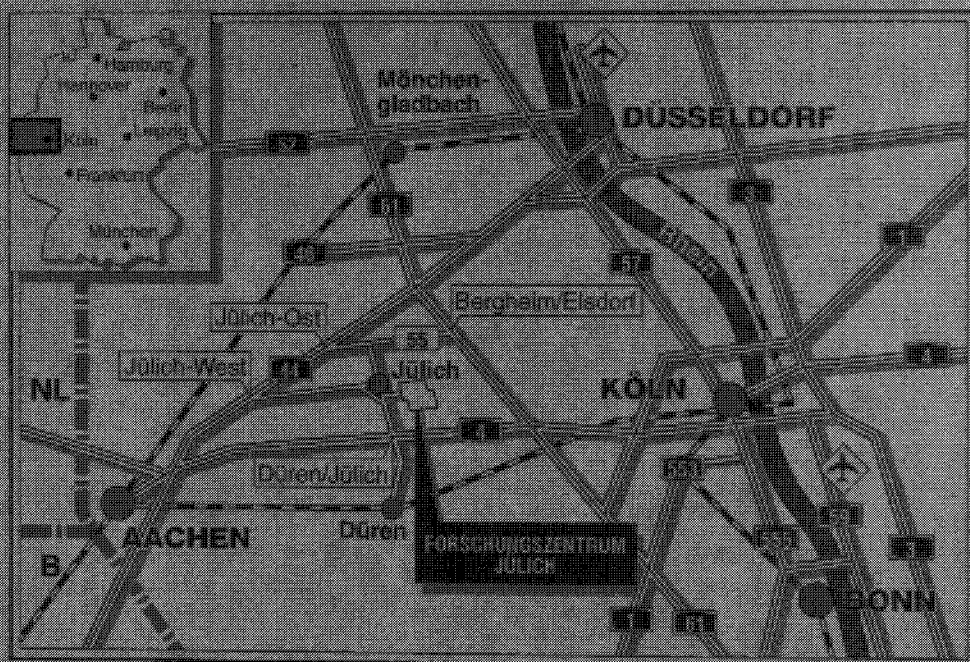


Zentralinstitut für Angewandte Mathematik

**Visualisierung verteilt berechneter,
mehrdimensionaler Datenfelder**

Ansgar Kempkes



Berichte des Forschungszentrums Jülich ; 3216
 ISSN 0944-2952
 Zentralinstitut für Angewandte Mathematik Jüli-3216

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
 D-52425 Jülich · Bundesrepublik Deutschland
 Telefon: 02461/61-6102 · Telefax: 02461/61-6103 · Telex: 833556-70 kfa d

Visualisierung verteilt berechneter, mehrdimensionaler Datenfelder

Ansgar Kempkes

Zusammenfassung

Im Bereich der Simulationsrechnung werden zunehmend Parallelrechner eingesetzt. Bei einem Ansatz zur Datenparallelisierung, der sogenannten *domain decomposition*, wird das gegebene Gesamtproblem in mehrere kleine, räumlich begrenzte Teilgebiete aufgeteilt. Diese Teilgebiete werden von den einzelnen Prozessoren des Parallelrechners simultan bearbeitet. Manche der Rechenverfahren, wie zum Beispiel das Schwarz'sche Verfahren, benutzen eine Gebietsaufteilung, bei denen sich die Teilgebiete überlappen. Daten aus diesen Überlappzonen werden während der Rechnung zwischen den Prozessorknoten ausgetauscht und bilden die neuen Randwertbedingungen der Teilgebiete im nächsten Iterationsschritt.

Nicht nur am Ende der Berechnung, sondern auch während des Rechnungslaufes stellt sich die Frage nach der Überprüfung und Validierung der verteilten Daten. Zu diesem Zweck gibt es zwar bereits eine Vielzahl generell leistungsfähiger Visualisierungswerkzeuge, doch waren bislang diese Visualisierungssysteme von ihrer Konzeption her nicht an die speziellen Belange der Darstellung paralleler Rechenergebnisse angepaßt.

In dieser Diplomarbeit wird ein Konzept zur Visualisierung verteilt berechneter Datenfelder erarbeitet, das auf der Erstellung eines Modellfeldes aus den Daten beruht. Von diesem Konzept ausgehend, werden Methoden und Verfahren vorgestellt, mit denen verteilte Daten selektiert, visualisiert und verglichen werden können. Als Anwendungsbeispiel dient die Visualisierung der Ergebnisse der parallelen Programme TRACE und ParTrace, mit denen Stofftransportvorgänge im Grundwasser berechnet werden können.

Abstract

Parallel and distributed computers are increasingly used in the field of computer simulations. One approach used for porting simulation problems to parallel computers is the data parallelism approach of domain decomposition. Within this approach, a given problem area is divided into several domains. All computing nodes of the parallel computer then calculate on each domain in parallel. Some parallel calculation methods based on domain decomposition, as for example the Schwarz Method, use overlap regions between the domains. During computation, data from these overlap regions are exchanged and used as boundary conditions for the next iteration step.

At the end of the computation, as well as during the simulation run, the question raises how to control and how to validate the data and the computing process. For these purposes, a manifold of efficient visualization tools exist, but they were so far not adapted to the special issues in visualizing parallel computed results.

In this diploma thesis, a concept for visualizing distributed data arrays is elaborated. The concept is based on generating a model field from the data. Methods and techniques for selecting, visualizing and comparing distributed data arrays are introduced. The visualization of the results of the parallel programs TRACE and ParTrace serves as an application example.

Inhaltsverzeichnis

1	Einleitung	1
2	Problemstellung	3
2.1	Hydrogeologische Grundbegriffe	3
2.2	Die Methode der Finiten Elemente	6
2.3	TRACE und ParTrace	8
2.4	Parallelität in TRACE und ParTrace	11
2.4.1	Gebietszerlegungen und Schwarz'sches Verfahren	11
2.4.2	Gemischte Speicherverwaltung	16
2.5	Problematik der Darstellung	17
3	Ansätze und Werkzeuge	19
3.1	Wissenschaftliche Visualisierung	19
3.2	Ansätze zur Visualisierung	20
3.2.1	Datenflußparadigma	20
3.2.2	Modellbasierter Ansatz	25
3.2.3	Klassifizierung von Feldern und Ansichten	26
3.2.4	Gittermodell	28
3.3	Visualisierungswerkzeuge	30
3.3.1	Schlüsselfertige Anwendungen	30
3.3.2	Modulare Visualisierungsumgebungen	31
4	DAVis: Konzeption	35
4.1	Anforderungen an die Visualisierung verteilt berechneter Datenfelder	35
4.2	Ansatz	37
4.3	Datenmodell und Datenstrukturen	41
4.3.1	Die Feld-Datenstruktur	41

4.3.2	Annahmen für die verteilte Datenmenge	46
4.3.3	Einbetten der verteilten Datenfelder in eine Feld-Datenstruktur	47
4.3.4	Partitions- und Domänengrenzen	55
4.3.5	Modellieren der Variablen X_C und Y_C	62
4.4	DAVis-Suite als Methodensatz	66
4.4.1	DAVisGenerate	67
4.4.2	DAVisDisplay	68
4.4.3	DAVisEmbed	70
4.4.4	DAVisModel und DAVisDiff	71
4.5	Benutzeroberfläche	72
5	DAVis: Realisierung	75
5.1	Datenstrukturen	75
5.1.1	Die Feld-Datenstruktur	75
5.1.2	Feld-Partitionierung	77
5.1.3	Domänen-Hierarchie	78
5.1.4	Weitere Datenstrukturen	81
5.2	Die Module der DAVis-Suite	82
5.2.1	AVS-Module	82
5.2.2	DAVisRead	84
5.2.3	DAVisDisplay	85
5.2.4	DAVisEmbed	88
5.2.5	DAVisModel	89
5.3	Benutzeroberfläche	90
5.3.1	Aufbau	90
5.3.2	DAVisMenu	92
5.3.3	Das Netzwerk der Module der DAVis-Suite	95
6	Visualisierung hydrogeologischer Daten	97
6.1	Die Aufgabe	97
6.2	Die Visualisierung der Sequenz aus ParTrace	97
6.3	Die Visualisierung der verteilten Daten aus TRACE	102
7	Zusammenfassung und Ausblick	107
	Quellenverzeichnis	109
A	Hülle um eine Gruppen-Domäne	115

Abbildungsverzeichnis

2.1	Bodenprofil und Grundwasserfluß	4
2.2	Das Programmpaket TRACE	9
2.3	Zur Erläuterung des Schwarz'schen Verfahrens	13
2.4	Streng reguläres Gitter finiter Elemente in TRACE	14
2.5	Aufteilung eines Gebietes in Domänen (<i>domain decomposition</i>)	15
3.1	Der Zyklus des wissenschaftlichen Arbeitens	20
3.2	Das Datenflußparadigma	21
3.3	Volumenvisualisierung	23
3.4	Modellbasierter Ansatz	25
3.5	Felder in AVS	33
4.1	Reale Visualisierung als Näherung einer idealen Visualisierung	37
4.2	Die reale Visualisierung verteilter Daten	38
4.3	Getrennte Modellierung von unabhängiger und abhängiger Variable .	41
4.4	Kiviat-Diagramm	45
4.5	Kiviat-Tube	46
4.6	Einbetten: Erzeugen der Feld-Partitionen	48
4.7	Einbetten von Datenfeldern: Fall A	49
4.8	Einbetten von Datenfeldern: Fall B	51
4.9	Aufteilung in Feld-Domänen	53
4.10	Reguläre Feld-Partitionierung	54
4.11	Einbettung nach Fall A und nach Fall B	56
4.12	Interpolation zwischen Domänen, ohne und mit Leerstellen	57
4.13	Leerstellen zwischen den Feld-Partitionen	58
4.14	Leerstellen zwischen den Feld-Partitionen und den Überlappzonen . .	59
4.15	Ansicht von einer Feld-Datenstruktur $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$	62

4.16	Koordinatenfolgen	63
4.17	Ansichten nach Modellierung	65
4.18	Überlappzonen in Referenzraum und Darstellungsraum	68
4.19	Hülle um eine Gruppendomäne	70
4.20	Vierstufiger Visualisierungsprozeß für die Benutzeroberfläche	73
5.1	Maschinenteil, aus verschiedenen Teilen zusammengesetzt	77
5.2	Nicht reguläre (a) und reguläre (b) Domänen-Aufteilung	79
5.3	Baumstruktur für die Domänen-Hierarchie	80
5.4	Eingefärbte Domänen und Überlappzonen	87
5.5	Selektion von Domänen	87
5.6	Die Oberfläche von DAVisTool	91
5.7	Das DAVis-Modul Isosurf	94
5.8	Das Netzwerk der Module der DAVis-Suite in AVS	95
6.1	Spezifizieren der Eingabe-Sequenz	98
6.2	Verschieben des FE-Gitters von ParTrace um ein halbes Element	99
6.3	Pipeline des Konzentrationsfeldes	100
6.4	Pipeline des Permeabilitätsfeldes	100
6.5	Konzentrationswolke im letzten Zeitschritt	101
6.6	Parametrierung von DAVisRead und DAVisDisplay	102
6.7	Selektion aller Domänen	103
6.8	Parametrierung von DAVisEmbed und HedgeHog	104
6.9	Erstelltes Netzwerk	104
6.10	Visualisierung von Schadstoffkonzentration und Fließfeld	105
A.1	Hülle um eine Gruppendomäne	115
A.2	Zerlegung des Problems	116
A.3	Polygonzug um eine Punktmenge	117
A.4	Die zu prüfende Punktmenge für die Randlinie * aus Abb. A.2	118
A.5	Vier zu unterscheidende Fälle im Referenzraum	119

Kapitel 1

Einleitung

Im Laufe der letzten 20 Jahre hat sich in Wissenschaft und Forschung die Simulation als drittes Standbein zwischen Theorie und Experiment durchsetzen können. Mit einer Simulation lassen sich Experimente ersetzen, deren Durchführung zu aufwendig, zu gefährlich oder aus anderen Gründen nicht realisierbar ist.

Die Entwicklung der Simulationsrechnung ist untrennbar mit der Entwicklung der Rechnersysteme und Algorithmen verbunden. Die wachsende Verfügbarkeit immer leistungsfähigerer Hardware und Software verhalfen der computergestützten Simulation zum Durchbruch. Eine besondere Rolle spielen dabei in der letzten Zeit die verteilten und parallelen Rechner und Algorithmen. Parallele Ansätze ermöglichen in überschaubarer Zeit die Lösung umfangreicher Simulationsaufgaben; in manchen Fällen erlauben sie sogar erst das Angehen von Problemen, deren Lösung vor einigen Jahren noch als viel zu aufwendig galt.

Kennzeichnend für die Entwicklung im Bereich der numerischen Simulationsrechnung und im Supercomputing sind die wachsenden Datenmengen. Typische Datensatzgrößen einer Simulationsrechnung liegen heutzutage über dem Gigabyte-Bereich. Als reine Zahlendarstellung sind diese Datenmengen nicht zu erfassen; nur in grafischer Form oder als Film lassen sich die wichtigen Inhalte und Ergebnisse den Daten entnehmen [FHö92]. Die Umsetzung von Dateninhalten in grafische Darstellungen nennt man Visualisierung [REar92]. Sie ist zu einem wichtigen Element wissenschaftlicher Arbeit geworden, stellt aber höchste Anforderungen an Speicherplatz, Datenkommunikation und Rechenleistung. Sie ist damit eng mit dem Höchstleistungsrechnen verknüpft. Viele der heute verfügbaren Hilfsmittel zur Visualisierung sind sehr leistungsfähig und besitzen eine intuitive Benutzerschnittstelle. Jedoch gibt es noch keine Visualisierungssysteme, die bereits von der Konzeption her auf die speziellen Belange des parallelen und verteilten Rechnens eingerichtet sind. In allen Fällen muß das Visualisierungswerkzeug an diese Belange angepaßt werden.

In dieser Arbeit wird ein neues Konzept vorgestellt, nach dem Ergebnisse verteilt oder parallel berechneter Simulationen, sogenannte verteilte Ergebnisse, visualisiert werden können. Aus dem Konzept wird exemplarisch eine Implementation entwickelt und innerhalb der modularen Visualisierungsumgebung AVS realisiert. Als Anwendungsbeispiel von Konzept und Implementation dient die Visualisierung der parallel

berechneten Ergebnisse der Programme TRACE und ParTrace, Programmen zur Berechnung des Schadstofftransports im Grundwasser.

Nach einer Einführung in die Problemstellung in Kapitel 2, bei der die wesentlichen Rechengänge und -modelle von TRACE und ParTrace erläutert werden und die Visualisierungsproblematik aufgestellt wird, gibt Kapitel 3 einen Überblick über die bereits vorliegenden Ansätze und Werkzeuge im Bereich der Visualisierung. Ausgehend von der generellen Visualisierungsproblematik werden in Kapitel 4 zunächst die Anforderungen an eine *Distributed Array Visualization* formuliert. Aufbauend auf dem modellbasierten Ansatz [KBro93] und dem Gittermodell [WHib94a] der Visualisierung wird anschließend der Ansatz "DAVis" erarbeitet. Zum DAVis-Ansatz gehören Datenmodelle und Datenstrukturen zur Repräsentation der verteilten Daten innerhalb des Visualisierungssystems. Eine Methoden-Suite ergänzt den Ansatz um die notwendigen Verfahren und Algorithmen zur Interaktion mit den verteilten Daten und zur Bearbeitung der spezifischen Datenstrukturen. Zur Methoden-Suite gehört auch ein geometrischer Algorithmus, mit dem die Hülle um eine Geometrie gefunden werden kann, die aus Blöcken zusammengesetzt ist. Dieser Algorithmus wird in Anhang A beschrieben. Das DAVis-Konzept wird in Kapitel 5 in eine Beispiel-Implementation unter AVS umgesetzt. Kapitel 6 zeigt die Lösung eines Visualisierungsproblems für einige Datensätze aus TRACE und ParTrace. Abschließend wird in Kapitel 7 eine Zusammenfassung und ein Ausblick gegeben.

Kapitel 2

Problemstellung

Die Problemstellung besteht in der Visualisierung verteilt berechneter Daten, wie sie beispielsweise nach und während des Programmlaufes von TRACE vorliegen. TRACE ist ein Simulationsprogramm der Hydrogeologie, mit dem der Schadstofftransport im Grundwasser untersucht werden soll; auf dieses Programm wird noch in Abschnitt 2.3 näher eingegangen. Zuvor soll hier eine kurze Erläuterung der Begriffe und Modelle der Hydrogeologie sowie der numerischen Simulation gegeben werden, soweit sie zur Erstellung des Konzeptes und für die Visualisierung notwendig sind. Daran anschließend werden die Probleme für die Visualisierung diskutiert, die sich aus diesen Vorgaben und Modellen ergeben.

2.1 Hydrogeologische Grundbegriffe

Mit der fortschreitenden Industrialisierung und der intensiven Bodennutzung in der Landwirtschaft stellen sich eine Reihe von Umweltproblemen. Eines davon, bedingt durch die Anreicherung von Schadstoffen im Boden, besteht in der bislang stetigen Verschlechterung der Grundwasserqualität. Um Böden geeignet von Schadstoffen sanieren zu können, sei es direkt nach einem Unfall oder längere Zeit nach dem Aufspüren vorhandener Altlasten, ist die Kenntnis der Stofftransportvorgänge im Boden von eminenter Bedeutung.

Grund- und Sickerwasser spielen bei diesen Transportvorgängen die Rolle des Trägers. Partikel werden entweder als im Wasser gelöste Teilchen oder als ungelöste Kolloide mit der Strömung weggeschwemmt. Das Sickerwasser transportiert die Partikel vertikal durch die ungesättigte Zone in die gesättigte Grundwasserzone, wo sie mit dem Grundwasser annähernd horizontal weiterbewegt werden (siehe Abb. 2.1). Die Bewegung der Partikel mit der Grundwasserströmung läuft recht langsam ab; Grundwasser fließt mit einer Geschwindigkeit von deutlich weniger als einem Meter pro Tag [BHöl84]. Deshalb lassen sich die Auswirkungen von Schadstoffen, die in das Bodensystem gelangen, oft erst nach Jahren erkennen. Zusätzlich wird die Untersuchung der Transportvorgänge durch die komplexen physikalischen, chemischen und

biologischen Prozesse im Boden erschwert. Beispielsweise sorbieren Böden die Schadstoffe für eine unbestimmte Zeit, oder neue Schadstoffe entstehen bei chemischen oder biologischen Prozessen im Boden. Trotzdem wird versucht, die Vorgänge im Boden durch hinreichend genaue mathematisch-physikalische Modelle anzunähern, um Vorhersagen über die Verteilung von Schadstoffen treffen zu können.

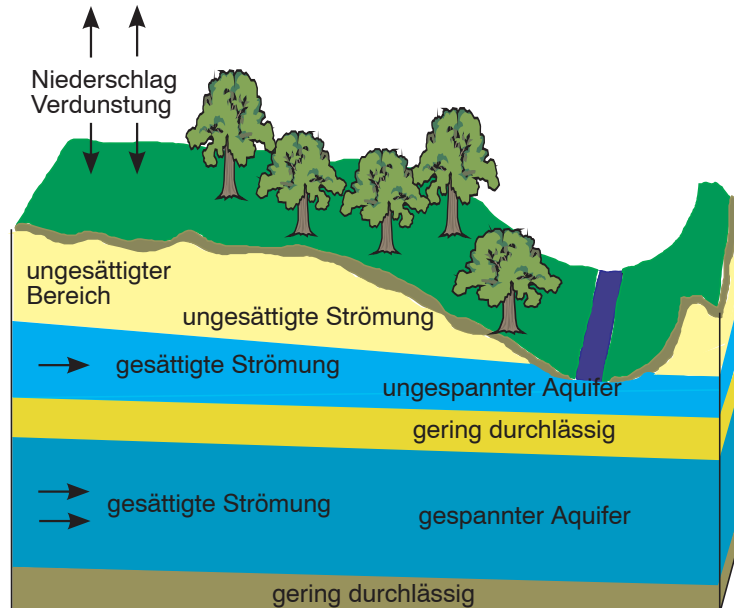


Abbildung 2.1: Bodenprofil und Grundwasserfluß

Erste mathematische Bodenmodelle, mit denen der Fluß des Grundwassers annähernd beschrieben werden konnte, stellten unabhängig voneinander Darcy 1856 sowie Buckingham 1907 auf. Die Bodenmodelle waren das Resultat ihrer Untersuchungen und Experimente, zum Beispiel an mit Sand gefüllten Röhren [CFet93, RFre79, GSpö86]. Nach diesen Ergebnissen ist der Wasserfluß $\vec{q}$ in Volumeneinheiten pro Fläche und Zeit in der gesättigten und der ungesättigten Zone der Gradient des totalen Potentials H und wird in der **Darcy-Gleichung** beschrieben als:

$$\vec{q} = \frac{\vec{Q}}{A} = -\mathbf{K}(h) \cdot \nabla H \quad . \quad (2.1)$$

Das totale Potential setzt sich aus dem hydrostatischen Potential h und dem Gravitationspotential z zusammen:

$$H = h + z \quad . \quad (2.2)$$

Das hydrostatische Potential h (auch Druckhöhe oder *pressure head* genannt) ist abhängig vom Wassergehalt, der Bodendichte und theoretisch auch der Temperatur. Praktisch spielt die Temperaturabhängigkeit aber keine Rolle, da die Temperaturen

im Boden als in etwa konstant zu 10°C angenommen werden können. Das Gravitationspotential z folgt aus der Höhe über einem Bezugspotential (siehe dazu auch [ONeu96]).

Der Tensor $\mathbf{K}$ in Gleichung (2.1) beschreibt die hydraulische Leitfähigkeit oder Permeabilität des Bodens; die Permeabilität wird in Längeneinheiten pro Zeit gemessen und hängt von der Sättigung Θ des Bodens mit Wasser ab. Der Sättigungsgehalt beträgt in der gesättigten Zone konstant 100 %; in der ungesättigten Zone jedoch ist er eine Funktion des hydrostatischen Potentials und kann dort empirisch durch die Mualem-van-Genuchten-Gleichung [MvGe80, HVer94] beschrieben werden. $\mathbf{K}$ kann also insgesamt als eine Funktion von h aufgefaßt werden. Zudem ist die Permeabilität ortsabhängig; üblicherweise wird sie für die Simulation als statistische Verteilung mit vorgegebenen Dichtespektren und Korrelationslängen modelliert.

Die Geschwindigkeit $\vec{v}_D = -\vec{q}$ wird die Darcy- oder Filter-Geschwindigkeit genannt und ist die mittlere Strömungsgeschwindigkeit des Wassers im betrachteten Volumen.

Für den Wassergehalt Θ gilt nun die Kontinuitätsgleichung

$$\frac{\partial \Theta}{\partial t} = -\nabla \vec{q} \quad . \quad (2.3)$$

Mit der Darcy-Gleichung (2.1) und einem Quellen- und Senkenterm S wird diese Kontinuitätsgleichung erweitert zu

$$F \cdot \frac{\partial h}{\partial t} = \nabla (\mathbf{K}(h) \cdot \nabla H) + S \quad (2.4)$$

mit dem Speicherkoeffizienten

$$F = \frac{\partial \Theta}{\partial h} \quad . \quad (2.5)$$

Der Speicherkoeffizient beschreibt, wie sich der Wassergehalt mit dem Druck ändert. Je kleiner F , desto schneller reagiert das System auf eine Druckänderung Δh .

Gleichung (2.4) wird die **Richards-Gleichung** genannt [LRic31]; sie beschreibt den Wasserfluß in gesättigten wie in ungesättigten Bodenzonen. Wie bei jeder Differentialgleichung müssen jedoch auch bei der Richards-Gleichung noch Randbedingungen vorgegeben werden, um eine eindeutige Lösung bestimmen zu können. Es existieren dazu vier verschiedene Arten von Randbedingungen:

1. Bei der **Dirichlet'schen Randbedingung** wird entweder das hydrostatische Potential h oder das totale Potential H am Rand $\partial\Omega$ des Gebietes Ω festgelegt:

$$h|_{\partial\Omega} = h_{D0}(t) \quad \text{oder} \quad H|_{\partial\Omega} = H_{D0}(t) \quad (2.6)$$

Die Dirichlet'sche Randbedingung ist eine Randbedingung erster Art.

2. Für die **Neumann'sche Randbedingung**, einer Randbedingung zweiter Art, wird für den Rand die Normalenableitung des totalen Potentials vorgegeben:

$$\vec{n} \cdot \nabla H|_{\partial\Omega} = H'_{N0}(t) \quad (2.7)$$

3. Die **Cauchy'sche Randbedingung**, ebenfalls eine Randbedingung zweiter Art, ähnelt der Neumann'schen Randbedingung; hier jedoch wird die Normalenableitung des hydraulischen Potentials vorbestimmt:

$$\vec{n} \cdot \nabla h|_{\partial\Omega} = h'_{C0}(t) \quad (2.8)$$

4. Die **variable Randbedingung** schließlich stellt eine Kombination der Randbedingungen erster und zweiter Art dar. Sie wird bevorzugt für die Grenzfläche Boden–Luft eingesetzt. Zu bestimmten Zeitpunkten werden am Rand Dirichlet'sche Randbedingungen angesetzt; zu den anderen Zeitpunkten sind Randbedingungen nach Cauchy angegeben.

Die Lösung einer Differentialgleichung kann bei ausschließlicher Angabe einer Randbedingung zweiter Art, wie der Neumann'schen oder der Cauchy'schen Randbedingung, nicht eindeutig bestimmt werden. Für mindestens einen Zeit- oder Ortspunkt muß eine Dirichlet'sche Randbedingung vorgegeben sein.

Mit der Richards- und Darcy-Gleichung, den Randbedingungen und den Bodeneigenschaften ergibt sich ein mathematisches Problem, das nur in sehr seltenen und zumeist unrealistischen Fällen geschlossen analytisch lösbar ist. Deshalb wird ein numerisches Rechenverfahren eingesetzt, mit der zumindest im Rahmen einer vorgegebenen Genauigkeit eine Näherungslösung gefunden werden kann. Zur numerischen Lösung eines solchen Differentialgleichungssystems existieren Standardverfahren; grundlegend für alle Standardverfahren ist dabei die Methode der finiten Elemente, die im folgenden kurz beschrieben werden soll.

2.2 Die Methode der Finiten Elemente

Im Bereich der Simulationsrechnung haben Finite-Elemente-Methoden (kurz FE-Methoden oder FEM) eine große Bedeutung erlangt. Mit ihrer Hilfe lassen sich viele Feldprobleme der Ingenieur- und Naturwissenschaften lösen. Ihre so überaus erfolgreiche Entwicklung begann zeitgleich mit dem Aufkommen der ersten großen Digitalrechner in den 60er Jahren [RClo60]. Zur Methode der finiten Elemente existiert mittlerweile eine umfangreiche Literatur [DBur88, KBat90, JRed93]. Hier sollen nur kurz die Grundzüge des Verfahrens skizziert werden.

Für die Methode der finiten Elemente wird ein Gebiet, auf dem eine kontinuierliche Funktion zu berechnen ist, auf eine Menge diskreter Punkte reduziert. Die Punktmenge bildet ein Gitter, in dem die Gitterpunkte zu Elementen zusammengefaßt werden. Im allgemeinen besitzen alle Elemente des Gitters die gleiche topologische Struktur; sie haben also die gleiche Form geometrischer Primitive, wie zum Beispiel Tetraeder, Hexaeder oder Prismen. Eine Grenzfläche eines Elementes ist gleichzeitig auch die Grenzfläche des benachbarten Elementes. Daher stammt die Bezeichnung "finit" (lat. *finire*, "durch Grenzen festlegen, durch Erklärungen bestimmen"). Mit einem hinreichend fein strukturierten Gitter der finiten Elemente lassen sich auch sehr komplexe Gebiete genau beschreiben.

Wenn ein Finite-Elemente-Gitter für ein Gebiet Ω aufgestellt ist, kann anschließend daran herangegangen werden, die Differentialgleichung der Form

$$A(u) = f \quad \text{in } \Omega, \quad (2.9)$$

mit A als linearen Differentialoperator auf u und f als einer vorgegebenen Funktion von u für jedes finite Element im Gebiet Ω numerisch als Näherung zu bestimmen. Dies geschieht mit Hilfe sogenannter Basisfunktionen Ψ_i , für die in der Regel Interpolationspolynome angesetzt werden. Die Punkte i der Elemente werden als Stützstellen für die Interpolation benutzt. Die Gesamtlösung

$$u_N = \sum_{i=1}^N u_i \Psi_i \quad (2.10)$$

als Näherung u_N über dem Gebiet Ω bestimmt sich dann aus den zusammengesetzten Näherungsfunktionen $u_i \Psi_i$ der einzelnen Gitterpunkte i , wobei Kontinuitätsgleichungen zwischen den Elementen erfüllt sein müssen. Für die Lösung u_N soll zusätzlich der Näherungsfehler minimal sein; um diese Forderung zu erfüllen, werden Variationsmethoden, wie zum Beispiel die Methoden nach Rayleigh-Ritz oder nach Galerkin eingesetzt, siehe [JRed93, KBat90]. Aus Gleichung (2.9) erhält man dann nach dem Ersetzen von u durch u_N , dem Ansetzen von Polynomen erster Ordnung als Basisfunktionen und mit der Variationsmethode ein Gleichungssystem der Form

$$\sum_{j=1}^N A_{ij} c_j = F_i, \quad (2.11)$$

das nach dem Vektor c aufzulösen ist. Seine Komponenten c_i sind die Koeffizienten der linearen Interpolationspolynome Ψ_i für jedes finite Element [RWim95, ONeu96].

Die Genauigkeit der Lösung u_N hängt entscheidend von der Dichte des Finite-Elemente-Gitters ab. Je dichter die Punkte i aneinander liegen, desto genauer wird die Näherung u_N . Andererseits steigt damit der Rechen- und Speicheraufwand, und folglich auch der Aufwand bei der Visualisierung der Lösung, wie später noch zu sehen sein wird.

Zumeist besteht das Problem in der Lösung von Differentialgleichungen für zwei oder drei Raumdimensionen. Bei manchen Problemstellungen interessiert jedoch auch das Zeitverhalten, so zum Beispiel bei transienten oder instationären Vorgängen. In diesen Fällen erweist es sich als vorteilhaft, auch die Zeit zu diskretisieren. Zu diesem Zweck kann die Methode der finiten Differenzen, auch FD-Methode genannt, herangezogen werden. Bei dieser Methode werden die Differentialquotienten der Variablen u nach der Zeit t in den Bestimmungsgleichungen durch finite Differenzenquotienten ersetzt:

$$\frac{\partial u}{\partial t} \approx \frac{u^{(n+1)} - u^{(n)}}{t^{(n+1)} - t^{(n)}} \quad (2.12)$$

Die Indizes (n) und $(n+1)$ bezeichnen zwei aufeinanderfolgende Zeitpunkte. Die Differenzengleichungen können nach einem zeitlichem Bezugspunkt mit einem

Wichtungskoeffizient ω gewichtet sein [ONeu96, RWim95]. Die Finite-Differenzen-Methode kann generell dort angewendet werden, wo Differentialquotienten in den Problemstellungen auftauchen; sie ist also nicht nur auf Differentiale nach der Zeit t beschränkt, sondern kann zum Beispiel auch für partielle Ableitungen nach dem Ort x eingesetzt werden.

Die beiden vorgestellten Methoden, die Finite-Elemente-Methode und die Finite-Differenzen-Methode, werden in der Beispielanwendung TRACE verwendet, um die Richards-Gleichung (2.4) zu lösen.

2.3 TRACE und ParTrace

TRACE

Mit dem Programmsystem TRACE (***T**ransport of **C**ontaminants in **E**nvironmental **S**ystems*) soll das Fließfeld des Wassers und die Ausbreitung von Schadstoffen simuliert werden. TRACE basiert auf zwei Programmen von Yeh, die am Oak Ridge National Laboratory Ende der 80er Jahre entwickelt wurden [GYeh87]. Seither wurden zahlreiche Erweiterungen und Modifikationen an TRACE vorgenommen; unter anderem wurde TRACE auf verschiedene Höchstleistungsrechner portiert, darunter auch auf die massiv-parallelen Rechner Intel Paragon und Cray T3D [HVer93], [RWim95]. Zum Programmpaket TRACE wurden mit der Zeit einige Tools hinzugefügt, wie Abb. 2.2 zeigt [ONeu96]. So wird mit dem Werkzeug Inpgen eine Eingabe-Datei für TRACE erstellt, die die Randbedingungen und die Bodenparameter enthält. Verschiedenste Merge-Programme fassen die verteilt vorliegenden Ergebnisse von TRACE zusammen. ParTrace kam in jüngster Zeit zum TRACE-Programmpaket hinzu; es wird im nächsten Abschnitt noch erklärt werden.

Die in Abschnitt 2.1 vorgestellte Richards-Gleichung (2.4) bildet die Grundlage aller Berechnungen in TRACE. Für die Berechnung nach der Methode der finiten Elemente wird sie zu einer Matrix-Gleichung umgeformt:

$$\mathbf{F} \cdot \frac{\partial}{\partial t} \vec{h} + \mathbf{K} \cdot \vec{h} = \vec{r}. \quad (2.13)$$

Im Vektor $\vec{r}$ sind der Quellen- und Senkenterm S sowie die Randbedingungen zusammengefaßt. Alle Vektoren dieser Matrixgleichung haben eine feste Länge von N Komponenten, mit denen die N Variablenwerte aller Knoten des Finiten-Elemente-Gitters beschrieben werden. Entsprechend liegen die Speichermatrix $\mathbf{F}$ und die Permeabilitätsmatrix $\mathbf{K}$ als $[N \times N]$ -Matrizen vor. Nach der zeitlichen Diskretisierung mit der Methode der Finiten Differenzen wird die Matrixgleichung (2.13) zusätzlich noch gewichtet, wie in [ONeu96, RWim95] beschrieben wird; die grundsätzliche Form aus Gleichung (2.13) ändert sich aber nicht. Das Gleichungssystem (2.13) ist nichtlinear, denn die Matrizen $\mathbf{F}$ und $\mathbf{K}$ sind abhängig vom Lösungsvektor $\vec{h}$. Zur Linearisierung des Gleichungssystems wird in TRACE das Picard-Verfahren benutzt [PHuy86]. Die so linearisierte Gleichung wird danach mit einem konjugierten

Der Verschiebungsvektor $\vec{x}'(t_0)$ beschreibt, um welche räumliche Distanz sich das Teilchen während des Zeitschritts Δt weiterbewegt hat. Mit der Diskretisierung der Zeit geht ein Fehler in die Rechnung ein, denn zur Modellierung der Teilchenbewegung werden nur die Kräfte zur Zeit t_0 herangezogen. Der Fehler wird aber umso kleiner, je kürzer der Zeitschritt Δt gewählt wird. Die Orte, an denen sich die Teilchen aufhalten können, sind beim Particle Tracking beliebig verteilt, im Gegensatz zur FE- und FD-Methode also nicht auf eine diskrete Punktemenge reduziert.

Die Stofftransportvorgänge sind jedoch noch nicht hinreichend genau charakterisiert, wenn ausschließlich das Strömungsfeld zur Berechnung herangezogen wird. Die Teilchenbahnen werden entscheidend auch durch den Effekt der Dispersion mitbestimmt. Die Dispersion entsteht durch das Zusammenwirken der Brown'schen Molekularbewegung (der Diffusion) mit dem Strömungsfeld. Weil das mikroskopische Strömungsfeld in den Bodenporen stark heterogen ist, haben die kleinen Bewegungen der thermischen Diffusion eine viel größere Auswirkung. Je grober die Leitfähigkeit im Boden verteilt ist, desto größer ist auch die Dispersion [ONeu96].

Die Dispersion geht nun als ein vom Zufall bestimmter Anteil in den Verschiebungsvektor $\vec{x}'$ ein, um den jedes Teilchen in ParTrace für jeden Zeitschritt verschoben wird. Um die Zahl der Teilchen in eine Konzentration umsetzen zu können, benutzt ParTrace das gleiche Finite-Elemente-Gitter wie TRACE. Für jeden Zeitschritt wird in jedem Element die Anzahl der enthaltenen Partikel gezählt. Mit dem jeweiligen Volumen des finiten Elements ergibt sich aus der Zahl der Partikel eine Konzentration c . Anders als TRACE, das Werte für die einzelnen Knoten des FE-Gitters berechnet, erhält man also aus ParTrace Werte für jedes finite Element.

Eine weitere wichtige Größe für die Modellierung der Stofftransportvorgänge ist, neben Fließfeld und Dispersion, die Sorption. Die Sorption ist ein Maß für die Fähigkeit des Bodens, die Partikel für eine bestimmte Zeit zu binden. Es gibt verschiedene Modellansätze zur Beschreibung der Sorption [ONeu96]; fast alle beschreiben die Sorption nichtlinear, wie zum Beispiel die Freundlich-Sorption. Hier wird das Verhältnis von gelösten zu sorbierten Teilchen in jedem finiten Element festgelegt zu

$$\frac{c_w}{c_s} = \frac{c_w}{k \cdot c_w^n} ; \quad (2.15)$$

gleichzeitig gilt

$$c = c_w + c_s , \quad (2.16)$$

also

$$c = c_w + k \cdot c_w^n , \quad (2.17)$$

die Konzentration gelöster Stoffpartikel läßt sich nur mit einem numerischen Verfahren ermitteln. Die Parameter k und n der Gleichung (2.17) werden, ähnlich wie die Permeabilität $\mathbf{K}$, als statistische Verteilung über dem gegebenen Gebiet modelliert. Eine detaillierte Darstellung dieser Sachverhalte und eine Erläuterung verschiedener Modelle der Sorption findet sich in [ONeu96].

2.4 Parallelität in TRACE und ParTrace

Die sequentielle Berechnung der Problemstellungen von TRACE oder ParTrace findet schnell ihre Grenzen. So muß beispielsweise für die realistische Nachbildung eines Gebietes der Größe $150\text{m} \times 50\text{m}$ bis in 10m Tiefe ein Finite-Elemente-Gitter mit etwa einer Million Knoten veranschlagt werden. Probleme dieser Größenordnung lassen sich selbst mit leistungsfähiger Rechner-Hardware nicht in annehmbarer Zeit lösen, wenn sequentielle Lösungsverfahren verwendet werden [RWim95].

Für beide Programme, TRACE und ParTrace, lassen sich Verfahren der Daten-Parallelisierung anwenden. Daten-Parallelisierung ist dadurch gekennzeichnet, daß die Aufgabe so aufgeteilt wird, daß "viele Prozessoren" eines Parallelrechners "mehr oder weniger die gleichen Operationen auf vielen Datenelementen" ausführen ([GAlm94, S. 239], *data parallelism*). Daten-Parallelisierung repräsentiert eine Form des SPMD-Modells (*single program, multiple data*): alle Prozessoren führen das gleiche Programm aus. Bei der Ausführung der Programmschritte sind sie jedoch nicht so aneinander gekoppelt, wie die Prozessoren des SIMD-Modells (SIMD: *single instruction, multiple data*; siehe [GAlm94, S. 240]).

Die Daten werden vor der Berechnung auf die Prozessorknoten aufgeteilt. Nur wenn während des Programmlaufs die Rechenlast dynamisch ausgeglichen werden soll oder es die Rechnung explizit erfordert, werden die Daten umverteilt. In TRACE wird eine Gebietszerlegung ausgenutzt, um das gegebene Gitter der finiten Elemente auf die Prozessoren aufzuteilen. ParTrace dagegen verteilt einzelne Datenobjekte zur Bearbeitung auf die Prozesselemente.

Im Gegensatz zur Daten-Parallelisierung steht die Kontroll- oder Funktionsparallelisierung. Bei der Funktionsparallelisierung wird die Gesamtaufgabe in gleichzeitig ausführbare, aber unterschiedliche Teilaufgaben aufgegliedert. Ein Beispiel für den Kontroll- oder Funktionsparallelismus (*control parallelism*) stellt die Parallelverarbeitung nach Art des Pipelining dar.

2.4.1 Gebietszerlegungen und Schwarz'sches Verfahren

Die Gebietszerlegung oder *domain decomposition* ist ein typisches Verfahren, um Problemstellungen, bei denen Differentialgleichungen mit der Finite-Elemente-Methode zu lösen sind, zu parallelisieren. Gebietszerlegungsverfahren sind Verfahren nach dem Prinzip "Teile und Herrsche" (*divide and conquer*). Das Gebiet, für das die Gleichung zu lösen ist, wird in mehrere Teilgebiete aufgeteilt. Jedes dieser Teilgebiete wird einem Prozessor zur Berechnung übergeben¹. Während der Berechnung werden geeignete Informationen ausgetauscht; nach der Berechnung wird die Gesamtlösung aus den Teillösungen zusammengestellt.

¹Es kann der Fall auftreten, daß die Anzahl der Prozessoren C kleiner als die Anzahl der Teilgebiete ist. Dann müssen einige oder sogar alle Prozessoren mehrere Teilgebiete sequentiell berechnen. Weil in dieser Arbeit die Resultate der Rechnung, die verteilten Datenfelder, behandelt werden, ist es ohne Belang, wieviele Prozessoren die Anzahl von P Datenfeldern berechnet haben. Deshalb wird im folgenden o.B.d.A. $C = P$ angenommen.

Eine Gebietszerlegung muß nicht unbedingt disjunkt im Sinne einer Partitionierung sein. Der Begriff der Partitionierung wird in der vorliegenden Arbeit wie folgt verstanden²:

Definition 1 (Partitionierung): Eine Anzahl P von Mengen $\mathcal{T}_p$ ($p = 1..P$) ist eine **Partitionierung** einer Gesamtmenge $\mathcal{T}$, wenn gilt:

1. $\mathcal{T}_p \subseteq \mathcal{T}$ und $\mathcal{T}_p \neq \emptyset$ für alle $1 \leq p \leq P$
 – alle Mengen $\mathcal{T}_p$ sind Teilmengen von $\mathcal{T}$ und nicht leer,
2. $\mathcal{T} = \bigcup_{p=1}^P \mathcal{T}_p$
 – die Mengen $\mathcal{T}_p$ decken die Gesamtmenge $\mathcal{T}$ vollständig ab,
3. $a \neq b$ ($1 \leq a \leq P, 1 \leq b \leq P$) $\Rightarrow \mathcal{T}_a \cap \mathcal{T}_b = \emptyset$
 – unterschiedliche Mengen $\mathcal{T}_a$ und $\mathcal{T}_b$ des Mengensystems von $\mathcal{T}$ haben unterschiedliche Elemente.

Die Mengen $\mathcal{T}_p$ werden **Partitionen** von $\mathcal{T}$ genannt.

Gebietszerlegungen brauchen Punkt 3 der Definition 1 nicht zu erfüllen; die Teilgebiete dürfen sich überlappen. Solche Teilgebiete werden im folgenden Domänen genannt, und eine Gebietszerlegung in einzelne Domänen wird hier als eine Domänen-Aufteilung definiert:

Definition 2 (Domänen-Aufteilung): Eine Anzahl D von Mengen $\mathcal{T}_d$ ($d = 1..D$) ist eine **Domänen – Aufteilung** einer Gesamtmenge $\mathcal{T}$, wenn gilt:

1. $\mathcal{T}_d \subseteq \mathcal{T}$ und $\mathcal{T}_d \neq \emptyset$ für alle $d = 1..D$ und
2. $\mathcal{T} = \bigcup_{d=1}^D \mathcal{T}_d$.

Die Mengen $\mathcal{T}_d$ heißen Domänen von $\mathcal{T}$.

Gerade die Überlappung von Teilgebieten wird bei einem besonderen Lösungsverfahren für Differentialgleichungen ausgenutzt, dem Schwarz'schen Verfahren. Das Schwarz'sche Verfahren wurde bereits Ende des vorigen Jahrhunderts von H. A. Schwarz vorgestellt [HSch90], freilich ohne zu wissen, wie gut sich das Verfahren mit den heutigen Parallelrechnern zur Lösung numerischer Probleme anwenden läßt.

Das Verfahren beruht auf der abwechselnden Berechnung der Lösung einer Differentialgleichung in sich überlappenden Teilgebieten. Zur Erläuterung sei die zweidimensionale Domänen-Aufteilung in Abb. 2.3 gegeben.

²In [IBro89][S. 547] wird in ähnlicher Weise die Zerlegung einer Menge definiert. Hier sei die "Zerlegung" als Oberbegriff sowohl zu "Partitionierung" als auch zu "Domänen-Aufteilung" (Definition 2) verstanden.

Das Gesamtgebiet Ω wurde in zwei Domänen Ω_1 und Ω_2 aufgeteilt. Der Teil des Randes $\partial\Omega_1$ von Ω_1 , der in das Teilgebiet Ω_2 hineinragt, aber nicht zu $\partial\Omega$ gehört, sei $\partial\Omega_{1(2)}$ genannt. Entsprechend bezeichne $\partial\Omega_{2(1)}$ den Teil des Randes $\partial\Omega_2$ im Teilgebiet Ω_1 . Die Ränder $\partial\Omega_{1(2)}$ und $\partial\Omega_{2(1)}$ werden Pseudo-Ränder genannt; die Werte auf diesen Rändern heißen Pseudo-Randwerte.

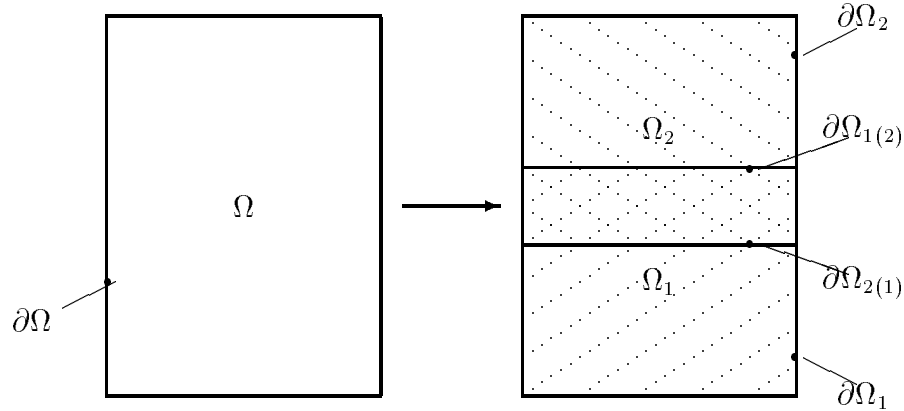


Abbildung 2.3: Zur Erläuterung des Schwarz'schen Verfahrens

Im ersten Schritt wird eine Lösung für das Teilgebiet Ω_1 berechnet. Dabei werden für $\partial\Omega_{1(2)}$ Schätzwerte als Randwerte angenommen. Der Rest von $\partial\Omega_1$ erschließt sich aus den Randwertvorgaben. Nachdem eine Lösung für Ω_1 gefunden wurde, wird im zweiten Schritt nach einer Lösung für das Teilgebiet Ω_2 gesucht. Dabei werden als Dirichlet'sche Randwerte für den Rand $\partial\Omega_{2(1)}$ die Werte des ersten Lösungsschrittes für Ω_1 benutzt. Die beiden Schritte werden mit den jeweils verbesserten Randwerten für $\partial\Omega_{1(2)}$ und $\partial\Omega_{2(1)}$ solange wiederholt, bis sich diese Randwerte im Rahmen einer gewünschten Genauigkeit nicht mehr ändern.

Für eine Parallelisierung ist das Schwarz'sche Verfahren in dieser alternierenden oder multiplikativen Form jedoch schlecht geeignet, denn in jedem Schritt wartet einer der Prozessoren auf die Ergebnisse des anderen Prozessors. Die Prozessoren arbeiten nie simultan. Deshalb wurde das multiplikative Schwarz'sche Verfahren zum additiven Verfahren variiert. Im ersten Schritt des additiven Verfahrens werden sowohl für $\partial\Omega_{1(2)}$ als auch für $\partial\Omega_{2(1)}$ Schätzwerte vorgegeben. Dann werden beide Teilgebiete unabhängig voneinander berechnet. Sind die Lösungen für die beiden Teilgebiete bestimmt, werden die gefundenen Werte für $\partial\Omega_{1(2)}$ und $\partial\Omega_{2(1)}$ als Dirichlet'sche Randwerte für den nächsten Schritt ausgetauscht. Die beiden Schritte "Berechnung einer Lösung, Austausch der Pseudo-Randwerte" werden auch hier solange wiederholt, bis eine vorgegebene Genauigkeit in den Randwerten erreicht ist. Allerdings konvergiert das additive Schwarz'sche Verfahren langsamer als das multiplikative Verfahren; nach Bjørstad benötigt es jedoch höchstens zweimal so viele Iterationen

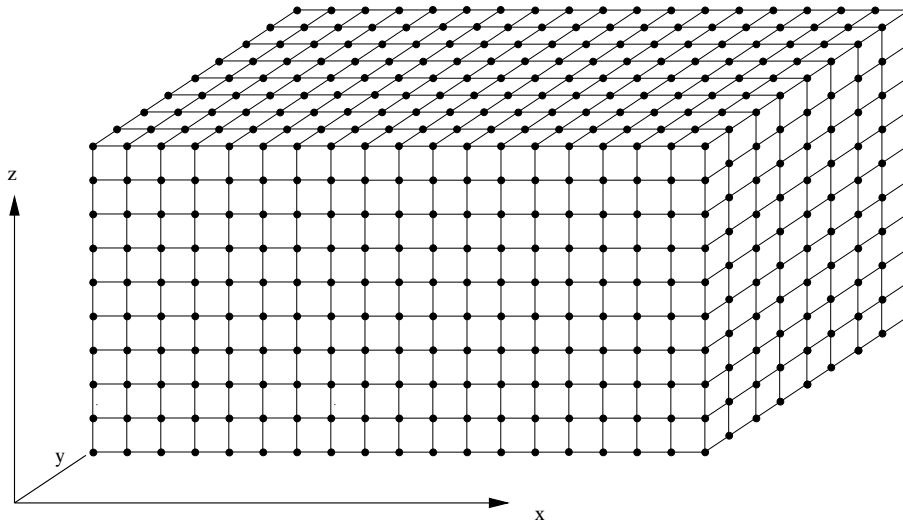


Abbildung 2.4: Streng reguläres Gitter finiter Elemente in TRACE

wie das alternierende, multiplikative Verfahren [PBjø89]. Der große Vorteil des additiven Verfahrens ist, daß es sich vollständig parallelisieren läßt.

Es soll nun die Domänen-Aufteilung für das Schwarz'sche Verfahren näher untersucht werden, wie sie das Programm TRACE in der jetzigen, parallelen Version verwendet. Der Einfachheit halber sei an dieser Stelle ein streng reguläres Gitter angenommen, wie es später noch in Abschnitt 4.3.1 auf Seite 43 vorgestellt wird. Ein streng reguläres Gitter besteht aus $N_x \times N_y \times N_z$ Knoten, die in regelmäßigen Abständen längs einer der drei Raumrichtungen angeordnet sind. Das Gitter ist dreidimensional und orthogonal. Die finiten Elemente setzen sich aus jeweils acht benachbarten Knoten zusammen und haben die Form kleiner Würfel. Abb. 2.4 zeigt ein Beispiel für ein solches streng reguläres Finite-Elemente-Gitter.

Das Gitter der finiten Elemente wird nun durch Bänder in Domänen aufgeteilt. Jedes Band g hat eine Breite b_g von einigen finiten Elementen (b_g ganzzahlig und ≥ 0) und erzeugt Überlappzonen dieser Breite von b_g Elementen zwischen den Teilgebieten. Die Knoten des Finite-Elemente-Gitters in diesen Überlappzonen gehören den Domänen zu beiden Seiten des Bandes an. Die Bänder sind immer nach ganzen Elementen ausgerichtet, folglich ist ein b_g Elemente breites Band $b_g + 1$ Knoten breit. Die Bänder liegen bei einem orthogonalen FE-Gitter immer senkrecht zu einer der drei Raumachsen und ändern ihre Orientierung und ihre Breite im Gitter nicht. Jedes Band umwickelt vollständig das gesamte Gebiet. Bänder gleicher Orientierung überlappen sich nicht gegenseitig. Eine durch solche Überlappbänder erzeugte Domänen-Aufteilung für das FE-Gitter in Abb. 2.4 zeigt Abb. 2.5.

Für eine numerische Berechnung eines FE-Problems nach dem Schwarz'schen Verfahren auf einem gegebenen Finite-Elemente-Gitter existiert eine optimale Domänen-Aufteilung. Zur Optimierung lassen sich drei Kriterien angeben:

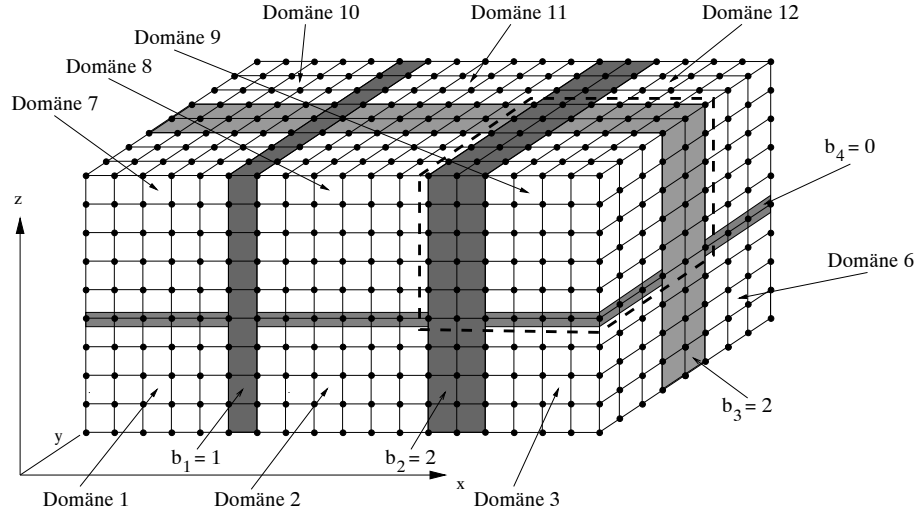


Abbildung 2.5: Aufteilung eines Gebietes in Domänen (*domain decomposition*)

1. Die Rechenlast soll gleichmäßig auf die Prozessoren aufgeteilt werden.
2. Es sollen möglichst geringe Kommunikationskosten beim Austausch der Pseudo-Randwerte entstehen.
3. Es sollen möglichst wenig Knoten in den Überlappzonen liegen, weil für sie die Lösung redundant berechnet werden.

In der vorliegenden Version von TRACE wird als Aufteilungsstrategie die sogenannte K-Funktion optimiert. Die K-Funktion ist das Verhältnis der Anzahl der Knoten, wie sie zur Berechnung herangezogen werden, zur Anzahl der Knoten im FE-Gitter bei einfacher Zählung:

$$K = \frac{\sum_{d=1}^D N_d}{N} \quad , \quad (2.18)$$

mit N_d : Anzahl der Knoten der Domäne d ($1 \leq d \leq D$) und N : Anzahl der Knoten im gesamten FE-Gitter; es ist $N = N_x \cdot N_y \cdot N_z$. Weil sich die Domänen überlappen können, gilt $K \geq 1$. In TRACE wird von einer festen Anzahl Domänen (oder Prozessoren) D und von einer festen Breite b aller Schnitte ausgegangen [RWim95]. Die Optimierung der K-Funktion läuft dann über alle Faktorzerlegungen von D mit $D = D_x \times D_y \times D_z$, mit ganzzahligen $D_i \geq 1, i = x, y, z$. Es wird dabei nach derjenigen Domänen-Aufteilung gesucht, für die K minimal ist.

In späteren Versionen soll TRACE auch auf verteilten Systemen unterschiedlicher Höchstleistungsrechner laufen [Meta95]. Für die Portierung auf heterogene verteilte Systeme wird auch eine neue Zerlegungsstrategie erforderlich werden, denn sowohl die Rechenleistung der einzelnen Prozessorknoten als auch die Kommunikationskosten zwischen den Prozessoren unterscheiden sich von Rechenknoten zu Rechenknoten. Die Teilgebiete können jedoch einstweilen noch als quaderförmig angenommen

werden (damit bleibt der Verwaltungsaufwand klein); allerdings müssen dann die Überlappbänder nicht mehr unbedingt das gesamte Gebiet umwickeln.

Für Berechnungen auf solchen heterogenen Systemen bedarf es anderer, allgemeinerer Optimierungsstrategien zur Aufteilung des Gitters in Domänen. Lösungen für diese Zerlegungsprobleme lassen sich vermutlich aus dem Ansatz der Rekursiven Spektralen Bisektion ableiten, wie sie bereits zur Partitionierung irregulärer FE-Gitter verwendet wird (siehe dazu [SBar94]). Die Optimierung derartiger Gebietszerlegungen ist jedoch noch Gegenstand der Forschung.

2.4.2 Gemischte Speicherverwaltung

Für ParTrace findet sich zunächst keine so anschauliche geometrische Entsprechung der Daten-Parallelisierung, wie im Fall der *domain decomposition* in TRACE. Die Daten-Parallelisierung in ParTrace liegt in der Verteilung von abstrakten Datenobjekten auf die einzelnen Prozessorknoten.

ParTrace wurde objektorientiert in C++ geschrieben [ONeu96]. Die mathematischen Objekte des Boden-Modells wurden in Datenobjekte umgesetzt. So gibt es unterschiedliche Objektklassen für finite Elemente, Moleküle und Teilchen. Die Molekül-Objekte beschreiben die chemischen und physikalischen Eigenschaften der Teilchen, beispielsweise mit Sorptionsparametern oder mit Parametern, die die nichtlinearen chemischen Reaktionen dieses Molekültyps charakterisieren. Die Molekül-Objekte wie auch die Element-Objekte sind statische Objekte, sie verändern sich während des Programmlaufs nicht und müssen nur einmal instanziiert werden.

Die Teilchen-Objekte sind die Objekte, die sich durch das Gitter der finiten Elemente bewegen können. Typischerweise rechnet ParTrace mit etwa einer Million solcher Teilchen. Im Gegensatz zu Molekülen oder finiten Elementen sind Teilchen dynamisch: sie bewegen sich, sie ändern ihren Molekültyp, und sie entstehen und vergehen. Um den enormen Verwaltungsaufwand zu umgehen, der entstünde, wenn für jedes Teilchen im Laufe seines Lebens Speicherplatz alloziert, umgelegt und freigegeben werden müßte, wurden Blöcke eingeführt, in denen eine bestimmte Anzahl Teilchen zusammengefaßt und abgespeichert werden können. Zu Beginn werden die Blöcke auf den Rechnerknoten verteilt alloziert; während der Rechnung variiert die Zahl der Teilchen innerhalb der Blöcke. Alle Teilchen eines Blocks gehören dem gleichen Molekültyp an. Eine weitere Objektklasse verwaltet den Wechsel von Teilchen zu anderen Blöcken, falls sich ihr Typ bei einer Reaktion ändert. Die Parallelisierung in ParTrace liegt damit in der Verteilung der Teilchen-Blöcke auf die einzelnen Rechnerknoten.

Die in dieser Arbeit vorgestellten Konzepte zur Visualisierung verteilt berechneter Datenfelder können zwar auch für diese Art der Parallelisierung genutzt werden; dazu müssen die verteilten Datenobjekte, die Blöcke aus ParTrace, allerdings in geeignete Datenfelder umgewandelt werden. Beispielsweise könnten die Daten aller Teilchen eines Blocks als ein Datenfeld aufgefaßt werden. Als geeignete Datenstruktur käme eine eindimensionale Feldstruktur, also eine Liste, in Betracht (siehe Abschnitt 4.3.1).

Der Schwerpunkt wird im weiteren auf verteilte Datenfelder im Sinne einer *domain decomposition* gelegt.

2.5 Problematik der Darstellung

Wie in diesem Kapitel gezeigt wurde, vollzieht sich die Simulation der Stofftransportvorgänge bislang in drei Schritten (vergleiche auch mit Abb. 2.2):

1. **Pre-Processing:** Ein hinreichend feines Finite-Elemente-Gitter wird als Abbild des zu untersuchenden Gebietes angefertigt. Aus Dichtespektren und Korrelationslängen wird für die Knotenmenge dieses FE-Gitters ein Bodenmodell konstruiert, das die statistische räumliche Verteilung der hydraulischen Leitfähigkeit $\mathbf{K}$ und Sorptionsparameter (zum Beispiel k und n bei einer Freundlich-Sorption nach Gleichung (2.17)) nachbildet.
2. **Berechnung des Fließfeldes:** Das Gesamtgebiet wird in Teilgebiete unterteilt. TRACE berechnet parallel für jedes Teilgebiet aus der diskretisierten Richards-Gleichung das hydrostatische Potential h . Die Lösungen der einzelnen Teilgebiete werden nach dem Schwarz'schen Verfahren berechnet. Mit Hilfe der Darcy-Gleichung erhält man das Strömungsfeld des Wassers für jedes Teilgebiet; durch Zusammenfügen der verteilten Ergebnisdaten erhält man die Lösung für das Gesamtgebiet.
3. **Berechnung der Konzentrationsverteilung:** ParTrace berechnet mit dem Particle-Tracking-Verfahren die Verteilung der Stoffkonzentration. Dazu erhält es von TRACE das Strömungsfeld des Gesamtgebietes und benutzt ein vom Benutzer bestimmtes Modell der Sorption.

In dieser Abfolge fehlt noch das **Post-Processing**. In diesem wichtigen Schritt würden die Ergebnisse aus TRACE und ParTrace aufbereitet und grafisch dargestellt werden. Erst mit einer Visualisierungskomponente können die Ergebnisse interpretiert und für etwaige Modellkorrekturen in TRACE oder ParTrace verwendet werden.

Die Visualisierungskomponente sollte im Post-Processing alle von TRACE oder ParTrace berechneten Felder mit verschiedenen Methoden der Visualisierung darstellen können. Gleichzeitig sollte sie aber auch für das Pre-Processing einsetzbar sein, damit beispielsweise die stochastisch generierten Verteilungen der Permeabilität und der Sorptionsparameter vor der Berechnung geprüft werden können.

Weitere Wünsche ergeben sich aus der Verteilung der Datenfelder: es sollte möglich sein, die verteilten Datenfelder zusammenzustellen, als auch einzeln zu visualisieren (eventuell könnten kleinere Workstations mit der Aufgabe, die gesamte Datenmenge zu visualisieren, überfordert sein, insbesondere, was die Interaktivität angeht).

Die physikalischen Phänomene in den einzelnen Domänen sollen mit den Phänomenen in den Nachbardomänen verglichen und Differenzmengen in den Überlappgebieten erstellt werden können. Zur eigentlichen Visualisierung sollten dabei umfangreiche Methoden zur Verfügung gestellt werden, zum Beispiel eine Vektorpfeil- oder Stromlinien-Darstellung bei Vektorfeldern, oder Profilschnitte und Volumendarstellungen bei skalaren Datensätzen. Dabei kommt den Farbtabelle eine wichtige Bedeutung zu, über die Skalarwerte als Falschfarben auf einer Geometrie, wie zum Beispiel einem Profilschnitt, dargestellt werden können. Weiterhin wäre es wünschenswert, bei zeitabhängigen Problemen eine Animation erstellen zu können. Mit einer Animation ließe sich die volle Bandbreite des Systems Auge–Hirn für das Erfassen der Dateninhalte nutzen.

Mögliche Werkzeuge, mit denen eine solche Visualisierungskomponente geschaffen werden könnte, werden im nächsten Kapitel vorgestellt. Mit keinem dieser Visualisierungswerkzeuge lassen sich jedoch die vorliegenden verteilt berechneten Datenfelder aus TRACE unmittelbar darstellen; alle Werkzeuge müssen mehr oder weniger aufwendig an die parallel erzeugten Datenstrukturen angepaßt werden.

Verteilte Datenfelder im Sinne einer *domain decomposition* lassen sich nicht nur in TRACE finden, sondern auch in einer Vielzahl anderer paralleler Anwendungen auch. Es ist daher von generellem Interesse, ein Visualisierungswerkzeug zu entwickeln, das gerade die Erfordernisse der Visualisierung parallel berechneter, wissenschaftlicher Daten berücksichtigt.

Weil sich die Größe eines Datensatzes einer Simulation aus den Ausmaßen in fünf Dimensionen zusammensetzt (drei Raumdimensionen, eine Zeitdimension, und eine Variablendimension, in der alle abhängigen Variablen der Simulation hintereinander aufgeführt werden), ergibt sich die Größe der Datenmenge als ein Produkt von fünf Faktoren. Das auszuwertende Datenvolumen wird daher sehr schnell größer, wenn ein oder mehrere Faktoren gleichzeitig anwachsen – so zum Beispiel, wenn die Auflösung eines Finite-Elemente-Gitters in alle drei Raumrichtungen erhöht wird, oder wenn die zeitliche Entwicklung eines Phänomens dargestellt werden soll. Unter diesem Gesichtspunkt erscheint auch dann eine Aufteilung des Datensatzes in mehrere kleine Teilgebiete zur Visualisierung und Bearbeitung sinnvoll, selbst wenn zur Berechnung kein Datenparallelisierungsverfahren benutzt wurde.

Kapitel 3

Ansätze und Werkzeuge

In diesem Kapitel soll untersucht werden, welche Ansätze und Werkzeuge zur Lösung des in Kapitel 2 umrissenen Problems zur Erfassung und Interpretation der verteilten Simulationsergebnisse bestehen. Die Visualisierung nutzt die hohe Bandbreite des Systems Auge–Hirn aus, um eine große Menge an Daten kompakt und schnell an den Endanwender zu vermitteln; sie kann also in diesem Zusammenhang zu einer Datenkompression genutzt werden. Eine *geschickte* wissenschaftliche Visualisierung lenkt zusätzlich das ”Augenmerk” des Benutzers auf die wichtigen Aspekte in seinem Datensatz. Was in diesem Zusammenhang ”geschickt” bedeutet, wird in Abschnitt 4.1 noch genauer formuliert und anschließend in den Kapiteln 4 und 5 untersucht werden. Zuvor wird jedoch noch erläutert werden müssen, was sich hinter dem Begriff ”Wissenschaftliche Visualisierung” selbst verbirgt und welche Ansätze und Werkzeuge zur Visualisierung bereits bestehen.

3.1 Wissenschaftliche Visualisierung

Die Visualisierung wissenschaftlicher Daten beschäftigt sich mit der Umsetzung von Daten in graphische Darstellungen [REar92]:

”Scientific Visualization is concerned with exploring data and information in such a way as to gain understanding and insight into the data.”

Visualisierung ist also zielgerichtet, mit ihr sollen Einblicke in die Daten gewonnen werden. Die hier betrachteten Daten sind Ergebnisse aus Simulation oder Experiment. Damit trägt die Visualisierung entscheidend dazu bei, die Sachverhalte in der Natur zu verstehen, die Gegenstand des Experiments oder der Simulation sind. Sie ordnet sich deshalb in den Zyklus des wissenschaftlichen Arbeitens ein; die Visualisierung ist eine Form der Datenaufbereitung (siehe Abb. 3.1).

Entsprechend wird die Visualisierung in vielfältigen Bereichen wissenschaftlicher Forschung angewandt: zum Beispiel in der Teilchenphysik zur Darstellung von Ergebnissen aus Kollisionsversuchen, in der Mechanik bei der Untersuchung des Fließverhaltens granularer Medien, oder in Meteorologie und Klimaforschung zur Untersuchung des menschlichen Einflusses auf globale Klimaveränderungen. Darüber

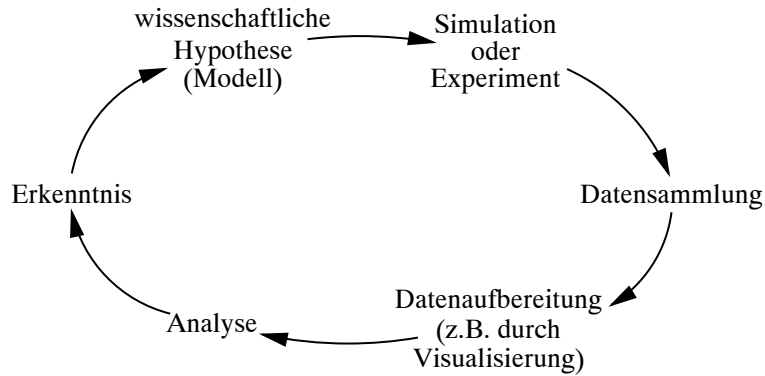


Abbildung 3.1: Der Zyklus des wissenschaftlichen Arbeitens

hinaus werden natürlich auch überall dort die Methoden der Visualisierung wissenschaftlicher Daten eingesetzt, wo große, aber weitgehend homogene Datenmengen anfallen und ausgewertet werden müssen. Beispiele hierfür sind in der Medizin die Computertomographie, in der Geologie die Ressourcen- und Bodenerkundung, oder in der Meteorologie die Auswertung von Wetterdaten.

Der zur Visualisierung umgekehrte Vorgang, also die Umsetzung graphischer Darstellungen in Daten, wird Computer Vision genannt [BMcC87, SOrt93]. Visualisierung und Computer Vision stehen bei vielen Anwendungen in engem Wechselspiel – beispielsweise in der Meteorologie, wenn die aus Satellitenbildern gewonnen Informationen dazu genutzt werden, um daraus dreidimensionale Modelle für die Wettervorhersage zu konstruieren.

Obwohl die computergestützte Visualisierung eine sehr junge Disziplin ist, existieren inzwischen zahlreiche Erklärungsansätze, Paradigmen und Datenmodelle zur Visualisierung (siehe dazu [KBro92, REar93, LRos94]). Die für die vorliegende Arbeit wichtigsten Ansätze werden im folgenden zusammengefaßt dargestellt.

3.2 Ansätze zur Visualisierung

Das Konzept zur Visualisierung verteilter Datenfelder, das in Kapitel 4 vorgestellt wird, baut auf drei Ansätzen zur Visualisierung auf: dem Datenflußparadigma von Upson [CUps89], dem modellbasierten Ansatz nach Brodlie [KBro93] und dem Gittermodell von Hibbard [WHib94a].

3.2.1 Datenflußparadigma

Das Datenflußparadigma ist das Prinzip, nach dem die meisten heutigen Visualisierungswerkzeuge arbeiten. Es wurde von C. Upson 1989 vorgestellt und wurde

erstmalig im Visualisierungssystem AVS (Application Visualization System) implementiert [CUps89]. Weitere Ausarbeitungen zu diesem Paradigma finden sich bei [RHab90].

Nach dem Datenflußparadigma werden die darzustellenden Daten sequentiell in einer sogenannten Visualisierungspipeline bearbeitet (siehe Abb. 3.2). Die drei Schritte heißen **Filtering**, **Mapping** und **Rendering**.

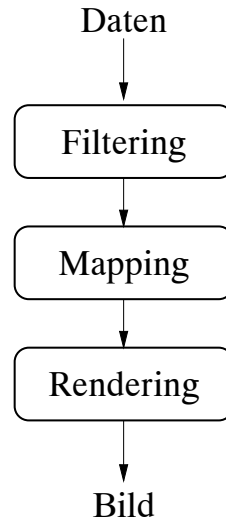


Abbildung 3.2: Das Datenflußparadigma

Filtering

Im ersten Schritt, dem Filterschritt, werden die interessierenden Daten aus dem ursprünglichen Datensatz abgeleitet. Beispiele wichtiger Filter-Operationen sind:

- Verschieben oder Skalieren des Datensatzes,
- Statistische Operationen, wie zum Beispiel die Erstellung eines Histogramms aus dem Datensatz oder die Bildung statistischer Werte wie Mittelwert oder Varianz.
- Grundlegende mathematische Operationen; dazu gehören unäre Feld-Operationen, wie zum Beispiel die Gradientenbildung auf einem Skalarfeld, oder die Addition oder Multiplikation der Feldwerte mit einer Konstanten. Aber auch binäre mathematische Feld-Operationen, zum Beispiel die Addition oder Multiplikation zweier Datensätze, gehören zu dieser Gruppe der Filter-Operationen.
- Anwenden einer Dichtefunktion auf den Datensatz. Mit der Dichtefunktion können bestimmte Wertebereiche des Datensatzes hervorgehoben oder herausgefiltert werden.

- Selektion einzelner Teile des Datensatzes. Dazu zählen Querschnitte oder blockförmige Ausschnitte, aber auch die Selektion eines Zeitschrittes oder die Selektion einer Variable aus einer Variablenstruktur. Die getroffene Auswahl braucht nicht zusammenhängend über die fünf möglichen Dimensionen verteilt zu sein; beispielsweise ist eine nützliche Filter-Operation die Reduktion des Datensatzes auf jedes n . Datenelement.
- Approximation unsicherer Daten, insbesondere wenn die Daten aus einem Experiment stammen.
- Interpolation zwischen vorgegebenen Stützpunkten. Eine Funktion wird in zwei Schritten interpoliert: der erste Schritt ist die Injektion, also die Erweiterung des Definitionsbereichs der Funktion, und der zweite Schritt besteht in der Mittelung der Werte zwischen den Stützstellen (siehe auch [SBar94]).

Mehrere Filter-Operationen können hintereinander ausgeführt werden.

Mapping

Der zweite Schritt konstruiert eine geometrische Repräsentation aus den gefilterten Daten. Je nach Struktur der erzeugten Geometrien können Mapping-Operationen in zwei Klassen eingeteilt werden: in die Klasse der Glyph-Mapper und in die Klasse der kontinuierlichen Mapper.

Beim Glyph-Mapping werden alle Werte des zuvor gefilterten Datensatzes in geometrische Ausprägungen sogenannter Glyphen umgesetzt. Glyphen sind kleine¹ geometrische Objekte, die einen Satz von Eigenschaften haben, wie zum Beispiel Position im Raum, Farbe, Textur, oder Größe. Für gewöhnlich wird die Eigenschaft "Position", die alle Glyphen haben, dazu benutzt, um jeden Ort im Raum mit einem Glyphen zu markieren, für den ein Wert im gefilterten Datensatz vorhanden ist.

Ein Beispiel für Glyphen sind kleine Vektorpfeile, mit denen der Betrag und die Richtung eines Vektorfeldes an bestimmten Punkten repräsentativ dargestellt werden kann. Ein weiteres Beispiel sind kleine Kugeln, deren Radius oder Farbe sich nach der Größe des Skalarfeldes an der gewählten Stelle richtet. Weitere Beispiele für Glyphen finden sich in [JFol94].

Kontinuierliches Mapping dagegen erzeugt große, zusammenhängende geometrische Objekte, denen ebenfalls Eigenschaften wie Farbe, Textur oder Abmessungen zugeordnet werden können. Bei diesen Objekten werden allgemein im Sinne einer Interpolation Orte gleicher Eigenschaften verbunden. Ein Beispiel für ein solches Mapping ist der Profilschnitt durch einen Datensatz – er verbindet Orte, für die eine Ebenengleichung $ax + by + cz + d = 0$ mit festen a, b, c und d erfüllt ist. Ein weiteres Beispiel ist die Stromlinie oder die Raumkurve, die ein Partikel in einem Vektorfeld durchläuft. Auch Isoflächen oder Isolinien gehören zu den kontinuierlichen

¹Wenn in diesem Zusammenhang von "klein" oder "groß" gesprochen wird, so ist damit "klein" bzw. "groß" gegenüber den Ausmaßen der räumlichen Darstellung des gesamten Datensatzes gemeint.

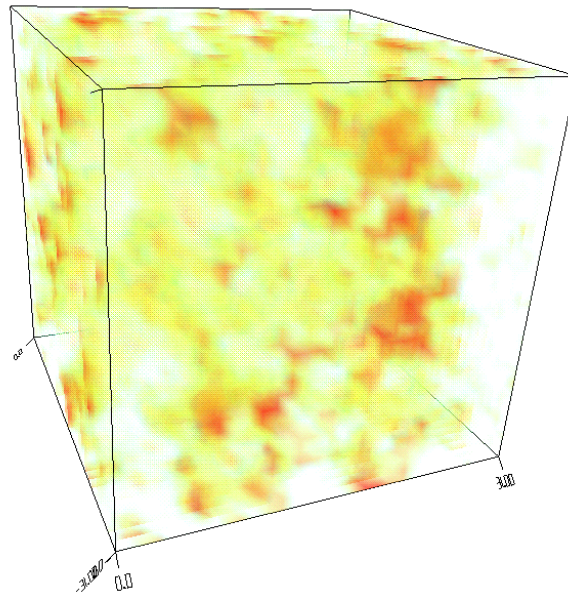


Abbildung 3.3: Beispiel für den Übergang von Glyphen-Mapping zu kontinuierlichem Mapping: Volumenvisualisierung

Mappern; sie verbinden Orte mit gleichen (Skalar-)Werten. Für das kontinuierliche Mapping wird immer mindestens ein Parameter benötigt, der bestimmt, welche Orte die Geometrie verbinden soll. Für eine Isofläche ist dies der sogenannte *Iso-Wert*, für den Schnitt durch den Datensatz müssen die vier Parameter a, b, c und d festgelegt werden.

Weil einerseits Glyphen sehr dicht aneinander platziert werden können, aber andererseits auch kontinuierliches Mapping diskrete Geometrien erzeugt, gibt es Mapping-Methoden, die sowohl dem Glyph-Mapping als auch dem kontinuierlichen Mapping zugeordnet werden können. Die Volumenvisualisierung ist dafür ein Beispiel, siehe Abb. 3.3. Hierbei besteht die Geometrie aus vielen kleinen Voxeln, denen Eigenschaften wie Position, Farbe und Transparenz zugeordnet sind. Voxel haben also Glyph-Eigenschaften. Gleichzeitig ist die Geometrie "lückenlos" aus den Voxeln zusammengesetzt und bildet für den Betrachter nach dem Rendering eine kontinuierliche Geometrie, in denen Gebiete gleicher Eigenschaften als zusammenhängend erkannt werden können. Ein ähnlicher Übergang vom Glyphen-Mapping zum kontinuierlichen Mapping lässt sich ebenfalls im Konzept der Streamballs finden, siehe dazu [RCra95].

Rendering

Der letzte Schritt in der Visualisierungspipeline besteht in der Gewinnung eines (meist) zweidimensionalen Bildes aus der im Mapping-Schritt erstellten Geometrie. Nach [JEnc93] lassen sich heutzutage drei Rendering-Techniken unterscheiden, die

sich alle aus einem Hidden-Surface-Algorithmus und einem Shading-Modell zusammensetzen.

Der Hidden-Surface-Algorithmus entfernt diejenigen Flächen, Linien und Punkte aus dem Abbild, die für den Betrachter in der zweidimensionalen Projektion der Geometrie nicht zu sehen sind. In diesem Algorithmus werden auch Effekte wie Depth-Cueing, Perspektive und Blickwinkel berücksichtigt, mit denen eine verbesserte Tiefenwirkung erzielt wird.

Das Shading-Modell bezieht die Beleuchtung der Geometrie und seine Oberflächeneigenschaften wie Glanz, Transparenz, oder Textur in die zweidimensionale Abbildung mit ein. Die gängigen Shading-Modelle sind heute das Flat-Shading, und das Phong-, sowie das Gouraud-Shading. Sehr einfache Shading-Modelle, wie zum Beispiel das Drahtgittermodell, brauchen die Oberflächeneigenschaften nicht zu berücksichtigen; deshalb kann schnell mit ihnen interagiert werden. Dies wird in vielen Visualisierungswerkzeugen mit einer *bounding box* ausgenutzt: wenn der Benutzer die Geometrie bewegen, drehen oder skalieren will, wird auf ein Drahtgittermodell umgeschaltet, solange die Maus bewegt wird. Erst bei Stillstand der Maus wird das aufwendigere Shading-Modell wieder eingesetzt.

Im Laufe der Zeit wurden Rendering-Techniken entwickelt, die dreidimensionale Szenen in sehr realistisch erscheinende zweidimensionale Abbilder umsetzen können. Der zuerst entwickelte Rendering-Ansatz war die projektive Methode, in der die Gesamtszene auf die Bildebene als perspektivische Projektion abgebildet wird. Beim Ray-Tracing wird umgekehrt eine Projektion der Punkte von der Bildebene in die Szene hinein untersucht, um Aufschluß über das Aussehen des Abbildes zu erhalten. Der Ansatz mit der bislang besten Realitätstreue ist der Radiosity-Ansatz, bei dem Energiebetrachtungen über Emmission, Reflektion und Absorption des Lichts zwischen den Objekten der Szene angestellt werden. Diese Technik ist allerdings auch die aufwendigste von allen drei Rendering-Methoden. Nähere Erläuterungen und Verweise auf die Entwickler der drei Rendering-Techniken finden sich in [JEnc93].

Das beim Rendering erzeugte Abbild der Geometrie muß nicht unbedingt ein einziges zweidimensionales Bild sein; die Erzeugung stereoskopischer Bilder gehört ebenfalls zur Klasse der Rendering-Operationen. Stereoskopie erzeugt naturgemäß den besten Tiefeneindruck einer Szene und ist damit am besten geeignet, um komplexe dreidimensionale Strukturen wiederzugeben.

Diese drei Schritte Filtering, Mapping und Rendering ergeben zusammen die Visualisierungspipeline. In dieser Visualisierungspipeline werden Daten nach dem in der Datenverarbeitung allgemein bekannten EVA-Prinzip (Eingabe, Verarbeitung, Ausgabe) bearbeitet. Das Datenflußparadigma beschreibt damit ein geeignetes Implementationsmodell für Visualisierungswerkzeuge; Daten bilden die Basis des Visualisierungsprozesses.

Die große Schwäche des Datenflußparadigmas besteht jedoch gerade in der Ausrichtung auf die bloße Verarbeitung von Daten. Wissenschaftler, die ihre Erkenntnisse

aus der Visualisierung gewinnen wollen, wünschen weniger eine Kompression, Zusammenfassung oder ähnlich unmittelbare Aufarbeitung ihrer Daten. Viel aussagekräftiger ist für sie die Darstellung eines Feldes aus ihrer physikalischen Vorstellung, in dem die interessierenden und darzustellenden Phänomene stattfinden. Von solchen Modellfeldern wird im modellbasierten Ansatz ausgegangen, der im folgenden vorgestellt wird.

3.2.2 Modellbasierter Ansatz

Im modellbasierten Ansatz von Ken Brodlie [KBro93] werden die Daten, seien sie exakt oder fehlerbehaftet, ausschließlich dazu benutzt, um ein Modellfeld (*underlying field*) zu konstruieren, siehe Abb. 3.4. Das Modellfeld soll der Anschauung des Benutzers von dem in Simulation oder Experiment untersuchten Feld entsprechen; die gewonnenen Daten sollen als abgetastete und quantisierte Werte dieses Feldes aufgefaßt werden. Für das in Simulation oder Experiment untersuchte Feld wird ein Zusammenhang der Art

$$Y = f(X) \quad (3.1)$$

zwischen einer unabhängigen Variablen X und einer abhängigen Variablen Y angesetzt. Soll also das untersuchte Feld als Modellfeld nachgebildet werden, so müssen die vorliegenden Daten in einem Modellerschritt (*modelling step*) zu den Variablen X und Y umgeformt werden. Typische Modellerschritte sind die Interpolation – denn das Modellfeld soll kontinuierlich beschrieben sein – und die Approximation, wenn die Daten unsicher vorgegeben sind.

Nachdem das Modellfeld konstruiert ist, wird davon eine Ansicht (*view*) erzeugt. Dazu wird eine geeignete Visualisierungsmethode ausgewählt. Welche Methode in diesem Zusammenhang "geeignet" ist, wird im allgemeinen vom Benutzer entschieden; die Visualisierungsmethode könnte aber auch, abhängig von Dateninhalt und -umfang, von regelbasierten Systemen vorgeschlagen werden.

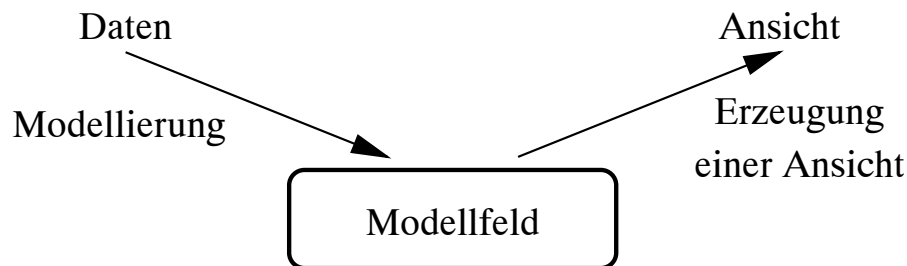


Abbildung 3.4: Modellbasierter Ansatz

Zur Implementation eignet sich der modellbasierte Ansatz jedoch nur bedingt. Dafür gibt es zwei Gründe. Zum einen können die Daten aus Simulationen oder Experimenten stammen, für die kein direkter Zusammenhang mit einem physikalischen Feld existiert. Beispiele dafür sind die Visualisierung soziologischer bzw. wirtschaftlicher

Daten oder die Visualisierung von Performance-Daten verteilter Rechnersysteme. In diesem Fall müssen die Daten auf ein imaginäres, künstliches Modell abgebildet werden [SHac94], dessen Ausgestaltung aber für jeden Anwendungsfall neu entwickelt werden muß.

Das andere Problem, den modellbasierten Ansatz zu einer Implementierung zu nutzen, liegt in der geforderten Kontinuität des Modellfeldes. Digitale Datenstrukturen können bestenfalls eine gute Annäherung an ein Feld-Kontinuum bieten. Gute Annäherungen kosten jedoch Rechenzeit bei der Modellierung, wie auch beim Erzeugen der Ansicht. Deswegen wird man auch hier für jeden Einzelfall entscheiden müssen, wie nahe das orts- und wertdiskretisierte Feld an das kontinuierliche Modellfeld herankommen soll.

Dennoch läßt sich der modellbasierte Ansatz mit dem Datenflußparadigma – als *dem* Implementationsmodell heutiger Visualisierungswerkzeuge – gut vergleichen: der erste Schritt, der Modellerschritt, umfaßt Teile der Filtering-Operation. Der zweite Schritt, das Erzeugen einer Ansicht vom Modellfeld, faßt den Rest der Visualisierungspipeline mit der Mapping- und der Rendering-Operation zusammen. Für den Filtering-Schritt ergibt sich damit eine saubere Trennung zwischen der Datenaufbereitung (wie zum Beispiel Interpolation, mathematische Operationen) und der Datenselektion (zum Beispiel Profilschnitte oder Ausschnitte des Datensatzes) [KBro93].

Mapping und Rendering werden nicht mehr unterschieden; die Unterscheidung zwischen den beiden Schritten ist für den in Computergraphik unbedarften Benutzer auch nicht auf den ersten Blick erkennbar – die Wahl einer Rendering-Technik ist für ihn eher "Ansichtssache" als die Auswahl eines mehr oder minder komplizierten Verfahrens zur Umsetzung von Geometrien in zweidimensionale Bilder.

Weil der modellbasierte Ansatz von einem Modellfeld der Anschauung ausgeht und der Visualisierungsprozeß in zwei Schritte unterteilt wird, deren Aufgabe und Inhalt unmittelbar einleuchtend sind, eignet sich der modellbasierte Ansatz sehr gut als Gestaltungsrichtlinie einer intuitiven Benutzeroberfläche.

In [KBro93] werden neben dem modellbasierten Ansatz auch Klassifizierungsschemata vorgestellt, nach denen Felder und Ansichten eingeteilt werden können. Sie sollen im folgenden Abschnitt kurz beschrieben werden.

3.2.3 Klassifizierung von Feldern und Ansichten

Das zugrundegelegte Feld wird im modellbasierten Ansatz als eine Funktion $Y = f(X)$ unabhängiger Variablen angesehen. Dabei können X und Y Aggregate oder Strukturen von Skalaren, Vektoren oder Tensoren sein. Als Beispiel sei das folgende Aggregat einer abhängigen Variable Y betrachtet:

$$Y : (y_1, y_{2.1}, y_{2.2}, y_{2.3}, y_3) \quad .$$

Diese Variablenstruktur enthält zwei skalare Komponenten y_1 und y_3 sowie drei Komponenten $y_{2.1}$, $y_{2.2}$ und $y_{2.3}$, die zu einem Vektor $\vec{y}_2 = (y_{2.1}, y_{2.2}, y_{2.3})$ zusammengefaßt werden können.

Sowohl die abhängige als auch die unabhängige Variable können nach ihrem Typ klassifiziert werden. Nach [KBro93] gibt es dazu zwei Fälle: die Variable ist entweder ordinal (O), dann läßt sich zwischen ihren Werten eine Ordnungsrelation einführen, oder nominal (N), dann sind die Werte der Variablen Namen, zwischen denen keine Ordnungsrelation besteht. Ferner können Variablen nach ihrem Rang und ihrer Dimension unterschieden werden. Variablen vom Rang 0 sind Skalare (S), Variablen vom Rang 1 heißen Vektoren (V), und schließlich werden Variablen vom Rang 2 Tensoren (T) genannt. Zur Klassifizierung wird in [KBro93] die folgende Notation für die abhängige Variable vorgeschlagen:

$$Y : Typ^{Rang_{Dimension}} ,$$

also beispielsweise O^{V_3} für eine ordinale abhängige Variable, die aus einem dreidimensionalen Vektor besteht, oder N^S für eine abhängige Variable, die einen einfachen Namen annehmen kann.

Für die unabhängige Variable wird die Notation etwas eingeschränkt:

$$X : Typ_{Dimension} ,$$

der Rang der unabhängigen Variablen wird auf 1 festgelegt, dafür werden aber zwei Operatoren eingeführt, die auf die unabhängige Variable angewandt werden können. Der eine Operator ist die Aggregation, die mit $[]$ gekennzeichnet wird; das Feld mit einer unabhängigen Variable des Typs $[O_2]$ besitzt demnach einen zweidimensionalen Definitionsbereich, in dem jedoch zusammenhängende Bereiche zu einem Wert der unabhängigen Variable zusammengefaßt werden. Die andere Operation ist die Restriktion; sie wird mit der Notation $\{ \}_{Dimension}$ geschrieben. So beschreibt $\{O_3\}_1$ einen Definitionsbereich, der durch eine Kurve im Raum gegeben ist.

Zur Beschreibung von Feldern als Funktionen von X werden die beiden Notationen von abhängiger und unabhängiger Variablen zusammengefaßt. Als Beispiele mögen die Felder der Hydrogeologie aus Abschnitt 2.1 mit dieser Notation beschrieben werden. Das Fließfeld des Wassers, ein Vektorfeld, wird dann geschrieben als

$$\vec{v}_D = \vec{v}_D(x, y, z) : O^{V_3}(O_3) ,$$

die Permeabilität, vereinfacht angenommen als Skalarfeld, als

$$K = K(x, y, z) : O^S(O_3) .$$

Die Konzentrationen von k Schadstoffen, zusammengefaßt zu einem Feld, können durch

$$\begin{aligned} (c_1, \dots, c_k) = \vec{c} &= \vec{c}(x, y, z) \\ &= (c_1, \dots, c_k)(x, y, z) : O^{k \cdot S}(O_3) , \end{aligned}$$

beschrieben werden, oder aber, wenn die Konzentrationen sowohl ortsabhängig als auch abhängig vom Name des Schadstoffs sind, mit der Beschreibung

$$c = c(x, y, z, Name) : O^S(O_3, N_1) .$$

Wenn von einem Modellfeld eine Ansicht erzeugt wird, kann in den seltensten Fällen das gesamte Feld auf einmal dargestellt werden, weil dreidimensionale Objekte nicht überdeckungsfrei in eine zweidimensionale Bildebene projiziert werden können. Vollständig können in zwei Dimensionen nur zwei- oder eindimensionale Objekte abgebildet werden. Deshalb wird die Ansicht auf Teile des Modellfelds beschränkt, zum Beispiel auf Orte gleicher Eigenschaften, wie im Fall des kontinuierlichen Mappings aus Abschnitt 3.2.1.

Die Ansichten des Modellfelds können mit der gleichen Notation wie das Modellfeld selbst beschrieben werden. Beispielsweise könnte die Visualisierung des Skalarfeldes der Permeabilität (ein Feld mit der Klassifizierung $O^S(O_3)$, siehe oben) als Isofläche geschrieben werden als:

$$K : O^S(O_3) \rightarrow O^S(O_2) \quad ,$$

das dreidimensionale Gesamtfeld wird auf ein zweidimensionales Unterfeld, die Isofläche, reduziert. Als letztes Beispiel sei das Fließfeld des Grundwassers in einer Stromliniendarstellung betrachtet, bei der die Stromlinien nach dem Betrag des Vektorfeldes eingefärbt sind:

$$\vec{v}_D : O^{V_3}(O_3) \rightarrow O^S(O_1)$$

Die Stromlinien sind eindimensionale Teilfelder des dreidimensionalen Vektorfelds; das ursprüngliche Vektorfeld wurde zu einem Skalarfeld reduziert.

Diese Notation befindet sich, wie in [KBro93] erwähnt, noch in der Entwicklung. So werden die Eigenschaften von Glyphen nicht berücksichtigt; zudem gibt es noch keine Notation für die in der Visualisierung so wichtige Operation der Diskretisierung – der Aggregierungsoperator $[]$ ist für die Zusammenfassung von Teilintervallen des Definitionsbereiches im Sinne einer Mittelung reserviert, nicht aber für die Beschränkung des Definitionsbereiches auf einzelne Abtastpunkte.

3.2.4 Gittermodell

Hibbard, Dyer und Paul untersuchen in ihrem Ansatz [WHib94a] die Eigenschaften von Visualisierungsfunktionen

$$D : U \rightarrow V \quad ,$$

die Daten (*data*) U in Darstellungen (*displays*) V umwandeln. Als Hilfsmittel wird ein Gittermodell (*lattice model*) für die Visualisierungsfunktion aufgestellt, mit dem die Datenobjekte und die Darstellungen nach ihrem Informationsgehalt geordnet werden können.

Eine Menge U bildet ein geordnetes Gitter, wenn zum einen eine binäre Ordnungsrelation (*partial order*) existiert und zum anderen, wenn es für jedes Paar $u_1, u_2 \in U$

eine kleinste obere Grenze $u_1 \vee u_2$ sowie eine größte untere Grenze $u_1 \wedge u_2$ gibt. Eine Ordnungsrelation " $\leq$ " ist wiederum dann gegeben, wenn für alle $u_1, u_2, u_3 \in U$ gilt:

$$\begin{aligned} u_1 \leq u_1, u_1 \leq u_2, u_2 \leq u_1 &\Rightarrow u_1 = u_2 \quad \text{und} \\ u_1 \leq u_2, u_2 \leq u_3 &\Rightarrow u_1 \leq u_3 \end{aligned}$$

Mit der Einführung einer Ordnungsrelation für Daten und Darstellungen ergibt sich gleichzeitig auch eine untere Grenze für den Informationsgehalt. Der geringste Informationsgehalt, den Daten oder Darstellungen in einem geordneten Gitter annehmen können, ist der undefinierte oder unbestimmte Informationsgehalt; er wird mit $\perp$ bezeichnet.

Der Zweck einer jeden Visualisierung ist, den Informationsgehalt der Daten in entsprechende Informationen in den Darstellungen umzusetzen. Für diese Abbildungen von Daten auf Darstellungen wurden zwei Ausdrucksbedingungen (*expressiveness conditions*) von Mackinlay [JMac86] eingeführt. Demnach drückt eine Darstellung dann eine Menge von Aussagen aus, wenn die Darstellung (1.) alle Aussagen der Menge codiert und (2.) nur diese codiert. Aussagenmengen werden in [JMac86] und [WHib94a] *facts* genannt und können beispielsweise Mengen von Relationen sein. Der Begriff der Darstellung ist also weniger als Auswahl einer Aussagen-Teilmenge zu verstehen – wie zum Beispiel beim modellbasierten Ansatz die Ansicht nur ein Teilfeld des gesamten Modellfeldes darstellt –, sondern vielmehr als eine Repräsentationsform aller Daten (zum Beispiel durch Voxel).

In [WHib94a] werden Aussagenmengen etwas präziser gefaßt durch Prädikate, d.h. logische Aussagen zwischen je zwei Objekten, die entweder unbestimmt, falsch oder wahr sind. Die erste Ausdrucksbedingung lautet dann, daß für jedes monotone Prädikat², das sich für die Datenobjekte aufstellen läßt, ein entsprechendes monotones Prädikat für die jeweilige Darstellung der Datenobjekte existiert. Die zweite Bedingung legt umgekehrt fest, daß jedes monotone Prädikat unter den Darstellungen ein Pendant unter den Daten besitzt. Sind beide Ausdrucksbedingungen erfüllt, so wird die Funktion $D : U \rightarrow V$ eine Darstellungsfunktion genannt. In [WHib94b] wird nachgewiesen, daß eine Funktion $D : U \rightarrow V$ dann und nur dann eine Darstellungsfunktion ist, wenn die Funktion eine Gitter-Isomorphie von U auf $\downarrow D(X)$ darstellt. $\downarrow D(X)$ bezeichnet dabei das vollständige geordnete Gitter aller Darstellungen, deren Informationsgehalt kleiner ist als die Darstellung der kleinsten oberen Grenze des vollständigen geordneten Gitters U . Die Gitter-Isomorphie bedeutet, daß für alle $u_1, u_2 \in U$ gilt

$$\begin{aligned} D(u_1 \vee u_2) &= D(u_1) \vee D(u_2) \quad \text{und} \\ D(u_1 \wedge u_2) &= D(u_1) \wedge D(u_2) \quad . \end{aligned}$$

Bijektive Abbildungen sind genau dann isomorph zwischen zwei Objekten, wenn alle Strukturen zwischen den beiden Objekten für beide Richtungen der Abbildung

²Ein Prädikat $P : U \rightarrow \{\text{unbestimmt}, \text{falsch}, \text{wahr}\}$ ist monoton, wenn es die Ordnungsrelation beibehält: $u_1, u_2 \in U : u_1 \leq u_2 \Rightarrow P(u_1) \leq P(u_2)$. Dabei gelte $\text{unbestimmt} < \text{falsch}$ und $\text{unbestimmt} < \text{wahr}$.

beibehalten werden. So erhält eine Ordnungsisomorphie die Ordnungsrelationen bei der Abbildung; eine Gruppenisomorphie bewahrt die Gruppenstruktur. Für Darstellungsfunktionen gilt also die Isomorphie bezüglich des Informationsgehaltes der Datenobjekte und ihrer Darstellungen.

Aus dem Gittermodell werden ein Daten- und ein Darstellungsmodell abgeleitet, in dem die Datenobjekte Annäherungen an ideale mathematische Objekte sind, und die realen Darstellungen ideale Darstellungen annähern. Im Datenmodell spielt dabei der undefinierte Wert $\perp$ eine wichtige Rolle: zum einen werden damit *missing values* im Gitter repräsentiert, also fehlende Daten. Zum anderen ermöglicht der undefinierte Wert aber auch, n -Datentupel in das geordnete Gitter einzubetten. Dazu ein Beispiel: Sei ein geordnetes Datengitter U gegeben, in dem die Zeit, die Temperatur und der Druck, also drei Skalare, zusammengefaßt sind. Objekte dieses geordneten Gitters sind Tripel der Form $(Zeit, Temperatur, Druck)$. Tupel der Struktur $(Temperatur, Druck)$ lassen sich als erweitertes Tripel $(\perp, Temperatur, Druck)$ genauso einfügen, wie auch beispielsweise eine Folge von Temperaturwerten über die Zeit $(Zeit_j, Temperatur_j)$, und zwar als eine Folge von Tripeln $(Zeit_j, Temperatur_j, \perp)$. Aus der Gitterordnung ergibt sich, daß ein Tripel $(Zeit_j, Temperatur_j, \perp)$ oder ein Tripel $(\perp, Temperatur, Druck)$ immer von geringerem Informationsgehalt ist, als ein vollständiges Tripel $(Zeit, Temperatur, Druck)$.

Teile des Gittermodells, hauptsächlich aber die Inhalte des modellbasierten Ansatzes, liefern die Grundlage für das im nächsten Kapitel vorgestellte Konzept zur Visualisierung verteilter Datenfelder. Zur Implementation wird auf das Datenflußparadigma zurückgegriffen. Zunächst soll jedoch noch in einer kurzen Übersicht dargestellt werden, welche Visualisierungswerkzeuge bereits existieren und eventuell als Implementationsplattform genutzt werden können.

3.3 Visualisierungswerkzeuge

Die Visualisierungswerkzeuge lassen sich grob in zwei Klassen unterscheiden: in die schlüsselfertigen Anwendungen und in die modularen Visualisierungsumgebungen.

3.3.1 Schlüsselfertige Anwendungen

Die schlüsselfertigen Anwendungen, auch monolithische Visualisierungswerkzeuge genannt, sind auf dedizierte Visualisierungsaufgaben zugeschnitten. Als Beispiele für diese Werkzeuge seien ohne Anspruch auf Vollständigkeit genannt:

- **Alias|Wavefront Data Visualizer**, von Wavefront Technologies, Version 3.0, 1993,
- **ISVAS** – zur Visualisierung von Volumendaten [ISVAS],

- **Vis-5D** – zur Visualisierung von Klima- und Meteorologiedaten [Vis5D],
- **VMD** – zur Visualisierung von Molekülstrukturen [VMD].

Schlüsselfertige Visualisierungswerkzeuge bieten einen festen, beschränkten Satz an Visualisierungsmethoden, die sich für den jeweiligen Aufgabenbereich am besten eignen. Auch der Satz an Datenformaten und -strukturen, mit denen die anfallenden Daten eingelesen und intern repräsentiert werden, ist eingeschränkt.

Der Vorteil der schlüsselfertigen Anwendungen liegt in der sehr kurzen Einarbeitungszeit für den Endanwender – die wenigen Methoden zur Visualisierung sind schnell erlernt, die Benutzeroberfläche ist in der Regel einfach strukturiert, und die Daten des Endanwenders lassen sich schnell an das Datenformat anpassen, sofern das Werkzeug nicht "fachfremd" eingesetzt wird.

Nachteilig ist jedoch, daß sich monolithische Anwendungen nur auf Programmcode-Ebene ändern und erweitern lassen. Damit wird es für den in Computergraphik unbedarften Endanwender ausgeschlossen sein, ein Prototyping neuer Visualisierungsmethoden anzugehen oder die oben genannten "fachfremden" Daten visualisieren zu können.

3.3.2 Modulare Visualisierungsumgebungen

Modulare Visualisierungsumgebungen (*Modular Visualization Environment*, MVE) bieten zur Zeit unter den Visualisierungswerkzeugen die größtmögliche Flexibilität und Erweiterbarkeit in bezug auf Visualisierungsmethoden und Datenstrukturen.

Beispiele für modulare Visualisierungsumgebungen sind:

- **AVS** (Application Visualization System) von Advanced Visualization Systems Inc., Version 5.0, 1993,
- **Visualization Data Explorer (DX)** von IBM Inc., Version 2.1, 1993,
- **Khoros** von Khoros Research Inc., Version 2.0.2, 1995, [Khoros],
- **IRIS Explorer** von Silicon Graphics, Inc., Version 3.0, 1995.

Eine vergleichende Übersicht über die drei MVEs AVS, Data Explorer und IRIS Explorer, sowie der monolithischen Anwendung Data Visualizer findet sich in [SBel96].

Modularer Ansatz

Modulare Visualisierungsumgebungen arbeiten nach dem Datenflußparadigma. Für jeden Schritt in der Visualisierungs-Pipeline sind ein oder mehrere Module zuständig. Die Module erhalten Daten an ihren Eingängen, verarbeiten sie, und leiten sie in ihren Modulausgängen an andere Module weiter. Der Benutzer kann die Visualisierungspipeline nach seinem Belieben in einem graphischen Netzwerkditor mit *point-and-click*-Verfahren zusammenstellen: die Module werden auf einer

Arbeitsfläche platziert, und werden über Datenleitungen miteinander verbunden. Die Datenleitungen können per Mausklick eingerichtet oder aufgelöst werden.

Das erstellte Modul-Netzwerk entspricht einem gerichteten Graphen; die Module repräsentieren dabei die Knoten des Graphen, die Datenleitungen sind die gerichteten Kanten des Graphen.

Der modulare Ansatz ergibt sich aus der Austauschbarkeit der Module gleicher Klasse: beispielsweise könnte ein Mapper-Modul "Stromlinien" gegen ein Mapper-Modul "Vektorpfeile" ausgetauscht werden, weil beide Module ein Vektorfeld (Datentyp des Moduleingangs) in eine Geometrie (Datentyp des Modulausgangs) umwandeln.

Bei manchen MVEs können die Module auf verschiedenen Host-Rechnern ablaufen. Die Aufgaben in der Visualisierungspipeline können also verteilt gelöst werden. Generell werden Daten zwischen den Modulen über *shared memory* ausgetauscht; damit wird eine redundante Datenhaltung vermieden, was insbesondere im Fall der verteilten Netzwerke von großem Nutzen ist.

Sind der Aufbau bestimmter Standard-Netzwerke und die Funktionen der wichtigsten Module erlernt, so kann ein Endanwender nach einer Einarbeitungszeit die wichtigsten Standard-Visualisierungen seiner Daten erstellen. Dazu müssen die Daten jedoch in ein solches Format gebracht werden, daß sie zu den vorgegebenen Datenstrukturen der MVE passen. Mittlerweile gibt es jedoch eine Fülle von selbstgeschriebenen Modulen aus der Benutzergemeinde der jeweiligen MVE; in vielen Fällen ist das gesuchte Lesemodul oder die gesuchte Funktion bereits implementiert worden und läßt sich frei von einem FTP-Server herunterladen.

Datenstrukturen

Modulare Visualisierungsumgebungen bieten entweder eine Vielzahl von Datenstrukturen (wie AVS) oder vereinen alle Datenstrukturen in einer einzigen universellen Feldstruktur (wie zum Beispiel im IBM Data Explorer mit dem Konzept der *fiber bundles*).

Als Beispiel für die vorgegebenen Datenstrukturen einer MVE seien die Feld-Datenstrukturen in AVS, Version 5, genannt. Dort kann ein Feld durch zwei unterschiedlichen Strukturen beschrieben werden: mit einem AVS-Feld oder mit einer UCD-Datenstruktur.

AVS-Feldern liegt ein n -dimensionales Datenfeld der Größe $N_1 \times N_2 \times \dots \times N_n$ zugrunde. Je nachdem, wie regulär die Ortskoordinaten im m -dimensionalen Darstellungsraum verteilt sind, unterscheidet man zwischen uniformen, rektilinearen und irregulären AVS-Feldern (siehe Abb. 3.5, für den zweidimensionalen Fall). Bei den **irregulären** Feldern sind die insgesamt $N_1 \cdot N_2 \cdot \dots \cdot N_n$ Ortskoordinaten beliebig im Raum verteilt. Jeder Ort muß explizit durch ein m -Tupel beschrieben werden. Die Ortskoordinaten von **rektilinearen** AVS-Feldern werden durch m Koordinatenfolgen beschrieben. Jede Folge bestimmt dabei alle Koordinaten von einer der m Raumrichtungen. Bei **uniformen** Feldern als den regelmäßigen AVS-Feldern sind diese Koordinatenfolgen äquidistant.

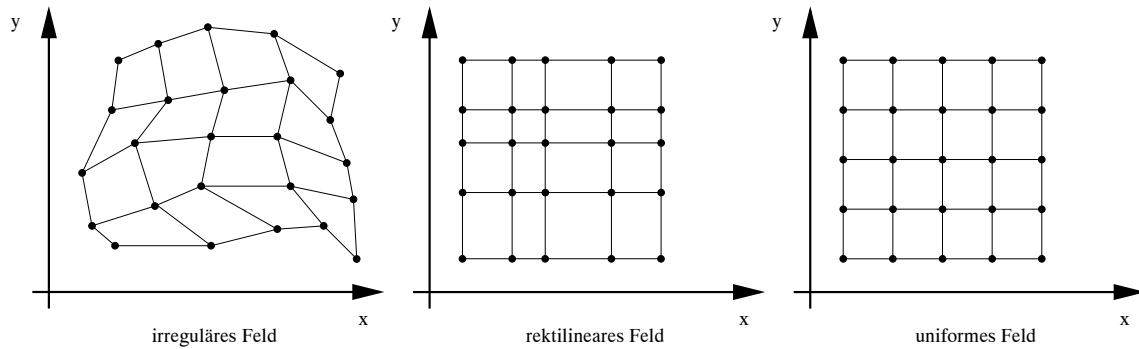


Abbildung 3.5: Felder in AVS

Mit dem Datentyp UCD (UCD steht für *unstructured cell data*) besteht eine zusätzliche Möglichkeit, dreidimensionale Felder zu modellieren. Dieser Datentyp ist speziell für Finite-Elemente-Anwendungen entwickelt worden; hier lassen sich nämlich die Knoten des FE-Gitters zu Zelltopologien zusammenfassen. Beispiele für solche Topologien sind Hexaeder, Tetraeder oder Prismen. Alle Zellen des FE-Gitters haben die gleiche Topologie und sind in einer UCD-Datenstruktur zusammengefaßt. Jedem Knoten, jeder Zelle, sowie der UCD-Struktur insgesamt können Werte der abhängigen Variablen zugeordnet werden. Die unabhängige Variable wird durch die Koordinatentripel der Knoten repräsentiert.

Zu jeder Zelle gehört eine Liste der Knoten, aus denen diese Zelle besteht; umgekehrt beschreibt eine Zellenliste für jeden Knoten, welchen Zellen er zugeordnet ist. Mit diesen Listenstrukturen erlauben UCDs eine freie Modellierung der Daten, insbesondere für Finite-Elemente-Anwendungen. Allerdings sind UCD-Daten gerade wegen der langen Knoten- und Elementlisten recht speicher- und rechenintensiv.

Erweiterungsmöglichkeiten

Die größten Vorteile modularer Visualisierungsumgebungen gegenüber den monolithischen Visualisierungswerkzeugen liegen in ihren Erweiterungsmöglichkeiten. Wiederum seien auch hier beispielhaft die Möglichkeiten der Visualisierungsumgebung AVS betrachtet.

In AVS, wie in vielen anderen MVEs auch, können Module selbst erstellt und zusammen mit den bereits vorhandenen Standard-Modulen des Visualisierungssystems in den Netzwerken verwendet werden. Damit reduziert sich der Aufwand zum Beispiel beim Prototyping einer neuen Visualisierungsmethode ganz erheblich, weil nur ein oder zwei Module neu geschrieben werden müssen; der übrigen Module stammen aus bestehenden Modul-Bibliotheken. Bei der Programmierung eines Moduls kann auf ein MVE-spezifisches API (*Application Programmer Interface*) zurückgegriffen werden. Die selbstgeschriebenen Module werden, wie die mitgelieferten Standard-Module auch, einzelnen Modulklassen zugeordnet.

Zusätzlich können in AVS eigene Datenstrukturen definiert und deren Instanzen zwischen den Modulen ausgetauscht werden. Zur Definition der benutzereigenen

Datenstrukturen ist jedoch nur eine Untermenge der Typen der jeweiligen Programmiersprache (wie C) zugelassen; Zeiger auf andere Datenobjekte sind beispielsweise nicht erlaubt. Der Grund liegt darin, daß das MVE (hier AVS) Teile der Speicherverwaltung selbst übernimmt, um Daten zwischen den Modulen via *shared memory* austauschen zu können.

Die Benutzerschnittstelle läßt sich in AVS, wie zum Teil auch in anderen MVEs, mit einem festen Satz von Gestaltungselementen frei konfigurieren. Die sogenannten Widgets, wie zum Beispiel die Schalter, Drehknöpfe oder Texteingabestellen der Module, lassen sich auf Panels oder Stacks beliebig gruppieren. Die Bedienelemente können auch gelöscht werden, um die Details der Implementierung vor dem Benutzer zu verbergen oder um ungültige Eingaben zu verhindern.

Ein weiterer wichtiger Aspekt bei der Gestaltung der Benutzerschnittstelle ist die Möglichkeit, die Visualisierung nicht nur interaktiv, sondern auch über Skript-Dateien steuern zu können. Solche Skripte werden als Reaktion auf Benutzereingaben gestartet und führen dann die in ihnen niedergelegten Anweisungen im *batch*-Betrieb nacheinander aus. Damit läßt sich beispielsweise der Aufbau von Netzwerken oder die Eingabe von Parametern automatisieren oder eine Animation erzeugen. Während des Ablaufs des Skripts sind keine weiteren Benutzereingaben notwendig.

Mit den Gestaltungsmöglichkeiten für die Benutzerschnittstelle und der Steuerung über Skripte können kompakte Endanwendungen erstellt werden. Gerade MVEs mit ihrer Funktionsfülle und ihrem weiten Gestaltungsfreiraum überfordern den Endanwender (*environment overkill*, [SHac94]) und belasten ihn mit unnötigem Ballast bei der Einarbeitung. Beschränkt man sich in der Endanwendung auf die notwendigen Funktionen, verkürzt sich die Einarbeitungszeit und die Akzeptanz für das Visualisierungswerkzeug wird gefördert.

Kapitel 4

Distributed Array Visualization: Konzeption

Aus den Problemstellungen und den Vorgaben der beiden vorigen Kapitel können nun die Anforderungen an die Visualisierung verteilt berechneter, mehrdimensionaler Datenfelder formuliert werden. Um diese Anforderungen zu erfüllen, wird ein Ansatz zur *Distributed Array Visualization* in Abschnitt 4.2 erstellt. Die zu diesem Ansatz zugehörigen Datenstrukturen und Methoden werden anschließend in den Abschnitten 4.3 und 4.4 erarbeitet.

4.1 Anforderungen an die Visualisierung verteilt berechneter Datenfelder

Bei der Visualisierung verteilter Daten treten generell zwei gegensätzliche, aber nicht unvereinbare Ziele auf:

1. Zum einen die **Visualisierung des zugrundeliegenden Phänomens** aus Physik, Biologie, Chemie oder einer anderen Naturwissenschaft. Dies bedeutet die Umsetzung der Daten aus Rechnung oder Messung in ein Modellfeld (oder in eine Modellgeometrie), von dem bereits eine konkrete, anschauliche Vorstellung vorliegen muß. Nach der Modellierung schließt sich die Visualisierung des Modells an.

Bei einer Visualisierung dieser Art sollten Details über die Struktur oder Verteilung der Daten vor dem Anwender verborgen bleiben, weil sie die Darstellung überfrachten würden. Der Eindruck von einem homogenen Feldmodell würde nur gestört werden.

2. Zu Prüf- und Kontrollzwecken, wie zum Beispiel bei einer Plausibilitätskontrolle oder bei einer Performance-Analyse, gehört andererseits jedoch gerade die **Visualisierung der logischen Struktur** der Daten aus Berechnung oder

Messung zu den Anforderungen an die Visualisierung verteilt berechneter Datenfelder.

Hierbei soll also weniger ein physikalisches Phänomen dargestellt werden. Vielmehr wird die darzustellende logische Struktur auf eine abstrakte, künstliche Modellgeometrie abgebildet. (Ein Beispiel für eine solche abstrakte Modellgeometrie ist die Kiviat-Tube, mit der Performance-Daten von Parallelrechnern abgebildet werden, siehe Abschnitt 4.3.1 und [SHac94].)

In einer Implementation sollte es dem Endanwender überlassen sein, zu welchem Grad er entweder das naturwissenschaftliche Phänomen, die logischen Strukturen oder beides zusammen visualisieren möchte. Die Entscheidung geht mit der Wahl einer Ansicht auf den Datensatz einher, findet also eher am Ende des Visualisierungsprozesses statt.

Im Fall einer Domänen-Aufteilung von Datenfeldern beinhaltet die Visualisierung der logischen Struktur die schematische Darstellung der Domänengrenzen. Die logische Verteilung der Daten läßt sich ohne weiteres im geometrischen Modell nachvollziehen, denn die Grenzen einer Datenfeld-Partition lassen sich geometrisch auf die Grenzen einer Domäne abbilden.

Speziell für verteilte Daten ergeben sich jedoch noch weitere Anforderungen an die Visualisierung:

3. Gerade bei der Berechnung von Finite-Elemente-Problemen durch parallele Berechnungsmethoden gibt es einerseits Gebiete im Gitter der finiten Elemente, die von mehreren Prozessoren gleichzeitig berechnet werden, andererseits aber auch Gebiete, deren Ergebnisse zwischen den Prozessoren im Verlauf der Berechnung ausgetauscht werden müssen. Für solche Gebiete soll eine **vergleichende Visualisierung** möglich sein, um zum Beispiel in Überlappzonen, wie sie für das Schwarz'sche Verfahren in Abschnitt 2.4.1 beschrieben wurden, Einblicke in den Verlauf der Berechnung zu erhalten.
4. Weil die Visualisierung einen Schritt in einem (Regel-)Kreis von Simulation, Datenaufbereitung, Analyse und Datensichtung, Modifikation und Feedback darstellt, ist eine **hohe Interaktivität** von großer Bedeutung für die Visualisierung. Die Visualisierung des Gesamtfeldes benötigt eventuell große Mengen an Speicherplatz oder dauert beim Bildaufbau zu lange, um mit ihr interagieren zu können. Deshalb sollen die verteilten Datenfelder vom Benutzer ausgewählt und sowohl einzeln, als auch zu Untermengen gruppiert visualisiert werden können. Auch die Parameter der Visualisierung sollen schnell variiert werden können.

Zur Forderung nach Interaktivität gehört die Forderung nach einer übersichtlichen Benutzerschnittstelle. Details der Implementation, wie zum Beispiel über Datenstrukturen, Netzwerke oder Plattform, sollen dem Endanwender verborgen bleiben. Gleichzeitig soll die Benutzerschnittstelle um eigene Visualisierungsmethoden erweiterbar sein.

5. Das Konzept zur Visualisierung verteilter Datenfelder soll prinzipiell **unabhängig von spezifischen Visualisierungssystemen** sein. Wenn zur Implementation eine modulare Visualisierungsumgebung ausgewählt wird, sollte diese Plattform gegen jede andere gleichwertige modulare Visualisierungsumgebung austauschbar sein.

Diese fünf grundsätzlichen Anforderungen an eine Visualisierung verteilt berechneter Datenfelder werden bei der Festlegung der Funktionen der **DAVis – Suite** in Kapitel 4.4 noch genauer zu betrachten sein.

4.2 Ansatz

Zur Visualisierung verteilter Datenfelder wurde zunächst nach einem Ansatz gesucht, mit dem sich insbesondere Anforderung 1 für die verteilten Daten erfüllen läßt. Dazu erscheint sinnvoll, für den Visualisierungsprozeß ein Modellfeld aus den verteilten Daten zu konstruieren, wie im modellbasierten Ansatz aus Abschnitt 3.2.2 vorgeschlagen wird. Das Modellfeld läßt sich dabei als eine Funktion einiger unabhängiger Variablen darstellen:

$$Y = f(X) \quad ,$$

mit $X = (x_1, \dots, x_{m'})$ und $Y = (y_1, \dots, y_{n'})$.

(4.1)

Die unabhängige Variable X kann ein Skalar oder ein Vektor beliebiger Länge m' sein. Die abhängige Variable Y kann dagegen aus mehreren Skalaren, Vektoren oder sogar Tensoren bestehen:

$$y_i = f_i(x_1, \dots, x_{m'}) \quad ,$$

mit y_i : Skalar, Vektor oder Tensor, $i = 1..n'$.

(4.2)

Die Abbildung f des Modellfeldes ist eindeutig; zu jeder Instanz der Variablen X existiert genau eine Instanz der Variablen Y und umgekehrt.

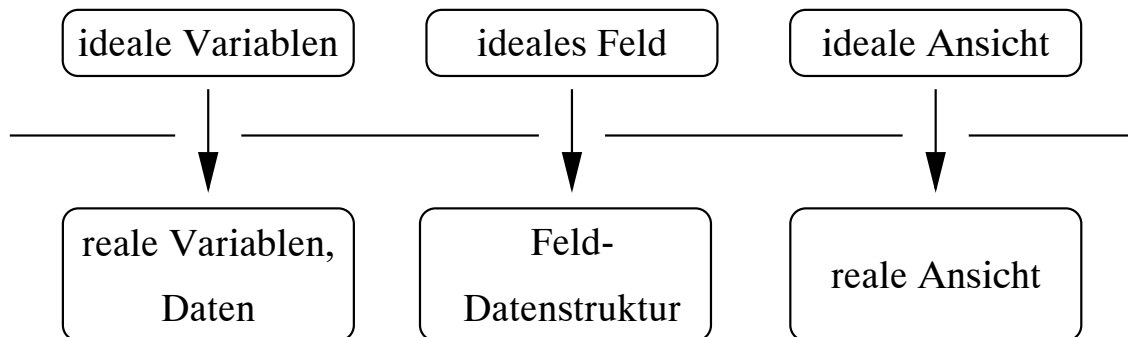


Abbildung 4.1: Reale Visualisierung als Näherung einer idealen Visualisierung

Das Modellfeld soll dem modellbasierten Ansatz nach kontinuierlich sein und dem idealen Feld der wissenschaftlichen Untersuchung entsprechen. Von dem aufgestellten empirischen Modell wird anschließend eine Ansicht erzeugt. Innerhalb eines realen, computergestützten Visualisierungssystems wird jedoch ein ideales Feld im Rechner nicht nachgebildet werden können. Stattdessen werden Eigenschaften des idealen Feldes auf die Inhalte einer realen, das heißt diskreten Feld-Datenstruktur abgebildet, und mit dieser Feld-Datenstruktur das ideale Feld angenähert. Ähnlich sind die wissenschaftlichen Daten (die realen Variablen) diskretisierte oder fehlerbehaftete Näherungen an ideale Variablen. Auch die erzeugte reale Ansicht kann nur eine Näherung an eine ideale Ansicht sein, denn reale Ansichten sind in Zeitschritten, Pixeln oder Voxeln diskretisiert. Sie sind damit in ihrer Auflösung begrenzt, was für ideale Ansichten prinzipiell nicht gilt (siehe dazu auch [WHib94a]). Somit ist der reale Visualisierungsprozeß insgesamt eine Näherung an einen idealen Visualisierungsprozesses, wie in Abb. 4.1 dargestellt wird.

Zur Visualisierung der verteilten Daten erscheint es nun – auch wegen der Forderung nach Interaktivität (siehe Anforderung 4) – sinnvoll, die verteilten Daten zuerst als reale Variablen in einer Feld-Datenstruktur zu sammeln und erst danach für die Ansicht als Modellvariablen zu modellieren. Damit gewinnt der Modellierungsschritt an Interaktivität, denn zu jeder neuen Modellierung müßten nicht erst die verteilten Daten einzeln gelesen werden, sondern die bereits zusammengestellten realen Variablen aus einer festen Feld-Datenstruktur. Damit ergibt sich der in Abb. 4.2 gezeigte Ablauf bei der realen Visualisierung.

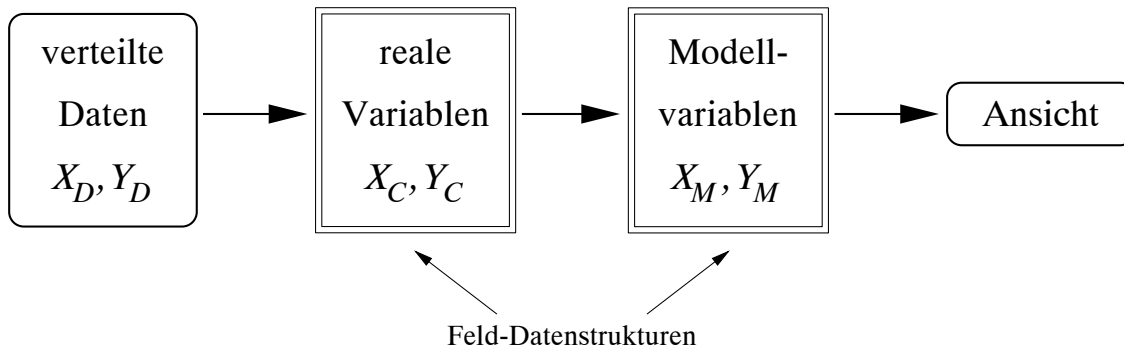


Abbildung 4.2: Die reale Visualisierung verteilter Daten

Die beiden Modellvariablen X_M und Y_M lassen sich als das Ergebnis zweier getrennter Modellierschritte für zwei reale Variablen X_C und Y_C auffassen:

$$\begin{aligned} g_Y : Y_C &\rightarrow Y_M \quad , \text{ und} \\ g_X : X_C &\rightarrow X_M \quad . \end{aligned} \tag{4.3}$$

Die getrennte Modellierung von unabhängiger und abhängiger Variable begründet sich aus möglicherweise unterschiedlichen Modellierschritten g_X und g_Y für die beiden Variablen. Stammen die Daten beispielsweise aus einer Simulation, so braucht

die unabhängige Variable nicht gemittelt oder approximiert zu werden, denn sie wurde bereits bei der Simulation als exakt angenommen. Viel eher wird man den Definitionsbereich der unabhängigen Variablen transformieren, einschränken oder erweitern wollen, wie zum Beispiel bei einer Injektion weiterer Knoten in ein FE-Gitter oder einer periodischen Erweiterung des Definitionsbereiches bei Fourier-Analysen.

Für den Wertebereich kommt dagegen keine Erweiterung im Sinne einer Injektion in Frage, denn der Wertebereich ist als das Ergebnis der Simulation oder Messung fest vorgegeben. Typische Modellerschritte der abhängigen Variablen sind vielmehr die Approximation, oder der zweite Teil einer Interpolation, die Mittelung von Werten. Ebenfalls als Modellerschritte für die abhängige Variable sind Transformationen oder Einschränkungen des Wertebereichs denkbar, so zum Beispiel durch Skalieren, Logarithmieren, Potenzieren der Werte oder durch sogenanntes *thresholding*, dem Entfernen von Werten, die größer oder kleiner als ein bestimmter Schwellenwert sind. Die Modellierung von abhängiger und unabhängiger Variable sollte also voneinander separiert geschehen. Trotzdem können die beiden Variablen nicht *unabhängig* voneinander modelliert werden, denn beispielsweise muß für die Mittelung der abhängigen Werte bei einer Interpolation bekannt sein, welche Stellen neu injiziert wurden.

Es läßt sich nun mit Hilfe dreier Referenzabbildungen r_{XC} , r_{YC} und r_C eine indirekte Zuordnung der abhängigen Variablen Y_C zur unabhängigen Variablen X_C aufstellen. Die Abbildungen r_{XC} und r_{YC} seien als eindeutig angenommen; sie könnten zum Beispiel als Speicheradressierungen zu den Datenfeldern von X_C und Y_C realisiert werden:

$$\begin{aligned} r_{XC} : R_{XC} &\rightarrow X_C \quad \text{und} \\ r_{YC} : R_{YC} &\rightarrow Y_C \quad . \end{aligned} \tag{4.4}$$

Wenn nun zu r_{XC} und r_{YC} eine eindeutige Referenzabbildung r_C

$$r_C : R_{XC} \rightarrow R_{YC} \tag{4.5}$$

so aufgestellt werden kann, daß f_C erhalten wird über

$$f_C = r_{YC} \circ r_C \circ r_{XC}^{-1} \quad , \tag{4.6}$$

dann ist die indirekte Zuordnung von Werten zu Orten erreicht. Während der Modellierung der realen Variablen X_C und Y_C zu den Modellvariablen X_M und Y_M wird die Zuordnung von abhängiger zu unabhängiger Variablen über die Abbildung r_C nicht verändert. Für r_M mit $r_M : R_{XM} \rightarrow R_{YM}$ soll also $r_M = r_C$ gelten, wenn die "Größe" des Definitions- und Wertebereiches von f während der Modellierung gleich bleibt. Die unabhängige Variable X_C kann mit der Referenz R_{XC} genau wie eine abhängige Variable behandelt werden – und zwar wie eine von R_{XC} abhängige Variable. X_C wird von Y_C entkoppelt, beide Variablen können separat voneinander modelliert werden. Nach der Modellierung wird für die Erzeugung der Ansicht mit der Referenz R_{YM} auf den Wert Y_M zurückgegriffen, der an dem durch die Referenz R_{XM} bestimmten Ort X_M dargestellt werden soll.

Formal ließen sich analog zu den Referenzabbildungen $r_{XC}/r_{XM}, r_{YC}/r_{YM}$ und r_C/r_M auch für die Datenmenge $X_D \cup Y_D$ drei Referenzabbildungen ansetzen mit:

$$\begin{aligned} r_{XD} : R_{XD} &\rightarrow X_D \quad \text{und} \\ r_{YD} : R_{YD} &\rightarrow Y_D \quad , \text{ sowie} \\ r_D : R_{XD} &\xrightarrow{k} R_{YD} \quad . \end{aligned} \tag{4.7}$$

Im Gegensatz zu r_M und r_C wäre jedoch r_D , wie das Zeichen $\xrightarrow{k}$ in Gleichung (4.7) andeuten soll, eine mehrdeutige Abbildung. Aber auch die Abbildung f_D ist mehrdeutig:

$$f_D = r_{YD} \circ r_D \circ r_{XD}^{-1} \Rightarrow f_D : X_D \xrightarrow{k} Y_D \quad , \tag{4.8}$$

denn f_D stellt das Feld dar, das man erhielte, wenn die Datenmenge ungegliedert in eine Datenmenge für die abhängige Variable und in eine Datenmenge für die unabhängige Variable aufgeteilt würde. Diese beiden Datenmengen X_D und Y_D müssen nicht die gleiche Anzahl von Instanzen enthalten. Beispielsweise können mehr Werte Y_D als Orte X_D vorhanden sein, wie nach einer Berechnung nach dem Schwarz'schen Verfahren.

Notwendig für die Erzeugung einer Ansicht ist jedoch, daß für jeden Wert, der nach den Ausdrucksbedingungen darzustellen ist, auch ein Ort vorhanden sein muß. Deshalb wurde die Abbildung f_M des Modellfeldes eingangs auch als eindeutig angesetzt. Für die Eindeutigkeit von f_M müssen nun drei Voraussetzungen erfüllt sein, wie der Gleichung (4.9) entnommen werden kann:

$$f_M = r_{YM} \circ r_M \circ r_{XM}^{-1} \tag{4.9}$$

Die erste Voraussetzung ist die Eindeutigkeit von r_{XM}^{-1} , also die Voreindeutigkeit von r_{XM} . Die zweite Voraussetzung ist die Eindeutigkeit der Abbildung r_M . Die dritte Voraussetzung ist die Eindeutigkeit von r_{YM} ; sie ist erfüllt, denn r_{YM} wurde bereits bei der Einführung als eindeutig angesetzt. Vor der Erzeugung einer Ansicht des Modellfeldes muß also noch erreicht werden, daß r_{XM}^{-1} und r_M eindeutig sind. In Abschnitt 4.3.3 wird gezeigt, wie durch die Einbettung der verteilten Daten in die Feld-Datenstruktur eine eindeutige Abbildung r_C erhalten werden kann, und in Abschnitt 4.3.5, wie schließlich die Abbildung f_M selbst eindeutig modelliert werden kann. Wenn nach der Einbettung der verteilten Daten eine eindeutige Abbildung r_C gefunden worden ist, so kann auch ohne Modellierschritt bereits eine Ansicht aus der Feld-Datenstruktur $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$ erzeugt werden, denn dann gibt es für jeden darzustellenden Wert genau einen Ort.

Insgesamt stellt sich der reale Visualisierungsprozeß für verteilte Datenfelder wie in Abb. 4.3 schematisch dar: aus einer in mehrere Datenfelder aufgeteilten Datenmenge $X_D \cup Y_D$ werden die realen Variablen X_C und Y_C gewonnen und in einer Feld-Datenstruktur abgelegt. Das Ziel der Einbettung der Datenmenge $X_D \cup Y_D$ in eine Feld-Datenstruktur ist die Aufhebung der Mehrdeutigkeit der formal aufgestellten Abbildung r_D . Die beiden realen Variablen werden anschließend getrennt voneinander auf die Modellvariablen X_M und Y_M abgebildet. Das Ziel der Modellierung der realen Variablen X_C und Y_C ist die vollständige Aufhebung der Mehrdeutigkeit von f_D .

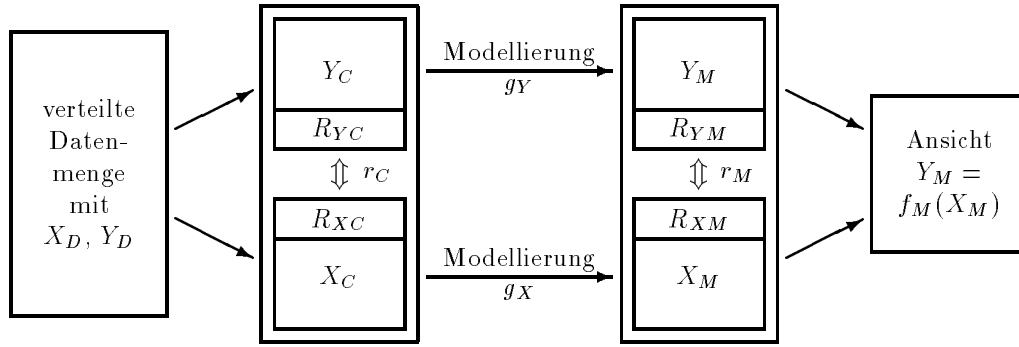


Abbildung 4.3: Getrennte Modellierung von unabhängiger und abhängiger Variable

4.3 Datenmodell und Datenstrukturen

Zur Umsetzung des vorgestellten Ansatzes zur Visualisierung verteilter Datenfelder müssen noch einige Datenmodelle und -strukturen entwickelt bzw. detaillierter ausgearbeitet werden. Die zentrale Datenstruktur ist dabei zweifelsohne die Feld-Datenstruktur; nach ihrer Beschreibung und der Festlegung der Prämissen, die für die verteilten Datenfelder gelten, kann die Einbettung der verteilten Daten in die Feld-Datenstruktur untersucht werden. Danach werden in Abschnitt 4.3.4 zwei weitere Datenstrukturen entwickelt, mit denen sich die Information über die ursprüngliche Verteilung der Daten im Rechner darstellen läßt. Schließlich folgt eine kurze Untersuchung, welche Modellerschritte die Abbildung $f_M : X_M \rightarrow Y_M$ zu einer eindeutigen Abbildung werden lassen.

4.3.1 Die Feld-Datenstruktur

Wie in Abschnitt 4.2 angesetzt wurde, soll eine Feld-Datenstruktur das ideale Modellfeld annähern und die Grundlage für das Visualisierungssystem zur Erzeugung einer Ansicht sein. Es empfiehlt sich nun, zwei Komponenten innerhalb der Feld-Datenstruktur einzurichten: eine Komponente $\underline{X}$, um alle Instanzen von X_C bzw. X_M aufzunehmen, sowie eine Komponente $\underline{Y}$ für alle Instanzen von Y_C bzw. Y_M . Die Strukturkomponenten $\underline{X}$ und $\underline{Y}$ bestehen jeweils aus m - oder n -dimensionalen Datenfeldern, deren Elemente alle vom gleichen Typ sind. Auf die Feldelemente eines Datenfeldes $\underline{X}$ oder $\underline{Y}$ wird mit den Referenzen R_X oder R_Y zugegriffen, die m - oder n -Tupel natürlicher Zahlen sind:

$$\begin{aligned} X &= \underline{X}[R_X] = \underline{X}[r_{X,1}, \dots, r_{X,m}] \\ Y &= \underline{Y}[R_Y] = \underline{Y}[r_{Y,1}, \dots, r_{Y,n}] \quad . \end{aligned} \quad (4.10)$$

Mit $\mathcal{R}(\underline{X})$ sei im folgenden die Menge aller Referenzen eines Feldes $\underline{X}$ bezeichnet. Die Referenzen R_X und R_Y in $\mathcal{R}(\underline{X})$ bzw. $\mathcal{R}(\underline{Y})$ haben verschiedene Dimensionen, weil die Variable X im Unterschied zu Y auch implizit beschrieben sein kann.

Wenn die unabhängige Variable X implizit beschrieben ist, dann befinden sich im Datenfeld $\underline{X}$ die Koordinaten x_i , und nicht die vollständigen Instanzen von X . Die abhängige Variable Y als das Ergebnis der Simulation kann nur explizit, also einzeln für jeden Knotenpunkt eines FE-Gitters, vorgegeben sein.

Eine Feld-Datenstruktur $\mathcal{F}(\underline{X}, \underline{Y})$, die aus den Datenfeldern $\underline{X}$ und $\underline{Y}$ besteht, beschreibt eine Abbildung $f : X \rightarrow Y$ explizit, wenn

1. in $\underline{X}$ Instanzen von X und in $\underline{Y}$ die zu X zugehörigen Instanzen von Y vorliegen;
2. die Dimensionen der Felder $\underline{X}$ und $\underline{Y}$ gleich sind, also $m = n$ gilt;
3. die Größen $\mathcal{S}(\underline{X})$ und $\mathcal{S}(\underline{Y})$ der beiden Felder $\underline{X}$ und $\underline{Y}$ gleich sind:

$$\mathcal{S}(\underline{X}) = \mathcal{S}(\underline{Y}) = N_1 \times N_2 \times \dots \times N_n \quad ; \quad (4.11)$$

4. für die Referenzabbildung $r : R_X \rightarrow R_Y$ gilt: $r = \text{id}$, also $r_{Y,i} = r_{X,i}$ für alle $i = 1..n$.

Die sich aus der Darstellung ergebende Geometrie für eine explizite Feld-Datenstruktur mit $m = n = 1$ wird im folgenden ein **eindimensionales irreguläres Gitter**¹ genannt. Für $m = n = 2$ heißt die Geometrie einer explizit beschriebenen Feld-Datenstruktur, in der jeweils vier Referenzen $R_{X,k}$, $k = 0..3$ zu einem Element $E(e_1, e_2)$ ($0 \leq e_i \leq N_i - 2$, $i = 1, 2$) zusammengefaßt sind, ein **irreguläres Quadrilateral-Gitter**. Für die vier Referenzen $R_{X,k}$ eines Elements $E(e_1, e_2)$ gilt dabei

$$\begin{aligned} R_{X,k} &= (r_{X1,k}, r_{X2,k}) \\ &= (e_1 + k \bmod 2, \\ &\quad e_2 + \lfloor k/2 \rfloor) \quad . \end{aligned} \quad (4.12)$$

Für $m = n = 3$ wird analog zum Quadrilateral-Gitter ein **irreguläres Hexaeder-Gitter** definiert. In diesem Gitter werden jeweils acht Referenzen $R_{X,k}$, $k = 0..7$ zu einem Element $E(e_1, e_2, e_3)$ ($0 \leq e_i \leq N_i - 2$, $i = 1..3$) zusammengefaßt, wobei gilt:

$$\begin{aligned} R_{X,k} &= (r_{X1,k}, r_{X2,k}, r_{X3,k}) \\ &= (e_1 + k \bmod 2, \\ &\quad e_2 + \lfloor k/2 \rfloor \bmod 2, \\ &\quad e_3 + \lfloor k/4 \rfloor) \quad . \end{aligned} \quad (4.13)$$

In der dreidimensionalen geometrischen Darstellung haben die Elemente dieses Gitters sechs Außenflächen, daher der Name Hexaeder-Gitter.

¹Die Begriffe "irreguläres Gitter" und "reguläres Gitter" sind nicht zu verwechseln mit dem Begriff des "geordneten Gitters" aus Abschnitt 3.2.4. Mit einem geordneten Gitter ist eine Menge mit einer Ordnungsrelation aus dem Gittermodell gemeint, und keine Geometrie.

Im Gegensatz zur **expliziten** Beschreibung eines Datenfeldes mit **irregulären** Gitterstrukturen steht die **implizite** Beschreibung einer Abbildung mit **regulären** Gitterstrukturen. Eine implizite Beschreibung der Abbildung $f : X \rightarrow Y$ durch die Feld-Datenstruktur $\mathcal{F}(\underline{X}, \underline{Y})$ liegt vor, wenn

1. in $\underline{X}$ einzelne Koordinaten von X und in $\underline{Y}$ die zu den Instanzen von X zugehörigen Instanzen der Variablen Y abgelegt sind;
2. $m = 2$ und $n = m'$ gilt (es ist $X = (x_1, \dots, x_{m'})$);
3. für die Größen der Felder $\underline{X}$ und $\underline{Y}$ gilt:

$$\begin{aligned}\mathcal{S}(\underline{X}) &= \max\{N_1, N_2, \dots, N_n\} \times n \quad , \\ \mathcal{S}(\underline{Y}) &= N_1 \times N_2 \times \dots \times N_n\end{aligned}\tag{4.14}$$

4. jeweils zu einer Referenz R_Y n Referenzen $R_X(i)$ ($i = 1..n$) zugeordnet werden:

$$r^{-1} : R_Y \xrightarrow{n} R_X(i)\tag{4.15}$$

und dabei gilt:

$$\begin{aligned}R_X(i) &= (r_{X,1}(i), r_{X,2}(i)) \\ &= (r_{Y,i}, i) \quad .\end{aligned}\tag{4.16}$$

Aus diesen n Referenzen $R_X(i)$ wird wie folgt eine Instanz von X gebildet:

$$\begin{aligned}X &= (x_1, \dots, x_n) \\ &= (\underline{X}[r_{X,1}(1), r_{X,2}(1)], \dots, \underline{X}[r_{X,1}(n), r_{X,2}(n)]) \\ &= (\underline{X}[r_{Y,1}, 1], \dots, \underline{X}[r_{Y,n}, n])\end{aligned}\tag{4.17}$$

Die sich aus einer impliziten Beschreibung der Abbildung f ergebende Geometrie wird ein **reguläres Gitter** genannt. Das Gitter ist **streng regulär**, wenn zusätzlich zur impliziten Beschreibung von f für alle $0 \leq r \leq N_i - 1$ gilt:

$$\underline{X}[r, i] = \underline{X}[0, i] + r \cdot \Delta_i\tag{4.18}$$

mit einem festen $\Delta_i \in \mathcal{R}$. Gilt für ein reguläres Gitter $\underline{X}[r, i] \leq \underline{X}[s, i]$ für alle $1 \leq i \leq n$, wenn $r < s$ ist, dann heißt das Gitter monoton.

Eine implizite Beschreibung läßt sich jederzeit in eine explizite Beschreibung umwandeln, indem die Instanzen von X nach Gleichung (4.17) aus den Koordinaten im Datenfeld $\underline{X}$ erzeugt, und anschließend nach folgender Vorschrift in ein Feld $\underline{X}'$ der Größe $\mathcal{S}(\underline{X}') = N_1 \times N_2 \times \dots \times N_n$ abgelegt werden:

$$\begin{aligned}\underline{X}'[R_{X'}] &= \underline{X}'[r_{X',1}, \dots, r_{X',n}] \\ &= (x'_1(r_{X',1}, \dots, r_{X',n}), \dots, x'_{m'}(r_{X',1}, \dots, r_{X',n})) \\ &= (\underline{X}[r_{X',1}, 1], \dots, \underline{X}[r_{X',n}, n])\end{aligned}\tag{4.19}$$

für alle $R_{X'} \in \mathcal{R}(\underline{X'})$. Die einzelnen Komponenten x'_i von $X' = X$ erhält man also aus

$$x'_i(r_{X',1}, \dots, r_{X',n}) = \underline{X}[r_{X',i}, i] \quad , \quad (4.20)$$

jede Koordinate x'_i hängt nur von $r_{X',i}$ und i selbst ab. Weil sich implizite Beschreibungen mit dieser Vorschrift in explizite Beschreibungsformen umwandeln lassen, und somit implizit beschriebene Abbildungen Spezialfälle expliziter Abbildungen sind, wird im folgenden von Feld-Datenstrukturen mit einer expliziten Beschreibung von X und Y ausgegangen.

Als Beispiel für die Wahl der Parameter n, n', m und m' einer Feld-Datenstruktur mag das Fließfeld des Grundwassers betrachtet werden. In Abschnitt 3.2.3 wurde das modellierte Fließfeld als ein Feld der Klassifikation $O^{V_3}(O_3)$ vorgestellt. Geht man davon aus, daß während der Modellierung die Dimensionen nicht verändert werden, gilt für ein solches Modellfeld $n' = 3$ (die abhängige Variable ist ein Vektor der Länge drei) und $m' = 3$ (die unabhängige Variable ist ein dreidimensionaler Ortsvektor). Über die Größen n und m sagt die Notation nach Brodlie *et al.*

$$Typ^{V_{n'}}(Typ_{m'}) \quad (4.21)$$

jedoch nichts aus, denn die Dimensionen n und m sind die Dimensionen des benutzten Rechenmodells, mit ihnen wird ein abstrakter Referenzraum aufgespannt. Referenzraum und Darstellungsraum müssen nicht von gleicher Dimension sein. Beispielsweise ist jedem Knoten des Finite-Elemente-Gitters im Rechenmodell von TRACE eine eindeutige Knotennummer zugewiesen. Diese Knotennummer wird von TRACE als Referenz auf den Wert des Fließfeldes an dieser Stelle benutzt. Die Referenz R_Y besteht aus einer einzigen Knotennummer und der Referenzraum ist eindimensional ($n = 1$). Die Größe m richtet sich nach der Beschreibungsform des Finite-Elemente-Gitters. Bei einer impliziten Beschreibung, das heißt, bei regulären Gittern, ist $m = 2$, wie oben erwähnt; bei einer expliziten Beschreibung für irreguläre Gitter gilt $m = n$.

Zwischen der Dimension der unabhängigen Variablen und der Referenz auf eine abhängige Variable läßt sich zusätzlich der folgende Zusammenhang aufstellen:

$$m' = n + \nu \quad , \quad (4.22)$$

mit $\nu = const.$, wenn die Ausdrucksbedingungen von Mackinlay eingehalten werden sollen. Die Verknüpfung dieser beiden Dimensionen läßt sich am Beispiel der sogenannten *Kiviat-Tubes* erläutern: Zur Darstellung der Prozessorauslastung aller Prozessoren eines Parallelrechners werden in ParaGraph [SHac94] sogenannte Kiviat-Diagramme verwendet. Die Auslastung eines Prozessors wird dabei durch die Länge einer Speiche repräsentiert, die vom Mittelpunkt eines Kreises mit dem Radius 100% ausgeht. Die Speichen aller Prozessoren werden an ihren Endpunkten miteinander verbunden. Wenn also alle Prozessoren zu 100% ausgelastet sind und damit die Gesamtauslastung des Parallelrechners optimal ist, überdecken die Verbindungslinien zwischen den Speichen den "100% -Kreis". In den anderen Fällen ergibt sich ein mehrzackiger Stern, wie er in Abb. 4.4 dargestellt ist.

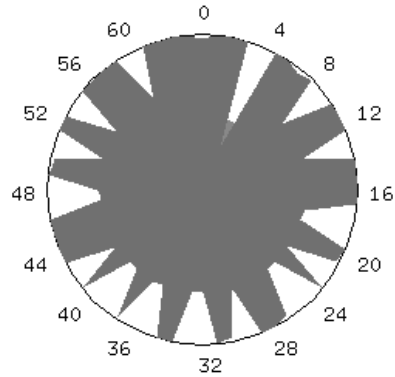


Abbildung 4.4: Kiviat-Diagramm

Bei einer solchen Visualisierung ist $m' = 2$, denn das Kiviat-Diagramm ist eine ebene, also zweidimensionale Darstellung. Die Referenz auf die abhängige Variable ist eindimensional ($n = 1$), denn als Referenz auf einen Auslastungswert dient die Nummer eines Prozessors.

Es interessiert nun, wie die Prozessorauslastung mit der Zeit variiert. Man sammelt also eine Menge Y von Auslastungswerten, deren Parameter und damit die Referenz aus dem Paar (*Prozessor*, *Zeitpunkt*) besteht. Damit wird $n = 2$, die Referenz R_Y wird um eine Dimension gegenüber der Untersuchung der Prozessorauslastung zu einem Zeitpunkt erhöht. In der Darstellung dieser Datenmenge könnte nun eine Folge von Momentaufnahmen solcher Kiviat-Diagramme hintereinander abgespielt werden. Dabei würden jedoch keine Vergleichsmöglichkeiten zwischen den Auslastungen zu verschiedenen Zeitpunkten bestehen. Eine *Kiviat-Tube*, wie sie Hackstadt in [SHac94] vorgeschlagen hat, löst dieses Problem. Dazu wird die Zeit auf die dritte Raumachse projiziert und die einzelnen Momentaufnahmen der Sequenz in einer dreidimensionalen Darstellung zusammengefaßt, wie Abb. 4.5 zeigt.

Für eine solche Darstellung, wie aber auch für die Sequenz zweidimensionaler Kiviat-Diagramme gilt $m' = 3$, denn Kiviat-Tubes sind dreidimensionale Geometrien, und eine Sequenz von Diagrammen erstreckt sich ebenfalls über drei Dimensionen: über zwei Raumdimensionen und über die Zeit als dritter Dimension. Wenn also die Dimension der Referenz R_Y um eins erhöht wird und alle Daten dargestellt werden sollen (siehe Ausdrucksbedingungen von Mackinlay im Abschnitt 3.2.4), folgt, daß der Darstellungsraum ebenfalls um eine Dimension erhöht werden muß.

Die Beziehung (4.22) gilt insbesondere dann, wenn verteilte Daten in eine Feld-Datenstruktur eingebettet werden. Bevor dieser Vorgang im Abschnitt 4.3.3 untersucht wird, folgt eine Zusammenfassung der Annahmen über die verteilten Daten, von denen im folgenden ausgegangen wird.

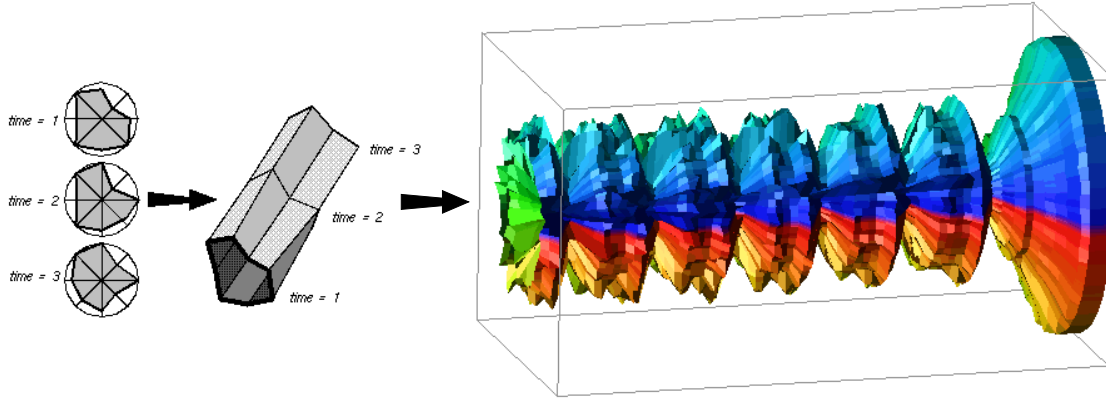


Abbildung 4.5: Kiviat-Tube

4.3.2 Annahmen für die verteilte Datenmenge

Es wird von einer Datenmenge $X_D \cup Y_D$ ausgegangen, deren Elemente die Instanzen X_D und Y_D der Daten sind. Zunächst sei die **Datenmenge** Y_D betrachtet. Y_D ist das Ergebnis der Simulation; als solches ist es, wie im Abschnitt 4.3.1 bereits erwähnt wurde, explizit für jeden Ort des Definitionsbereiches beschrieben.

Die Elemente der Datenmenge Y_D sind auf P Datenfelder $\underline{Y}_{D,p}$ verteilt:

$$Y_{D,p} = \underline{Y}_{D,p}[R_{YD,p}] \quad \text{mit } 1 \leq p \leq P; \quad (4.23)$$

diese Datenfelder können in Dateien, lokalen Speichern von Rechnerknoten oder sonstigen Speichermedien vorliegen. Für die Referenzen, die abhängige Variable selbst, und die Größe der Datenfelder gilt

$$\begin{aligned} R_{YD,p} &= (r_{YD,p,1}, \dots, r_{YD,p,n_D}) \quad , \\ Y_{D,p} &= (y_{D,p,1}, \dots, y_{D,p,n'_D}) \quad \text{und} \\ \mathcal{S}(\underline{Y}_{D,p}) &= N_{p,1} \times N_{p,2} \times \dots \times N_{p,n_D} \end{aligned} \quad (4.24)$$

für alle Datenfelder $\underline{Y}_{D,p}$, $1 \leq p \leq P$. Die Referenz R_{YD} und die abhängige Variable Y_D haben also für alle Datenfelder jeweils die gleiche Dimension n_D bzw. n'_D , denn alle Daten stammen aus einer Simulation und sollten vom gleichen Elementtyp sein, damit sie sich überhaupt sinnvoll miteinander vergleichen lassen (siehe Anforderung 3).

Die **Datenmenge** X_D kann auf zwei Weisen vorliegen. Zum einen kann die unabhängige Variable gemeinsam mit der abhängigen Variablen auf die P lokalen Speicher in eigenen Datenfeldern $\underline{X}_{D/pr,p}$ verteilt sein (*private memory*):

$$X_{D,p} = \underline{X}_{D/pr,p}[R_{XD,p}] \quad . \quad (4.25)$$

Für die unabhängige Variable X_D gilt dann, analog zu Gleichung (4.24)

$$\begin{aligned} R_{XD,p} &= (r_{XD,p,1}, \dots, r_{XD,p,n_D}) \quad , \\ X_{D,p} &= (x_{D,p,1}, \dots, x_{D,p,n_D}) \quad \text{und} \\ \mathcal{S}(\underline{X}_{D/pr,p}) &= \mathcal{S}(\underline{Y}_{D,p}) \\ &= N_{p,1} \times N_{p,2} \times \dots \times N_{p,n_D} \end{aligned} \quad (4.26)$$

für alle Datenfelder $\underline{X}_{D/pr,p}$ ($1 \leq p \leq P$).

Zum anderen können alle Elemente von X_D vor der Berechnung in einem gemeinsamen Speicher abgelegt worden sein. Auf diesen gemeinsamen Speicher (*shared memory*) greifen die Prozessoren während der Berechnung zurück. Dann ist

$$X_D = \underline{X}_{D/sh}[R_{XD}] \quad , \quad (4.27)$$

$$\begin{aligned} R_{XD} &= (r_{XD,1}, \dots, r_{XD,n_D}) \quad , \\ X_D &= (x_{D,1}, \dots, x_{D,n_D}) \quad \text{und} \\ \mathcal{S}(\underline{X}_{D/sh}) &= M_1 \times M_2 \times \dots \times M_{n_D} \quad . \end{aligned} \quad (4.28)$$

Im allgemeinen wird gelten

$$\sum_{p=1}^P N_{p,1} \cdot N_{p,2} \cdot \dots \cdot N_{p,n_D} \geq M_1 \cdot M_2 \cdot \dots \cdot M_{n_D} \quad , \quad (4.29)$$

denn wegen der Mehrdeutigkeit der Abbildung $f_D : X_D \xrightarrow{k} Y_D$ (siehe Abschnitt 4.2) werden mehr Werte Y_D in den Datenfeldern $\underline{Y}_{D,p}$ als Orte X_D im Datenfeld $\underline{X}_{D/sh}$ vorliegen.

Mit diesen Voraussetzungen kann nun an den ersten Schritt zur Visualisierung der verteilten Datenfelder $\underline{Y}_{D,p}$ und $\underline{X}_{D/pr,p}$ bzw. $\underline{X}_{D/sh}$ herangegangen werden, nämlich an die Einbettung der verteilten Daten in eine Feld-Datenstruktur.

4.3.3 Einbetten der verteilten Datenfelder in eine Feld-Datenstruktur

Die verteilten Datenfelder können je nach lokalem Speicher p unterschiedlich groß sein und verschiedene Ausmaße haben, wie dies schon in den Gleichungen (4.24), (4.26) und (4.28) angesetzt wurde.

In Abschnitt 3.2.4 wurde an einem Beispiel gezeigt, wie Tupel in ein geordnetes Gitter eingebettet werden konnten, das aus Tripel-Objekten bestand. Ganz ähnlich lassen sich auch kleine Datenfelder in ein geordnetes Gitter einbetten, dessen Objekte Datenfelder einer bestimmten maximalen Größe sind. Alle diese einzelnen Datenfelder bilden zusammengekommen ein großes Datenfeld; das geordnete Gitter stellt sich damit selbst als Datenfeld dar. Dies sei zunächst anhand der Einbettung der Datenfelder $\underline{Y}_{D,p}$ in das Datenfeld $\underline{Y}_C$ gezeigt.

Dazu werden zunächst P Datenfelder $\underline{Y}_{C,p}$ der Größe

$$\mathcal{S}(\underline{Y}_{C,p}) = N_{max,1} \times N_{max,2} \times \dots \times N_{max,n_D} \quad (4.30)$$

angelegt; dabei bestimmen sich die $N_{max,i}$ ($i = 1..n_D$) aus

$$N_{max,i} = \max\{N_{1,i}, N_{2,i}, \dots, N_{p,i}\} \quad . \quad (4.31)$$

Danach werden alle Elemente der P Datenfelder $\underline{Y}_{C,p}$ zuerst mit dem undefinierten Wert $\perp$ vorbelegt:

$$\underline{Y}_{C,p}[R_{YC,p}] := \perp \quad \text{für alle } R_{YC,p} \in \mathcal{R}(\underline{Y}_{C,p}). \quad (4.32)$$

Anschließend werden die Werte aus den Datenfeldern $\underline{Y}_{D,p}$ in die Felder $\underline{Y}_{C,p}$ kopiert, und zwar mit

$$\underline{Y}_{C,p}[R_{YD,p}] := \underline{Y}_{D,p}[R_{YD,p}] \quad \text{für alle } R_{YD,p} \in \mathcal{R}(\underline{Y}_{D,p}). \quad (4.33)$$

In Abb. 4.6 wird das Ergebnis dieser drei ersten Schritte beispielhaft für vier Datenfelder mit $n = 2$ gezeigt. Jedes der Datenfelder enthält eine mehr oder weniger große Menge $\mathcal{L}(\underline{Y}_{C,p})$ an Referenzen, die auf unbestimmte Werte verweisen. In der Skizze werden diese unbestimmten Referenzen schematisch als leere Kreise (o) dargestellt.

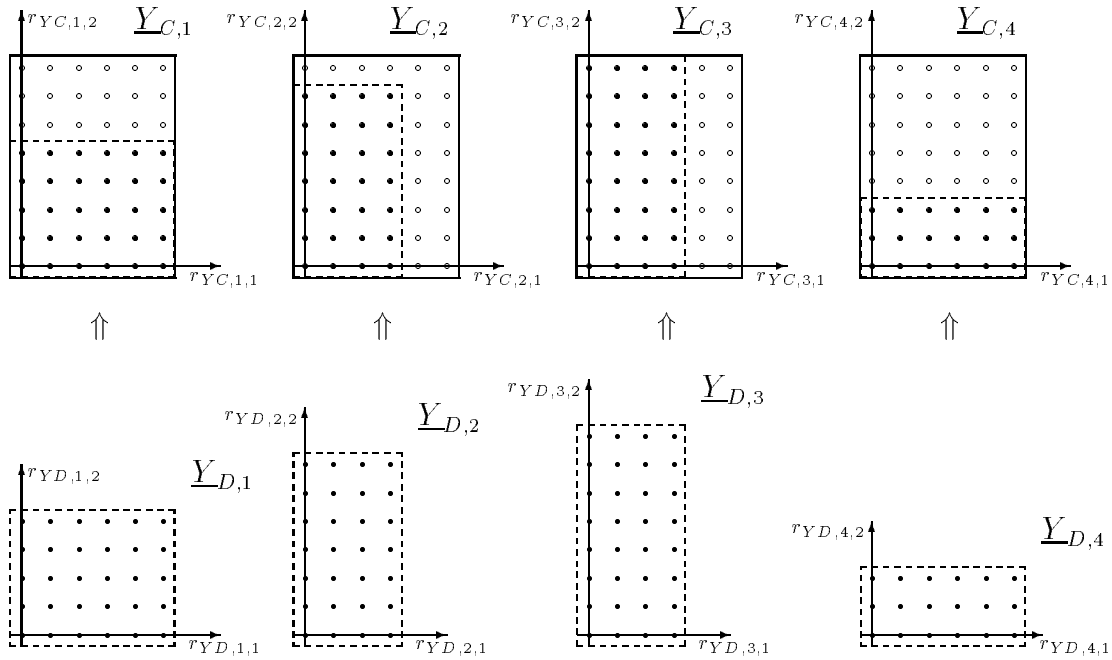


Abbildung 4.6: Einbetten von Datenfeldern: Erzeugen der $P = 4$ Datenfelder $\underline{Y}_{C,p}$

Es gibt nun zwei Möglichkeiten, die Datenfelder $\underline{Y}_{C,p}$ zu einem geordneten Gitter zusammenzufügen. Die beiden Möglichkeiten ergeben sich daraus, daß p als neue Variable hinzugekommen ist, und sich deshalb eine oder mehrere der Größen n, n', m

oder m' beim Übergang von den Daten X_D und Y_D zu den realen Variablen X_C und Y_C erhöhen. Dazu ein Beispiel: Sei wieder das Fließfeld aus dem vorigen Abschnitt 4.3.1 betrachtet. Dieses Feld sei in P Teilgebiete aufgeteilt; dies läßt sich formal mit der Notation $P \cdot O^{V_3}(O_3)$ beschreiben. Zum einen kann nun die neue Variable p als neue unabhängige Variable aufgefaßt werden. Damit wäre das Feld mit der Notation $O^{V_3}(O_4)$ charakterisiert, die Dimension des Darstellungsraumes hätte sich also um eins erhöht ($m'_C = m'_D + 1$) und nach Gleichung (4.22) auch die Dimension des Referenzraumes für die abhängige Variable ($n_C = n_D + 1$). Die Stärke, Richtung und Orientierung des Vektorfeldes würde von einem Ort und einem Teilgebiet abhängen. Für den Fall $n_C = n_D + 1$ ergibt sich dann als Größe des Feldes $\underline{Y}_C$ für die reale Variable:

Fall A ($n_C = n_D + 1$):

$$\mathcal{S}(\underline{Y}_C) = N_{max,1} \times N_{max,2} \times \dots \times N_{max,n_D} \times P \quad . \quad (4.34)$$

Die Datenfelder $\underline{Y}_{C,p}$ werden mit der Anweisung

$$\begin{aligned} \underline{Y}_C[R_{YC,p}, p] &:= \underline{Y}_{C,p}[R_{YC,p}] \quad , \text{ das heißt} \\ \underline{Y}_C[r_{YC,p,1}, r_{YC,p,2}, \dots, r_{YC,p,n_D}, p] &:= \underline{Y}_{C,p}[r_{YC,p,1}, r_{YC,p,2}, \dots, r_{YC,p,n_D}] \end{aligned} \quad (4.35)$$

für alle $R_{YC,p} \in \mathcal{R}(\underline{Y}_{C,p})$ in das Datenfeld $\underline{Y}_C$ eingebettet. Abb. 4.7 zeigt das Ergebnis dieses Vorgangs für die vier Datenfelder aus Abb. 4.6. Ein Nachteil des Zusammenfügens der Datenfelder nach dieser Methode ist jedoch, daß m'_C nicht größer als 4 werden darf, wenn das Feld später noch geschlossen dargestellt werden soll: im Darstellungsraum stehen in allen Visualisierungssystemen nur bis zu vier Dimensionen zur Verfügung – drei Raumdimensionen und eine Zeitdimension.

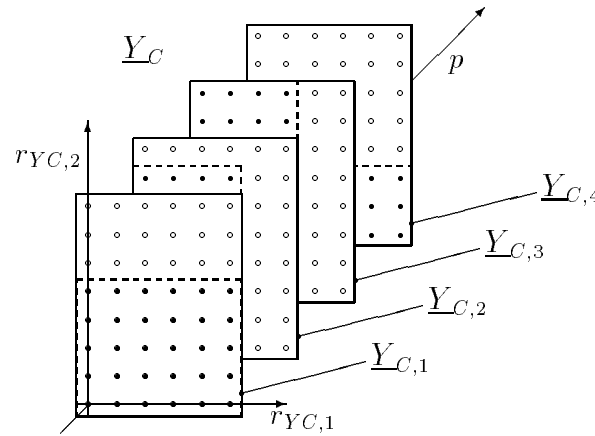


Abbildung 4.7: Einbetten von Datenfeldern: p als neue unabhängige Variable

Die andere Möglichkeit, die P Datenfelder zusammenzufügen, besteht darin, die neue Variable p zu den abhängigen Variablen zu zählen. Die Notation für das Fließfeld

wäre dann $O^{V_3+S}(O_3)$, es würde also vom Ort abhängen, in welchem Teilgebiet man sich befindet. In diesem Fall würde nur die Dimension n'_D um eins erhöht werden, und an der Dimension n des Referenzraumes für die abhängige Variable würde sich nichts ändern:

Fall B ($n'_C = n'_D + 1$):

Vor dem Zusammenfügen der Datenfelder $\underline{Y}_{C,p}$ wird P in n_D Faktoren zerlegt:

$$P = P_1 \cdot P_2 \cdot \dots \cdot P_{n_D} \quad ; \quad (4.36)$$

dabei ist $1 \leq P_i \leq P$ für alle $i = 1..n_D$. Eine solche Faktorzerlegung existiert für jedes $P \in \mathcal{N}^+$.² Nach dieser Faktorzerlegung von P wird dann ein Datenfeld der Größe

$$\mathcal{S}(\underline{Y}_C) = (N_{max,1} \cdot P_1) \times (N_{max,2} \cdot P_2) \times \dots \times (N_{max,n_D} \cdot P_{n_D}) \quad (4.37)$$

angelegt, in das die Datenfelder $\underline{Y}_{C,p}$ nach der Vorschrift

$$\begin{aligned} & \underline{Y}_C [p_1 \cdot N_{max,1} + r_{YC,p,1}, \\ & \quad p_2 \cdot N_{max,2} + r_{YC,p,2}, \\ & \quad \dots, \\ & \quad p_{n_D} \cdot N_{max,n_D} + r_{YC,p,n_D}] := \underline{Y}_{C,p} [r_{YC,p,1}, r_{YC,p,2}, \dots, r_{YC,p,n_D}] \end{aligned} \quad (4.38)$$

hineinkopiert werden. Dabei erhält man die p_i ($i = 1..n_D$) aus einer Zerlegung von p nach dem Horner-Schema

$$p = (\dots(p_n \cdot P_{n-1} + p_{n-1}) \cdot P_{n-2} + p_{n-2}) \dots \cdot P_1 + p_1 + 1 \quad (4.39)$$

mit $0 \leq p_i \leq P_i - 1$ und den P_i nach der Faktorzerlegung in (4.36). Für die vier Datenfelder aus dem Beispiel nach Abb. 4.6 wird in Abb. 4.8 gezeigt, wie die Einbettung in diesem Fall nach einer Faktorzerlegung von $P = 4 = 2 \cdot 2$ geschieht. Wie bereits erwähnt, ändert sich bei einem Zusammenfügen nach dieser Methode n nicht, es ist $n_C = n_D$.

Im folgenden werden die Datenfelder $\underline{Y}_{D,p}$ die **Feld-Partitionen** des Datenfeldes $\underline{Y}_C$ genannt, und zwar unabhängig davon, ob p als unabhängige oder als abhängige Variable gewählt wurde:

Definition 3 (Feld-Partitionierung): Es gelte, daß für verschiedene Felder $\underline{F}_a$ und $\underline{F}_b$ auch deren Referenzen verschieden sind, also die Referenzmengen $\mathcal{R}(\underline{F}_a)$ und $\mathcal{R}(\underline{F}_b)$ disjunkt sind.

Eine Anzahl P von Datenfeldern $\underline{F}_p$ mit $\mathcal{R}(\underline{F}_p) \neq \emptyset$ für alle $p = 1..P$ ist dann eine **Feld-Partitionierung** eines Datenfeldes $\underline{F}$, wenn gilt:

²Es existieren sogar mindestens n_D solcher Zerlegungen für jedes $P \in \mathcal{N}^+$. Beweis: Für jedes $P \in \mathcal{N}^+$ gilt: $P = 1 \cdot 1 \cdot \dots \cdot 1 \cdot P$ (Produkt aus n_D Faktoren). Mit Vertauschung der Faktoren erhält man die übrigen $n_D - 1$ Zerlegungen.

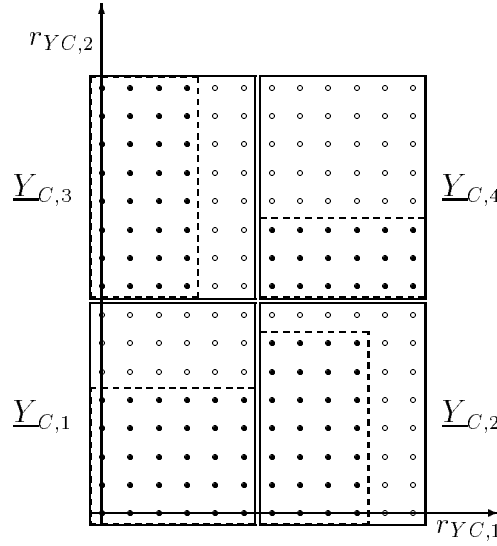


Abbildung 4.8: Einbetten von Datenfeldern: p als neue abhängige Variable

1. Es existiert eine eindeutige Abbildung g **von**³ der Menge

$$\mathcal{R}_{\cup P}(\underline{F}_p) = \bigcup_{p=1}^P \mathcal{R}(\underline{F}_p)$$

in die Menge $\mathcal{R}(\underline{F})$:

$$g : \mathcal{R}_{\cup P}(\underline{F}_p) \rightarrow \mathcal{R}(\underline{F}) \quad ,$$

mit anderen Worten: g ist eine injektive Abbildung, die eine Referenz $R_{F,p}$ in $\underline{F}_p$ **in** eine Referenz R_F in $\underline{F}$ abbildet. Für den Definitions- und Wertebereich von g gilt:

$$\begin{aligned} \mathcal{D}(g) &= \mathcal{R}_{\cup P}(\underline{F}_p) \quad , \\ \mathcal{W}(g) &\subseteq \mathcal{R}(\underline{F}) \quad . \end{aligned}$$

Die Menge $\mathcal{L}(\underline{F}) = \mathcal{R}(\underline{F}) \setminus \mathcal{W}(g)$ wird die Menge der unbestimmten Referenzen des Feldes $\underline{F}$ genannt.

2. Für alle Referenzen $R_F \in \mathcal{L}(\underline{F})$ gilt:

$$\underline{F}[R_F] = \underbrace{(\perp, \dots, \perp)}_{n \text{ Komponenten}}$$

Die Datenfelder $\underline{F}_p$ heißen **Feld-Partitionen** von $\underline{F}$.

³siehe auch [IBro89, S. 550]

Die Abbildung g der Definition für die verteilten Datenfelder $\underline{Y}_{D,p}$ in das Datenfeld $\underline{Y}_C$ ergibt sich aus den Gleichungen (4.32) und (4.33), sowie aus (4.35) oder (4.38), je nachdem, ob p als neue unabhängige Variable (Fall A) oder als abhängige Variable (Fall B) gezählt wird.

Wie die Datenfelder $\underline{Y}_{D,p}$ können die nach dem *private memory*-Prinzip verteilten Datenfelder der unabhängigen Variable $\underline{X}_{D/pr,p}$ analog zu den Vorschriften nach Gleichung (4.35) (Fall A) oder (4.38) (Fall B) in ein Datenfeld $\underline{X}_C$ eingebettet werden. Das entstehende Datenfeld $\underline{X}_C$ ist genauso groß wie das Datenfeld $\underline{Y}_C$; zudem sind die den unbestimmten Referenzen in $\underline{Y}_C$ entsprechenden Referenzen in $\underline{X}_C$ ebenfalls unbestimmt, denn in Gleichung (4.26) wurden die Felder $\underline{X}_{D,p}$ und $\underline{Y}_{D,p}$ als jeweils gleich groß angesetzt.

Liegt die unabhängige Variable dagegen nach dem *shared memory*-Prinzip in einem einzigen Datenfeld $\underline{X}_{D/sh}$ vor, dann wird dieses einzelne Datenfeld der Größe

$$\mathcal{S}(\underline{X}_{D/sh}) = M_1 \times M_2 \times \dots \times M_{n_D} \quad (4.40)$$

in P Feld-Domänen $\underline{X}_{D/sh,p}$ aufgeteilt:

Definition 4 (Aufteilung in Feld-Domänen): Eine Anzahl D von Datenfeldern $\underline{F}_d$ mit $\mathcal{R}(\underline{F}_d) \neq \emptyset$ für alle $d = 1..D$ ist dann eine Aufteilung des Datenfeldes $\underline{F}$ in **Feld-Domänen**, wenn gilt:

Es existiert eine eindeutige Abbildung g **von**⁴ der Menge

$$\mathcal{R}_{\cup D}(\underline{F}_d) = \bigcup_{d=1}^D \mathcal{R}(\underline{F}_d)$$

auf die Menge $\mathcal{R}(\underline{F})$

$$g : \mathcal{R}_{\cup D}(\underline{F}_d) \rightarrow \mathcal{R}(\underline{F}) \quad ,$$

mit anderen Worten: g ist eine Funktion, die eine Referenz $R_{F,d}$ in $\underline{F}_d$ **auf** eine Referenz R_F in $\underline{F}$ abbildet. Für den Definitions- und Wertebereich von g gilt:

$$\begin{aligned} \mathcal{D}(g) &= \mathcal{R}_{\cup D}(\underline{F}_d) \\ \mathcal{W}(g) &= \mathcal{R}(\underline{F}) \quad . \end{aligned}$$

Die Datenfelder $\underline{F}_d$ heißen **Feld-Domänen** von $\underline{F}$.

Die Feld-Domänen werden jeweils so aus dem Feld $\underline{Y}_{D/sh}$ ausgewählt, daß jeweils eine Feld-Domäne $\underline{X}_{D/sh,p}$ ($p = 1..P$) die Instanzen von X enthält, die zu den Instanzen von Y im Datenfeld $\underline{Y}_{D,p}$ gehören; die Felder $\underline{X}_{D/sh,p}$ und $\underline{Y}_{D,p}$ sollen also die gleiche Größe

$$\begin{aligned} \mathcal{S}(\underline{X}_{D/sh,p}) &= \mathcal{S}(\underline{Y}_{D,p}) \\ &= N_{p,1} \times N_{p,2} \times \dots \times N_{p,n_D} \quad , \end{aligned} \quad (4.41)$$

⁴siehe ebenfalls [IBro89, S.550]

haben; damit könnten sie paarweise zu Feld-Datenstrukturen $\mathcal{F}(\underline{X}_{D/sh,p}, \underline{Y}_{D,p})$ zusammengefaßt werden. Die resultierenden Feld-Domänen $\underline{X}_{D/sh,p}$ können nun genauso in das Datenfeld $\underline{X}_C$ eingebettet werden, wie dies für die Felder $\underline{Y}_{D,p}$ und $\underline{Y}_C$ gezeigt wurde.

Abb. 4.9 zeigt eine Aufteilung eines Feldes $\underline{X}_{D/sh}$ in vier Feld-Domänen $\underline{X}_{D/sh,p}$, die genau die gleiche Größe haben, wie die Datenfelder $\underline{Y}_{D,p}$ aus Abb. 4.6.

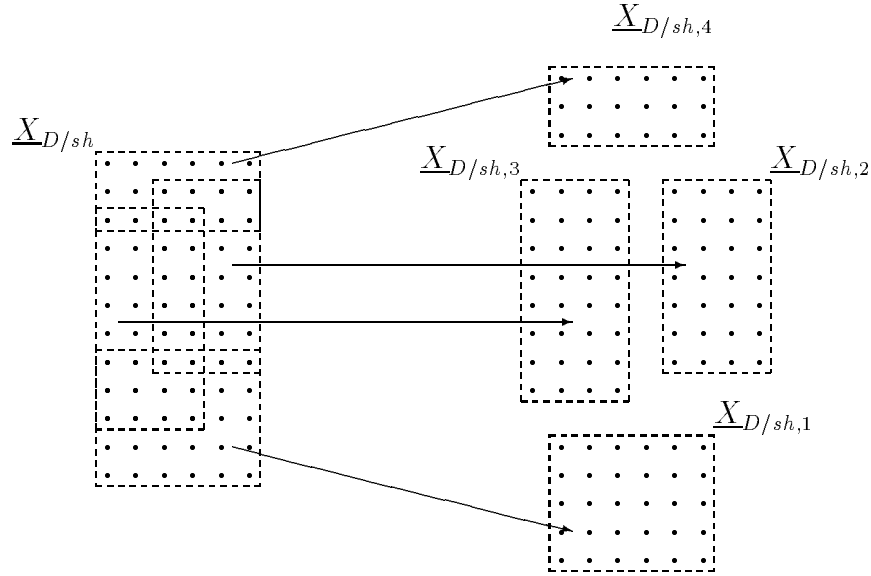


Abbildung 4.9: Aufteilung in Feld-Domänen

Die Einbettung der verteilten Daten in eine Feld-Datenstruktur kann in ungünstigen Fällen unverhältnismäßig viel Speicherplatz kosten, wenn nämlich das Maximum $N_{max,i}$ aller Feld-Größen sehr viel größer ist als alle anderen $N_{p,i}$. Dann wird das Verhältnis der unbestimmten Referenzen in $\underline{Y}_C$ und $\underline{X}_C$ zu den belegten Referenzen im Datenfeld sehr groß und der Speicher wird unzureichend genutzt. Nun existieren jedoch Feld-Partitionierungen und Aufteilungen in Feld-Domänen, für die sowohl $\mathcal{L}(\underline{Y}_C) = \emptyset$, als auch $\mathcal{L}(\underline{X}_C) = \emptyset$ gilt:

Definition 5 : Eine Aufteilung in Feld-Partitionen oder Feld-Domänen heißt **regulär**, wenn die Implikation

$$p_i = q_i \quad \Rightarrow \quad N_{p,i} = N_{q,i}$$

für beliebige Teilfelder $\underline{F}_p$ und $\underline{F}_q$ von $\underline{F}$ gilt. Dabei stammen p_i und q_i aus den Horner-Schemata für p und q im Sinne der Faktorzerlegung $P = P_1 \cdot P_2 \cdot \dots \cdot P_n$ ($p, q = 1..P$):

$$\begin{aligned} p &= (\dots(p_n \cdot P_{n-1} + p_{n-1}) \cdot P_{n-2} + p_{n-2}) \dots \cdot P_1 + p_1 + 1 \\ q &= (\dots(q_n \cdot P_{n-1} + q_{n-1}) \cdot P_{n-2} + q_{n-2}) \dots \cdot P_1 + q_1 + 1 \end{aligned}$$

Für den Fall, daß alle Feld-Partitionen $\underline{F}_p$ gleich groß sind, wird die Feld-Partitionierung **streng regulär** genannt.

Wenn die Einbettung der Datenfelder nach Fall B geschieht, läßt sich bei einer solchen regulären Partitionierung oder Aufteilung Speicherplatz sparen, indem die Feld-Partitionen des Feldes $\underline{Y}_C$ und $\underline{X}_C$ nahtlos aneinander gefügt werden; die Menge der unbestimmten Referenzen $\mathcal{L}(\underline{Y}_C)$ und $\mathcal{L}(\underline{X}_C)$ ist dann tatsächlich gleich der leeren Menge ($\emptyset$). Ein Beispiel für eine reguläre Feld-Partitionierung zeigt Abb. 4.10. Die Faktorzerlegung von P lautet dort $P = 6 = 3 \cdot 2$, und p bestimmt sich aus $p = p_2 \cdot 3 + p_1 + 1$.

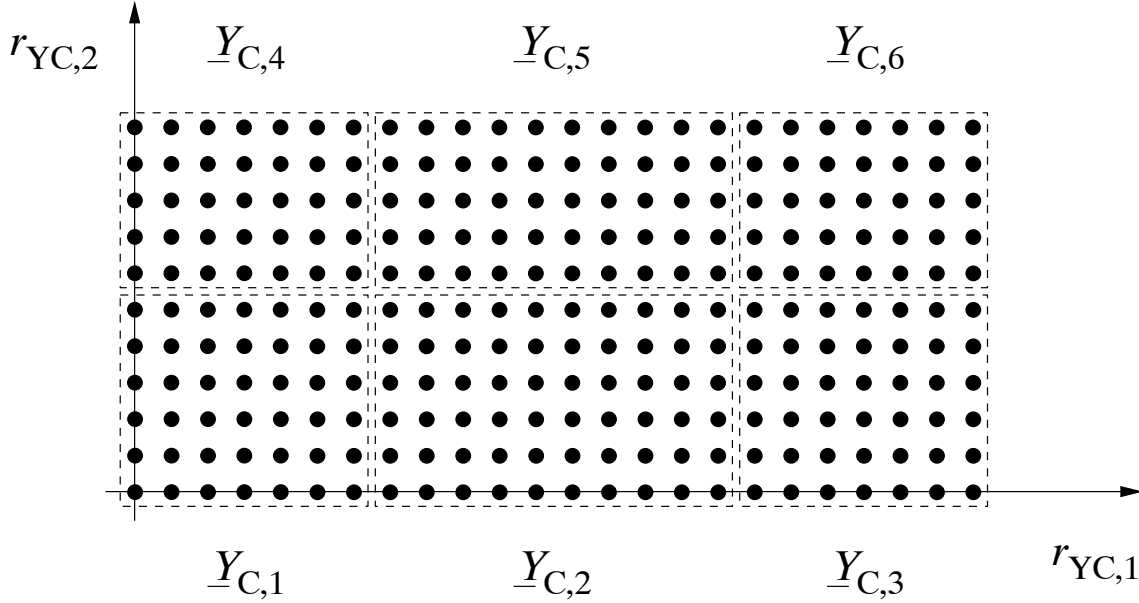


Abbildung 4.10: Reguläre Feld-Partitionierung

Die Größe der Datenfelder $\underline{Y}_C$ und $\underline{X}_C$ setzt sich in diesem Fall aus den Summen der Seitenlängen der Feld-Partitionen längs einer Dimension zusammen:

$$\begin{aligned} \mathcal{S}(\underline{Y}_C) &= \mathcal{S}(\underline{X}_C) \\ &= \sum_{p=0}^{P_1-1} N_{(p,0,\dots,0),1} \times \sum_{p=0}^{P_2-1} N_{(0,p,\dots,0),2} \times \dots \times \sum_{p=0}^{p_{n_D}-1} N_{(0,0,\dots,p),n_D} \quad (4.42) \end{aligned}$$

Die einzelnen Datenfelder $\underline{Y}_{D,p}$ können direkt, also ohne den Zwischenschritt über die Datenfelder $\underline{Y}_{C,p}$, in das Feld $\underline{Y}_C$ eingebettet werden:

$$\begin{aligned} \underline{Y}_C \Big[&\sum_{p'=0}^{P_1-1} N_{(p',0,\dots,0)} + r_{YD,p,1}, \\ &\sum_{p'=0}^{P_2-1} N_{(0,p',\dots,0)} + r_{YD,p,2}, \\ &\dots, \\ &\sum_{p'=0}^{p_{n_D}-1} N_{(0,0,\dots,p')} + r_{YD,p,n_D} \Big] := \underline{Y}_{D,p} [r_{YD,p,1}, r_{YD,p,2}, \dots, r_{YD,p,n_D}] \quad (4.43) \end{aligned}$$

Nachdem die verteilten Datenfelder in die Datenfelder $\underline{X}_C$ und $\underline{Y}_C$ nach den Gleichungen (4.35), (4.38) oder (4.43) eingebettet sind, gehört zu jeder Instanz der Datenmenge X_C auch eine Instanz der Datenmenge Y_C und umgekehrt; beide Mengen enthalten gleich viele Elemente. Zwischen beiden Mengen besteht eine eindeutige Abbildung $r_C : R_{X_C} \rightarrow R_{Y_C}$, mit denen die Wert-Instanzen den Ort-Instanzen zugeordnet werden. Damit ist mit der Einbettung die Mehrdeutigkeit von r_D aufgehoben.

Die Feld-Datenstruktur $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$ beinhaltet jedoch noch nicht alle Informationen über die verteilte Datenmenge, denn die Information über die ursprüngliche Verteilung der Daten ging zumindest zum Teil verloren. Deshalb soll im nächsten Abschnitt untersucht werden, wie die Aussagen über die Verteilung der Daten mit einer eigenen Datenstruktur (oder sogar innerhalb der Feld-Datenstruktur selbst !) codiert werden können.

4.3.4 Partitions- und Domänengrenzen

Um sowohl Anforderung 2 als auch die Ausdrucksbedingungen nach Mackinlay (siehe Abschnitt 3.2.4) erfüllen zu können, muß die Information über die ursprüngliche Verteilung der Daten innerhalb der Darstellung codiert sein. Für die Konzeption der Datenstrukturen, die diese Verteilungsinformation bis hin zur Erzeugung der Ansicht übermitteln sollen, muß die abschließende Darstellung mit einbezogen werden, denn erst anhand der Darstellung läßt sich überprüfen, ob alle Aussagen über die Verteilung der Daten in der Darstellung umgesetzt werden konnten oder nicht. Wie bereits in Abschnitt 4.2 erwähnt wurde, kann von den Daten bereits dann eine Ansicht erzeugt werden, sobald sie in die Feld-Datenstruktur $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$ eingebettet wurden. Mit der Erzeugung einer vorläufigen Ansicht nach der Einbettung kann folglich kontrolliert werden, welche Aussagen über die Verteilung der Daten bereits in $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$ codiert sind, und welche Aussagen noch mit der Modellierung zur Feld-Datenstruktur $\mathcal{F}(\underline{X}_M, \underline{Y}_M)$ in die Darstellung umzusetzen sind.

Zu dieser Überprüfung werden zwei Kriterien angesetzt:

1. Daten verschiedener Partitionen sollen voneinander räumlich getrennt und Daten einer Partition gemeinsam dargestellt werden.
2. Die Überlappzonen sollen als die gemeinsamen Bereiche zwischen den Domänen erkennbar sein.

Im Fall A, also der Wahl von p als neuer unabhängiger Variablen, ist eine Trennung zwischen den Werten verschiedener Partitionen in der Darstellung zu erkennen, denn wegen $m'_C = m'_D + 1$ werden die einzelnen Partitionen in bestimmten Abständen entlang einer Achse des Darstellungsraumes aufgetragen. Beispielsweise ergibt sich für $n_D = m'_D = 2$, also für zweidimensionale Feld-Partitionen, eine dreidimensionale Darstellung mit $n_C = m'_C = 3$, in der alle Partitionen hintereinander an der z-Achse aufgereiht dargestellt werden. Eine solche Darstellung erfüllt das erste Kriterium,

denn einerseits werden Daten aus verschiedenen Partitionen voneinander getrennt dargestellt (die Partitionsnummer p ist diskret, es gibt daher keinen kontinuierlichen Übergang zwischen den Partitionen), andererseits werden aber die Daten einer Feld-Partition zusammen dargestellt (nämlich auf der gleichen xy -Ebene). Ein Beispiel zeigt Abb. 4.11a.

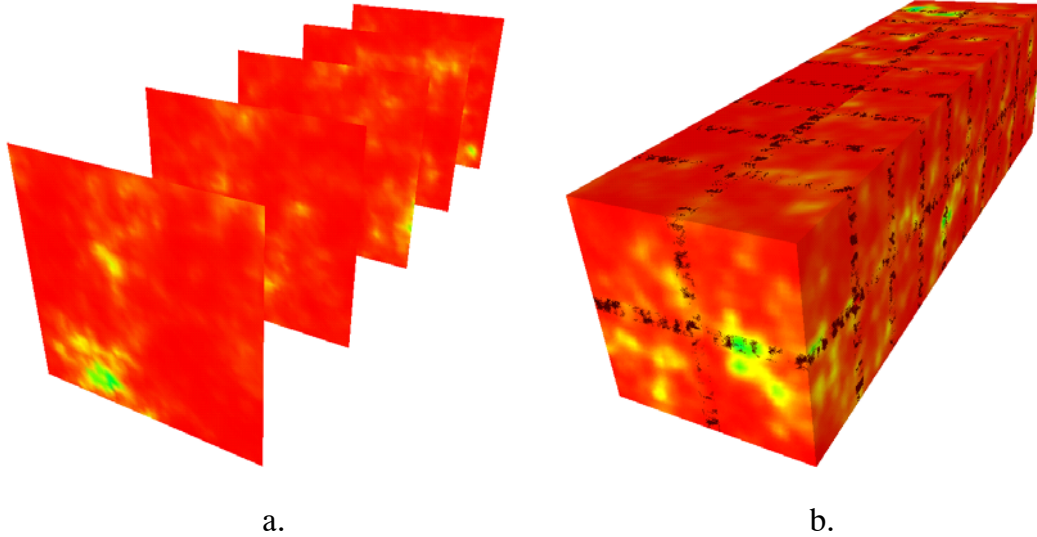


Abbildung 4.11: Einbettung nach Fall A und nach Fall B

Die gemeinsamen Bereiche zwischen den Domänen sind jedoch in dieser Darstellung (noch) nicht zu erkennen. Sie müssen noch durch einen Modellierschritt hervorgehoben werden.

Für den Fall B, also der Wahl von p zu einer neuen abhängigen Variablen, lassen sich die Überlappzonen zwischen den Domänen schwach erkennen, wie Abb. 4.11b für ein dreidimensionales Beispiel eines Skalarfeldes zeigt ($m'_C = 3$). Weil zwar für jede Instanz von Y_C eine Instanz X_C vorhanden ist und umgekehrt, aber die Orte in den Überlappzonen mehrfach als Instanzen von X_C vertreten sind, hat die Erzeugung der Ansicht zur Folge, daß gleichzeitig mehrere Werte an einem Ort dargestellt werden müssen. Dabei entstehen Interferenzerscheinungen, durch die die gemeinsamen Gebiete zwischen den Domänen gekennzeichnet sind.

Die Grenzen zwischen den Domänen sind jedoch in der Darstellung nicht zu erkennen. Erst im nachfolgenden Modellierschritt könnten beispielsweise die Koordinaten der unabhängigen Variablen so manipuliert werden, daß die Domänen voneinander abgerückt dargestellt werden können. Dazu bedarf es zuvor jedoch noch eines kleinen Tricks, um die Domänen wirklich getrennt voneinander darstellen zu können: üblicherweise versucht nämlich das Visualisierungssystem bei der Erzeugung einer Ansicht mit kontinuierlichen Mapping-Verfahren zwischen den Stützstel-

len zu interpolieren⁵. Folglich würde auch zwischen denjenigen Stützstellen interpoliert werden, die verschiedenen Domänen angehören, wie Abb. 4.12a es schematisch für $n = 1$ zeigt. Zwischen den beiden Domänen wird der Wert linear interpoliert. Damit würde das erste Kriterium verletzt, denn die Daten verschiedener Domänen werden nicht eindeutig voneinander getrennt dargestellt.

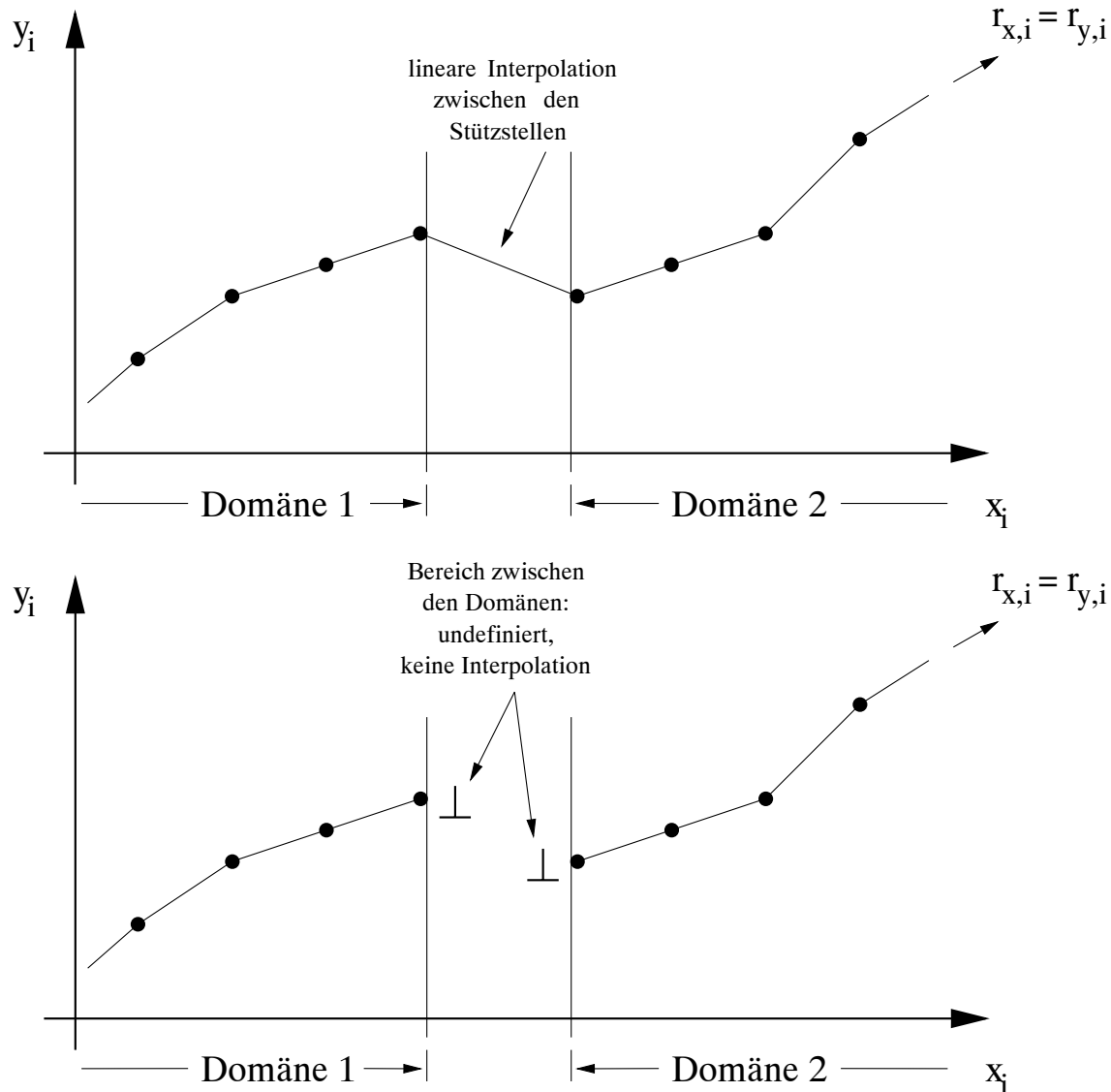


Abbildung 4.12: Interpolation zwischen Domänen, ohne und mit Leerstellen

Um dies zu verhindern, werden Leerstellen so zwischen den Feld-Partitionen der Datenfelder $\underline{X}_C$ und $\underline{Y}_C$ eingefügt, wie Abb. 4.13 für die reguläre Feld-Partitionierung

⁵Die Interpolation zwischen den gegebenen Elementen von X_C und Y_C ist also nicht nur eine Aufgabe im Modellerschritt, sondern wird auch von den *kontinuierlichen* Mapping-Verfahren durchgeführt.

$$\begin{aligned}
&= \left(\sum_{p=0}^{P_1-1} N_{(p,0,\dots,0),1} + 2 \cdot \ell \cdot (P_1 - 1) \right) \\
&\times \left(\sum_{p=0}^{P_2-1} N_{(0,p,\dots,0),2} + 2 \cdot \ell \cdot (P_2 - 1) \right) \\
&\times \dots \\
&\times \left(\sum_{p=0}^{P_{n_D}-1} N_{(0,0,\dots,p),n_D} + 2 \cdot \ell \cdot (P_{n_D} - 1) \right) \quad (4.45)
\end{aligned}$$

bei regulären Feld-Partitionierungen. Der Parameter ℓ ist gleich eins, wenn nur die Domänen voneinander getrennt werden; er ist gleich drei, wenn zusätzlich auch die Domänen und ihre jeweiligen Überlappzonen in der Ansicht getrennt werden sollen.

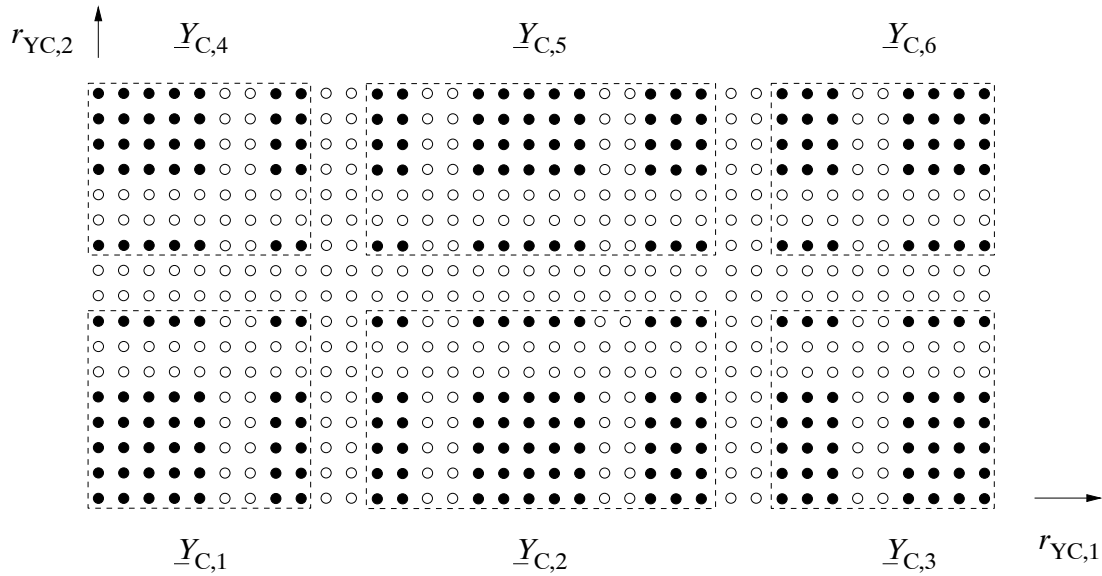


Abbildung 4.14: Leerstellen zwischen den Feld-Partitionen und den Überlappzonen

Für einen Modellerschritt wie zum Beispiel dem Entfernen der Domänen voneinander muß die Lage der Feld-Partitionen im Feld $\underline{X}_C$ bekannt sein, damit die "richtigen" Elemente von X_C "richtig" manipuliert werden. Auch andere Methoden werden solche Angaben benötigen, wie im nächsten Abschnitt noch erläutert wird. Deshalb erscheint es sinnvoll, eine Datenstruktur aufzustellen, in der alle wichtigen Informationen über jede einzelne Feld-Partition enthalten sind. Zu diesen Informationen gehören (bei regulären Feld-Partitionen):

1. die **Nummer** oder ID p der Domäne;
2. die **Zerlegung** $(p_1, \dots, p_n)$ von p nach dem Horner-Schema (siehe Gleichung (4.39));

3. die **"kleinste" Referenz** $R_{XC,p,min}$ der Feld-Partition p , d.h. die Referenz mit den kleinsten Komponenten $r_{XC,p,i}$ für $i = 1..n$;
4. die **"größte" Referenz** $R_{XC,p,max}$ der Feld-Partition, also entsprechend 3 die Referenz mit den größten Komponenten $r_{XC,p,i}$ für $i = 1..n$. Alternativ könnte auch die Größe der Feld-Partition $\mathcal{S}(\underline{X}_{C,p})$ angegeben werden.
5. die **Breite der Überlappzone** für jede "Seite" der Feld-Partition. In einem n -dimensionalen Datenfeld $\underline{X}_{D/sh}$ kann eine Feld-Domäne von $2 \cdot n$ Seiten überlappt werden, wenn die Aufteilung regulär ist. Entsprechend werden also $2 \cdot n$ Angaben für jede Feld-Partition nötig sein, mit denen die Breite der Überlappzone als eine Anzahl von Referenzen R_{XC} festgelegt wird.

Für die Feld-Partition $\underline{Y}_{C,2}$ in Abb. 4.13 könnte die Datenstruktur in C-Notation dann wie folgt aussehen und belegt werden:

```
#define LOWER 0
#define UPPER 2
#define LEFT 1
#define RIGHT 3

typedef struct _ref_2d { /* Referenz im zweidimensionalen Fall */
    int r1, r2;          /* Referenz besteht aus zwei Komponenten */
} ref_2d;

typedef struct _f_part_2d { /* Datenstruktur einer Feld-Partition */
    int id;              /* ID einer Feld-Partition */
    int p[2];            /* p nach Horner-Schema zerlegt */
    ref_2d minref, maxref; /* kleinste und groesste Referenz */
    int width[4];        /* zweidimensionale Feld-Partition */
} f_part_2d;            /* => vier Ueberlappzonen */
                        /* width: Breite der Ueberlappzone */
                        /*          in Knoten */

:
f_part_2d xc_partition[12];
:

xc_partition[1].id = 2;
xc_partition[1].p[0]=1; xc_partition[1].p[1]=0;
xc_partition[1].minref.r1 = 9; xc_partition[1].minref.r2 = 0;
xc_partition[1].maxref.r1 = 18; xc_partition[1].maxref.r2 = 5;
xc_partition[1].width[LOWER] = 0; xc_partition[1].width[UPPER] = 1;
xc_partition[1].width[LEFT] = 2; xc_partition[1].width[RIGHT] = 3;
```

Wie dieses Beispiel zeigt, müssen 11 Angaben zu je 4 Byte (normale Integerdarstellung) spezifiziert werden; pro Feld-Partition werden also 44 Byte an Verwaltungsinformation fällig, für das Beispiel mit 12 Feld-Partitionen in Abb. 4.13 also insgesamt 528 Bytes. Die gleiche Information läßt sich jedoch auch mit erheblich weniger Speicheraufwand übermitteln, und zwar, indem die Struktur der regulären Feld-Partitionierung ausgenutzt wird. Statt jede Partition einzeln zu spezifizieren, werden die Leerstellen bzw. Trennungen g zwischen den Feld-Partitionen festgelegt, und zwar durch

1. ihre **Orientierung** i_g – die Leerstellen / die Trennmarkierung liegt ”senkrecht zur r_{XC,i_g} -Achse”;
2. ihre **Position** s_g – die Trennung zwischen zwei Feld-Partitionen liegt quer zur r_{XC,i_g} -Achse bei $r_{XC,i_g} = s_g$ und $r_{XC,i_g} = s_g + 1$;
3. ihre **Breite** w_g – die Trennung zwischen zwei Feld-Partitionen liegt quer zur r_{XC,i_g} -Achse bei $r_{XC,i_g} = s_g$ und $r_{XC,i_g} = s_g + 1$, die zugehörige Überlappzone umfaßt w_g Referenzen beiderseits der Trennung, und zwar von $r_{XC,i_g} = s_g - w_g$ bis $r_{XC,i_g} = s_g + 1 + w_g$, siehe Abb. 4.13.

In C-Notation könnte damit die gesamte Feld-Partitionierung aus Abb. 4.13 codiert werden durch die Zeilen

```
typedef _gap_type { /* Leerstellen/Trennmarkierungen zwischen den
                    Feld-Partitionen */
    int orientation; /* Orientierung */
    int position;    /* Position */
    int width;       /* Breite in Knoten */
} gap_type;

:
gap_type gaps[3];
:

gaps[0].orientation = 0; gaps[0].position = 7; gaps[0].width = 2;
gaps[1].orientation = 0; gaps[1].position = 19; gaps[1].width = 3;
gaps[2].orientation = 1; gaps[2].position = 6; gaps[2].width = 1;
```

Zur Repräsentation einer regulären Feld-Partitionierung durch die Trennmarkierungen werden $3 \cdot 4$ Bytes pro Trennmarkierung benötigt; bei einer Feld-Partitionierung nach einer Faktorzerlegung von $P = P_1 \cdot P_2 \cdot \dots \cdot P_n$ also insgesamt

$$\left(\sum_{i=1}^n (P_i - 1) \right) \cdot 12 \quad \text{Bytes,} \quad (4.46)$$

denn in jeder Richtung gibt es $P_i - 1$ Trennungen durch das Datenfeld. Für die in Abb. 4.14 gezeigte Feld-Partitionierung werden also nur $(3 + 2) \cdot 12 = 60$ Bytes benötigt. Der Nachteil der Ablage der Feld-Partitionierung als schnitt-orientierte Datenstruktur liegt darin, daß die Daten aller Partitionen, wie sie in einer Datenstruktur `f_part_2d` unmittelbar vorliegen, zuerst noch abgeleitet werden müssen. Die Vorteile sind der geringere Speicheraufwand und der geringere Kommunikationsaufwand beim Austausch dieser Datenstrukturen zwischen den Modulen.

4.3.5 Modellieren der Variablen X_C und Y_C

Mit der Einbettung der verteilten Datenfelder in die Feld-Datenstruktur $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$ konnte erreicht werden, daß jeder Wert-Instanz Y_C eine Orts-Instanz X_C zugeordnet ist. Damit kann von $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$ bereits eine Ansicht erzeugt werden, wie in Abb. 4.15 für ein Skalarfeld gezeigt wird. Als problematisch für die Visualisierung

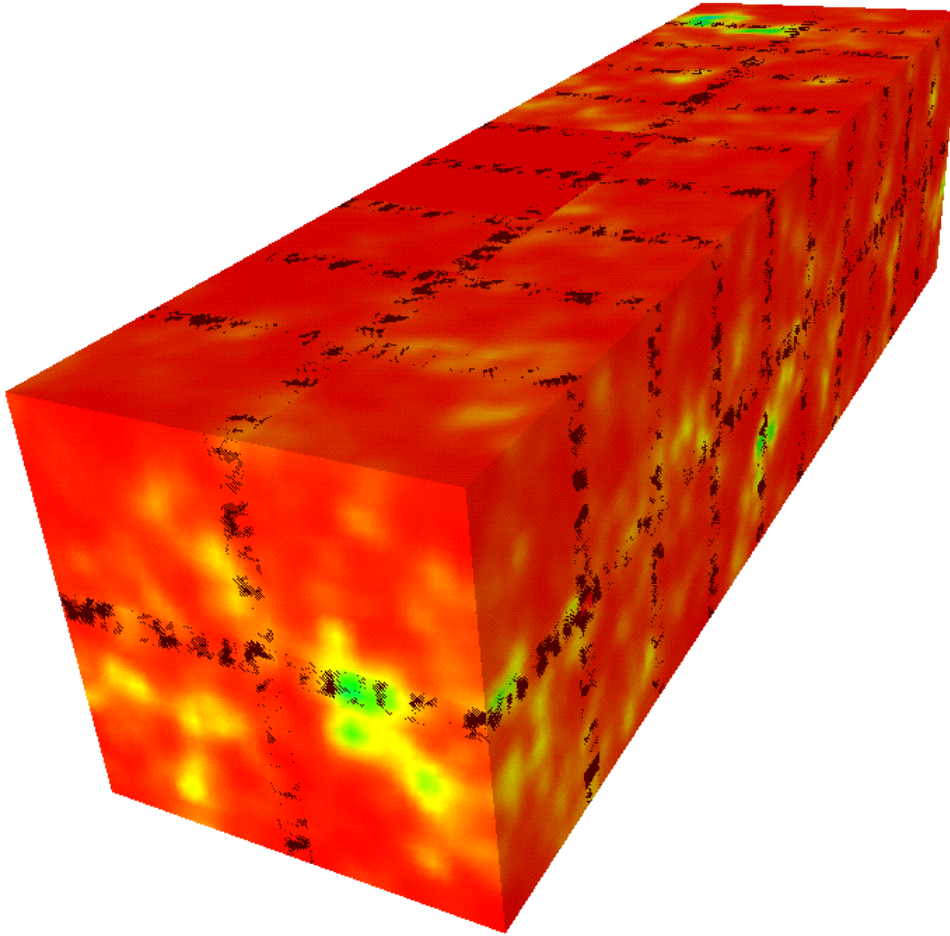


Abbildung 4.15: Ansicht von einer Feld-Datenstruktur $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$

stellen sich jedoch diejenigen Gebiete heraus, in denen sich zwei oder mehr Domänen überlappen. Dort gibt es Interferenzerscheinungen, bei denen das Visualisierungssystem nicht klar entscheiden kann, welchen der k Werte es am Ortspunkt darstellen soll. Das Problem liegt in der Mehrdeutigkeit der Abbildung $r_{XC}^{-1} : X_C \xrightarrow{k} R_{XC}$, mehrere Referenzen R_{XC} können auf ein und denselben Ort verweisen.

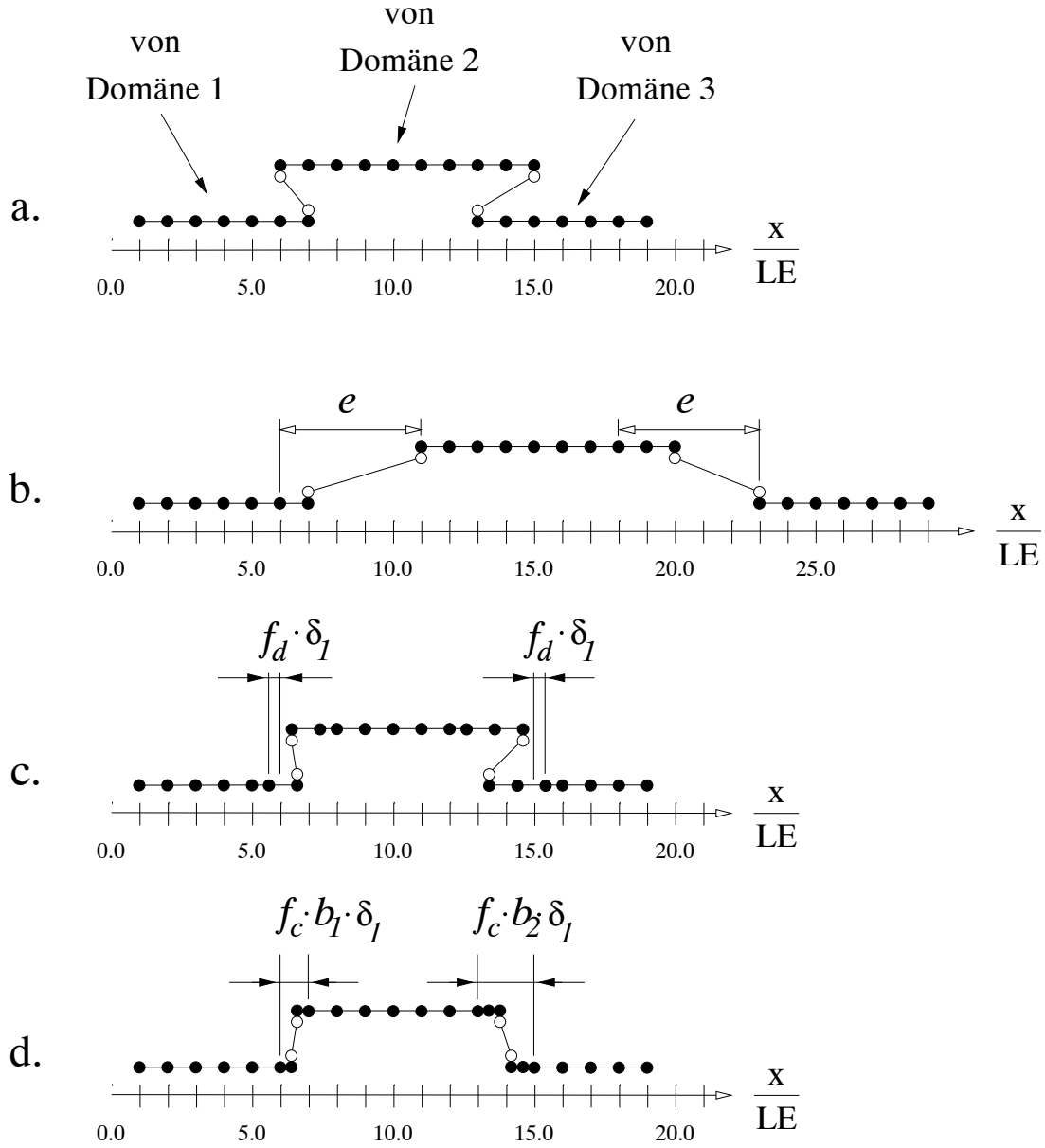


Abbildung 4.16: Koordinatenfolgen: Grundzustand (a), Explosionszeichnung (b), Verrücken um einen Versatz (c), Zusammenziehen der Überlappzonen (d)

Diese Mehrdeutigkeit lässt sich durch eine Manipulation der unabhängigen Variablen im Modellerschritt beheben. Dies sei am Beispiel einer Explosionszeichnung gezeigt.

Abb. 4.16a zeigt eine Situation für das regulär partitionierte Feld aus Abb. 4.13. Der Einfachheit halber sei angenommen, daß die Feld-Datenstruktur implizit beschrieben ist, und deshalb das Datenfeld $\underline{X}_C$ aus zwei Koordinatenfolgen besteht: eine Folge für die Koordinaten in x-Richtung und eine Folge für die Koordinaten in y-Richtung. $\underline{X}_C$ ist damit $\mathcal{S}(\underline{X}_C) = 28 \times 2$ Elemente groß, denn es gibt 28 Referenzen $r_{YC,1}$ im Datenfeld $\underline{Y}_C$, siehe Abb. 4.13.

Die Abb. 4.16a zeigt die Koordinatenfolge $x_{C,1}(r_{XC,1})$. Die Kreise über dem Zahlenstrahl stellen die Referenzen dar; jede der 28 Referenzen befindet sich oberhalb derjenigen Koordinate auf dem Zahlenstrahl, worauf sie verweist. Die Kreise sind als Kette miteinander verbunden, damit soll die Reihenfolge der Referenzen $r_{XC,1} = 0..M_i - 1 = 0..27$ verdeutlicht werden. Wie der Abbildung Abb. 4.16a entnommen werden kann, gibt es im Grundzustand zwei Überlappgebiete zwischen den Koordinaten 6.0 LE und 7.0 LE sowie zwischen 13.0 LE und 15.0 LE. Die eine Überlappzone ist 2 Knoten, die andere drei Knoten breit. Jeweils zwei Referenzen verweisen dort auf die gleiche Koordinate. Die unabhängige Variable wird nun in Abb. 4.16b so manipuliert / modelliert, daß die Domäne 2 um 5.0 Längeneinheiten und die Domäne 3 um 10.0 Einheiten nach rechts verschoben werden. Dieser Vorgang läßt sich allgemein mit der folgenden Zuweisungsvorschrift beschreiben:

$$\underline{X}_M[r_{X,i}, i] := \underline{X}_C[r_{X,i}, i] + p_i \cdot e \cdot \delta_i \quad \text{für alle } r_{X,i} = 0..M_i - 1. \quad (4.47)$$

In dieser Gleichung bezeichnet $p_i = p_i(r_{X,i})$ eine fortlaufende Nummer, mit der die Domänen in i -Richtung durchnumeriert werden, $p_i = 0..P_i$, siehe Gleichung (4.39). Mit e wird die Anzahl der Elemente bezeichnet, um die die Domänen voneinander entfernt dargestellt werden sollen, und δ_i stellt ein Skalierungsfaktor zwischen Referenz- und Darstellungsraum dar; dazu könnte beispielsweise bei einem streng regulären Gitter der Abstand zwischen den Knoten in den drei Raumrichtungen verwendet werden.

In der Abb. 4.17b wird das Ergebnis dieses Modellierschrittes mit einer Ansicht eines Skalarfeldes dargestellt. Wie dort ersichtlich ist, überlappen sich die Domänen nicht mehr.

Ein Nachteil dieses Modellierschrittes nach Art einer Explosionszeichnung besteht darin, daß die Außenabmessungen des Gesamtfeldes nicht mehr mit denen des ursprünglichen Feldes übereinstimmen, vergleiche Abb. 4.17a. Es gibt jedoch Modellierschritte, die die Größe des Gesamtgebietes nicht verändern: zum einen das Verrücken der Punkte in den Überlappzonen um einen kleinen Versatz von f_d Elementen ($0 \leq f_d \leq 1$), siehe dazu Abb. 4.16c:

$$\underline{X}_M[r_{X,i}, i] := \underline{X}_C[r_{X,i}, i] \pm f_d \cdot \delta_i \quad \text{für alle } r_{X,i} \text{ in einer Überlappzone;} \quad (4.48)$$

dabei hängt das Vorzeichen davon ab, ob die Referenz zu einer "linken" Überlappzone gehört (dann wird der Versatz $f_d \cdot \delta_i$ hinzuaddiert) oder zu einer "rechten" Überlappzone (dann wird der Versatz $f_d \cdot \delta_i$ von der ursprünglichen Koordinate subtrahiert). Das Resultat des Modellierschrittes zeigt Abb. 4.17c.

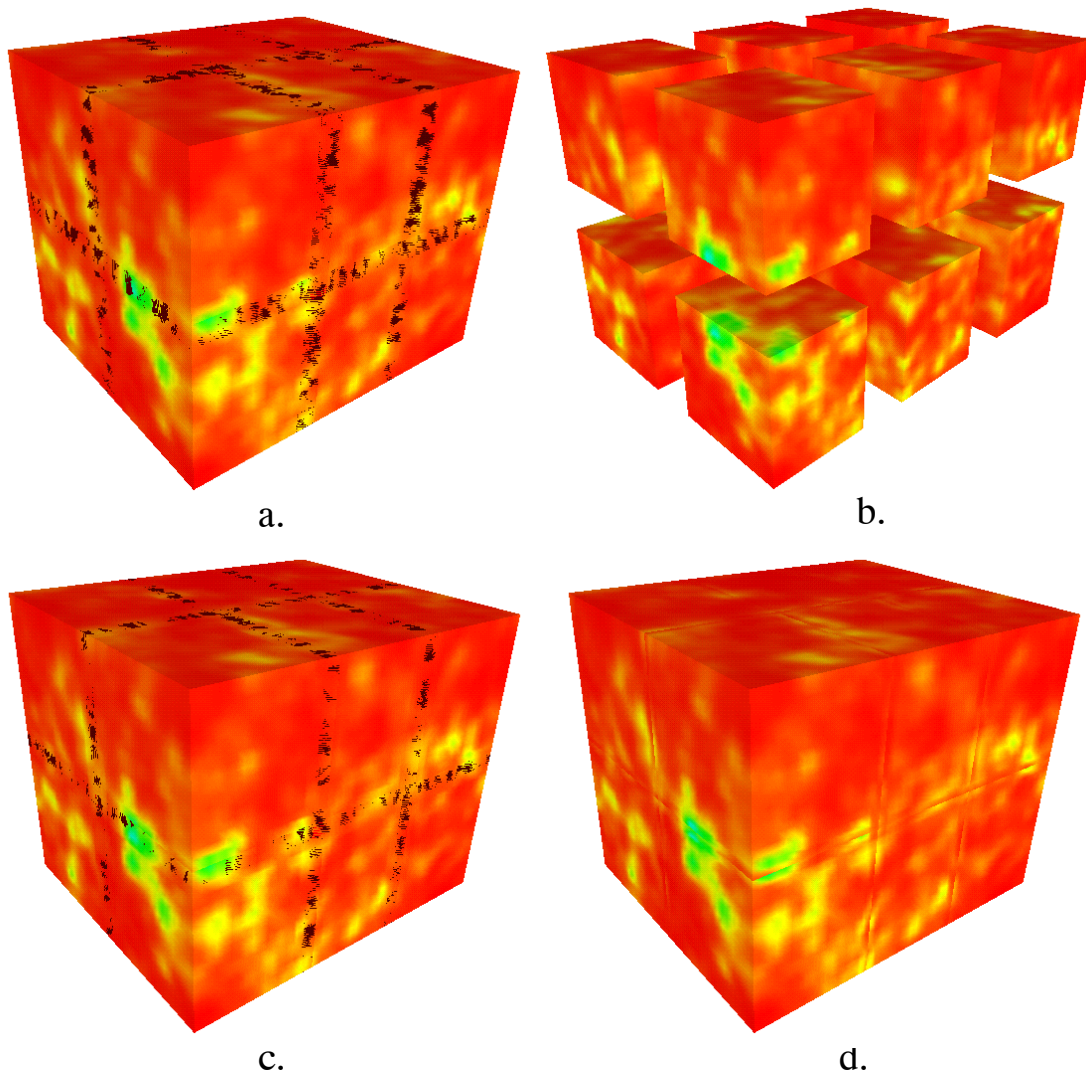


Abbildung 4.17: Ansichten nach den Modellierschritten: Grundzustand (a), Explosionszeichnung (b), Versatz (c), Kompression (d)

Der andere Modellierschritt, der ebenfalls nicht die Abmessungen des Gesamtfeldes verändert, ist die Kompression der Überlappzonen um einen Faktor f_c ($0 \leq f_c \leq 1$) (siehe Abb. 4.16d):

$$\underline{X}_M[r_{X,i}, i] := \underline{X}_C[r_{X,i}, i] \pm f_c \cdot b_g \cdot \delta_i \quad \text{für alle } r_{X,i} \text{ in einer Überlappzone; } (4.49)$$

dabei gilt auch hier das positive Vorzeichen für den Fall, daß die Referenz $r_{X,i}$ zu einer "linken Überlappzone" gehört, und das negative Vorzeichen für den Fall, daß $r_{X,i}$ zu einer "rechten" Überlappzone gehört.

Bei einer Modellierung nach den Vorschriften "Verrücken der Überlappzonen um einen Versatz" oder "Kompression der Überlappzonen" besteht jedoch keine

Möglichkeit, die Gesamtgeometrie so zu drehen und zu skalieren, daß die Frontflächen zweier sich ursprünglich überlappender Domänen auf einen Blick miteinander verglichen werden können.

In den drei bisher vorgeschlagenen Modellerschritten wurde immer jeweils die unabhängige Variable X manipuliert, um letztendlich eine eindeutige Abbildung $f_M : X_M \rightarrow Y_M$ zu erhalten. Es könnte aber auch die abhängige Variable Y manipuliert werden, und zwar in dem Sinne, daß in den Überlappzonen alle Werte durch ihr geometrisches Mittel oder durch ihre Differenz ersetzt werden. Damit werden allerdings die Ausdrucksbedingungen nach Mackinlay verletzt, denn es gehen Daten aus dem ursprünglichen Datensatz verloren. Diese Modellerschritte können deshalb nicht zu den Darstellungsfunktionen gezählt werden.

4.4 DAVis-Suite als Methodensatz

Nachdem nun die wesentlichen Datenstrukturen und -modelle zur Visualisierung verteilter Datenfelder erarbeitet wurden, kann nun daran herangegangen werden, die darauf aufbauenden Verfahren und Algorithmen in einem Methodensatz, dem Methodensatz der DAVis-Suite (***D**istributed **A**rray **V**isualization*), zusammenzustellen. Dabei wird bewußt der Bezug auf eine bestimmte Implementationsplattform vermieden, um Anforderung 5 erfüllen zu können. Erst im nächsten Kapitel wird beschrieben, wie die hier vorgestellten Methoden beispielhaft in die modulare Visualisierungsumgebung AVS implementiert wurden.

In Abschnitt 4.3 deuteten sich bereits zwei wesentliche Methoden zur Visualisierung der verteilten Daten an: zum einen das Einbetten der verteilten Datenfelder in eine Feld-Datenstruktur, und zum anderen die Modellierung der realen Variablen. Die Methoden zur Einbettung werden im Modul **DAVisEmbed** zusammengefaßt. Dagegen wurden die Methoden zur Modellierung auf die Module **DAVisModel** und **DAVisDiff** aufgeteilt. DAVisModel modelliert ausschließlich die unabhängige Variable, DAVisDiff nur die abhängige Variable. Weil mit der Manipulation der abhängigen Variablen die Ausdrucksbedingungen verletzt werden können, kann DAVisDiff zur Klasse der Mapper gezählt werden, also zu der Klasse von Modulen, die eine Ansicht von einer Feld-Datenstruktur erzeugen.

DAVisEmbed, DAVisModel und DAVisDiff basieren auf der in Abschnitt 4.3.1 vorgestellten Feld-Datenstruktur. Sie orientieren sich an den Erfordernissen, die sich aus Anforderung 1 ergeben. Zwei weitere Module, und zwar **DAVisGenerate** und **DAVisDisplay**, sollen dem Benutzer ermöglichen, mit der Information über die ursprüngliche Verteilung umzugehen; mit den in diesen Modulen implementierten Methoden sollen einerseits reguläre Feld-Partitionierungen interaktiv erstellt und andererseits schematisch dargestellt werden können. Mit DAVisGenerate und DAVisDisplay soll also die Anforderung 2 erfüllt werden.

4.4.1 DAVisGenerate

Mit dem Modul DAVisGenerate sollen vorgegebene Datenfelder interaktiv im Sinne einer regulären Feld-Partitionierung aufgeteilt werden können. Die visuelle Rückkopplung der Eingaben an den Benutzer wird dabei DAVisDisplay überlassen, das von DAVisGenerate eine Datenstruktur ähnlich der Struktur `gap_type` aus Abschnitt 4.3.4 erhält. Die Benutzereingabe für DAVisGenerate orientiert sich an den Überlappzonen zwischen den Domänen. Damit bleibt dem Benutzer erspart, jede Domäne einzeln durch die Angabe zweier Eckpunkte spezifizieren zu müssen. Einerseits muß nämlich bei einer Eingabe von Punkten im Raum mit einer zweidimensionalen Maussteuerung die erstellte Geometrie häufig gedreht oder skaliert werden, bis der gewünschte Punkt der Geometrie hinreichend genau selektiert werden kann. Andererseits kann die Anzahl der Domänen sehr groß werden, insbesondere dann, wenn eine Gebietszerlegung aus zum Beispiel 1024 Teilgebieten zum Einsatz auf einem massiv-parallelen Rechner zu erstellen ist. Die Orientierung an den Überlappzonen zwischen den Domänen reduziert also nicht nur den Speicherbedarf, wie schon anhand der Datenstrukturen `f_part_2d` und `gap_type` gezeigt wurde, sondern erleichtert zudem die Benutzereingabe. Die Spezifizierung einer Überlappzone durch drei Parameter verdoppelt die Anzahl d der Domänen, die von einem entsprechenden Überlappband umwickelt werden. Demgegenüber müßten, um die gleiche Aufteilung zu erhalten, $3 \cdot d$ Eckpunkte neu spezifiziert werden.

Jede Überlappzone hat eine feste Breite von b' Elementen bzw. w' Knoten; in der schematischen Darstellung können die Überlappzonen durch Bänder dargestellt werden, die um das gesamte Gebiet gewickelt werden. Die Elemente innerhalb des Bandes gehören dann sowohl dem Teilgebiet links als auch dem Teilgebiet rechts des Bandes zu. Ein Band g wird vom Benutzer durch drei Parameter festgelegt:

- der **Orientierung** i'_g des Bandes,
- der **Position** s'_g des Bandes (gemessen an den Knoten des Finiten-Elemente-Gitters), und
- der **Breite** b'_g des Bandes in Elementen.

Die gestrichenen Größen i'_g , s'_g und b'_g entsprechen den Größen i_g , s_g und w_g des Referenzraumes; Abb. 4.18 zeigt die Umsetzung anhand der Überlappzonen `gaps[0]` und `gaps[1]` aus Abschnitt 4.3.4. Ist das Gitter irregulär, so muß das Überlappband nicht notwendigerweise senkrecht zu einer der Raumachsen orientiert sein und überall die gleiche "metrische" Breite besitzen. Entscheidend sind die Parameter i_g , s_g und w_g im Referenzraum; die Parameter i' , s' und b' aus der Benutzereingabe müssen also noch in die Parameter für den Referenzraum umgesetzt werden. Sie ergeben sich aus den Gleichungen

$$\begin{aligned} i_g &= i'_g \quad , \\ w_g &= b'_g + 1 \quad \text{sowie} \\ s_g &= s'_g + \sum_{\gamma=g_i}^g w_\gamma + (g - g_i) \cdot 2 \quad ; \end{aligned}$$

darin bezeichnet g_i die Nummer der ersten Trennung mit der Orientierung in r_i -Richtung, siehe Abb. 4.18.

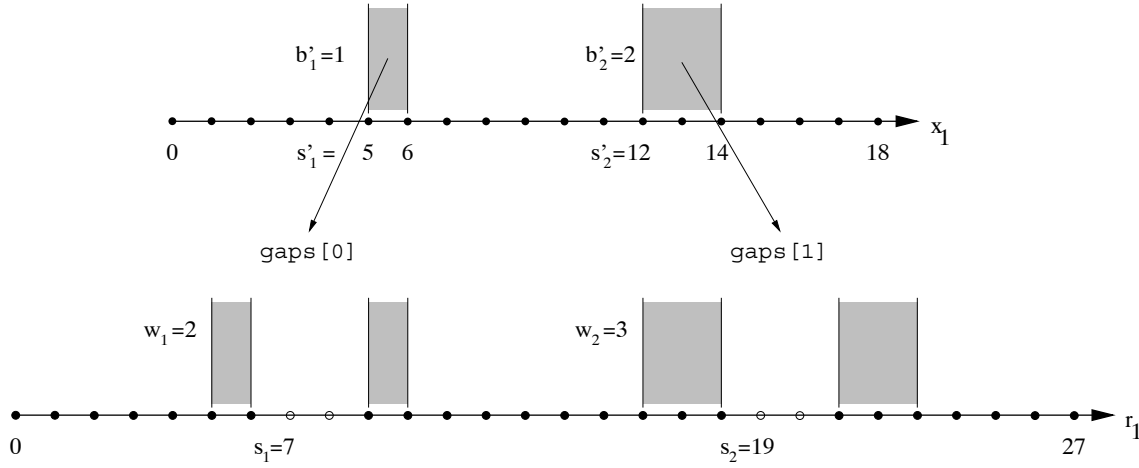


Abbildung 4.18: Überlappzonen in Referenzraum und Darstellungsraum

Die drei Parameter eines jeden Bandes – Orientierung, Position und Breite – sollen in der Implementation für jede Überlappzone auch im Nachhinein noch editierbar sein. Allerdings muß bei jeder Eingabe einer neuen Position oder einer neuen Breite einer Überlappzone überprüft werden, ob sich nicht Überlappzonen gleicher Orientierung gegenseitig überlappen.

Die Zahl der Benutzereingaben ließe sich noch weiter reduzieren, wenn eine Optimierungsvorschrift für die Gebietszerlegung besteht und für DAVisGenerate als (externe) Unterroutine vorliegt. Bei einer Optimierung nach der K-Funktion (siehe Abschnitt 2.4.1) beispielsweise müßten dann nur noch die Zahl der Teilgebiete und die Breite der Überlappzonen zwischen den Teilgebieten festgelegt werden, um ein vorgegebenes Gitter optimieren zu können.

Mit DAVisGenerate sollen die Domänen-Aufteilungen als verständliche Textdateien abgespeichert und gelesen werden können, damit einerseits auf früher erstellte Aufteilungen wieder zurückgegriffen werden kann, andererseits aber auch externe Tools Aufteilungen an DAVisGenerate übermitteln können. Die Text-Dateien sollen wahlweise entweder die Trennungen zwischen den Partitionen beschreiben oder alle Partitionen einzeln mit ihrer Lage und ihren Ausmaßen.

4.4.2 DAVisDisplay

Das Modul DAVisDisplay erfüllt zwei wichtige Aufgaben: zum einen die **graphisch-schematische Darstellung der Domänen-Aufteilung**, die in jede beliebige Ansicht mit hinein integriert werden kann, und zum anderen die Möglichkeit zur

interaktiven Selektion eines oder mehrerer Teilgebiete. Zudem dient DAVISDisplay zur grafischen Ausgabe und Rückkopplung der Benutzereingaben von DAVISGenerate.

Die graphisch-schematische Darstellung der Domänen-Aufteilung umfaßt:

- die Darstellung aller Domänen mit ihrer äußeren Hülle. Gruppen-Domänen sollen grafisch also genau so als eine Einheit dargestellt werden wie die Einzel-Domänen. Die Hülle soll dabei mit verschiedenen Render-Methoden visualisiert werden können, zum Beispiel opak, transparent, oder als Drahtgittermodell. Ebenfalls soll die Möglichkeit bestehen, die Hülle nach einem Skalarwert einzufärben, zum Beispiel nach der Partitionsnummer oder sogar nach einem skalaren Feldwert;
- die Darstellung der Überlappzonen, zum Beispiel als Bänder um die Domänen;
- die Darstellung der Grenzen des Gesamtgebiets,
- die Einblendung von Labeln, also von Textmarkierungen wie zum Beispiel den Partitionsnummern, in die Modellgeometrie.

In der schematischen Darstellung sollen sich ebenfalls die Domänen voneinander entfernen lassen, ihre Überlappzonen um einen kleinen Versatz verücken lassen etc.; DAVISDisplay greift also prinzipiell auf die gleichen Modellialgorithmen zurück wie auch DAVISModel, siehe Abschnitt 4.4.4.

Die Aufgabe, eine äußere Hülle um eine Einzel-Domäne zu finden, ist einfach zu lösen, denn in regulären Gittern besteht die Außenfläche einer Domäne aus sechs rechteckigen Seitenflächen, die sich zu einem Quader zusammensetzen. In irregulären Gittern können diese Seitenflächen deformiert sein; statt der sechs Rechtecke setzt sich dann die Hülle aus sechs Netzen zusammen. Netze sind nichts weiter als irreguläre Quadrilateral-Gitter, wie sie in Abschnitt 4.3.1 beschrieben wurden; sie bestehen aus $M_1 \times M_2$ Knotenpunkten, mit denen die deformierte Seitenfläche nachmodelliert werden kann. Die Hülle einer Einzel-Domäne setzt sich also entweder aus sechs deformierten oder sechs rechteckigen, ebenen Seitenflächen zusammen. Die Koordinaten der Eck- oder Netzpunkte können einfach aus der Feld-Datenstruktur ausgelesen werden.

Die Aufgabe, eine Hülle um eine Gruppen-Domäne zu finden, kann jedoch nicht so trivial gelöst werden, denn Gruppen-Domänen müssen nicht notwendigerweise quaderförmig sein, wie das Beispiel in Abb. 4.19 zeigt. Die gezeigte Gruppen-Domäne besteht aus sieben einzelnen Domänen. Die gesamte Domänen-Aufteilung besteht aus $3 \times 2 \times 2 = 12$ Domänen.

Ein Algorithmus zur Bestimmung der Hülle um eine solche Gruppen-Domäne wird im Anhang A vorgestellt.

Die andere Teilaufgabe des Moduls DAVISDisplay ist die interaktive Selektion der Teilgebiete

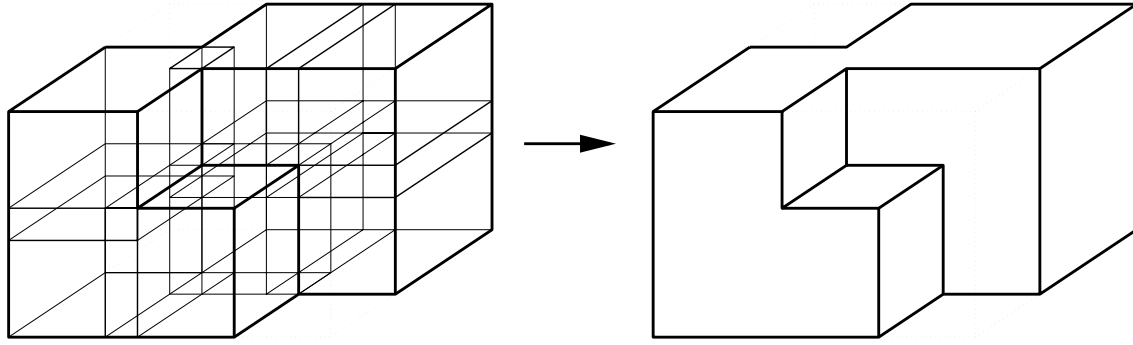


Abbildung 4.19: Hülle um eine Gruppendomäne

- zur **Gruppierung der Domänen**, das heißt, dem Zusammenfassen mehrerer Einzel- oder Gruppen-Domänen zu einer Gruppen-Domäne;
- zum **Auflösen von Gruppen-Domänen**, das heißt, dem Auftrennen von Gruppen-Domänen in einzelne Domänen;
- zum **Einlesen der Daten** aus den selektierten Teilgebieten. Dazu leitet DAVisDisplay die Information darüber, welche Domänen selektiert wurden, an DAVisEmbed als eine Bitfolge weiter.

Die Selektion soll dabei interaktiv am Bildschirm erfolgen. Domänen werden selektiert, indem sie einfach in der schematischen Darstellung angeklickt werden. Damit nicht jede Domäne bei der Selektion mehrerer benachbarter Teilgebiete einzeln angeklickt werden muß, sollte auch ein *sweeping*, also ein gleitendes Selektieren bei gedrückter Maustaste, möglich sein. Weitere Selektionsmethoden für mehrere Domänen, wie zum Beispiel das Spezifizieren zweier "Eck"-Domänen zur Selektion eines ganzen Blocks von Domänen, oder die Selektion aller Domänen, sind ebenfalls denkbar.

Die Selektionsmechanismen werden bei der Realisierung in Abschnitt 5.2.3 noch genauer erläutert.

4.4.3 DAVisEmbed

Mit dem Modul DAVisEmbed werden die Daten X_D und Y_D der verteilten Datenmenge in die Feld-Datenstruktur nach den Vorschriften aus den Gleichungen (4.35), (4.38) oder (4.43) eingebettet. Von DAVisDisplay erhält das Modul als Bitfolge die Information, welche Domänen der Benutzer ausgewählt hat und welche Datenfelder $\underline{X}_{D,p}$ und $\underline{Y}_{D,p}$ in die Felder $\underline{X}_C$ und $\underline{Y}_C$ geladen werden sollen. Die Größe der Felder wird nach Gleichung (4.45) berechnet. Zwischen den Feld-Partitionen $\underline{X}_{C,p}$ bzw. $\underline{Y}_{C,p}$ sowie zwischen den Teilgebieten und den Überlappzonen werden, wie in

Abschnitt 4.3.4 erklärt, Leerstellen eingefügt; damit lassen sich die Domänen mit DAVISModel in der Darstellung voneinander trennen.

Zum Einlesen der Daten aus der Datenmenge kann eine externe Lesemethode benutzt werden, die auf das spezifische Datenformat der Datenmenge zugeschnitten ist. An die Lesemethode wird nur die Bedingung gestellt, daß sie nach Angabe von Partitionsnummer, einem Zeitschritt und dem Stammnamen der Partitionen die entsprechende Daten-Partition DAVISEmbed als Datenfeld zur Verfügung stellt. Es können also verschiedenste Datenformate gelesen werden, indem einfach nur die Lesemethode gegen eine andere ausgetauscht wird. Damit deutet sich eine Zweiteilung des Moduls DAVISEmbed an: der erste Teil (ein Server) steuert das Lesemodul an, der zweite Teil (ein Client) nimmt das vom Lesemodul eingelesene Datenfeld entgegen und bettet es in die Feld-Datenstruktur $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$ ein.

4.4.4 DAVISModel und DAVISDiff

DAVISModel und DAVISDiff formen die realen Variablen X_C und Y_C aus DAVISEmbed zu Modellvariablen X_M und Y_M um. Ziel der Modellierung ist in beiden Fällen, eine eindeutige Abbildung $f_M : X_M \rightarrow Y_M$ zu erhalten (siehe Abschnitt 4.2). Dies läßt sich zum einen dadurch erreichen, daß alleine die unabhängige Variable modelliert wird; zum anderen aber auch dadurch, daß nur die abhängige Variable modelliert wird.

DAVISModel enthält ausschließlich Methoden zur Modellierung der unabhängigen Variable; es bearbeitet nur die Daten im Feld $\underline{X}_C$. Die entsprechenden Modellabbildungen wurden bereits in Abschnitt 4.3.5 vorgestellt. Mit der Vergabe verschiedener Orte für die Werte in den Überlappzonen läßt sich die Abbildung f_M zu einer eindeutigen Funktion modellieren.

DAVISDiff soll Methoden zur Bildung von Differenzmengen und zur Mittelung der Werte in den Überlappzonen beinhalten. Im Unterschied zu DAVISModel modelliert DAVISDiff also ausschließlich die abhängige Variable im Feld $\underline{Y}_C$.

In einer Überlappzone, in der sich k Domänen gegenseitig überlappen, gibt es $k \cdot (k - 1)$ Möglichkeiten, eine Differenzmenge der Werte zweier Domänen zu bilden – aus den k Domänen wird zuerst die Domäne ausgewählt, deren Werte in der Überlappzone die Minuenden bei der Differenzbildung darstellen, und danach aus den $k - 1$ Domänen die Domäne, deren Werte in der Überlappzone die Subtrahenden sind. Bei einer dreidimensionalen Darstellung mit einer regulären Feld-Partitionierung können Überlappzonen auftreten, in denen sich bis zu acht Domänen gegenseitig überlappen. Für diese Überlappzonen können folglich 56 (!) verschiedene Differenzmengen gebildet werden, die keinesfalls gleichzeitig dargestellt werden könnten. Jedoch wird man im allgemeinen nur solche Domänen vergleichen wollen, die direkte Nachbarn in einer der drei Raumrichtungen sind. Dies beschränkt die Anzahl der Differenzmengen in den soeben betrachteten "achtfachen" Überlappzonen auf 24. Weitere Einschränkungen müssen durch Benutzereingaben gemacht

werden. Wenn für alle Überlappzonen bestimmt ist, welche Differenzmengen zu berechnen sind, werden alle Werte in den Überlappzonen durch die jeweiligen Differenzen ersetzt. Die eindeutigen Werte außerhalb der Überlappzonen werden auf den undefinierten Wert $\perp$ gesetzt, um Fehlinterpretationen zu vermeiden.

Für die Mittelung ist keine Spezifikation notwendig, welche Domänen an der Bildung beteiligt sind; in einer Überlappzone werden alle Werte für einen Ort zur Bildung eines Mittelwertes herangezogen.

Durch den Ersatz mehrerer Werte durch einen eindeutigen Wert an den Orten in den Überlappzonen wird die Abbildung f_M zu einer eindeutigen Funktion.

4.5 Benutzeroberfläche

Wie in Abschnitt 4.2 schon hervorgehoben wurde, eignet sich der modellbasierte Ansatz zur Gestaltung einer Benutzeroberfläche. Der Benutzer legt im ersten Schritt fest, welche Datenmenge visualisiert werden soll. Danach werden Parameter zur Modellierung vorgegeben, und das Visualisierungssystem modelliert nach diesen Parametern ein Modellfeld. Schließlich wählt der Benutzer aus einer Vielzahl von möglichen Ansichten die für das erzeugte Modellfeld passende Ansicht aus. Es ergeben sich also drei Stufen im Visualisierungsprozeß: zuerst die Selektion der Daten, dann die Modellierung, und schließlich die Erzeugung einer Ansicht, wie auch schon in Abb. 4.1 gezeigt wurde.

In Hinblick auf Anforderung 3 wäre es jedoch wünschenswert, mehrere Ansichten zu einer Ansicht zusammenfassen und vergleichen zu können. Deshalb wurde zur Konzeption einer Benutzeroberfläche, mit der alle Module der DAVis-Suite in die bereits vorhandenen Module des Visualisierungssystems eingebunden werden sollen, ein vierstufiger Visualisierungsprozeß angesetzt, der aus den Schritten Dateneingabe – Modellierung – Ansicht erzeugen – Ansichten zusammenfassen besteht (*data – model – view – viewer*, siehe Abb. 4.20). Innerhalb der DAVis-Benutzeroberfläche gibt es damit vier Arten von DAVis-Modulen:

1. die **Daten-Module**, die die Standard-Datenformate der MVE oder die verteilten Daten mit den Modulen der DAVis-Suite lesen und in eine Feld-Datenstruktur ablegen können;
2. die **Module zur Modellierung**, mit denen die Daten einer Feld-Datenstruktur so manipuliert werden können, daß aus ihnen eine Ansicht erzeugt werden kann. Diese Module ließen sich demnach in solche Module unterteilen, die nur die unabhängige Variable modellieren, solche, die nur die abhängige Variable modellieren, und in solche, die beide Variablen gleichzeitig modellieren
3. die **View- oder Mapper-Module**, mit denen eine Ansicht einer Feld-Datenstruktur erzeugt werden kann, wie zum Beispiel die Visualisierung eines

Vektorfeldes einer Feld-Datenstruktur mit Hilfe von Stromlinien. Die Mapper-Module lassen sich nach Glyph-Mappern und kontinuierlichen Mappern ordnen;

4. die **Viewer-Module**, mit denen mehrere Ansichten nach Bedarf zusammengestellt werden können. Viewer-Module sollen untereinander koppelbar sein, das heißt auf Wunsch sollen Kamera-Einstellung, Beleuchtung, etc. von einer Ansicht auf eine andere Ansicht übertragen werden können.

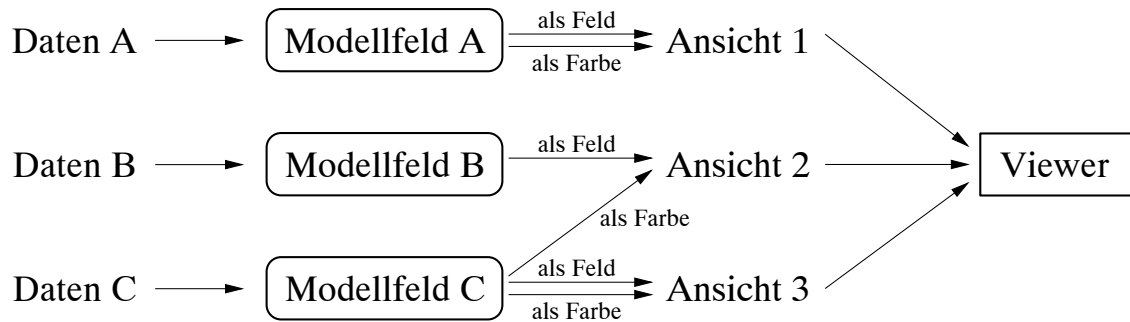


Abbildung 4.20: Vierstufiger Visualisierungsprozeß für die Benutzeroberfläche

Zwischen den "Modulen der DAVis-Suite" und den "DAVis-Modulen" besteht ein kleiner begrifflicher Unterschied; die "Module der DAVis-Suite" sind die Module DAVisGenerate, DAVisDisplay, DAVisEmbed und DAVisModel mit DAVisDiff. Unter den "DAVis-Modulen" werden die Module der Benutzeroberfläche verstanden; sie können aus mehreren Modulen der DAVis-Suite zusammengesetzt sein. Beispielsweise besteht das DAVis-Modul **Distributed Data**, mit dem verteilte Daten gelesen werden können, aus den Modulen DAVisGenerate, DAVisDisplay, und DAVisEmbed aus der DAVis-Suite. Mit DAVisGenerate wird dem System mitgeteilt, wie das Gesamtfeld aufgeteilt ist, mit DAVisDisplay lassen sich die gewünschten Domänen zur Visualisierung auswählen, und mit DAVisEmbed schließlich können die selektierten Datenfelder in eine Feld-Datenstruktur eingebettet werden. Die Granularität des DAVis-Modulsatzes ist im Vergleich zum ursprünglichen Modulsatz der MVE grober; DAVis-Module sind in der Regel aus mehreren MVE-Modulen oder Modulen der DAVis-Suite zusammengesetzt.

Die Anzahl der Ein- und Ausgänge der DAVis-Module wurde reduziert und ihr Typ vereinheitlicht. Dazu sei nochmal Abb. 4.20 betrachtet: Daten-Module besitzen einen Felddausgang. Module zur Modellierung haben einen Feldein- und einen Felddausgang; View-Module haben zwei Feldeingänge und einen Geometrie-Ausgang. Viewer-Module haben einen Geometrie-Eingang, über den mehrere Geometrien eingelesen werden können. Der Inhalt jedes Modulausgangs kann gleichzeitig zu mehreren Moduleingängen der nächsten Stufe weitergeleitet werden. Diese mehrfache Verwendung von Modulausgängen erlaubt die Erzeugung verschiedener Views des gleichen Modellfelds sowie die Modellierung ein- und derselben Datenmenge mit verschiedenen Modellerschritten.

Alle View-Module können mindestens zwei Skalarfelder in eine farbige Geometrie umsetzen. Als Beispiel sei das "Hedgehog"-Modul genannt, mit dem Vektorfelder abgebildet werden können. An beliebigen Stellen des Feldgebietes, die der Benutzer vorgeben kann, werden Richtung und Größe des Vektorfeldes durch entsprechend skalierte Vektorpfeile dargestellt. Die Vektorpfeile können zusätzlich noch eingefärbt werden. Mit vier Skalaren lassen sich also die drei Komponenten und die Farbe des Vektorpfeils festlegen. Das "Hedgehog"-Modul kann damit gleichzeitig *eine* Ansicht von vier Skalarfeldern ($n' = 4$), oder von einem zweidimensionalen Vektorfeld und zwei Skalarfeldern ($n'_1 = 2, n'_2 = 2$), oder von einem dreidimensionalen Vektorfeld und einem Skalarfeld ($n'_1 = 3, n'_2 = 1$) erstellen. Weitere Beispiele für DAVis-View-Module sind das Modul Isosurface, mit dem eine Isofläche in einem skalaren Datensatz erzeugt und mit dem Skalarfeld am Farbeingang eingefärbt werden kann, oder der Orthogonalschnitt, bei dem der Feldeingang zur Erzeugung eines Höhenprofils auf dem Profilschnitt verwendet werden kann.

Die Benutzeroberfläche soll erweiterungsfähig sein, das heißt selbst zusammengebaute DAVis-Module sollen zusätzlich von der Benutzeroberfläche aus ansprechbar sein und in den jeweiligen Visualisierungsprozeß mit eingebunden werden können.

Kapitel 5

Distributed Array Visualization: Realisierung

In diesem Kapitel wird eine beispielhafte Implementierung des DAVis-Konzeptes mit AVS 5.0 [AVSU92] vorgestellt. Dabei wurde die Implementierung zunächst auf diejenigen Datenstrukturen und Methoden reduziert, die für die Visualisierung der Daten von TRACE am notwendigsten erschienen. Weil die jetzige Version von TRACE noch nicht auf verteilten Systemen läuft, wurde der Algorithmus zum Erstellen einer Hülle um Gruppen-Domänen noch nicht implementiert. Auch wurden zur Implementation des Beispiels an AVS und TRACE angepaßte Datenstrukturen verwendet.

5.1 Datenstrukturen

Im Ansatz wurden drei Datenstrukturen vorgestellt, für die nun die Aufgabe lautet, sie auf die von AVS angebotenen Datentypen abzubilden. Die erste und wichtigste Datenstruktur ist die Feld-Datenstruktur mit den zwei Strukturkomponenten X und Y; die beiden anderen Datenstrukturen sind die Liste der Trennungen zwischen den Feld-Partitionen, und die Baumstruktur mit der Hierarchie der Gruppen- und der Einzel-Domänen. Im Anschluß werden die Datenstrukturen vorgestellt, die abweichend von der Konzeption in der Implementation verwendet wurden.

5.1.1 Die Feld-Datenstruktur

Zur Umsetzung der Feld-Datenstruktur bieten sich in AVS zwei Datentypen an: zum einen der Datentyp `AVS_field` und zum anderen der Datentyp `UCD` (*unstructured cell data*). Ein AVS-Feld (eine Datenstruktur vom Typ `AVS_field`) besitzt zwei Datenfelder namens `data` und `points`. In `data` werden die Feldwerte eingetragen, in `points` hingegen die Koordinaten der Orte, für die diese Feldwerte gelten. AVS-Felder können in drei Ausprägungen auftreten: als *irregular*-, *regular*- und als *uniform*-Feld. Ein *irregular*-Feld ist ein explizit beschriebenes Feld, das heißt,

alle Feldorte sind einzeln mit allen Koordinaten vorgegeben. AVS-Felder der Ausprägung **regular** oder **uniform** sind implizit beschriebene Felder. Für sie werden im Datenfeld **data** n Folgen von Koordinaten abgelegt; daraus folgen die Feldorte. Bei **uniform**-Feldern bestehen diese Folgen aus zwei Elementen, dem Anfangs- und dem Endwert der Koordinate in jeder Richtung; dazwischen werden die Koordinaten in gleichmäßigen Abständen aufgeteilt. **Uniform**-Felder sind also streng reguläre Feldbeschreibungen.

Der Datentyp **UCD** hingegen besteht aus drei hierarchisch geordneten Komponenten: der Komponente für die UCD-Struktur insgesamt, der Komponente für die Elemente der UCD-Struktur, und einer Komponente für die Knoten in der UCD-Struktur. Eine UCD-Struktur besteht aus mehreren Elementen, und jedes Element besteht wiederum aus einer festen Anzahl von Knoten. Für jede der drei Komponenten können Feldwerte angegeben werden, aber nur für Knoten gibt es das Attribut "Ort" in einem dreidimensionalen Raum (Struktur-Komponenten **x**, **y** und **z**). Jeweils acht Knoten können so zu einem Element verbunden werden, daß ein irreguläres Hexaeder-Gitter entsteht, wie es auf Seite 42 beschrieben wurde. Dazu gibt es für jedes Element einer UCD-Struktur eine Knoten-Verbindungsliste, in der festgelegt ist, welche acht Knoten zu diesem Element gehören.

Zunächst erscheint der Datentyp **UCD** geeigneter für die spezielle Anwendung "TRACE / ParTrace" zu sein, denn zum einen können sowohl knoten- als auch elementorientierte Datenfelder problemlos in einer UCD-Datenstruktur zusammengelegt werden; zum anderen gibt es die Möglichkeit, den Elementen bestimmte Materialeigenschaften zuzuordnen. Dies läßt sich beispielsweise beim computer-unterstützten Design von Maschinenteilen ausnutzen, wie Abb. 5.1 zeigt. Das Maschinenteil, ein Stoßdämpfer, setzt sich aus verschiedenen Teilen zusammen, wie die Explosionszeichnung rechts zeigt. Jedem Element (das Modell besteht aus insgesamt 3901 Elementen) wurde die Materialeigenschaft "Zugehörigkeit zum Teil T " zugeordnet. Ähnlich ließe sich auch die Zugehörigkeit zu einem der P verteilten Datenfelder in einer UCD-Datenstruktur repräsentieren, eben als eine Art "Materialeigenschaft".

Zur Realisierung fiel jedoch die Wahl nicht auf den Datentyp **UCD**, sondern auf den Datentyp **AVS_field**. Der Grund liegt in der Anforderung 5: der Datentyp **UCD** ist eine spezifische AVS-Struktur und seine Fähigkeiten stehen nicht in jedem Visualisierungssystem zur Verfügung, wie zum Beispiel die Fähigkeit, Materialeigenschaften für jedes Element festlegen zu können. Der Datentyp **AVS_field** dagegen ist recht einfach aufgebaut: im Wesentlichen besteht er aus den zwei Datenfeldern **data** und **points**. Eine dazu äquivalente Datenstruktur läßt sich in jedem Visualisierungssystem finden. Zudem belegt dieser Datentyp weniger Speicherplatz, da Knoten- und Elementlisten entfallen.

Was in Abschnitt 4.3.1 als Referenz R_X und R_Y bezeichnet wurde, stellt sich für den Datentyp **AVS_field** als die Indizes der Datenfelder **points** und **data** dar. Die Referenzabbildung r ist im Fall des **irregular**-Feldes die Identität; im Fall der **regular**- oder **uniform**-Felder ist dies eine Abbildung nach den Gleichungen (4.16) und (4.17).

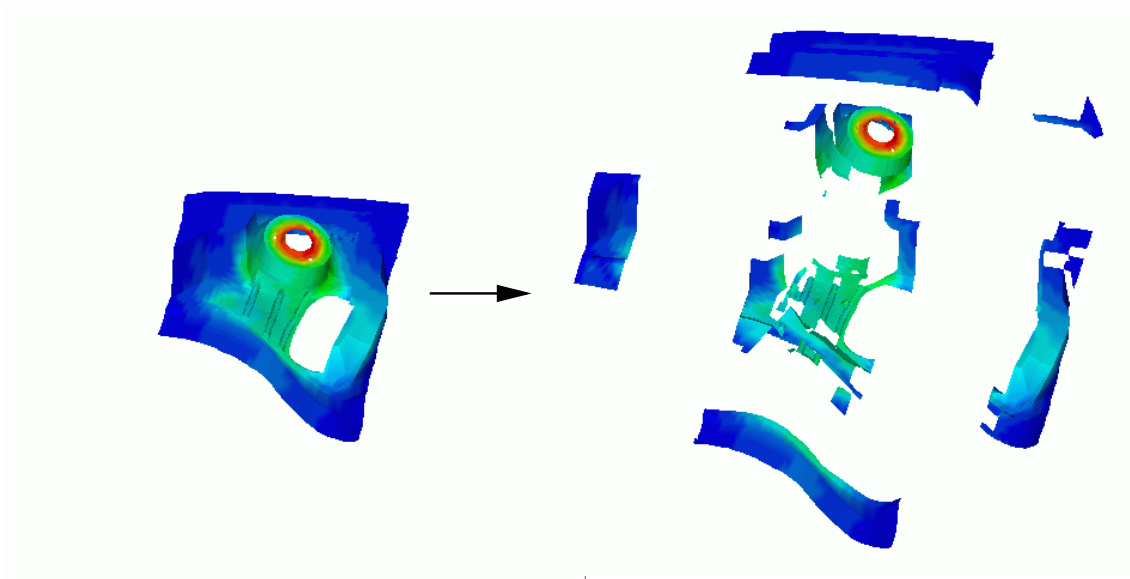


Abbildung 5.1: Maschinenteil, aus verschiedenen Teilen zusammengesetzt

Daten in den AVS-Feldern können mit verschiedenen Genauigkeiten angegeben werden, wie zum Beispiel als `float`, `double`, `integer`, etc. Vorläufig reicht für die Anwendungen TRACE und ParTrace die Genauigkeit `float` völlig aus. Der Datentyp `double` mit 8 Byte Fließkommadarstellung wäre allenfalls für die Darstellung der Permeabilität interessant, denn $\mathbf{K}$ erstreckt sich über mehrere Größenordnungen. Aber auch die Permeabilität $\mathbf{K}$ kann mit einer `float`-Darstellung hinreichend genau dargestellt werden, wenn sie – wie allgemein in der Hydrogeologie üblich – logarithmiert dargestellt wird.

5.1.2 Feld-Partitionierung

Weil der Typ der Feld-Partitionierung für die Implementierung auf reguläre Feld-Partitionierungen beschränkt wurde, könnte die Information über die ursprüngliche Verteilung der Daten in einer Datenstruktur repräsentiert werden, die die Überlappzonen zwischen den Domänen beschreibt bzw. sich an den Trennungen zwischen den Feld-Partitionen orientiert. Aus einer solchen Datenstruktur ließen sich ohne weiteres die Lage und Ausmaße der Feld-Partitionen ableiten; zudem würde sie weniger Speicherplatz belegen als eine entsprechende Datenstruktur, in der für jede Partition Informationen über Lage und Ausmaß abgelegt sind, wie bereits der Vergleich in Abschnitt 4.3.4 zeigte. Mit den in AVS vorhandenen Standard-Datenstrukturen lassen sich die drei Parameter Orientierung, Position und Breite für jedes Überlappband jedoch nur schlecht repräsentieren. Deshalb muß auf eine benutzerdefinierte Datenstruktur ausgewichen werden. Benutzerdefinierte Datenstrukturen können wie üblich

che C-Strukturen (`struct`) definiert werden, allerdings gibt es einige Einschränkungen in Bezug auf die erlaubten Datentypen innerhalb der Struktur. Deshalb mußte die Datenstruktur wie folgt angelegt werden:

```
typedef struct _ud_gaps_type {
    char orientation[64];
    int position[64];
    char width[64];
} ud_gaps_type;
```

Damit wird ein Datentyp `ud_cut_type` definiert, der aus den drei Datenfeldern `orientation`, `position` und `width` besteht. Der umgekehrte Weg, also ein Datenfeld, das aus Strukturen besteht, wäre in AVS als benutzerdefinierte Datenstruktur nicht zugelassen; Datenfelder können nicht zwischen den Modulen ausgetauscht werden. Auch darf innerhalb der benutzerdefinierten Datenstruktur kein Makro aus einer Preprozessor-Anweisung verwendet werden. Deshalb mußte die Datenfeldgröße von 64 Elementen explizit für jedes Datenfeld angegeben werden. Die Anzahl der Überlappbänder durch das Gebiet wird damit auf maximal 64 beschränkt, und folglich auch die Anzahl der Teilgebiete in einem dreidimensionalen Gesamtgebiet auf maximal $(22 + 1) \cdot (21 + 1) \cdot (21 + 1) = 11132$, wenn die 64 Überlappbänder gleichmäßig auf die drei Raumrichtungen verteilt sind, bzw. auf maximal $64 + 1 = 65$, wenn alle Überlappzonen die gleiche Orientierung besitzen¹. Für die meisten Feld-Partitionierungen dürfte jedoch eine solche Anzahl von Trennungen ausreichen. Werden mehr Überlappzonen bei einer Gebietszerlegung benötigt, so müßte der Wert 64 in der Datenstruktur `ud_gaps_type` im Quelltext entsprechend geändert und alle Module der DAVis-Suite neu kompiliert werden.

In Visualisierungssystemen, in denen es keine benutzerdefinierten Datentypen gibt, muß eine andere Möglichkeit gefunden werden, diese Information abzulegen und zwischen den Methoden austauschen zu können. Dazu könnte beispielsweise ein Vektorfeld mit den Dimensionen $n = 1$ und $n' = 3$ benutzt werden, mit

$$\begin{aligned} \underline{Y}[R_Y] &= (y_1(R_Y), y_2(R_Y), y_3(R_Y)) \\ &= (y_1(r_Y), y_2(r_Y), y_3(r_Y)) \quad , \end{aligned} \quad (5.1)$$

dabei kann $y_1(r_Y)$ mit der Orientierung, $y_2(r_Y)$ mit der Position, und $y_3(r_Y)$ mit der Breite eines Trennbandes mit der Nummer r belegt werden.

5.1.3 Domänen-Hierarchie

Einzelne Domänen sollen zu Gruppen-Domänen zusammengestellt werden können. Damit lassen sich einige Arten von Gebietszerlegungen mit den in Abschnitt

¹Allgemein: mit c Trennbändern lassen sich dreidimensionale Gebiete in maximal $\lfloor c/3 + a \rfloor \cdot \lfloor c/3 + b \rfloor \cdot \lfloor c/3 + 1 \rfloor$ Teilgebiete zerlegen; dabei ist $a = 2$ und $b = 2$, falls $c \bmod 3 = 2$, oder $a = 2$ und $b = 1$, falls $c \bmod 3 = 1$, oder $a = 1$ und $b = 1$, falls $c \bmod 3 = 0$. Die maximale Anzahl von Teilgebieten ergibt sich demnach für den Fall, daß in jeder Dimension etwa gleich viele Überlappbänder vorliegen.

4.4 vorgestellten Methoden bearbeiten, die nicht direkt auf eine reguläre Feld-Partitionierung zurückzuführen sind. Ein Beispiel für eine solche Aufteilung in Domänen zeigt Abb. 5.2a. Hier haben zwar alle Überlappbänder eine bestimmte Orientierung, Position und Breite, aber die Bänder umwickeln nur Teile des Gesamtgebietes, sie sind nicht vollständig. Eine solche Gebietszerlegung läßt sich nun dadurch intern als reguläre Feld-Partitionierung repräsentieren, indem einige Teilgebiete als Gruppen-Domänen aufgefaßt werden. Die Gruppen-Domänen bestehen aus mehreren Einzel-Domänen; alle Einzel-Domänen können jedoch so in der Feld-Datenstruktur abgelegt werden, daß sich eine reguläre Feld-Partitionierung ergibt, wie Abb. 5.2b für das Beispiel in Abb. 5.2a zeigt.

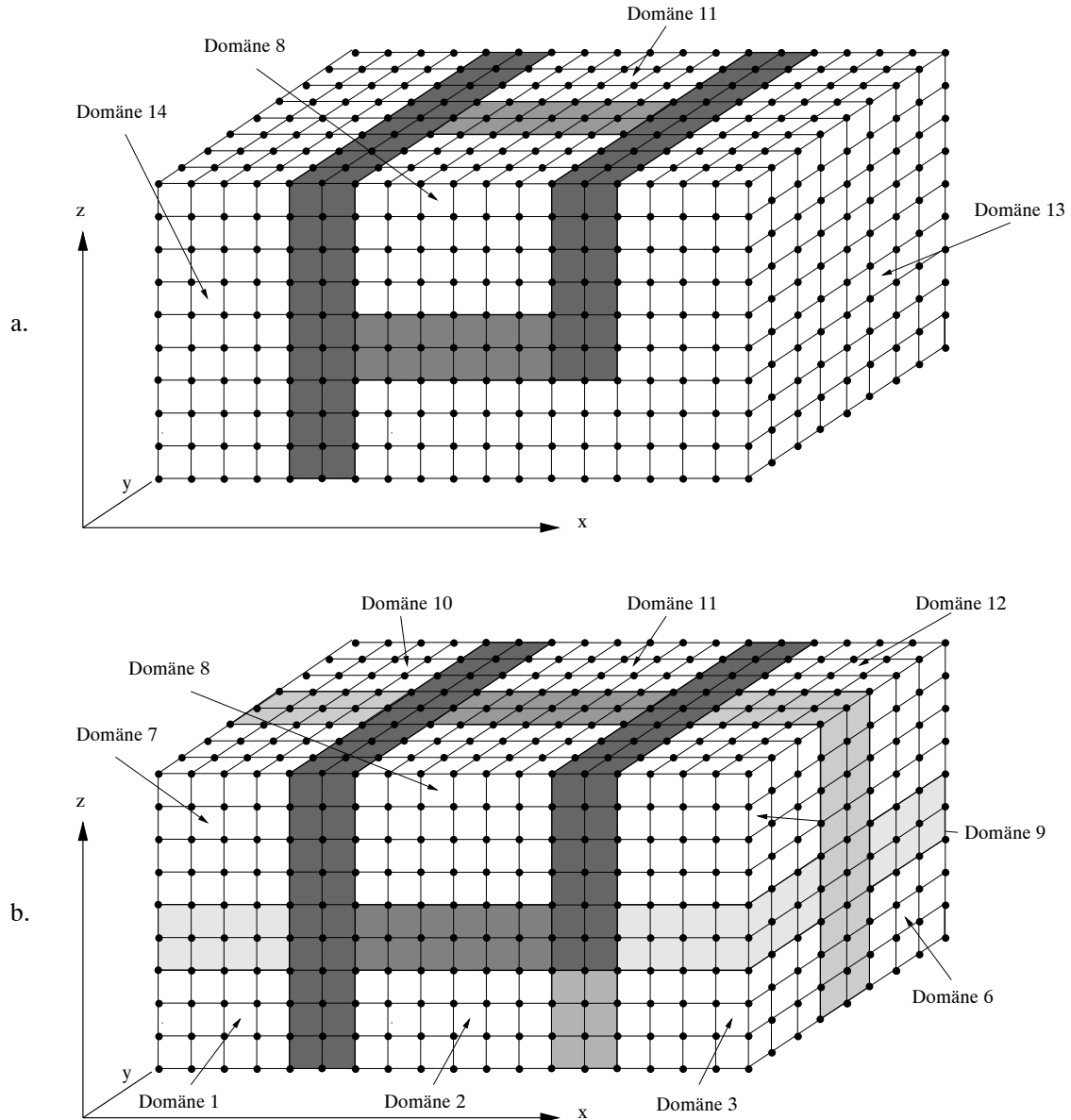


Abbildung 5.2: Nicht reguläre (a) und reguläre (b) Domänen-Aufteilung

Wenn Gruppen-Domänen aus Einzel-Domänen bestehen, dann ergibt sich daraus eine mindestens zweistufige Hierarchie unter den Domänen. Es bietet sich an, eine Baumstruktur für alle Domänen anzulegen, wie sie Abb. 5.3 für die Domänenaufteilung aus Abb. 5.2a zeigt.

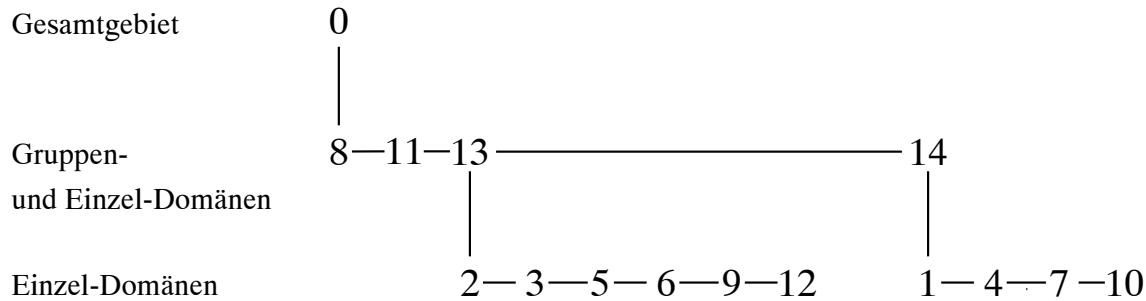


Abbildung 5.3: Baumstruktur für die Domänen-Hierarchie

In diesem Baum gibt es eine dreistufige Hierarchie: das Gesamtgebiet, die Domäne mit der Nummer Null, belegt die oberste Hierarchiestufe, zu ihr gehören alle Gruppen- und Einzel-Domänen. In die darunterliegende Hierarchiestufe werden alle Gruppen-Domänen eingeordnet, sowie alle Einzel-Domänen, die zu keiner Gruppen-Domäne gehören. Die unterste Hierarchiestufe ist die Stufe derjenigen Einzel-Domänen, die einer Gruppen-Domäne angehören. Weil die Anzahl der Einzel-Domänen in einer Gruppen-Domäne variabel ist, müssen die Baumknoten der Gruppen-Domänen auf Listen von Einzel-Domänen verweisen, siehe Abb. 5.3.

Für die Implementation wird nun im folgenden eine dreidimensionale Feld-Partitionierung angenommen, das heißt, daß $n = 3$ und folglich $P = P_1 \cdot P_2 \cdot P_3$ für die reguläre Feld-Partitionierung der Einzel-Domänen gilt. Jede Feld-Partition besitzt also höchstens $2 \cdot n = 6$ Nachbarn. Weil insbesondere das Modul DAVisDisplay beim Erstellen einer Hülle um Gruppen-Domänen häufig auf die Information zurückgreifen muß, welche Feld-Partitionen zu einer gegebenen Partition Nachbarn sind, ist es ratsam, diese Information mit in die Baumstruktur einzubeziehen. Damit ergibt sich die folgende elementare Datenstruktur in C-Notation für alle Domänen:

```

#define LEFT 0
#define RIGHT 3
#define FRONT 1
#define BACK 4
#define BOTTOM 2
#define TOP 5

typedef struct _ud_domains {
    int id;          /* ID-Nummer fuer jede Domaene */
    ud_domains
        *parent_domain,      /* uebergeordnete Domaene */

```

```

    *child_domain,          /* untergeordnete Domaene */
    *sibling_domain,        /* nebengeordnete Domaene, das heisst, eine
                             Domaene, die ebenfalls parent_domain
                             untergeordnet ist */
    *neighbour_domain[6]; /* die sechs Nachbarn im Referenzraum */
} ud_domains;

```

In dieser Form kann die Datenstruktur jedoch nicht als benutzerdefinierte Datenstruktur in AVS implementiert werden, denn als Strukturkomponenten sind Zeiger nicht erlaubt. Deshalb werden fünf Datenfelder innerhalb der Datenstruktur eingesetzt:

```

typedef struct _domains_type {
    int id[1024];
    int par_dom[1024];
    int chd_dom[1024];
    int sib_dom[1024];
    int ngb_dom[6][1024];
} ud_domains;

```

Damit gibt es auch eine obere Grenze für die Anzahl der Domänen im Gesamtgebiet; hier wurde sie beispielhaft auf 1024 festgelegt. Wie schon bei der Datenstruktur `ud_gaps_type` erwähnt wurde, lässt sich in der Typdefinition einer benutzerdefinierten Datenstruktur kein Makro einer Pre-Prozessor-Anweisung einsetzen.

5.1.4 Weitere Datenstrukturen

Bei der ersten Implementierung des DAVIS-Konzeptes wurden zunächst nur solche Gebietszerlegungen betrachtet, wie sie auch in der vorliegenden Version von TRACE benutzt werden. Dies bedeutet, daß alle Domänen quaderförmig sind, die Überlappzonen überall im Gesamtgebiet gleich breit sind, und das Gitter der finiten Elemente streng regulär ist. Daher konnte die Aufteilung des Gesamtgebietes in Domänen durch ein eindimensionales AVS-Feld dreidimensionaler Vektoren dargestellt werden. Jeder Vektor dieses AVS-Feldes beschreibt die Raumdiagonale einer Domäne mit seinem Anfangs- und seinem Endpunkt; damit kann die Information über Lage und Abmessungen aller Domänen zwischen den Modulen übermittelt werden. Weil das Gitter der finiten Elemente in TRACE streng regulär ist, kann von den Koordinaten auf die Referenzen in der Feld-Datenstruktur geschlossen werden, wenn Δ_i für jede Raumrichtung bekannt ist.

Eine weitere Datenstruktur wurde eingerichtet, die unter anderem alle Parameter enthält, die zur Optimierung der Gebietszerlegung benötigt werden. Sie lautet:

```

struct partinfostruct {
    int np;                /* Gewuenschte Anzahl Domaenen im */

```

```

/* Gesamtgebiet (= Anzahl Teilgebiete in */
/* TRACE */
int p[3];      /* Anzahl der Partitionen in den drei
                Raumrichtungen, aus der K-Funktion
                berechnet */
int overlap;   /* Breite der Ueberlappzone in Elementen */
float d[3];    /* Delta i, die Abstaende zwischen zwei
                Knoten im streng regulaeren Gitter von
                TRACE */
char selection[512]; /* Eine Bitfolge fuer 512 x 8 = 4096 Do-
                    maenen: hier setzt DAVISDisplay die-
                    jenigen Bits auf 1, die zu den Domaenen
                    gehoeren, die der Benutzer ausgewaehlt
                    hat. */
}

```

Die Datenstruktur `partinfostruct` enthält also auch Informationen darüber, welche Domänen der Benutzer in `DAVISDisplay` ausgewählt hat und wie groß die Abstände zwischen den Knoten im streng regulären Gitter der finiten Elemente sind.

5.2 Die Module der DAVIS-Suite

Wie schon in Abschnitt 4.4 erläutert wurde, gibt es vier wesentliche Methoden, die für die Visualisierung verteilter Daten benötigt werden: die Einbettung (`DAVISEmbed`), die Modellierung (`DAVISModel` und `DAVISDiff`), die schematische Darstellung der Domänen (`DAVISDisplay`), und die Generierung einer Domänenaufteilung (`DAVISGenerate`). In der Implementierung wurde zunächst noch auf das Modul `DAVISDiff` verzichtet. Statt des Moduls `DAVISGenerate` wurde ein Modul namens `DAVISRead` implementiert, das eine Datei mit einer Beschreibung der Domänen-Aufteilung nach der Art, wie sie von `TRACE` benutzt wird, lesen kann.

Im folgenden wird die Implementation dieser vier Module beschrieben. Zuvor wird jedoch noch kurz erläutert werden müssen, wie AVS-Module aufgebaut sind, und welche Möglichkeiten bei der Implementation der Algorithmen bestehen.

5.2.1 AVS-Module

Die modulare Visualisierungsumgebung AVS arbeitet, wie in Abschnitt 3.2.1 bereits erwähnt wurde, nach dem Datenflußparadigma. Der Datenfluß bei der Visualisierung wird auf ein Netzwerk von Modulen abgebildet, wobei die Schritte der Visualisierungspipeline auf vier Modultypen verteilt werden: den Eingabe-Modulen (*input modules*), den Filter-Modulen (*filter modules*), den Mapper-Modulen (*mapper modules*) und den Ausgabe-Modulen (*output modules*). Jedes Modul kann mehrere Daten-

und Parameter-Eingänge sowie Daten-Ausgänge besitzen; diese Ein- und Ausgänge werden über Datenleitungen miteinander verbunden und bauen damit einen gerichteten Graphen, ein Netzwerk, auf.

In AVS besteht der Programmcode für jedes Modul aus mindestens zwei Teilen: dem Beschreibungsteil (*module description*) und dem eigentlichen Kern des Moduls, dem Ausführungsteil (*compute routine*). Weiterhin können eine Initialisierungsroutine (*init routine*) und eine Routine zur Auflösung des Moduls (*destroy routine*) für jedes Modul festgelegt werden.

Im Beschreibungsteil wird festgelegt, welche Ein- und Ausgänge das Modul besitzt und von welchem Datentyp diese Ein- und Ausgänge sind. Außerdem wird in diesem Teil die Benutzerschnittstelle mit den Eingabe-Parametern festgelegt. Zu jedem Eingabe-Parameter kann hierzu das sogenannte *widget* bestimmt werden, wie zum Beispiel für einen ganzzahligen Parameter, ob seine Eingabe über eine Wählscheibe, eine Text-Eingabestelle oder über einen Schieber erfolgen soll. Solche *widgets* können frei auf Panels, in Fenstern oder auf Stacks platziert werden; damit kann die Benutzeroberfläche flexibel gestaltet werden. Default- und Grenzwerte für die Eingabe-Parameter werden ebenfalls Beschreibungsteil festgesetzt.

Zum Beschreibungsteil gehören weiterhin die Deklaration der Namen der *compute*-, der *init*- und der *destroy routine*, sowie die Angabe des Header-Files, in dem benutzerdefinierte Datenstrukturen definiert sind.

Im Ausführungs-, Initialisierungs- und Auflösungssteil des Moduls werden dann mit Hilfe des API (*application programmer's interface*) die eigentlichen Algorithmen und Verfahren des Moduls implementiert. Der Initialisierungssteil des Moduls wird bei der Instanzierung des Moduls aufgerufen, d.h., wenn ein Modul aus der Bibliothek auf die Arbeitsfläche gezogen wird. Leider können für diesen Teil des Moduls nicht alle Routinen des APIs genutzt werden [AVSD92]. Wenn ein Modul vernichtet werden soll (in AVS wird es dazu auf ein Hammer-Icon auf der Arbeitsfläche gezogen), so wird zuvor noch die Auflösungsroutine durchlaufen.

Der Ausführungsteil ist der Hauptbestandteil des Moduls. Er besteht aus einer Funktion, die immer dann aufgerufen wird, wenn sich ein Eingang des Moduls geändert hat.² Weil die Module *demand driven* arbeiten, also nur dann "angestoßen" werden, wenn sich einer ihrer Daten- oder Parameter-Eingänge ändert, ist es sinnvoll, mit Zuständen zu arbeiten: sobald der Ausführungsteil des Moduls angestoßen wird, wird zuerst geprüft, in welchem Zustand sich das Modul befindet. Erst danach wird der für diesen Zustand vorhergesehene Programmteil ausgeführt. Beispiele für solche Zustände werden noch in den Abschnitten 5.2.3 und 5.2.4 gegeben werden.

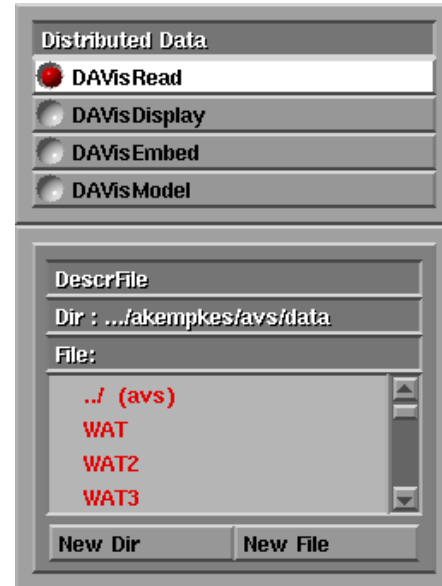
²Es gibt auch AVS-Module, die als sogenannte Coroutinen arbeiten. Sie arbeiten kontinuierlich und der Aufruf ihres Ausführungsteiles ist unabhängig von der Änderung der Eingänge. Keines der hier besprochenen Module muß als Coroutine implementiert werden; alle Module sind sogenannte Subroutinen.

5.2.2 DAVisRead



Mit dem Modul DAVisRead werden Dateien gelesen, in denen Parameter für eine Gebietszerlegung nach dem Optimierungsverfahren der K-Funktion hinterlegt sind; in der jetzigen Version von TRACE werden nur Gebietszerlegungen dieses Typs verwendet. Das Dateiformat lautet für ein Beispiel mit einem Gitter der finiten Elemente von $150 \times 40 \times 40$ Knoten und einer Zerlegung in 40 Teilgebiete wie folgt:

```
#Nx Ny Nz
#-- Anzahl der Knoten in x,y,z-Richtung:
150 40 40
#Delta x Delta y Delta z
#-- Abstände zwischen den Knoten in
#   x,y,z-Richtung:
5.0 5.0 5.0
#Einheiten in x-Richtung
m
#Einheiten in y-Richtung
m
#Einheiten in z-Richtung
m
#np
#-- Anzahl der Teilgebiete im
#   Gesamtgebiet
40
#L
#-- Breite der Überlappzonen
#   zwischen den Domänen
3
```



Wie im Beispiel zu sehen ist, dürfen beliebig viele Kommentarzeilen (das sind Zeilen mit einem #-Zeichen als erstem Zeichen in der Zeile) zwischen den Angaben eingefügt werden. Das Modul DAVisRead liest diese Datei aus und gibt an seinem linken Modulausgang eine Datenstruktur vom Typ `partinfostruct` aus; am rechten Ausgang ein eindimensionales AVS-Feld dreidimensionaler Vektoren. Das AVS-Vektorfeld beschreibt die Domänen einer optimierten Teilgebietszerlegung des streng regulären Gitters in `np` Teilgebiete so, wie es in Abschnitt 5.1.4 gezeigt wurde. Die Daten für die Struktur `partinfostruct` können einfach aus der Datei in die Struktur hinüberkopiert werden. Die Bitfolge `selection` wird vollständig mit Nullen belegt.

Der Benutzer kann die Datei aus einem Datei-Selektor (siehe Bild) auswählen; sobald die Datei vollständig gelesen wurde, liegt an einem Ausgang die Datenstruktur

vom Typ `partinfostruct`, und am anderen Ausgang ein AVS-Vektorfeld mit den entsprechend aktualisierten Dateninhalten vor.

5.2.3 DAVisDisplay



In Abschnitt 4.4.2 wurden zwei Aufgabengebiete für das Modul DAVisDisplay angedeutet: zum einen die grafisch-schematische Darstellung der Domänen-Aufteilung, und zum anderen die Möglichkeit zur Selektion der Teilgebiete. Weil in der jetzigen Version von TRACE noch kein Bedarf darin besteht, Domänen zu Gruppen-Domänen gruppieren zu können, wurde vorläufig noch auf die Implementation des Verfahrens für die konvexe Hülle um Gruppen-Domänen verzichtet; aus diesem Grund benötigt auch kein Modul der DAVis-Suite eine Datenstruktur zur Repräsentation der Domänen-Hierarchie.

Für die grafisch-schematische Darstellung der Domänen-Aufteilung liest DAVisDisplay die beiden Datenstrukturen aus, die es von DAVisRead erhält: das AVS-Feld und die `partinfostruct`-Struktur. Aus den Angaben in diesen Datenstrukturen erzeugt DAVisDisplay eine AVS-Geometrie, die an ein Standard-AVS-Modul weitergeleitet werden kann, den Geometry Viewer. DAVisDisplay selbst besitzt keine grafische Ausgabeschnittstelle oder eine Möglichkeit zur grafisch-schematischen Darstellung der Domänen-Aufteilung. Erst der Geometry Viewer stellt die AVS-Geometrie aus DAVisDisplay grafisch dar. Dort sind auch die Möglichkeiten zur interaktiven Manipulation der Geometrie bereitgestellt: so lassen sich die AVS-Geometrien drehen, vergrößern und verschieben. Darstellungsparameter, wie zum Beispiel Kameraeinstellung, Beleuchtung oder Render-Methode, können innerhalb des Geometry Viewers verändert werden.

Der Geometrie-Eingang des Geometry Viewers kann mit mehreren AVS-Geometrien belegt werden; alle diese Geometrien werden im gleichen Darstellungsraum vom Geometry Viewer dargestellt. So läßt sich auch die Geometrie, die von DAVisDisplay erstellt wurde, in andere Geometrien einblenden, wie zum Beispiel in eine Ansicht eines von DAVisModel modellierten Feldes. Weil in DAVisModel die Lage und Größe der Domänen verändert werden kann, wurden auch für DAVisDisplay Möglichkeiten zur interaktiven "Modellierung" der schematischen Darstellung eingerichtet. So lassen sich zum Beispiel die Domänen voneinander entfernen ("Explode") oder verkleinern ("Shrink"), und können somit der Ansicht des Feldes angepaßt werden.

Der Benutzer kann einige Darstellungsparameter der Domänen von DAVisDisplay aus steuern. Selektierte und nicht selektierte Domänen sollen in der Darstellung unterschieden werden können, daher wurden beispielhaft die folgenden Darstellungsmöglichkeiten zur Implementation ausgewählt:

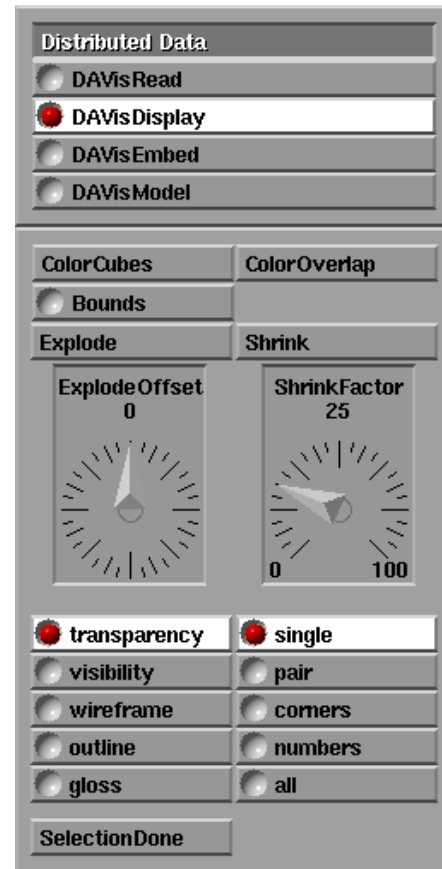
transparency: Die selektierten Domänen werden mit einer opaken Außenhülle, die unselektierten Domänen mit einer transparenten Außenhülle dargestellt.

visibility: Die selektierten Domänen werden mit einer opaken Außenhülle dargestellt; die unselektierten Domänen werden aus der Darstellung ganz entfernt.

wireframe: Die selektierten Domänen werden mit einer opaken Außenhülle dargestellt; die unselektierten Domänen werden als Drahtgittermodell dargestellt.

outline: Alle Domänen werden mit einer opaken Außenhülle dargestellt; die selektierten Domänen werden zusätzlich mit einem hellen Drahtgitter umrahmt.

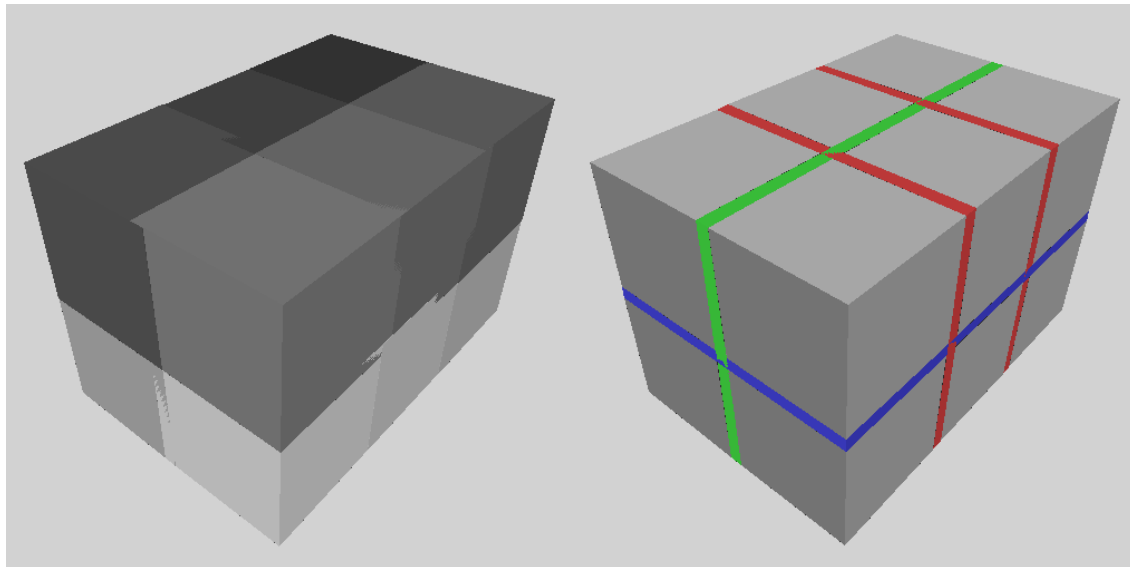
gloss: Auch hier werden alle Domänen mit einer opaken Außenhülle dargestellt; in der Darstellung glänzen jedoch die selektierten Domänen, während die nicht selektierten Domänen eine stumpfe, matte Außenhülle haben.



Auf die Unterscheidung von selektierten und nicht selektierten Domänen durch unterschiedliche Farben wurde verzichtet, da eine andere Option bereits auf eine Farbcodierung (bzw. Grauwertkodierung) zurückgreift: die Anzeige der Partitions- bzw. Domänennummer, die den einzelnen Teilgebieten automatisch von DAVisDisplay vergeben werden. Dabei erhält das Teilgebiet mit den kleinsten Koordinaten die Nummer 1. Die Nummer wird so weiter aufwärts gezählt, daß sie in positiver x-Richtung jeweils um eins, in y-Richtung jeweils um die Anzahl der Domänen im Teilgebiet in x-Richtung, und in z-Richtung jeweils um die Anzahl der Domänen einer xy-Lage wächst. Nach dieser Numerierung können die Domänen mit einem zunehmend dunkleren Grauton eingefärbt werden, wie Abb. 5.4a zeigt.

Hilfreich zur Orientierung ist auch die Darstellung der Überlappzonen durch farbige Bänder, wie Abb. 5.4b zeigt. Sie läßt sich bei Bedarf ein- oder ausschalten.

Die zweite Aufgabe, die von DAVisDisplay erfüllt werden soll, ist die Möglichkeit zur interaktiven Selektion der Teilgebiete. Dazu wird die Eigenschaft des Geometry Viewers ausgenutzt, Mausbewegungen, Mausklicks etc. im Darstellungsfenster abfragen zu können. Bei jedem Ereignis wird eine Struktur vom Typ `upstream_geom` vom Geometry Viewer ausgefüllt und an das Modul zurückgeschickt, dessen Geometrie gerade involviert ist. Die Struktur `upstream_geom` enthält unter anderem Informationen über den Zustand des Mausknopfes (wurde der Mausknopf gerade ge-



a.

b.

Abbildung 5.4: Eingefärbte Domänen und Überlappzonen

drückt, wurde die Maus dazu bewegt, oder wurde der Mausknopf gerade losgelassen), und den Namen des Objektes, auf das der Mauszeiger gerade wies. DAVisDisplay kann also Informationen darüber erhalten, welche Domäne der Benutzer im Darstellungsfenster angeklickt, d.h. selektiert hat, und diese Information an DAVisEmbed weiterleiten.

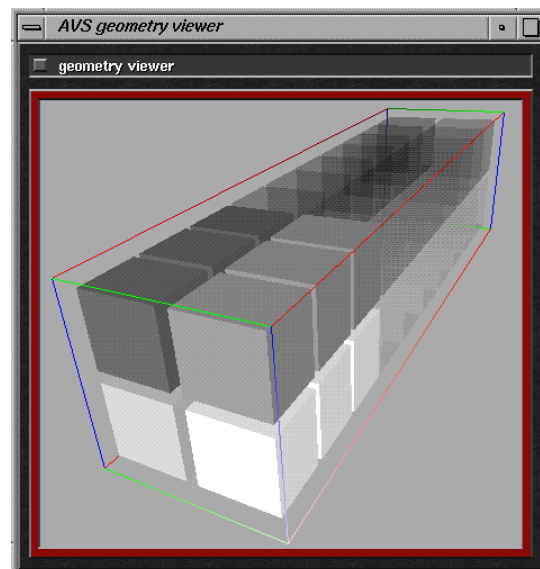


Abbildung 5.5: Selektion von Domänen

Die Domänen können auf verschiedene Weisen ausgewählt werden:

single: Einzelne Domänen werden durch einen Mausklick ausgewählt. Es ist ein *sweeping* möglich, also die Auswahl derjenigen Domänen, die während einer Mausbewegung mit gedrückter Taste berührt wurden.

pair : Durch Anklicken einer Überlappzone werden die Domänen beiderseits einer Überlappzone ausgewählt. Für diese Auswahlmethode muß allerdings die Darstellung der Überlappzonen eingeschaltet sein.

corners: Nach der Auswahl zweier Domänen gelten alle Domänen innerhalb des Blocks, der durch die beiden Domänen aufgespannt wird, als ausgewählt. In Abb. 5.5 wurden zum Beispiel die untere Domäne vorne rechts sowie die dritte Domäne von vorne, links oben ausgewählt. Ein *sweeping* ist bei dieser Selektionsmethode ebenfalls möglich.

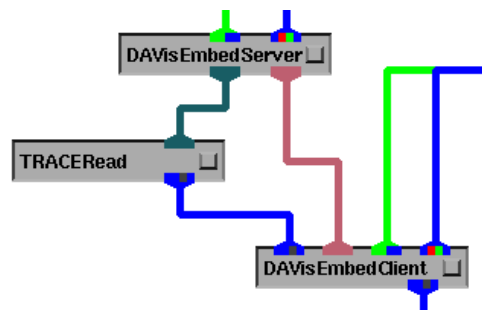
numbers: Nach der Auswahl zweier Domänen mit den Nummern a und b gelten alle Domänen mit Partitionsnummern von a bis b als ausgewählt. Auch hier können die Domänen gleitend mit einem *sweeping*-Verfahren selektiert werden.

all: Nach Anklicken einer beliebigen Domäne gelten alle Domänen als selektiert oder als nicht selektiert, je nach dem, ob eine unselektierte oder selektierte Domäne angeklickt wurde.

Alle Auswahlmethoden (bis auf die Methode "all") invertieren den Selektionszustand der Domänen, die ausgewählt wurden. War eine Domäne also vor der Auswahl nicht selektiert, so ist sie nach der Auswahl durch eine der obigen Methoden selektiert und umgekehrt.

Mit **SelectionDone** wird die Selektion als Bitfolge an DAVisEmbed weitergeleitet; aus der Bitfolge bestimmt DAVisEmbed, welche Feld-Partitionen es einlesen soll.

5.2.4 DAVisEmbed



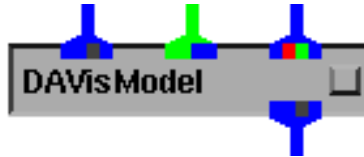
Die Aufgabe, die verteilten Datenfelder in eine Feld-Datenstruktur einzubetten, wurde auf drei Module verteilt: DAVisEmbedClient, DAVisEmbedServer und ein Lesemodul. Der Grund für die Aufteilung der Aufgabe liegt in der gewünschten

Flexibilität, möglichst viele verschiedene Dateiformate lesen zu können; die Lesemethode für die verteilten Datenfelder soll also modular austauschbar sein. Das Client-Modul DAVisEmbedClient übernimmt die Einbettung der Datenfelder in eine Feld-Datenstruktur, wie in Abschnitt 4.3.3 beschrieben wurde. Einstweilen wurde die Einbettung auf den Fall B und auf reguläre Feld-Partitionierungen beschränkt, die Partitionsnummer p wird also den abhängigen Variablen zugeordnet.

DAVisEmbedClient erhält zunächst die Information von DAVisDisplay, welche Domänen der Benutzer visualisieren möchte. Daraufhin wird eine entsprechend dimensionierte Feld-Datenstruktur, ein AVS-Feld, alloziert, in der genügend Platz sowohl für alle selektierten Partitionen, als auch für die Trennmarkierungen ist.³

Nach der Allokierung können die verteilten Datenfelder eingelesen und in die Feld-Datenstruktur $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$ eingebettet werden. Dazu schickt das Client-Modul nacheinander für jede Feld-Partition eine Anforderung an das Server-Modul. Das Server-Modul ist eigentlich ein Server-Agent; es setzt lediglich einen Dateinamen aus einem Stammnamen und der Partitionsnummer zusammen und schickt ihn mit der Aufforderung, diese Datei zu lesen, an das Lesemodul weiter. Das Lesemodul liest die geforderte Datei ein und schickt die Daten in einem AVS-Feld, also einer Feld-Datenstruktur, an das Client-Modul. DAVisEmbedClient bettet das AVS-Feld in die Feld-Datenstruktur $\mathcal{F}(\underline{X}_C, \underline{Y}_C)$ ein und sendet danach die Aufforderung an DAVisEmbedServer, das nächste Feld vom Lesemodul einlesen zu lassen.

5.2.5 DAVisModel



Mit DAVisModel läßt sich die unabhängige Variable in der Feld-Datenstruktur aus DAVisEmbed interaktiv manipulieren und modellieren. Wie auch in DAVisDisplay lassen sich die Domänen voneinander entfernen (*explode*), um den Blick auf die Überlappzonen freizugeben. Weitere Modelliermöglichkeiten bestehen für die Überlappzonen. Die Orte in den Überlappzonen können entweder um einen kleinen Versatz verrückt (*displacement*) oder näher an die jeweilige Domäne herangezogen werden (*compress*), um eine eindeutige Abbildung zu erhalten. Die Vorschriften zur Modellierung wurden bereits in Abschnitt 4.3.5 vorgestellt. Nachdem die Orte $\underline{X}_M[r_i, i]$ berechnet sind, werden in einem Schritt zur Nachbearbeitung die Koordinaten der Leerstellen auf die Nachbarwerte gesetzt, wie in Gleichung (4.44) beschrieben wurde.

Weil in AVS kein undefinierter Wert vorgesehen ist, wurde die Option eingebaut, die Leerstellen mit anderen Werten zu belegen: entweder mit den Randwerten der Domänen oder mit einem vom Benutzer spezifizierten Wert (**user value**).

³Wenn der Benutzer also mehrere Teilgebiete auswählt, die zusammengekommen keinen blockförmigen Ausschnitt des Gesamtgebietes bilden, wird es nach der Einbettung Feld-Partitionen in $\underline{Y}_C$ geben, die vollständig mit dem undefinierten Wert $\perp$ belegt sind.

5.3 Benutzeroberfläche

Das Zusammenstellen von Modulen zu Netzwerken in AVS erfordert genaue Kenntnisse über die Funktion und den Gebrauch der Module. Um den Anwender von der Aufgabe zu entlasten, AVS-Netzwerke selbst aufbauen zu müssen, wurde eine Benutzeroberfläche geschaffen, mit der alle Module der DAVIS-Suite sowie auch die Standard-AVS-Module automatisiert zu Netzwerken verknüpft werden können. Wie bereits in Abschnitt 4.5 erläutert wurde, gibt es innerhalb der Oberfläche vier Arten von DAVIS-Modulen: Daten-Module, Module zur Modellierung der Daten, View-Module und Viewer-Module. Diese Vierstufigkeit des Visualisierungsprozesses prägt auch den Aufbau der Benutzeroberfläche.

5.3.1 Aufbau

Charakteristisch für die Benutzeroberfläche sind das Hauptpanel am unteren Rand des Bildschirms und die Menuleiste am oberen Rand, siehe Abb. 5.6. Der Benutzer kann aus der Menüleiste ein DAVIS-Modul auswählen. Das System instanziiert dann dieses Modul, d.h. es legt das Modul auf die Arbeitsfläche des Netzwerk-Editors von AVS. Die Module werden dabei automatisch auf der Arbeitsfläche so platziert, daß von links nach rechts vier Spalten mit den jeweils instanziierten Daten-, Modell-, View- und Viewer-Modulen eingerichtet werden. Die DAVIS-Module setzen sich in den meisten Fällen aus mehreren Standard-AVS-Modulen oder Modulen der DAVIS-Suite zusammen. Bei der Instanzierung wird daher auf Skripte zurückgegriffen, in denen genau beschrieben ist, aus welchen AVS-Modulen sich das DAVIS-Modul zusammensetzt und wie die AVS-Module untereinander verknüpft sind. Im Skript wird auch festgelegt, wo sich die Ein- und Ausgänge des DAVIS-Moduls befinden.

Sobald ein Modul neu instanziiert wird, erscheint sein Name in einem kleinen Sichtfenster im Hauptpanel. Mit den Schaltern "LinkXX" oder "UnlinkXX" können die DAVIS-Module miteinander verbunden oder bestehende Verbindungen zwischen den Modulen aufgelöst werden. Um erkennen zu können, ob zwei Module miteinander verbunden sind oder nicht, erscheint in einem Sichtfenster ein Pfeil, wenn eine Verbindung zwischen den Modulen besteht, ansonsten bleibt das Fenster leer.

Wenn eines der Schaltknöpfe "SelectXX" betätigt wird, erscheint eine Liste aller Module, die bislang aus der Modulkategorie XX instanziiert wurden. Der Benutzer kann daraus ein Modul auswählen. Nach der Auswahl erscheint der Name des Moduls im Sichtfenster und es können Verbindungen von oder zu diesem Modul eingerichtet oder aufgelöst werden. Besitzt ein Modul Eingabeparameter, so kann das zugehörige Panel mit den Bedienelementen für die Parameter des Moduls mit dem Schalter "XXPanel" aufgerufen werden. Nochmaliges Drücken des Knopfes "XXPanel" läßt das Panel wieder verschwinden.

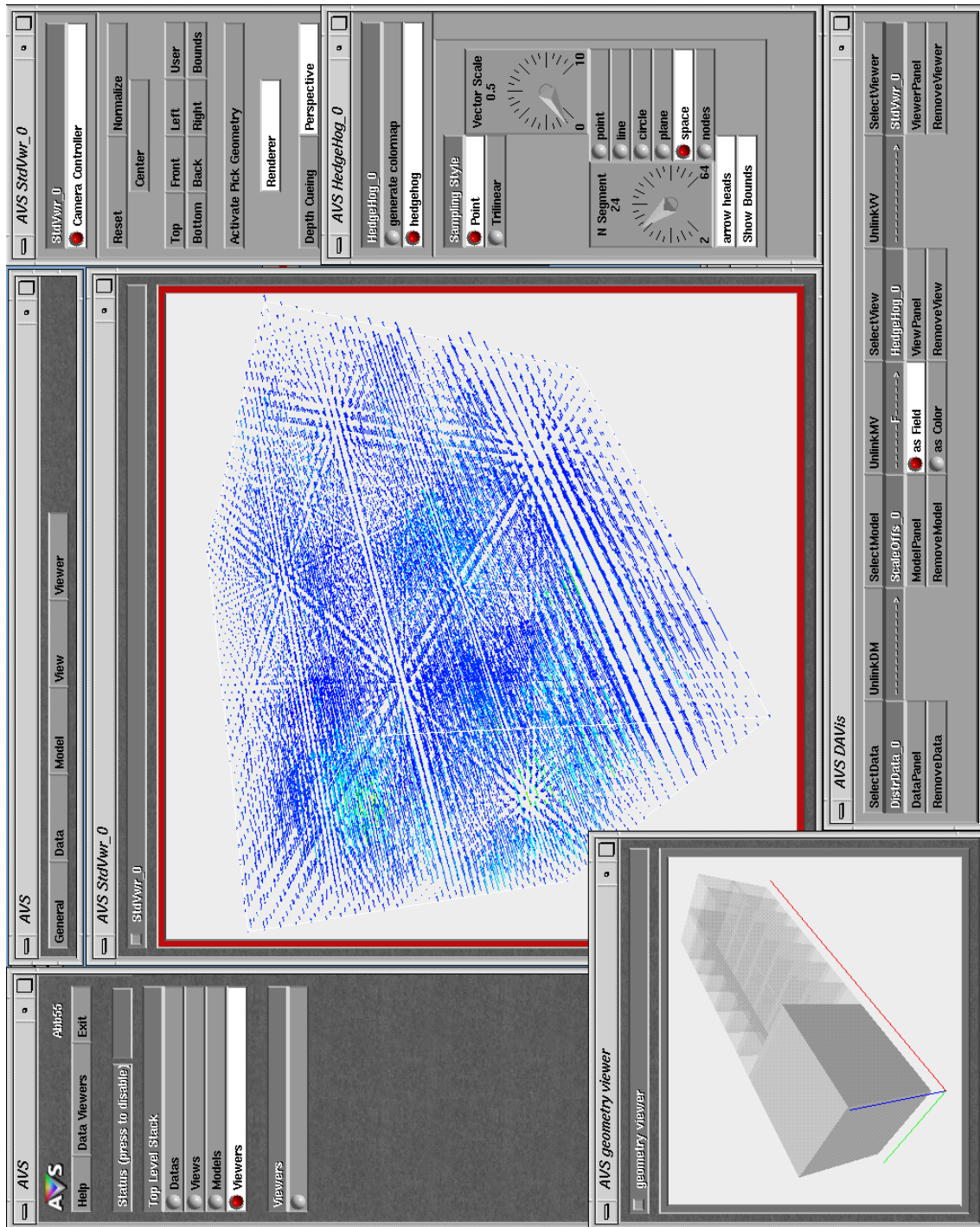


Abbildung 5.6: Die Oberfläche von DAVisTool

5.3.2 DAVisMenu

Das Modul DAVisMenu erledigt alle Aufgaben, die im Zusammenhang mit der Benutzeroberfläche stehen, also die Verwaltung der Module und des Gesamtnetzwerks, die Abfrage von Benutzereingaben, soweit sie die Instanzierung von Modulen oder Verbindungen betreffen, und die Bereitstellung der Panele, mit denen die Modulparameter eingestellt werden können.

DAVisMenu kann sich in vier Zuständen befinden: im Zustand `INIT_STATE`, im Zustand `MAIN_STATE`, im Zustand `SELECTOR_WAIT`, oder im Zustand `MODPORT_WAIT`. Der Zustand `INIT_STATE` wird nur am Anfang bei der Instanzierung von DAVisMenu angenommen, also wenn das Modul DAVisMenu auf die Arbeitsfläche von AVS gebracht wird. Es wird dann sofort ein Skript gestartet, das die Bedienelemente der Oberfläche in der in Abb. 5.6 gezeigten Weise auf Panele und in der Menüleiste arrangiert. Danach befindet sich das Modul DAVisMenu im Zustand `MAIN_STATE`.

Im Zustand `MAIN_STATE` können DAVis-Module neu instanziiert, ausgewählt oder entfernt werden oder Verbindungen zwischen den Modulen eingerichtet oder aufgelöst werden. Dieser Zustand ist also tatsächlich der Zustand, in dem sich das Modul die meiste Zeit befindet. Sollen Module neu instanziiert werden, so wird ein Instanzierungsskript für das entsprechende Modul geladen, ausgeführt und die Bedienelemente auf den Panele angeordnet. Die AVS-Module, aus denen das DAVis-Modul besteht, erhalten eindeutige Namen, damit sie im weiteren korrekt angesprochen werden können. Nach der Instanzierung wird in den Zustand `MODPORT_WAIT` geschaltet.

Wenn ein Modul aus einer Liste ausgewählt werden soll (der Benutzer hat auf einen "SelectXX"-Knopf gedrückt), so wird die Liste angezeigt und vom Zustand `MAIN_STATE` in den Zustand `SELECTOR_WAIT` übergegangen.

Im Zustand `SELECTOR_WAIT` wartet DAVisMenu darauf, daß der Benutzer ein DAVis-Modul aus der Modulliste auswählt. Sobald der Benutzer ein Modul selektiert hat, entfernt DAVisMenu die Liste und das selektierte Modul erscheint im entsprechenden Sichtfenster des Hauptpanels. Danach wird der Zustand des Moduls wieder auf den Zustand `MAIN_STATE` gesetzt.

Im Zustand `MODPORT_WAIT` wartet das Modul DAVisMenu darauf, daß sich sein Eingang `ModPorts` ändert. In jedem Instanzierungsskript eines DAVis-Moduls gibt es nämlich eine Zeile mit dem Befehl, eine Zeichenkette an diesen Eingang `ModPorts` von DAVisMenu zu senden. Die Zeichenkette beschreibt, wo die Ein- und Ausgänge des DAVis-Moduls unter den AVS-Modulen zu finden sind. Als Beispiel hierzu sei das Instanzierungsskript des DAVis-Moduls `Isosurf` betrachtet, ein View-Modul zur Erzeugung einer Isofläche:

```
#!/usr/bin/avs -network
version 5.02 (50.62 sg2)
module "generate colormap" -alias Isosurf_GC
module "field legend" -alias Isosurf_FL \
    -ex $Path/avs_library/field_legend
```

```

module "color range" -alias Isosurf_CR -ex $Path/avs_library/mongo
module isosurface -alias Isosurf_IS -ex $Path/avs_library/tile
port_vis Isosurf_IS:level -on

port_connect Isosurf_GC:0 Isosurf_CR:1
port_connect Isosurf_GC:0 Isosurf_FL:1
port_connect Isosurf_FL:0 Isosurf_IS:3
port_connect Isosurf_CR:0 Isosurf_IS:2

manipulator Isosurf_GC:colormap -w color_editor \
    -xy 10,10 -wh 236,416
manipulator "Isosurf_GC:lo value" -w typein_real \
    -xy 10,426 -wh 118,22
manipulator "Isosurf_GC:hi value" -w typein_real \
    -xy 128,426 -wh 118,22
manipulator Isosurf_IS:level -w dial -xy 10,10 -wh 90,130
manipulator "Isosurf_IS:optimize surf" -w toggle \
    -xy 100,10 -wh 118,22
manipulator "Isosurf_IS:optimize wire" -w toggle \
    -xy 100,32 -wh 118,22
manipulator "Isosurf_IS:generate normals" -w toggle \
    -xy 100,54 -wh 118,22
manipulator "Isosurf_IS:flip normals" -w toggle \
    -xy 100,76 -wh 118,22
panel Isosurf_CR -hide

parm_set MenuMgr:ModPorts \
    "InF@Isosurf_IS:0 &Isosurf_FL:0 \
    InC@Isosurf_IS:1 &Isosurf_CR:0 \
    Out@Isosurf_IS:0"

# End of file

```

Das DAVis-Modul **Isosurf** setzt sich aus vier AVS-Modulen zusammen (siehe die vier **module**-Befehle am Anfang des Skripts): dem Modul **generate colormap**, das eine Farbtabelle erzeugt, dem Modul **field legend**, das eine Farbskala einrichtet, dem Modul **color range**, das die Farbtabelle auf die Feldgrößen normiert, und dem Modul **isosurface**, das die Isofläche erzeugt. Alle diese AVS-Module erhalten einen Alias-Namen (siehe Option **-alias**), der mit dem Namen des DAVis-Moduls, hier **Isosurf**, beginnt. Dies ist notwendig, damit DAVisMenu erkennt, welche AVS-Module zu welchem DAVis-Modul gehören, denn beispielsweise wird das AVS-Modul **generate colormap** auch in anderen DAVis-Modulen verwendet. Die **port_connect**-Befehle legen die internen Verbindungen im DAVis-Modul fest, wie sie in Abb. 5.7a gezeigt werden: vom **generate colormap**-Modul wird eine Farbtabelle an die Module **color range** und **field legend** geschickt (gelbe Leitungen),

das Modul `field legend` leitet einen Zahlenwert an das Modul `isosurface` weiter (violette Leitung), und schließlich gibt das Modul `color range` seinerseits eine Farbtabelle an das Modul `isosurface` weiter. Weil ein AVS-Modul mehrere Ein- und Ausgänge haben kann, gehört zu jeder Spezifikation eines Modulein- oder -ausgangs eine sogenannte Portnummer, die vom jeweiligen Modulnamen durch einen Doppelpunkt getrennt wird.

Mit den `manipulator`-Befehlen werden die Bedienelemente des DAVis-Moduls übersichtlich auf Panels oder Stacks arrangiert. Mit der Verknüpfungslogik haben sie nichts zu tun, deshalb wird auf eine Erläuterung des `manipulator`-Befehls an dieser Stelle verzichtet (siehe dazu [AVSD92]). Wichtiger ist vielmehr der `parm_set`-Befehl, mit dem die oben angesprochene Zeichenkette an das Modul DAVisMenu (alias `MenuMgr`) gesendet wird. Wie schon in Abschnitt 4.5 erwähnt wurde, besitzt ein View-Modul zwei Eingänge – einen Feldeingang und einen Farbeingang –, sowie einen Ausgang. Das `Isosurf`-Modul soll nun mit Hilfe des Feldes am Feldeingang eine Isofläche erstellen; zum Einfärben dieser Isofläche soll das Feld am Farbeingang verwendet werden. Die Zeichenkette

```
InF@Isosurf_IS:0 &Isosurf_FL:0 \
InC@Isosurf_IS:1 &Isosurf_CR:0 \
Out@Isosurf_IS:0
```

teilt DAVisMenu mit, wo es den Feld- und den Farbeingang, sowie den Ausgang unter den AVS-Modulen finden kann: das Feld am Feldeingang soll an die Moduleingänge `Isosurf_IS:0` und `Isosurf_FL:0` geleitet werden, das Feld am Farbeingang hingegen an die Moduleingänge `Isosurf_IS:1` und `Isosurf_CR:0`. Der Ausgang des DAVis-Moduls `Isosurf` befindet sich am Modul `Isosurf_IS`; es ist dort der Port mit der Nummer Null. Das DAVis-Modul `Isosurf` wird mit den anderen Modulen so verbunden, wie Abb. 5.7 zeigt.

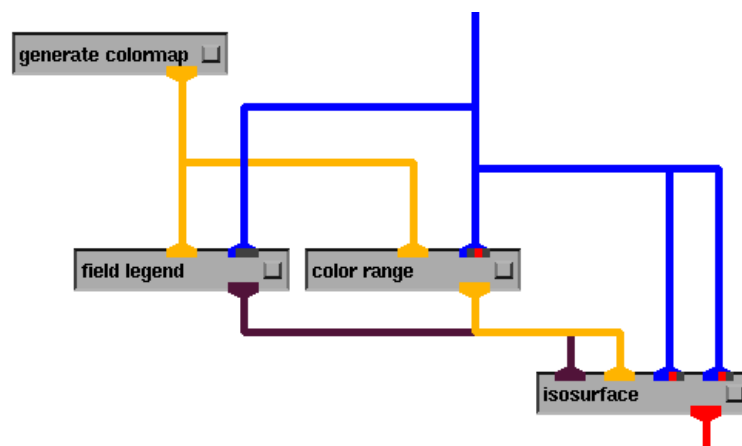


Abbildung 5.7: Das DAVis-Modul Isosurf

Im Zustand `MODPORT_WAIT` wartet DAVisMenu auf diese Zeichenkette. Sobald das Modul die Zeichenkette erhalten hat, wird sie analysiert und in einer Datenstruktur abgelegt. Die Information über die Modulein- und -ausgänge soll leicht wieder aufzufinden sein, denn schließlich wird sie bei jedem Erstellen oder Auflösen einer Verbindung zwischen zwei DAVis-Modulen benötigt. Nachdem DAVisMenu den Zustand `MODPORT_WAIT` abgearbeitet hat, geht es wieder in den Zustand `MAIN_STATE` über.

5.3.3 Das Netzwerk der Module der DAVis-Suite

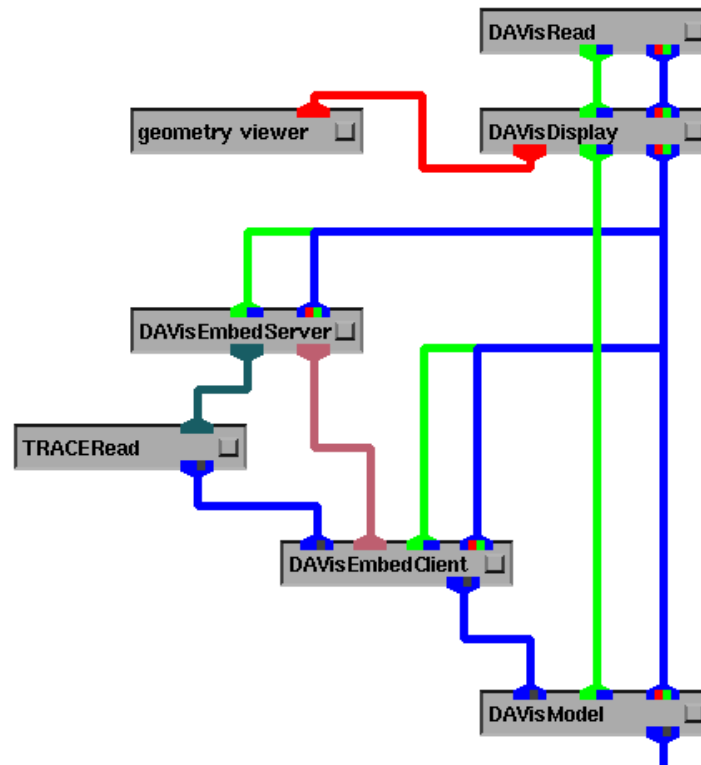


Abbildung 5.8: Das Netzwerk der Module der DAVis-Suite in AVS

Aus den Modulen DAVisRead, DAVisDisplay, DAVisEmbed und DAVisModel lässt sich nun ein Netzwerk zusammenstellen, wie es in Abb. 5.8 dargestellt ist. Darin leitet DAVisRead die Information über die Domänen-Aufteilung als benutzerdefinierte Datenstruktur (linker Ausgang) und als AVS-Feld dreidimensionaler Vektoren (rechter Ausgang) an das Modul DAVisDisplay weiter. DAVisDisplay generiert aus diesen Informationen eine Geometrie, die es im AVS-Standard-Modul "Geometry Viewer" grafisch darstellen lässt. Aus den Benutzereingaben entnimmt DAVisDisplay, welche Domänen visualisiert werden sollen und sendet sie innerhalb der Benutzerdefinierten Datenstruktur an DAVisEmbedServer weiter. DAVisEmbedServer, DAVisEmbedClient und das Lesemodul, hier TRACERead, lesen die gewünschten Feld-Partitionen

ein und stellen sie in einer Feld-Datenstruktur zusammen, die anschließend an DAVisModel weitergereicht wird. Mit DAVisModel wird das Feld modelliert; dieses Feld wird anschließend den Viewer- oder Mapper-Modulen der Visualisierungsumgebung am Ausgang von DAVisModel zur Verfügung gestellt.

Kapitel 6

Visualisierung hydrogeologischer Daten

In diesem Kapitel wird beispielhaft gezeigt, wie mit den in der vorliegenden Arbeit aufgestellten Methoden hydrogeologische Daten der Programme TRACE und ParTrace visualisiert werden können.

6.1 Die Aufgabe

Vom Programm ParTrace wurde eine Sequenz von Konzentrationsverteilungen über mehrere Zeitschritte auf einem $150 \times 40 \times 40$ Knoten = 240.000 Knoten großen Finite-Elemente-Gitter berechnet. Diese soll zusammen mit der Information über die Verteilung der Permeabilität als Animation visualisiert werden; gleichzeitig soll das Fließfeld, das von TRACE verteilt berechnet wurde, in einer geeigneten Form in die Visualisierung eingeblendet werden.

6.2 Die Visualisierung der Sequenz aus ParTrace

Für die Visualisierung der Daten aus TRACE und ParTrace wird das Werkzeug DAVisTool verwendet, das in das *Applications*-Menü von AVS eingebunden wurde. Das zu DAVisTool zugehörige Skript hat die einzige Aufgabe, das Modul DAVisMenu zu instanzieren und die weitere Kontrolle an DAVisMenu zu übergeben. DAVisMenu erzeugt nach dem Aufruf das Hauptpanel am unteren und eine Menüleiste am oberen Rand des Bildschirms. Mit Menüleiste und Hauptpanel kann der Benutzer die DAVis-Module so einrichten und miteinander verbinden, wie es für die Visualisierungsaufgabe nötig ist.

Zunächst werden die Daten-Module eingerichtet. Die Dateien, die die Konzentrationsverteilungen des Schadstoffes beinhalten, liegen im AVS-Format vor (zum AVS-Format siehe [AVSU92]), können also mit dem Standard-AVS-Modul `read field`

eingelassen werden. Zur Visualisierung mehrerer Dateien in einer zeitlichen Abfolge wurden weitere Module zum `read field`-Modul beigefügt und daraus das DAVis-Modul `AnimAVSData` erstellt. Es wird aus der Menü-Leiste unter den Daten-Modulen ausgewählt; daraufhin erscheint im ersten Sichtfenster des Hauptpanels der Name des ersten Daten-Moduls, `AnimAVSData_0`. Damit ist das erste DAVis-Modul instanziiert. Durch Anklicken von "DataPanel" erscheint das Eingabe-Panel des Moduls (siehe Abb. 6.1) und die zu lesende Datei-Sequenz kann spezifiziert werden.

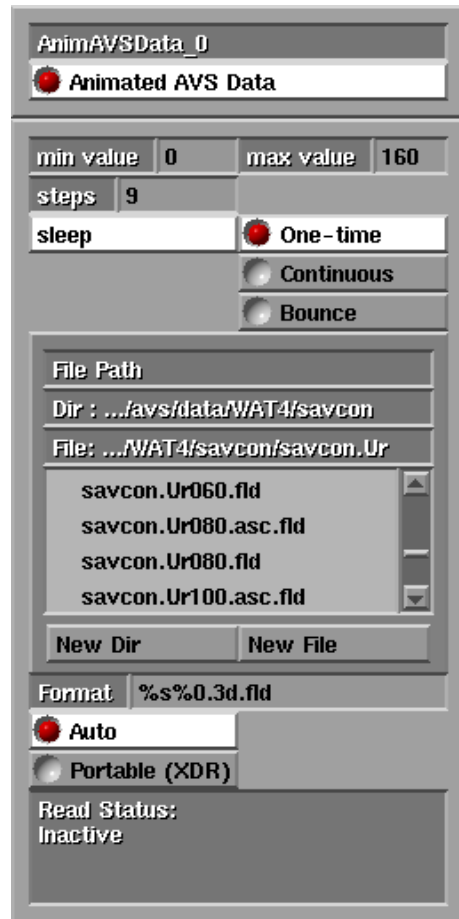


Abbildung 6.1: Spezifizieren der Eingabe-Sequenz

In diesem Fall liegen 9 Dateien mit Zeitschritten von 0 bis 160 Tagen aus einer Simulation von ParTrace vor. Der Dateiname beginnt mit dem Stamm `savcon.Ur`. Daran schließen sich drei Ziffern an, die den jeweiligen Zeitschritt angeben, sowie der Dateisuffix `.fld`. Der Stammname wird unter "New File" eingetragen; die Zeitschritte unter "min value" und "max value". Unter "Format" schließlich wird angegeben, wie der Dateiname zusammengesetzt wird; dazu wird die gleiche Notation verwendet wie beim `printf`-Befehl in C [BKer88, S. 243]. Mit dem Platzhalter `%s` wird der Stammname angesprochen, und der Platzhalter `%0.3d` reserviert die Stelle für den Zeitschritt.

Sobald mit den nächsten Schritten die Visualisierungspipeline vollständig aufgebaut ist, wird die Animation durch Ausschalten des "Schlafzustandes" ("sleep") begonnen

(siehe dazu auch [AVSM93, S. 43]. Es wird empfohlen, zunächst die Einstellung "one-time" beizubehalten, damit eine eventuelle fehlerbehaftete Eingabe bei der Spezifizierung des Dateinamens abgefangen werden kann.

Die Daten aus ParTrace sind elementorientiert, d.h. die Konzentrationswerte gelten jeweils für ein Element des FE-Gitters, und nicht für die Knoten. Um anschließend eine korrekte vergleichende Visualisierung mit den Ergebnissen aus TRACE zu erstellen – die Werte aus TRACE sind als Knotenwerte definiert –, wird ein Trick angewandt: als Modellierungsschritt wird zu den Koordinaten der unabhängigen Variablen jeweils die halbe Länge eines finiten Elements hinzuaddiert; damit "rutscht" das Gitter mit den Werten aus ParTrace genau zwischen das später hinzugefügte Gitter der Werte aus TRACE. Das Modul, das diesen Modellerschritt durchführt und eigens für DAVisTool geschrieben wurde, heißt **ScaleOffset**. Es wird aus der Menüleiste ausgewählt; die Parameter lassen sich nach Anklicken von "ModelPanel" so eingeben, wie Abb. 6.2 zeigt. Das hier verwendete Finite-Elemente-Gitter ist streng regulär; jedes Element hat die Form eines Würfels mit einer Kantenlänge von 5 Einheiten.

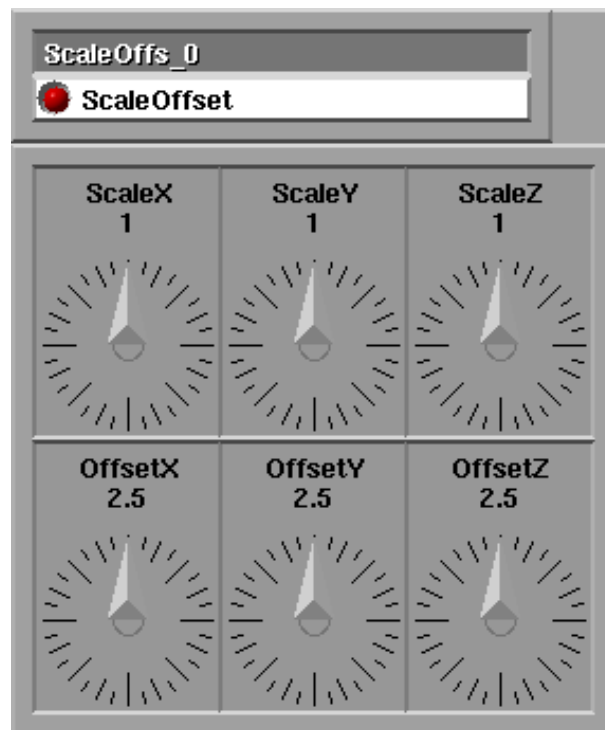


Abbildung 6.2: Verschieben des FE-Gitters von ParTrace um ein halbes Element

Für die Visualisierung der Konzentrationswerte empfiehlt sich die Darstellung als Isofläche. Dazu wird das Modul **Isosurf** durch Auswahl aus der Menüleiste unter "View" instanziiert. Die Visualisierungspipeline wird mit einem Viewer-Modul vom Typ **Standard Viewer** abgeschlossen. Jetzt müssen nur noch die Module untereinander verbunden werden; dies geschieht durch Anklicken der Knöpfe "LinkDM", "LinkMV" und "LinkVV", siehe Abb. 6.3.



Abbildung 6.3: Pipeline des Konzentrationsfeldes

Damit können die Daten aus **AnimAVSData** mit **ScaleOffset** modelliert werden. Mit Hilfe des Moduls **Isosurf** wird von diesem Modellfeld eine Ansicht als Isofläche erzeugt. Im Viewer wird die Isofläche schließlich dargestellt. Der Isowert der Oberfläche läßt sich an einer Farbskala einstellen.

Bisher wurde noch nicht berücksichtigt, daß das **Isosurf**-Modul – wie jedes View-Modul im DAVisTool – neben dem Feld-Eingang, der mit "LinkMV" soeben mit dem **ScaleOffset**-Modul verbunden wurde, auch einen Farb-Eingang besitzt. An diesem Farb-Eingang kann ein weiteres Modellfeld eingespeist werden, mit dessen Werten die Isofläche eingefärbt wird. Für die Beispiel-Visualisierung soll die Isofläche der Schadstoffkonzentration mit dem Wert der Permeabilität eingefärbt werden. Die Permeabilität liegt ebenfalls als elementorientierter Datensatz vor; weil sie statisch ist, kann das Daten-Modul **StandardAVSData** verwendet werden, um den Datensatz einmalig zu lesen. Die Permeabilitätsdaten werden so modelliert wie die Schadstoffkonzentrationen, d.h. mit einem zweiten **ScaleOffset**-Modul, das genauso parametrisiert wird, wie das **ScaleOffset**-Modul für die Schadstoffkonzentration (siehe Abb. 6.2). Mit den Modulen **StandardAVSData_1**, **ScaleOffset_1**, **Isosurf_0** und

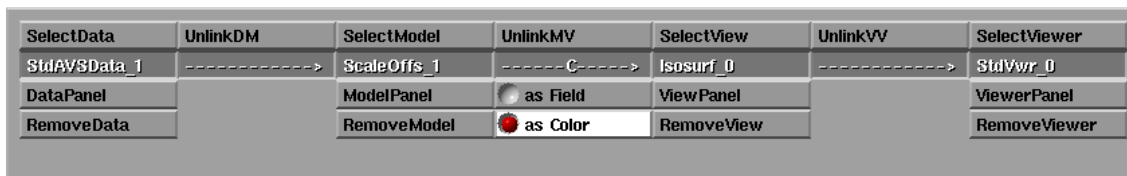


Abbildung 6.4: Pipeline des Permeabilitätsfeldes

StdVwr_0 wird eine zweite Visualisierungspipeline aufgebaut, wie sie in Abb. 6.4 dargestellt ist. Der Ausgang von **ScaleOffset_1** wird mit dem Farbeingang des Moduls **Isosurf_0** verbunden; dies erreicht man durch Auswahl von "LinkMV ... as Color".

Wenn nun das Modul **AnimAVSData** "aufgeweckt" wird, erscheinen im Viewer nacheinander neun Geometrien, die die Schadstoffkonzentration für einen bestimmten Wert als Isofläche darstellen. Danach kehrt das Modul **AnimAVSData** wieder in den "Schlafzustand" zurück. Die Animation läßt sich jedoch auch durch Anwahl von "continuous" oder "bounce" beliebig oft wiederholen.

Die Kameraeinstellung der dargestellten Szenerie lässt sich verändern. Hält man die mittlere Maustaste gedrückt und bewegt dabei die Maus, so dreht sich die Kamera um das Objekt; mit der rechten Maustaste kann die Kamera verschoben werden. Hält man gleichzeitig die Shift-Taste und die rechte Maustaste gedrückt und bewegt die Maus nach zurück, so wird die Kamera an die Geometrie herangefahren. Somit lässt sich eine Szene wie in Abb. 6.5 einstellen.

Die Achsen in Abb. 6.5 stammen aus dem DAVis-View-Modul `Axes`; dieses Modul benötigt weder Feld- noch Farbeingang. Es wird direkt an das Viewer-Modul `Std_Vwr_0` angeschlossen.

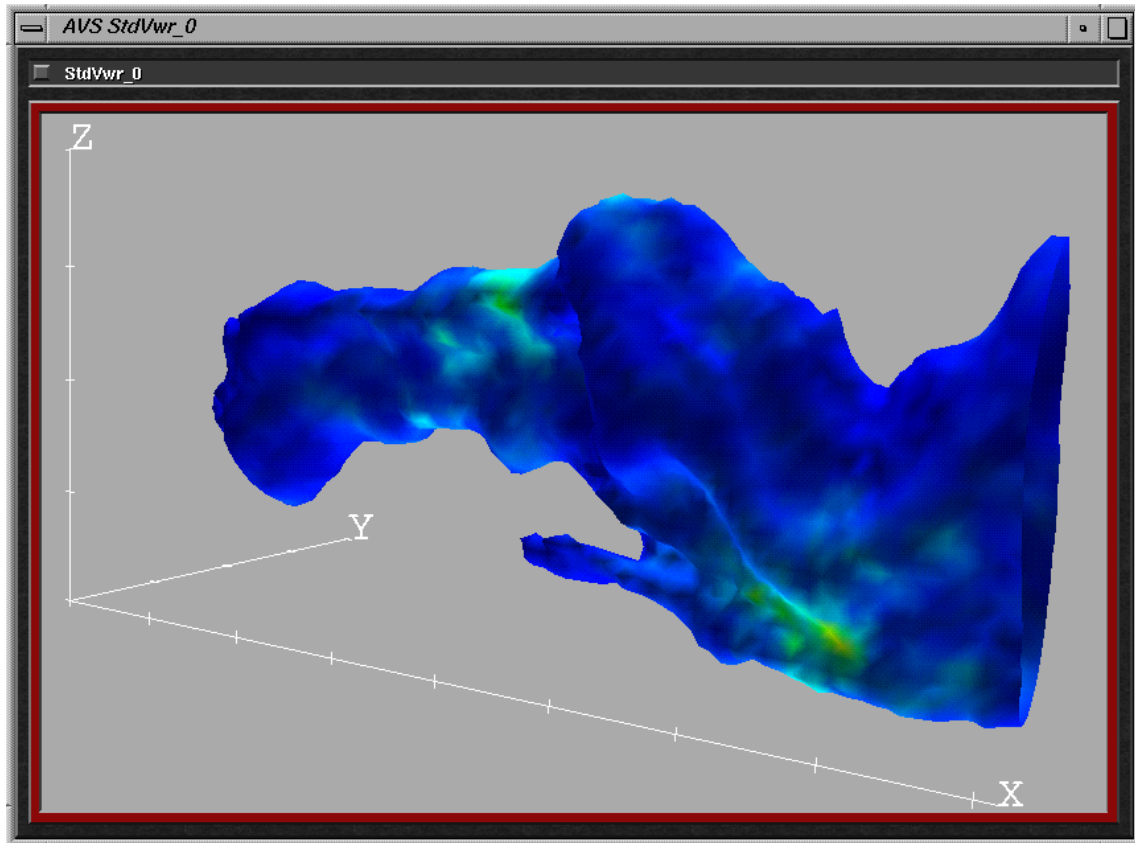


Abbildung 6.5: Konzentrationswolke im letzten Zeitschritt

Im Bild lässt sich durch die Einfärbung der Isofläche ein Zusammenhang zwischen Konzentration und Permeabilität erkennen: dort, wo die Permeabilität, also die Durchlässigkeit im Boden, größer ist, schnürt sich die Konzentrationswolke ein; dort, wo die Permeabilität im Boden gering ist, weitet sich die Wolke aus. Vermutlich wird die Fließgeschwindigkeit an den eingeschnürten Stellen größer sein, als an den blau eingefärbten Stellen. Dies wird im nächsten Aufgabenteil untersucht.

6.3 Die Visualisierung der verteilten Daten aus TRACE

Die Daten aus TRACE liegen verteilt vor; deswegen wird das Modul `DistrData` als Daten-Modul aus der Menüleiste ausgewählt. Während der Instanzierung des DAVis-Moduls wird ein Netzwerk aufgebaut, das dem Netzwerk aus Abb. 5.8 ähnelt. Das zugehörige Daten-Panel enthält die Eingabestellen für die Parameter der Module `DAVisRead`, `DAVisDisplay`, und `DAVisEmbed`, wie sie im vorigen Kapitel bereits besprochen wurden. Die Module `DAVisRead` und `DAVisDisplay` werden dabei so wie in Abb. 6.6 parametrisiert.

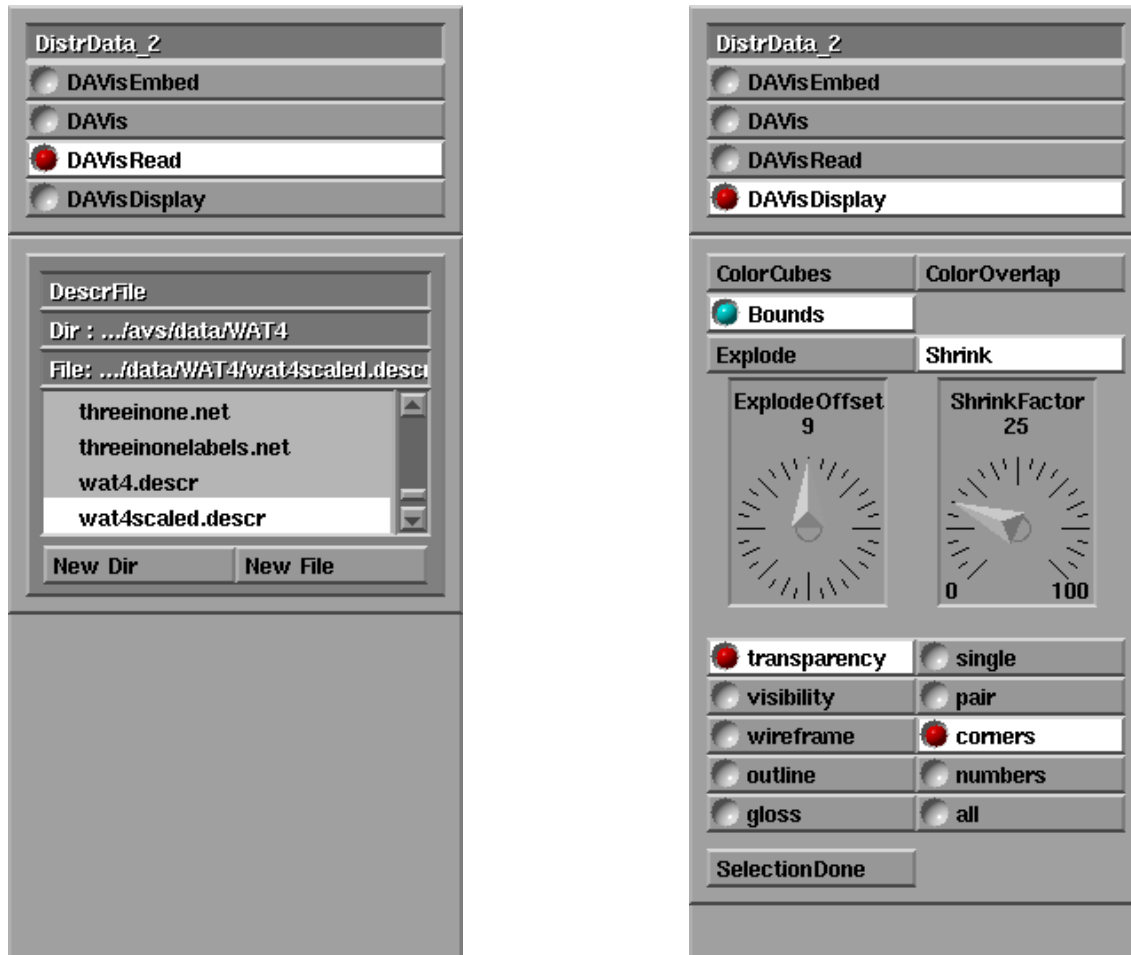


Abbildung 6.6: Parametrierung von `DAVisRead` und `DAVisDisplay`

Nun können diejenigen Domänen selektiert werden, die visualisiert werden sollen. Weil sich im Laufe der Sequenz die Konzentrationswolke durch das gesamte Gebiet bewegt, werden alle Domänen im Geometry Viewer selektiert, wie Abb. 6.7 zeigt.

`DAVisEmbed` gehört ebenfalls zum DAVis-Modul `DistrData`; es wird wie in Abb. 6.8 links dargestellt parametrisiert. Unter "New File" wird, ähnlich wie in `AnimAVSData`,

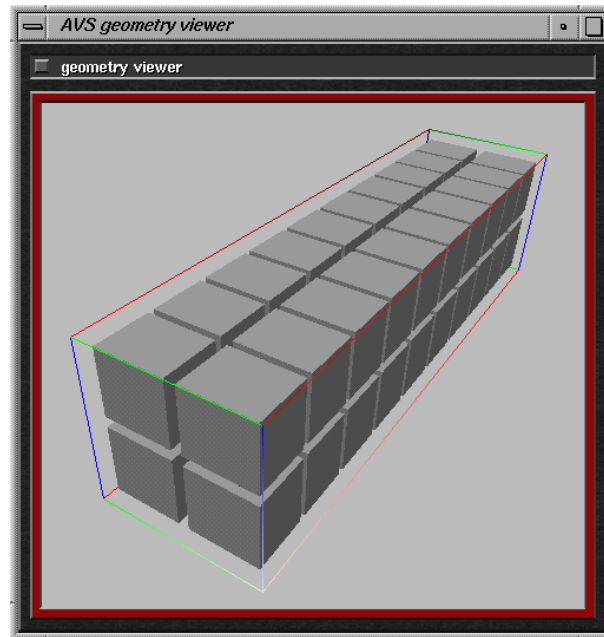


Abbildung 6.7: Selektion aller Domänen

der Stammname der einzelnen Feld-Partitionen angegeben, und unter "FileName-Pattern" das Format der Dateinamen. Der Platzhalter `%P` steht für die Partitionsnummer. Optional können auch hier Zeitschritte unter "TimeStep" festgelegt oder extern (zum Beispiel mit einem `animated integer`-Modul) vorgegeben werden; für diese Fälle kann der Platzhalter `%T` im Dateinamen verwendet werden. Wenn alle Domänen in `DAVisDisplay` selektiert worden sind, wird der Knopf "SelectionDone" betätigt; mit "NextPlease" in `DAVisEmbed` wird der Einlesevorgang gestartet. Nacheinander werden alle selektierten Partitionen in eine Feld-Datenstruktur eingelesen.

Als Modul zur Modellierung dient `DAVisModel`. Weil die Darstellung des Fließfeldes abstandsgetreu in die Darstellung eingeblendet werden soll, eignen sich nur die Modellerschritte "ovberlapped", "displaced" oder "compressed". Einer dieser Modellerschritte wird auf dem Modellpanel zu `DAVisModel` ausgewählt. Als View-Modul wird das `HedgeHog`-Modul eingesetzt; es erzeugt an einzelnen Stellen im Raum Vektorpfeile, die sich nach dem Betrag und der Richtung des Vektorfeldes an diesen Stellen richten. Das Modul wird so parametrisiert, wie die Abb. 6.8 rechts zeigt. Die Vektorpfeile können eingefärbt werden; am ehesten eignet sich dazu der Betrag des Vektorfeldes, ein Skalarfeld. Es bedarf damit eines Modellierungsschrittes, der aus dem Vektorfeld von `DistrData` das Betragsfeld bildet. Diesen Modellierungsschritt leistet das `DAVis`-Modul `VecMag`. Es wird instanziiert und der Ausgang von `DistrData_2` mit seinem Eingang verbunden; das Betragsfeld wird an den Farb-Eingang von `HedgeHog_2` weitergeleitet. Die Darstellung des Fließfeldes mit Vektorpfeilen, wie sie das View-Modul `HedgeHog` erzeugt, wird in den Viewer `StdVwr_0` geschaltet.

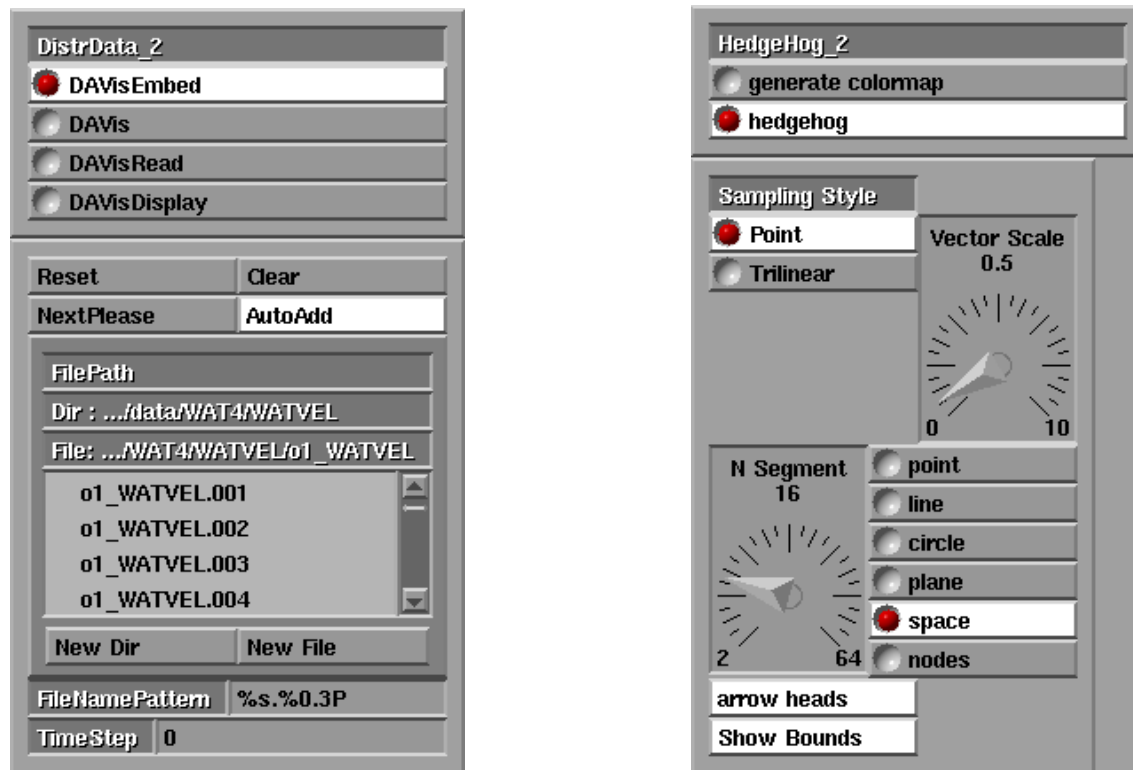


Abbildung 6.8: Parametrierung von DAVISEmbed und HedgeHog

Schematisch stellt sich das erstellte Netzwerk so wie in Abb. 6.9 dar. Es besteht aus insgesamt 11 DAVIS-Modulen, die zu fünf Visualisierungspipelines zusammengeschaltet worden sind. Die resultierende Visualisierung zeigt Abb. 6.10.

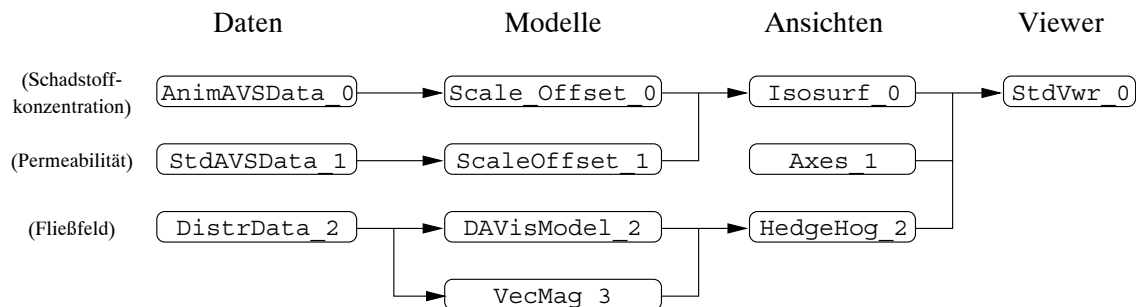


Abbildung 6.9: Erstelltes Netzwerk

Der Animation kann entnommen werden, daß dort, wo sich Einschnürungen in der Konzentrationswolke befinden, auch die Fließgeschwindigkeit des Wassers größer ist.

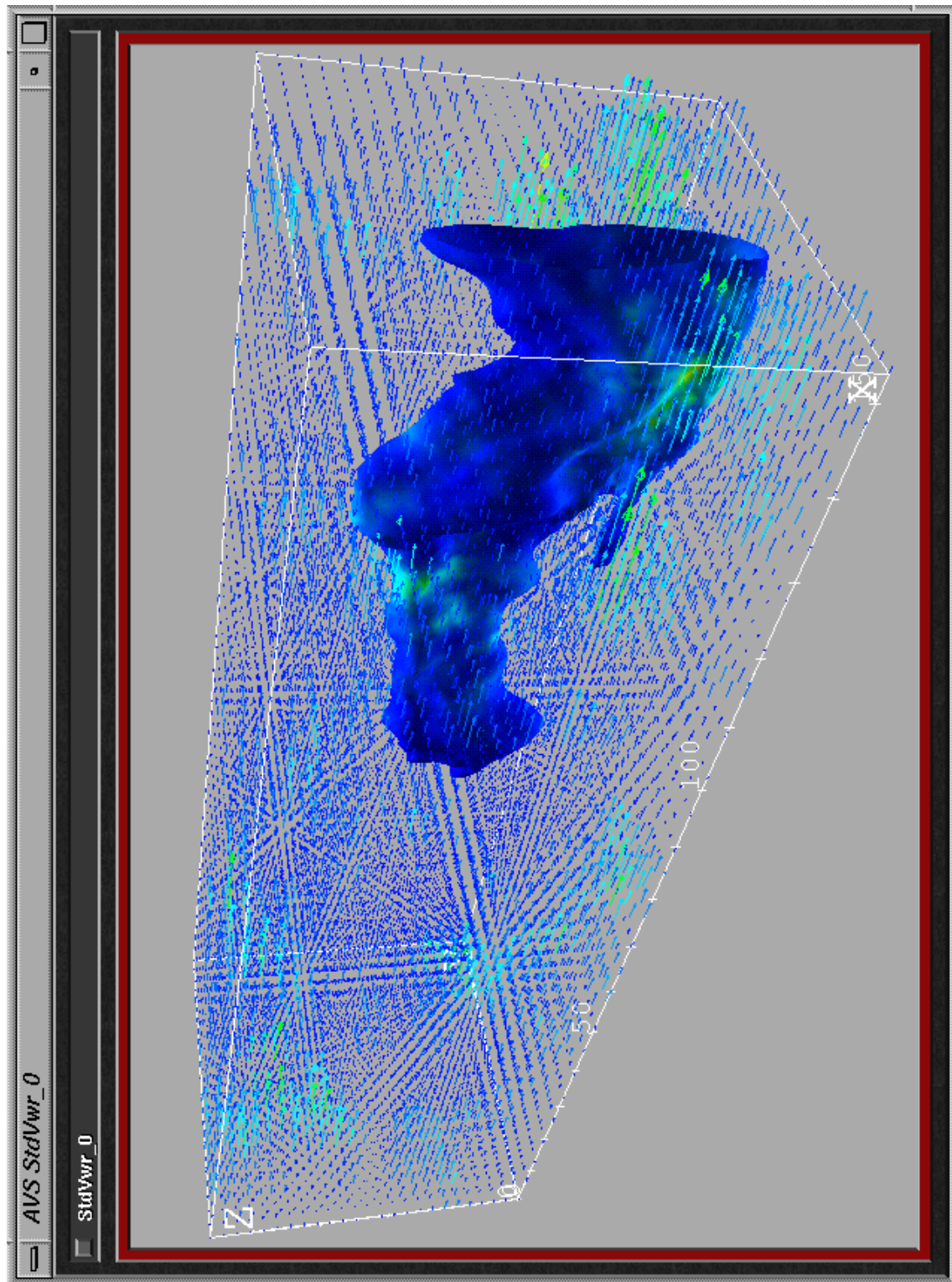


Abbildung 6.10: Visualisierung von Schadstoffkonzentration und Fließfeld

Kapitel 7

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein neues Konzept zur Visualisierung verteilter Daten vorgestellt, die zugehörigen Datenstrukturen und Methoden entwickelt und in einer auf AVS basierenden Implementation realisiert. Um die fünf Anforderungen an das gesuchte Visualisierungskonzept – Visualisierung des zugrundeliegenden Phänomens, der logischen Struktur, Möglichkeiten zur vergleichenden Visualisierung, hohe Interaktivität und Unabhängigkeit von der gewählten Implementationsplattform – zu erfüllen, mußten die bereits vorhandenen Ansätze und Werkzeuge der Visualisierung an die speziellen Belange der Visualisierung verteilter Daten angepaßt werden.

Die Grundlage des Konzepts bildet der modellbasierte Ansatz von Brodlie [KBro93]. Mit der Auffassung von Visualisierung als Darstellung von Modellfeldern, die aus den vorliegenden Daten abgeleitet werden, läßt sich die eigentliche Aufgabe der wissenschaftlichen Visualisierung, die Darstellung des zugrundeliegenden Phänomens, erfüllen. Ein idealisiertes Modellfeld, wie es im modellbasierten Ansatz vorgeschlagen wurde, läßt sich jedoch computergestützt nicht realisieren; deshalb mußte das ideale Modellfeld auf eine reale Feld-Datenstruktur abgebildet werden. Bei dieser Umsetzung half das Gittermodell von Hibbard [WHib94a], bei dem ideale mathematische Objekte auf reale Datenobjekte abgebildet werden. Beispielsweise entspricht die dem Modellfeld zugrundegelegte Funktion $Y = f(X)$ einem Datenfeld. Für die Zusammenfassung der verteilten Datenfelder in ein Gesamt-Datenfeld wurde weiterhin das Konzept der Einbettung in ein geordnetes Gitter benötigt; die Einbettung der verteilten Daten in eine Feld-Datenstruktur stellt sich hier als Grundvoraussetzung für die Visualisierung dar. Die anschließende Modellierung der Daten läßt sich dazu nutzen, eine eindeutige Abbildung $Y = f(X)$ von unabhängigen auf abhängige Variablen zu erstellen.

Um die Forderung nach einer vergleichenden Visualisierung zu erfüllen, müssen sich mehrere Ansichten zu einer Darstellung zusammenfassen lassen können. In dieser Beziehung mußte der modellbasierte Ansatz angepaßt werden. Die Forderungen nach Visualisierung der Struktur und hoher Interaktivität werden durch die entwickelten Methoden der DAVis-Suite erfüllt, namentlich durch die Methoden DAVisGenerate und DAVisDisplay, die Verfahren zur Selektion, Darstellung und Manipulation der Teilgebiete bereitstellen, und durch die Einbindung der Module der DAVis-Suite in

eine nach dem DAVis-Konzept aufgebaute Benutzeroberfläche. Das DAVis-Konzept wurde von Anfang an unabhängig von der jetzigen Implementationsplattform AVS erstellt; damit sollte die DAVis-Suite auch auf andere Visualisierungssysteme portiert werden können.

Die Festlegung auf eine spezielle Implementationsplattform (wie zum Beispiel AVS) bietet vielfältige Möglichkeiten, die DAVis-Suite weiter auszubauen, insbesondere wenn eine enge Kopplung an die Simulationsprogramme wie TRACE oder ParTrace gewünscht ist. So kann das Visualisierungssystem als eine interaktive Pre-Processing-Komponente für die Simulation genutzt werden (siehe dazu auch [SOrt93]). Weitere Entwicklungsmöglichkeiten liegen in der Erstellung spezifischer Module zur Erzeugung einer Ansicht, wie zum Beispiel eines Moduls, bei dem sich tatsächlich einzelne Partikel dynamisch in einem veränderlichen Fließfeld bewegen – bislang gibt es ein solches Particle-Tracking innerhalb der gewählten Visualisierungsumgebung AVS nur für stationäre Fließfelder.

Auch in Hinsicht auf die in Kapitel 5 vorgestellte Beispiel-Implementation ergeben sich einige Erweiterungsmöglichkeiten. Bislang wurde die Gebietszerlegung auf sehr spezielle Fälle beschränkt; spätere Versionen der DAVis-Suite sollten auch allgemeine Gebietszerlegungen in irregulären Gittern bearbeiten und visualisieren können. Wie die Entwicklung einer Datenstruktur zur Repräsentation der Partitions- und Domänengrenzen in Abschnitt 4.3.4 zeigte, wird es dabei günstiger sein, weniger die durch die Aufteilung entstandenen, einzelnen Teilgebiete zu beschreiben, als vielmehr die Aufteilung an sich. Weitere Ausbaumöglichkeiten ergeben sich in der vergleichenden Visualisierung, wie zum Beispiel beim Modul DAVisDiff, das in der Beispiel-Implementation bisher noch nicht realisiert wurde.

Weitere Ausbaumöglichkeiten ergeben sich in der vergleichenden Visualisierung, wie zum Beispiel beim Modul DAVisDiff, das in der Beispiel-Implementation noch nicht realisiert wurde. Denkbar wäre auch die Umstellung der Module der DAVis-Suite auf den AVS-Datentyp UCD. Dazu müßten allerdings die DAVis-Module intern umstrukturiert werden. Zudem wäre die Anzahl der darstellbaren Dimensionen auf drei festgelegt. Bei der Verwendung des UCD-Datentyps muß außerdem die Anforderung nach Unabhängigkeit von der Implementationsplattform fallengelassen werden.

Literaturverzeichnis

- [GAlm94] G. S. Almasi, A. Gottlieb, *Highly Parallel Computing*. 2. Aufl., The Benjamin/Cummings Publishing Company Inc., 1994
- [AVSU92] Advanced Visual Systems Inc., *AVS User's Guide*, Benutzerhandbuch zu AVS 5.0, Ausgabe 4, 1992
- [AVSD92] Advanced Visual Systems Inc., *AVS Developer's Guide*, Entwicklerhandbuch zu AVS 5.0, Ausgabe 4, 1992
- [AVSM93] Advanced Visual Systems Inc., *AVS Module Reference*, Referenzhandbuch zu den Modulen in AVS 5.0, Ausgabe 5, 1993
- [KBat90] K.-J. Bathe, *Finite-Elemente-Methoden*. Springer-Verlag, Berlin, 1990
- [SBar94] S. T. Barnard, H. D. Simon, *Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems*. Concurrency: Practice and Experience, Vol. 6 (2), pp. 101-117, John Wiley & Sons, Ltd., April 1994
- [PBjØ89] P. E. Bjørstad, *Multiplicative and Additive Schwarz Methods: Convergence in the Two-Domain Case*. T. F. Chan, R. Glowinski, J. Periaux, O. Widlund (editors), Domain Decomposition Methods, SIAM, Philadelphia.
- [KBro92] K. W. Brodlie *et al.*, *Scientific Visualization – Techniques and Applications*. Springer-Verlag, Berlin Heidelberg, 1992
- [KBro93] K. W. Brodlie, *A Classification Scheme for Scientific Visualization*. Animation and Scientific Visualizations, Tools & Applications, edited by R. A. Earnshaw and D. Watson, pp. 125-140, Academic Press Ltd., 1993
- [IBro89] I. N. Bronstein, K. A. Semendjajew, *Taschenbuch der Mathematik*. 24. Aufl., hrsg. von G. Grosche, V. Ziegler und D. Ziegler, Verlag Harri Deutsch, Thun und Frankfurt/Main, 1989
- [DBur88] D. S. Burnett, *Finite Element Analysis*. Addison-Wesley, Reading, 1988

- [RClo60] R. W. Clough, *The Finite Element Method in Plane Stress Analysis*. Proc. 2nd ASCE Conference on Electronic Computation, Pittsburgh, pp. 345-378, 1960
- [RCra95] R. A. Crawfis, *Vector Field Visualization*. SIGGRAPH '95, Advanced Techniques for Scientific Visualization, Course Notes, 1995
- [REar92] R. A. Earnshaw, *What is Scientific Visualization?* K. W. Brodlie *et al.*, Scientific Visualization – Techniques and Applications, Springer-Verlag, Berlin Heidelberg, 1992
- [JEnc93] J. L. Encarnacao *et al.*, *Advanced Research and Development Topics in Animation and Scientific Visualization*. R. A. Earnshaw *et al.*, Animation and Scientific Visualizations, Tools & Applications, pp. 55-59, Academic Press Ltd., 1993
- [JFol94] J. Foley, B. Ribarsky, *Next-Generation Data Visualization Tools*. L. Rosenblum *et al.*, Scientific Visualization – Advances and Challenges, pp. 103-127, Academic Press, San Diego, 1994
- [REar93] R. A. Earnshaw *et al.*, *Animation and Scientific Visualizations, Tools & Applications*. Academic Press Ltd., 1993
- [SBel96] S. Beliën, R. Leenders, *Comparison of Visualization Techniques and Packages*. Version 2.0, <http://www.sara.nl/Rik/REPORT.update>, 1996
- [CFet93] C. W. Fetter, *Contaminant Hydrology*. Macmillan Publishing Company, New York, 1993
- [RFre79] R. A. Freeze, J. A. Cherry, *Groundwater*. Prentice-Hall International, London, 1979
- [ISVAS] Th. Frühauf, M. Göbel, H. Haase, K. Karlsson, Fraunhofer Gesellschaft/IGD, *ISVAS – Interactive System for Visual data Analysis*. Public Domain Software, <ftp://ftp.igd.fhg.de>, 1995
- [MvGe80] M. Th. van Genuchten, *A Closed-Form Equation for Predicting the Hydraulic Conductivity of Unsaturated Soils*. Soil Science Society of America Journal, Vol. 44, pp. 892-898, 1980
- [RHab90] R. B. Haber, D. A. McNabb, *Visualization Idioms: A Conceptual Model for Scientific Visualization Systems*. B. Shriver, G. M. Nielson und L. J. Rosenblum (ed.), Visualization in Scientific Computing, IEEE Computer Society Press, pp. 74-93, 1990

- [SHac94] S. Th. Hackstadt, *Prototyping Advanced Parallel Program and Performance Visualizations*. Thesis for the degree of Master of Science, erhältlich über <http://www.cs.uoregon.edu/~hacks/research/papers/thesis/>, 1994
- [WHib94a] W. L. Hibbard, C. R. Dyer und B. E. Paul, *A Lattice Model for Data Display*. Proc. Visualization '94, pp. 310-317, 1994
- [WHib94b] W. L. Hibbard, C. R. Dyer, *A Lattice Theory of Data Display*. Tech. Report No. 1226, Computer Sciences Department, University of Wisconsin-Madison, 1994
- [BHöl84] B. Hölting, *Hydrogeologie*. 2. Aufl., Ferdinand Enke Verlag Stuttgart, 1984
- [FHoß92] F. Hoßfeld, *Supercomputer – Instrument und Gegenstand der Forschung*, Supercomputing in der KFA Jülich, J.-F. Hake, S. Knecht (Hrsg.), STE-Bericht, S. 81-90, Forschungszentrum Jülich, Jülich, 1992
- [PHuy86] P. S. Huyakorn, E. P. Springer, V. Guvanasen, T. D. Wadsworth, *A Three-Dimensional Finite-Element Model for Simulating Water Flow in Variably Saturated Porous Media*. Water Resources Research, Vol. 22 (13), pp. 1790-1808, 1986
- [BKer88] B. W. Kernighan, D. M. Ritchie, *Programmieren in C*. 2. Ausgabe (ANSI C), Carl Hanser Verlag, München, Wien; Prentice Hall International, London, 1988
- [Khoros] Khoros Research Inc., *Khoros*. Freier Zugriff u.a. über <ftp.khoros.com>, 1995
- [JMac86] J. Mackinlay, *Automating the Design of Graphical Presentations of Relational Information*. ACM Transactions on Graphics, Vol. 5 (2), pp. 110-141, 1986
- [BMcC87] B. H. McCormick, T. A. DeFanti und M. D. Brown, *Visualization in Scientific Computing*. ACM SIGGRAPH Computer Graphics, Vol. 21 (6), 1987
- [Meta95] Regionales Testbett in NRW: Metacomputing, siehe dazu <http://www.kfa-juelich.de/zam/RD/coop/metacomputer.html>, 1995
- [ONeu96] Olaf Neuendorf, *Numerische 3D-Simulation des Stofftransportes in einem heterogenen Aquifer*. Dissertation am Lehrstuhl für Geologie, Geochemie und Lagerstätten des Erdöls und der Kohle an der RWTH Aachen, bearbeitet am Institut für die Chemie der Geosphäre 4, Forschungszentrum Jülich, 1996 (Veröffentlichung in Vorbereitung)

- [SOrt93] S. Ortmanns, *Modelle der Kooperation von Graphik-Workstations und Höchstleistungsrechnern bei Visualisierungsproblemen*. Berichte des Forschungszentrums Jülich, Jül-2719, Jülich, 1993
- [JRed93] J. N. Reddy, *An Introduction to the Finite Element Method*. McGraw-Hill, Inc., 1993
- [LRic31] L. A. Richards, *Capillary Conduction of Liquids through Porous Media*. Physics 1, pp. 318-333, 1931
- [LRos94] L. Rosenblum *et al.*, *Scientific Visualization – Advances and Challenges*. Academic Press, San Diego, 1994
- [HSch90] H. A. Schwarz, *Über einen Grenzübergang durch alternierendes Verfahren*. Gesammelte mathematische Abhandlungen, Vol. 2, S. 133-143, Springer-Verlag, Berlin, 1890
- [RSed92] R. Sedgewick, *Algorithmen in C*. Addison-Wesley, Bonn, München, Paris, 1992
- [GSpo86] G. Sposito, *The Physics of Soil Water Physics*. Water Resource Research, Vol. 22 (9), pp. 83-88, 1986
- [CUps89] C. Upson, T. Faulhaber, D. Kamins, D. Schlegel, D. Laidlaw, J. Vroom, R. Gurwitz, und A. van Dam, *The Application Visualization System: a Computational Environment for Scientific Visualization*. IEEE Computer Graphics and Applications Vol. 9 (4), pp. 30-42, 1989
- [HVer93] H. Vereecken, G. Lindenmayr, A. Kuhr, D. H. Welte, A. Basermann, *Numerical Modelling of Field Scale Transport in Heterogenous Variably Saturated Porous Media*. Interner Bericht des Zentralinstituts für Angewandte Mathematik, Forschungszentrum Jülich, KFA-ZAM-IB-9301, 1993
- [HVer94] H. Vereecken, G. Lindenmayr, O. Neuendorf, U. Döring, R. Seidemann, *TRACE – A Mathematical Model for Reactive Transport in 3D Variably Saturated Porous Media*. Interner Bericht des Institut für die Chemie der Geosphäre 4, Forschungszentrum Jülich, KFA-ICG4-IB-501494, 1994
- [Vis5D] W. L. Hibbard, B. E. Paul, A. Battaiola, *Vis-5D*. Version 3.3, Public Domain Software, <ftp://iris.ssec.wisc.edu>, 1995
- [VMD] B. Humphrey, A. Dalke, J. Phillips, Theoretical Biophysics Group, Beckman Institute, Univ. of Illinois at Urbana-Champaign, und K. Hamer, Univ. of Illinois, sowie J. Leech, Univ. of North Carolina, *Visual Molecular Dynamics*. Public Domain Software, <ftp://ftp.ks.uiuc.edu>, 1995

- [RWim95] Ralf Wimmershoff, *Entwicklung und Implementierung einer dreidimensionalen Partitionierungsstrategie für das Programm TRACE auf einem massiv-parallelen Rechner*. Berichte des Forschungszentrums Jülich, Jül-3157, Jülich, 1995
- [GYeh87] G. T. Yeh, *3DFEMWATER: A Three Dimensional Finite Element Model of Water Flow through Saturated-Unsaturated Media*. Oak Ridge National Laboratory, Publication No. 2904, 1987

Anhang A

Bestimmung der Hülle um eine Gruppen-Domäne

Für das Modul DAVisDisplay wird ein Algorithmus benötigt, der es erlaubt, die Hülle um eine Gruppen-Domäne zu finden, die alle Einzel-Domänen der Gruppen-Domäne enthält. Als Beispiel sei die Gruppen-Domäne in Abb. A.1 betrachtet.

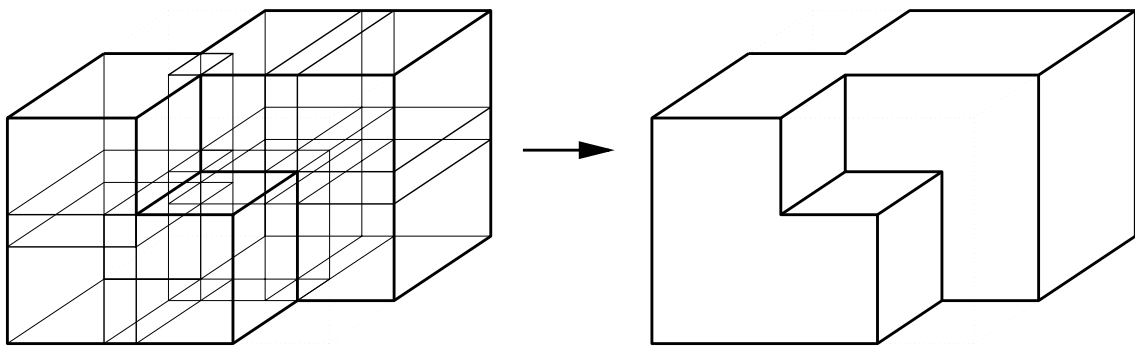


Abbildung A.1: Hülle um eine Gruppendomäne

Die dort gezeigte Gruppen-Domäne besteht aus sieben einzelnen Domänen einer Aufteilung von $3 \times 2 \times 2 = 12$ Domänen. Die äußere Hülle um die Gruppen-Domäne setzt sich aus 13 Seitenflächen zusammen, von denen fünf Flächen keine rechteckige Form besitzen.

Die Aufgabe "Finde eine dreidimensionale Hülle um die Gruppen-Domäne" kann nun gelöst werden, indem sie in mehrere kleine Teilaufgaben der Art "Finde eine Randlinie um die Gruppen-Domäne" zerlegt wird. Dazu sei Abb. A.2 betrachtet.

Die sieben einzelnen Domänen der Gruppen-Domäne wurden voneinander abgerückt, damit besser ersichtlich ist, aus welchen Eckpunkten die zweidimensionalen Randlinien bestehen (siehe Abb. A.2a). Außerdem wurden die Domänen durchnummeriert, wie es aus der $3 \times 2 \times 2$ -Aufteilung folgt. Zunächst wird die Aufgabe in

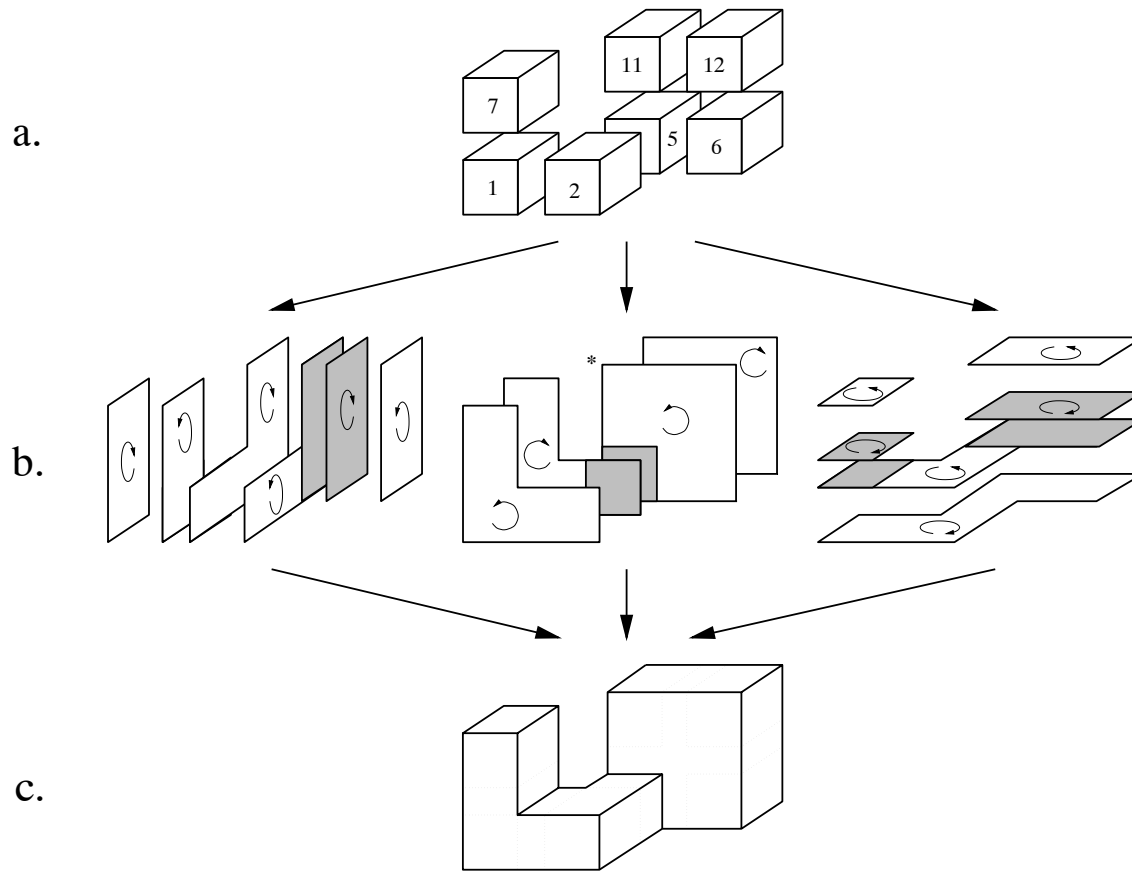


Abbildung A.2: Zerlegung des Problems

drei Teilaufgaben aufgeteilt, nämlich in die Aufgaben, alle Ränder gleicher Raumorientierung i zu finden (Abb. A.2b). Diese drei Teilaufgaben werden nochmals unterteilt in $2 \cdot P_i$ Teilaufgaben, bei denen nach den beiden Randlinien oberhalb und unterhalb einer "Domänenlage" gesucht wird. Die gefundenen Randlinien spannen Flächen auf, die die gesuchten Seitenflächen der Gruppen-Domäne bilden. Allerdings gehören die in Abb. A.2 schraffiert gekennzeichneten Anteile nicht zur Hülle der Gruppen-Domäne, siehe (Abb. A.2c). Das Problem ist gelöst, wenn ein Verfahren gefunden werden kann, wonach die zweidimensionalen Hüllen aus Abb. A.2b ohne die markierten Anteile erzeugt werden können.

Ein solches Verfahren existiert; es ist eine modifizierte Version des "Einwickel"-Algorithmus', wie er in [RSed92] zur Bestimmung einer konvexen Hülle um eine zweidimensionale Punktmenge benutzt wird. Dabei ist die konvexe Hülle um eine Punktmenge das kleinste konvexe Polygon, das alle Punkte der Menge enthält. Wenn das Polygon entgegen dem Uhrzeigersinn durchlaufen wird, sollen sich links davon alle Punkte der Menge befinden, und rechts davon keine. Beim Einwickel-Verfahren wird zunächst nach einem Punkt gesucht, der sicher zur konvexen Hülle

um die Punktmenge gehört. Dies ist zum Beispiel der Punkt S_0 mit der kleinsten y-Koordinate. Man nimmt nun eine in diesem Punkt beginnende, in positive x-Richtung verlaufende Halbgerade und "schwenke" sie nach oben, bis sie auf einen anderen Punkt S_1 der Punktmenge trifft. S_1 gehört ebenfalls zur konvexen Hülle, denn links der Halbgerade befinden sich alle Punkte der Menge, rechts davon aber keine. Der Punkt S_1 wird als neuer Drehpunkt einer Halbgeraden benutzt. Die in S_1 beginnende Halbgerade wird nun weiter so entgegen dem Uhrzeigersinn gedreht, bis ein weiterer Punkt der Punktmenge getroffen wird. Dieser Punkt S_2 gehört ebenfalls zur konvexen Hülle und dient als neuer Drehpunkt, usw. Das Verfahren wird solange fortgesetzt, bis der Startpunkt wieder erreicht ist und sich damit der gesuchte Polygonzug um die Punktmenge schließt, siehe Abb. A.3.

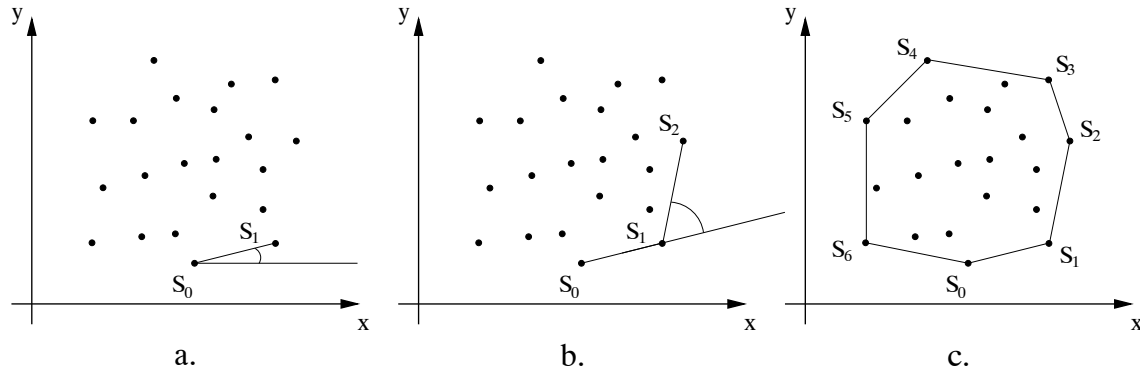


Abbildung A.3: Polygonzug um eine Punktmenge

Das "Schwenken" der Halbgeraden läuft auf eine Bestimmung von Winkeln zwischen einer Strecke und einer Halbgerade hinaus, wie in Abb. A.3 angedeutet ist. Im Prinzip muß die gesamte Punktmenge in jedem Zwischenschritt nach dem Punkt durchsucht werden, für den dieser Winkel gerade am kleinsten ist – ein Sortiervorgang, der den Zeitbedarf des "Einwickel"-Algorithmus bestimmt.

Für den Fall der Suche nach der Randlinie ober- oder unterhalb einer Domänenlage reduziert sich dieser Aufwand jedoch auf das Abprüfen von maximal drei benachbarten Referenzen pro Streckenabschnitt, denn die Referenzen bilden im Referenzraum ein streng reguläres Gitter. Für das Suchverfahren sind nur die 'Eck'-Referenzen der Einzel-Domänen, wie zum Beispiel die Referenzen R_0 , R_1 und R_2 der Einzel-Domäne 6 in Abb. A.4 wichtig, denn nur an diesen Eckpunkten kann sich die Richtung der Randlinie ändern. Die Abb. A.4 zeigt die zu prüfende Punktmenge für die in Abb. A.2b mit einem Stern markierte Randlinie. Diese Punktmenge liegt in einer r_1r_3 -Ebene des Referenzraumes. In der Abbildung sind die Punkte, die zur Gruppen-Domäne gehören, mit einem ausgefüllten Kreis ($\bullet$) markiert; die Punkte, die nicht zur Gruppen-Domäne gehören, mit einem kleinen Kreis ($\cdot$). Die in Abschnitt 4.3.4 besprochenen Leerstellen zwischen den Feld-Partitionen werden mit einem leeren Kreis ($\circ$) dargestellt.

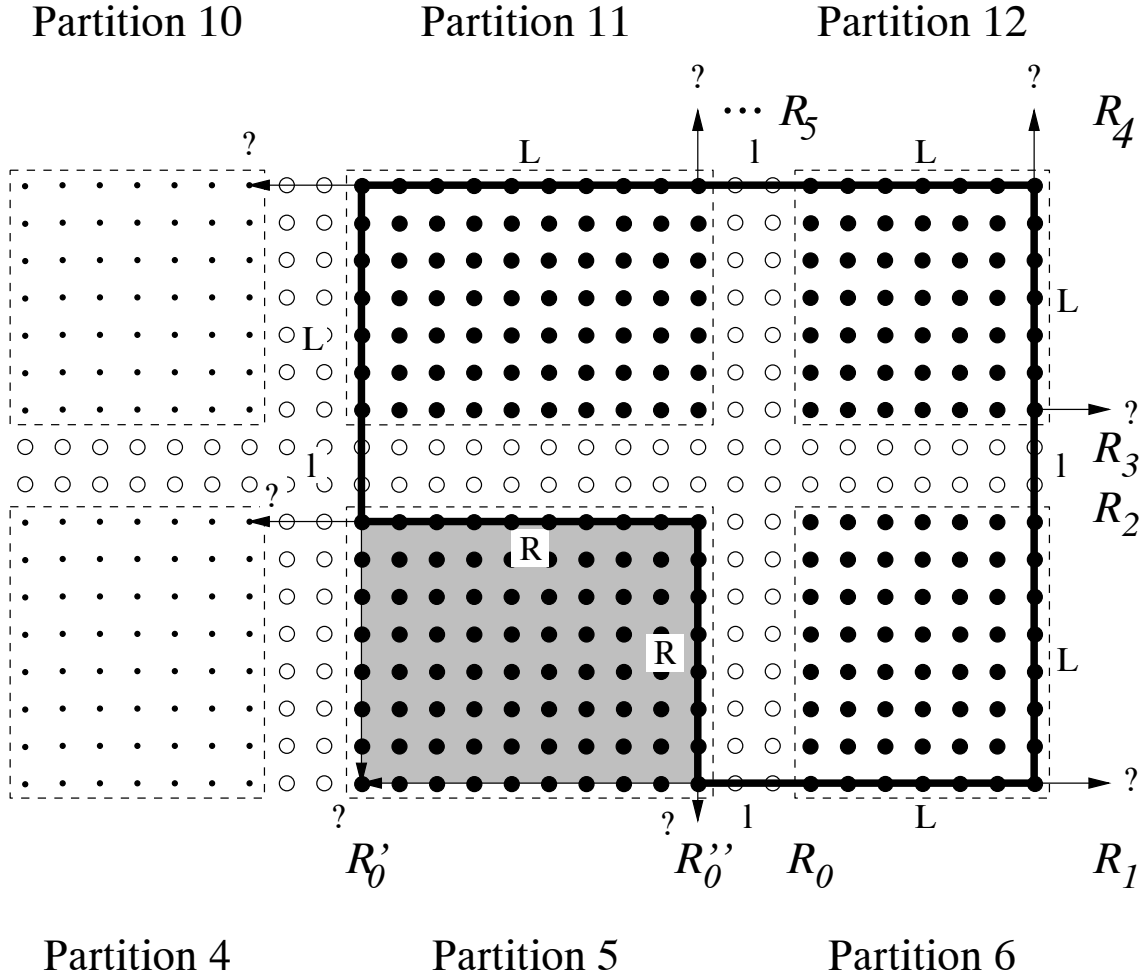


Abbildung A.4: Die zu prüfende Punktmenge für die Randlinie $*$ aus Abb. A.2

Zwischen den Eckpunkten verläuft die Randlinie immer in einer festen Richtung; so gehören von R_0 aus in positive r_1 -Richtung alle Referenzen incl. der Referenz R_1 zur Randlinie. Nun gibt es an jedem Eckpunkt nur drei mögliche Richtungen, in denen die Randlinie weiterlaufen kann: entweder sie biegt um 90° nach rechts ab, oder sie verläuft geradeaus weiter, oder sie biegt um 90° nach links ab. Sie kann die Orientierung nicht um 180° ändern, denn dann würde sie keine Referenzen konvex umschließen. Die Entscheidung an jedem Eckpunkt, in welche der drei Richtungen die Randlinie weiterverläuft, hängt nun von der Zugehörigkeit derjenigen Feld-Partitionen zur Gruppen-Domäne ab, die in der Nachbarschaft des nächsten Abschnittes der Randlinie liegen. Dazu lassen sich vier Fälle unterscheiden, die in Abb. A.5 schematisch im Referenzraum dargestellt sind. Die Feld-Partitionen, die nicht der Gruppen-Domäne zuzuordnen sind, sind darin mit einem kleinen ausgefüllten Kreis markiert, die zur Gruppen-Domäne zugehörigen Partitionen dagegen mit einem großen ausgefüllten Kreis. Der jeweils nächste Abschnitt der Randlinie wird als Pfeil zwischen zwei Eckpunkten markiert, denn die Randlinie wird in einem be-

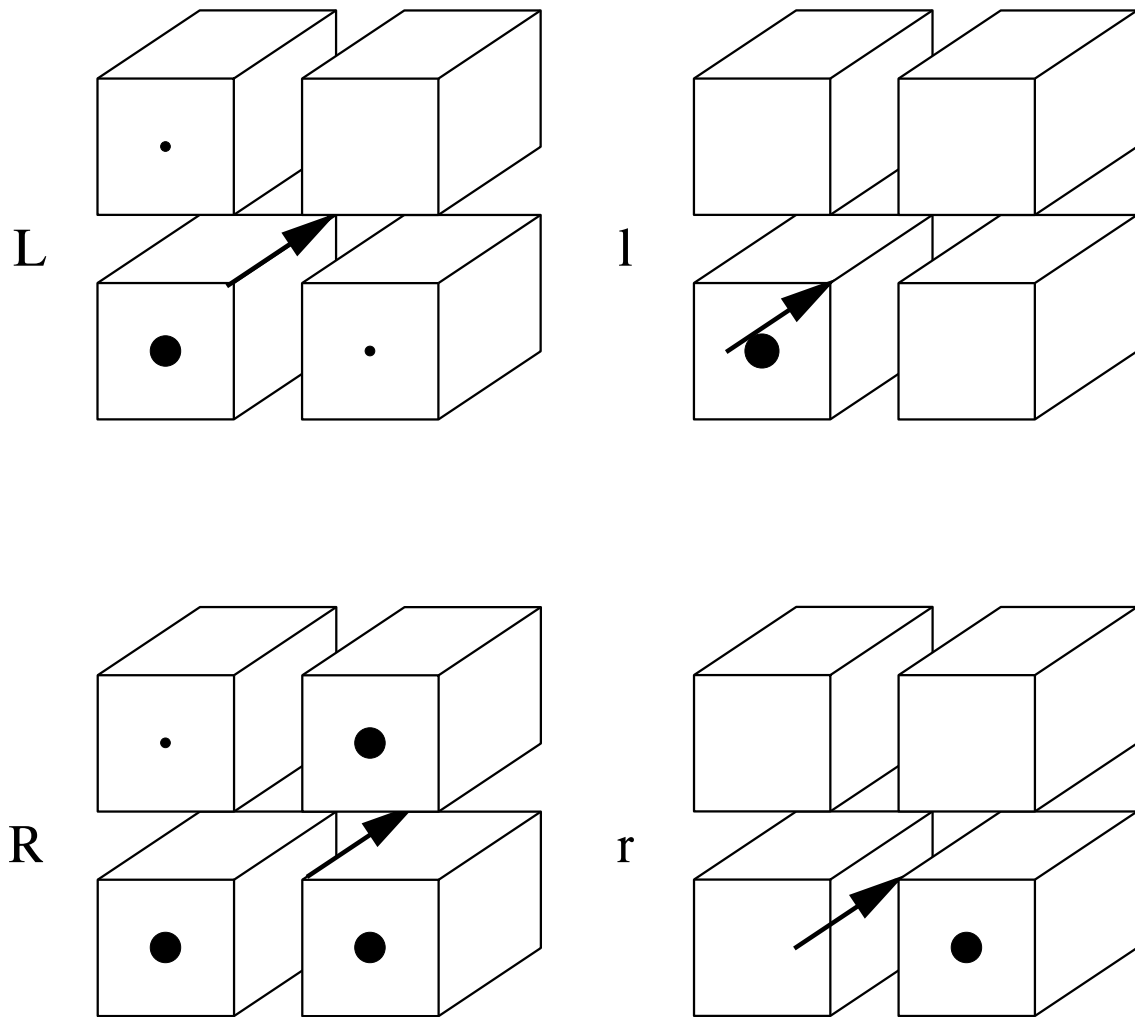


Abbildung A.5: Vier zu unterscheidende Fälle im Referenzraum

stimmt den Drehsinn durchlaufen, besitzt also eine Orientierung. Die vier Fälle werden wie folgt charakterisiert:

Fall 'L' : Der Randlinien-Abschnitt verläuft längs einer Kante einer Partition, die zur Gruppen-Domäne gehört. Diese Partition liegt "links unterhalb" der Strecke. Die Partition "links oberhalb", als auch die Partition "rechts unterhalb" des Abschnittes sollen nicht zur Gruppen-Domäne gehören.

Fall 'l' : Der Randlinien-Abschnitt "überquert" eine Trennmarkierung, d.h. die Leerstellen zwischen zwei Feld-Partitionen. Dabei soll die Partition "unterhalb der Randlinie, links voraus" zur Gruppen-Domäne gehören.

Fall 'R' : Der Abschnitt verläuft längs einer Kante einer Partition, die zur Gruppen-Domäne gehört. Diese Partition liegt jedoch, im Gegensatz zum Fall

'L', "rechts unterhalb" des Abschnitts. Die Partitionen "links unterhalb" sowie "rechts oberhalb" des Abschnitts sollen ebenfalls zur Gruppen-Domäne gehören. Dagegen gehört die Partition "rechts oberhalb" der Strecke nicht zur Gruppen-Domäne.

Fall 'r' : Der Abschnitt der Randlinie "überquert" wie im Fall 'l' eine Trennmарkierung. Anders als dort gehört jedoch die Partition "unterhalb der Randlinie, rechts voraus" zur Gruppen-Domäne.

Partitionen, auf die während der Überprüfung auf Zugehörigkeit zur Gruppen-Domäne nicht zugegriffen werden kann, weil sie außerhalb des Gesamtfeldes liegen würden, gelten als "nicht zugehörig zur Gruppen-Domäne".

An jedem Eckpunkt wird nun jeweils nach einem Schema geprüft, ob einer dieser vier Fälle erfüllt ist, wenn die Randlinie für den nächsten Streckenabschnitt (1.) nach rechts abbiegt, oder (2.) geradeaus weiterläuft oder (3.) nach links abbiegt. Wenn einer der Fälle erfüllt ist, wird entsprechend abgelenkt und erneut geprüft. Das Prüfschema lautet:

Beginne das Verfahren bei dem Punkt, der die kleinsten Komponenten $(r_{0,a}, r_{0,b})$ in der untersuchten Ebene besitzt, zur Gruppen-Domäne gehört, und für den für einen von dort beginnenden Abschnitt bis zum nächsten Eckpunkt der Fall 'L' gilt. Setze P auf diesen Eckpunkt und F auf den Fall 'L'.

Solange P nicht der Startpunkt ist, untersuche:

Wenn F ...

- ... der Fall 'L' ist, dann
 - lege von P aus den Abschnitt geradeaus voran und prüfe, ob für ihn Fall 'l' gilt; gilt er, so setze F auf 'l';
 - wenn 'l' nicht zutrifft, dann lege den nächsten Abschnitt stattdessen von P aus nach links und setze F auf 'L'.
- ... Fall 'l' ist, dann
 - lege den Abschnitt von P aus nach rechts und prüfe, ob für ihn Fall 'l' zutrifft; trifft er zu, so setze F auf 'l';
 - wenn 'l' nicht zutrifft, dann lege den Abschnitt stattdessen von P aus geradeaus voran und prüfe auf den Fall 'L'; trifft er zu, so setze F auf 'L';
 - wenn auch 'L' nicht zutrifft, dann lege den Abschnitt von P aus nach links und setze F auf 'R'.
- ... Fall 'R' ist, dann
 - lege den Abschnitt von P aus nach rechts und prüfe, ob für ihn Fall 'R' zutrifft; trifft er zu, so setze F auf 'R';

- wenn 'R' nicht zutrifft, dann lege den Abschnitt stattdessen von P aus geradeaus voran und prüfe auf den Fall 'r'; trifft er zu, so setze F auf 'r';
- wenn auch 'r' nicht zutrifft, dann lege den Abschnitt von P aus nach links und setze F auf 'l'.
- ... Fall 'r' ist, dann
 - lege den Abschnitt von P aus geradeaus voran und prüfe, ob für ihn Fall 'R' zutrifft, trifft er zu, so setze F auf 'R';
 - wenn 'R' nicht zutrifft, dann lege den Abschnitt stattdessen von P aus nach links und setze F auf 'r'.

setze P auf den Punkt, der sich an der Pfeilspitze des neuen Abschnitts befindet und prüfe, ob der Punkt P der Startpunkt ist.

Wie das Verfahren eine Randlinie findet, sei am Beispiel der mit einem Stern markierten Randlinie aus Abb. A.2b gezeigt. Zunächst wird der Punkt R'_0 untersucht, der Punkt mit den kleinsten Komponenten $(r'_{0,1}, r'_{0,3})$ in dieser $r_1 r_3$ -Ebene, der zur Gruppen-Domäne gehört. Auf ihn trifft jedoch für keine Richtung der Fall 'L' zu, denn die Domäne oberhalb der Ebene der Randlinie, die Domäne 2, gehört zur Gruppen-Domäne. Daher kann 'L' von R'_0 aus nicht erfüllt werden, aber auch nicht vom Eckpunkt R''_0 ; dieser Punkt kann auch nicht der Startpunkt sein. So wird R_0 betrachtet. Für diesen Punkt trifft der Fall 'L' zu, denn die Partition Nummer 3 gehört nicht zur Gruppen-Domäne, und die Partition "rechts unterhalb" des Abschnittes würde außerhalb des Gesamtfeldes liegen, wird also als "nicht zugehörig" zur Gruppen-Domäne gewertet. Der erste Punkt P ist also die Referenz R_1 . Von dort wird nun in positive r_1 -Richtung auf Fall 'l' geprüft. Die Partition "unterhalb des Randlinien-Abschnittes links voraus" würde außerhalb des Gesamt-Gebietes liegen, wird also als "nicht zugehörig zur Gruppen-Domäne" gewertet. Der nächste Abschnitt der Randlinie wird damit von R_1 aus nach links gelegt, also in positive r_3 -Richtung. Von R_2 aus trifft Fall 'l' zu, R_3 ist gefunden. Von R_3 aus wird zunächst auf 'l' in positive r_1 -Richtung geprüft. Dies trifft nicht zu, dafür aber Fall 'L' mit dem Punkt R_4 . Den weiteren Ablauf des Verfahrens läßt sich der Abb. A.4 entnehmen. Zu bemerken ist, daß die Randlinien in abwechselnden Orientierungssinn durchlaufen werden, so wie es in Abb. A.2b mit kleinen Pfeilen markiert ist. Diesem Orientierungssinn entsprechend wechselt auch, welche Domänen sich oberhalb und welche sich unterhalb der Ebene der Randlinie befinden.

Mit diesem Verfahren werden alle weiteren Randlinien als Polygonzüge um die Gruppendomäne gefunden. Die Polygonzüge spannen Flächen auf, die die Seitenflächen der Gruppen-Domäne bilden, wie Abb. A.2c zeigt.

Danksagung

Die vorliegende Arbeit wurde als Diplomarbeit im Fach Elektrotechnik am Lehrstuhl für Technische Informatik und Computerwissenschaften der Rheinisch-Westfälischen Technischen Hochschule (RWTH) Aachen erstellt. Dem Lehrstuhlinhaber Herrn Prof. Dr. F. Hoßfeld danke ich für die Möglichkeit, diese Arbeit am Zentralinstitut für Angewandte Mathematik (ZAM) des Forschungszentrums Jülich (KFA) anfertigen zu können.

Mein besonderer Dank gilt Herrn Dietrich Bartel für die Betreuung der Arbeit und für seine zahlreichen konstruktiven Anregungen, Diskussionen und Hinweise, die zum Gelingen dieser Arbeit beigetragen haben.

Ebenfalls danken möchte ich Herrn Dr. H. Vereecken und seinen MitarbeiterInnen am ICG-4 für die Bereitstellung der Beispieldaten und ihre ständige Hilfsbereitschaft. Auch die gute Arbeitsatmosphäre unter den Studenten und Doktoranden am ZAM möchte ich an dieser Stelle hervorheben.

Für die ständige Unterstützung während der Zeit des Studiums danke ich meinen Eltern und meinen Geschwistern sowie meinen Freunden.