Visualisierung der Ergebnisdaten von parallelen FE-Berechnungen auf Workstations

Diplomarbeit vorgelegt von:

Michael Burghardt

Matrikelnummer: 970583

Darmstadt, den 7. August 1995

Inhaltsverzeichnis

1 Vorwort **4**

2 Visualisierung von FEM Ergebnisdaten **5**

2.1 Klassischer FEM Ablauf 5

2.1.1 Präprozessor 5

2.1.2 FE-Berechnung 6

2.1.3 Postprozessor 7

2.2 Moderne Simulationssoftware 8

2.3 FEM auf Parallelrechnern 10

2.3.1 Hardwarebeschreibung 10

2.3.2 FE-Software auf Parallelrechnern 11

2.4 Ausgabedaten – Visualisierung 12

2.4.1 Visualisierung statischer Probleme 13

2.4.2 Visualisierung dynamischer Probleme 14

2.5 AVS 14

2.5.1 AVS-Module 14

2.5.2 AVS Kern System 15

2.5.3 Visualisierung 16

3 Das FE-Programm FLOW **20**

3.1 Prinzip der Moving Boundary Method 20
3.2 Parallelisierungsansatz, dynamische Lastverteilung 23
3.2.1 Berechnungsablauf 24
3.2.2 Zu berechnendes Gebiet 24
3.2.3 Message-Passing-Interface 26
3.2.4 Netzaufteilungsalgorithmus 27
3.3 Speicherung der Elementgeometrie in FLOW 29

4 PORTAL **31**

4.1 Allgemeine Beschreibung 31
4.2 Aufbau des Senders 33
4.2.1 Paket-Strukturen von PORTAL 33
4.2.2 Initialisieren des Sockets 34
4.2.3 Speicherplatz 34
4.2.4 Starten des AVS-Moduls 36
4.2.5 Senden der Daten 37
4.2.6 Beenden der Übertragung 37
4.3 Aufbau des Empfängers 38
4.4 Verwendung von UCD in AVS 40
4.5 Datenstruktur zum Übertragen von UCD 41
4.6 Implementierung der Datenstruktur in PORTAL 44
4.7 Verwenden von PORTAL mit UCD in AVS 46
4.8 Implementierung von PORTAL in FLOW 47
4.8.1 Erstellen der darzustellenden Daten 47
4.8.2 Versenden der Daten 49
4.9 Optimierung von PORTAL für die Anbindung von FLOW 50

5 Realisierung des Videofilms **56**

5.1 Allgemeine Einführung 56

5.2 Dienstleistungen des HRZ 57

5.2.1 Digitale Einzelbildaufzeichnung 57

5.2.2 Video-Mitschnitt 59

5.2.3 Nachbereiten, Kopieren, Digitalisieren 60

5.3 Erstellen des Videos 61

5.3.1 Erstellen der Einzelbilder mit AVS 61

5.3.2 Vor- und Nachspann, Zwischentitel 62

5.3.3 Verwenden des Videoservers 63

5.3.4 Nachbearbeitung 65

6 Zusammenfassung und Ausblick **66**

6.1 Zusammenfassung 66

6.2 Ausblick 66

A Erklärung **72**

B Quellcode **73**

Kapitel 1

Vorwort

Bei der Parallelisierung von FE-Programmen am Institut für Numerische Methoden und Informatik im Bauwesen der Technischen Hochschule Darmstadt ergab sich die Notwendigkeit einer Visualisierung der Ergebnisdaten dieser Programme. Während einer zeitabhängigen Simulationsberechnung entstehen große Mengen darzustellender Daten. Die Datenmenge, bei einer Berechnung über mehrere Tausend Rechenschritte, erreicht Ausmaße, die auf Speichermedien nur schlecht zwischengespeichert werden können. Hieraus entstand die Forderung nach einer On-Line-Visualisierung.
Im Rahmen dieser Arbeit wurde die Schnittstelle PORTAL erweitert und in das FE-Programm FLOW implementiert. Eine Optimierung der Schnittstelle war aufgrund der Besonderheit des Parallelrechners notwendig. Des weiteren wurde ein Videofilm produziert, der die Ergebnisse dieser Arbeit darstellt.

Mein Dank gilt besonders Herrn Dr.-Ing. Lutz Lämmer für seine kooperative Betreuung sowie Herrn Robert Kämpf vom Hochschulrechenzentrum für seine Unterstützung bei der Produktion des Videofilms.

Kapitel 2

Visualisierung von FEM Ergebnisdaten

2.1 Klassischer FEM Ablauf

Berechnungen nach dem klassischen Finite-Element-Ansatz gliedern sich in drei aufeinanderfolgende Abschnitte:

- Präprozeß
- Berechnung
- Postprozeß

2.1.1 Präprozessor

Der *Präprozessor* dient der Eingabe des zu berechnenden Körpers oder Gebietes. Dieser sollte eine möglichst einfache Benutzerführung haben. Dies kann durch Pull-down-Menü-Technik, Fenstertechnik oder interaktive graphische Eingabe realisiert sein. Zur Berechnung ist es notwendig, daß das Gebiet oder der

Körper in einzelne finite Elemente zerlegt, d.h. *diskretisiert*, wird. Um dem Nutzer aufwendige Eingabearbeiten abzunehmen, ist hier ein *Netzgenerator* notwendig, der das Gebiet in einzelne Elemente zerlegt [Möller, 1992]. Dieser sollte neben der Aufteilung des Netzes auch in Teilgebieten Verfeinerungen an kritischen Stellen vornehmen können. Eine Kontrolle über die erzeugten Eingabedaten ist zweckmäßig und kann z.B. durch eine graphische Ausgabe geschehen. Da häufig vor der Berechnung Pläne des Gebietes vorhanden sind, die als Eingabe für den Präprozessor dienen und oft vorher mit einem CAD Programm erstellt wurden, ist eine Eingabeschnittstelle sinnvoll. Diese sollte möglichst viele Eingabeformate lesen können.

2.1.2 FE-Berechnung

Der nächste Schritt im Berechnungsablauf ist die Berechnung selber. In ihr werden zuerst die Daten eingelesen, die der Präprozessor erzeugt hat. Die Ergebnisse der FE-Berechnung stehen hinterher in den Ausgabedateien zur Verfügung.
Die Finite-Element-Methode ist ein Näherungsverfahren zur Lösung von *Differentialgleichungen*. Ihr liegt im Gegensatz zum Differenzenverfahren eine integrale Formulierung zugrunde. Die Näherung basiert entweder auf dem Variationsprinzip (Ritzsches Verfahren) oder auf dem Arbeitsprinzip (Prinzip der virtuellen Verrückungen) [Meißner, 1989]. Bei Problemstellungen aus der Dynamik, d.h. bei zeitabhängigen Aufgaben, ist eine Integration der Differentialgleichung in Zeitrichtung durchzuführen.
Der übliche Ablauf einer Berechnung ist folgender:

- Berechnung der Elementmatrix für jedes Element
- Addition der Elementmatrizen zu einer Gesamtmatrix

- Aufbau der rechten Seite aus äußeren Belastungen
- Einarbeiten der Randbedingungen in das Gleichungssystem
- Lösen des Gleichungssystems

Randbedingungen können in der Statik z.B. Lager oder Belastungen aus Kräften sein. Bei Aufgabenstellungen aus der Dynamik, wie etwa der Hydraulik, bestehen die Randbedingungen aus Zuflüssen, vorgegebenen Wasserhöhen oder undurchlässigen Rändern. Randbedingungen ändern sich in einigen Fällen bei zeitabhängigen Problemen während der Laufzeit des Programms.

2.1.3 Postprozessor

Der letzte Abschnitt der Berechnung ist der *Postprozeß*. Er dient der Auswertung der berechneten Daten. Die berechneten Daten bestehen meist aus Vektoren oder Skalaren, die entweder den Knoten oder den Elementen des Gebietes zugeordnet werden. Diese Ergebnisse können in der Statik Verschiebungen oder auch Schnittgrößen wie Momente sein. Der Postprozessor muß aus den Ergebnissen und der den Ergebnissen zugrundeliegenden Elementgeometrie einen konsistenten Datensatz erzeugen. Er sollte die berechneten Daten übersichtlich darstellen können. Hierfür bietet sich eine graphische Ausgabe an. Bei dieser ist es notwendig, einzelne Teile des Gebietes genauer betrachten zu können. Zoom-Funktionen sowie die Möglichkeit der Schnittbildung bei dreidimensionalen Körpern sind hier sinnvoll. Bei statischen Berechnungen sollte neben der Darstellung der Kraft- und Momentenverläufe auch eine Visualisierung des deformierten Systems vorgesehen sein. Einzelne Ergebniswerte der Berechnung, z.B. Extremalwerte, sollten über eine interaktive Eingabe ermittelt werden können. Die Ergebnisse der Berechnung

können normalerweise auf zwei Arten weiterverarbeitet werden. Entweder können sie über eine Schnittstelle an ein anderes Programm, z.B. an ein Bemessungsprogramm, weitergegeben werden, oder sie werden zur manuellen Bemessung auf Papier ausgegeben. Dies sollte in einer frei definierbaren Form geschehen können. Bei der 2D-Simulation von hydraulischen Problemen ist oft eine dreidimensionale Betrachtung der Ergebnissse sinnvoll. Die Ergebnisse, meist Wasserhöhen plus Geschwindigkeiten oder Schadstoffkonzentrationen, sind auf diese Art gut als Höhe plus Farbe gleichzeitig darstellbar. Der Postprozessor muß die Ergebnisdaten mit den 2D-Netzdaten zu einer 3D-Geometrie zusammenfassen. Danach muß er die Umrechnung der 3D-Daten in ein zweidimensionales Rasterbild realisieren, welches dann visualisiert wird. Der Betrachter sollte die Möglichkeit haben, die Ergebnisgeometrie aus beliebigen Sichten zu betrachten. Für einige Anwendungen ist die Möglichkeit der Bildung von Schnitten notwendig. Für diese Visualisierung benötigt man Rechner mit großer Rechenleistung im Grafikbereich, da die Umwandlung der 3D-Geometrien auf 2D-Rasterbilder sowie die Schnittführung sehr aufwendig sind. Wenn das numerische Modell schon aus 3D-Daten besteht, sind die Darstellungen der Ergebnisdaten nur noch als Farben oder als Vektorpfeile in den Knoten darstellbar. Diese Geometrie muß dann ebenfalls in ein 2D-Rasterbild umgewandelt werden, um sie visualisieren zu können.

2.2 Moderne Simulationssoftware

Im Bereich der *Simulationssoftware* ist die klassische Dreiteilung im Bearbeitungsablauf nicht vorhanden. Diese Simulationen stammen meist aus dem Bereich der Dynamik und haben eine Zeitabhängigkeit. Hier werden die Anfangsdaten zuerst wie bei der herkömmlichen FE-Berechnung mit einem Präprozessor erzeugt.

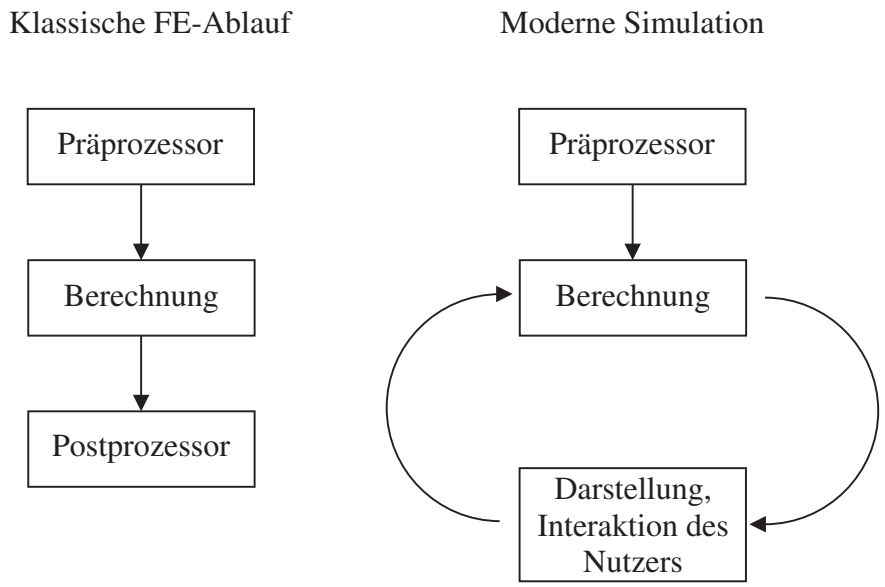

Abbildung 2.1: Berechnungsablauf einer FE-Berechnung

Hieran schließt ebenfalls eine FE-Berechnung an. Dieser folgt dann die erste Auswertung der Daten über eine Visualisierung. Nun kann der Anwender Einfluß auf den Programmablauf durch Manipulation der Daten nehmen. Hieran schließt wieder eine Berechnung an. Der Abschnitt Berechnung und anschließende Darstellung wiederholt sich nun für die zu rechnenden Zeitschritte.

Die wiederholte Berechnung erfordert eine hohe Rechenleistung des Systems, wie sie z.B. Parallelrechner besitzen. Die Darstellung der Daten ist auf Parallelrechnern direkt nicht möglich. Hieraus folgt, daß man für die On-Line-Visualisierung andere Rechner, z.B. eine graphische Workstation, benötigt. Zwischen beiden Rechnern ist dann eine Übertragungsschnittstelle notwendig.

An das Visualisierungsprogramm sind hier besondere Anforderungen zu stellen. Es muß eine Eingabeschnittstelle besitzen, über die man On-Line Daten einlesen kann, die dann automatisch dargestellt werden. Um dem Nutzer ein Eingreifen in den Ablauf der Simultation zu ermöglichen, ist hierfür eine interaktive Kontrolle über das Berechnungsprogramm sinnvoll. Die simultane Darstellung der Ergebnis-

se verlangt vom darstellenden Programm möglichst kurze Visualisierungszeiten, damit das Rechenprogramm nicht auf die Darstellung warten muß.

2.3 FEM auf Parallelrechnern

2.3.1 Hardwarebeschreibung

Nach der Taxonomie von Flyn lassen sich Computer in vier Kategorien einteilen:

- Single Instruction Single Data (SISD)
- Single Instruction Multiple Data (SIMD)
- Multiple Instruction Single Data (MISD)
- Multiple Instruction Multiple Data (MIMD)

Unter *SISD-Rechnern* versteht man Computer, die ihre Aufgaben sequentiell abarbeiten. Sie können nur eine Instruktion gleichzeitig auf einen Datensatz anwenden. SISD-Rechner sind z.B. Personal-Computer oder Workstations.
SIMD-Rechner können im Gegensatz zu SISD-Rechnern gleichzeitig eine Anweisung auf mehrere Datensätze ausführen. Diese Rechner haben mehrere CPUs, auf denen zusammen nur ein Programm abläuft. Dieses Programm wendet simultan die gleichen Anweisungen auf verschiedene Daten an. SIMD Rechner sind z.B. Vektorrechner. Sie können z.B. die Vektoraddition zweier Vektoren parallel addieren. Hierbei addiert jede CPU einen Teil des einen Vektors mit dem zugehörigen Teil des anderen Vektors.
MISD-Rechner sollen mit mehreren Prozessoren einen Datensatz bearbeiten. Dieser Ansatz findet in der Praxis keine Anwendung.

MIMD-Rechner sind im Prinzip mehrere eigenständige Rechner, die miteinander über Links kommunizieren können. Die einzelnen Rechner haben eine eigene CPU sowie einen eigenen Speicher. Je nach Ausstattung der Rechner können für die Kommunikation eigene CPUs vorhanden sein. Eine zentrale Instanz für das gesamte System ist nicht vorgesehen. Die Rechner arbeiten in einem Netzwerk zusammen und sind autark mit einem eigenen Programm beschäftigt [Lämmer, 1993]. Herkömmliche SISD-Rechnern lassen sich über Schnittstellen, wie z.B. Ethernet, zu einem MIMD-Rechner verbinden. Rechner vom Typ MIMD bezeichnet man auch als Parallelrechner.

2.3.2 FE-Software auf Parallelrechnern

FE-Programme lassen sich an vielen Stellen *parallelisieren*. Für die Parallelisierung existieren zwei unterschiedliche Vorgehensweisen. Bei der einen beruht die Parallelisierung darauf, daß den Prozessoren Knoten zugewiesen werden, bei der anderen auf der Verteilung der Elemente auf die Prozessoren. In dieser Abhandlung wird nur auf die zweite Möglichkeit eingegangen. Das Erstellen der Elementmatrizen ist parallel möglich. Jeder Prozessor berechnet hierbei die Elementmatrizen seiner Elemente. Der nächste Schritt in der FE-Berechnung, das Zusammenführen der Elementmatrizen zur Gesamtmatrix, kann ebenfalls parallel vollzogen werden. Jeder Prozessor stellt hierbei den Teil der Gesamtmatrix auf, in dem die Knoten seiner Elementmatrizen stehen. Einige Knoten des Systems sind jeweils auf mehreren Prozessoren vorhanden, da sie zu Elementen gehören, die auf den Rändern der Gebiete der Prozessoren liegen. Für diese Knoten ist eine Kopplung der Werte erforderlich. Die Superposition der Matrix sowie deren globaler Austausch ist als nächstes notwendig. Beide Schritte können gleichzeitig

geschehen. Die hierfür notwendigen Ansätze sind in [Geijn, 1992] beschrieben.
Die Berechnung der rechten Seite des Gleichungssystems kann analog zur Ermittlung der Gesamtmatrix vollzogen werden. Die Prozessoren ermitteln jeweils aus der Belastung im Bereich ihrer Elemente die Resultierenden in den Knoten. Für jeden Knoten wird ein einzelner Vektor mit der Belastung aufgestellt. Diese Vektoren werden auf jedem Prozessor zu einem Gesamtvektor zusammengefügt. Danach ist ein globaler Austausch der Gesamtvektoren sowie deren Superposition notwendig.
Nach dem Einarbeiten der Randbedingungen, d.h. dem Streichen von Zeilen und Spalten aus der Gesamtmatrix und der rechten Seite, muß das entstandene Gleichungssystem gelöst werden. Zur parallelen Lösung von Gleichungssystemen existieren eine Reihe von Ansätzen. So sind das PCG-Verfahren, die Cholesky-Faktorisierung oder auch die Gauß-Elimination parallelisiert worden. Die Iterativen Löser, wie z.B. das PCG-Verfahren, bieten meist einen Geschwindigkeitsvorteil. Genauere Angaben können u.a. [Frommer, 1990] und [Lämmer, 1993] entnommen werden.

2.4 Ausgabedaten – Visualisierung

Zur Visualisierung der Ausgabedaten der FE-Berechnung benötigt man ein Programm, welches die Daten interpretieren und darstellen kann. In einigen Fällen ist die direkte Interpretation der Daten nicht möglich und man benötigt hierfür einen Filter, der die Ausgabedaten des FE-Programms in Eingabedaten des darstellenden Programms umwandelt.
Die FE-Ausgabedaten beinhalten nicht immer die komplette Information über das Gebiet. Manchmal geben z.B. Programme zur Simulation von Strömungs-

vorgängen nur Ergebnisdaten zu den einzelnen Knoten aus. Hier muß dann der Postprozessor die Verknüpfung der Ergebnisdaten mit der Elementgeometrie der Eingabedaten des FE-Programms herstellen.
Ein Visualisierungsprogramm sollte eine Reihe von Optionen bieten. Diese Anforderungen hängen teilweise von der Art der Berechnung ab. In allen Fällen sind die Möglichkeiten notwendig, Randbedingungen darzustellen, Ausschnitte der Geometrie zu vergrößern, Schnitte durch die Geometrie zu legen sowie Beschriftungen anzubringen.

2.4.1 Visualisierung statischer Probleme

Zum Postprozeß im Bereich der ebenen, linearen Statik, wie z.B. nach der Berechnung von Scheiben, Platten, ebenen Fachwerken oder Rahmen, sind Programme ausreichend, die zweidimensionale Darstellungen beherrschen. Hier werden meist Spannungen farblich dargestellt. Die Visualisierung der Verformung und Verzerrungen können durch die Verformung der Elemente bzw. Verschiebung der Knoten dargestellt werden.
Bei der Berechnung räumlicher Tragwerke ist eine 2D-Darstellung meist nicht mehr ausreichend. So sollte man hier auf Programme zurückgreifen, die dreidimensionale Daten visualisieren können. Hierbei ist es notwendig, das Bauwerk aus beliebiger Sicht betrachten zu können. Die Bildung von Schnitten durch das Gebäude ist Voraussetzung für die Auswertung. Dem Nutzer sollte die Möglichkeit geboten werden, einzelne Ergebniswerte, z.B. durch Anklicken mit der Maus, abzufragen.

2.4.2 Visualisierung dynamischer Probleme

Bei der Darstellung von Strömungsvorgängen in der Hydraulik bzw. in der Grundwasserhydraulik handelt es sich um dynamische Vorgänge, die eine wiederholte Berechnung und Darstellung erforderlich machen. Diese Anforderung ist an den Postprozessor zu stellen. Eine weitere Option des Programms sollte die Darstellbarkeit von dreidimensionalen Daten sein. Beliebige Ansichten auf das Gebiet sollten möglich sein, d.h. Rotieren und Verschieben der Geometrie auf dem Bildschirm sind zweckmäßig. Die Schnittbildung durch das Gebiet ist hier genauso wie bei der Darstellung von statischen Problemen notwendig. Eine Verbesserung der Bildqualität kann durch Lichteffekte erreicht werden.

2.5 AVS

AVS ist ein sehr umfangreiches Programm zur graphischen Darstellung von Ergebnisdaten. Es bietet Möglichkeiten zur Visualisierung einzelner Bilder wie auch ganzer Folgen von Bildern. AVS besteht aus zwei Teilen: dem AVS Kern System und den AVS-Modulen.[AVS, 1992]

2.5.1 AVS-Module

AVS-Module sind Unterprogramme oder Funktionen zur Eingabe, Ausgabe oder Manipulation von Daten. Eingaben können von der Tastatur, aus Dateien oder von einem Socket geschehen. Die Ausgabe der Daten kann auf den Bildschirm oder in Dateien vorgenommen werden. Module bilden den Hauptbestandteil des Programms AVS. Über In- und Outputports können die Module untereinander

kommunizieren. Die Datentypen der Ports sind durch AVS vorgegeben. So sind Ports für 1D-, 2D- und 3D-Datenfelder vorgesehen sowie In- und Outputports für spezielle Anwendungen, wie z.B. Elementstrukturen, Unsigned Cell Data (UCD). AVS unterscheidet zwischen zwei Arten von Modulen: *Subroutine Modules* und *Coroutine Modules*. Der Unterschied zwischen den beiden liegt im Startverhalten der Module. *Subroutine Modules* starten ihre Arbeit nur dann, wenn an einer Eingabeschnittstelle Daten ankommen oder wenn sich Parameter durch das *Control Panel* ändern. *Coroutine Modules* starten ihre Arbeit immer dann, wenn sie es selbst erfordern, z.B. wenn sie über einen Socket Informationen bekommen.
Die AVS-Module bilden auch die Programmierschnittstelle von AVS. Eigene Module können während des Programmablaufs eingebunden werden. Hierfür kann man diese in eigenen Bibliotheken zusammenfassen oder in schon bestehende einfügen. Als Programmiersprachen stehen C und FORTRAN zur Verfügung. Bei der Verwendung von In- und Outputports müssen die vorgegebenen Datentypen von AVS eingehalten werden. Dies wird durch AVS-Bibliotheken erleichtert, die beide Programmiersprachen unterstützen.

2.5.2 AVS Kern System

Das AVS Kern System beinhaltet den *Netzwerkeditor*, das *Control Panel* des Editors sowie *Kommunikationsroutinen*. Mit Hilfe des Netzwerkeditors ist es sehr einfach möglich, Module zusammenzufügen. Durch Ziehen mit der Maus lassen sich die Module aus der Modul-Palette auf die Arbeitsfläche kopieren. Zu diesem Zeitpunkt liest AVS die *Description Function* des Moduls. Danach kennt AVS die Ein- und Ausgabeschnittstellen des Moduls sowie deren Datentypen. Der Datentyp einer Schnittstelle wird durch farbige Symbole dargestellt.
Durch Anklicken der Schnittstellensymbole lassen sich diese verbinden. Hierdurch

entstehen Datenkanäle, über die ein Modul Daten an ein anderes Modul schicken kann. Ports lassen sich nur verbinden, wenn sie Daten vom gleichen Typ übertragen können. Ein einzelner In- oder Outputport kann mit mehreren anderen Ports anderer Module kommunizieren.

Das *Control Panel* ermöglicht dem Benutzer die Manipulation des Ablaufs der Module. Mit ihm können Parameter verändert werden, aber auch über einen Browser Dateinamen für Ein- und Ausgabedateien bestimmt werden. Parameter sind Werte, die im Control Panel über Knöpfe oder Regler festgelegt werden und im Modul zur Steuerung des Ablaufs oder zur Manipulation der Daten verwendet werden können.

2.5.3 Visualisierung

Für die Visualisierung von Daten auf dem Bildschirm bietet AVS zwei Möglichkeiten an: den Graph Viewer und den Geometry Viewer. Der *Graph Viewer* ist für die Darstellung von Bitmaps, Graphen oder Kurven bestimmt. Es bestehen hier Möglichkeiten zum Verzerren der Zeichnung, Beschriftung der Achsen sowie der Farbveränderung der Zeichnung.

Zum Visualisieren von Elementstrukturen ist der *Geometry Viewer* besser geeignet. Als Eingabedaten kann er komplette Geometriestrukturen einlesen. Dies sind Knoten, die durch Flächen oder Linien verbunden sind. Der *Geometry Viewer* rechnet diese 3D-Geometrien in eine 2D-Pixelgrafik um und stellt sie auf dem Bildschirm da.

Er bietet dem Anwender eine Fülle von Möglichkeiten, die Geometrie zu betrachten. Der Blickwinkel des Betrachters ist über die Kameraposition bestimmbar. Hierbei kann die Position der Kamera sowie der Punkt, auf den die Kamera zielt, mit räumlichen Koordinaten festgelegt werden. Rotation, Translation und Ska-

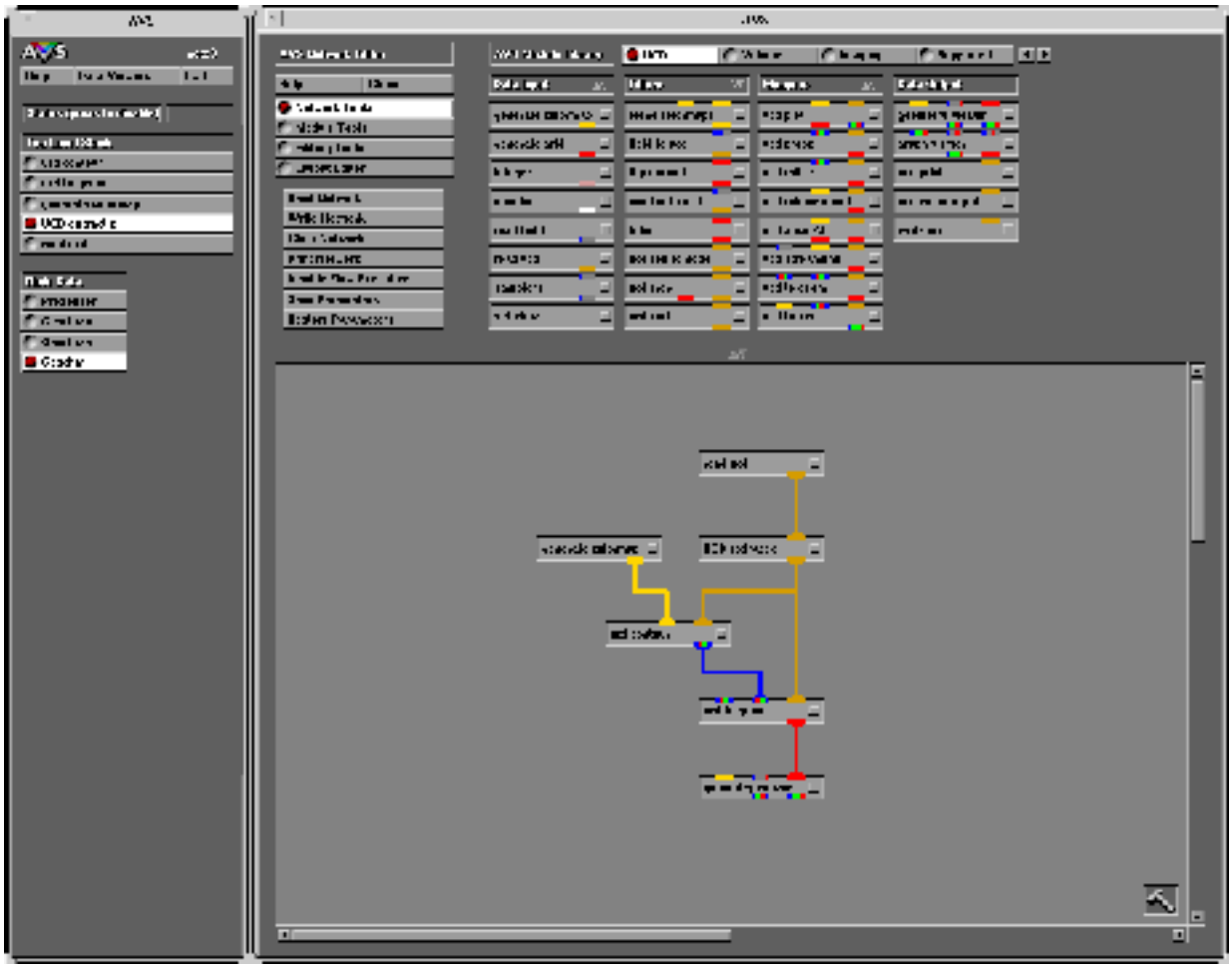

Abbildung 2.2: AVS Bildschirm: links Control Panel, oben Module Library, unten Netzwerkeditor

lierung des Körpers sind durch Eingabe von Zahlenwerten sowie durch Anklicken mit der Maus möglich. Auf die gleiche Art kann die Geometrie auch verzerrt werden.

Die Verbesserung der dreidimensionalen Wirkung kann mit Lichtquellen realisiert werden, da Schatteneffekte eine plastische Darstellung bewirken. Hierfür kann die Position der Lichtquelle wie auch der Punkt, auf den die Lichtquelle strahlt, festgelegt werden. Es besteht die Möglichkeit, jeder einzelnen Lichtquelle auch eine eigene Farbe zuzuordnen. Spiegelungen von einem Körper auf den anderen sind

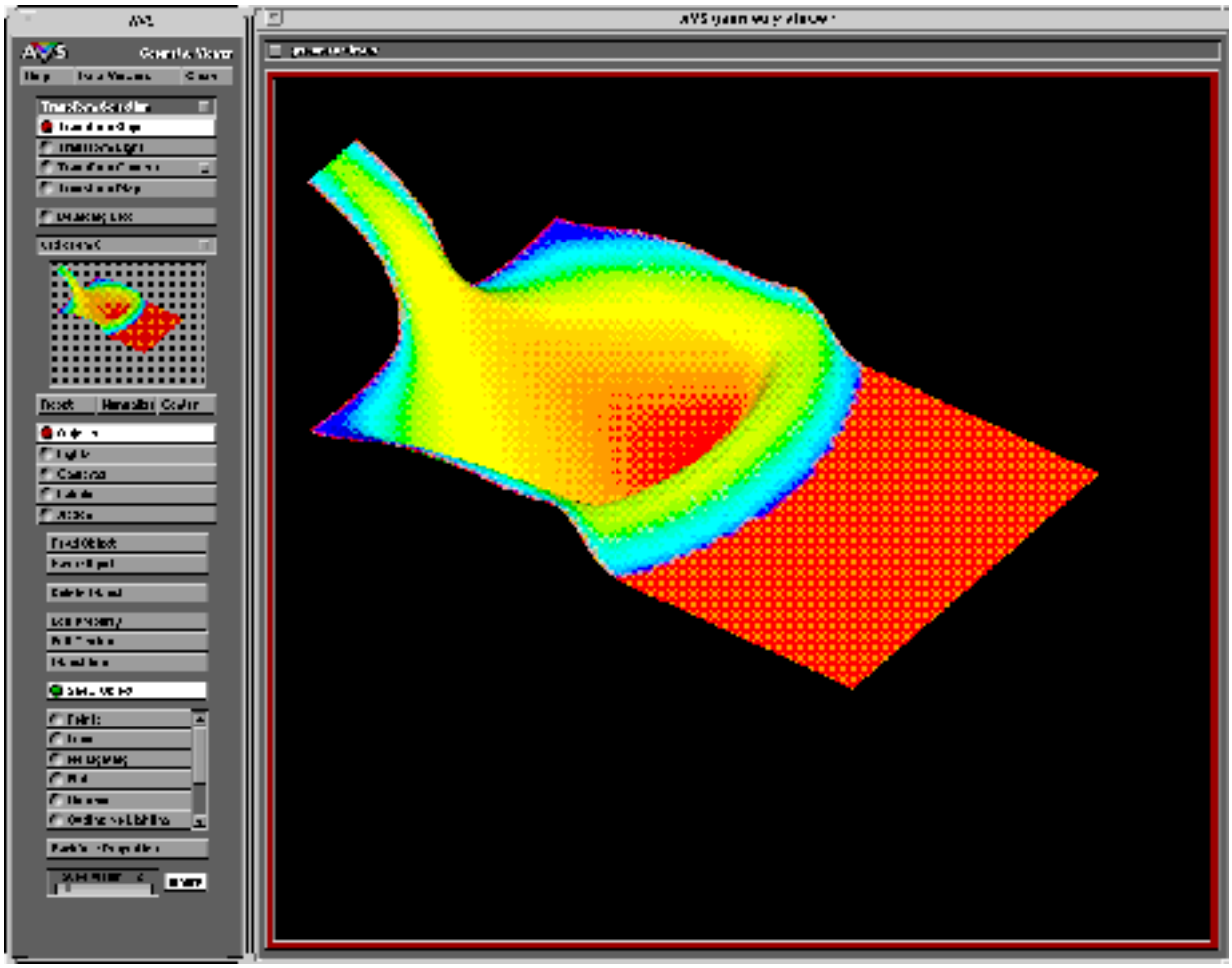

Abbildung 2.3: AVS Geometry Viewer

ebenfalls darstellbar. Das Einstellen einer Hintergrundfarbe sowie die Änderung der Farben der darzustellenden Objekte sind ebenso vorgesehen. Objekten können auch Oberflächeneigenschaften und Materialien zugewiesen werden.

Die Darstellung kann als *Flächenmodell* oder als *Drahtmodell* geschehen. Im Flächenmodell ist das Aufsplitten in einzelne Elemente möglich. Hierbei wird jedes Element in seinen Ausdehnungen etwas verkleinert, wodurch ein Zwischenraum zwischen den Elementen entsteht.

Bei einigen Anwendungen ist es nötig, Schnitte durch die Geometrie zu erzeugen. AVS bietet die Option, Schnittflächen zu generieren, aber auch Ausschnitte in

Form einer Boundary Box zu erstellen.

Häufig ist das Speichern oder Ausdrucken der erzeugten Pixelgrafiken notwendig. Direktes Ausdrucken wird von AVS nicht unterstützt. Da Farbausdrucke meist teuer sind, ist das Erzeugen von Bilddateien sinnvoller. Sie können vor dem Drucken erst einmal mit einem Bildbearbeitungsprogramm, wie z.B. XVIEW, betrachtet und bearbeitet werden. Bilddateien lassen sich auch direkt in Dokumente einbauen.

Ein Grundgedanke bei der Verwendung von Modulen ist die Idee der *Public-Domain*. Programmierern stehen eine große Anzahl von Public-Domain-Modulen[1] zur Verfügung. Eigene Module können so auch anderen Nutzern zugänglich gemacht werden. Zum Erstellen von Bilddateien oder Filmdateien gibt es Module, wie z.B. `WRITE_ANY_IMAGE` . Es kann die 2D-Pixelgrafiken des *Geometry Viewers* in die meisten Bilddateiformate umwandeln. Die unterstützten Formate sind unter anderem: eps, xbm, tiff, gif, ps. Ein Modul zum Erzeugen von Filmdateien ist `Create_MPEG` . Es speichert eine Serie von Bildern nacheinander in ein MPEG-File.

[1]Module können z.B. vom FTP-Server der Universität Heidelberg (ftp.embl-heidelberg.de) aus dem Verzeichnis /pub/avs_mirror/avs_modules bezogen werden.

Kapitel 3

Das FE-Programm FLOW

Als Beispielprogramm zur Visualiserung von FE-Daten auf Parallelrechnern wird das Programm *FLOW* verwendet [ULM, 1994]. FLOW dient zur Simulation von oberflächlichem Abfluß. Es basiert auf der Moving Boundary Method und besitzt eine hierauf basierende *dynamische Lastverteilung.*

3.1 Prinzip der Moving Boundary Method

Bei der Simulation von Wasserströmungen mit der FE-Methode ist es notwendig, die Wasserspiegeländerungen in Zeitschritten zu betrachten. Zu Beginn der Berechnung ist nur ein Teil des Gebiets benetzt. Die Anzahl der benetzten Elemente eines FE-Netzes ändert sich daher während jedes Rechenschrittes. Grundgedanke der *Moving Boundary Method* ist, immer nur die zum betrachteten Zeitpunkt benetzten Elemente in der Berechnung zu berücksichtigen. Trockene Elemente werden nicht betrachtet. Ein Element wird als benetzt definiert, wenn in einem Knoten des Elementes die Summe aus initialer Wasserhöhe H und der Wasserspiegeländerung ζ größer ist als ein vorher definierter Wert ϵ.
Während jedes Rechenschritts wird eine Elementmatrix für jedes Element be-

rechnet. Durch ein *Galerkin Lumping* dieser Matrix erhält man eine Diagonalmatrix [ULM, 1994]. Das hieraus folgende Gleichungssystem kann für jedes Element getrennt berechnet werden. Die Ergebnisse der Berechnung, Wasserhöhen und Geschwindigkeiten in den Knoten, müssen über die zugehörigen Knoten superponiert werden.

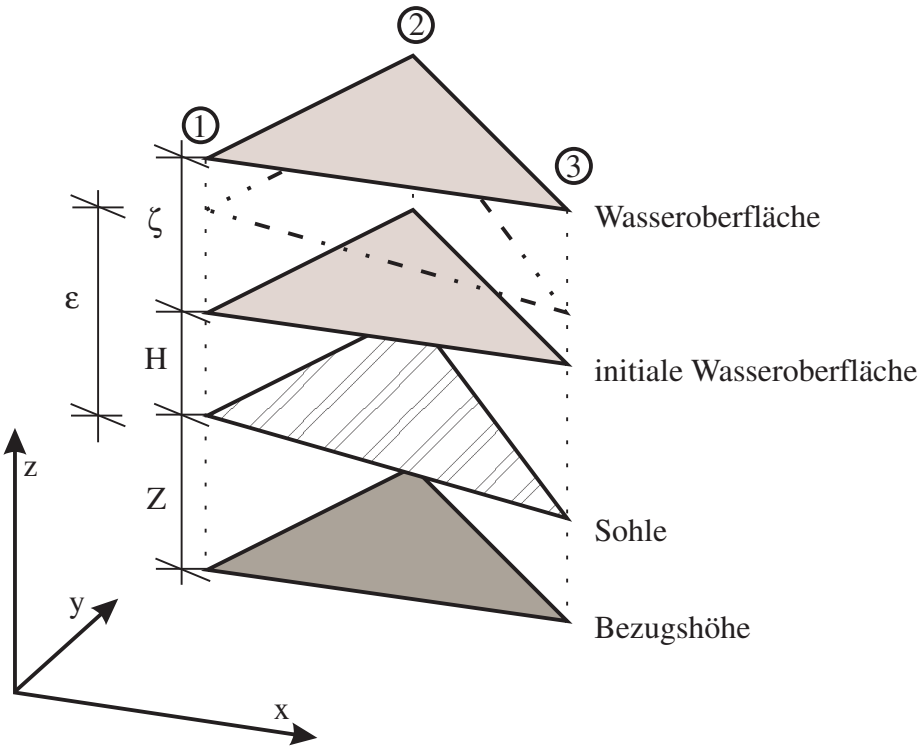

Abbildung 3.1: Definition benetztes Element

Als Ansatz der FE-Berechnung wurden folgende Gleichungen verwendet:

1. Massenbilanz

$$\dot{u}_i + u_j u_{i,j} + g(H + \zeta + Z)_{,i} - A(u_{i,j} + u_{j,i})_{,j} + \frac{g n^2 \sqrt{u_k u_k}}{(H+\zeta)^{\frac{3}{4}}} u_i = 0$$

2. Impulsbilanz

$$\dot{\zeta} + ((H + \zeta)u_i)_{,i} = 0$$

Hierbei bedeutet:

u_i:	Geschwindigkeit
ζ:	Wasserhöhe
H:	Anfangswasserhöhe
Z:	z-Koordinate der Sohle
g:	Erdbeschleunigung
A:	Viskosität
n:	Reibungsbeiwert

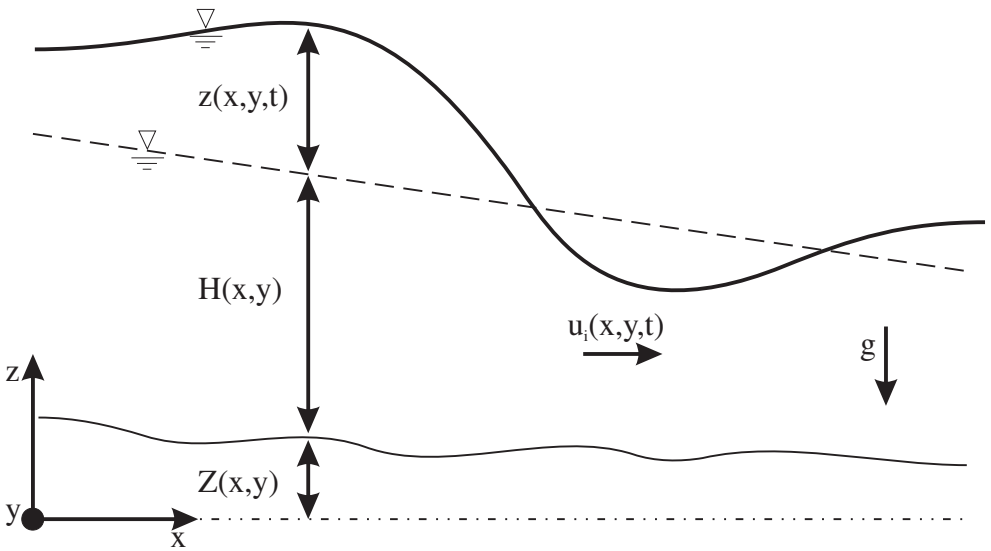

Abbildung 3.2: Erläuterung zu den Gleichungen

Die Gleichungen werden nach der Standard-Galerkin-Methode diskretisiert. Hier-

bei werden als Ansatzfunktion lineare Dreiecke gewählt. Das Programm verwendet eine *Zwei-Schritt-Methode* zur Lösung der Gleichungen. Der erste Rechenschritt kann wie folgt beschrieben werden:

$$\bar{M}u_i^{n+\frac{1}{2}} = \bar{M}u_i^n + \tfrac{1}{2}\Delta t K_1(u_i^n, \zeta^n)$$
$$\bar{M}\zeta_i^{n+\frac{1}{2}} = \bar{M}\zeta_i^n + \tfrac{1}{2}\Delta t K_2(u_i^n, \zeta^n)$$

Der zweite Rechenschritt:

$$\bar{M}u_i^{n+1} = \bar{M}u_i^n + \tfrac{1}{2}\Delta t K_1(u_i^{n+\frac{1}{2}}, \zeta^{n+\frac{1}{2}})$$
$$\bar{M}\zeta_i^{n+1} = \bar{M}\zeta_i^n + \tfrac{1}{2}\Delta t K_2(u_i^{n+\frac{1}{2}}, \zeta^{n\frac{1}{2}})$$

3.2 Parallelisierungsansatz, dynamische Lastverteilung

Der Parallelisierungsansatz beruht auf der *Gebietszerlegungsmethode*. Hierbei wird jedem Prozessor ein Teil der Elemente zugeordnet. Benetzte Elemente sowie ein Teil der trockenen Elemente werden bei der Berechnung berücksichtigt. Dadurch ist eine Umverteilung der Elemente häufig notwendig. Trockene Elemente werden in die Berechnung mit einbezogen, wenn sie möglicherweise in den nächsten Rechenschritten benetzt werden.

Für die FE-Berechnung müssen, wie oben erläutert, die Geschwindigkeit sowie die Wasserhöhe während jedes Rechenschrittes superponiert werden. Hierfür ist ein Austausch der Werte für Knoten, die auf mehreren Prozessoren liegen, vorzunehmen. Um diesen Austausch zu minimieren, ist eine Optimierung der Gebietszerlegung erforderlich. Die Gebiete sollten eine minimale Länge der Außenkanten besitzen.

3.2.1 Berechnungsablauf

Der Programmablauf sieht wie folgt aus: Zuerst werden alle Daten, wie z.B. Netzgeometrie und initiale Wasserhöhen, aus Dateien eingelesen. Nun wird das zu berechnende Gebiet in Abhängigkeit von den benetzten Elementen bestimmt. Am Anfang kann die Benetzung aus initaler Wasserhöhe oder aus einer Randbedingung resultieren. Dieses Gebiet wird auf die Prozessoren verteilt. Als nächstes folgt die Berechnung der Unbekannten für die einzelnen Elemente sowie die Superposition der Ergebniswerte. Hierfür wird teilweise ein Austausch zwischen den Prozessoren notwendig. Nach der Berechnung wird nun festgestellt, ob die vorderste Elementreihe des Gebietes benetzt ist. Gegebenenfalls wird das Gebiet neu verteilt. Nun wiederholen sich die Schritte: Berechnung und – wenn nötig – der Austausch.

3.2.2 Zu berechnendes Gebiet

Da während jedes Rechenschrittes die Anzahl der benetzten Elemente bzw. Knoten zunimmt, müssen in jedem Rechenschritt Elemente bzw. Knoten berücksichtigt werden, die zum Zeitpunkt der Verteilung noch gar nicht benetzt sind. Die Vorausberechnung der möglicherweise benetzten Elemente geschieht auf eine einfache Weise: Es wird angenommen, daß sich die maximale Geschwindigkeit v_{max} im Gebiet nicht stark ändert. Um starken Änderungen vorzubeugen, wird die Geschwindigkeit mit einem Sicherheitsbeiwert η von zwei multipliziert. Da eine Verteilung der Elemente auf die Prozessoren eine gewisse Zeit in Anspruch nimmt, wird dies nicht nach jedem Rechenschritt angestrebt, sondern erst nach einer bestimmten Anzahl von Rechenschritten. Wenn man die Fließzeit Δt für diese Rechenschritte mit der mit Sicherheit beaufschlagten maximalen Geschwindigkeit multipliziert, erhält man den maximal zu erreichenden Fließweg s_{max}:

$$s_{max} = v_{max} \cdot \eta \cdot \Delta t$$

Die Anzahl n der zusätzlich zu verteilenden Elementeschichten wird dann aus dem Quotienten der maximalen Fließstrecke und der minimalen Elementlänge gebildet. Die minimale Elementlänge l_{min} läßt sich aus der Geometrie des Netzes einfach berechnen.

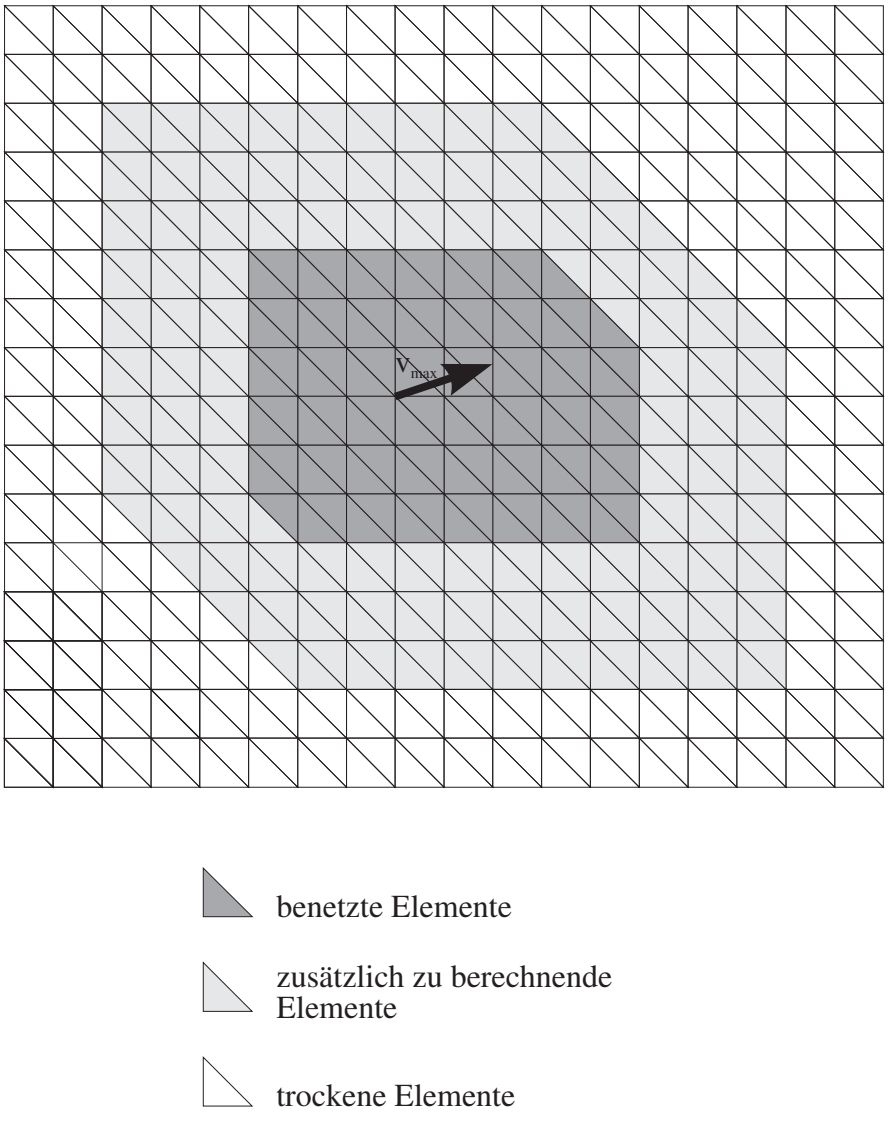

Abbildung 3.3: Zu berechnendes Gebiet

$n = \frac{s_{max}}{l_{min}}$

Aus der Anzahl der zusätzlich zu verteilenden Elemente und der Anzahl der benetzten Elemente ergibt sich nun die Anzahl der Elemente, die zu Beginn verteilt werden müssen. Für diese Anzahl gibt es eine Mindestgröße, die von der Anzahl der Prozessoren abhängt. Diese sollte eingehalten werden.

3.2.3 Message-Passing-Interface

Das *Message-Passing-Interface* wurde bei der Programmierung von FLOW als Kommunikationsbibliothek gewählt. Zweck des MPI ist die Handhabung der Kommunikation der Prozessoren. Grundlegender Mechanismus ist der Nachrichtenaustausch durch Senden und Empfangen [Lämmer, 1993]. Die Prozessoren sind hier eigenständige Rechner mit lokalem Speicher. Über Kommunikationskanäle (Links) tauschen die einzelnen CPUs Daten aus. Das Message-Passing-Interface ist auf vielen Parallelrechnern als Kommunikationsbibliothek vorhanden. So kann FLOW auf einem Parsytek Power X'plorer, einem Parsytek Super-Cluster sowie einem Sun-Workstation-Cluster mit demselben Quellcode übersetzt werden. Eine Hardwareunabhängigkeit wurde hier realisiert.
Charakteristika von MPI sind:

- Unabhängigkeit von der Hardwareplattform
- Effiziente Kommunikation
- Unterstützung heterogener Umgebungen
- Sprachschnittstellen zu C und FORTRAN77
- Sprachunabhängige Semantik der Schnittstelle

Die Optimierung von FLOW wurde für den Parsytek Power X'plorer durchgeführt. Hierbei wurde die Architektur des Rechners ausgenutzt, indem Kommunikation und Berechnung auf den Kommunikations- bzw. Rechenprozessoren gleichzeitig durchgeführt werden. Bei der Berechnung werden aus diesem Grund die Elemente auf jedem Prozessor in zwei Gruppen eingeteilt. Die eine Gruppe besteht aus Elementen, die auf einem Rand eines Gebietes zu einem anderen Prozessor liegen, die andere aus den restlichen Elementen. Die Elemente auf dem Rand werden während jedes Rechenschrittes zuerst berechnet. Danach werden gleichzeitig die inneren Elemente berechnet sowie die Knoten auf den Rändern ausgetauscht.

3.2.4 Netzaufteilungsalgorithmus

Im Programm ist der *Netzaufteilungsalgorithmus* nach C. Farhat [Farhat, 1988] implementiert. Er dient dazu, eine Zuordnung der Elemente auf die Prozessoren zu erreichen. An einen solchen Algorithmus sind folgende Anforderungen zu stellen: Alle Teilgebiete sollen gleich groß sein, damit die Rechenlast auf allen Prozessoren gleich ist. Um einen möglichst geringen Austausch zwischen den Prozessoren zu haben, ist eine minimale Außenlänge der Gebiete notwendig. Eine weitere Anforderung an einen solchen Algorithmus ist, daß nur zusammenhängende Gebiete entstehen. Die Anzahl der Nachbarn eines Gebietes sollte möglichst gering sein.

Während der Aufteilung des Gebietes wird zuerst eine Wichtung der einzelnen Knoten vorgenommen. Diese ist gleich der Anzahl der an einen Knoten angrenzenden Elemente. Danach wird der Knoten mit der geringsten Wichtung gesucht, der auf einer Außengrenze des Gesamtgebietes liegt. Dem Teilgebiet werden nun

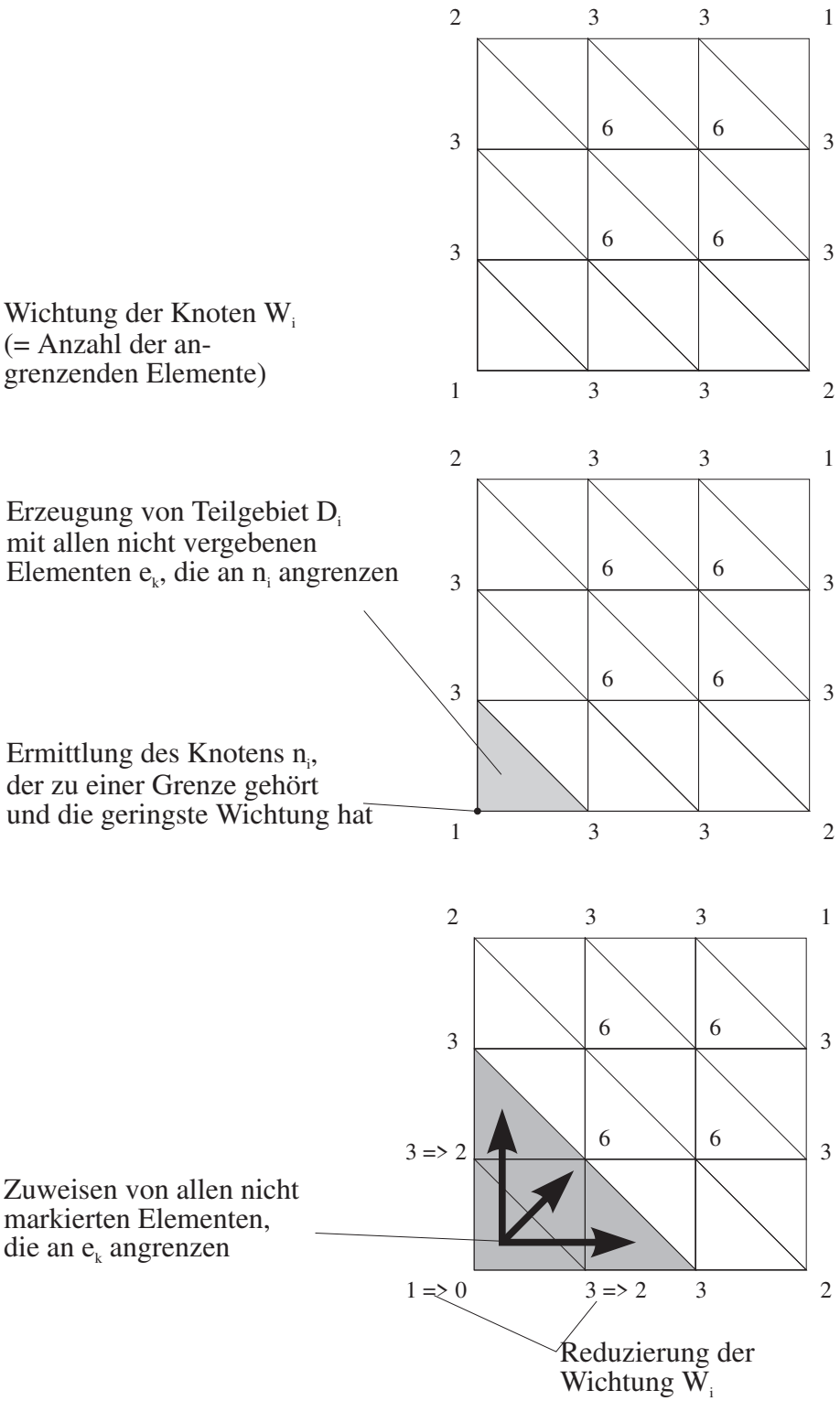

Abbildung 3.4: Netzaufteilungsalgorithmus

alle Elemente zugeordnet, die an diesen Knoten anstoßen. Als nächstes werden alle nicht vergebenen Elemente, die an das Teilgebiet angrenzen, dem Teilgebiet zugeordnet. Die Wichtung der Knoten wird als nächstes reduziert und die den Teilgebieten zugeordneten Elemente werden markiert. Die Vorgänge Zuweisen, Reduzieren und Markieren werden nun solange wiederholt, bis dem Teilgebiet ausreichend viele Elemente zugeordnet wurden. Die Anzahl der Elemente je Teilgebiet ergibt sich aus der Division der Gesamtanzahl durch die Anzahl der gewünschten Teilgebiete. Das Teilgebiet, welches zuletzt ermittelt wird, erhält deshalb – wenn nötig – eine etwas geringere Anzahl. Für das zweite Teilgebiet wird nun wiederum der Knoten mit der geringsten Wichtung gesucht und der Vorgang wiederholt sich.

3.3 Speicherung der Elementgeometrie in FLOW

Zur Visualisierung ist es notwendig, auf die Elementgeometrie von FLOW zurückzugreifen. Die *Datenstruktur* von FLOW zur Speicherung der Elementgeometrie wird in FORTRAN generiert. Die Daten werden in Feldern gespeichert. Es existieren jeweils Felder für die x- und y-Koordinaten der einzelnen Knoten. In einem weiteren Feld befinden sich die zugehörigen Knotennummern. Die Ergebnisdaten des Programms, Wasserhöhe und die Geschwindigkeiten in beiden Koordinatenrichtungen, sind in weiteren Feldern gespeichert. Diese Daten gehören jeweils auch zu den Knoten.

Die Speicherung der Elemente geschieht über vier Felder. In einem der Felder steht die Elementnummer, in den restlichen die zugehörigen Knotennummern.

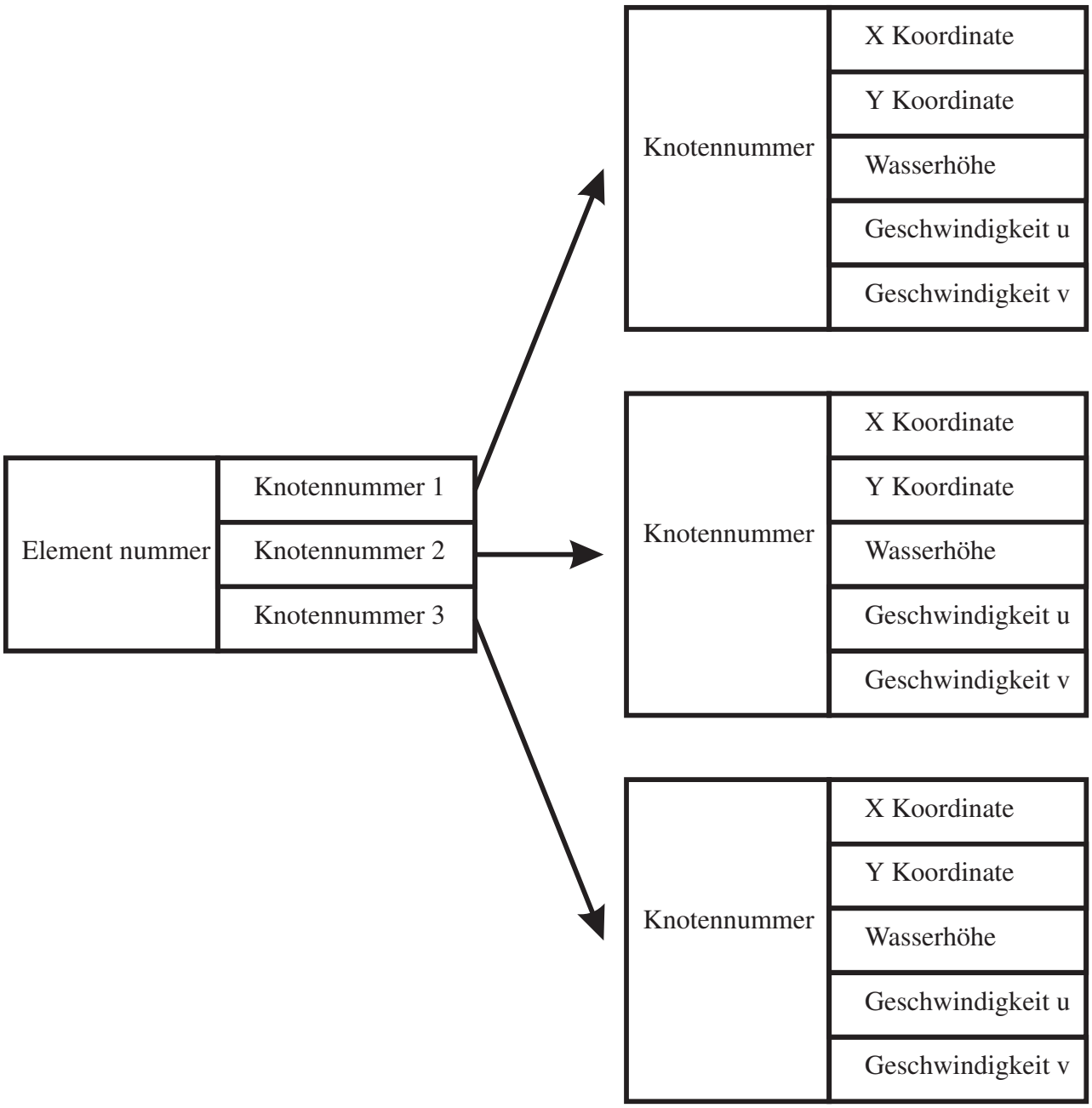

Abbildung 3.5: Datenstruktur von FLOW

Kapitel 4

PORTAL

4.1 Allgemeine Beschreibung

Die Schnittstelle *PORTAL* (Parallel aRchitecture data-TrAnsfer Library) ist zur Darstellung wissenschaftlicher *Simulationen* entwickelt worden [Rowlan, 1993]. Diese Simulationen sind oft zeitabhängig und produzieren während jedes Rechenschritts eine große Menge an Ergebnisdaten. Diese Daten wurden früher auf Datenträgern gespeichert und dann hinterher in einem getrennten Durchlauf dargestellt. Das Programm AVS ist für diese Art der Darstellung konzipiert worden. Das Zwischenspeichern der Daten benötigt nicht zu rechtfertigende Mengen an Speicherplatz auf Festplatten oder ähnlichen Medien. Ein Eingreifen des Nutzers in den Programmablauf ist nur dann möglich, wenn er die Ergebnisdaten während des Programmablaufs interpretieren kann. Dies ist nur bei einer On-Line-Visualisierung möglich. Aus diesen Gründen wurde die Schnittstelle PORTAL entwickelt. Sie ermöglicht die Übertragung der während einer Berechnung anfallenden Daten über einen Internet-Socket zu dem Programm AVS und dadurch die dortige On-Line-Visualisierung.

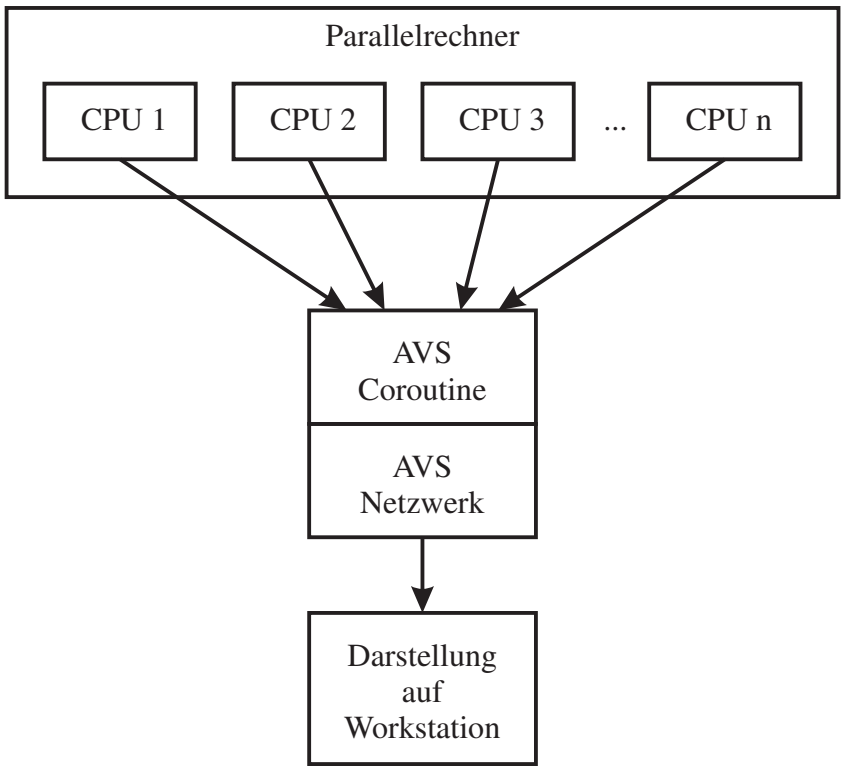

Abbildung 4.1: Aufbau der Bibliothek PORTAL für parallele Aufgaben

PORTAL unterstützt Berechnungsprogramme auf normalen Workstations sowie auf Parallelrechnern. Ergebnisdaten von Berechnungen auf Parallelrechnern liegen normalerweise auf verschiedenen Prozessoren vor. Für die Abstimmung der Daten aufeinander gibt es zwei Möglichkeiten: Sie können entweder im Parallelrechner synchronisiert und dann von einer CPU aus über PORTAL an AVS versandt werden oder die Daten können von mehreren CPUs aus verschickt und dann auf dem empfangenden Rechner synchronisiert werden.

4.2 Aufbau des Senders

Um Daten von einem Programm zu einem anderen über einen Socket zu senden, ist eine Initialisierung des Sockets vom Sender aus nötig. Zur Initialisierung steht die Funktion `DS_init_socket_write()` zur Verfügung. Diese öffnet den Kommunikationskanal zum anderen Rechner und stellt die Verbindung zum empfangenden Programm her.
Als nächstes muß PORTAL Speicherplatz für die zu übertragenden Daten anfordern. Hierfür sind die Funktionen `DS_create_XX_handle()` entwickelt worden, wobei `XX` für den Datentyp steht. Das Visualisierungsprogramm wird nun durch die Funktion `DS_AVSinit()` initialisiert. Während dieser Initialisierung erfährt AVS, welcher Datentyp übertragen wird. Anschließend wird mit dem Unterprogramm `DS_send_data_vectors()` der Datenvektor mit den darzustellenden Daten abgeschickt. Der Aufruf dieses Unterprogramms wird nach jedem Berechnungsschritt des Rechenprogramms wiederholt.
Um die Kommunikation zu beenden, ruft der Sender die Funktion `DS_close_socket_write()` auf. Diese gibt den Socket für ein anderes Programm frei, welches Daten an das empfangende Modul senden möchte.

4.2.1 Paket-Strukturen von PORTAL

Zur Übertragung von Informationen kennt PORTAL drei Arten von Paket-Strukturen: Datenpakete, AVS-Initialisierungspakete und Kontrollpakete. Alle sind in PORTAL als Struktur deklariert.

```
struct PACKET
{
  unsigned long  int tag;              /* Beschreibung der Daten */
```

```
  unsigned long  int length;          /* Groesse in Byte */
  void *         data;                /* Datenvektor*/
};
```

Kontroll-Pakete dienen zum Öffnen, Schließen und Überwachen der Sockets. Sie bestehen aus einem Paket mit einer Kennung, wobei sie nur manchmal Daten übertragen, d.h. die Länge des Datenpaketes ist dann Null. AVS-Initialisierungspakete sind zum Initialisieren der AVS-Module konzipiert worden. Sie beinhalten Informationen zur Konfiguration des Moduls. Das Senden der Daten geschieht mit Datenpaketen. In ihnen steht die Kennung der Daten, die Länge des Datenvektors sowie der Datenvektor selbst.

4.2.2 Initialisieren des Sockets

Zur Initialisierung der Kommunikation über den Socket wird die Funktion `DS_init_socket_write()` verwendet. Diese versucht, den Socket, der vorher vom Empfänger geöffnet wurde, zu finden. Hierfür wird die Funktion `init_socket_write` aufgerufen. Diese kann entweder einen Internet- oder einen UNIX-Socket bearbeiten. Der Sender schickt nun ein Kontrollpaket zum Empfänger, um ihm mitzuteilen, daß die Kommunikation beginnt. Hierfür ist die Funktion `send_hello` zuständig. Diese schickt das erste Kontrollpaket zum Empfänger. Die Kennung ist hierbei `SO_HELO`.

4.2.3 Speicherplatz

Die Funktion `DS_create_XX_handle()` fordert Speicherplatz für die zu übertragenden Daten an. Für jeden Datentyp ist hier eine eigene Datenstruktur vorge-

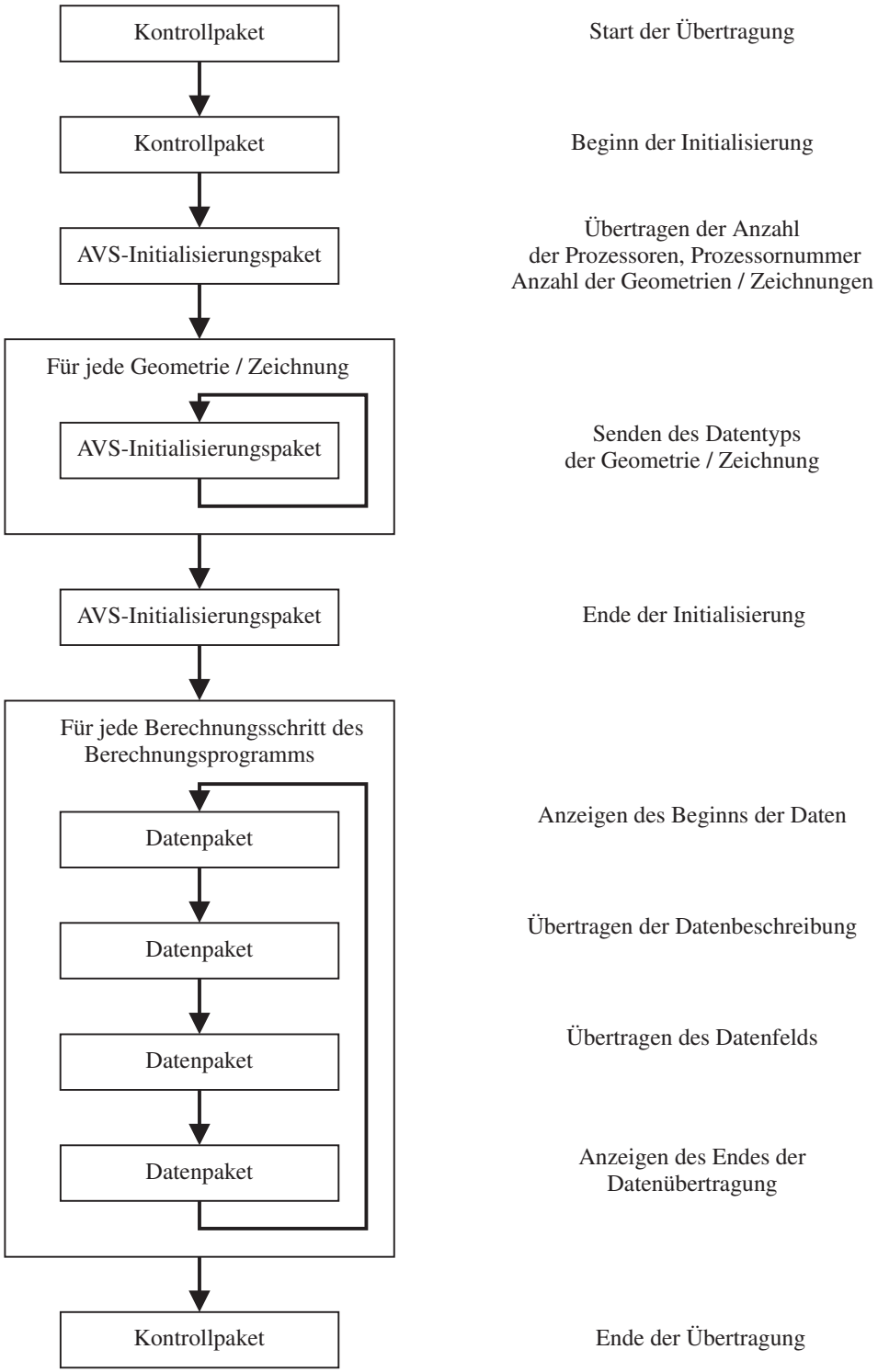

Abbildung 4.2: Reihenfolge der Pakete bei der Übertragung durch PORTAL

sehen. Die Datenstruktur für 2D-Daten ist z.B.:

```
struct DIM_ARRAY_2D_DATA
{
  unsigned long int data_type;          /* Datentyp */
  char              name[16];           /* Name der Daten */
  long int          user_flag;          /* Flag */
  long int          machine_id;         /* Rechnernummer */
  int               x_dim;              /* x Dimension */
  int               y_dim;              /* y Dimension */
  int               x_start;            /* Erste x Position */
  int               y_start;            /* Erste y Position */
  int               x_end;              /* Letzte x Position */
  int               y_end;              /* Letzte y Position */
  int               major_order_1;      /* X_DIM oder Y_DIM */
  void *            data_ptr;           /* Daten */
};
```

In der Datenstruktur stehen die Beschreibung der Daten sowie ein Zeiger auf die Daten. Aus der Beschreibung läßt sich später die Größe des Datenfeldes berechnen.

4.2.4 Starten des AVS-Moduls

Zur Zeit ist AVS die einzige Anbindung von PORTAL an ein Visualisierungsprogramm, die der Öffentlichkeit zugänglich gemacht wurde. In AVS muß ein Modul vor seiner Verwendung initialisiert werden. Dies geschieht mit der Funktion `DS_AVSinit()`. Sie sendet dem Empfänger zuerst ein Kontrollpaket. In dem darauffolgenden Initialisierungspaket stehen Informationen, wieviele Prozessoren

Daten senden, die eigene Prozessornummer sowie die Anzahl der Zeichnungen bzw. Geometrien, die hintereinander verschickt werden. Hierauf folgt für jede Zeichnung bzw. Geometrie das AVS-Initialisierungspaket mit der Beschreibung der Daten. Aus dieser Information kann das AVS-Modul seinen Outputport mit der Kennung des Datentyps generieren. Um dem Empfänger anzuzeigen, daß nun die Konfiguration des Sockets und des Moduls beendet ist, sendet `DS_AVSinit()` zuletzt ein Initialisierungspaket mit der Kennung `AVSINIT_END_TRANS` zum Anzeigen des Endes der Initialisierung des Moduls.

4.2.5 Senden der Daten

Mit der Funktion `DS_send_data_vectors()` werden die Daten zum AVS-Modul übertragen. Sie überprüft zuerst, ob es sich bei den ankommenden Daten um den richtigen Datentyp handelt. Als nächstes wird nun ein Datenpaket mit der Kennung, daß die Datenübertragung beginnt, verschickt. Danach folgt das Versenden der Beschreibung der Daten. Hieraus ermittelt der Empfänger die Größe der zu erwartenden Daten, die danach verschickt werden. Zuletzt wird durch ein Datenpaket mit der Kennung `DS_END_TRANS` dem Empfänger angezeigt, daß die Datenpakete vollständig übertragen wurden.

4.2.6 Beenden der Übertragung

Das Unterprogramm `DS_close_socket_write()` ist ähnlich aufgebaut wie das Unterprogramm `DS_init_socket_write()`. Es hat die Aufgabe, dem Empfänger mitzuteilen, daß alle Daten gesendet wurden. Dies geschieht durch das Senden eines Kontrollpaketes mit der Kennung `SO_GDBY`. Danach wird durch den Aufruf der Funktion `close(*sock)` der Socket von der Senderseite aus geschlossen. Nun ist es jedem anderen Programm möglich, den freigewordenen Socket als Eingabe-

socket in das noch bestehende AVS-Modul zu nutzen.

4.3 Aufbau des Empfängers

Bei Verwendung von AVS als Ausgabeprogramm ist der Empfänger der Daten in der Programmierschnittstelle von AVS integriert, d.h. er ist ein AVS-Modul. Hierbei handelt es sich um ein Coroutine Modul, welches die eigene Ausführung selber starten kann, wenn z.B. Daten am Socket ankommen. Es braucht hierfür keinen Impuls von AVS wie das Drücken einer Taste oder das Empfangen von Daten durch einen Inputport.

Nachdem das Modul aus der Modulbibliothek in den Netzwerkeditor kopiert wurde, wird es initialisiert. Hierbei wird der Socket geöffnet und dem Benutzer dessen Nummer angezeigt. Danach gelangt es in eine Endlosschleife, in der es ständig überprüft, ob Daten am Socket angekommen sind. Hierfür wird die Funktion `is_new_information` verwendet. Der Empfänger wartet nun auf das erste Kontrollpaket. Dieses muß die Kennung `SO_HELO` haben. Hierauf wird ein zweites Kontrollpaket erwartet. Im anschließend empfangenen AVS-Initialisierungspaket steht die Information, wieviele Prozessoren Daten senden, die Prozessornummer des Senders sowie die Anzahl der Zeichnungen bzw. Geometrien, die hintereinander empfangen werden sollen. Danach wird für jedes Datenpaket ein AVS-Initialisierungspaket empfangen. In ihm steht die Datenbeschreibung. Das Ende der Initialisierung wird durch den Empfang eines Paketes mit der Kennung `AVSINIT_END_TRANS` bestätigt. In diesem Moment erhalten die Outputports des Moduls ihre farbige Signatur.

In einer Schleife wird nun beim Empfang eines Datenpaketes immer zuerst die Kennung des Paketes überprüft. Anhand dieser Kennung kann das Modul ent-

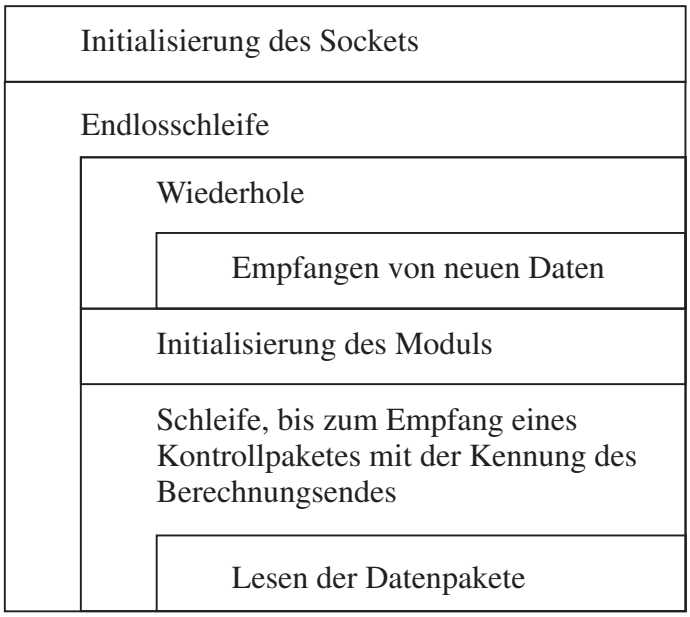

Abbildung 4.3: Vereinfachter Aufbau des Empfängers

scheiden, wie weiter zu verfahren ist.

Handelt es sich bei der Kennung um die Kennung für den Beginn der Datenübertragung, liest das Modul zuerst die Beschreibung der Daten und anschließend solange Daten aus dem Socket, bis ein Datenpaket mit der Kennung für das Ende der Daten kommt. Dies geschieht alles analog zum oben beschriebenen Versenden der Pakete. Bei der Verwendung mehrerer Sender wiederholt sich jeweils das Empfangen der Pakete. Die Daten werden in ein Format, welches dem des Outputports entspricht, umgesetzt. Dies geschieht meist durch Kopieren der Felder. Die umgewandelten Daten werden dann über den Outputport des Moduls an das nächste Modul weitergegeben.

Nachdem das Übertragen der Datenpakete beendet ist, kann das nächste Paket vom Socket zwei verschiedene Kennungen besitzen: Entweder die Kennung für Daten, wobei sich der oben beschriebene Vorgang des Kopierens wiederholt, oder die Kennung für das Ende der Übertragung. In diesem Fall kehrt das Modul in

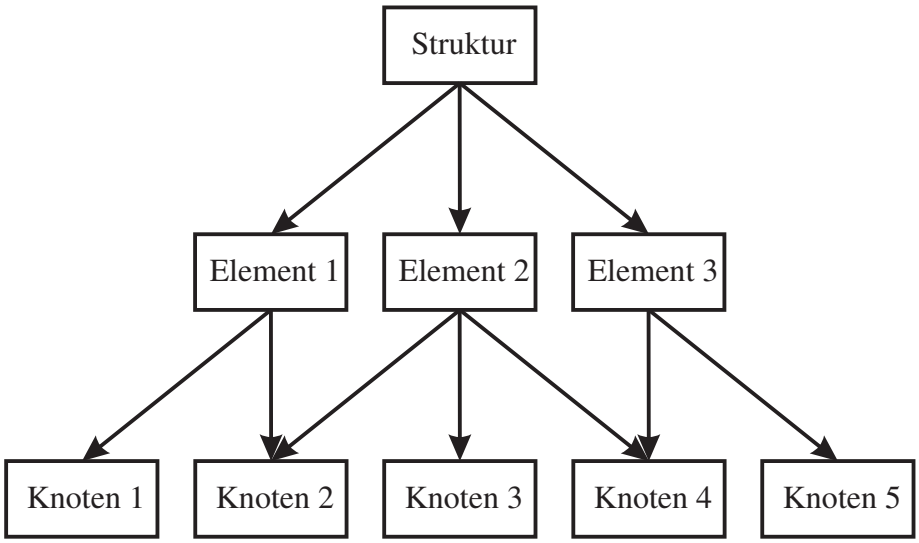

Abbildung 4.4: Datenstruktur von UCD

die äußere Endlosschleife zurück und wartet so lange, bis ein anderes Programm den Socket eröffnet, um Daten zu senden.

4.4 Verwendung von UCD in AVS

Zum Darstellen von Elementgeometrien, wie man sie bei der FE-Berechnung benötigt, bietet AVS den Datentyp *Unstructured Cell Data* (UCD). Jede Elementgeometrie besteht aus vielen Elementen, jedes Element aus mehreren Knoten. Die Datenstruktur hierfür ist hierarchisch aufgebaut. Die Geometrie hat Zeiger auf ihre Elemente. Die Elemente zeigen auf ihre Knoten. Mehrere Elemente können hierbei auf den selben Knoten zeigen.

AVS kann verschiedene Elemente bearbeiten. Das einfachste Element ist der Punkt. Eine Linie besteht aus zwei Knoten, kann aber auch einen zusätzlichen Mittelknoten aufweisen. UCD bietet zwei verschiedene ebene Elemente an: zum einen ein Dreieck, zum anderen ein Viereck. Beide Elemente können optional auch Mittelknoten besitzen. Zur Darstellung räumlicher Elemente bietet AVS vier Ty-

pen an. Es werden Tetraeder, Pyramiden, Prismen und Würfel unterstützt. Auch die räumlichen Elemente können Mittelknoten aufweisen. Diese müssen auf den Kanten liegen. Mittelknoten auf Flächen von 3D-Elementen sind nicht vorgesehen.

Ergebnisdaten der Berechnung können auf drei verschiedene Arten der Struktur zugewiesen werden. Die erste Möglichkeit ist, daß die gesamte Struktur einen Wert zugewiesen bekommt. Die Belastung kann auch für die Elemente angegeben werden. Die dritte Möglichkeit ist das Zuweisen der Ergebniswerte zu den Knoten. Jede Struktur kann mehrere Lastfälle besitzen. Die Auswahl der darzustellenden Ergebnisse wird über Knöpfe des AVS-Kontrollfelds gewählt.

AVS speichert die gesamte Geometrie in einer Struktur ab. Daten werden mit speziellen Funktionen in die Struktur geschrieben. Die genaue Beschreibung der Struktur sowie der Funktionen kann dem Kapitel E von [AVS, 1992] entnommen werden.

4.5 Datenstruktur zum Übertragen von UCD

Für den Aufruf der Funktion `DS_create_UCD_handle()` müssen die gesamten Daten inklusive ihrer Beschreibung in eine Datenstruktur gepackt werden. Hierfür wurde die Struktur `DIM_TYPE_UCD_DATA` entwickelt. In ihr stehen zum einen die Beschreibung des Sockets genauso wie in den Strukturen der anderen `DS_create_XX_handle()` Funktionen. Des weiteren enthält sie die Beschreibung der UCD Geometrie. Sie besteht aus der Anzahl der Knoten, der Elemente, der Lastfälle je Knoten, der Lastfälle je Element, der Lastfälle je Geometrie sowie der Anzahl der Knoten je Element. Als letztes steht in der Struktur noch der Zeiger auf das Datenfeld. Dieser zeigt auf das Datenfeld mit der Knotengeometrie, Elementgeo-

metrie und den Lastfällen.

```
struct DIM_TYPE_UCD_DATA
{
  unsigned long int data_type;
  char              name[16];
  long int          user_flag;
  long int          machine_id;
  int               num_nodes;   /* Anzahl der Knoten */
  int               num_cells;   /* Anzahl der Elemente */
  int               num_ndata;   /* Anzahl der LF je Knoten */
  int               num_cdata;   /* Anzahl der LF je Element*/
  int               num_mdata;   /* Anzahl der LF je Modell*/
  int       num_nodes_element;   /* Anzahl der Knoten je Element */
  void *             data_ptr;   /* Zeiger auf Datenfeld */
};
```

Im *Datenfeld* stehen zuerst alle x-Koordinaten der Knoten. Diese müssen vom Typ Float sein. Als nächstes kommen das Maximum und das Minimum der x-Koordinaten. Hierauf folgen die y- und z-Koordinate mit ihren Extremalwerten. Dann folgt die Beschreibung der Elemente. Die drei Knotennummern eines Elementes müssen hintereinander im Feld stehen. Sie sind vom Typ Integer. Hierauf folgen, wenn vorhanden, die Ergebnisdaten an den Knoten, die wie auch die anderen Ergebnisdaten vom Typ Float sind. Als letztes folgen die Ergebnisdaten in den Elementen bzw. im gesamten Gebiet.
Diese Datenstruktur hat gegenüber den Möglichkeiten von AVS geringe Einschränkungen: Es kann je zu übertragender Geometrie nur jeweils ein Berech-

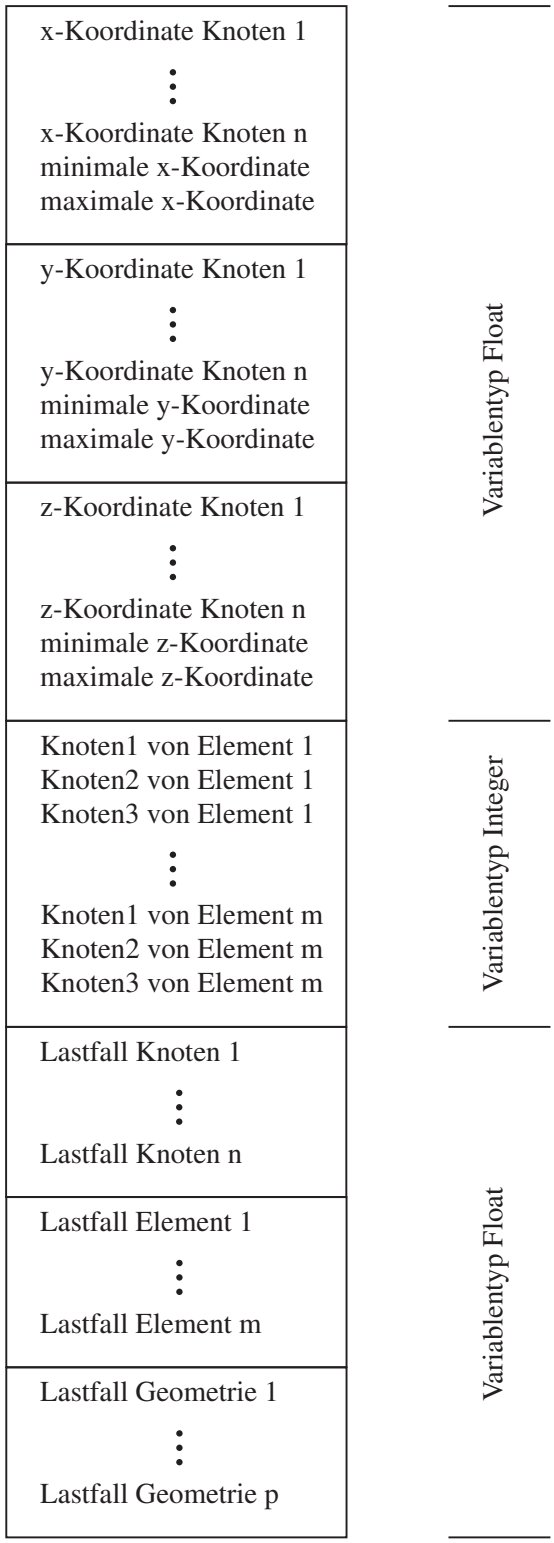

Abbildung 4.5: Aufbau des Datenfeldes

nungsergebnis pro Knoten, Element und Gesamtgeometrie übertragen werden. Diese Einschränkung ist durch die Minimierung der Übertragungszeit zu begründen. Diese wird kleiner, wenn die Menge der Daten, die zu übertragen ist, geringer wird. Hierdurch entsteht für den Nutzer keine Einschränkung. Er kann die Auswahl der Ergebnisse auch durch das sendende Programm steuern, welches dann einen anderen Datensatz senden muß.
Die Datenstruktur ermöglicht es so, daß mit minimaler Datenmenge die Darstellung realisiert wird. Die Datenmenge für ein Beispiel aus FLOW mit 3411 Knoten und 6560 Knoten beträgt inklusive der Beschreibung nur 133 KByte.
Geringere Datenmengen können nur durch eine völlige Änderung des kompletten Aufbaus von PORTAL erreicht werden. Hierfür müßte es eine Zweiteilung im Ablauf der Übertragung geben. Im ersten Teil müßte die komplette Geometrie übertragen werden. In einigen Anwendungen ändert sich während der Berechnung nur ein Teil der Daten. Im hier prototypisch untersuchten Programm FLOW bleiben die x- und y-Koordinaten des Netzes sowie die Zuordnung der Knoten zu den Elementen konstant. Hier könnte man bei allen Schritten nach dem ersten nur noch die sich ändernden z-Koordinaten sowie die FE-Ergebnisdaten übertragen.

4.6 Implementierung der Datenstruktur in PORTAL

Zur Verwendung des Datentyps UCD in PORTAL mußte zuerst eine neue Datenbeschreibung angelegt werden. Die Beschreibungen stehen normalerweise in der Datei `datasock.h`. Die neue Datenbeschreibung für UCD heißt `DS_TYPE_UCD`. Ihr wurde eine noch nicht vergebene Datenspezifikation zugewiesen.

Im oben beschriebenen Ablauf des Senders ist das Anfordern des Speicherplatzes für die Datenstruktur beschrieben. Die Funktion zum Anfordern des Speichers für den Datentyp UCD ist `DS_create_UCD_handle()`. In ihr wird der Speicherplatz für die Struktur `DIM_TYPE_UCD_DATA` bereitgestellt.
Auf der Seite des Empfängers waren umfangreichere Änderungen von PORTAL notwendig. Das AVS-Modul `read_socket` wurde deshalb erweitert. Im Ablauf des Moduls war eine Erweiterung der Konfiguration notwendig. Nach dem Erhalt der AVS-Initialisierungspakete kann das Modul die Initialisierung eines Outputports für Daten vom Typ UCD vornehmen. Als weiteres werden AVS-Parameter für die Knotendaten, die Elementdaten sowie die Modelldaten erzeugt. Diese sind erforderlich für die Darstellung der Ergebnisdaten der FE-Berechnung.
Ankommende Datenpakete werden von der Funktion `copy_DS_UCD_data` weiterverarbeitet. Diese belegt zuerst den Speicherplatz für die Datenstruktur UCD mit dem Aufruf der AVS-Funktion `UCDstructure_alloc`. Danach wird der Datenvektor der übertragenen Struktur in seine Bestandteile zerlegt und in Felder bzw. Variablen kopiert. Diese Felder bzw. Variablen werden anschließend mit speziellen AVS-Routinen in die UCD-Struktur von AVS kopiert.
Die ersten Daten, die in die Struktur kopiert werden, sind die Knotennummern. Hierfür ist die Funktion `UCDnode_set_information` zuständig. Die Knotennumerierung wird nicht über den Socket übertragen, sondern direkt im AVS-Modul erzeugt. Dies geschieht zur Minimierung der zu übertragenden Daten. Zur Beschreibung der Knoten werden nun die Knotenkoordinaten mit dem Unterprogramm `UCDstructure_set_node_positions` an AVS übergeben. AVS benötigt die Extremalwerte der Koordinaten zur Darstellung. Diese werden mit der Funktion `UCDstructure_set_extent` übergeben. Die Elementtopologie wird danach mit dem Unterprogramm `UCDcell_set_information` in die Struktur kopiert. Hierauf folgt noch die Übergabe der Ergebniswerte der FE-Berechnung der einzelnen

Knoten, Elemente bzw. des gesamten Modells. Bei den Knoten ist hierfür die AVS-Funktion `UCDstructure_set_node_components` zuständig. Der kommentierte Quellcode hierzu ist im Anhang B abgedruckt.

4.7 Verwenden von PORTAL mit UCD in AVS

Zur Darstellung von Daten mit AVS benötigt man immer ein *Netzwerk*, in dem die ankommenden Daten verarbeitet und manipuliert werden, um sie dann im Graph Viewer oder dem Geometry Viewer betrachten zu können. Zur Darstellung von Daten des Typs UCD, die mit PORTAL von einem Berechnungsprogramm aus übertragen werden, verwenden wir meist das in Abbildung 4.6 dargestellte Netzwerk.

Das Modul `read socket` ist hierbei die oben beschriebene PORTAL-Schnittstelle. Der Outputport des Moduls ist in diesem Fall vom Datentyp UCD. Welche Ergebnisdaten der FE-Berechnung dargestellt werden, wird mit `UCD extract.c` festgelegt. Hier kann die Auswahl zwischen den Ergebnissen für Knoten, für Elemente oder das gesamte Modell geschehen. Das Modul `generate colormap` ermöglicht, die Farbverteilung zu verändern. Dies ist z.B. notwendig, wenn die Zehnerpotenzen der Ergebniswerte stark unterschiedlich sind, aber trotzdem die kleinen und die großen Ergebniswerte in der Darstellung erkennbar sein sollen. Hierfür kann dann eine logarithmische Verteilung der Farben eingestellt werden.

Im AVS-Modul `ucd contour` wird die Verbindung zwischen den UCD-Daten und der Farbverteilung hergestellt. Die Umwandlung der UCD-Daten in die 3D-Eingabedaten des Geometry Viewers geschieht in `ucd to geom`. Mit diesem Modul kann der Anwender festlegen, ob er die Darstellung als Draht- oder Flächenmodell wünscht. Des weiteren kann man hier bei der Verwendung des Flächen-

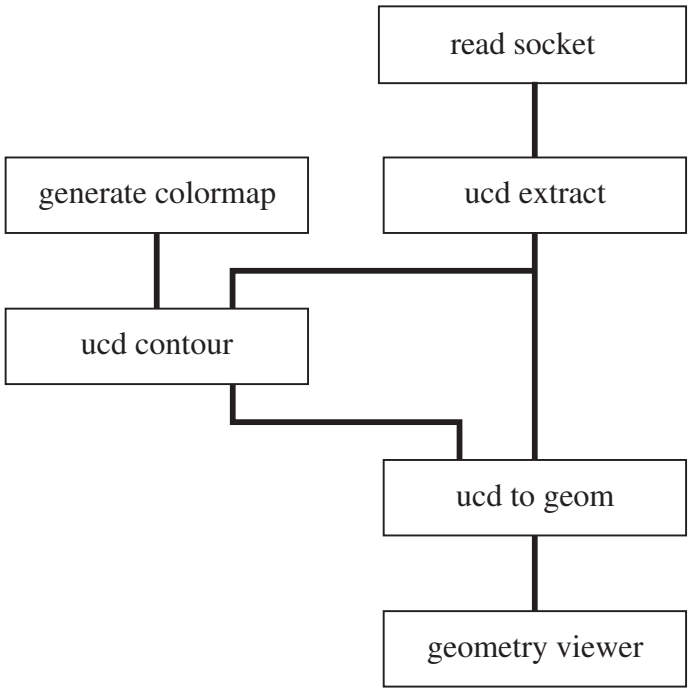

Abbildung 4.6: Netzwerk zum Darstellen von UCD

modells die Aufteilung der Elemente darstellen, indem man die Ausmaße der einzelnen Elemente etwas verkleinert, wodurch dann kleine Zwischenräume zwischen den Elementen entstehen. Die Beschreibung des Moduls Geometry Viewer ist im Kapitel 2.5.3 nachzulesen.

4.8 Implementierung von PORTAL in FLOW

4.8.1 Erstellen der darzustellenden Daten

Das Versenden der Daten mit PORTAL geschieht bei der Implementierung von FLOW von einem Prozessor aus. Dieser schickt folgende Daten an das Visualisierungsprogramm:

- Anzahl der Knoten, Elemente, Geometrien, Lastfälle
- Knotenkoordinaten

- Elementgeometrie
- Ergebnisdaten in den Knoten
- Extremalwerte der Knotenkoordinaten
- Extremalwerte der Ergebnisdaten in den Knoten

Der Teil der Daten, der sich während jedes Rechenschrittes ändert, muß vor jeder Darstellung auf dem Prozessor vollständig vorliegen, der die Daten an AVS weitergibt. Diese Daten sind die Wasserhöhen der Berechnung, die in der Darstellung die z-Koordinaten ergeben sowie die Geschwindigkeiten in den Knoten und die Verteilung der Elemente auf die Prozessoren.
Die z-Koordinate wird vor der Darstellung mit dem Unterprogramm `f_hyper_cube` aktualisiert. Dieses Unterprogramm tauscht die Daten im *Hypercube* aus und addiert die Werte der Knoten. Dieser Austausch ist durch die Aufteilung der Elemente auf die Prozessoren notwendig, da jeder Prozessor nur Wasserhöhen kennt, die zu Knoten gehören, deren Elemente er berechnet.
Für die Darstellung der Geschwindigkeit ist kein zusätzlicher Austausch notwendig. Die Geschwindigkeiten in den Knoten liegen nach jedem Rechenschritt getrennt nach den Koordinatenrichtungen für jeden Knoten vor. Zur Visualisierung der Geschwindigkeit muß nur die Resultierende aus den Komponenten gebildet werden.
Zur Darstellung der Verteilung der Elemente auf die Prozessoren ist ebenfalls kein Austausch notwendig. Auf jedem Prozessor liegt die Information vor, welche CPU welches Element berechnet.
Bei der Visualisierung werden die Extremalwerte der Koordinaten und der darzustellenden Ergebnisdaten benötigt. Die x- und y-Koordinaten ändern sich während des Programmablaufes nicht. Sie können daher zu Beginn der Berechnung einmal

ermittelt werden. Die Ermittlung erfolgt auf dem Prozessor, der die Daten an PORTAL weitergibt.
Die Ermittlung der Minima und Maxima muß für die z-Koordinate sowie für die Geschwindigkeit vor jeder Darstellung durchgeführt werden, da auch diese sich ändern können. Hierfür läuft auf jedem Prozessor eine Schleife über seine Elemente, in der er die Extremwerte für die Knoten seiner Elemente bestimmt. Die Extremwerte müssen dann noch ausgetauscht werden. Hierfür wird wiederum das Unterprogramm `f_hyper_cube` verwendet. Es bietet die Möglichkeit, nicht nur Werte von jedem Prozessor während des Austausches aufzuaddieren, es kann auch den maximalen oder minimalen Wert des auszutauschenden Wertes als Resultat des Austausches liefern.

4.8.2 Versenden der Daten

Zum Übertragen der Daten wurde das Unterprogramm `sende` entwickelt. Es öffnet und schließt den Socket und übernimmt alle Socketkommunikationen.
Zuerst ruft das Unterprogramm die Funktion `DS_init_socket_write` auf, die den Socket initialisiert. Danach kopiert die Funktion die gesamten Daten für die Visualisierung, d.h. die Knotenkoordinaten, die Elementgeometrie sowie die darzustellenden Ergebnisse, in ein Feld. Dies geschieht in der in Abbildung 4.4 angegebenen Reihenfolge. Als nächstes wird das AVS-Modul durch den Aufruf des Unterprogramms `DS_AVSinit()` initialisiert und das Datenfeld mit der Funktion `DS_send_data_vectors` an das AVS-Modul über den Socket geschickt. Als letztes wird der Socket jedesmal wieder geschlossen.
Das Unterprogramm `sende` wurde so entwickelt, daß es auch als *Thread* gestartet werden kann. Ein Prozeß, der als Thread aufgerufen wird, kann parallel zum

Hauptprozeß laufen. Beide Prozesse verwenden hierbei den gleichen Adreßraum. Daraus folgt, daß die Daten für die Visualisierung erst einmal kopiert werden müssen. Würde dies nicht geschehen, würden sich die zu visualisierenden Daten während der Weiterberechnung des Rechenprogramms verändern und die Darstellung wäre nicht konsistent. Die beiden gleichzeitig laufenden Prozesse werden zusammen als *Task* bezeichnet[Ludwig, 1993].

4.9 Optimierung von PORTAL für die Anbindung von FLOW

Für die Implementierung von PORTAL in FLOW wurde PORTAL optimiert. Socket-Schnittstellen sind Schnittstellen zwischen zwei Prozessen. Die Schnittstellen sind nicht genormt, stellen aber einen Industriestandard dar. Dies liegt wahrscheinlich an der leichten Verständlichkeit und der guten Einfügung der Sockets in die UNIX-Umgebung [Santifaller, 1990]. Es sind zwei Arten von Sockets zu unterscheiden: *Internet-Sockets* und *UNIX-Domain-Sockets*.

Ein Internet-Socket ermöglicht den Austausch von Daten zwischen zwei Programmen auf zwei Rechnern, die über das Internet verbunden sind. Die Geschwindigkeit der Kommunikation zweier Prozesse über einen Internet-Socket hängt stark von der Vernetzung ab. Sie ist einerseits bedingt durch die Art des verwendeten Netzwerkes, andererseits durch andere Kommunikationen im Netz. Die Anbindung des Parsytek Power X'plorers an das Internet ist besonders ungünstig, da dieser nur über eine Workstation indirekt erreichbar ist. Deshalb erreicht das Programm PORTAL über einen Internet-Socket vom Power X'plorer aus eine Übertragungsleistung von etwa 20 bis 30 Kilobyte pro Sekunde. Sie ist somit weit von der theoretischen Übertragungsleistung der Links des Parallelrechners

und der möglichen Übertragungsleistung des Internets entfernt.
Aufgrund der genannten Gründe wurde die Kommunikation auf einen UNIX-Domain-Socket umgestellt. Ein UNIX-Domain-Socket kann die Daten nur lokal übertragen. Die Übertragungsleistung eines solchen Sockets ist wesentlich höher als die eines Internet-Sockets. Des weiteren besteht kein Unterschied zwischen beiden Sockets.
Die Übertragungsleistung vom Parallelrechner zur Workstation konnte so auf bis zu 400 Kilobyte pro Sekunde gesteigert werden. Hierdurch wurde die Zeit für den Datenaustausch auf ein Zwanzigstel verringert.
Die Initialisierung des Sockets mußte deshalb auf der Seite des sendenden Prozesses sowie auf der Seite des empfangenden Prozesses überarbeitet werden. Der empfangende Prozeß initialisiert nun den Socket mit dem Unterprogramm `init_socket_read` :

```
void init_socket_read(int *sock, char *pathname)
{
        char            output[256];
        struct sockaddr server;
     /* Erstellen eines Unix-Sockets */
        *sock = socket(AF_UNIX, SOCK_STREAM, 0);
         if (*sock < 0) {
                perror("opening stream socket");
                exit(1);
        }
        server.sa_family = AF_UNIX;
        strcpy(server.sa_data, pathname);
        unlink(pathname);
     /* Zuweisen des Namens */
```

```
        if (bind(*sock, (struct sockaddr *) & server,
                 sizeof(struct sockaddr)) < 0) {
                perror("socket_bind");
                exit(1);
        }
        if (listen(*sock, 5) < 0) {
                perror("listen");
                exit(1);
        }
        sock_dputs("listen succeeded");
        sock_dputs(output);
        return;
}
```

Auf der Seite des Senders mußte das Unterprogramm `init_socket_write` geändert werden:

```
void init_socket_write(int *sock, char *pathname)
{
        struct sockaddr server;
     /* Erstellen eines Unix-Sockets */
        *sock = socket(AF_UNIX, SOCK_STREAM, 0);
        server.sa_family = AF_UNIX;
        strcpy(server.sa_data, pathname);
     /* Initialisieren der Verbindung */
        if (connect(*sock, (struct sockaddr *)
                        & server, 110) < 0) {
                perror("connecting stream socket");
```

```
                exit(1);
        }
        return;
}
```

Die maximale Übertragungsrate bei der Datenübertragung über einen UNIX-Socket erreicht man beim Parsytek Power X'plorer, wenn die Größe der einzelnen Datenpakete in einem Bereich von maximal 8 Kilobyte liegt. Das Festlegen der Datenpaketgröße wird vom Sender vorgenommen. Das hierfür verwendete Unterprogramm ist `send_data`. In der Originalversion von PORTAL wurde das Verschicken mit Hilfe der Socket-Kommunikationsroutine `writev` durchgeführt. `writev` verschickt eine Strukur vom Typ `iovec`. Diese ist wie folgt definiert:

```
struct iovec {
    caddr_t    iov_base;    /* Anfangsadresse der Daten */
    int        iov_len;     /* Blocklaenge */
}
```

Zuerst wird hierbei die Größe (`iov_len`) des zur Übertragung anstehenden Datenpaketes an den Empfänger geschickt. Danach werden die Daten selbst in einem Stück gesendet. Eine Geometrie, die als UCD in AVS dargestellt werden soll, kann bei der Übertragung leicht eine Größe von mehr als 100 Kilobyte erreichen. Wenn nun ein Datenpaket dieser Größe auf einmal gesendet wird, erreicht man nur eine sehr geringe Transferrate.
Das Unterprogramm `send_data` wurde deshalb verändert. Als Socket-Kommunikationsroutine wurde `write` verwendet. Diese ermöglicht es relativ einfach, das Versenden der Daten in kleinen Paketen zu realisieren. Eine Schleife wurde entwickelt, in der in jedem Durchlauf nur ein Teil des zu übertragenden Datenfeldes

mit festgelegter Größe des Paketes gesendet wird.

```
      /* Zeiger auf Anfang des Datenfeldes setzen */
ptr = (char *) data->data;
     /* Festlegen der Blocklaenge des aktuellen Blocks */
aktueller_block = blocklaenge;
n_left = data->length;
if (n_left < blocklaenge)
        aktueller_block = n_left;
while ((n_left > 0)) {
        n_write = write(sock, ptr, aktueller_block); /* Senden der Daten */
        if (n_write > 0) {
     /* Zeiger auf naechsten Block setzen, aktuelle
        Blocklaenge bestimmen                              */
                n_left = n_left - n_write;
                ptr = ptr + n_write;
                if (n_left < blocklaenge)
                        aktueller_block = n_left;
        } else if (n_write == 0)
                fail--;
}
```

Durch die beschriebenen Änderungen konnte die Übertragungszeit für eine zu visualisierende Geometrie stark reduziert werden. Für einen Beispieldatensatz mit 3411 Knoten und 6500 Elementen verringerte sich durch die Optimierungen die Übertragungszeit von etwa 12 Sekunden auf nur noch etwa eine halbe Sekunde.

Die Darstellung bremst jetzt den Rechenprozeß nur noch gering. Die Verlängerung der Gesamtrechenzeit durch die Darstellung ist auf ein vertretbares Maß reduziert worden.

Kapitel 5

Realisierung des Videofilms

5.1 Allgemeine Einführung

Zur Produktion von Videofilmen stellt das *Hochschulrechenzentrum* (HRZ) der TH Darmstadt allen Instituten der TH sowie allen Instituten anderer hessischer Hochschulen das Videostudio zur Verfügung [Kämpf, 1995]. Grund für die Einrichtung einer *Videoanlage* durch das HRZ ist, daß man bei Vorträgen Simulationen auf Rechnern schlecht auf Overhead-Folien demonstrieren kann. Ein kurzer Videofilm zum Ende eines Vortrages kann hier vieles verdeutlichen, was Folien nicht zeigen können. Ein Vortrag mit der Vorführung eines Filmes bleibt durch seine Besonderheit eher im Gedächtnis des Auditoriums als ein Vortrag, in dem die üblichen Folien gezeigt werden.

5.2 Dienstleistungen des HRZ

Zur Produktion von Videofilmen stellt das HRZ folgende Dienstleistungen zur Verfügung:

- Digitale Einzelbildaufzeichnung
- Mitschnitt von Darstellungen auf Computerbildschirmen
- Video-Nachbereitung
- Video-Kopierservice
- Video-Digitalisierung

5.2.1 Digitale Einzelbildaufzeichnung

Die *digitale Einzelbildaufzeichnung* bietet die Möglichkeit, einzelne vorher erstellte Bilder zu einem Film zusammenzufügen. Man kann hier mit voller Videoauflösung arbeiten. Dies beinhaltet die maximale Auflösung sowie die Anzahl der Bilder pro Sekunde. Zum Zwischenspeichern der Bildsequenzen dient eine *digitale Bildplatte*. Diese kann, genauso wie Video, 25 Bilder pro Sekunde mit einer Auflösung von 720x576 Pixeln speichern. Gespeichert werden hierbei je Bildpunkt ein Byte Helligkeit und je Bildpunkt ein Byte Farbinformation. Die digitale Bildplatte hat deshalb eine Datenrate weit oberhalb von SCSI Festplatten. Sie überträgt zum Videorekorder eine Datenrate von etwa 20 MByte/s [Kämpf, 1995].

Die Einzelbildaufzeichnung kommt einem Drucken von Bildern auf Papier nahe. Der Unterschied ist nur, daß hier nicht auf Papier gedruckt, sondern auf Videoband kopiert wird. Die Automatisierung wurde über einen lpr/lpq *Videoserver*

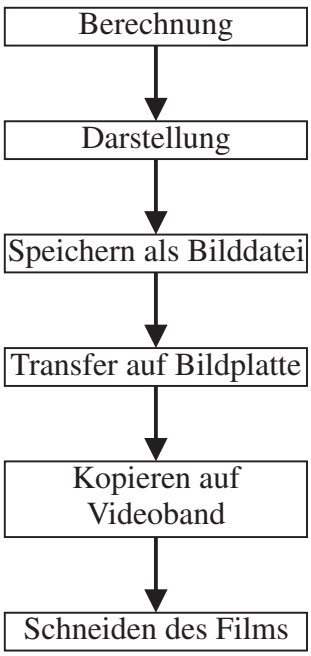

Abbildung 5.1: Ablauf bei der Einzelbildaufzeichnung

realisiert. Mit einem vom HRZ entwickelten Verfahren ist ein automatisches, bildgenaues Kopieren von der Bildplatte auf ein Videoband möglich.
Der Videoserver bietet folgende Funktionen an:

- Transfer der Bilddaten auf Bildplatte
- Übernahme der Bildsequenzen auf Videoband
- Aufzeichnung von Standbildern
- Simulation einer unendlich großen Bildplatte durch vollautomatisches Kopieren auf Videoband
- Zugriff und Steuerung des Servers durch eine Workstation

Zur Erstellung eines Videofilms aus Einzelbildern können drei verschiedene *Bildformate* verwendet werden: GIF, PPM und Postscript. Die beiden Rasterbildformate GIF und PPM können einen beliebige Größe haben. Optimal ist eine Größe

von 720x576 Pixel, da hier keine Skalierung während der Aufzeichnung notwendig ist. Eine Skalierung bringt immer ein Verlust der Bildqualität mit sich. Bei der Erstellung der Einzelbilder ist immer eine Besonderheit von Video zu beachten: Die Bildpunkte auf einem Fernsehbildschirm sind nicht quadratisch. Video hat ein Seitenverhältnis von 4:3 (720:576). Der Bildschirm weist aber ein Verhältnis von 5:4 auf. Aufgrund dieser Gegebenheit ist bei der Darstellung von Kreisen oder anderen geometrischen Körpern, bei denen man später eine Verzerrung vermeiden möchte, vorher eine entgegengesetzte Verzerrung notwendig.
Postscript wurde eigentlich als Programmiersprache für Drucker entwickelt. Sie stellt einen Industriestandard dar und wird heute von den meisten Grafikprogrammen unterstützt, weshalb auch eine Anbindung des Videoservers entwickelt wurde. Da Postscript nicht pixelorientiert ist, ist eine Skalierung der Zeichnungen nicht notwendig. Postscript hat aber auch einen Nachteil: Eine Datei im Postscriptformat kann sehr groß und die Konvertierung in ein Rasterbild sehr aufwendig und langsam sein.

5.2.2 Video-Mitschnitt

Das HRZ stellt einen *Scanconverter* zum Mitschnitt von Darstellungen auf einem Computerbildschirm zur Verfügung. Die Aufzeichnung hat etwa 800x600 Pixel, was dem Format von normalen Videobildern nicht entspricht. Die Darstellung entspricht der PAL-Norm und bietet nur eine Bildwiederholfrequenz von 50Hz, die im Zeilensprungverfahren arbeitet. Ein Flimmern auf Bildschirmen ist hier fast unvermeidbar.
Eine Schwierigkeit bei einem Video-Mitschnitt ist, daß die Aufzeichnung nur im Videostudio möglich ist, d.h. die Software muß auf Rechnern im Studio des HRZ laufen. Ein weiteres Problem ist, daß der Mitschnitt quasi live stattfindet, so daß

man während der Bedienung des Programms keine Fehler machen darf, da diese später auf dem Video zu sehen sind.

5.2.3 Nachbereiten, Kopieren, Digitalisieren

Nachdem ein Film mit einer der oben genannten Methoden aufgezeichnet wurde, ist es meist notwendig, ihn nachzuarbeiten. Das HRZ stellt hierfür folgende Möglichkeiten zur Verfügung:

- Schneiden von Filmen
- Ergänzung durch Unter- und Zwischentitel
- Erzeugen von Vor- und Nachspann
- Vertonung

Die Qualität von Videofilmkopien hängt stark vom technischen Aufwand beim Kopieren ab. Das HRZ bietet daher auch den Service, Videos zu kopieren bzw. Kopien vom Masterband herzustellen. Kopien können in drei verschiedenen Formaten erzeugt werden: BETA, VHS und S-VHS.
Häufig besteht auch ein Interesse, ein Video im World-Wide-Web des Internet zu präsentieren. Hierfür kann im Videostudio jedes beliebige Video zu einem MPEG weiterverarbeitet werden oder direkt aus den Quelldateien ein MPEG-File erzeugt werden.

5.3 Erstellen des Videos

Zum Erstellen des Videos wurde die Methode der Einzelbildaufzeichnung verwendet. Als Bildformat wurde aufgrund der geringeren Größe der Dateien das GIF-Format gewählt. Zuerst mußten einzelne Bilder aus AVS heraus erzeugt werden.

5.3.1 Erstellen der Einzelbilder mit AVS

Zum Erstellen von Einzelbildern mit AVS gibt es zwei Möglichkeiten: entweder mit einem Snapshot-Programm oder mit einem speziellen AVS-Modul.
Mit dem Modul `WRITE_ANY_IMAGE` können diese Bilder erzeugt werden. Das Modul kann über den AVS-Inputport 2D-Pixelgrafiken einlesen. Vom Outputport des *Geometry Viewers* erhält man solche Bitmaps. Das Modul ist komfortabel ausgestattet. So können sehr viele verschiedene Grafikdateiformate geschrieben werden. Die Bedienung ist wie bei den meisten AVS-Modulen durch Window-Technik sehr einfach. Dieser Komfort hat aber auch Nachteile. Die Geschwindigkeit des Moduls läßt zu wünschen übrig. So kann das Erstellen einer GIF-Bilddatei je nach Größe und Anzahl der Farben bis zu 50 Sekunden dauern.
Um mit einem Snapshot-Programm die Bilder für einen Film aufzunehmen, muß man nach der Berechnung jedes Bildes einen Snapshot des Geometry Viewer-Fensters machen. Dieser muß dann manuell abgespeichert werden. Danach ist eine Nachbereitung des Bildes notwendig. Der Geometry Viewer hat innerhalb des Fensters immer einen roten Rahmen um die Zeichnung. Dieser Rahmen muß vor der Weiterverarbeitung von Hand entfernt werden, was mit vertretbarem Aufwand nur bei wenigen Bildern möglich ist und nicht bei der für einen Film erforderlichen großen Anzahl. Wegen dieser Nachteile ist die Verwendung von AVS-Modulen zur Erstellung der Bilder vorzuziehen.

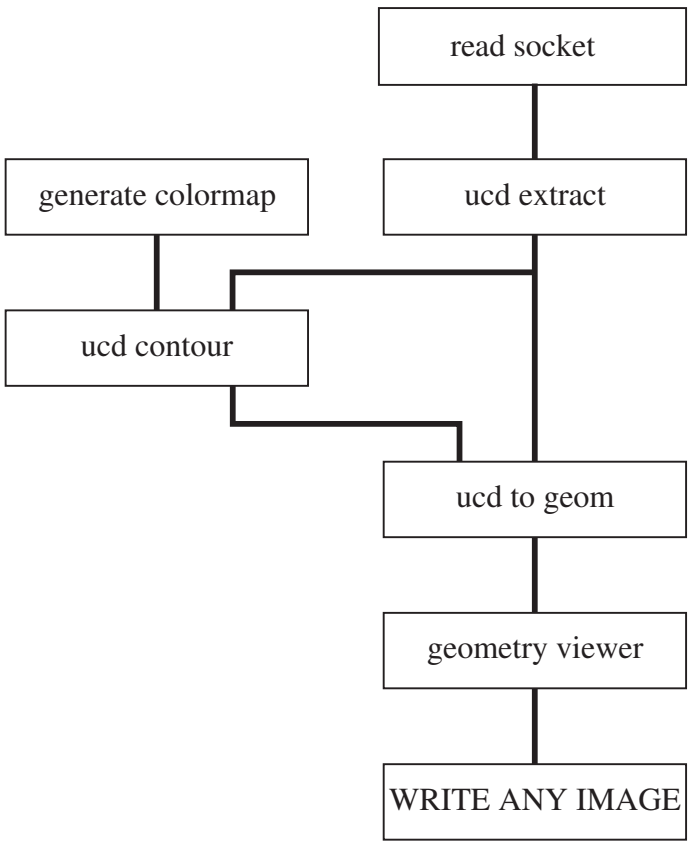

Abbildung 5.2: Netzwerk zum Erstellen der Bilddateien

5.3.2 Vor- und Nachspann, Zwischentitel

Zur Zeit kann das Hochschulrechenzentrum noch keine Software speziell zum Erzeugen eines Vor- und Nachspanns oder von Zwischen- und Untertiteln zur Verfügung stellen.

Aus Einzelbildern lassen sich die oben genannten Textparts eines Videofilms erzeugen. Hierfür eignen sich fast alle Programme zur Erzeugung von Grafiken oder Zeichnungen. Sie müssen die Option besitzen, GIF-, PPM- oder Postscript-Dateien schreiben zu können. Im PC-Bereich steht hier ein großes Angebot an Software zur Verfügung, wie z.B. Corel DRAW oder Autosketch. Die Textteile

des Films zur Darstellung der Ergebnisse aus FLOW wurden mit Corel DRAW erstellt. In Corel DRAW wurden die Texte und Formeln sowie das Logo der Technischen Hochschule zu Zeichnungen zusammengefügt. Zum Umwandeln der Zeichnungen bietet das Programm Exportfunktionen an. Die Exportfunktion zum Erstellen einer GIF-Datei hat eine Unzulänglichkeit, die erst spät bemerkt wurde: Der Nutzer gibt eine bestimmte Größe der zu erstellenden GIF-Datei an. Corel DRAW erzeugt nun eine Pixelgrafik in ähnlicher Größe. Die Abweichung lag bei Bildern von 720x576 Punkten bei etwa 20 Pixeln. Wenn diese Bilder nun nachträglich von einem anderen Programm auf die richtige Größe skaliert wurden, kam es zu Verzerrungen in den Texten. Dies wurde durch einen kleinen Trick umgangen. Zuerst wurde ein Rechteck für den Hintergrund erzeugt und als GIF-Datei abgespeichert. Diese Datei wurde danach mit dem Programm XView auf die richtige Größe gebracht. Der richtig skalierte Hintergrund wurde dann mit Corel DRAW durch die Texte und Formeln zu der gewünschten Zeichnung erweitert und zu einer GIF-Datei in der jetzt richtigen Größe gespeichert.

5.3.3 Verwenden des Videoservers

Zur Verwendung des *Videoservers* benötigt man ein Skript, in dem der Ablauf des Videos beschrieben wird. Dieses muß in einer vom HRZ entwickelten *Steuersprache* geschrieben werden. Diese Steuersprache ermöglicht sowohl die Einzelbildaufzeichnung auf die Bildplatte als auch die automatische Übernahme der Bilder durch den Rekorder. Zum Erstellen des Videos zur Darstellung der Ergebnisdaten des Programms FLOW wurde folgendes Skript verwendet:

```
. /local/video/bin/donaldlpr
```

```
mlpr RESET  TIMECODE=1:02:00:00
alpr RESET   FORMAT=GIF FILTER=FIT
tlpr  0  FRAMES=10:00  titel/titel1.gif
tlpr  0 FRAMES=10:00 titel/titel2.gif
tlpr  0 FRAMES=5:00  titel/titel10.gif
tlpr  0 FRAMES=5:00 titel/titel3.gif
tlpr  0 FRAMES=5:00 titel/titel4.gif
tlpr  A MAXF=100  geschw/geschw_*
tlpr  0 FRAMES=5:00 titel/titel5.gif
tlpr  2 vert_08/vert_08_*
tlpr  0 FRAMES=5:00 titel/titel6.gif
tlpr  2 vert_04/vert_04_*
tlpr  0 FRAMES=5:00 titel/titel11.gif
tlpr  0 FRAMES=10:00 titel/titel7.gif
tlpr  0 FRAMES=20:00 titel/titel8.gif
```

Die erste Zeile des Skriptes dient der Einbindung eines Include-Files, das die Verbindung zwischen den UNIX-Rechnern und der Videohardware herstellt. Mit dem `mlpr`-Befehl wird der Videorekorder initialisiert, die Startzeit festgelegt sowie das Band zurückgespult. Als nächstes wird die Bildplatte ebenfalls initialisiert und das Bildformat festgelegt. Hierfür ist der Befehl `alpr` zuständig. Nach diesen Anweisungen ist der Videoserver für die Produktion eines Videofilms initialisiert.
Mit dem Befehl `tlpr` können einzelnen Bilder zuerst auf die Bildplatte und dann automatisch auf das Videoband kopiert werden. Zur Ausführung des Befehls werden einige Informationen benötigt, die in Spalten hinter dem Befehlsaufruf stehen. In der ersten Spalte steht, ob ein Bild als Standbild gehalten werden soll oder wie oft eine Bildersequenz wiederholt werden soll. Eine 0 in der ersten Spalte bedeutet, daß das Bild als Standbild gehalten werden soll. Hier wird dann in der

zweiten Spalte die Zeit angegeben, wie lange das Bild gezeigt werden soll. Die Zeitangaben sind in dem Format `HH:MM:SS:FF` anzugeben. Hierbei steht `HH` für Stunden, `MM` für Minuten, `SS` für Sekunden und `FF` für einzelne Bilder. Führende Nullen dürfen entfallen. Der Name des Bildes steht in der dritten Spalte. Steht eine 2 in der ersten Spalte der Beschreibung, so handelt es sich um eine Sequenz von Bildern, die zweimal wiederholt wird. In der dritten Spalte stehen die Namen der Dateien. Die Namen der Dateien sind nicht beliebig. Die Namen aller Bilder einer Sequenz müssen den gleichen Anfang haben. Die letzten Stellen der Namen dienen zur Numerierung. Im Aufruf wird dann der erste Teil der Namen sowie die Numerierung durch eine Wildcard angegeben. Ein Sonderfall bei der Erstellung des Videos war die Darstellung der Geschwindigkeitsverteilung. Bei dieser Darstellung wurden mehr als 750 Bilder verwendet. Dadurch ist es notwendig, in der zweiten Spalte beim Aufruf von `tlpr` ein A anzugeben. Diese Notwendigkeit ergibt sich aus der Größe der Bildplatte.

5.3.4 Nachbearbeitung

Eine *Nachbearbeitung* des Films ist notwendig, wenn zu Beginn der Videoproduktion nicht alle Quelldateien vorhanden sind, eine Vertonung nicht angestrebt wird oder Untertitel gewünscht werden.

Bei der Produktion des oben beschriebenen Videofilms war kein Nachbearbeiten notwendig. Durch eine umfangreiche Vorbereitung des Videoprojekts waren zu Beginn der Produktion alle Bilddateien, d.h. sowohl die Titelbilder als auch die Filmsequenzen, vorhanden. Da das Video später bei Tagungen zur Illustration von Vorträgen verwendet werden soll, wurde auf eine Vertonung sowie die Verwendung von Untertiteln verzichtet.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Durch die Erweiterung der Schnittstelle PORTAL wurde eine brauchbare Möglichkeit geschaffen, On-Line-Visualisierungen von Simulationsrechnungen auf Parallelrechnern zu realisieren. Durch Optimierungen im Bereich der Kommunikation wurden die Austauschzeiten zwischen dem Parallelrechner sowie der darstellenden Workstation auf ein Minimum reduziert. Das Programm AVS ist eine sehr komfortable Möglichkeit zur Darstellung von Elementgeometrien, wie sie bei FE-Berechnungen vorkommen.

6.2 Ausblick

Im Bereich der Visualisierung von Ergebnisdaten, die auf einem Parallelrechner berechnet wurden, sind eine Reihe von weiteren Ansätzen denkbar. Welcher Ansatz verwendet wird, ist immer abhängig von der verwendeten Hardware und Software.

Eine Erweiterung der vorhandenen Schnittstelle ist wie folgt vorstellbar: Die

Schnittstelle zwischen dem Berechnungsprogramm und dem visualisierenden Programm ist über mehrere Links von mehreren Prozessoren aus möglich. Hierbei wäre dann die Synchronisation der Daten durch das Ausgabeprogramm oder die Schnittstelle zu vollziehen. Diese Vorgehensweise wäre sinnvoll, wenn das darstellende Programm wesentlich weniger Zeit benötigen würde als das berechnende.
Eine weitere vorstellbare Optimierung der Schnittstelle ist, daß die Menge der zu übertragenden Daten auf ein Minimum reduziert wird. Dies kann einerseits dadurch geschehen, daß die Schnittstelle auf einen Datentyp reduziert wird und andererseits nur noch die Ergebnisdaten überträgt.
Im ersten Fall würde man sich die meisten Kommunikationen für die Verwaltung von AVS ersparen können. Es wäre hierbei nur eine geringe Geschwindigkeitszunahme zu erwarten, da die Datenmengen für die Verwaltung nur sehr klein gegenüber den Datenmengen der Ergebnisdaten sind. Der Vorteil dieser Lösung dürfte eher in der Programmierung der Schnittstelle liegen. Man müßte keine Rücksicht auf Datenstrukturen nehmen, die eigentlich gar nicht übertragen werden sollen und kann dadurch einen verbesserten Transfer der Daten vom Parallelrechner zur darstellenden Workstation erreichen. In der zweiten Variante steckt ein größeres Potential der Optimierung. Die Menge der Ergebnisdaten macht im Vergleich zu den verschickten Daten nur einen geringen Teil aus. So ist z.B. in dem mehrmals angeführten Beispiel die Menge der ausgetauschten Daten 133 Kilobyte je Darstellung, während die Menge der entstandenen Ergebnisdaten nur bei etwa 27 Kilobyte liegt. Die restlichen Daten bestehen nur aus der Netzgeometrie.
Es ist sinnvoll, die Schnittstelle anders aufzubauen. Für die Übertragung der Netzgeometrie stehen zwei Varianten zur Verfügung. Entweder kann sie während der ersten Übertragung der Daten mit transferiert werden oder sie wird zu Beginn der Darstellung vom darstellenden Programm direkt aus einer Datei eingelesen.

Eine weitere sinnvolle Erweiterung der Schnittstelle wäre es, wenn eine Interaktion des Nutzers ermöglicht würde. Das Visualisierungsprogramm würde so zu der Bedienungsoberfläche des Berechnungsprogramms. Für diese Erweiterung ist eine größere Umstrukturierung der Schnittstelle notwendig, da hier Kommunikationen in beide Richtungen notwendig werden.

Die Geschwindigkeit der Darstellung von Geometrien ist auf normalen Workstations nicht immer befriedigend. Wenn das visualisierende Programm z.B. Lichtreflexe auf die Geometrie berechnen soll, ist die Verwendung von speziellen Grafikrechnern, wie einer Silicon Graphics, der Nutzung einfacher Workstations vorzuziehen. Für diese Darstellungsart ist wiederum eine andere Schnittstellenart notwendig. Diese Schnittstelle müßte dann dreigeteilt sein. Die Kommunikation direkt vom Parallelrechner zu einem Grafikrechner ist nur über einen Internetsocket möglich. Da dieser dann nur eine sehr geringe Übertragungsrate erreichen würde, ist eine Zwischenstation auf der Workstation direkt am Parallelrechner notwendig. Der Datentransfer zwischen dem Parallelrechner und der Workstation könnte dann über einen schnellen UNIX-Socket realisiert werden, während die Übertragung der Daten zum Grafikrechner dann über einen normalen Internetsocket vollzogen werden könnte. Hierdurch wäre eine optimale Übertragungsrate, aber auch der bestmögliche Datentransfer zu einem extern stehenden Grafikrechner mit ausreichender Darstellungsleistung verwirklicht.

Abbildungsverzeichnis

2.1 Berechnungsablauf einer FE-Berechnung 9

2.2 AVS Bildschirm 17

2.3 AVS Geometry Viewer 18

3.1 Definition benetztes Element 21

3.2 Erläuterung zu den Gleichungen 22

3.3 Zu berechnendes Gebiet 25

3.4 Netzaufteilungsalgorithmus 28

3.5 Datenstruktur von FLOW 30

4.1 Aufbau der Bibliothek PORTAL für parallele Aufgaben 32

4.2 Reihenfolge der Pakete bei der Übertragung durch PORTAL . . . 35

4.3 Vereinfachter Aufbau des Empfängers 39

4.4 Datenstruktur von UCD 40

4.5 Aufbau des Datenfeldes 43

4.6 Netzwerk zum Darstellen von UCD 47

5.1 Ablauf bei der Einzelbildaufzeichnung 58

5.2 Netzwerk zum Erstellen der Bilddateien 62

Literaturverzeichnis

[AVS, 1992] Advanced Visual Systems Incorporation: AVS Developers Guide, Waltham, 1992.

[Farhat, 1988] Fahat, C.: A simple and efficient automatic FEM domain decomposer, Computer & Structures, -28(1988), -579-602.

[Frommer, 1990] Frommer, A.: Lösung linearer Gleichungssysteme auf Parallelrechnern, Braunschweig, 1990.

[Geijn, 1992] van de Geijn, R.: On global combine operations, Univ. of Texas, Austin, Texas, October 1992.

[Kämpf, 1995] Kämpf, R.: Computeranimation: WWW-Seiten des HRZ der TH-Darmstadt, Darmstadt, 1995.

[Lämmer, 1993] Lämmer, L.: Skript zur Vorlesung: Paralleles Rechnen für das Bauingenieurwesen, Darmstadt 1993.

[Ludwig, 1993] Ludwig, T.: Automatische Lastverteilung für Parallelrechner, Mannheim, 1993.

[Meißner, 1989] Meißner, U., Menzel, A.: Die Methode der finiten Elemente, Hannover, 1989.

[Möller, 1992] Möller, B.: Umdruck zur Vorlesung: Bauinformatik III, SS 1992, Darmstadt, 1992.

[Rowlan, 1993] Rowlan, J.S., Wightman, B.T.: PORTAL: A communication library for run-time visualization of distributed, asynchronous data, Argonne, 1993.

[Santifaller, 1990] Santifaller, M.: TCP/IP und NFS in Theorie und Praxis, Unix in lokalen Netzen, 1990.

[ULM, 1994] Umetsu,T. Lämmer, L., Meißner, U.: Two-step explict finite element method for shallow water flow using a scalable parallel approach. Proc. X. Intl. Conf. on Computational Methods in Water Resources 1994.

Anhang A

Erklärung

Hiermit erkläre ich, daß ich diese Arbeit selbständig und nur mit den angegebenen Quellen angefertigt habe.

Darmstadt, den 7. August 1995

Anhang B

Quellcode

Quellcode zum Kopieren der Geometrie in die AVS - Struktur:

```
#include <stdio.h>
#include <string.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/geom.h>
#include <avs/ucd_defs.h>
#include <avs/field.h>
#include <math.h>

/* Socket includes */
#include <portal/sockets.h>
#include <portal/datasock.h>
#include <portal/AVSinit.h>
#include "AVSinterface.h"
#define  NUM_CELL_TYPES  8
extern char    *AVSstatic;

/*-----------------------------------------------------*
 *                                                     *
 *               ****  add_cell  ****                 *
 *                                                     *
 *-----------------------------------------------------*/

/* Einfuegen eines Elementes in die Struktur*/

add_cell(model, cell, ucd_cell_type, n, rnode_list, mat_id)
        int             cell, ucd_cell_type, n, *rnode_list, mat_id;
```

```
        UCD_structure **model;
{

        static char    *cell_type[] = {"pt", "line", "tri", "quad", "tet",
        "pyr", "prism", "hex"};

        int             i, j, num_me_nodes, num_nodes, cell_found, me_flag,
                        node_list[40];

        if (ucd_cell_type < NUM_CELL_TYPES) {
                me_flag = 0;
                num_nodes = UCD_num_nodes[ucd_cell_type];

                for (i = 0; i < num_nodes; i++)
                        node_list[i] = rnode_list[i] - 1;

                if (n != num_nodes) {
                        for (i = num_nodes, num_me_nodes = 0; i < n; i++)
                                if (rnode_list[i] != 0) {
                                        node_list[num_nodes + num_me_nodes] =
                                                rnode_list[i] - 1;
                                        me_flag = me_flag | (0x1 <<
                                                (i - num_nodes));
                                        num_me_nodes++;
                                }
                }
                UCDcell_set_information(*model, cell, cell + 1,
                cell_type[ucd_cell_type], mat_id - 1, ucd_cell_type,
                me_flag, node_list);

        }
}



/*-----------------------------------------------------*
 *                                                     *
 *            ****  copy_DS_UCD_data  ****             *
 *                                                     *
 *-----------------------------------------------------*/
/* Kopieren der Geometrie in die Struktur */

int
```

```
copy_DS_UCD_data(UCD_structure ** output, void *from, long int type, int size,
                 int secondtime)
{

        /* Definition der Strukturen  */


        typedef struct _ctype_struct {
                int             id, mat, n, cell_type;
        }               Ctype;

        typedef struct _ntype_struct {
                int             n, node_list[20];
        }               Ntype;

        typedef struct _model_stats {
                char            node_data_labels[100], cell_data_labels[100],
                                model_data_labels[100], node_data_units[100],
                                cell_data_units[100], model_data_units[100];

                int             num_nodes, num_cells, num_node_data,
                                num_cell_data, num_model_data, node_active_list[20],
                                cell_active_list[20], model_active_list[20],
                                num_node_comp, node_comp_list[20],
                                num_cell_comp, cell_comp_list[20], num_model_comp,
                                model_comp_list[20], num_nlist_nodes;
        }               Mtype;


        typedef struct _ucd_stats {
                int             size;
        }               Ucd_Stats_Type;

        /* Variablendeklaration */

        char            string[40], tmp_str[80], file_name[80], model_name[20];
        DS_UCD         *temp = (DS_UCD *) from;
        int             i, num_nodes, num_cells, num_node_data, num_cell_data,
                        num_model_data, cell_tsize, node_csize, ucd_flags,
                        node, cell, util_flag;
        float           x, y, z, *data, xmin = 0.0, xmax = 0.0, ymin = 0.0,
                        ymax = 0.0, zmin = 0.0, zmax = 0.0, min_extent[3],
                        max_extent[3];
```

```
float          *xc, *yc, *zc, *node_data, *cell_data, *model_data,
               *max_node_data, *min_cell_data, *max_cell_data,
               *min_model_data, *max_model_data, *min_node_data;
Ntype          *cell_nlists;
Ctype          *cells;
static char    *cell_type[] = {"pt", "line", "tri", "quad", "tet",
"pyr", "prism", "hex"};
Mtype           model_stats;
float          *data_real;
Ucd_Stats_Type *ucd_stats;
static int      clist[] = {1, 2, 3, 4};


cell_nlists = NULL;
if (AVSstatic == NULL) {
        ucd_stats = (Ucd_Stats_Type *) malloc(sizeof(Ucd_Stats_Type));
        AVSstatic = (char *) ucd_stats;
} else
        ucd_stats = (Ucd_Stats_Type *) AVSstatic;

/* Allgemeine Beschreibung */

model_stats.num_nodes = temp->num_nodes;
model_stats.num_cells = temp->num_cells;
model_stats.num_node_data = temp->num_node_data;
model_stats.num_cell_data = temp->num_cell_data;
model_stats.num_model_data = temp->num_model_data;

/* alloc der Felder fuer Koordinaten (xc,yc,zc) und der Elemente
   (cells, cell_nlists) */

cells = (Ctype *) malloc(sizeof(Ctype) * model_stats.num_cells);
cell_nlists = (Ntype *) malloc(sizeof(Ntype) * model_stats.num_cells);
xc = (float *) malloc(sizeof(float) * model_stats.num_nodes);
yc = (float *) malloc(sizeof(float) * model_stats.num_nodes);
zc = (float *) malloc(sizeof(float) * model_stats.num_nodes);

/* Lesen der Knotenkoordinaten aus temp->data_ptr */

data_real = (float *) (temp->data_ptr);
for (i = 0; i < model_stats.num_nodes; i++)
        xc[i] = *data_real++;
xmin = *data_real++;
```

```
xmax = *data_real++;
for (i = 0; i < model_stats.num_nodes; i++)
        yc[i] = *data_real++;
ymin = *data_real++;
ymax = *data_real++;
for (i = 0; i < model_stats.num_nodes; i++)
        zc[i] = *data_real++;
zmin = *data_real++;
zmax = *data_real++;

/* Lesen der Elementdaten (nur Elemente vom Typ tri) aus
   temp->data_ptr */

model_stats.num_nlist_nodes = 0;
for (i = 0; i < model_stats.num_cells; i++) {
        cells[i].cell_type = 2;
        cells[i].id = 1;
        cells[i].mat = 1;
        cell_nlists[i].node_list[0] = (int) *data_real++;
        cell_nlists[i].node_list[1] = (int) *data_real++;
        cell_nlists[i].node_list[2] = (int) *data_real++;
        cell_nlists[i].n = 3;
        cells[i].n = 3;
        model_stats.num_nlist_nodes = model_stats.
                                      num_nlist_nodes + 3;
}


/* Lesen der Knotenwerte aus temp->data_ptr */

model_stats.node_active_list[0] = 1;
strcpy(model_stats.node_data_labels, "hoehen");
strcpy(model_stats.node_data_units, "lb/in**2");
strcpy(string, "hoehen, lb/in**2");
model_stats.node_comp_list[0] = 1;
model_stats.node_comp_list[1] = 0;
model_stats.num_node_comp = 1;
for (i = 1; i < model_stats.num_node_comp; i++)
        model_stats.node_active_list[i] = 0;
node_data = (float *) malloc(sizeof(float) *
       model_stats.num_nodes * model_stats.num_node_data);
min_node_data = (float *) malloc(sizeof(float) *
      model_stats.num_node_data);
```

```
max_node_data = (float *) malloc(sizeof(float) *
      model_stats.num_node_data);
for (i = 0; i < model_stats.num_nodes; i++) {
        node_data[i] = *data_real++;
}
xmin = xmax = node_data[0];
for (i = 1; i < model_stats.num_nodes; i++) {
        min_node_data[0] = (node_data[i] < min_node_data[0]
                         ? node_data[i] : min_node_data[0]),
        max_node_data[0] = (node_data[i] > max_node_data[0]
                         ? node_data[i] : max_node_data[0]);
}
for (i = 1; i < model_stats.num_node_comp; i++)
        model_stats.node_active_list[i] = 0;

/* Kopieren der Daten nach AVS */

util_flag = 0;
num_nodes = model_stats.num_nodes;
num_cells = model_stats.num_cells;
num_node_data = model_stats.num_node_data;
num_cell_data = model_stats.num_cell_data;
num_model_data = model_stats.num_model_data;
cell_tsize = node_csize = model_stats.num_nlist_nodes;
ucd_stats->size = cell_tsize;
ucd_flags = UCD_INT | UCD_MATERIAL_IDS | UCD_NODE_NAMES |
          UCD_CELL_NAMES | UCD_CELL_TYPES | UCD_MID_EDGES;
strcpy(model_name, "ucd_bild");
if (secondtime == 0)
        *output = (UCD_structure *) UCDstructure_alloc
                  (model_name, num_model_data, ucd_flags,
                   num_cells, cell_tsize,num_cell_data,
                   num_nodes, node_csize, num_node_data,
                   util_flag);

/* Knotenwerte */

for (node = 0; node < num_nodes; node++) {
        x = xc[node], y = yc[node], z = zc[node];
        if (node) {
                xmin = (x < xmin ? x : xmin),
                   xmax = (x > xmax ? x : xmax);
                ymin = (y < ymin ? y : ymin),
```

```
                ymax = (y > ymax ? y : ymax);
              zmin = (z < zmin ? z : zmin),
                zmax = (z > zmax ? z : zmax);
        } else {
              xmin = xmax = x;
              ymin = ymax = y;
              zmin = zmax = z;
        }
        if (!UCDnode_set_information(*output, node,
                node + 1, 0, clist)) {
              AVSerror("Error in read_ucd:
                can't set node %d info.\n", node);
              return (0);
        }
}

UCDstructure_set_node_positions(*output, xc, yc, zc);

/* Extremalwerte */

min_extent[0] = xmin, min_extent[1] = ymin,
   min_extent[2] = zmin;
max_extent[0] = xmax, max_extent[1] = ymax,
   max_extent[2] = zmax;

UCDstructure_set_extent(*output, min_extent, max_extent);

/* Elemente  */

for (cell = 0; cell < num_cells; cell++) {

        add_cell(output, cell, cells[cell].cell_type,
           cell_nlists[cell].n, cell_nlists[cell].node_list,
           cells[cell].mat);
}

/* Ergebnisdaten fuer Knoten, Elemente, Geometrie */

if (num_node_data) {
    UCDstructure_set_node_components(*output,
       model_stats.node_comp_list, model_stats.num_node_comp);
    UCDstructure_set_node_active(*output,
       model_stats.node_active_list);
```

```
            UCDstructure_set_node_labels(*output,
               model_stats.node_data_labels, ".");
            UCDstructure_set_node_units(*output,
               model_stats.node_data_units, ".");
            if (model_stats.num_node_comp > 1) {
                 for (i = 0; model_stats.node_data_labels[i] != '.'; i++)
                            string[i] = model_stats.node_data_labels[i];
                 string[i] = '\0';
            } else
                strcpy(string, model_stats.node_data_labels);
            UCDstructure_set_node_minmax(*output, min_node_data, max_node_data);
            if (!UCDstructure_set_node_data(*output, node_data)) {
                AVSerror("Error in read_ucd: can't set node data.\n");
                return (0);
           }
      }
      free(xc);
      free(yc);
      free(zc);
      if (num_node_data) {
           free(node_data);
           free(min_node_data);
           free(max_node_data);
      }
      if (num_cell_data) {
           free(cell_data);
           free(min_cell_data);
           free(max_cell_data);
      }
      if (num_model_data) {
            free(model_data);
            free(min_model_data);
            free(max_model_data);
      }
      free(cells);

      if (cell_nlists)
            free(cell_nlists);
      free(ucd_stats);
}
```