Simulationen zum EDDA-Experiment mit GISMO und MICRES

von

Uwe Roß

Diplomarbeit in Physik
angefertigt im
Institut für Strahlen- und Kernphysik

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität

Bonn

im November 1994

Referent: Prof. Dr. F. Hinterberger
Korreferent: Prof. Dr. J. Ernst

Zusammenfassung

Das interne Experiment EDDA am Beschleuniger COSY wurde zur Messung der Anregungsfunktion der Proton-Proton-Streuung im Bereich kinetischer Energien zwischen 250 und 2500 MeV mit hoher Energie- und Winkelauflösung konzipiert. Im Rahmen dieser Arbeit wird ausgehend von der am SLAC entworfenen $C{+}{+}$-Klassensammlung GISMO ein Programm zur Monte-Carlo-Simulation des EDDA-Detektors entwickelt. Durch die Einbindung des Codes MICRES in GISMO, wird eine akkurate Beschreibung hadronischer Prozesse bei mittleren Energien erreicht. Im Experiment dient ein CH_2-Fädchen als Target. Neben den zu untersuchenden *pp*-Stößen, kommt es zu Konkurrenzreaktionen am Kohlenstoff. Eine Simulation des Kohlenstoffuntergrundes wird vorgestellt und mit einer experimentellen Bestimmung verglichen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Das EDDA-Experiment

Das Experiment EDDA[1] wurde zur Messung der Anregungsfunktionen der elastischen Proton-Proton-Streuung über einen großen Energie- und Winkelbereich konzipiert. Die Messungen werden mit Hilfe eines internen Targets am Beschleunigerring COSY[2] durchgeführt. Die kinetische Energie des Protonenstrahls erstreckt sich dabei über einen Bereich von 250 bis 2500 MeV. Die Datenaufnahme während der Hochbeschleunigung und die Mittelung über viele Beschleunigungsrampen erlauben eine Energieauflösung ΔE bis 1 MeV. Im genannten Energiebereich deckt der Detektor einen Raumwinkel zwischen 85% und 89% von 4π im CM-System ab, wobei eine Winkelauflösung $\Delta\theta_{cms}$ zwischen 3° und 5° erreicht wird [Hin 89].

In der ersten Phase des Experiments werden unpolarsierte Wirkungsquerschnitte mit einem aluminiumbedampften Polypropylenfädchentarget von wenigen μm Dicke gemessen. In der zweiten Phase sollen mittels polarisiertem Protonenstrahl und polarisiertem Wasserstoffatomstrahltarget die Vektoranalysierstärke und Polarisationskorrelationsparameter gemessen werden.

Die angestrebte hohe relative Genauigkeit bei der Messung von Anregungsfunktionen und Winkelverteilungen ermöglicht die Fortsetzung der Streuphasenanalyse hin zu höheren Energien. Selbst schwache Anomalien der Anregungsfunktionen, mit Resonanzexkursionen von nur 5%, sollten beobachtbar sein und so Aufschluß über Existenz bzw. Nichtexistenz dibaryonischer Resonanzen geben.

1.2 Die Geometrie des EDDA-Detektors

Abb.1.1 zeigt schematisch den Aufbau des Detektors. Er besteht aus vier Lagen szintillierenden Polyvinyls bzw. Polystyrens. Die äußere Lage ist ein Zylinder aus 29 Halbringpaaren. Nach innen schließt sich eine aus 32 sich überlappenden Latten gebaute Lage an. Die äußeren Lagen sind seit Sommer 1994 vollständig

[1] Excitation Function Data Acquisition Designed for Analysis of Phase Shifts

[2] Das Cooler Synchroton des Forschungszemtrums Jülich beschleunigt Protonen und leichte Ionen mit hoher Impulsschärfe und geringer Strahlausdehnung [May 89].

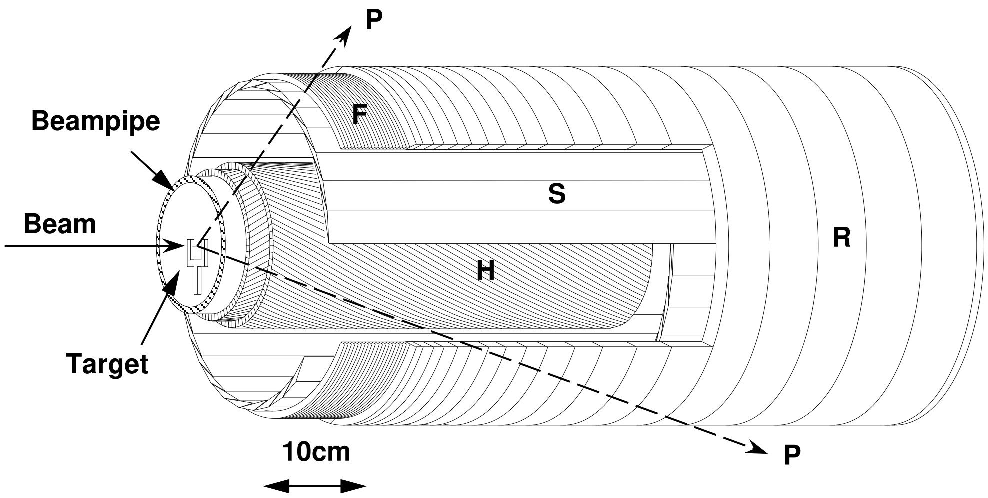

Abbildung 1.1: Der EDDA-Detektor mit Ringlage R (und F), Latten(Stab)lage S, Helixlage H, Strahlrohr und Target.

aufgebaut. Die beiden inneren Lagen, von denen nur eine in Abb.1.1 gezeichnet ist, bestehen aus je einer Schicht links- bzw. rechtsdrehender Helizes aus szintillierenden Fasern. Fertigstellung und Einbau der Helixlagen werden im Frühjahr 1995 abgeschlossen sein.

Latten und Ringe dienen nicht nur der Winkelrekonstruktion, sondern werden auch zum Triggern auf elastische pp-Ereignisse benutzt. Elastische Streuung identischer Teilchen ist durch Koplanarität und eine kinematische Korrelation der Streuwinkel im Laborsystem gekennzeichnet, was sehr restriktive Triggerbedingungen erlaubt, um Untergrundereignisse aus inelastischer pp-Streuung auszuschließen [Sch 91]. Nur wenn gegenüberliegende Latten (koplanarer Trigger) und kinematisch erlaubte Kombinationen von Halbringen (kinematischer Trigger) feuern, wird der Detektor ausgelesen.

In der zweiten Phase des Experiments reichen Latten- und Ringlagen allein zur Rekonstruktion der Streuwinkel nicht aus, da wegen der Ausdehnung des Atomstrahltargets der Vertex nur ungenügend bekannt ist. Mit Hilfe der Helixlagen sind nach analytischen Abschätzungen Vertexbestimmungen mit Standardabweichungen um 1 mm möglich [Lah 92].

1.3 Offene Fragen bei der Simulation von EDDA

Zum EDDA-Experiment wurden bereits eine Reihe von Simulationen durchgeführt. Die ersten Untersuchungen zum Beitrag inelastischer pp-Ereignisse [Sch 91] und von $p^{12}C$-Reaktionen [The 92] zum Untergrund wurden ohne Berücksichtigung der Detektorgeometrie gemacht. Für die Bestimmung des Kohlenstoffuntergrundes wurde der hadronische Eventgenerator MICRES entwickelt. Detailierte Studien zum Einfluß von δ-Elektronen und inelastischen pp-Ereignissen auf den Untergrund und ein Vergleich verschiedener Varianten des koplanaren Triggers finden sich in [Gro 93]. Bei diesen Simulationen wurden die erzeugten Teilchen mit Hilfe des Programmpakets GEANT durch den Detektor *getrackt*. Die Helixlagen wurden dabei jedoch nur als Zylinder ohne innere Struktur behandelt.

Diese Arbeit beschäftigt sich mit der Simulation des Kohlenstoffuntergrundes unter Berücksichtigung der z.Zt. aufgebauten Detektorgeometrie. Das für diese Untersuchungen zu schreibende Programm soll ferner leicht auf die vollständige Detektorgeometrie erweiterbar sein.

Die zu erwartenden Schwierigkeiten bei der Programmierung der Helixlagen in GEANT machen es sinnvoll, die Verwendung eines anderen Simulationspaketes zu erwägen. Dieses sollte nicht nur die Simulation der Helizes ermöglichen, sondern auch die Verwendung des MICRES-Codes zur Simulation hadronischer Ereignisse im Target und beim Tracking erlauben. Die vielversprechendste Alternative zu GEANT ist derzeit die *C++-Klassensammlung* GISMO. GEANT und GISMO werden in Kap. 2 verglichen. Da GISMO geeigneter erscheint, wird es in Kap. 3 ausführlicher beschrieben, wobei auch kurz auf die Simulation der Helixlagen eingegangen wird. Kap. 4 behandelt den hadronischen Eventgenerator MICRES und seine Implementierung in GISMO. Die Auswertung erster Experimente mit der Außenschale des Detektors zeigt den erwarteten scharfen Peak im Korrelationsplot von kinematischem und koplanarem Trigger [Roh 94]. Eine entsprechende Simulation mit Beiträgen der Protonstreung an Wasserstoff, Kohlenstoff und Aluminium zur Triggerrate wird in Kap. 5 vorgestellt und mit den experimentellen Ergebnissen verglichen.

1.4 Einige Bezeichnungen

Im folgenden sei noch auf einige begriffliche Unterscheidungen hingewiesen, die im Rahmen von Simulationen sinnvoll sind, aber vom physikalischen Standpunkt aus zunächst unnatürlich erscheinen mögen. Ein *Event-Generator* ist ein Code, der eine Liste von Teilchen mit Impulsen erzeugt. Dies kann z.B. ein Programm sein, das eine Liste von Ejektilen aus einem radioaktiven Zerfall anlegt. Ein weiteres Beispiel ist der Code MICRES, der die Teilchen im Ausgangskanal eines Hadron-Kern-Stoßes angibt. Physikalisch macht es dabei keinen Unterschied, ob ein Strahlteilchen an einem Kern im Target streut, oder ob ein Sekundärteilchen einer solchen Reaktion auf seinem Weg durch den Detektor einen Stoß erleidet. Im engeren Sinn ist mit *Event-Generator* jedoch nur der Code gemeint, der Ereignisse im Target simuliert. Für die Simulation des Teilchenflugs — das sog. *Tracking* — durch den Detektor wird noch eine Funktion benötigt, die berechnet, wie häufig ein Stoß in einem gegebenem Medium stattfinden soll. Diese Funktion zusammen mit dem Code, der den Stoß beschreibt, wird als *Tracking*- oder *Shower*[3] -Code bezeichnet.

[3] Die Bezeichnung *Shower* ist genaugenommen nur dann gerechtfertigt, wenn die Energie ausreicht, eine Vielzahl von Sekundärteilchen — einen Teilchen-Schauer — zu erzeugen.

Kapitel 2

Die Wahl der geeigneten Simulations-Software

2.1 Allgemeine Überlegungen

Eine Simulation soll experimentelle Daten nicht nur möglichst genau, sondern auch in möglichst kurzer Zeit reproduzieren. Je vollständiger und genauer die Simulation sein soll, desto größer ist aber auch der Aufwand für die Programmentwicklung und desto länger die Rechenzeit.

Anstatt einen eigenen Simulationscode von Grund auf neu zu schreiben, bietet sich in unserem Falle an, mit einem der zur Verfügung stehenden Softwarepakete GEANT bzw. GISMO zu arbeiten, die speziell für die Simulation von Experimenten der Hochenergiephysik geschrieben wurden. Neben der eingesparten Entwicklungszeit, stellt die Unterstützung durch große Arbeitsgruppen, die Verbesserungen und Erweiterungen zur Verfügung stellen, einen Vorteil dar. Für die Simulation des EDDA-Experiments müssen beide Codes jedoch noch erweitert bzw. verändert werden. Während sich dies beim durchweg in FORTRAN geschriebenen — und von daher unübersichtlichen — GEANT grundsätzlich schwierig gestaltet, gehört die Option der Veränderung — genauer des Überschreibens — des GISMO-Codes zu dessen wesentlichen Eigenheiten. Die Schnittstellen werden dabei strikt beibehalten, so daß Verständlichkeit des Programmtextes und die Kompatibilität zwischen verschiedenen Versionen gewährleistet sind. Dies liegt an der konsequenten Verwendung der objektorientierten Sprachelemente der Programmiersprache C++ in GISMO. Im folgendnen werden die beiden Softwarepakete mit ihren Vor- und Nachteilen — im allg. und bezogen auf die Simulation des EDDA-Experiments — kurz vorgestellt und verglichen.

2.2 GEANT

Das Simulationspaket GEANT wird vom CERN zur Verfügung gestellt und ist in der Version 3.14 bei den bisherigen Simulationen des EDDA-Experimentes bereits benutzt worden [Gro 93]. Darüber hinaus wurde der für die Mittelenergiephysik geschriebene Eventgenerator MICRES als *hadronischer Shower*

Code implementiert, der bei kinetischen Energien unterhalb 4.5 GeV den Hochenergie-Code GHEISHA ersetzt [The 92]. Zu GEANT gibt es eine Dokumentation derjenigen Routinen, die vom Benutzer angesprochen werden [CER 87].

Die Helixlage des EDDA-Detektors kann mit den Standard-Volumenelementen von GEANT3.14 nicht realisiert werden. Die Programmierung neuer Volumentypen durch GEANT-Benutzer ist nicht dokumentiert und verlangt daher zeitintensives Studium des FORTRAN-Programmtextes, um Kenntnis über die Schnittstellen (COMMON-Blöcke und SUBROUTINEn) zu gewinnen. Da MICRES auch zur Simulation von Ereignissen im Target verwendet werden soll, muß es aus GEANT wieder „extrahiert“ werden, d.h. die Schnittstellen müssen offengelegt werden.

2.3 GISMO

GISMO wird seit Frühjahr 1993 vom SLAC vertrieben. Es enthält wie GEANT den in FORTRAN geschriebenen GHEISHA *Shower Code*. Alle anderen Routinen — insbes. auch die Schnittstelle zu GHEISHA — sind in C++ geschrieben. Zu GISMO gibt es eine kurze, an Beispielen orientierte Einführung [Atw 93a]. Darüber hinaus ist der Programmtext an den entscheidenden Stellen weitgehend selbsterklärend.

Mit der Installation des GISMO-Programmpaketes Version 0.4[1] begannen wir im März 1994. Nach nur etwa einem Monat Installations- und Testphase konnten bereits folgende Erfolge erzielt werden:

- GISMO Vers.0.4 lief auf DEC-ULTRIX-MIPS und DEC-ALPHA-AXP Arbeitsplatzrechnern.

- Die Geometrie des EDDA-Detektors war auf dem Stand der GEANT-Simulation zu dieser Zeit, d.h., daß nur die Helixlagen fehlten und kleinere Korrekturen durchgeführt werden mußten.

- Für die Simulation der Ereignisse im Target standen der Bertini-Code in der von P.Cloth [Clo 88] modifizierten Version INC77 und ein Generator zur Erzeugung elastischer *pp*-Ereignisse zur Verfügung.

- Ein entsprechender Treiber erlaubt die Ausgabe von Bildern simulierter Ereignisse im *Postscript*-Format.

Auch mit GISMO ist die Programmierung der Helix nicht mit den Basisvolumentypen möglich. Die Realisierung neuer Volumenelemente ist in GISMO jedoch vergleichsweise einfach. MICRES kann nicht ohne weiteres als *Shower Code* oder *Target Event Generator* genutzt werden. Es muss erst aus GEANT extrahiert werden,um in GISMO implementiert werden zu können.

[1] Mittlerweile benutzen wir HEPGISMO, vgl. Kap. 3.1.

2.4 GISMO vs. GEANT

GEANT ist ein — in vielen Fällen — bewährtes Programmpaket. GISMO ist neu und rechnet — zur Zeit — weder schneller noch genauer als GEANT. Im Gegensatz zu GEANT ist GISMO jedoch von vornherein so konzipiert, daß es unter Gewährleistung der Kompatibilität zwischen verschiedenen Versionen flexibel und erweiterbar ist [Bur 93]. Unsere ersten Erfahrungen mit GISMO bestätigen, daß GISMO diesem Anspruch gerecht wird.

Die weitere Arbeit konzentriert sich daher auf die Simulation des EDDA-Experiments auf der Basis von GISMO. Auf die Programmierung der Helix wird dabei nur kurz in Kap. 3.3.1 eingegangen. Die Extraktion von MICRES aus GEANT und die Implementierung in GISMO wird detailiert in Kap. 4 beschrieben.

Kapitel 3

Der Simulations-Code GISMO[1]

3.1 Einleitung

GISMO ist eine Sammlung *objektorientierter Klassen* (eine sog. *class library*), die einen Rahmen für die Programmentwicklung bei der Simulation von Teilchentransport durch Materie bilden. Anwendungsgebiete sind nicht nur Experimente mit Teilchendetektoren in der Hoch- und Mittelenergiephysik, sondern auch die Bestrahlung menschlichen Gewebes in der medizinischen Diagnose und Therapie ist z.B. denkbar. Die Sammlung umfaßt Klassen zur Beschreibung von Geometrien, Materialdaten, Teilchen und Wechselwirkungen. Während der Datenaustausch zwischen diesen Klassen festgelegt ist, kann die Art und Weise, wie die Daten verarbeitet werden, leicht verändert oder erweitert werden. Als Beispiel sei die Implementation einer neuen Wechselwirkung genannt (Vgl. Kap. 4.6). Die Konzeption und Entwicklung von GISMO ist eng verbunden mit dem Paradigma der *objektorientierten Programmierung* (OOP) [Atw 90], [Atw 92], [Bur 93]. Die Geschichte von GISMO wird in [Atw 93b] dargestellt.

HEPGISMO

Zur Zeit gibt es zwei Versionen von GISMO: neben GISMO, das vom SLAC vertrieben wird, noch HEPGISMO[2] [Bur 94] von T. Burnett, das einige Vorzüge gegenüber GISMO hat. GISMO behandelt die hadronischen und elektromagnetischen *Tracking-Codes*, GHEISHA und EGS, als Grundbestandteile, bei HEPGISMO sind sie von einer Basiswechselwirkungsklasse abgeleitet. Dasselbe gilt für Materialdaten, die mit bestimmten Wechselwirkungsklassen verbunden sind. Das *Tracking* bei HEPGISMO ist vereinfacht. Es ist zu erwarten, daß diese Verbesserungen in Zukunft Einzug in GISMO halten werden [Atw 93a],

[1] Graphical Interface for Simulation and Monte Carlo with Objects. Die Bezeichnung *Graphical Interface* rührt von der GISMO-Urversion her, die nur auf NeXT-Stations läuft [Atw 92].

[2] T.Burnett benutzt in seiner GISMO-Version die *Class Library for High Energy Physics* CLHEP. Diese stellt einige Zufallszahlengeneratoren, zwei Vektorklassen und einige nützliche Programmierhilfen zur Verfügung. Eine Spezialisierung auf Hochenergiephysik ist damit nicht verbunden.

[Bur 94]. Die folgende Darstellung bezieht sich daher auf HEPGISMO. An den Stellen, an denen es vom SLAC-GISMO abweicht, wird im Text explizit noch einmal hingewiesen.

3.2 Objektorientierte Programmierung (OOP)

Wie oben erwähnt, ist GISMO ohne objektorientierte Programmierung nicht denkbar. GISMO gehört damit zu einer ganzen Reihe neuerer Programm-Codes für verschiedene physikalische Anwendungen (wie z.B. Datenanalyse, Messen und Regeln, Simulation), die objektorientierte Techniken nutzen [Arn 92]. Zum besseren Verständnis — nicht nur von GISMO — folgt daher eine kurze Einführung in OOP.

OOP ist eine Programmiertechnik und keine Programmiersprache, allerdings unterstützt nicht jede Sprache OOP. Die meisten objektorientierten Programmiersprachen bauen auf vorhandenen Programmiersprachen auf. Während bei der konventionellen *prozeduralen* Programmierung die Beschreibung der ablaufenden Prozesse im Vordergrund steht, liegt bei der OOP das Hauptgewicht auf der Beschreibung realer oder abstrakter Objekte, die durch ihre Eigenschaften und möglichen Aktionen bestimmt sind. Durch OOP werden insbesondere umfangreiche Programme wesentlich handhabbarer und flexibler. Zum Verständnis von OOP gehört die Kenntnis einiger weniger Schlüsselwörter, die im folgenden kurz genannt und in Anhang A an einfachen Beispielen erläutert werden sollen. Anhang A enthält auch Hinweise auf ein- und weiterführende Literatur.

Schlüsselbegriffe[3]

Objektorientierte Sprachen erlauben dem Programmierer die Definition eigener Variablentypen, *Klassen* genannt. Jede dieser Klassen kann aus mehreren Variablen, *Datenelementen*, verschiedener Typen und Funktionen, sog. *Elementfunktionen* oder auch *(Klassen-)Methoden*, die auf den Datenelementen operieren, aufgebaut sein. Die Variablen eines bestimmten Klassentyps werden als *Objekte dieser Klasse* bezeichnet. Der Zugriff auf die Elemente, also das Lesen und Schreiben der Datenelemente eines Objekts und der Aufruf von Klassenmethoden, kann eingeschränkt werden. Die Elemente sind dann *gekapselt*. Es ist erlaubt, neue Klassen von bereits definierten abzuleiten. Die Eigenschaften der Basisklasse müssen dabei nicht neu beschrieben werden, sie werden *vererbt*.

3.3 Die wichtigsten GISMO-Klassen[4]

3.3.1 Geometrie

Die Basisklassen für Geometrie sind `Volume`, `Surface` und `Ray`. `Volume` legt die Elemente eines allgemeinen Volumens fest. Von `Volume` sind alle Klassen, die

[3] Im folgenden Abschnitt sind die Schlüsselbegriffe bei ihrer ersten Nennung *kursiv* gedruckt.
[4] Im folgenden stehen Klassen- und Elementnamen in `Schreibmaschienenschrift`. Ihre Namen sprechen im allg. für sich.

konkrete Formen beschreiben, abgeleitet. GISMO stellt einige solcher Klassen wie z.B. `Box` oder `Tube` bereits zur Verfügung. Diese Formen werden aufgefaßt als Listen endlicher, orientierter Oberflächen. Die Eigenschaften dieser Oberflächen werden durch die Klasse `Surface` bestimmt, von der wiederum spezielle Oberflächen wie `Rectangle` und `Cylinder` abgeleitet sind. Die wichtigsten Datenelemente sowohl in `Volume`, als auch in `Surface`, enthalten die Lage und Orientierung im Raum. Die wichtigsten Methoden bewegen die Objekte im Raum und berechnen zu einem gegebenem Punkt ob er innerhalb oder außerhalb[5] liegt, seinen kürzesten Abstand und seinen Abstand entlang eines bestimmten Weges. Wege werden in `Ray` und der davon abgeleiteten Klasse `Helix` definiert. Bei den Volumina werden die Abstandsberechnungen von den Oberflächen in der Liste übernommen. Der Abstand zum Volumen ist der kürzeste Abstand zu einer Oberfläche aus der Liste. Dies vereinfacht die Definition neuer Volumina. Diese Methoden können jedoch auch überschrieben werden, was sinnvoll sein kann, weil

- in einigen Fällen der Programmablauf deutlich beschleunigt wird.
- bei komplizierten Formen die Parametrisierung des Volumens durch die begrenzenden Oberflächen nicht geeignet erscheint.

Z.B. werden bei der Simulation des EDDA-Detektors die helixförmigen szintillierenden Fasern durch die Faserseelen und den Faserradius parametrisiert [Gro 94]. Die Liste der Oberflächen bleibt in diesem Falle leer.

3.3.2 Materialien[6]

Materialien werden in HEPGISMO durch `Material` und `MatData` erklärt. `Material` enthält die Daten, die ein Material in jedem Fall haben muß, damit der Teilchentransport in HEPGISMO funktioniert. `MatData` beschreibt das Format für Materialdaten, die von bestimmten Event-Generatoren wie z.B. EGS oder MICRES benötigt werden (vgl. Kap.4.6.2). Beide Klassen lesen aus speziellen Materialdatenfiles, die zur Zeit nur mit Hilfe des PEGS4 Preprozessors zu EGS4 [Nel 85], [Bie 94] erzeugt werden können.

3.3.3 Äußere Felder

Die Klasse `Field` erlaubt die Beschreibung konstanter Magnetfelder.

3.3.4 Medien

Bei der Simulation eines Experiments unterteilt man den Versuchsaufbau in kleinere Einheiten, denen geometrische Eigenschaften, Materialeigenschaften und — zumindest annähernd — ein konstantes Magnetfeld im Innern zugeschrieben werden kann. In GISMO entsprechen diese Teile Objekten der Klasse `Medium`. Jedes `Medium`-Objekt enthält also je ein Objekt der Klassen `Volume`,

[5] Dies macht auch für Oberflächen Sinn, da sie orientiert sind.

[6] In diesem Punkt unterscheiden sich SLAC-GISMO und HEPGISMO.

`Material` und `Field`. `Medium`-Objekte können in einer Baumstruktur angeordnet werden, d.h., daß jedes `Medium`-Objekt weitere `Medium`-Objekte umfassen bzw. selbst Element eines `Medium`-Objekts sein kann. Ferner kann jedem `Medium`-Objekt ein `Detector`-Objekt zugeschrieben werden, das festlegt, ob und welche Informationen nach dem Durchgang eines Teilchens aufgeschrieben werden.

3.3.5 Detektoren

Benutzereigene Detektoren können von der Klasse `Detektor` abgeleitet werden. Dabei müssen die Methoden `Detector::score(MCParticle *)`, die die gewünschten Daten — z.B. Energieverlust — akkumuliert, solange sich das Teilchen in einem `Medium` befindet, und `Detector::generateResponse()`, die die Daten abspeichert, den Anforderungen entsprechend neu definiert werden.

3.3.6 Teilchen

Es gibt in HEPGISMO zwei Teilchenklassen: `GParticle` und die davon abgeleitete Klasse `MCParticle`. Während über `GParticle` auf die Teilchendaten wie z.B. Masse, Ladung und Impuls zugegriffen werden kann, übernehmen `MCParticle`-Methoden — insbes. `MCParticle::propagate(Medium *)` — die Berechnung des Teilchentransports[7]. Die Teilchentypen — p, e^-, K_L^0 etc. — sind mit ihren Wechselwirkungen und evtl. Zerfallsmechanismen in der Klasse `PDT`[8] tabelliert. Die entsprechenden Daten — wie Masse und Spin — stehen in der Klasse `PDTEntry`. Der Zugriff auf diese Daten wird durch die Klasse `PData` und die davon abgeleitete Klasse `AntiPData` geregelt.

3.3.7 Wechselwirkungen

Die Basisklasse für Wechselwirkungen heißt `Interactor`. Mit ihren Methoden `interactionLength(GParticle *, Medium *)` und `interact(GParticle *, Medium *)` ist sie wesentlicher Teil des Trackings. Auf `Interactor` wird im Zusammenhang mit der Implementierung von MICRES in Kap. 4.6 noch eingegangen.

[7]Die Unterscheidung zwischen Gismo- und Monte-Carlo-Particle ist historisch bedingt. In SLAC-GISMO wird das *Tracking* von zwei Klassen, `MCTrack` und `MCVertex`, beschrieben. In HEPGISMO wird dies vereinfachend von einer Klasse, `MCParticle`, übernommen.

[8]Particle Data Table

Kapitel 4

Der hadronische Eventgenerator MICRES

4.1 Einleitung

Der *Event-Generator* MICRES simuliert die inelastische Streuung von Hadronen bei mittleren kinetischen Energien — bis zu einigen GeV — mit Kernen. Er besteht aus drei Teilen, die auf zunächst unabhängig voneinander entstandenen Codes aufbauen und von D.Theis zur Verwendung in GEANT erweitert und zusammengefügt wurden [The 92].

Der Kaskaden-Code INC [Ber 63][1] beschreibt Reaktionen am Kern durch quasifreie Stöße an seinen Nukleonen. Die Hadron-Nukleon-Stöße selbst werden mit HADRIN [Hän 84][Hän 86] berechnet. Die Wege des Projektils wie auch der bei der Kaskade entstehenden Sekundärteilchen werden in der selben Art und Weise verfolgt, bis sie entweder einen Stoß erleiden, den Kern verlassen oder ihre Energie unter einen Grenzwert sinkt. Im letzteren Fall wird ihre Energie zur Anregungsenergie des Restkerns addiert. Die Verdampfung des nach dem Stoß angeregten Restkerns kann mit EVAP-4 [Gut 70][2] berechnet werden. Abb.4.1 zeigt schematisch den möglichen Verlauf einer intranuklearen Kaskade mit nachfolgender Verdampfung. Ferner gibt es zwei Routinen, die die totalen Hadron-Nukleon-Wirkungsquerschnitte bzw. die *geometrischen* Wirkungsquerschnitte der Kerne angeben. Diese werden für die Normierung benötigt.

4.2 Die intranukleare Kaskade: INC

4.2.1 Voraussetzungen

Allgemein geht man bei der *intranuklearen Kaskade* davon aus, daß der differentielle Wirkungsquerschnitt einer inelastischen Hadron-Kern-Wechselwirkung durch die differentiellen Wirkungsquerschnitte der elementaren Zweiteilchenkollisionen im Kern bestimmt ist. Nach theoretischen Untersuchungen [Gol 64] ist

[1] Ausgangspunkt für MICRES war der Bertini-Code INC77 von P.Cloth [Clo 88].

[2] MICRES enthält das auf EVAP-4 aufbauende Verdampfungsmodell des HERMES-Pakets [Clo 88].

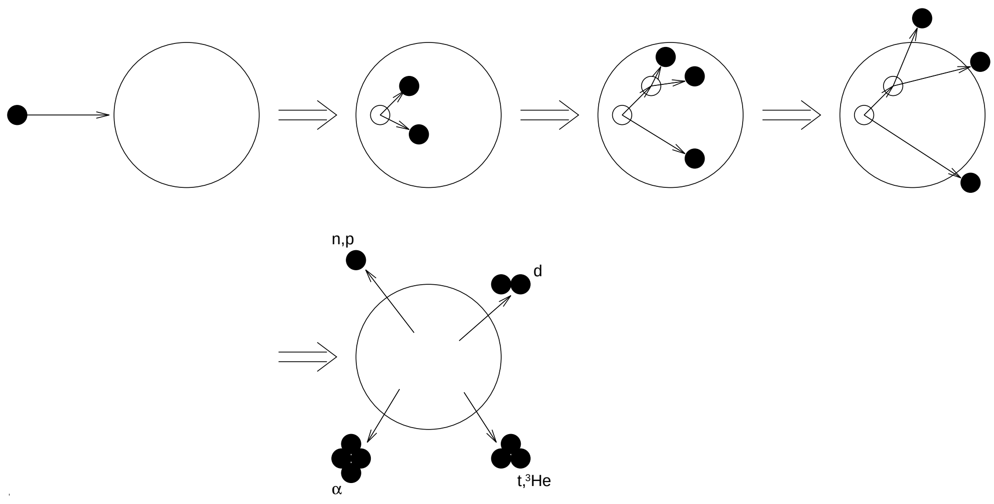

Abbildung 4.1: Eine intranukleare Kaskade und mögliche Verdampfungprozesse des angeregten Restkerns.

dafür hinreichend, daß

- die kinetische Energie des Projektils groß ist gegenüber der Bindungsenergie des streuenden Nukleons (*weak binding, impulse approximation*).
- der Impulsübertrag groß ist im Vergleich zum Fermiimpuls des Nukleons (*quasifreie Streuung*).
- die deBroglie-Wellenlänge des einlaufenden Teilchens wesentlich kleiner ist als seine mittlere freie Weglänge.

4.2.2 Das Kernmodell

Die Nukleonendichte ρ_N wird bei MICRES wie im Bertini-Modell [Ber 63] berechnet. Der Kern ist dabei in drei konzentrische Kugelschalen unterteilt gedacht, in denen die Dichte jeweils konstant ist. Die Protonendichte $\rho_{p,i}$ in jeder Schale ist so gewählt, daß die aus Elektron-Kern-Streuung bekannte Verteilung (4.1) angenähert wird. Die Größen r_0 und a sind an experimentelle Daten angepaßte Konstanten. ρ_0 ist so gewählt, daß das Integral über alle drei Schalen gerade der Kernladungszahl Z entspricht. Die Neutronendichte $\rho_{n,i}$ und die Nukleonendichte ρ_N werden aus den Verhältnissen (4.2) von Neutronen bzw. Nukleonen zu Protonen des Kerns berechnet.

$$\rho_p(r) = \frac{\rho_0}{e^{\frac{r-r_0}{a}} + 1} \tag{4.1}$$

$$\rho_{p,i} = \rho_{n,i}\,\frac{Z}{A-Z} = \rho_{N,i}\,\frac{Z}{A} \quad , \quad i = 1,2,3 \tag{4.2}$$

Für die Berechnung der intranuklearen Kaskade werden die Fermienergien E^f (4.3) und ~impulse p^f (4.4) der Nukleonen benötigt. Diese erhält man aus den Dichten, indem jede Schale als Fermigas (siehe z.B. [May 84]) behandelt wird.

Für die Bindungsenergie des am schwächsten gebundenen Nukleons wird 7 MeV angenommen.

$$E^f_{h,i} = \frac{(3\pi^2\hbar^3\rho_{h,i})^{2/3}}{2m} \tag{4.3}$$

$$p^f_{h,i} = \sqrt{2mE^f_{h,i}} \tag{4.4}$$

$$\text{mit} \quad h = p, n, N \quad \text{und} \quad i = 1, 2, 3 \quad .$$

4.2.3 Das Tracking durch den Kern

Nachdem Gesamtenergie, -impuls und -ladung von Projektil und Targetkern berechnet sind, wird der Eintrittspunkt des Projektils in den Kern gleichverteilt auf der kreisförmigen Querschnittsfläche des Kerns gewürfelt. Der Schritt im Kern erfolgt in Richtung des Impulses mit einer Weite s, die exponentialverteilt um die mittlere freie Weglänge λ liegt.

$$s = -\lambda \ln \xi \ , \quad \xi \in (0, 1) \text{ gleichverteilt.} \tag{4.5}$$

Die mittlere freie Weglänge λ eines Teilchens im Kern — bzw. in der entsprechenden Zone — wird berechnet (4.6) aus seinen totalen Wirkungsquerschnitten $\sigma_{_N}(E_{kin})$ für Stöße mit Protonen bzw. Neutronen und den Nukleonendichten $\rho_{_N}$. Die verwendeten Wirkungsquerschnitte zeigt Abb.(4.2).

$$\lambda^{-1} = \sum_{N=n,p} \rho_{_N} \sigma_{_N} \tag{4.6}$$

Ist s größer als die Entfernung zum Verlassen der jeweiligen Schale, dann wird das Teilchen an ihre Grenze gesetzt und die Schrittweite neu bestimmt. Andernfalls wird am neuen Ort eine Reaktion — Stoß mit Neutron oder Proton oder Pion-Absorption — ausgewählt. Dazu wird der totale Wirkungsquerschnitt bei dieser Energie — unter Berücksichtigung der kinetischen Energie des Nukleons und dem Einfluß des Kernpotentials auf des Projektil — für jede der drei genannten Reaktionen ermittelt. Die kinetische Energie des Nukleons ergibt sich aus seinem Impuls p, der gemäß der Verteilung (4.7) gewürfelt wird. Die Konstante c_i ist so gewählt, daß die Normierungsbedingung (4.8) erfüllt ist. Dem Einfluß des Kernpotentials wird durch Addition der Potentialtiefe $V_i = E^f_i + 7\,\text{MeV}$ zur Gesamtenergie des Projektils Rechnung getragen.

$$f_i(p) = c_i p^2 \tag{4.7}$$

$$\int_0^{p^f_i} f_i(p)\, dp = \int_{Schale\ i} \rho_i\, d^3r \tag{4.8}$$

Ist die ausgewählte Reaktion eine Hadron-Nukleon-Kollision, dann wird diese von HADRIN im Ruhesystem des getroffenen Nukleons durchgeführt. Nach dem Stoß und der Rücktransformation ins Laborsystem müssen die kinetischen Energien aller Nukleonen größer als die Fermienergie in der entsprechenden Region sein — andernfalls ist das Pauli-Prinzip verletzt. In diesem Fall wird so verfahren, als ob keine Reaktion stattgefunden hätte. Die Teilchen des Ausgangskanals werden im Prinzip genauso behandelt wie das Projektil. Nachdem alle Teilchen entweder absorbiert wurden oder den Kern verlassen haben, werden die Zahl der Sekundärteilchen, ihre Gesamtenergien und ihre Impulskomponenten und ferner die Massen- und Ladungszahl, Anregungs- und Rückstoßenergie und Rücktoßimpulskomponenten des Restkerns gespeichert. Dabei wird auf die Erhaltungssätze für Energie, Impuls und Ladung geachtet.

4.2.4 Die Schnittstelle

Die Schnittstelle zu MICRES ist die SUBROUTINE INC. Sie erhält als Parameter die kinetische Energie des Projektils und die Komponenten des normierten Impulsrichtungsvektors sowie zwei Integerzahlen, die das Projektil bzw. den Targetkern kennzeichnen. Zurückgegeben werden die Massen- und Ladungszahl, Anregungs- und Rückstoßenergie sowie die Komponenten des normierten Rückstoßimpulsrichtungsvektors des Restkerns. Die Zahl der Sekundärteilchen, ihre Teilchentypen, ihre Gesamtenergien und — nichtnormierten — Impulskomponenten werden in der SUBROUTINE STORE im COMMON GCKING gespeichert.

4.2.5 Grenzen des Modells

Das Bertini-Modell ist eher für schwere als für leichte Kerne geeignet. MICRES sollte für Massenzahlen $A < 4$ nicht verwendet werden. Die maximale kinetische Energie des Projektils sollte $T_{max} = 4.5$ GeV nicht überschreiten. Es sei noch einmal betont, daß nur nichtelastische Reaktionen mit dem Kern beschrieben werden.

4.3 Der Elementarstoß: HADRIN

HADRIN — ursprünglich Teil des Kaskaden-Codes NUCRIN/HADRIN von K. Hänßgen und J. Ranft [Hän 84], [Hän 86] — beschreibt elastische und inelastische Hadron-Nukleon-Ereignisse. Bei inelastischen Reaktionen werden die Produktion (4.9) und der Zerfall (4.10) von ein oder zwei Resonanzen angenommen.

$$\begin{aligned} h_{in} + N &\longrightarrow h_1 + h_2^* \\ h_{in} + N &\longrightarrow h_1^* + h_2^* \\ h_{in} + N &\longrightarrow h_1^* \end{aligned} \tag{4.9}$$

$$h^* \longrightarrow \cdots \longrightarrow h_1^{decay} + \ldots + h_n^{decay} \tag{4.10}$$

Hierbei bezeichnen h_{in} das Projektil, N das Nukleon, h_i^* Resonanzen und h_1 und h_i^{decay} Hadronen im Ausgangskanal. Dabei werden alle Resonanzen berücksichtigt, die mit mehr als 2% zum totalen inelastischen Wirkungsquerschnitt beitragen. Dies ist ein Vorteil gegenüber dem orginalen Bertini-INC, bei dem nur die

Projektile	Sekundärteilchen
$p, \overline{p}$	$p, \overline{p}$
$n, \overline{n}$	$n, \overline{n}$
π^+, π^-, π^0	π^+, π^-, π^0
K^+, K^-	K^+, K^-, K_L^0, K_S^0
	$\Lambda, \overline{\Lambda}, \Sigma^+, \Sigma^-, \Sigma^0$
	$\gamma, e^+, e^-, \mu^+, \mu^-, \nu$

Tabelle 4.1: Die in der intranuklearen Kaskade behandelten Teilchen [Fil 92].

Δ_{33}-Resonanz berücksichtigt wird. Dagegen beschreibt HADRIN die Winkelverteilung bei elastischer NN-Streuung, elastischer πN-Streuung sowie bei $\pi^{\pm}N$-Ladungsaustauschreaktionen nicht so gut wie der Bertini-Code. Für diese Reaktionen werden daher in MICRES bei Projektilimpulsen $\vec{p}_{lab} = 0.05 \ldots 3\,\mathrm{GeV}/c$ differentielle Wirkungsquerschnitte nach Streuphasenanalysen berechnet. Die Verwendung von HADRIN erweiert die Menge zulässiger Projektile um Antinukleonen und Kaonen, während die Bertini-Kaskade nur Nukleonen und Pionen akzeptiert. Eine Übersicht gibt Tabelle 4.1.

4.3.1 Die Schnittstelle

HADRIN wird über die SUBROUTINE HADRIN1 aufgerufen. Dabei werden Gesamtenergie, Impuls und die Komponenten des Impulsrichtungsvektors im Laborsystem des Projektils und Code-Nummern für Projektil und Targetnukleon übergeben. Die Daten der Sekundärteilchen werden — ähnlich wie im INC — in den COMMON FINLSP geschrieben. Allerdings werden bei HADRIN die Impulse und normierte Richtungsvektoren gespeichert.

4.4 Das Verdampfungsmodell: EVAP

EVAP wurde unverändert aus dem *High Energy Transport Code* HETC [Clo 88] der KFA Jülich übernommen. Es ist eine erweiterte Version des EVAP-4-Codes [Gut 70]. Bei der Verdampfung zerfällt der angeregte Compound-Kern (A, Z) unter Aussendung eines Neutrons, Protons, Deuterons, Tritons, ^3He-Kerns oder α-Teilchens in einen Kern (A', Z'), der auf die selbe Weise in einen Kern (A'', Z'') zerfallen kann. Die Zerfallsreihe endet, wenn die Anregungsenergie für keines der sechs Teilchen zum Verlassen des Kerns ausreicht. EVAP benutzt neuere Kernmassentabellen als EVAP-4. Das Verfahren zur Berechnung nicht tabellierter Kernmassen wurde verbessert. Im Gegensatz zu EVAP-4 kann zwischen zwei verschiedenen Energieniveaudichteparametern gewählt werden. Ferner kann bei EVAP die Winkelverteilung der Ejektile unter Berücksichtigung der Kernbewegung anisotrop erfolgen. EVAP erlaubt auch die Berücksichtigung der Kernspaltung.

4.4.1 Die Schnittstelle

Der Aufruf des Verdampfungs-Code geschieht über die SUBROUTINE EVAP. Ihr werden die Massen- und Ladungszahl, Rückstoßenergie und ~impulsrichtung und die Anregungsenergie des Kerns übergeben. Zurückgegeben werden Massenzahl und Kernladung sowie die Anregungsenergie nach der Verdampfung. Die SUBROUTINE EVASTO speichert die Daten der Zerfallsprodukte im COMMON GCKING (vgl. Kap. 4.2.4).

4.5 Anwendung

Für MICRES bieten sich zwei Anwendungsbereiche. Zum einen der *stand alone Betrieb*, bei dem stets auf das selbe Target geschossen wird und die Reaktionsdaten direkt in Histogramme geschrieben werden, um z.B. Wirkungsquerschnitte für ausgewählte Ereignisse abzuschätzen. Zum anderen als *Event Generator* in einem *Shower Code*, bei dem ein Hadron in Abhängigkeit vom Medium und seiner Energie mit verschiedenen Kernen wechselwirken kann, wobei die Häufigkeit — ähnlich wie in Gl. (4.5) und (4.6) — von totalen Wirkungsquerschnitten abhängt. Die Details werden in Kapitel 4.6 beschrieben. Der nächste Abschnitt erklärt zunächst, wie aus den simulierten Daten Wirkungsquerschnitte gewonnen werden.

4.5.1 Normierung

Wegen der exponentiellen Verteilung (4.5) der Schrittlänge im Kern passiert ein Teil der Projektile den jeweiligen Targetkern ohne mit einem seiner Nukleonen zu wechselwirken. Das Verhältnis der Anzahl n_{react} der *erfolgreichen* zur Gesamtzahl n_{try} der Aufrufe von INC ist proportional zum totalen Reaktionsquerschnitt σ_{react} des Kerns. Die Zahl n_{react} — und damit auch σ_{react} — ist abhängig von der kinetischen Energie E_{kin} des Projektils.

$$\sigma_{react}(E_{kin}) = \frac{n_{react}(E_{kin})}{n_{try}} \, \sigma_{geo} \tag{4.11}$$

Die Normierungskonstante σ_{geo} ist der energie- und projektilunabhängige geometrische Wirkungsquerschnitt, den die SUBROUTINE GETLNC in Abhängigkeit von Ladungs- und Massenzahl zurückgibt.

Bei Wasserstofftargets wird man HADRIN direkt aufrufen. Da bei jedem Aufruf eine — elastische oder inelastische — Wechselwirkung stattfindet, ist die Kenntnis der energieabhängigen totalen Wirkungsquerschnitte notwendig. Diese liefert die SUBROUTINE HXSCT in Abhängigkeit von Teilchentyp und kinetischer Energie des Projektils. Für Nukleonen und Pionen sind die Werte in Abb.(4.2) gezeigt.

4.5.2 Initialisierung

Die für den MICRES-Code benötigten Informationen über Kerne und Wirkungsquerschnitte werden in der Initialisierungsphase — vor dem ersten Aufruf

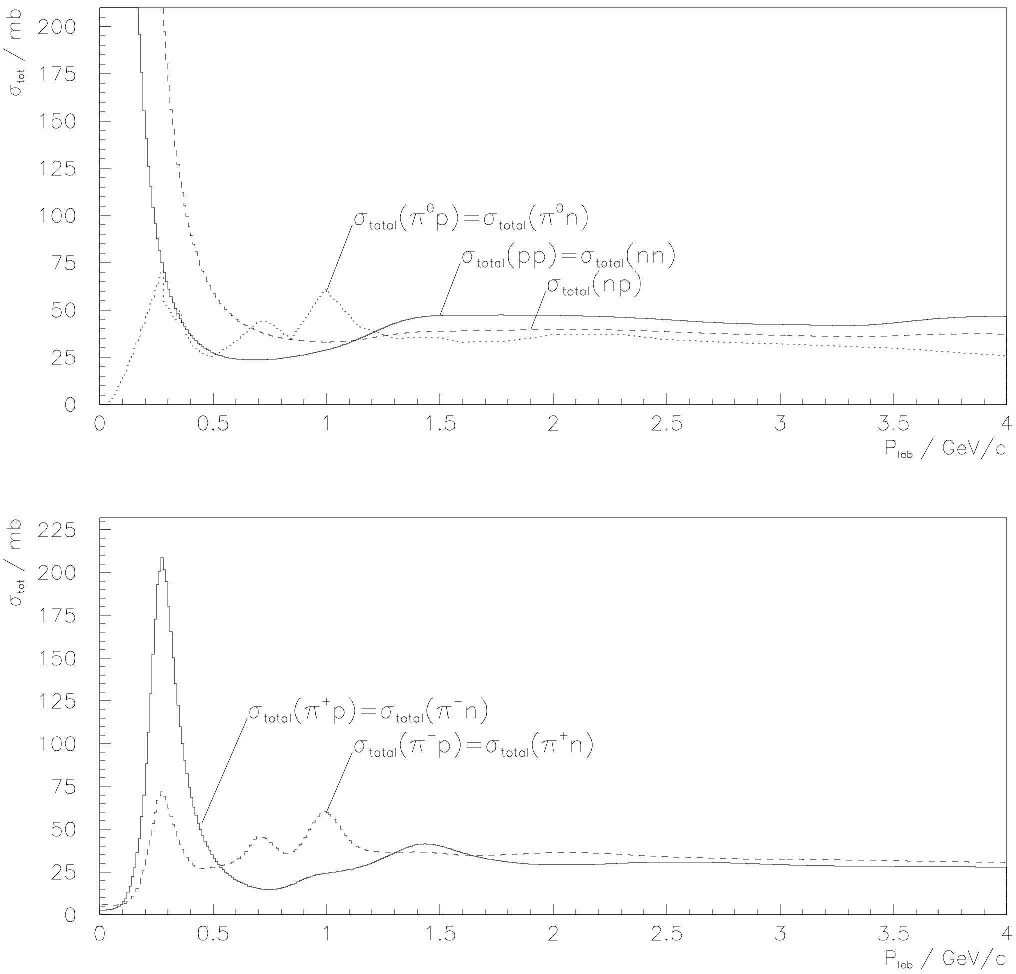

Abbildung 4.2: Die in MICRES benutzten totalen Wirkungsquerschnitte für πN- und NN-Streuung.

von INC oder HADRIN1 — in COMMON-Blöcke geschrieben. Für die intranukleare Kaskade und den Verdampfungs-Code lesen die Routinen INCINIT und EVAPIN die entsprechenden Daten aus Files. Die von HADRIN genutzten Werte stehen in DATA-Blöcken in den Routinen ACFSIN, HADINI und CHANWX.

4.6 Die Implementierung von MICRES in GISMO

Für die Implementierung neuer Eventgeneratoren in GISMO sind drei Klassen von besonderer Bedeutung:

- `MatData` dient als Basisklasse für solche Klassen, die mit `Material`-Objekten (s. Kap. 3.3.2) bestimmte Daten und Methoden verbinden, insbesondere um Wirkungsquerschnitte für das Tracking zur Verfügung zu stellen.

- `Interactor` definiert die Schnittstelle für die Implementierung der Teilchenwechselwirkung in Materie.

- `Generator` legt das Format für Event-Generatoren im engeren Sinne von Kap. 1.4 fest.

Für die Implementierung von MICRES wurden von diesen die Klassen `MicresData`, `Micres` und `MicresGenerator` abgeleitet. Im folgenden werden die wichtigsten Elemente der MICRES-Klassen beschrieben.

4.6.1 Die C++-Klasse Micres

Die Klasse `Micres`, abgeleitet von `Interactor`, enthält die für das Tracking notwendigen Methoden `interactionLength` und `interact`. Im Konstruktor von `Micres` werden die Initialisierungsroutinen (s. Kap. 4.5.2) aufgerufen. `interactionLength` gibt für ein Teilchen die mittlere freie Weglänge λ für hadronische Reaktionen in einem gegebenem Material zurück. Diese hängt von den Teilchendichten und Reaktionsquerschnitten der Elemente des Compounds ab.

$$\lambda^{-1} = \sum_i \lambda_i^{-1} \tag{4.12}$$

$$\lambda_i^{-1} = \rho_i \sigma_i \tag{4.13}$$

Es liegt in der Natur von MICRES, daß dieser Wert für Wasserstoff und Materialien, die Wasserstoff enthalten, von der kinetischen Energie und der Teilchenart abhängt, für alle anderen aber eine Materialkonstante darstellt (vgl. Kap. 4.5).

Typischerweise ist neben dem hadronischen `Interactor` noch ein elektromagnetischer `Interactor` — z.B. `EGSInteractor` — definiert. Während des Trackings werden alle `Interactor` nach ihrer `interactionLength` gefragt, um die dann exponentialverteilt aktuelle Schrittlängen gewürfelt werden. Sofern der Abstand zur Geometriegrenze nicht noch kürzer ist, wird der kürzeste der gewürfelten Schritte gegangen und die entsprechende `interact`-Methode aufgerufen. Diesen Teil des Trackings übernehmen Methoden der Klasse `MCParticle`.

Wird `Micres::interact` aufgerufen und ist das Material in dem die Wechselwirkung stattfinden soll ein Compound mit n Elementen, dann wird zunächst gemäß der Wahrscheinlichkeit p_i (4.14) ein Targetkern gewürfelt. Ist das Target ein Wasserstoffkern wird die SUBROUTINE HADRIN1 (s. Kap 4.3.1) aufgerufen, andernfalls die SUBROUTINE INC (s. Kap. 4.2.4) und anschließend optional die SUBROUTINE EVAP (s. Kap. 4.4.1).

$$p_i = \frac{\lambda}{\lambda_i}\ , \quad i = 1, ..., n \tag{4.14}$$

4.6.2 Die C++-Klasse MicresData

Für die Methoden `interactionLength` und `interact` werden die Materialdaten in einem bestimmten Format gespeichert, das durch die von `MatData` abgeleitete Klasse `MicresData` festgelegt wird. Materialien, die Wasserstoff enthalten, werden dabei anders behandelt als andere. `MicresData` enthält die Zahl der Elemente außer Wasserstoff und ein Array mit den Inversen λ_i^{-1} der mittleren freien Weglängen. Ist auch Wasserstoff enthalten, wird ein Flag gesetzt und die Dichte $\sharp H/\mathrm{cm}^3$ gespeichert. Die Werte werden in der Methode `MicresData::read(istream &, Material &)` berechnet, die sowohl aus Materialdatenfiles im PEGS4-Format (s. Kap. 3.3.2) als auch aus den COMMON-Blöcken von MICRES liest.

4.6.3 Die C++-Klasse MicresGenerator

`MicresGenerator`, abgeleitet von `Generator`, hat als wichtigstes Element ein `Micres`-Objekt. Durch dieses kann in der Methode `getEvent()` durch den Zugriff auf `Micres::interact` ein Ereignis im Targetkern erzeugt werden.

Kapitel 5

Anwendung und Test: Der Einfluß des Untergrundes auf die Triggerrate

5.1 Das Experiment

Der Aufbau des Detektors ist in Kap. 1.2 beschrieben. Gemessen wurde bei fester kinetischer Energie $T_p = 793$ MeV, da hierfür Vergleichswerte vorlagen [Sim 93]. Als Target diente ein $5{\times}4\,\mu\mathrm{m}^2$ CH_2-Fädchen, das mit $15.7\,\mu\mathrm{g/cm}^2 Al$ bedampft war. Zur Bestimmung des Untergrundes wurde ein aluminiumbedampftes Kohlenstoffädchen von $7\,\mu$m Durchmesser verwendet. Auf den Trigger soll unten etwas ausführlicher eingegangen werden. Die Ergebnisse wurden in [Roh 94] veröffentlicht.

Der Trigger

Die besonderen kinematischen Verhältnisse beim elastischen *pp*-Stoß, die sehr einschränkende Triggerbedingungen erlauben, wurden schon in Kap. 1.2 erwähnt. Genauer gilt:

- Die Ereignisse sind koplanar, d.h. die Azimuthwinkel ϕ_i der beiden Protonen im Ausgangskanal erfüllen

$$\Delta\phi \equiv |\phi_1 - \phi_2| = 180°. \tag{5.1}$$

  Im Experiment gilt die koplanare Triggerbedingung dann als erfüllt, wenn mindestens eine Latte und eine der drei gegenüberliegenden Latten gefeuert hat (*koplanarer 1-3-Trigger*).

- Für die Streuwinkel θ_i im Laborsystem besteht die Beziehung

$$\tan\theta_1 \cdot \tan\theta_2 = \frac{1}{\gamma_{cm}^2} = \left(1 + \frac{T_p}{2m_p}\right)^{-1}. \tag{5.2}$$

Hierbei gehört γ_{cm} zur Lorentz-Transformation vom Labor- ins Schwerpunktsystem. Aus Gl. (5.2) ist ersichtlich, daß die Korrelation der Streuwinkel und damit auch die Korrelation zweier Halbringe von der kinetischen Energie T_p abhängt. Im Experiment wurde jedoch bei allen Energien der selbe kinematische Trigger verwendet und zwar so, daß für jeden Halbring der gesamte Energiebereich zwischen 0.5 und 2.5 GeV abgedeckt wurde (*offener kinematischer Trigger*).

Koplanarer und kinematischer Trigger werden *verundet*, d.h. es wird nur dann ausgelesen, wenn beide Triggerbedingungen gleichzeitig erfüllt sind. Trägt man für eine feste Energie die Zählraten gegen $\Delta\phi$ und $\tan\theta_1 \cdot \tan\theta_2$ in einem Lego-Plot auf, sollte sich also ein scharfer Peak bei $\Delta\phi = 180°$ und $\tan\theta_1 \cdot \tan\theta_2 = \gamma^{-2}$ zeigen. Bei $T_p = 793\,\mathrm{MeV}$ gilt $\gamma^{-2} = 0.703$. Abb.5.1 zeigt einen solchen Korrelationsplot am Beispiel der simulierten Daten, der keine wesentlichen Unterschiede zum entsprechenden Diagramm der experimentellen Daten aufweist.

5.2 Die Simulation

Bei der Simulation werden Wasserstoff, Kohlenstoff und Aluminium zunächst unabhängig voneinander behandelt. Für jedes Ereignis im Target werden die Sekundärteilchen durch den Detektor *getrackt*, und für jedes Detektorelement werden die bei einem Ereignis akkumulierten Energieverluste in eine Datei geschrieben. Der Inhalt dieser Datei wird anschließend mit Hilfe eines Konverterprogrammes in das Format übersetzt, in dem auch die im Experiment gespeicherten Daten stehen. Die weitere Auswertung erfolgt dann analog zum Experiment, wobei die Histogramme für Wasserstoff, Kohlenstoff und Aluminium entsprechend den erwarteten relativen Häufigkeiten (s.u.) addiert werden.

Relative Häufigkeiten von Reaktionen an ^1H,^{12}C und ^{27}Al

Die Häufigkeiten, mit denen im Target Streuung an Wasserstoff, Kohlenstoff bzw. Aluminium stattfindet, kann im Prinzip wie in Gl. (4.14) aus den Teilchendichten und Wirkungsquerschnitten berechnet werden. Die Teilchendichten (5.3) von H und C ergeben sich einfach aus der Dichte ρ (in g/cm^3) von Polypropylen. Für Al ergibt sich die Teilchendichte (5.4) aus der Flächenbelegung m/F (in μg/cm^2) durch die Bedampfung. Hierbei sind A und U Querschnittsfläche und Umfang des Targetfädchens. $V = \ell \cdot A$ ist das Volumen eines Fädchens einer festen aber beliebigen Länge ℓ. N_A ist die Avogadro-Konstante.

$$\frac{1}{2}\frac{\sharp H}{V} = \frac{\sharp C}{V} = \frac{\sharp CH_2}{V} = \rho \cdot \frac{N_A}{m_{mol(CH_2)}} \tag{5.3}$$

$$\frac{\sharp Al}{V} = \frac{m}{F}\frac{N_A}{m_{mol(Al)}}\frac{U}{A} \tag{5.4}$$

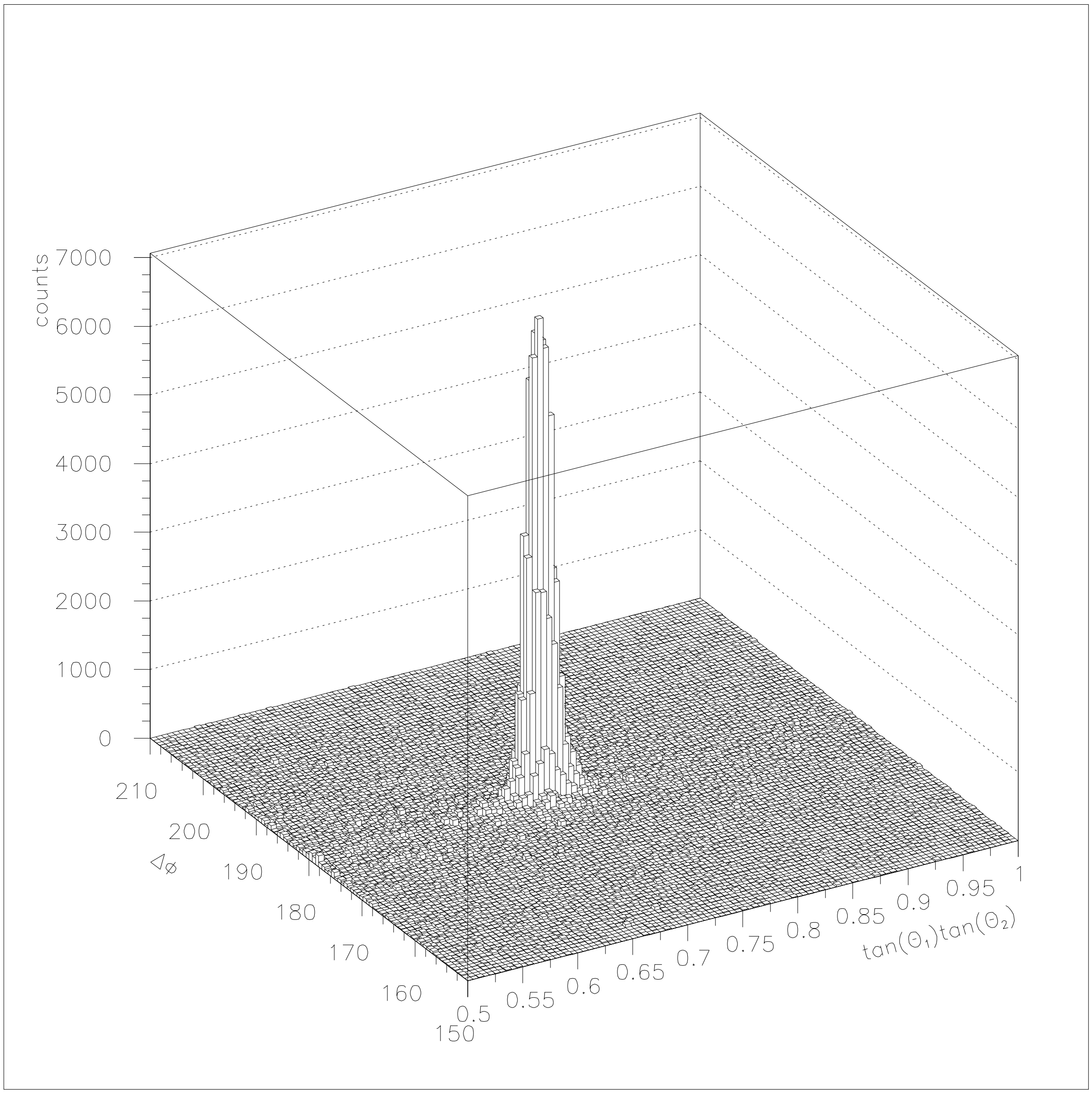

Abbildung 5.1: Zählrate vs. $\Delta\phi$ und $\tan\theta_1 \cdot \tan\theta_2$. Das Diagramm zeigt simulierte Daten.

Die Wirkungsquerschnitte für *pp*-, *pC*- und *pAl*-Streuung bei dieser Energie können mit der *stand-alone*-Version von MICRES ermittelt werden (s. Kap. 4.5). Sie liegen in Übereinstimmung mit experimentellen Werten (vgl. etwa [Ray 79]). Tabelle 5.1 zeigt die so bestimmten Häufigkeiten.

i	p_i
H	0.233
C	0.672
Al	0.095
$\sum_i$	1.000

Tabelle 5.1: Häufigkeiten

5.3 Der Vergleich

Für einen quantitativen Vergleich zwischen Experiment und Simulation sind die Projektionen der oben beschriebenen Lego-Plots auf die $\tan\theta_1 \cdot \tan\theta_2$-Achse am besten geeignet, die Abb. 5.2 zeigt. In guter Näherung stimmen beide überein. Bei genauem Hinsehen fallen jedoch zwei Unstimmigkeiten auf:

- Legt man unter den Peak einen Gauß-Fit, dann liegt der Mittelwert im Experiment beim erwarteten Wert $\tan\theta_1 \cdot \tan\theta_2 = 0.703$, bei der Simulation jedoch bei $\tan\theta_1 \cdot \tan\theta_2 = 0.71$. Die Ursachen hierfür konnten noch nicht geklärt werden. Eine Simulation mit nur elastischen *pp*-Stößen zeigt dieselbe Verschiebung, so daß ein Fehler im Targeteventgenerator unwahrscheinlich ist. Weitere mögliche Ursachen wären z.B. ein zu großer Abstand vom Target zur Ringlage in der Simulation oder eine zu ungenaue Berechnungen der Kleinwinkelstreuung.

- Der Untergrund von Kohlenstoff und Aluminium liegt im Experiment relativ zum Peak höher als in der Simulation. Auch hierfür sind die Ursachen nicht klar. Nach [Ray 79] macht bei $T_p \approx 800$ MeV der Reaktionsquerschnitt für $p^{12}C$-Streuung etwa zwei Drittel des totalen Wirkungsquerschnittes aus. Durch elastische Streuung beschleunigte Kohlenstoffkerne erreichen kinetische Energien bis in die Größenordnung 100 MeV und werden sicherlich im Strahlrohr gestoppt. Möglicherweise tragen dabei entstehende Sekundärteilchen zum Untergrund bei. Entsprechende Eventgeneratoren stehen allerdings z.Zt. nicht zur Verfügung.

5.4 Ausblick

Durch die Verbindung von MICRES und GISMO steht ein geeignetes Werkzeug für Simulationen zum EDDA-Experiment zur Verfügung. Mit diesem können

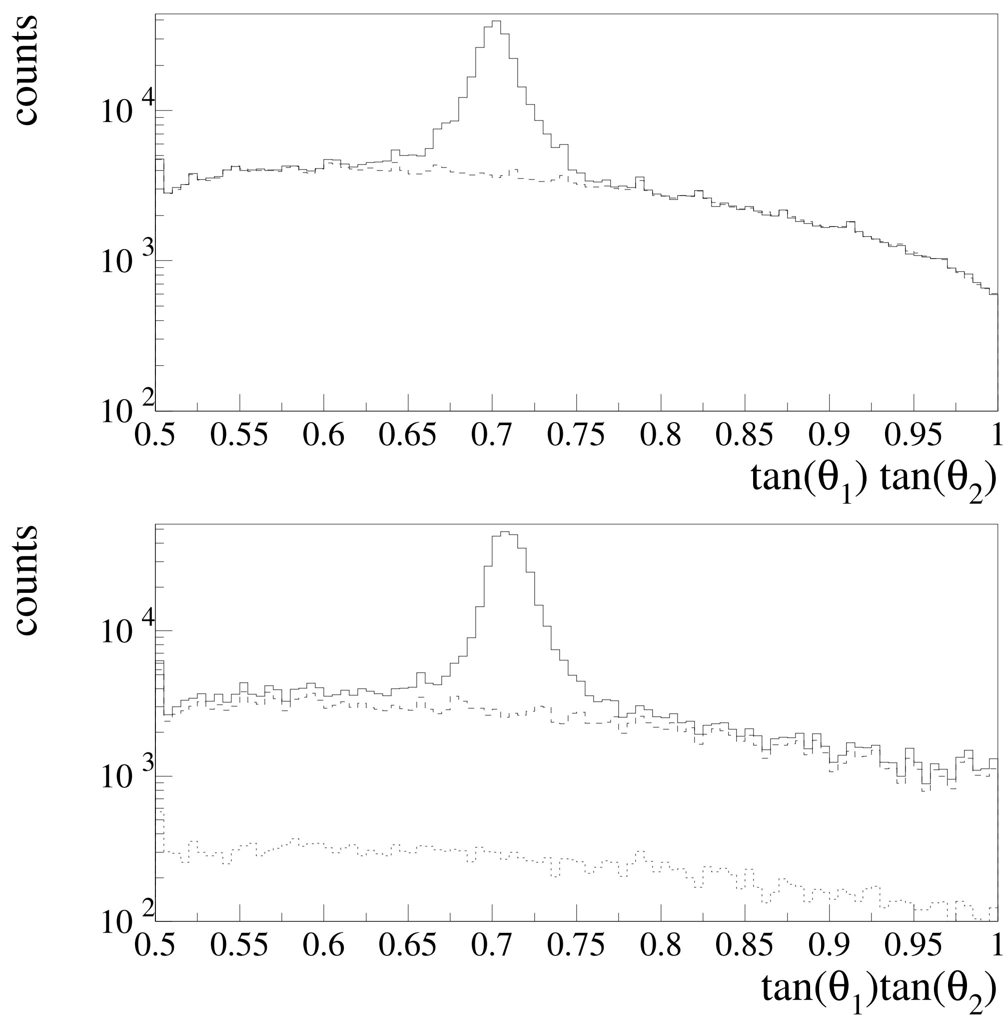

Abbildung 5.2: Zählrate vs. $\tan\theta_1 \cdot \tan\theta_2$ in Experiment (oben) und Simulation (unten) für $p^{27}Al$ (gepunktet, nur in der Simulation), $p^{27}Al + p^{12}C$ (gestrichelt) und $pp + p^{27}Al + p^{12}C$ (durchgezogen).

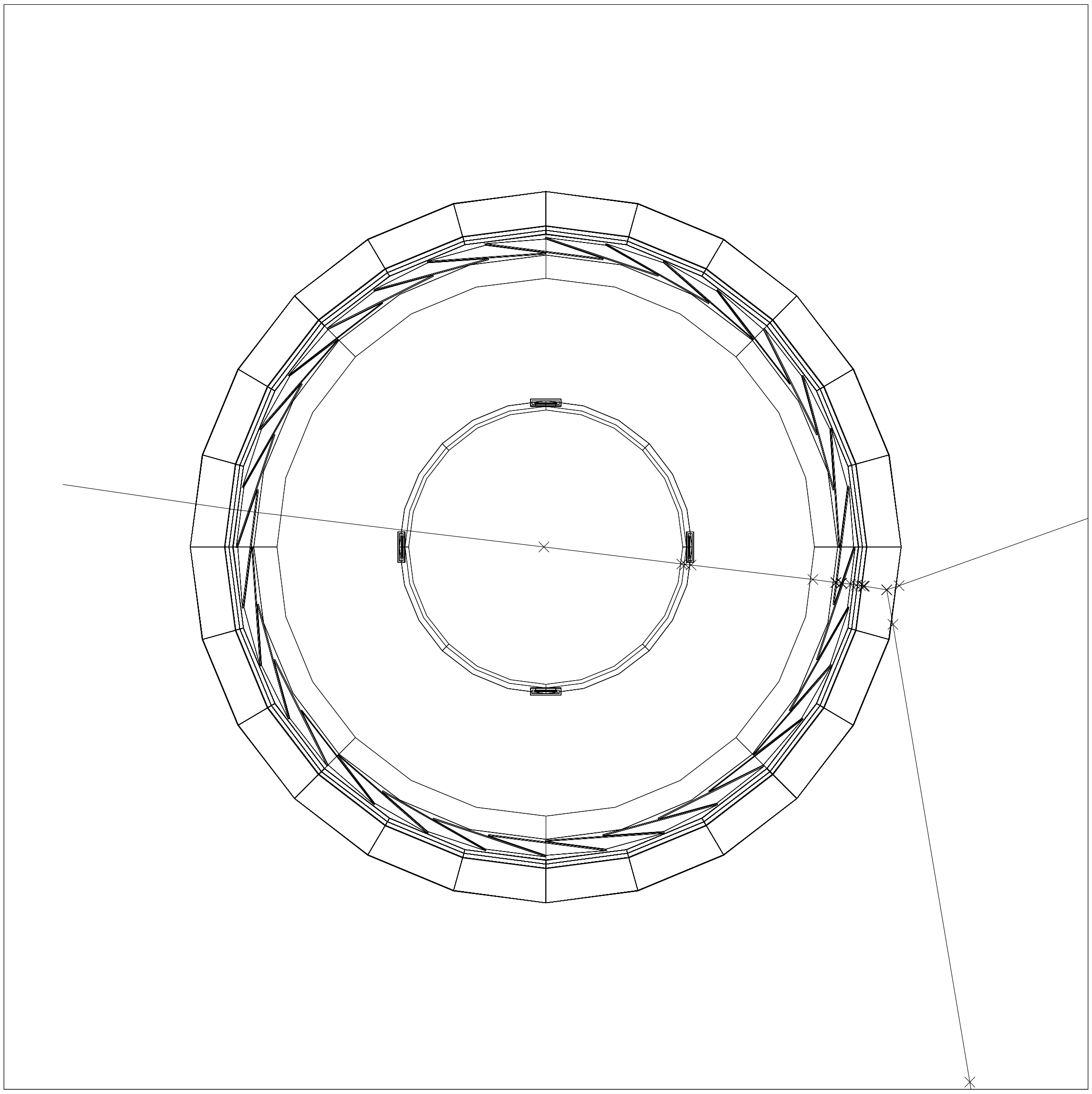

Abbildung 5.3: Blick in den EDDA-Detektor. Zu sehen ist ein simuliertes elastisches pp-Ereignis im Target. Eines der beiden Protonen stößt in der Ringlage mit einem Kohlenstoffkern: $^{12}C(p,pp)^{11}B$.

die von R. Groß-Hardt [Gro 93] begonnenen Untersuchungen zu Triggereffizienz und Detektorakzeptanz vervollständigt werden, und zwar mit der kompletten Geometrie und im gesamten Energiebereich des EDDA-Experiments. Ferner können Fragen in Zusammenhang mit den Helixlagen angegangen werden. Hierzu zählen insbesondere vergleichende Simulationen zum Energieverlust von Elektronen im Testaufbau mit einer ^{90}Sr-Quelle [Jes 95] und Protonen bei Messungen an COSY. Weiterhin zählen dazu Tests von Verfahren zur Vertex- und θ-Rekonstruktion.

Daneben soll auch an der Verbesserung des Simulationsprogrammes gearbeitet werden. Neben der Klärung der in Kap. 5.3 aufgeworfenen Fragen, wären weitere Target-Event-Generatoren — z.B. eine Alternative zu MICRES für leichte Kerne und eine Alternative zu HADRIN, bei der die Spins der streuenden Protonen berücksichtigt werden — wünschenswert. Mit der Steigerung der Benutzerfreundlichkeit wurde schon begonnen. Auch programmierunerfahrene sind nun nach kurzer Anlernphase in der Lage, mit Hilfe eines Texteditors Dateien mit den nötigen Informationen über eine spezielle Simulation — Geometrie, Event-Generator, Energie des Projektils etc. — anzufertigen. Zur Kontrolle können Drahtgittermodelle von Geometrie und simulierten Ereignissen auf dem Bildschirm oder einem Drucker ausgegeben werden (vgl. Abb.5.3). Allerdings erscheinen z.Zt. Kreise noch als Polygone. Auch ein *hidden-line*-Modus — d.h., verdeckt liegende Linien werden nicht gezeichnet — wäre eine hilfreiche Erweiterung.

Anhang A

Eine kurze beispielorientierte Einführung in OOP[1]

In Ergänzung zu Kap. 3.2 sollen hier mit Hilfe kurzer Programmfragmente in C++ die wichtigsten Elemente der OOP erklärt werden. Die Beispiele sind knapp und einfach gehalten und sollten auch demjenigen, der nicht C++ [Str 92] aber irgendeine andere Programmiersprache beherrscht, verständlich sein. Die Beispiele können nur einen kleinen Einblick in das geben, was prinzipiell mit OOP möglich ist, ihre wahren Vorzüge offenbart OOP erst bei umfangreichen Programmieraufgaben.

Beispiel 1:

```
    .
    .
    .

class Particle
    {
        float m;
        float p;

        float kineticEnergy();
    };

float Particle::kineticEnergy()
    {
        return (p * p) / (2 * m);
    }

    .
    .
    .
```

Beispiel 2:

```
.
.
.

float T;

// Definition eines Objekts
Particle proton;

// Zugriff auf Datenelemente
proton.m = 938.27;
proton.p = 10;

// Aufruf einer Methode
T = proton.kineticEnergy();

.
.
.
```

[1] Ausführliche Einführungen in OOP finden sich z.B. in [Boo 94] und in den Sprachbeschreibungen zu *Smalltalk* [Gol 93] und *C++* [Str 92]. Wie die *natürliche* Sicht auf physikalische Objekte durch eine OOP unterstützende Sprache (*objective-C*) eine klare Software-Repräsentation findet, zeigt R. Blankenbecler am Beispiel der Klassifikation von Elementarteilchen [Bla 90]. Für FORTRAN-kundige zeigt P.F. Kunz anhand typisch physikalischer Fragestellungen Möglichkeiten und Grenzen objektorientierter Programmierung in FORTRAN [Kun 90].

Die ersten beiden Beispiele sollen die Begriffe *Klasse, Objekt* und *Methode* erläutern. Zunächst betrachten wir in Beispiel 1 die Definition der *Klasse* `Particle`, die ein Teilchen mit den beiden Eigenschaften Masse und Impuls beschreiben soll. Die *Klassen-Methode* `kineticEnergy()` berechnet die klassische kinetische Energie von Teilchen. Beispiel 2 zeigt die Definition des *Objekts* `proton` der *Klasse* `Particle`. Die Definition hat das selbe Format wie Variablen-Definitionen von Standard-Typen wie z.B. `float`. Beispiel 2 zeigt auch den Zugriff auf die *Objektelemente*, der stets das folgende Format hat:

- `Objekt.Elementfunktion(Variablenliste)` bzw.
- `Objekt.Datenelement`

In Beispiel 2 ist die Variablenliste leer.

Bei der Definition eines Objekts wird diesem vom Compiler ein eigener Bereich im Speicher reserviert, in den die Datenelemente des Objekts und ein Verweis auf die Lage der Klassenmethoden im Speicher geschrieben werden. Insbesondere werden die Klassenmethoden nur einmal für die gesamte Klasse und nicht für jedes Objekt extra gespeichert. Ist z.B. `elektron` ein weiteres `Particle`-Objekt, dann wird also in den Zeilen

```
T = proton.kineticEnergy();
T = elektron.kineticEnergy();
```

jedesmal die selbe — und nicht die gleiche — Methode

`Particle::kineticEnergy()`

aufgerufen. Soll bei der Definition Speicher nicht nur reserviert, sondern auch initialisiert werden, muß dies in benutzereigenen *Klassen-Konstruktoren* festgelegt werden. Konstruktoren haben stets den selben Namen wie die Klasse selbst. Eine Klasse kann mehrere Konstruktoren haben, die sich dann durch ihre Parameterlisten unterscheiden (s.u.).

Ein weiterer zentraler Punkt der OOP ist die *Kapselung* von Elementen. In den obigen Beispielen kann auf Masse und Impuls der Teilchenobjekte von jedem Teil eines Programms aus zugegriffen werden. Da sich aber die Teilchenmasse nicht ändert, ist es sicher-

Beispiel 3:

```
    .
    .
    .

class Particle
    {
  private:
        float m;

  public:
        float p;

        Particle(float _m, float _p);
        float getMass();
        float kineticEnergy();
    };

Particle::Particle(float _m, float _p)
    {
        m = _m;
        p = _p;
    }

float Particle::getMass()
    {
        return m;
    }

float Particle::kineticEnergy()
    {
        return (p * p) / (2 * m);
    }

    .
    .
    .
```

lich sinnvoll, sie vor dem Überschreiben zu schützen. Beispiel 3 zeigt, wie Beispiel 1 entsprechend verändert werden muß. Die Masse ist jetzt als `private` erklärt. Das schützt sie vor dem Überschreiben und dem Lesen außerhalb der Klasse. Sie ist *gekapselt*. An den Wert des Elements `m` kommt man nun nur durch die als `public` deklarierte Methode `getMass()`. Gesetzt werden kann der Wert nur noch bei ihrer Konstruktion durch den neu definierten Konstruktor `Particle(float m, float p)`. Bei der Definition, die nun für ein Proton mit z.B. $p = 10 MeV/c$

```
Particle proton(938.27,10);
```

lautet, wird auch ein Wert an die Variable `p` übergeben, dieser kann aber wie bisher überschrieben werden. Eine typische Vorgehensweise wäre beispielsweise gewesen, einen zweiten Konstruktor `Particle::Particle(float _m)` zu definieren, bei dem `p = 0` gesetzt wird.

Zum Schluß sollen die Begriffe *Ableitung* und *Vererbung* erläutert werden.

Beispiel 4:

```
    .
    .
    .

class RelativisticParticle : Particle
    {
        RelativisticParticle(float m, float p);
        float kineticEnergy();
    };

RelativisticParticle::RelativisticParticle(float m, float p) : Particle(m, p)
    {
    }

float RelativisticParticle::kineticEnergy()
    {
        return sqrt( m*m + p*p ) - m ;
    }

    .
    .
    .
```

Beispiel 4 zeigt die Definition der Klasse `RelativisticParticle` durch *Ableitung* von der Klasse `Particle` [2]. `RelativisticParticle` soll die selben Elemente wie ihre *Basisklasse* enthalten, nur die Berechnung der kinetischen Energie soll nach dem relativistischen Gesetz erfolgen. Daher muß die entsprechende Methode neu geschrieben werden. Die Definition des Konstruktors kann ebenfalls durch Ableitung erfolgen. Wird nun durch

```
RelativisticParticle proton_2(938.27,1800);
```

ein Objekt `proton_2` definiert, so ist z.B. der Aufruf

```
mass = proton_2.getMass();
```

genauso wie für ein `Particle`-Objekt erlaubt, obwohl getMass() für `RelativisticParticle`-Objekte nicht explizit erklärt wurde. Die Kenntnis der `Particle`-Eigenschaften wurden an die abgeleitete Klasse *vererbt*.

Die Möglichkeiten, die sich durch Ableitung und Vererbung eröffnen sind enorm. Programm-Code kann nicht nur kürzer und übersichtlicher gehalten werden, auch Veränderungen lassen sich einfacher anbringen, ohne daß der alte Code verändert werden muß. D.h., sollten die Neuerungen nicht gut gewesen sein, kann problemlos der status quo wiederhergestellt werden.

[2] Hierbei sind zwei eigentlich unzulässige Vereinfachungen gemacht worden:

1. Es muß `class RelativisticParticle : public Particle` heißen, sonst sind die `Particle`-Eigenschaften konventionsgemäß `private`.
2. In Beispiel 3 hätte die Methode `Particle::kineticEnergy()` als `virtual float` deklariert werden müssen. Nur *virtuelle* Funktionen können so problemlos überschrieben werden.

Literaturverzeichnis

[Arn 92] C. Arnault, *Object Oriented Systems in High Energy Physics*, Proceedings of the International Conference on Computing in High Energy Physics '92, Annecy, France, 21-25 September 1992, CERN 92-07, 21 December 1992.

[Atw 90] W.B. Atwood, R. Blankenbecler, P. Kunz, T. Burnett and K.M. Storr, *Monte Carlo and Detector Simulation in OOP*, SLAC-PUB-5359, October 1990; published in "Proceedings of the Symposium on Detector Research and Development for the Superconducting Super Collider", Fort Worth, Texas, October 15-18, 1990.

[Atw 92] W.B. Atwood, T. Burnett, R. Cailliau, D.R. Myers and K.M. Storr, *Gismo: An Object-Oriented Program for High-Energy Physics Event Simulation and Reconstruction*, Int. Jour. o. Mod. Phys. **C3**, (1992) 459-478.

[Atw 93a] B. Atwood, *The Gismo Lectures*, available via anonymous ftp on URL ftp://ftp.slac.stanford.edu/pub/pfkeb/GismoLectures .

[Atw 93b] W. Atwood, A. Breakstone, D. Britton, T. Burnett, D. Myers and G. Word, *The Gismo Project*, SLAC-PUB-6135, April 1993, Submitted to C++ Report.

[Ber 63] H.W. Bertini, *Low-Energy Intranuclear Cascade Calculation*, Phys. Rev. **131**, 1801 (1963).

[Bie 94] A.F. Bielajew, H. Hirayama, W.R. Nelson, D.W.O. Rogers, *History, overview and recent improvements of EGS4*, SLAC-PUB-6499, June 1994; available via anonymous ftp on URL ftp://nrcnet0.nrc.ca/pub/egs4/lecture_notes .

[Bla 90] R. Blankenbecler, *Object Oriented Programming for Simulation Problems in Physics*, SLAC-PUB-5251, June 1990.

[Boo 94] G. Booch, *Objektorientierte Analyse und Design*, Addison-Wesley, Bonn [u.a] 1994.

[Bur 93] T.H. Burnett, *Gismo: An Object-Oriented Approach to Particle Transport and Detector Modeling*, contribution to MC93, International Conference on Monte Carlo Simulation in High

Energy and Nuclear Physics, Tallahassee, Florida-USA, February 22-26, 1993; available via HepGismo-Home-Page on URL http://squark.phys.washington.edu/people/Burnett/hepgismo/ .

[Bur 94] T. Burnett, *HepGismo – the CLHEP version of SLAC's Gismo*, HepGismo-Home-Page on URL http://squark.phys.washington.edu/people/Burnett/hepgismo/ .

[CER 87] Data Handling Division, *Computer Code GEANT 3*, CERN Program Library DD/EE/84-1 (1987).

[Clo 88] P. Cloth, D. Filges, R.D. Neef, G. Sterzenbach et al., *HERMES, A Monte Carlo Program System for Beam-Materials Interaction Studies*, Jül-2203, May 1988.

[Fil 92] D. Filges, P. Cloth and D.Theis, *Improvements and Perspectives of INC-Models in the Medium- and High-Energy Range*, International Symposium on Nuclear Data Evaluation Methodology, Brookhaven National Laborotory, Upton, NY, USA,12-16 October 1992.

[Gol 93] A. Goldberg and D. Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Bonn [u.a.] 1993.

[Gol 64] M.L. Goldberger and K.M. Watson, *Collision Theory*, John Wiley & Sons, New York 1964.

[Gro 93] R. Groß-Hardt, *Simulation des EDDA-Detektors – erste Ergebnisse*, Diplomarbeit, ISKP Bonn 1993.

[Gro 94] R. Groß-Hardt, private Mitteilung.

[Gut 70] M.P. Guthrie, *EVAP-4: Another Modification of a Code to Calculate Particle Evaporation from Excited Compound Nuclei*, ORNL-TM-3119, September 1970.

[Hän 84] K.Hänßgen and J. Ranft, Nucl. Sci. Eng. **88**, 537 (1984).

[Hän 86] K.Hänßgen and J. Ranft, Comp. Phys. Commun. **39**, 37 (1986).

[Hin 89] F. Hinterberger, *Resonanzexperimente*, CANU-Arbeitstreffen über Polarisationsphysik an COSY, Physikzentrum Bad Honnef, 14./15. Februar 1989, Jül-Spez-496, April 1989.

[Jes 95] M. Jeske, Diplomarbeit in Vorbereitung, ISKP Bonn 1995.

[Kun 90] P.F. Kunz, *Object Oriented Programming*, SLAC-PUB-5241, April 1990; Invited paper given at 8th Conference on Computing in High Energy Physics, Santa Fe, NM, April 9-13, 1990.

[Lah 92] U. Lahr, *EDDA - Ein Detektor zur Messung von Anregungsfunktionen der elastischen Proton-Proton Streuung an COSY*, Doktorarbeit, ISKP-Report Nr. 2/1992, Bonn 1992.

[May 84] T. Mayer-Kuckuk, *Kernphysik*, B.G. Teubner, Stuttgart 1984.

[May 89] T. Mayer-Kuckuk, *Das Kühler-Synchrotron COSY und seine physikalischen Perspektiven*, Rheinisch-Westfälische Akademie der Wissenschaften, Vorträge N 359, Westdeutscher Verlag 1988.

[Nel 85] W.R. Nelson, H. Hirayama, D.W.O. Rogers, *PEGS4 User Manual*, Appendix 3 of SLAC-265.

[Ray 79] L. Ray, *Proton-nucleus total cross sections in the intermediate energy range*, Phys. Rev. **C 20**, 1857 (1979) and references therein.

[Roh 94] H. Rohdjeß, *Study of pp elastic scattering with an internal COSY experiment*, Proceedings of the International Conference on Physics with GeV-Particle Beams, Jülich 22-25 August 1994; available via URL http://merlin.iskp.uni-bonn.de/edda/publ/ .

[Sch 91] F. Schwandt, *Aufbau und Test des Triggersystems für das EDDA-Experiment*, Diplomarbeit, ISKP Bonn 1991.

[Sim 93] A.J. Simon, G. Glass et al., *Absolute pp-elastic cross sections from 492 to 793 MeV*, Phys. Rev. **C 3**, 662 (1993).

[Str 92] B. Stroustrup, *Die Programmiersprache C++*, 2. ueberarb. Auflage, Addison-Wesley, Bonn [u.a.] 1992.

[The 92] D. Theis, *Inelastic Hadron-Nucleus Collisions for General Purpose Detector Simulations at Intermediate Energies*, doctor thesis, ISKP Bonn 1992.

Danksagung

An dieser Stelle möchte ich allen Mitarbeitern danken, die zum Gelingen dieser Arbeit beigetragen haben, insbesondere

Prof. Dr. R. Maschuw für die Möglichkeit, diese Arbeit am Institut für Strahlen- und Kernphysik durchführen zu können,

Prof. Dr. F. Hinterberger, unter dessen Leitung diese Arbeit entstand, für ständige Diskussionsbereitschaft und vielfältige Unterstützung,

Prof. Dr. J. Ernst für zahlreiche Anregungen und Vorschläge,

Dipl. Phys. R. Groß-Hardt, von dem ich viel über Programmierung gelernt habe und dessen Unterstützung in großem Maße zum Gelingen dieser Arbeit beigetragen hat,

Dr. H. Rohdjess, Dipl. Phys. F. Schwandt, Dipl. Phys. O. Diehl für zahlreiche Anregungen und Gespräche,

Dipl. Phys. H.P. Engelhardt, cand. phys. M. Jeske, Dr. W. Wiedmann und Dipl. Phys. R. Ziegler für die freundschaftliche Zusammenarbeit

und den Mitgliedern der Arbeitsgruppen Prof. Dr. Bisplinghoff und Priv. Doz. Dr. Eversheim.

Besonderer Dank gebührt meinen Eltern, die mir das Studium der Physik ermöglichten.

Ich versichere, die vorliegende Arbeit selbstständig angefertigt und alle wesentlichen Hilfsmittel angegeben zu haben.

Bonn, den 30. November 1994

Uwe Roß