4)%P)
      ENDIF

      IF (IHEADZ2 .GT. 0) THEN
        HELPCUR(5)%P => HEADS(IHEADX1,IHEADY1,IHEADZ2)%P
        IF (IHEADX2 .GT. 0) THEN
          HELPCUR(6)%P => HEADS(IHEADX2,IHEADY1,IHEADZ2)%P
        ELSE
          NULLIFY(HELPCUR(6)%P)
        ENDIF
        IF (IHEADY2 .GT. 0) THEN
          HELPCUR(7)%P => HEADS(IHEADX1,IHEADY2,IHEADZ2)%P
        ELSE
          NULLIFY(HELPCUR(7)%P)
        ENDIF
        IF ((IHEADX2 .GT. 0) .AND. (IHEADY2 .GT. 0)) THEN
          HELPCUR(8)%P => HEADS(IHEADX2,IHEADY2,IHEADZ2)%P
        ELSE
          NULLIFY(HELPCUR(8)%P)
        ENDIF
      END IF

      P(1:3) = (/ X, Y, Z /)
      LG = .FALSE.
      DO J=1,8
        DO WHILE (ASSOCIATED(HELPCUR(J)%P))
          ITET = HELPCUR(J)%P%TETNR
C  CELL I ALREADY TESTED BEFORE ?
         IF (.NOT.LG(ITET)) THEN
            PC1(1:3)= (/ XTETRA(NTECK(1,ITET)), YTETRA(NTECK(1,ITET)),
     .                   ZTETRA(NTECK(1,ITET)) /)
            PC2(1:3)= (/ XTETRA(NTECK(2,ITET)), YTETRA(NTECK(2,ITET)),
     .                   ZTETRA(NTECK(2,ITET)) /)
            PC3(1:3)= (/ XTETRA(NTECK(3,ITET)), YTETRA(NTECK(3,ITET)),
     .                   ZTETRA(NTECK(3,ITET)) /)
            PC4(1:3)= (/ XTETRA(NTECK(4,ITET)), YTETRA(NTECK(4,ITET)),
     .                   ZTETRA(NTECK(4,ITET)) /)

            IF ((MIN(PC1(1),PC2(1),PC3(1),PC4(1)) <= X) .AND.
     .          (MAX(PC1(1),PC2(1),PC3(1),PC4(1)) >= X) .AND.
```

```
.          (MIN(PC1(2),PC2(2),PC3(2),PC4(2)) <= Y) .AND.
.          (MAX(PC1(2),PC2(2),PC3(2),PC4(2)) >= Y) .AND.
.          (MIN(PC1(3),PC2(3),PC3(3),PC4(3)) <= Z) .AND.
.          (MAX(PC1(3),PC2(3),PC3(3),PC4(3)) >= Z)) THEN

         V1 = CAL_VOL (PC1,PC2,PC3,P)
         V2 = CAL_VOL (PC3,PC2,PC4,P)
         V3 = CAL_VOL (PC1,PC3,PC4,P)
         V4 = CAL_VOL (PC1,PC4,PC2,P)

         IF ((ABS(V1+V2+V3+V4-VOL(ITET)) < 1.D-3*VOL(ITET)) .AND.
.            (MIN(V1,V2,V3,V4) >= -EPS5*VOL(ITET))) THEN
           LEARCT=ITET
           RETURN
         END IF
       END IF
     END IF
     HELPCUR(J)%P => HELPCUR(J)%P%NEXT
   END DO
 END DO

 WRITE (6,*) ' POINT ',X,Y,Z
 WRITE (6,*) ' OUTSIDE OF ALL TETRAHEDRONS '
 LEARCT=0
 RETURN
 END
```

A.5. TRIQUAINT

```
      subroutine triquaint (vector, r, s, t)

!  calculate shape function for triquadratic brick

      USE PRECISION
      implicit none
      real(dp), intent(out) :: vector(27)
      real(dp), intent(in) :: r, s, t
      real(dp) :: halb, viertel, achtel, emr, ems, emt, epr, eps, ept,
     .            emrq, emsq, emtq

      halb = 0.5_dp
      viertel = 0.25_dp
      achtel = 0.125_dp

      emr = 1._dp - r
      ems = 1._dp - s
      emt = 1._dp - t

      epr = 1._dp + r
      eps = 1._dp + s
      ept = 1._dp + t

      emrq = 1._dp - r*r
      emsq = 1._dp - s*s
      emtq = 1._dp - t*t

      vector(1) = -achtel * r * s * t * emr * ems * emt
      vector(2) = viertel * s * t * emrq * ems * emt
      vector(3) = achtel * r * s * t * epr * ems * emt
      vector(4) = viertel * r * t * emr * emsq * emt
      vector(5) = -halb * t * emrq * emsq * emt
      vector(6) = -viertel * r * t * epr * emsq * emt
      vector(7) = achtel * r * s * t * emr * eps * emt
      vector(8) = -viertel * s * t * emrq * eps * emt
      vector(9) = -achtel * r * s * t * epr * eps * emt

      vector(10) = viertel * r * s * emr * ems * emtq
      vector(11) = -halb * s * emrq * ems * emtq
      vector(12) = -viertel * r * s * epr * ems * emtq
      vector(13) = -halb * r * emr * emsq * emtq
      vector(14) = emrq * emsq * emtq
      vector(15) = halb * r * epr * emsq * emtq
      vector(16) = -viertel * r * s * emr * eps * emtq
      vector(17) = halb * s * emrq * eps * emtq
      vector(18) = viertel * r * s * epr * eps * emtq

      vector(19) = achtel * r * s * t * emr * ems * ept
      vector(20) = -viertel * s * t * emrq * ems * ept
      vector(21) = -achtel * r * s * t * epr * ems * ept
      vector(22) = -viertel * r * t * emr * emsq * ept
      vector(23) = halb * t * emrq * emsq * ept
      vector(24) = viertel * r * t * epr * emsq * ept
      vector(25) = -achtel * r * s * t * emr * eps * ept
      vector(26) = viertel * s * t * emrq * eps * ept
      vector(27) = achtel * r * s * t * epr * eps * ept

      return
      end subroutine triquaint
```

A.6. DFTRIQUA

```
      subroutine dftriqua (vector, r, s, t)

!  calculate first partial derivatives

      USE PRECISION

      implicit none
      real(dp), intent(out) :: vector(3,27)
      real(dp), intent(in) :: r, s, t
      real(dp) :: halb, viertel, achtel, emr, ems, emt, epr, eps, ept,
     .            emrq, emsq, emtq, em2r, em2s, em2t, ep2r, ep2s, ep2t

      halb = 0.5_dp
      viertel = 0.25_dp
      achtel = 0.125_dp

      emr = 1._dp - r
      ems = 1._dp - s
      emt = 1._dp - t

      em2r = 1._dp - 2._dp * r
      em2s = 1._dp - 2._dp * s
      em2t = 1._dp - 2._dp * t

      epr = 1._dp + r
      eps = 1._dp + s
      ept = 1._dp + t

      ep2r = 1._dp + 2._dp * r
      ep2s = 1._dp + 2._dp * s
      ep2t = 1._dp + 2._dp * t

      emrq = 1._dp - r*r
      emsq = 1._dp - s*s
      emtq = 1._dp - t*t

      vector(1,1) = -achtel * s * t * em2r * ems * emt
      vector(2,1) = -achtel * r * t * emr * em2s * emt
      vector(3,1) = -achtel * r * s * emr * ems * em2t

      vector(1,2) = viertel * s * t * (-2._dp) * r * ems * emt
      vector(2,2) = viertel * t * emrq * em2s * emt
      vector(3,2) = viertel * s * emrq * ems * em2t

      vector(1,3) = achtel * s * t * ep2r * ems * emt
      vector(2,3) = achtel * r * t * epr * em2s * emt
      vector(3,3) = achtel * r * s * epr * ems * em2t

      vector(1,4) = viertel * t * em2r * emsq * emt
      vector(2,4) = viertel * r * t * emr * (-2._dp) * s * emt
      vector(3,4) = viertel * r * emr * emsq * em2t

      vector(1,5) = -halb * t * (-2._dp) * r * emsq * emt
      vector(2,5) = -halb * t * emrq * (-2._dp) * s * emt
      vector(3,5) = -halb * emrq * emsq * em2t

      vector(1,6) = -viertel * t * ep2r * emsq * emt
      vector(2,6) = -viertel * r * t * epr * (-2._dp) * s * emt
      vector(3,6) = -viertel * r * epr * emsq * em2t
```

```
vector(1,7) = achtel * s * t * em2r * eps * emt
vector(2,7) = achtel * r * t * emr * ep2s * emt
vector(3,7) = achtel * r * s * emr * eps * em2t

vector(1,8) = -viertel * s * t * (-2._dp) * r * eps * emt
vector(2,8) = -viertel * t * emrq * ep2s * emt
vector(3,8) = -viertel * s * emrq * eps * em2t

vector(1,9) = -achtel * s * t * ep2r * eps * emt
vector(2,9) = -achtel * r * t * epr * ep2s * emt
vector(3,9) = -achtel * r * s * epr * eps * em2t


vector(1,10) = viertel * s * em2r * ems * emtq
vector(2,10) = viertel * r * emr * em2s * emtq
vector(3,10) = viertel * r * s * emr * ems * (-2._dp) * t

vector(1,11) = -halb * s * (-2._dp) * r * ems * emtq
vector(2,11) = -halb * emrq * em2s * emtq
vector(3,11) = -halb * s * emrq * ems * (-2._dp) * t

vector(1,12) = -viertel * s * ep2r * ems * emtq
vector(2,12) = -viertel * r * epr * em2s * emtq
vector(3,12) = -viertel * r * s * epr * ems * (-2._dp) * t

vector(1,13) = -halb * em2r * emsq * emtq
vector(2,13) = -halb * r * emr * (-2._dp) * s * emtq
vector(3,13) = -halb * r * emr * emsq * (-2._dp) * t

vector(1,14) = (-2._dp) * r * emsq * emtq
vector(2,14) = emrq * (-2._dp) * s * emtq
vector(3,14) = emrq * emsq * (-2._dp) * t

vector(1,15) = halb * ep2r * emsq * emtq
vector(2,15) = halb * r * epr * (-2._dp) * s * emtq
vector(3,15) = halb * r * epr * emsq * (-2._dp) * t

vector(1,16) = -viertel * s * em2r * eps * emtq
vector(2,16) = -viertel * r * emr * ep2s * emtq
vector(3,16) = -viertel * r * s * emr * eps * (-2._dp) * t

vector(1,17) = halb * s * (-2._dp) * r * eps * emtq
vector(2,17) = halb * emrq * ep2s * emtq
vector(3,17) = halb * s * emrq * eps * (-2._dp) * t

vector(1,18) = viertel * s * ep2r * eps * emtq
vector(2,18) = viertel * r * epr * ep2s * emtq
vector(3,18) = viertel * r * s * epr * eps * (-2._dp) * t


vector(1,19) = achtel * s * t * em2r * ems * ept
vector(2,19) = achtel * r * t * emr * em2s * ept
vector(3,19) = achtel * r * s * emr * ems * ep2t

vector(1,20) = -viertel * s * t * (-2._dp) * r * ems * ept
vector(2,20) = -viertel * t * emrq * em2s * ept
vector(3,20) = -viertel * s * emrq * ems * ep2t

vector(1,21) = -achtel * s * t * ep2r * ems * ept
vector(2,21) = -achtel * r * t * epr * em2s * ept
vector(3,21) = -achtel * r * s * epr * ems * ep2t
```

```
vector(1,22) = -viertel * t * em2r * emsq * ept
vector(2,22) = -viertel * r * t * emr * (-2._dp) * s * ept
vector(3,22) = -viertel * r * emr * emsq * ep2t

vector(1,23) = halb * t * (-2._dp) * r * emsq * ept
vector(2,23) = halb * t * emrq * (-2._dp) * s * ept
vector(3,23) = halb * emrq * emsq * ep2t

vector(1,24) = viertel * t * ep2r * emsq * ept
vector(2,24) = viertel * r * t * epr * (-2._dp) * s * ept
vector(3,24) = viertel * r * epr * emsq * ep2t

vector(1,25) = -achtel * s * t * em2r * eps * ept
vector(2,25) = -achtel * r * t * emr * ep2s * ept
vector(3,25) = -achtel * r * s * emr * eps * ep2t

vector(1,26) = viertel * s * t * (-2._dp) * r * eps * ept
vector(2,26) = viertel * t * emrq * ep2s * ept
vector(3,26) = viertel * s * emrq * eps * ep2t

vector(1,27) = achtel * s * t * ep2r * eps * ept
vector(2,27) = achtel * r * t * epr * ep2s * ept
vector(3,27) = achtel * r * s * epr * eps * ep2t

return
end subroutine dftriqua
```

A.7. RPSCUT

```
      subroutine rpscut (aorig,values)

      use precision
      use parmmod
      use ctetra
      use ctrig
      use cgeom
      use ccona
      use cplot
      use module_avltree

      implicit none

      REAL(DP), INTENT(IN) :: AORIG(*)
      REAL(DP), INTENT(OUT) :: VALUES(*)

      integer, allocatable, save :: tetra_kanten(:,:), kanten(:,:),
     .             iedge(:), icono(:), itetno(:)
      integer, save :: kanten_nummer(4,4) = reshape(
     .    (/ -1,  1,  2,  3,
     .        1, -1,  4,  5,
     .        2,  4, -1,  6,
     .        3,  5,  6, -1 /), (/ 4, 4 /) )
      integer, save :: angrenzende_seiten(2,6) = reshape (
     .     (/ 1, 2,
     .        1, 4,
     .        2, 4,
     .        1, 3,
     .        2, 3,
     .        3, 4 /), (/ 2, 6 /) )
      integer, save :: nkanten, ifirst=0
      real(dp) :: timi, timen, second_own
      real(dp), allocatable, save :: spar(:)
      integer :: i, itri

      if (ifirst == 0) then
        ifirst = 1
        allocate (tetra_kanten(6,ntet))
        allocate (kanten(2,3*ntet))
        tetra_kanten=0
        kanten=0
        nkanten=0
        timi = second_own()
        call suche_kanten
        timen = second_own()
        write (0,*) ' cpu time spend in suche_kanten2 ',timen-timi,' sec'

        call schneide_kanten
        call berechne_koordinaten
        call bilde_dreiecke

        deallocate (tetra_kanten)
        deallocate (kanten)
        deallocate (iedge)
        deallocate (icono)
        deallocate (spar)
      end if
```

```
      do i=1,ntrii
        values(i) = aorig(itetno(i))
      end do

      return


      contains

      subroutine suche_kanten

      implicit none
      integer :: itet, j, nxt_side, akt_tet, akt_edge, isi, ik
      integer :: ic, i, nump1, nump2, noedge, jc, no_side, nxt_tet

      integer :: kpunkte(2,6) = reshape (
     .           (/ 1,2,  1,3,  1,4,  2,3,  2,4,  3,4 /), (/ 2, 6 /) )

      nkanten = 0
! für jeden Tetraeder
      do itet=1,ntet

! durchsuche die Kanten des Tetraeders
        do ik=1,6

! die Kante ist schon bei einem anderen Tetraeder gefunden worden
          if (tetra_kanten(ik,itet) /= 0) cycle

! die Kante ist neu
          nump1 = kpunkte(1,ik)
          nump2 = kpunkte(2,ik)
          ic = nteck(nump1,itet)
          jc = nteck(nump2,itet)

          nkanten = nkanten+1
          kanten(1,nkanten) = ic
        kanten(2,nkanten) = jc
        tetra_kanten(ik,itet) = nkanten

! durchlaufe alle Nachbartetraeder und markiere die gemeinsame Kante

          noedge = ik
        akt_tet = itet
        akt_edge = noedge
          no_side = angrenzende_seiten(1,akt_edge)
        isi = 0
        do
          nxt_tet = ntbar(no_side,akt_tet)
          if (nxt_tet == 0) then
            isi = isi+1
! an beiden Seiten bis zum Rand gelaufen
          if (isi > 1) exit
              no_side = angrenzende_seiten(2,akt_edge)
            nxt_tet = ntbar(no_side,itet)
            if (nxt_tet == 0) exit
            akt_tet = itet
          end if
          nxt_side = ntseite(no_side,akt_tet)
          if (nxt_tet == itet) exit
! bestimme die Kantennummer auf dem neuen Tetraeder
          do j=1,4
            if (nteck(j,nxt_tet) == ic) nump1 = j
            if (nteck(j,nxt_tet) == jc) nump2 = j
```

```
      end do
      noedge = kanten_nummer(nump1,nump2)
      tetra_kanten(noedge,nxt_tet) = nkanten
      no_side = angrenzende_seiten(1,noedge) +
.               angrenzende_seiten(2,noedge) - nxt_side
      akt_tet = nxt_tet
    end do

    end do  ! ik

  end do  ! itet
  return
  end subroutine suche_kanten


  subroutine schneide_kanten
  implicit none
  integer :: ik, k1, k2, ic
  real(dp) :: p1(3), p2(3), v(3), a(3)
  real(dp) :: d, xnen, t, x, y, z, dst
  logical :: inserted

  type(TAVLTree), pointer :: baum

  allocate(spar(nkanten))
  allocate(iedge(nkanten))
  allocate(icono(nkanten))

  spar = 1.E30_dp
  iedge = 0
  icono = 0
  nrknot = 0
  baum => NewTree()

  a(1:3) = cutplane(2:4)
  d = cutplane(1)

  do ik=1,nkanten

    k1 = kanten(1,ik)
    k2 = kanten(2,ik)

    p1(1) = xtetra(k1)
    p1(2) = ytetra(k1)
    p1(3) = ztetra(k1)

    p2(1) = xtetra(k2)
    p2(2) = ytetra(k2)
    p2(3) = ztetra(k2)

    v = p2 - p1

    xnen = sum(a*v)
    if (abs(xnen) < eps10) cycle

    t = -(d + sum(a*p1)) / xnen

    if((t > -eps6) .and. (t < 1._dp+eps6)) then

      x = xtetra(k1) + t*(xtetra(k2)-xtetra(k1))
      y = ytetra(k1) + t*(ytetra(k2)-ytetra(k1))
      z = ztetra(k1) + t*(ztetra(k2)-ztetra(k1))
```

```
          dst = sqrt( (xtetra(k2)-xtetra(k1))**2 +
     .                (ytetra(k2)-ytetra(k1))**2 +
     .                (ztetra(k2)-ztetra(k1))**2 )

          ic = nrknot + 1
          inserted=.false.
          call insert (baum, x, y, z, dst, ic, inserted)

          if (inserted) then
            nrknot = nrknot + 1
            iedge(nrknot) = ik
          end if

          spar(ik) = t
          icono(ik) = ic
        end if

      end do

      call DestroyTree(baum)

      end subroutine schneide_kanten


      subroutine berechne_koordinaten

      implicit none
      real(dp) :: a(3), b(3), c(3), p(3), orig(3)
      real(dp) :: am(2,2), amm1(2,2), rhs(2), x(3)
      real(dp) :: spat, bnorm, cnorm, detam
      integer :: i, ik, k1, k2

! a ist die richtung der normalen zur schnittebene
      a(1:3) = cutplane(2:4)

! berechne vektor b, b senkrecht zu a
! d.h.  a1*b1 + a2*b2 + a3*b3 = 0

      if (abs(a(1)) > eps10) then
        b(1) = -(a(2)+a(3))/a(1)
        b(2) = 1._dp
        b(3) = 1._dp
      else if (abs(a(2)) > eps10) then
        b(1) = 1._dp
        b(2) = -(a(1)+a(3))/a(2)
        b(3) = 1._dp
      else if (abs(a(3)) > eps10) then
        b(1) = 1._dp
        b(2) = 1._dp
        b(3) = -(a(1)+a(2))/a(3)
      else
        write (6,*) ' Problem in berechne_koordinaten '
        write (6,*) ' die Koeffizienten der Schnittebene sind 0 '
        write (6,*) cutplane
        call exit (1)
      end if

! berechne vektor c = a x b ==> c ist senkrecht zu a und b

      c(1) = a(2)*b(3) - b(2)*a(3)
      c(2) = a(3)*b(1) - b(3)*a(1)
      c(3) = a(1)*b(2) - b(1)*a(2)
```

```
! berechne spatprodukt, (a x b)c > 0 ==> rechtsystem, sonst c=-c

      spat =  a(1)*b(2)*b(3) + a(2)*b(3)*c(1) + a(3)*b(1)*c(2)
     .      - c(1)*b(2)*a(3) - c(2)*b(3)*a(1) - c(3)*b(1)*a(2)
      if (spat < 0._dp) c = -c

! normiere b und c
      bnorm = sqrt(sum(b*b))
      cnorm = sqrt(sum(c*c))

      b = b / bnorm
      c = c / cnorm

! b und c sind eine orthonormalbasis der schnittebene


      nknot = nrknot
      nknots = nknot
      ntri = ntet / 2
      ntris = ntri

      call dealloc_ctrig
      call alloc_ctrig

      ik = iedge(1)
      k1 = kanten(1,ik)
      k2 = kanten(2,ik)
      orig(1) = xtetra(k1) + spar(ik)*(xtetra(k2)-xtetra(k1))
      orig(2) = ytetra(k1) + spar(ik)*(ytetra(k2)-ytetra(k1))
      orig(3) = ztetra(k1) + spar(ik)*(ztetra(k2)-ztetra(k1))

      am(1,1) = sum(b*b)
      am(1,2) = sum(b*c)
      am(2,1) = sum(b*c)
      am(2,2) = sum(c*c)

      detam = am(1,1)*am(2,2) - am(1,2)*am(2,1)

      if (abs(detam) < eps10) then
        write (6,*) ' problem in berechne_koordinaten '
        write (6,*) ' gls zur koordinatentransformation nicht loesbar'
        call exit (1)
      end if

      amm1(1,1) = am(2,2) / detam
      amm1(1,2) = -am(1,2) / detam
      amm1(2,1) = -am(2,1) / detam
      amm1(2,2) = am(1,1) / detam

      do i=1,nrknot

        ik = iedge(i)
        k1 = kanten(1,ik)
        k2 = kanten(2,ik)

        x(1) = xtetra(k1) + spar(ik)*(xtetra(k2)-xtetra(k1))
        x(2) = ytetra(k1) + spar(ik)*(ytetra(k2)-ytetra(k1))
        x(3) = ztetra(k1) + spar(ik)*(ztetra(k2)-ztetra(k1))

        rhs(1) = sum(b*(x-orig))
        rhs(2) = sum(c*(x-orig))
```

```
        xtrian(i) = amm1(1,1)*rhs(1) + amm1(1,2)*rhs(2)
        ytrian(i) = amm1(2,1)*rhs(1) + amm1(2,2)*rhs(2)
      end do

      end subroutine berechne_koordinaten



      subroutine sort_ueberpruefen(ipunkt)

      implicit none

      integer, intent(inout), dimension(4) :: ipunkt
      real(dp), dimension(4,2)             :: punkt
      real(dp), dimension(2,2)             :: a
      real(dp), dimension(2)               :: b, t, cen
      real(dp)                             :: x1, x2
      real(dp)                             :: d1, d2, d3, d4, d, deta
      logical                              :: lsw1, lsw2
      integer                              :: ih, i

      punkt(1:4,1) = (/ (xtrian(ipunkt(i)), i=1,4) /)
      punkt(1:4,2) = (/ (ytrian(ipunkt(i)), i=1,4) /)

      cen(1)=sum(punkt(1:4,1))*0.25_dp
      cen(2)=sum(punkt(1:4,2))*0.25_dp

      d1 = sqrt(sum((punkt(1,:)-cen)**2))
      d2 = sqrt(sum((punkt(2,:)-cen)**2))
      d3 = sqrt(sum((punkt(3,:)-cen)**2))
      d4 = sqrt(sum((punkt(4,:)-cen)**2))
      d = 1._dp/max(min(d1,d2,d3,d4),eps10)

      lsw1=.true.
      lsw2=.true.
      do while (lsw1 .or. lsw2)

!  test p1-p2 and p3-p4
        a(1,1) = punkt(2,1) - punkt(1,1)
        a(2,1) = punkt(2,2) - punkt(1,2)
        a(1,2) = punkt(3,1) - punkt(4,1)
        a(2,2) = punkt(3,2) - punkt(4,2)

        b(1) = punkt(3,1) - punkt(1,1)
        b(2) = punkt(3,2) - punkt(1,2)

        a = a*d
        b = b*d

        deta = a(1,1)*a(2,2) - a(2,1)*a(1,2)
        if (abs(deta) < eps10) then
! p1-p2 und p3-p4 sind parallel
        lsw1 = .false.
        else
        x1 = (b(1)*a(2,2) - b(2)*a(1,2))/deta
        x2 = (b(2)*a(1,1) - b(1)*a(2,1))/deta
          if ((x1 >= 0._dp) .and. (x1 <= 1._dp) .and.
     .        (x2 >= 0._dp) .and. (x2 <= 1._dp)) then
!  intersection found, switch points 2 and 3
            t(1:2) = punkt(2,1:2)
            punkt(2,1:2) = punkt(3,1:2)
            punkt(3,1:2) = t(1:2)
          ih = ipunkt(2)
```

```
            ipunkt(2) = ipunkt(3)
            ipunkt(3) = ih
            lsw1 = .true.
          else
            lsw1 = .false.
          end if
        end if

!  test p1-p4 and p2-p3
        a(1,1) = punkt(4,1) - punkt(1,1)
        a(2,1) = punkt(4,2) - punkt(1,2)
        a(1,2) = punkt(2,1) - punkt(3,1)
        a(2,2) = punkt(2,2) - punkt(3,2)

        b(1) = punkt(2,1) - punkt(1,1)
        b(2) = punkt(2,2) - punkt(1,2)

        a = a*d
        b = b*d

        deta = a(1,1)*a(2,2) - a(2,1)*a(1,2)
        if (abs(deta) < eps10) then
! p1-p4 und p2-p3 sind parallel
        lsw2 = .false.
        else
        x1 = (b(1)*a(2,2) - b(2)*a(1,2))/deta
        x2 = (b(2)*a(1,1) - b(1)*a(2,1))/deta

          if ((x1 >= 0._dp) .and. (x1 <= 1._dp) .and.
     .        (x2 >= 0._dp) .and. (x2 <= 1._dp)) then
!  intersection found, switch points 3 and 4
            t(1:2) = punkt(3,1:2)
            punkt(3,1:2) = punkt(4,1:2)
            punkt(4,1:2) = t(1:2)
            ih = ipunkt(3)
            ipunkt(3) = ipunkt(4)
            ipunkt(4) = ih
            lsw2 = .true.
          else
            lsw2 = .false.
          end if
        end if

      end do                ! do while

      end subroutine sort_ueberpruefen



      subroutine bilde_dreiecke

      implicit none
      integer :: iknot(6), ipunkt(4)
      integer :: itet, icount, j, ih, i1, i2, i3, ico
      real(dp) :: ar, artri3

      if (.not.allocated(itetno)) allocate (itetno(ntri))
      itetno = 0

      ntrii = 0
      do itet=1,ntet
!        iknot(1:6) = (/ (icono(tetra_kanten(j,itet)),j=1,6) /)
```

```
!        icount = count(iknot > 0)
        icount=0
        iloop: do i=1,6
          ico = icono(tetra_kanten(i,itet))
          if (ico > 0) then
            do j=1,icount
              if (ico == iknot(j)) cycle iloop
            end do
            icount = icount + 1
            iknot(icount) = ico
          end if
        end do iloop

        if (icount == 3) then
! schnittgebilde ist ein Dreieck
          ntrii = ntrii + 1
          necke(1:3,ntrii) = iknot(1:3)
          i1=necke(1,ntrii)
          i2=necke(2,ntrii)
          i3=necke(3,ntrii)
          itetno(ntrii) = itet
          ar = artri3(xtrian(i1),ytrian(i1),0._dp,
     .                xtrian(i2),ytrian(i2),0._dp,
     .                xtrian(i3),ytrian(i3),0._dp)
          if (ar < 0._dp) then
! orientierung des dreiecks stimmt nicht, drehe die punkte 2 und 3
            ih = necke(2,ntrii)
            necke(2,ntrii) = necke(3,ntrii)
            necke(3,ntrii) = ih
          end if

        else if (icount ==4) then
! schnittgebilde ist ein Viereck
          ipunkt = iknot(1:4)

! sorge dafuer, dass die Linie p1,p2,p3,p4 ein Viereck bildet und
! sich nicht schneidet
          call sort_ueberpruefen (ipunkt)
! bilde erstes Dreieck p1, p2, p3
          ntrii = ntrii + 1
          necke(1:3,ntrii) = (/ ipunkt(1), ipunkt(2), ipunkt(3) /)
          i1=necke(1,ntrii)
          i2=necke(2,ntrii)
          i3=necke(3,ntrii)
          itetno(ntrii) = itet
          ar = artri3(xtrian(i1),ytrian(i1),0._dp,
     .                xtrian(i2),ytrian(i2),0._dp,
     .                xtrian(i3),ytrian(i3),0._dp)
          if (ar < 0._dp) then
! orientierung des dreiecks stimmt nicht, drehe die punkte 2 und 3
            ih = necke(2,ntrii)
            necke(2,ntrii) = necke(3,ntrii)
            necke(3,ntrii) = ih
          end if

! bilde zweites Dreieck p1, p3, p4
          ntrii = ntrii + 1
          necke(1:3,ntrii) = (/ ipunkt(1), ipunkt(3), ipunkt(4) /)
          i1=necke(1,ntrii)
          i2=necke(2,ntrii)
          i3=necke(3,ntrii)
          itetno(ntrii) = itet
          ar = artri3(xtrian(i1),ytrian(i1),0._dp,
```

```
     .                xtrian(i2),ytrian(i2),0._dp,
     .                xtrian(i3),ytrian(i3),0._dp)
          if (ar < 0._dp) then
! orientierung des dreiecks stimmt nicht, drehe die punkte 2 und 3
            ih = necke(2,ntrii)
            necke(2,ntrii) = necke(3,ntrii)
            necke(3,ntrii) = ih
        end if

        end if
      end do

      do itri=1,ntrii
        xcom(itri) = (xtrian(necke(1,itri)) + xtrian(necke(2,itri)) +
     .                xtrian(necke(3,itri))) / 3._dp
        ycom(itri) = (ytrian(necke(1,itri)) + ytrian(necke(2,itri)) +
     .                ytrian(necke(3,itri))) / 3._dp
      end do

      end subroutine bilde_dreiecke


      end subroutine rpscut
```

Danksagung

Herzlich bedanken möchte ich mich bei Herrn Prof. Dr. Jörg Keller für die Vergabe dieses Themas.

Bedanken möchte ich mich auch beim Institut für Plasmaphysik für die Möglichkeit diese Arbeit teilweise während der Arbeitszeit erstellen zu können.

Mein besonderer Dank gilt Herrn Prof. Dr. Detlev Reiter für viele hilfreiche Diskussionen und klärende Hinweise zu Physik und Statistik.

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfaßt und keine anderen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Düren, den