Parallele Netzgenerierung für ebene und räumliche Problemstellungen aus dem Bauwesen

Michael Burghardt

Technische Universität Darmstadt
Institut für Numerische Methoden
und Informatik im Bauwesen

Stand: 18. August 2000

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Problemstellung 1

1.2 Zielsetzung 2

1.3 Aufbau der Arbeit 2

2 Stand der Forschung **5**

2.1 Netzgenerierung 5

2.1.1 Strukturierte Netze 5

2.1.2 Unstrukturierte Netze 8

2.1.3 Hybride Netze 8

2.2 Finite-Elemente-Methode 9

2.2.1 Allgemeines Vorgehen 10

2.2.2 Bedingungen für die Netzgenerierung 11

2.3 Gebietszerlegungsverfahren 18

2.4 Partitionierungsalgorithmen 20

2.4.1 Geometrische Algorithmen 20

2.4.2 Graphenbasierte Algorithmen 22

2.4.3 Algebraische Algorithmen 23

2.5 Parallele Netzgenerierung 24

2.6 Parallelisierung 26

2.6.1 Motivation 26

2.6.2 Klassifikation von Rechnern 27

2.6.3 Parallelisierungsparadigma 32

2.6.4 Message-Passing 32

2.6.5 Daten-Parallele-Programmierung 36

2.6.6 Leistungsdaten 40

3 Netzgenerierung **41**

3.1 Unstrukturierte Netze 41

3.1.1 Netzbasierte Methode 42

3.1.2 Advancing-Front-Methode . . . 42
3.1.3 Delaunay-Triangulierung . . . 44
3.1.4 Generierung von Viereckelementen . . . 52
3.1.5 Generierung von Vierecken aus Dreiecken . . . 54
3.1.6 Generierung von Hexaederelementen . . . 55
3.2 Glättung und Netzverbesserungen . . . 56
3.2.1 Glättung . . . 56
3.2.2 Patchweises Verbessern . . . 57
3.3 Netzverfeinerung . . . 59
3.3.1 Dreieckelemente . . . 59
3.3.2 Viereckelemente . . . 59
3.3.3 Tetraederelemente . . . 62
3.3.4 Hexaederelemente . . . 63
3.4 Qualitätsbeschreibung . . . 63
3.4.1 Knotenorientierte Kriterien . . . 65
3.4.2 Elementorientierte Kriterien . . . 66
3.4.3 Bewertung der Kriterien . . . 71

4 Parallele Netzgenerierung 73
4.1 Vorgehensweise . . . 73
4.2 Beschreibung der Geometrie . . . 74
4.3 Teilung der Geometrie . . . 75
4.4 Geometrieverarbeitung . . . 76
4.5 Lastverteilungsfunktion . . . 78
4.6 Beispiel . . . 80
4.6.1 Laufzeitverhalten . . . 81
4.6.2 Netzqualität . . . 84

5 Netzgenerierung geotechnischer Systeme 85
5.1 Motivation . . . 85
5.2 Projektionstechnik . . . 85
5.3 Verbessertes Erzeugen der Bauteil-Elemente . . . 87
5.3.1 Wände . . . 88
5.3.2 Pfähle . . . 89
5.4 Schneiden von Bauteilen . . . 90
5.4.1 Wände . . . 90
5.4.2 Pfähle . . . 91
5.5 Randbedingungen . . . 91
5.5.1 Natürliche Randbedingungen . . . 92

5.5.2 Wesentliche Randbedingungen 94
5.6 Kontaktproblematik 94
5.6.1 Pfähle 94
5.6.2 Kontaktflächen 95

6 Objektorientierte Modellierung **97**
6.1 CAD-Modell 97
6.1.1 Baugrund-Modell 97
6.1.2 Bauzustandsmodell 98
6.1.3 Konstruktionselemente-Modell 98
6.2 Geometrie-Modell 99
6.3 Finite-Elemente-Modell 101
6.4 Geometrisch sortierte Datenstrukturen 102
6.5 Paralleler Ablauf 106
6.6 Implementierung 110

7 Anwendungsbeispiele **111**
7.1 Hochbauplatte - Liederhalle Stuttgart 111
7.1.1 Diskretisierung 1 112
7.1.2 Diskretisierung 2 114
7.1.3 Effektivität der Parallelisierung 116
7.2 Gründung - TREPTOWERS Berlin 117
7.2.1 Bauablauf 118
7.2.2 Parallele Netzgenerierung 122
7.2.3 Effektivität der Parallelisierung 124

8 Zusammenfassung und Ausblick **127**
8.1 Durchgeführte Arbeiten und Ergebnisse 127
8.2 Möglichkeiten für die Weiterentwicklung 128

Abbildungsverzeichnis

2.1 *Netzarten* 5
2.2 *Abbildung des strukturierten Netzes* 6
2.3 *Abbildung auf ein Einheitsviereck* 6
2.4 *Multiblock: O-Typ Netz* 8
2.5 *Multiblock: C-Typ Netz* 8
2.6 *Multiblock: H-Typ Netz* 8
2.7 *Randbedingungen* 14
2.8 *Fehlerindikator: Zienkiewicz/Zhu* 15
2.9 *Neugenerierung* 17
2.10 *h–Version* 17
2.11 *p–Version* 17
2.12 *r–Version* 18
2.13 *Implizite Partitionierung* 18
2.14 *Substruktur-Technik* 19
2.15 *Koordinaten-Bisektion* 21
2.16 *Hauptachsen Bisektion* 22
2.17 *Greedy-Algorithmus* 22
2.18 *Spektral-Bisektion* 24
2.19 *Spektral-Bisektion* 24
2.20 *Parallele Netzgenerierung* 25
2.21 *von-Neumann-Rechner* 27
2.22 *SISD Rechner* 29
2.23 *SIMD Rechner mit Distributed Memory* 30
2.24 *SIMD Rechner mit Shared Memory* 30
2.25 *MIMD Rechner mit Shared Memory* 31
2.26 *MIMD Rechner mit Distributed Memory* 31
2.27 *Architektur von RMI* 38
2.28 *Name-Service von CORBA [Inprise 1999]* 39

3.1 *Netzbasierte Methode* 42

3.2 *Advancing-Front-Methode* . . . 43
3.3 *Problematische Hexaedervernetzung* . . . 44
3.4 *Delaunay-Triangulierung* . . . 45
3.5 *Recursive-Split-Algorithmus* . . . 46
3.6 *Radial-Sweept-Algorithmus* . . . 47
3.7 *Step-by-Step-Algorithmus* . . . 48
3.8 *Modified-hierarchical-Algorithmus* . . . 48
3.9 *Incremental-Algorithmus* . . . 49
3.10 *Watson's Algorithmus* . . . 50
3.11 *Einfügen von Punkten* . . . 51
3.12 *Ruppert's Algorithmus* . . . 52
3.13 *Viereckelemente: lokale Koordinaten* . . . 52
3.14 *Viereckelemente: Punktmengen* . . . 53
3.15 *Viereckelemente: einzelne Punkte* . . . 53
3.16 *Generierung der Vierecke aus Dreiecken* . . . 54
3.17 *Patchweises Verbessern* . . . 57
3.18 *Patchweises Verbessern* . . . 58
3.19 *Übergangsnetze zur Verfeinerung von Dreiecknetzen* . . . 59
3.20 *Übergangsnetze der zweier-Verfeinerung von Vierecken* . . . 60
3.21 *Übergangsnetze der dreier-Verfeinerung von Vierecken* . . . 60
3.22 *Lokale Verfeinerung von Netzen* . . . 61
3.23 *Verfeinerung Viereckelemente* . . . 61
3.24 *Rekursive Verfeinerung* . . . 62
3.25 *Tetraeder* . . . 62
3.26 *Tetraeder* . . . 63
3.27 *Tetraeder* . . . 64
3.28 *Topologie-Kriterium* . . . 66
3.29 *Umkreisradius-Kriterium* . . . 66
3.30 *Umkreisradius-Kriterium* . . . 67
3.31 *Form-Kriterium* . . . 68
3.32 *Form-Kriterium* . . . 69
3.33 *Geometrie-Kriterium* . . . 69
3.34 *Gemischter Indikator* . . . 70

4.1 *Rekursives Teilen mit der Schwerachsenmethode* . . . 74
4.2 *Mehrere Polygonzüge auf einem Prozessor nach Teilung* . . . 75
4.3 *Teilung einer Geometrie* . . . 76
4.4 *ϵ-Umgebung eines Punktes P* . . . 77
4.5 *ϵ-Umgebung einer Linie* . . . 77

4.6 *Überprüfung Punkt in Polygon* . . . 78
4.7 *Lastverteilungsfunktion von Verfeinerungen* . . . 79
4.8 *FE-Netz mit Verfeinerungen* . . . 79
4.9 *Diskretisiertes Netz* . . . 80
4.10 *Partitioniertes Netz* . . . 80
4.11 *Partitionierung* . . . 81
4.12 *Zeiten* . . . 82
4.13 *Speed-up* . . . 83
4.14 *Effizienz* . . . 83
4.15 *Form-Kriterium* . . . 84
4.16 *Form-Kriterium* . . . 84

5.1 *Projektionstechnik* . . . 86
5.2 *Projektionstechnik* . . . 86
5.3 *Hexaeder-Elemente-Netz* . . . 87
5.4 *Hexaeder-Elemente-Netz* . . . 87
5.5 *Patchweises Verbessern* . . . 88
5.6 *Vergrößertes Wandloch in Dreiecknetz* . . . 88
5.7 *Verschneiden der Wände* . . . 89
5.8 *Pfähle* . . . 89
5.9 *Schnitt durch Wand* . . . 90
5.10 *Finite-Elemente-Netz mit Schnitt durch Wand* . . . 91
5.11 *Schnitt durch Pfahl* . . . 91
5.12 *Freiheitsgrade eines Hexaederelementes* . . . 92
5.13 *Lineare Formfunktion* . . . 92
5.14 *Quadratische Formfunktion* . . . 93
5.15 *Lineare Formfunktion* . . . 93
5.16 *Quadratische Formfunktionen* . . . 93
5.17 *Kraftübertragung am Pfahl* . . . 94
5.18 *Kontaktfläche am Pfahl* . . . 95
5.19 *Geschnittener Pfahl* . . . 96

6.1 *Klassendiagramm Baugrund-Modell* . . . 97
6.2 *Klassendiagramm Bauzustands-Modell* . . . 98
6.3 *Klassendiagramm Konstruktionselemente-Modell* . . . 99
6.4 *Klassendiagramm Geometrie* . . . 100
6.5 *Klassendiagramm Finite-Elemente-Modell* . . . 101
6.6 *Quadtree-Struktur* . . . 102
6.7 *Netze zum Vergleichen der Zugriffszeiten* . . . 103

6.8 *Zeiten zum Suchen in der Datenstruktur* . . . 104
6.9 *Zeiten zum Suchen in der Datenstruktur* . . . 105
6.10 *Zeiten zum Suchen in der Datenstruktur* . . . 105
6.11 *Zeiten zum Suchen in der Datenstruktur* . . . 106
6.12 *Schematische Darstellung des parallelen Ablaufes* . . . 107
6.13 *Zustandsdiagramm* . . . 109

7.1 *Liederhalle: Grundriss der Platte* . . . 111
7.2 *Liederhalle: Elementnetz Diskretisierung 1* . . . 112
7.3 *Liederhalle: Partitionierung mit vier Prozessoren* . . . 113
7.4 *Liederhalle: Partitionierung mit acht Prozessoren* . . . 114
7.5 *Liederhalle: Elementnetz Diskretisierung 2* . . . 115
7.6 *Liederhalle: Speed-up* . . . 116
7.7 *Liederhalle: Effizienz* . . . 116
7.8 *TREPTOWERS: Schnitt und Grundriss* . . . 117
7.9 *TREPTOWERS: Modellierung mit GTIS* . . . 118
7.10 *TREPTOWERS: Bauzustand nach dem Einbringen der Spundwände* . . . 119
7.11 *TREPTOWERS: Bauzustand nach Aushub* . . . 120
7.12 *TREPTOWERS: Bauzustand nach Fertigstellen der Gründung* . . . 120
7.13 *TREPTOWERS: Schnitt durch die Baugrube* . . . 121
7.14 *TREPTOWERS: Finite-Elemente-Netz der Gründung* . . . 121
7.15 *TREPTOWERS: Netz eines Prozessors* . . . 122
7.16 *TREPTOWERS: Netz eines Prozessors* . . . 122
7.17 *TREPTOWERS: Partitionierung* . . . 123
7.18 *TREPTOWERS: Speed-up* . . . 124
7.19 *TREPTOWERS: Effizienz* . . . 124

Tabellenverzeichnis

2.1 *Reguläre Punkte* 6

4.1 *Elementanzahlen der Netze der Diskretisierung 1* 82
4.2 *Elementanzahlen der Netze der Diskretisierung 2* 82

7.1 *Liederhalle: Elementanzahlen der Netze der Diskretisierung 1* 113
7.2 *Liederhalle: Elementanzahlen der Netze der Diskretisierung 2* 115
7.3 *TREPTOWERS: Elementanzahlen* 123

1 Einleitung

1.1 Problemstellung

Aktuelle Ingenieurbauwerke sind auf Grund ihrer Komplexität mit den klassischen manuellen und vereinfachten Berechnungsmodellen nur eingeschränkt abbildbar. Neben dem physikalischen Experiment hat sich die numerische Simulation in den letzten Jahren als gutes Werkzeug für den Ingenieur bei dem Entwurf und der Bemessung von Ingenieurbauwerken entwickelt. Neben der exakteren Bemessung von einfachen Bauteilen wie Scheiben und Platten, die eine Kosteneinsparung in der Ausführung bewirkt, hat sich die numerische Berechnung gerade im Bereich komplexerer Aufgabenstellungen als vorteilhaft erwiesen.

Einer der wichtigsten Vertreter der numerischen Approximationsverfahren ist die Methode der finiten Elemente. Mit Hilfe dieser Methode werden die verschiedensten Problemstellungen aus dem Bauwesen, wie die Schnittgrößenermittlung in der Statik, die Berechnung von Grundwasserspiegelhöhen, die Simulation des Stofftransports im Grundwasser oder die Setzungsberechnung von Gebäuden durchgeführt.

Der herkömmliche Ablauf bei der Anwendung der Finite-Elemente-Methode gliedert sich in drei Teile: die Erstellung eines Modells, die Simulationsrechnung und die Auswertung der Berechnungsergebnisse. Die Erstellung des Modells unterteilt sich wiederum in mehrere Teile. Ausgegangen wird von einer Beschreibung des Problems in Form von Zeichnungen, CAD-Daten, Messgrößen oder Sondierungsdaten. Bei kontinuierlichen Problemen ist eine Diskretisierung des Systems, d.h. eine Aufteilung des Kontinuums in kleine Elemente, erforderlich. Bei einfachen Problemstellungen ist eine manuelle Diskretisierung möglich, während bei komplexeren dreidimensionalen und möglicherweise zeitabhängigen Systemen eine automatische Vernetzung erforderlich ist.

Der numerische Aufwand in der Simulationsrechnung von großen Finite-Elemente-Modellen übersteigt häufig die Rechenleistung sowie den adressierbaren Speicher herkömmlicher Rechner. Spezielle Parallelrechner sowie zu einem virtuellen Großrechner zusammengeschaltete Arbeitsplatzrechner bieten hier Abhilfe. Die parallelisierte Simulation erfordert eine Aufteilung des Problems in gekoppelte Teilprobleme, die dann verteilt gleichzeitig berechnet werden. Bei der Finite-Elemente-Berechnung ist hierfür eine Partitionierung des FE-Netzes erforderlich, die in der herkömmlichen Vorgehensweise im Anschluss an die Erzeugung des Netzes und vor der Berechnung durchgeführt wird.

1.2 Zielsetzung

Für die Durchführung von Finite-Elemente-Berechnungen auf Parallelrechnern ist ein partitioniertes Netz notwendig. Die Teilnetze dürfen sich nicht überlappen und müssen an den Kanten kompatibel sein. Kompatibel bedeutet hier, dass zwei Teilgebiete, die sich an einer Koppelkante treffen, auf der Kante gleich viele Knoten und Elemente besitzen. Die Elemente müssen hierbei auch mit den gleichen Ansatzfunktionen beschrieben werden.

Bei der Diskretisierung großer Problemstellungen stellen der zur Verfügung stehende sowie der adressierbare Hauptspeicher eine Grenzen dar. Diskretisierungen mit einer größeren Anzahl von Elementen sind hier nicht möglich. Bei der sequentiellen Generierung von Netzen, die anschließend partitioniert werden, besteht dieses Problem ebenfalls. Trotz der parallelen FE-Simulation und des größeren Speichers ist die Beherrschung größerer Probleme mit dieser Vorgehensweise nicht möglich.

In der hier vorliegenden Arbeit wird ein Ansatz zur parallelen Generierung von FE-Netzen vorgestellt, mit dem automatisch partitionierte Netze erzeugt werden. Alternativ zur herkömmlichen Vorgehensweise wird bei diesem Ansatz nicht ein komplettes Netz erzeugt und später aufgeteilt, sondern es wird eine Partitionierung der Beschreibung des Problems vorgenommen und anschließend eine kompatible Vernetzung der Teilprobleme durchgeführt. Hierdurch erreicht man, dass die Diskretisierung nie auf einem einzelnen Prozessor komplett vorliegen muss und so auch wesentlich größere Problemstellungen beherrschbar werden.

Die Anwendbarkeit der parallelen Netzgenerierung wird sowohl für zweidimensionale ebene Tragwerke als auch für räumliche und zeitabhängige Baugrund-Tragwerk-Strukturen untersucht.

Für die Diskretisierung von Baugrund-Tragwerk-Strukturen existieren zur Zeit keine umfassenden Software-Lösungen. Im Rahmen dieser Arbeit wird daher nach Möglichkeiten zur Vernetzung der Baugrundstruktur sowie der darin enthaltenen Gründungs- und Verbauelementen gesucht. Hierbei werden sowohl spezielle Anforderungen aus der Finite-Elemente-Modellierung des Systems als auch Anforderungen aus der Parallelisierung des Netzgenerators berücksichtigt.

1.3 Aufbau der Arbeit

Kapitel 2 ist eine Zusammenstellung des Standes der Forschung und dient zur Einordnung der Problemstellung. Hierbei wird näher auf die *Netzgenerierung*, die *Finite-Elemente-Methode*, die *Gebietszerlegungsverfahren*, die *parallele Netzgenerierung* und allgemein auf die *Parallelisierung* von Problemstellungen eingegangen.

In **Kapitel 3** wird die Vorgehensweise zur Generierung von Netzen erläutert. Hierbei werden sowohl Algorithmen zur *Netzgenerierung* von Dreieck-, Viereck-, Tetraeder- und Hexaedernetzen vorgestellt als auch die Möglichkeiten zur *Verbesserung von Diskretisierungen* dargestellt. Algo-

rithmen zur *Netzverfeinerung* werden erläutert und eine geometrische *Qualitätsbeschreibung* der Diskretisierungen wird untersucht.

Der Ansatz zur parallelen Netzgenerierung wird in **Kapitel 4** vorgestellt. Hierbei wird auf die notwendige *Darstellung der Geometrie*, die *Teilung des Problems*, die *Geometrieverarbeitung* und auf *Lastverteilung* eingegangen.

Kapitel 5 dient der Erläuterung der Netzgenerierung für geotechnische Systeme. Detailliert wird hierbei auf die *Projektionstechnik*, den *Einbau von Bauteil-Elementen* und auf *Rand- und Übergangsbedingungen* eingegangen. Beim *Einbau von Bauteil-Elementen* werden spezielle Lösungen für die Parallelisierung vorgestellt.

Auf die Objektorientierte Modellierung des Systems wird in **Kapitel 6** eingegangen. Hierbei wir das Augenmerk auf die Teilmodelle *CAD-Modell*, *Geometrie-Modell* und *Finite-Elemente-Modell* gelegt. *Geometrisch sortierte Datenstrukturen* werden für den Anwendungsfall Baugrund-Tragwerk-Struktur untersucht und die Implementierung des Systems wird erläutert.

In **Kapitel 7** wird die Leistungsfähigkeit der parallelen Netzgenerierung an zwei Beispielen demonstriert: Das eine Beispiel ist eine zweidimensionale *Hochbauplatte mit eine großen Aussparung* der Liederhalle in Stuttgart. Das anderen Anwendungsbeispiel ist die räumliche *Diskretisierung der Gründung und des Baugrubenverbaus* des TREPTOWERS in Berlin.

Kapitel 8 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf Möglichkeiten der Weiterführung der Arbeit.

2 Stand der Forschung

2.1 Netzgenerierung

Die Beschreibung vieler physikalischer Phänomene ist mit Hilfe von Differentialgleichungen möglich, die in der Regel nur für Sonderfälle exakt gelöst werden können. Zur numerischen Lösung dieser Gleichungen gibt es eine Reihe von Verfahren wie z.B. die Finite-Differenzen-Methode, die Rand-Elemente-Methode oder die Finite-Elemente-Methode. Bei kontinuierlichen Systemen ist eine Diskretisierung des Systems zur Anwendung dieser Methoden erforderlich. Diskretisierungen lassen sich in drei Arten klassifizieren: strukturierte Netze, unstrukturierte Netze und hybride Netze.

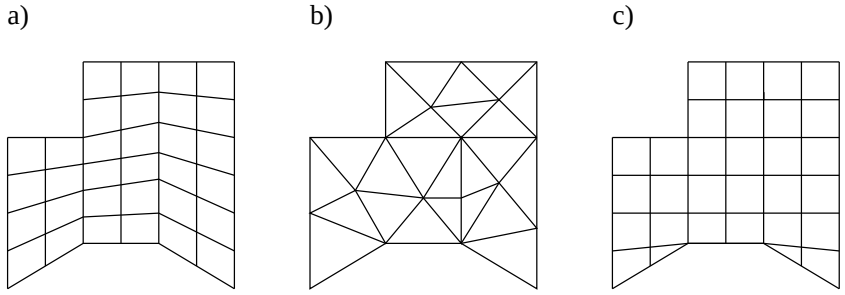


Abbildung 2.1: *a) Strukturiertes Netz, b) Unstrukturiertes Netz, c) Hybrides Netz*

2.1.1 Strukturierte Netze

Strukturierte Netze werden durch die folgende Definition beschrieben:

Definition 1 *Ein Elementnetz wird als strukturiert bezeichnet, wenn an allen inneren Knoten des Netzes regulär sind.*

Ein regulärer Knoten wird wie folgt definiert:

Definition 2 *Ein Knoten wird als regulär bezeichnet, wenn die Anzahl der an ihn grenzenden Elemente die optimale Anzahl ist. Die optimale Anzahl von Elementen ergibt sich aus Tabelle 2.1.*

Bei strukturierten Netzen ist, wie in Abbildung 2.1 a) dargestellt, die Anzahl der an einen Knoten angrenzenden Elemente für jeden inneren Knoten gleich und entsprechend der optimalen Anzahl.

Strukturierte Netze können aus ein-, zwei- oder dreidimensionalen Elementen bestehen. Die Eckpunkte der Elemente liegen in einer strukturierten Ordnung vor. Eine Abbildung des Netzes

Element-Typ	optimale Element-anzahl je Knoten
Dreieck	6
Viereck	4
Tetraeder	24
Hexaeder	8

Tabelle 2.1: *Reguläre Punkte: Optimale Anzahl der an einen Knoten angrenzenden Elemente*

auf ein Netz mit regelmäßigen Elementen ist möglich. Für ein Netz aus Viereckelementen ist die Abbildung auf ein Netz mit regelmäßigen Rechtecken in Abbildung 2.2 dargestellt.

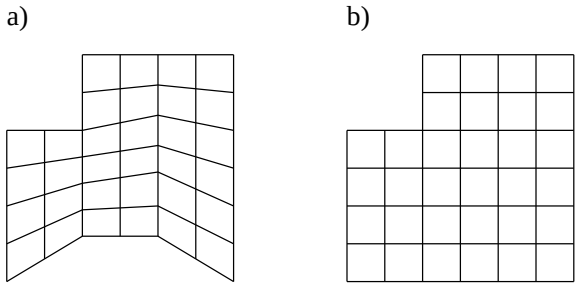


Abbildung 2.2: *a) Strukturiertes Netz, b) Abbildung des strukturierten Netzes*

2.1.1.1 Algebraische Methode

Die algebraische Vorgehensweise bei der Generierung strukturierter Netze funktioniert für zweidimensionale Systeme. Eine Anwendung der Technik auf dreidimensionale Gebilde ist aktueller Forschungsgegenstand. Um mit dieser Methode die Netze zu generieren, wird die Geometrie auf ein Einheitsviereck projiziert und mit Hilfe von Interpolationsfunktionen, vergleichbar mit den Formfunktionen, zerlegt. Aufgrund der Abbildung ist eine Vernetzung nur von einfachen Geometrien möglich. Das Verfahren ist durch seine Einfachheit sehr effizient [Banks 1995].

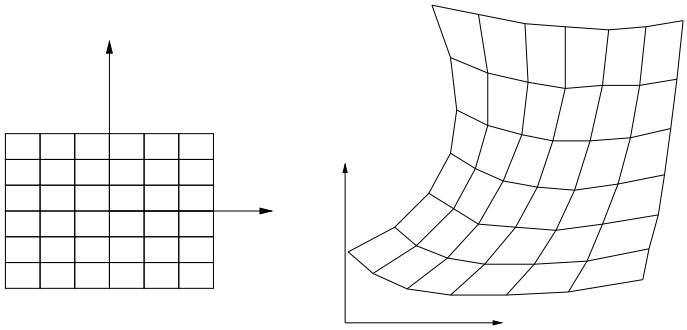


Abbildung 2.3: *Abbildung der Geometrie auf ein Einheitsviereck, Elementierung*

2.1.1.2 Netzgenerierung durch Lösung partieller Differentialgleichungen

Die Generierung von strukturierten Netzen durch die Lösung partieller Differentialgleichungen ist für zwei- und dreidimensionale Gebiete möglich. Der Grundgedanke der Methode liegt in der Definition einer Differentialgleichung, wie z.B. der Poisson Gleichung für kartesische Koordinaten in Gebietskoordinaten, deren Lösung auf dem Gebietskoordinatensystem das zu erzeugende Netz in kartesischen Koordinaten ist. Hierbei muss eine analytische Lösung für die Differentialgleichung möglich sein. Diese ist in den meisten Fällen nur für einfache Geometrien, wie z.B. Rechtecke, möglich.

Elliptische Differentialgleichungen werden zur Vernetzungen von berandeten Gebieten verwendet, während hyperbolische und parabolische Differentialgleichungen zur Vernetzung unberandeter Gebiete verwendet werden [Filipiak 1996].

Die Qualität der Netze, die mit Hilfe von partiellen Differentialgleichungen ermittelt wurden, ist besser, als die Qualität der Netze, die durch Interpolationsfunktionen ermittelt wurden. Der Aufwand zur Generierung des Netzes ist jedoch wesentlich höher.

2.1.1.3 Multiblock

Die beiden dargestellten Algorithmen zur globalen Generierung von strukturierten Netzen können nur einfache Geometrien vernetzen. Zur Vernetzung komplizierterer Geometrien bietet die Multiblock-Methode die Möglichkeit, die Geometrie in einfachere, vernetzbare Teilgeometrien zu zerteilen, um diese dann hinterher mit einer der oben beschriebenen Vorgehensweisen zu vernetzen. Eine Anwendung des Verfahrens ist sowohl für zweidimensionale Dreieck- und Viereckknetze als auch für dreidimensionale Tetraeder- und Hexaedernetze möglich. Bei Dreieck- und Tetraedernetzen weisen die Elemente an den Teilungslinien bzw. -flächen eine schlechtere Qualität auf. Die betroffenen Knoten können unter Umständen sogar irregulär sein.

Die Geometrie wird in vernetzbare Teilgebiete aufgeteilt, die *Blocks* genannt werden. Diese *Blocks* werden nach dem Generieren der Teilnetze zum *Multiblock* zusammengefasst. Hierbei ist auf Kompatibilität an den Verbindungskanten der Teilnetze zu achten.

Eine Automatisierung des Aufteilungsprozesses ist nur teilweise möglich. Ein Ansatz hierfür ist die Mittelachsen- bzw. Mittelflächentechnik [Price u. a. 1995], [Price und Armstrong 1997] oder die Aufteilung unter Verwendung des eingeschlossenen Voronoi Graphen [Sheffer u. a. 1998]. In den meisten Fällen ist jedoch eine Steuerung der Aufteilung durch den Nutzer notwendig. Diese Notwendigkeit tritt besonders bei den Sonderfällen des Multiblock-Verfahrens, den O-, C- und H-Typ-Netzen, auf. Diese Typen von Netzen werden durch Zusammenlegen von jeweils zwei Knoten zu einem Netzknoten aus regelmäßigen Rechtecknetzen erzeugt. Verwendet werden diese Typen von Netzen um Löcher zu generieren, ohne Knoten zu erhalten, an denen die Anzahl der angrenzenden Elemente nicht der üblichen Anzahl entspricht. Die Buchstaben-Namen *O* für O-Typ, *C* für C-Typ und *H* für H-Typ entsprechen jeweils der Form der Netze.

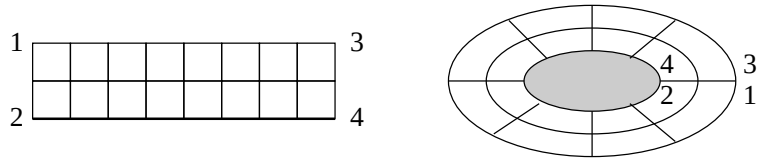


Abbildung 2.4: *Multiblock: O-Typ Netz*

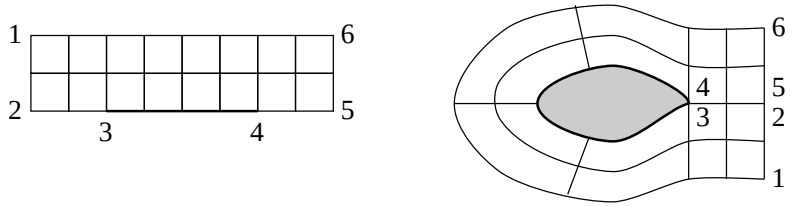


Abbildung 2.5: *Multiblock: C-Typ Netz*

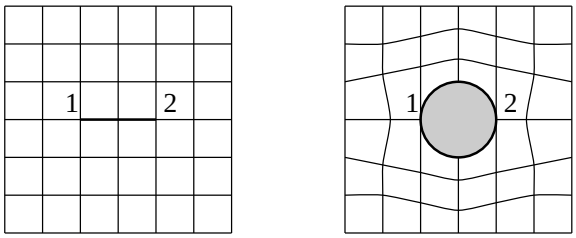


Abbildung 2.6: *Multiblock: H-Typ Netz*

2.1.2 Unstrukturierte Netze

In unstrukturierten Netzen ist die Anzahl der Nachbarknoten bzw. Nachbarelemente für die einzelnen Knoten unterschiedlich (Abbildung 2.1 b)). Eine wesentlich größere Flexibilität in Hinsicht auf die Anpassung an die Geometrie als auch bei Verfeinerungen zeichnen unstrukturierte Netze im Vergleich zu strukturierten Netzen aus. Die beiden wichtigsten Methoden zur Erzeugung unstrukturierter Netze sind die Advancing-Front Methode und die Delaunay Triangulierung. Die Algorithmen sind in Abschnitt 3.1 dargestellt.

2.1.3 Hybride Netze

Hybride Netze bestehen aus verschiedenen Elementarten. Zur Erzeugung dieser Netze werden Vorgehensweisen der strukturierten und der unstrukturierten Netzgenerierung kombiniert. Zweidimensionale Netze bestehen aus Dreiecken und Vierecken, dreidimensionale Netze aus Tetraedern und Hexaedern (Abbildung 2.1 c)). Die resultierenden Netze sind daher unstrukturiert.

Hybride Netze bieten im Vergleich zu strukturierten Netzen eine wesentlich größere geometrische Flexibilität. Ein weiterer Vorteil hybrider Netze liegt in der geringeren Anzahl an Elementen, die die Netze aufweisen [Shaw 1999]. Zur Erzeugung hybrider Netze gibt es verschiedene Algorithmen. [Owen 1999] stellt ein Verfahren zur Erzeugung hexaeder-dominanter Netze vor, das auf Basis eines Viereckenetzes auf der Geometrieoberfläche eine Tetraedervernetzung durchführt und diese dann in das hybride Netz konvertiert. [Thompson und Soni 1999] erzeugen viereck- und hexaeder-dominante Netze. Hierbei wird ausgehend von strukturierten Netzen durch Glättung sowie Einfügen und Löschen von Kanten und Elementen eine Anpassung an die Geometrie vorgenommen.

2.2 Finite-Elemente-Methode

Die Methode der Finiten Elemente gehört zu den wichtigsten und am häufigsten verwendeten numerischen Lösungsverfahren zur Lösung von Differentialgleichungen im Bauwesen. Entwickelt wurde die Methode zur Lösung stukturmechanischer Problemstellungen, für die es im Allgemeinen keine analytische Lösung gibt.

Bei den zu berechnenden Systemen unterscheidet man zwischen kontinuierlichen und diskreten Systemen. Diskrete Systeme haben eine endliche Anzahl von Teilsystemen mit jeweiligen Zustandsgrößen, die aus einer algebraischen Beschreibung berechnet werden können. Die Aufteilung des Systems in kleine Teilsysteme, auch Elemente genannt, ist durch das System vorgegeben. In diesen Elementen lassen sich die Zustandsgrößen exakt darstellen [Betten 1997]. Im Bauwesen sind Beispiele für diskrete Systeme Fachwerke oder Rahmen. Kontinuierliche Systeme lassen sich nur durch Differentialgleichungen beschreiben. Die Zustandsgrößen sind an jeder Stelle unterschiedlich. Eine Diskretisierung kontinuierlicher Systeme, d.h. eine Zerlegung in kleine Elemente, ist daher notwendig. Für diese Elemente lassen sich dann Gleichungen aufstellen. Innerhalb der Elemente kann die mechanische Theorie teilweise exakt simuliert werden. Durch die Übergänge zwischen den Elementen, an denen Unstetigkeiten und Sprünge in den Zustandsgrößen vorkommen, entsteht ein Fehler in der Berechnung. Die Genauigkeit der Berechnung hängt daher unter anderem von der Diskretisierung des Systems ab.

Auf die ermittelten diskreten Elemente wird die jeweils zutreffende mechanische Theorie angewendet. Die Lösung des Systems wird im Zusammenwirken aller diskreten Elemente unter Berücksichtigung von Rand- und Übergangsbedingungen berechnet.

Die Finite-Elemente-Methode ist, mit Ausnahme der Lösung diskreter Systeme, aufgrund der notwendigen Diskretisierung und der Interpolation innerhalb der Elemente, ein Approximationsverfahren. Die Ergebnisse der Berechnung sind in Abhängigkeit von der Diskretisierung mit Fehlern behaftet. Mit Hilfe adaptiver Verfahren ist eine automatische Minimierung des Approximationsfehlers möglich [Olden 1998].

2.2.1 Allgemeines Vorgehen

Zur Darstellung der Vorgehensweise der Finite-Elemente-Methode wird vereinfachend von einem linearen System ausgegangen.

Die allgemeine Vorgehensweise gliedert sich in folgende Schritte:

Schritt 1: **Diskretisierung des Systems:** Das System wird geometrisch in kleine Elemente analog zu den zu wählenden Ansatz- und Formfunktionen und der mechanischen Theorie zerlegt. Hierbei wird die Anzahl der Freiheitsgrade des Systems von unendlich auf eine endliche Anzahl reduziert [Zienkiewicz 1975]. Geometrie und Materialübergänge sind bei der Diskretisierung zu beachten. Unter Umständen ist eine Näherung der Geometrie, wie z.B. bei Kreisbögen, notwendig.

Schritt 2: **Ermittlung der Elementmatrizen:** Die Ansatzfunktionen für jedes Element zur Interpolation der Systemgrößen und der Geometrie werden unter Berücksichtigung der Stetigkeitsanforderungen der Variationsaufgabe ausgewählt. Mit Hilfe des Extremalprinzips wird die gewählte Ansatzfunktion in das Funktional eingesetzt. Hierdurch entsteht eine Funktion, die aus Gebiets- und Randintegralen besteht. Ein Teil der Gebietsintegrale ergibt die Elementsteifigkeitsmatrix, während die restlichen Integrale die Randbedingungen beschreiben. Als Beispiel für eine solche Funktion ist hier die Funktion einer Scheibe [Girkmann 1954] dargestellt:

$$\begin{aligned} \delta E \;=\; & \underbrace{\iint\limits_{A} \delta\hat{v}_\alpha(-\bar{q}^\alpha - E^{\alpha\beta\gamma\delta} v_{\gamma|\delta\beta})\, dA}_{Elementmatrix} \\ & - \underbrace{\int\limits_{\Gamma_v} \delta\hat{v}_\alpha(n^{\alpha\beta} - \hat{n}^{\alpha\beta}) n_\beta \, d\Gamma_v}_{statischeRandbedingung} \\ & - \underbrace{\int\limits_{\Gamma_\sigma} \delta\hat{v}_\alpha(n^{\alpha\beta} - \hat{n}^{\alpha\beta}) n_\beta \, d\Gamma_\sigma}_{geometrischeRandbedingung} = 0 \end{aligned} \tag{2.1}$$

Eine symbolische Lösung der Integration ist nur in einfachen Fällen möglich, in schwierigeren Fällen ist daher eine numerische Integration z.B. nach Gauß [Bathe 1990] notwendig.

Schritt 3: **Zusammenbau des Gesamtsystems:** Die vorher berechneten Elementmatrizen werden zu einer Gesamtmatrix für das gesamte System unter Berücksichtigung der Koppel- und Übergangsbedingungen zwischen den einzelnen Elementen superponiert. Die Gesamtmatrix ist zu diesem Zeitpunkt singulär.

Schritt 4: **Einarbeitung von Rand- und Übergangsbedingungen:** Nachdem die Gesamtmatrix K aufgestellt wurde, wird im nächsten Schritt das Gleichungssystem der Form

$$\delta v^T \; K \cdot v = F \; \delta v^T \tag{2.2}$$

zusammengebaut. v ist hierbei der Unbekanntenvektor und F der Rechteseitevektor mit den Belastungen des Systems. Die aus den oben beschriebenen Anteilen der Integrale der Randbedingungen berechneten Vektoren werden in den Vektor F eingebaut. Freiheitsgrade mit vorgegebenen Werten, wie z.B. vorgegebene Verschiebungen bei Aufgabenstellungen aus dem Bereich der Strukturmechanik oder vorgegebene Wasserhöhen bei Problemstellungen aus dem Bereich der Strömungsmechanik, und Belastungen des Systems sind analog zur Diskretisierung entsprechend den gewählten Ansatzfunktionen in den Vektor v einzubauen. Zeilen und Spalten, bei denen der zugehörige Wert des Vektors v Null ist, werden gestrichen. Das Gleichungssystem ist nun lösbar.

Schritt 5: **Lösung des Gleichungssystems:** Zur Lösung des Gleichungssystems stehen eine Reihe von Algorithmen zur Verfügung. Bei der Auswahl des Gleichungslösers sind die Eigenschaften des Gleichungssystems zu berücksichtigen. Bei linearen Gleichungssystemen können direkte Löser, wie z.B. der Gauß-Algorithmus oder der Cholesky-Algorithmus, Iterative Löser, wie z.B. Jacobi-Verfahren oder Gauß-Seidel-Verfahren, oder Semi-Iterative Verfahren, wie das Gradientenverfahren, verwendet werden [Bronstein 1997]. Bei nichtlinearen Gleichungssystemen stehen z.B. das Newtonsche Verfahren, das nichtlineare Gauß-Seidel-Verfahren oder das nichtlineare CG-Verfahren zur Verfügung [Törnig und Spellucci 1988].

Schritt 6: **Elementweise Berechnung der Unbekannten:** Nachdem der primäre Unbekanntenvektor v bestimmt wurde, werden hieraus die sekundären Größen innerhalb der Elemente, wie z.B. Schnittgrößen bei Aufgabenstellungen aus dem Bereich der Strukturmechanik oder Durchflüsse bei Problemstellungen aus dem Bereich der Strömungsmechanik, sowie die noch fehlenden Werte der rechten Seite F berechnet.

2.2.2 Bedingungen für die Netzgenerierung

Für die Netzgenerierung ergeben sich aus der Methode der Finiten Elemente eine Reihe von Bedingungen. Zum einen wird die Form der Elemente und die Anzahl der Knoten je Element vorgegeben. Des weiteren bestimmt die Finite-Elemente-Berechnung die Größe der Elemente bzw. Stellen mit notwendigen Verfeinerungen des Netzes.

2.2.2.1 Elemente

Bei der Verwendung der Methode der Finiten Elemente ist, wie oben beschrieben, eine Diskretisierung der Geometrie notwendig. Diese Diskretisierung geschieht durch eine Zerlegung in kleine, der mechanischen Theorie genügenden Elemente.

In Abschnitt 2.2.1 wurden die zu lösenden Flächen- und Randintegrale beschrieben. Diese Integrale werden elementweise gelöst. Hierfür ist eine Interpolationsfunktion innerhalb der Elemente notwendig. Als Interpolationsfunktionen werden häufig Polynome verwendet.

Das Pascalsche Polynomschema ist ein einfaches Hilfsmittel zur Auswahl der notwendigen Polynomterme.

$$\begin{array}{ccccc} 1 & \eta & \eta^2 & \eta^3 & \cdots \\ \xi & \xi\eta & \xi\eta^2 & \xi\eta^3 & \\ \xi^2 & \xi^2\eta & \xi^2\eta^2 & \xi^2\eta^3 & \\ \xi^3 & \xi^3\eta & \xi^3\eta^2 & \xi^3\eta^3 & \\ \vdots & & & & \ddots \end{array} \tag{2.3}$$

Für eindimensionale Elemente werden nur Terme mit einer Unbekannten verwendet. Dies kann durch die Auswahl der ersten Spalte oder Zeile des Polynomschemas geschehen. Bei zweidimensionalen Elementen sind zur Interpolation zwei Unbekannte notwendig. Für rechteckige Elemente ist die Auswahl eines rechteckigen Abschnitts, für dreieckige Elemente die Auswahl eines dreieckigen Abschnitts des Polynomschemas notwendig. Untere Polynomglieder dürfen hierbei nicht weggelassen werden.

Als Beispiel ist hier das Polynom für ein Dreieckelemente mit sechs Knoten angegeben. Zur Beschreibung des Elementes sind sechs Polynomterme notwendig, weshalb die quadratischen Anteile mitgenommen werden müssen:

$$x^i = a^i{}_1 + a^i{}_2\xi + a^i{}_3\eta + a^i{}_4\xi^2 + a^i{}_5\xi\eta + a^i{}_6\eta^2$$

Die Auswahl der verwendeten Polynomterme hängt von verschiedenen Anforderungen ab [Meißner und Menzel 1989]:

Anforderung 1: Im Inneren der Elemente müssen die Ansatzfunktionen stetig sein.

Anforderung 2: Benachbarte Elemente müssen auf der Kante die gleichen Ansätze und gleich viele Knoten besitzen, d.h. sie müssen kompatibel sein.

Anforderung 3: Das mechanische Problem muss durch den Ansatz richtig abgebildet werden können. Dies bedeutet, dass z.B. Starrkörperverschiebungen bei der Deformationsmethode möglich sein müssen.

Anforderung 4: Die Geometrie des zu approximierenden Körpers soll möglichst gut abgebildet werden können.

Anforderung 5: Niedrige Polynomglieder dürfen nicht weggelassen werden, d.h. die Ansatzfunktion muss vollständig sein.

Anforderung 6: Die Ansatzfunktionen müssen in der Regel ableitbar sein. Je nach Problemstellung müssen mehrere Ableitungen der Ansätze gebildet werden können. Die höchste notwendig zu bildende Ableitung darf konstant sein.

Anforderung 7: Die aus den Ansatzfunktionen berechneten Formfunktionen müssen linear unabhängig voneinander sein.

Anforderung 8: Die Ansatzfunktion muss invariant gegenüber Drehung des Koordinatensystems sein.

Die Diskretisierung selbst erzeugt durch die verwendeten Ansatzfunktionen einen Fehler. Der Approximationsfehler ergibt sich entsprechend der Taylorreihe aus den abgeschnittenen Gliedern des Terms. Für ein eindimensionales Element sieht das Polynom folgendermaßen aus:

$$\underbrace{x^i = a^i{}_1 + a^i{}_2\xi + a^i{}_3\xi^2 + a^i{}_4\xi^3}_{Ansatzfunktion} \quad \vdots \quad \underbrace{+a^i{}_5\xi^4 + a^i{}_6\xi^5 + \cdots}_{Fehler} \tag{2.4}$$

Bei der Auswahl von nur vier Polynomgliedern (kubischer Ansatz) würde das Polynom an der gestrichelten Linie abgeschnitten. Den Approximationsfehler erhält man aus dem abgeschnittenen Teil des Polynoms und ist in diesem Fall vom Grad ξ^4.

2.2.2.2 Randbedingungen

Mit den bisher beschriebenen Möglichkeiten lässt sich die innere Funktionsweise eines Systems beschreiben. Die innere Funktionsweise ist immer eine Systemantwort auf die von außen eingebrachten Bedingungen, auch Randbedingungen genannt. Randbedingungen sind an das System von außen aufgebrachte Zustandsgrößen.

Bei der Diskretisierung von Systemen für die Finite-Elemente-Berechnung sind die Randbedingungen des Systems mit den Elementen und Knoten des resultierenden Netzes, wie in Abschnitt 2.2.1 beschrieben, zu verbinden. Prinzipiell unterscheidet man zwischen den wesentlichen und den natürlichen Randbedingungen.

Wesentliche und natürliche Randbedingungen lassen sich jeweils noch einmal in homogene und inhomogene Randbedingungen unterteilen. Homogen bedeutet in diesem Fall, dass der Wert der Randbedingung Null ist, während der Wert bei inhomogenen Randbedingungen ungleich Null ist.

Wesentliche Randbedingungen werden in der Literatur [Schwarz 1991] auch als Dirichletsche Randbedingungen bezeichnet. Aufgrund der für diese Arbeit verwendeten Anwendungsbeispiele werden hier die Randbedingungen der Deformationsmethode beschrieben. Bei der Anwen-

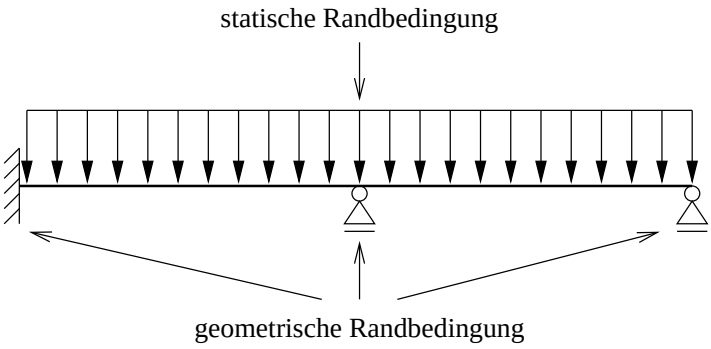


Abbildung 2.7: *Randbedingungen am Beispiel Balken*

dung der Deformationsmethode sind die wesentliche Randbedingungen geometrische Randbedingungen.

Geometrische Randbedingungen sind definiert als vorgegebene Deformationen an bestimmten Stellen des Systems. Beispiele hierfür sind Auflagerreaktionen, wie Lager, Einspannungen oder vorgegebene Lagerabsenkungen (siehe Abbildung 2.7), die als vorgegebene $v_i = x$ in der Gleichung 2.2 zu sehen sind.

In das Gleichungssystem lassen sich homogene wesentliche Randbedingungen ($v_i = 0$) leicht einbauen. Hierfür wird normalerweise die Zeile und Spalte des entsprechenden Freiheitsgrades der Matrix und der rechten Seite gestrichen. Inhomogene wesentliche Randbedingungen sind schwieriger in das Gleichungssystem einzubauen. Ein vorgegebener Wert für den Lösungsvektor, wie z.B. eine vorgegebene Stützenabsenkung, müssen durch Umformen des Gleichungssystems auf die rechte Seite gebracht werden.

Für die Netzgenerierung bedeuten diese Randbedingungen, dass an den Stellen der beabsichtigten Lagerung Knoten des Elementnetzes vorhanden sein müssen. In einem Nachlauf nach der Netzgenerierung ist eine Zuordnung der vorgegebenen Deformationen zu den Freiheitsgraden notwendig.

Natürliche Randbedingungen werden häufig auch Cauchy Randbedingungen genannt. Der homogene Fall der Cauchy Randbedingung wird als Neumannsche Randbedingung bezeichnet. Bei der Verwendung der Deformationsmethode sind die natürlichen Randbedingungen die statischen Randbedingungen.

Statische Randbedingungen sind vorgegebene rechte Seiten ($F_i = x$) in der Gleichung 2.2. Knoten ohne Belastung und Lagerung lassen sich als Knoten mit statischer homogener Randbedingung beschreiben. Hierbei ist $F_i = 0$.

Die Berücksichtigung der natürlichen Randbedingungen sind in Bezug auf die Netzgenerierung komplizierter. An Stellen mit punktförmigen natürlichen Randbedingungen sind Knoten in das Netz einzufügen und eine Verbindung zu den Randbedingungen herzustellen.

Bei linienförmigen Belastungen ist z.B. bei Scheibenproblemen der 2. Term der Gleichung 2.1 auszuwerten. Die Lösung des Integrals $\int_{\Gamma_v} \delta\hat{v}_\alpha(n^{\alpha\beta} - \hat{n}^{\alpha\beta})n_\beta \, d\Gamma_v$ ist hierfür notwendig. Das Ergebnis wird auf die Knoten verteilt. Hierfür ist zur Integration die Formfunktion bezüglich des Randes der betroffenen Elemente zu berücksichtigen. Die Formfunktion bezüglich des Randes ist immer eine Dimension geringer als die des Elementes. So ist z.B. für eine Linienlast bei der Verwendung von Viereckelementen mit quadratischem Ansatz eine eindimensionale quadratische Formfunktion zu wählen.

2.2.2.3 Adaptivität

Unter Adaptivität in Bezug auf die Finite-Elemente-Methode versteht man ein Anpassen der Diskretisierung an die Qualität der Lösung. Die Qualität einer Finite-Elemente-Simulation hängt von der verwendeten Diskretisierung ab. Mit Hilfe von Fehlerindikatoren und Fehlerestimatoren kann die Güte der Simulation abgeschätzt werden.

Fehlerestimatoren wurden zur Abschätzung der Qualität im gesamten Berechnungsgebiet entwickelt. Für die Verfeinerung und Optimierung von Finite-Elemente-Netzen ist der lokale Fehler von Interesse. Dieser wird von Fehlerindikatoren angezeigt.

Fehlerindikatoren: Zur Berechnung des Fehlers einer Näherungslösung ist der Vergleich mit der exakten Lösung notwendig. Fehlerindikatoren gehen den Weg, aus der Berechnung eine verbesserte Lösung zu ermitteln und damit die Näherungslösung zu vergleichen.

Zienkiewicz/Zhu-Fehlerindikator: Bei dem Fehlerindikator nach Zienkiewicz/Zhu-Fehlerindikator [Zienkiewicz und Zhu 1991] wird eine Interpolation der Schnittgrößen über die Elemente in einem Knoten vorgenommen.

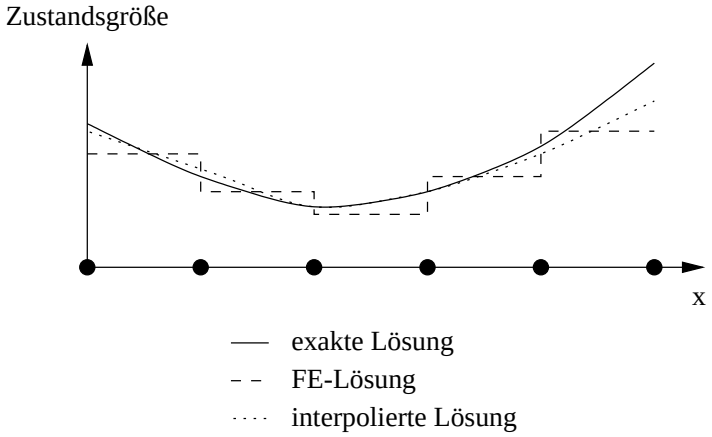


Abbildung 2.8: *Interpolation einer Zustandsgröße nach Zienkiewicz/Zhu*

Die Interpolation einer Zustandsgröße mit linearer Ansatzfunktion für den eindimensionalen Fall ist in Abbildung 2.8 dargestellt. Eine Verbesserung der interpolierten Lösung gegenüber der

Finite-Elemente-Lösung ist klar zu erkennen. Der Fehlerindikator an einem Knoten ergibt sich dann aus der Differenz der interpolierten Lösung und der Finite-Elemente-Lösung.

Meißner-Wibbeler-Fehlerindikator: Der Meißner-Wibbeler-Fehlerindikator [Meißner und Wibbeler 1990] berechnet aus den Sprüngen der Finite-Elemente-Näherung fehlerhafte Rand- und Übergangsbedingungen, die als Fehlerlasten interpretiert werden können.

Aus einer patchweisen Betrachtung wird die verbesserte Lösung zum Vergleich mit der Finite-Elemente-Approximation gewonnen. Das Modell wird patchweise verfeinert und so erhält man einen lokal verbesserten Verlauf der Lösung.

Ein Elementpatch besteht aus allen Elementen, zu denen der zu betrachtende Knoten, für den der Fehlerindikator berechnet werden soll, gehört. Für diese Elemente wird dann eine Verfeinerung nach der h–Version oder p–Version entsprechend Abschnitt 2.2.2.4 vorgenommen. Bei der verfeinerten Berechnung werden die Fehlerlasten mit angesetzt.

Babushka-Fehlerindikator: Der Babushka-Fehlerindikator berechnet das Fehlermaß aus den Sprüngen der Zustandsgrößen an den Elementkanten. Die Residuen der einzelnen Zustandsgrößen werden analog zur ersten Variation des Gesamtpotentials unter Berücksichtigung der Fehlerlasten entlang der Elementkante aufintegriert und gewichtet und so zu dem Fehlerindikator zusammengefasst.

2.2.2.4 Adaptive Strategien

Die Fehlerindikatoren geben Auskunft über die Qualität der Finite-Elemente-Simulation an bestimmten Stellen des Systems. Zur Minimierung des Fehlers sind adaptive Strategien zur Verbesserung des Netzes notwendig. Zum einen ist es möglich das Netz neu zu generieren, zum anderen kann eine Verbesserung der bestehenden Diskretisierung durchgeführt werden.

Neugenerierung: Die Diskretisierung wird komplett verworfen. Aus der erhaltenen Fehlerverteilung werden Annahmen für die Elementgrößenverteilung für das neu zu erstellende Netz getroffen. Das Netz wird anhand dieser Annahmen, wie in Abbildung 2.9 dargestellt, neu generiert. Durch Interpolation ist eine Zuordnung der im ersten Schritt berechneten Zustandsgrößen des Systems zu dem neuen Netz möglich. Hierdurch lässt sich der Rechenaufwand z.B. bei der Gleichungslösung mit iterativen Lösern bei der Finte-Elemente-Berechnung minimieren. Eine Reduzierung der Anzahl der Elemente an Stellen mit kleinem, fast verschwindenden Fehler wird automatisch durchgeführt.

h–Version: Die h–Version ist eine Methode zur Anpassung eines bestehenden Netzes an die ermittelte Fehlerverteilung. Hierbei werden entsprechend der Verteilungsfunktion, die aus der Fehlerverteilung resultiert, in das Netz an Stellen mit großem Fehler feinere Elemente eingefügt. Hierdurch erhöht sich die Anzahl der Elemente. An den Übergängen zwischen Regionen mit

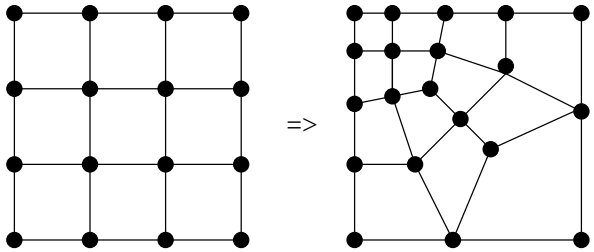


Abbildung 2.9: *Neugenerierung des Netzes unter Beachtung der Fehlerverteilung*

kleineren und größeren Elementen sind Übergangselemente mit einer von der üblichen Anzahl abweichenden Anzahl von Knoten je nach Vorgehensweise notwendig. An Stellen mit geringem Fehler wird standardmäßig keine Veränderung vorgenommen. Um hier Vergröberungen des Netzes zu bekommen ist eine komplizierte und zeitaufwendige Prozedur notwendig.

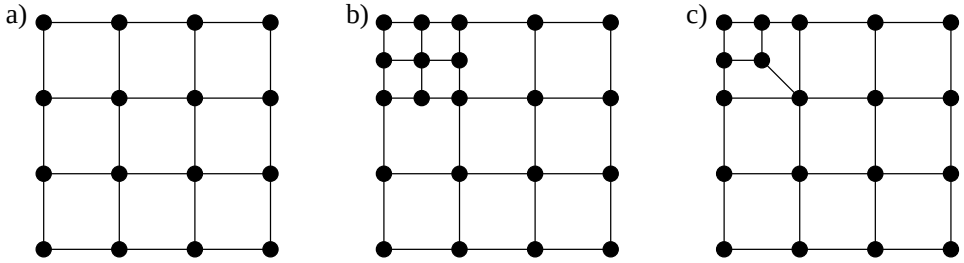


Abbildung 2.10: *h–Version, a) Ausgangsnetz, b) mit Übergangselementen, c) mit regulären Elementen*

p–Version: Die p–Version ermöglicht eine Verringerung des Fehlers in der Finite-Elemente-Berechnung ohne Änderung der Lage der Elemente. Der Polynomgrad der Ansatzfunktionen der betroffenen Elemente wird erhöht. Hierdurch entstehen neue Knoten und die Anzahl der Freiheitsgrade steigt. Eine neue Netzgenerierung ist hierfür nicht notwendig. Der numerische Aufwand innerhalb der Finite-Elemente-Analyse steigt dafür stärker an als bei den anderen Methoden.

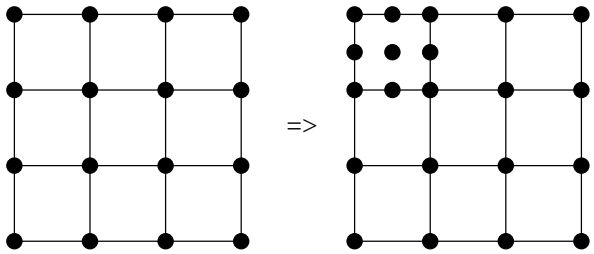


Abbildung 2.11: *p–Version*

r–Version: Die Verschiebung der Knoten ist Grundgedanke der r–Version. Für diese Verschiebung ist ebenfalls die Fehlerverteilung notwendig. Die Knoten werden entlang des Gradienten der Fehlerverteilung zu Stellen mit schlechterem Fehler verschoben. Dieses Vorgehen ist bei komplizierten Geometrien recht aufwendig. Ein maximaler Gesamtfehler ist durch die Anzahl der Freiheitsgrade des Systems vorgegeben. Da keine weiteren Freiheitsgrade in das System eingeführt werden, kann bei dieser Art der Optimierung nur der Fehler auf ein gewisses Maß reduziert werden. Durch die konstante Anzahl an Freiheitsgraden erhöht sich der Aufwand der Finite-Elemente-Berechnung im Gegensatz zu den anderen Versionen nicht.

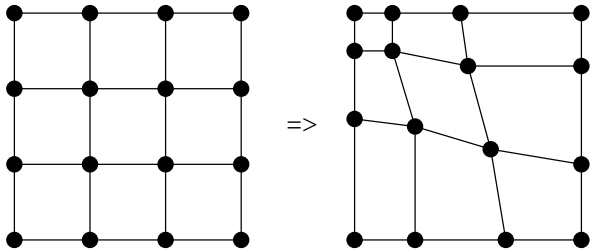


Abbildung 2.12: *r–Version*

2.3 Gebietszerlegungsverfahren

Die im Abschnitt 2.2 beschriebene Methode der Finiten Elemente hat sich in den letzten Jahren zur numerischen Simulation von Ingenieurproblemstellungen als geeignet erwiesen. Bei der Verwendung der im Abschnitt 2.6 beschriebenen Parallelrechner existieren zwei Vorgehensweisen zur Parallelisierung, die beide auf einer Gebietszerlegung basieren:

Implizite Partitionierung: Bei der impliziten Partitionierung wird das komplette Gleichungssystem der Form $K \cdot v = F$ aufgestellt und dann, wie in Abbildung 2.13 dargestellt, auf die Prozessoren verteilt und parallel gelöst. Hierfür ist ein hoher Kommunikationsaufwand notwendig. Das Aufstellen des Gleichungssystems wird meist sequentiell durchgeführt.

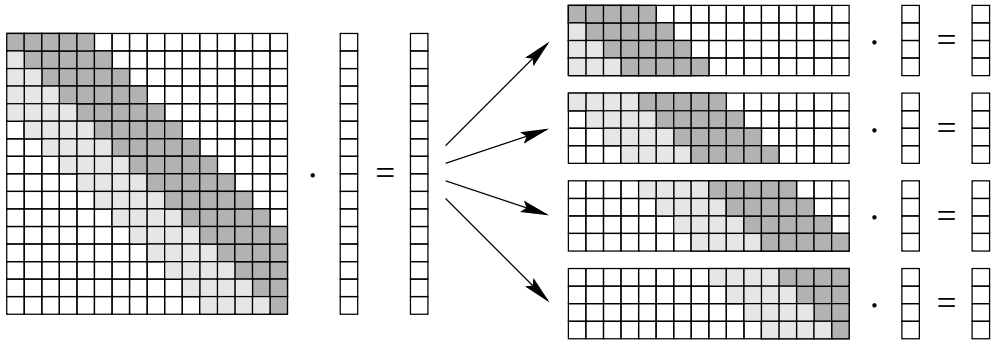
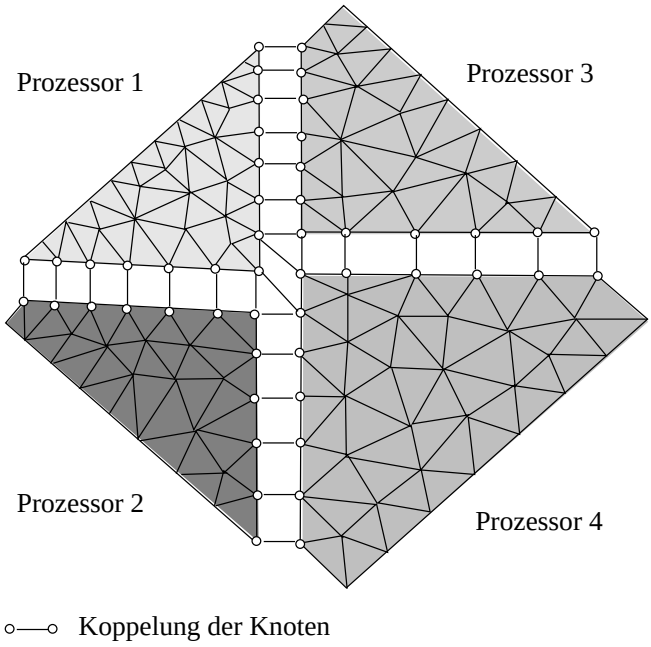


Abbildung 2.13: *Implizite Partitionierung*

Explizite Partitionierung: Bei der expliziten Partitionierung wird das Netz, wie in Abbildung 2.14 demonstriert wird, zerteilt. Jeder Prozessor erhält ein Teilnetz, für das er das Gleichungssystem aufstellt.



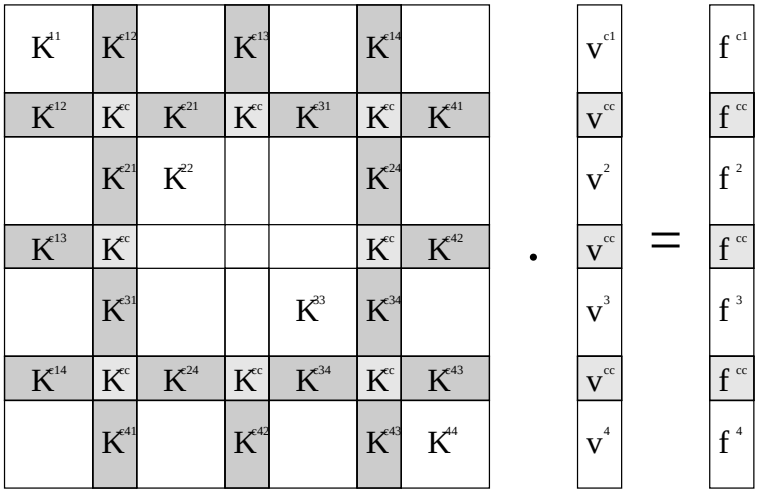


Abbildung 2.14: *Substruktur-Technik*

Ein Verfahren, welches hierfür gut geeignet ist, ist die Substruktur-Technik. Diese wurde entwickelt, um Problemstellungen, die zu groß für den Hauptspeicher waren, mit Hilfe der Finiten Elemente in mehreren Schritten zu lösen. Bei der Finite-Elemente-Berechnung auf einem Parallelrechner berechnet jeder Prozessor entsprechend der Substrukturtechnik das Gleichungssystem für die ihm zugewiesenen Elemente. Knoten entlang der Teilungsgrenzen bezeichnet man als Kop-

pelknoten und sind auf jedem Prozessor vorhanden, der ein Element mit einem Koppelknoten besitzt.

Folgende Schritte sind bei der parallelen Berechnung erforderlich:

Schritt 1: Aufstellen und Lösen des Gleichungssystems mit Dirichletscher Randbedingung an den Koppelknoten

Schritt 2: Aufstellen des Koppelknotensystems

Schritt 3: Berechnen des Koppelknotensystems

Schritt 4: Berechnung im Inneren der Teilgebiete unter Berücksichtigung der Lösung für die Koppelknoten

2.4 Partitionierungsalgorithmen

Für eine parallele Finite-Elemente-Berechnung ist eine Aufbereitung der Eingabedaten in Form einer Gebietszerlegung notwendig. Die Aufbereitung der Daten beinhaltet eine Zuordnung der Elemente, Knoten und der zugehörenden Randbedingungen zu den Prozessoren. Der Schritt der Aufbereitung kann entweder separat durch ein Programm, durch einen Nachlauf der Netzgenerierung oder als Vorlauf vor der Finite-Elemente-Analyse geschehen.

Die Gebietszerlegung ordnet die einzelnen Elemente den Prozessoren zu und beeinflusst somit deren Rechenlast. Der geometrische Zusammenhang der Teilgebiete wirkt sich auf die Länge der Kommunikationsränder aus, was einen direkten Einfluss auf die notwendige Kommunikationsmenge während der Finite-Elemente-Berechnung hat. Die Partitionierungsalgorithmen benutzten entweder graphenbasierte, algebraische oder geometrische Ansätze.

2.4.1 Geometrische Algorithmen

Geometrische Algorithmen teilen ein Netz nach geometrischen Gesichtspunkten auf. Innerhalb des Netzes wird ein Punkt festgelegt, durch den eine Schnittlinie gelegt wird, entlang der dann das Netz bzw. Teilnetz in zwei Teile geteilt wird. Dieser Schritt wir rekursiv wiederholt, bis die Anzahl der gewünschten Teilgebiete erreicht ist.

2.4.1.1 Koordinaten-Bisektion

Die rekursive Koordinaten-Bisektion (siehe [Bokhari 1981] und [Chrisochoides u. a. 1994b]) teilt ein Finite-Elemente-Netz rekursiv entlang einer gewählten Koordinatenachse des beschreibenden Koordinatensystems. Die Knoten des Netzes werden in Gruppen mit gleicher Anzahl im Abstand zur Koordinatenachse geordnet. Hieraus ergibt sich die Teilung des Netzes.

Aufgrund des geringen Aufwands von $\log_2 n$ Operationen bei n Knoten ist die Methode sehr schnell. Der Algorithmus garantiert eine Lastbalance. Die Koppelknotenränder haben oft

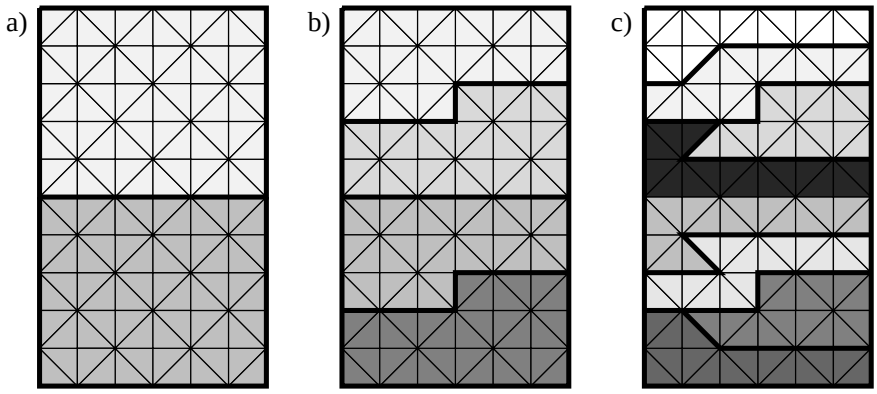


Abbildung 2.15: *Koordinaten-Bisektion: a) zwei Teilgebiete, b) vier Teilgebiete, c) acht Teilgebiete*

eine ungünstige, nicht optimale Länge und es kann im Extremfall zu nicht zusammenhängenden Gebieten kommen (siehe Abbildung 2.15). Hierdurch ergibt sich bei der Finite-Elemente-Berechnung ein unnötig hoher Kommunikationsaufwand [Lämmer 1997].

2.4.1.2 Hauptachsenmethode

Die Hauptachsenmethode (siehe [Farhat und Lesoinne 1993], [Nour-Omid u. a. 1987], [Belkhale und Prithviraj 1990], [Simon 1991], [Chrisochoides u. a. 1994a]) teilt das Gebiet entlang der Hauptachse, die das größere Flächenträgheitsmoment aufweist. Die Achsen können aus den Eigenvektoren I_1, I_2 und I_3 der Matrix

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \tag{2.5}$$

bestimmt werden. Die Matrixelemente lassen sich mit Hilfe des Gaußschen Integralsatzes berechnen. Eine Koordinatentransformation in den Schwerpunkt ist notwendig.
Schwerpunktskoordinaten:

$$x_s = \frac{\int x dA}{\int dA} \tag{2.6}$$

$$y_s = \frac{\int y dA}{\int dA} \tag{2.7}$$

Die Trägheitsmomente berechnen sich dann wie folgt:

$$I_{xx} = \int y^2 dA \tag{2.8}$$

$$I_{yy} = \int x^2 dA \tag{2.9}$$

$$I_{xy} = \int x \cdot y dA \tag{2.10}$$

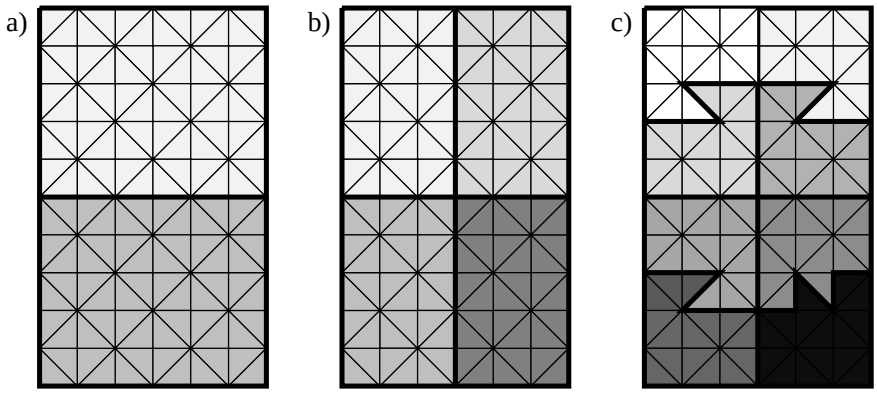


Abbildung 2.16: *Hauptachsen Bisektion: a) zwei Teilgebiete, b) vier Teilgebiete, c) acht Teilgebiete*

Der numerische Aufwand der Partitionierung ist wegen der Berechnung der Integrale wesentlich höher als bei der Koordinaten-Bisektion. Dieser höhere Aufwand rechtfertigt sich durch die wesentlich günstigeren Formen der Teilgebiete, die eine geringere Länge der Außenkante besitzen und daher weniger Kommunikationsaufwand während der nachfolgenden Berechnung bewirken [Lämmer 1997](siehe Abbildung 2.16). Eine Verbesserung der Partitionierung ist mit der Kernighan-Lin-Heuristik [Kernighan und Lin 1970] und dem Algorithmus der Hilfreichen Mengen [Diekmann u. a. 1994] möglich.

2.4.2 Graphenbasierte Algorithmen

Der wichtigste Vertreter der graphenbasierten Algorithmen ist der Greedy-Algorithmus. Der Algorithmus wird in [Farhat 1988] und [Al-Nasra und Nguyen 1991] näher beschrieben.

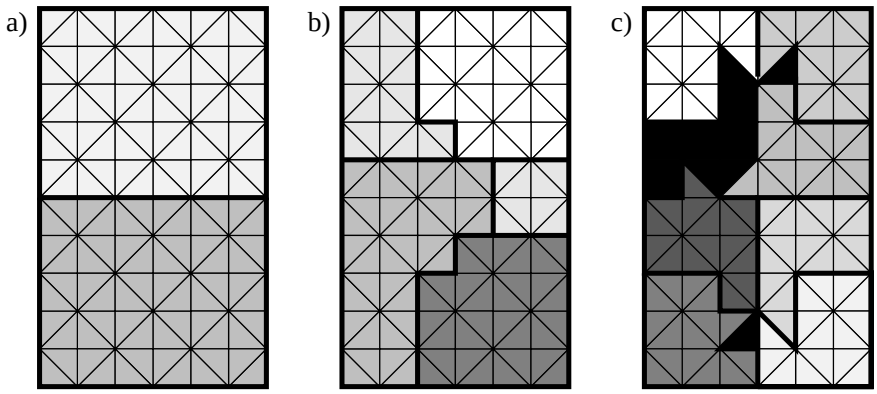


Abbildung 2.17: *Greedy-Algorithmus: a) zwei Teilgebiete, b) vier Teilgebiete, c) acht Teilgebiete*

Der Algorithmus teilt ein Gebiet im Gegensatz zu den rekursiven Algorithmen in beliebig viele Teilgebiete. Der Algorithmus lässt sich wegen der graphenbasierten Vorgehensweise für

eindimensionale, zweidimensionale und dreidimensionale Gebiete anwenden. Durch seinen internen Aufbau und durch Abbruchkriterien bei der Generierung der Teilgebiete garantiert er Teilgebiete mit gleicher Anzahl von Elementen.

Der Algorithmus weist zu Beginn jedem Knoten des Netzes eine Wichtung zu, die der Anzahl der an ihn grenzenden Elemente entspricht. Danach werden nacheinander für jedes Teilgebiet folgende Schritte ausgeführt:

Schritt 1: Der Knoten mit der geringsten Wichtung wird ermittelt.

Schritt 2: Alle nicht vergebenen Elemente, die den Knoten referenzieren, werden dem Teilgebiet zugeordnet.

Schritt 3: Entlang der temporären Gebietsgrenze werden alle Elemente, die noch nicht an ein anderes Teilgebiet vergeben wurden, dem Teilgebiet zugeordnet.

Schritt 4: Die Wichtungen der Knoten der vergebenen Elemente werden reduziert.

Schritt 5: Die Schritte „Zuordnen der Elemente“ und „Reduzierung der Wichtung“ werden solange wiederholt, bis die Anzahl der zugewiesenen Elemente der Anzahl der Elemente dividiert durch die Anzahl der Teilgebiete entspricht.

Für die Zuweisung der Elemente zu den Teilgebieten benötigt man eine Datenstruktur, die in den meisten Finiten-Elemente-Programmen nicht vorliegt. Jeder Knoten muss Zeiger auf die auf ihn referenzierenden Elemente haben und jedes Element sollte ohne Aufwand seine Nachbarelemente finden.

Für die ersten Teilgebiete wird eine Optimierung der Gebietsform und der Kantenlängen erreicht. Diese haben normalerweise eine etwa quadratische Form und somit eine minimale Anzahl von Koppelknoten. Hierdurch wird eine Minimierung der Kommunikation dieser Teilgebiete erreicht. Die zuletzt entstehenden Teilgebiete können aus mehreren Teilen bestehen, wie es in Abbildung 2.17 c) mit dem schwarzen Teilgebiet demonstriert wird. Dieses Teilgebiet besteht aus mehreren Teilen und hat daher eine unnötig hohe Anzahl von Koppelknoten. Das Verhältnis zwischen inneren Knoten und Koppelknoten ist ungünstig. Die Kommunikation mit relativ vielen anderen Prozessoren ist notwendig [Burghardt 1995], [Lämmer 1997].

2.4.3 Algebraische Algorithmen

Einer der wichtigsten Vertreter der algebraischen Algorithmen ist die Spektral-Bisektion, die auch Eigenwert-Bisektion [Pothen u. a. 1990], [Barnes und Hoffmann 1982], [Powers 1988] oder Simon's Methode [Khan und Topping 1996] genannt wird. Zwischen der Spektral-Bisektion und der Finite-Elemente-Berechnung besteht ein direkter mechanischer Zusammenhang. Zur Partitionierung wird die Laplace-Matrix (siehe Abbildung 2.18) aufgestellt. Diese quadratische Matrix hat für jedes Element eine Zeile und Spalte. Auf der Hauptdiagonalen steht die Anzahl der an

ein Element angrenzende Elemente. Für Elemente, die miteinander verbunden sind, steht auf den beiden zugehörigen Nebendiagonalenelementen eine -1. Die restlichen Matrixeinträge sind 0.

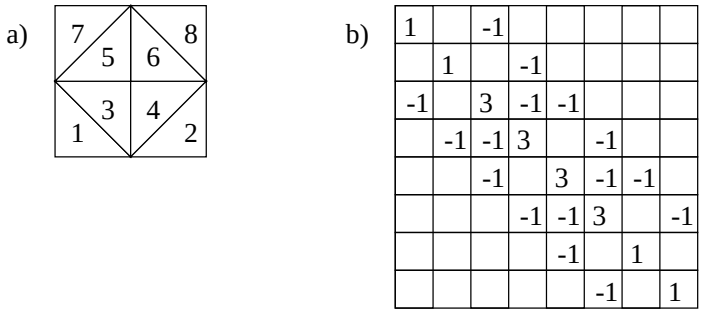


Abbildung 2.18: *Spektral-Bisektion: a) Elemente-Netz, b) Laplace-Matrix*

Durch Berechnung des zweiten Eigenvektors dieser Matrix erhält man die Partitionierung der Elemente in zwei Teilgebiete. Der zweite Eigenvektor wird auch Fiedlervektor genannt [Fiedler 1973], [Fiedler 1975].

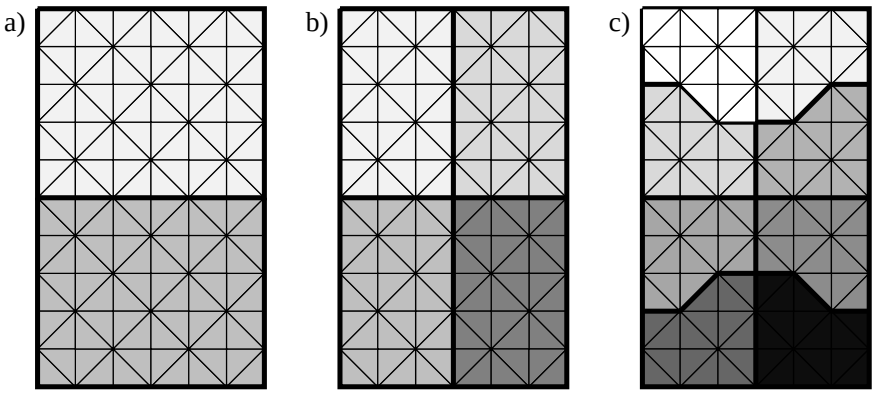


Abbildung 2.19: *Spektral-Bisektion: a) zwei Teilgebiete, b) vier Teilgebiete, c) acht Teilgebiete*

[Simon 1991] zeigt an Beispielen, dass dieses Verfahren das schnellste Verfahren der Klasse der Bisektionsverfahren ist. Die Qualität der Aufteilung ist gut (siehe Abbildung 2.19), wobei die Anzahl der Koppelknoten relativ gering ist [Lämmer 1997].

2.5 Parallele Netzgenerierung

Zur Parallelisierung der Netzgenerierung existiert ein Standardansatz, der unter anderem in [Rypl und Bittnar 1997], [Khan und Topping 1996] und [Wilson und Topping 1996] beschrieben wird. Dieser Ansatz ist für Dreieckelemente, Viereckelemente und Tetraederelemente anwendbar. Netzgeneratoren für Hexaederelemente lassen sich mit dem beschriebenen Ansatz nicht parallelisieren, da Knoten und Kanten auf den Teilgebietsgrenzen vorgegeben werden müssen und zur

Zeit noch kein Algorithmus vorliegt, der diese Eigenschaft erfüllt. Zur Generierung der Netze sind die folgenden Schritte notwendig:

Schritt 1: Ein grobes Hintergrundnetz wird unter Zuhilfenahme des gewählten Algorithmus erzeugt. Dieses Hintergrundnetz muss alle geometrischen Bedingungen, die das endgültige Netz berücksichtigen soll, erfüllen. Des weiteren sollten alle Elemente des temporär vorliegenden Netzes eine ähnliche Größe aufweisen. Ein Beispiel für ein Ausgangsnetz ist in Abbildung 2.20 a) dargestellt.

Schritt 2: Das Ausgangsnetz wird im nächsten Schritt in n Teilgebiete zerteilt. Hierfür werden Algorithmen, wie in Abschnitt 2.3 beschriebenen, verwendet. n ist hierbei Anzahl der zur Verfügung stehenden Prozessoren. (siehe Abbildung 2.20 b))

Schritt 3: Die Teilnetze werden parallel auf den Prozessoren verfeinert. Hierbei ist an den Koppelkanten die Kompatibilität der Teilnetze zu berücksichtigen, die eine gewisse Kommunikation zwischen den Prozessoren erfordert oder durch Vorgabe der Knoten auf den Gebietsgrenzen realisiert werden kann. Eine Glättung innerhalb der Teilgebiete ist notwendig. (siehe Abbildung 2.20 c))

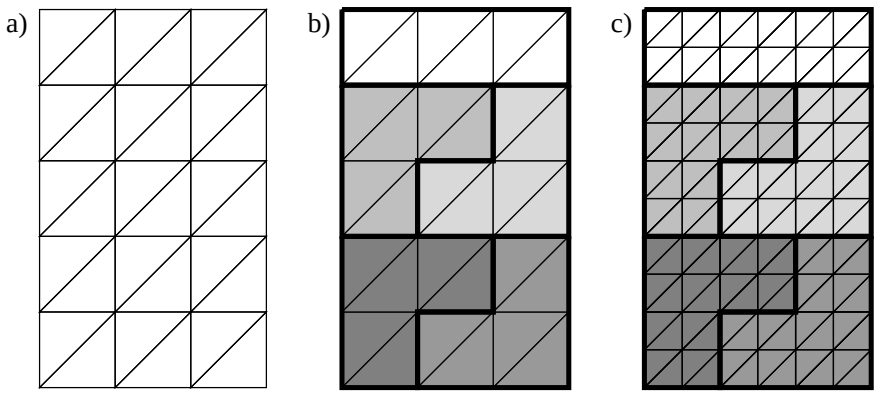


Abbildung 2.20: *Parallele Netzgenerierung: a) grobes Netz, b) Partitioniertes Netz, c) feines Finite-Elemente-Netz*

Eine alternative Methode zur parallelen Generierung von Elementnetzen stellt [Mitchell und Vavasis 1997] für den n-dimensionalen Fall dar, die [Meißner u. a. 1996] für zweidimensionale Netze beschreiben. Hierbei wird die Originalgeometrie rekursiv in mehrere Teile geteilt und eine Randdiskretisierung für jedes Teilgebiet vorgegeben. Eine parallele Vernetzung der Teilgebiete ist anschließend ohne Kommunikation möglich, wodurch eine sehr gute Effizienz der parallelen Netzgenerierung erreicht wird.

2.6 Parallelisierung

2.6.1 Motivation

Die Parallelisierung von Programmen zur Lösung von Problemstellungen hat sich in den letzten Jahren zu einer Standardvorgehensweise zur Verringerung der Rechenzeit im Bereich der Ingenieurwissenschaften, aber auch im Bereich der Quanten- und Plasmaphysik, der Materialwissenschaften und besonders im Bereich der Wettervorhersage entwickelt. Die Parallelisierung eines Problems bedeutet die Aufteilung des Problems in kleine, unabhängige Teilprobleme. Bei der Aufteilung eines Problems in Teilprobleme muss folgendes beachtet werden:

- die Datenstruktur des Programms,
- Datenabhängigkeiten innerhalb des Programms,
- Unabhängigkeit der Daten,
- Berechnungszeiten und Dominanz für die zu separierenden Teile,
- Lastbalance der gleichzeitig abzuarbeitenden Teilprobleme,
- das Laufzeitverhalten der zu parallelisierenden Programmteile.

Eine Parallelisierung macht daher oft ein Reengineering der Software notwendig. Des weiteren ist die Anschaffung eines Parallelrechners eine große Investition, die sich wirtschaftlich rechnen muss. Die geringere Laufzeit und somit schnelleren Antwortzeiten des Rechners rechtfertigen diesen Aufwand nicht immer.

Ein Parallelrechner stellt oft ein Vielfaches der Speicherkapazität eines sequentiellen Rechners zur Verfügung bzw. ein vielfach größerer Speicher ist überhaupt adressierbar. Hierdurch ergeben sich Vorteile, wie z.B. die Beherrschbarkeit von größeren Problemstellungen oder eine genauere Simulation durch höhere Detailliertheit.

In einigen Anwendungsfeldern, wie z.B. der Wettervorhersage, ist es notwendig, dass die Simulationsrechnung schneller abläuft als das physikalische Geschehen dauert, um als Vorhersage funktionieren zu können. Diese hohe Geschwindigkeit der Berechnung ist bei der notwendigen Detailliertheit des Modells nur auf einem Parallelrechner möglich.

Die Steigerung der Rechenkapazität eines sequentiellen Rechners auf eine Rechenleistung, die äquivalent zu der von Parallelrechnern ist, ist aufgrund der Fertigungstechnik im Bereich der Chip-Herstellung bzw. durch physikalische Grenzen, wie z.B. der Lichtgeschwindigkeit als maximale Geschwindigkeit für die Übertragung von Informationen, nicht möglich. Parallelrechner, die diese Chips dann nutzen würden, wären trotzdem um ein Vielfaches schneller.

Bis Ende der achtziger Jahre wurden meist spezielle Parallelrechner mit eigenen Betriebssystemen und speziellen Prozessoren entwickelt. Seit Anfang der neunziger Jahre wird verstärkt versucht, durch Verwendung von Standardkomponenten wie z.B. Personal-Computer-Prozessoren (PowerPC oder Intel Pentium), die Preise der Rechner zu verringern.

Eine andere Entwicklung ist die Nutzung von Rechner-Netzwerken als Parallelrechner. Hierbei werden eine Reihe von Personal-Computern und/oder Workstations zu einem virtuellen Parallelrechner durch eine Software zusammen geschaltet. Zum Austausch von Daten und Instruktionen dient die normale Netzwerkverkabelung der Rechner. Diese Nutzung hat im Gegensatz zu speziellen Parallelrechnern den Vorteil, dass in vielen Firmen nachts die Rechner ungenutzt sind und daher fast keine Kosten entstehen, wenn man sie zu dieser Zeit als Parallelrechner einsetzt.

2.6.2 Klassifikation von Rechnern

Zur Beschreibung der Funktionalität eines Parallelrechners wurden verschiedene Klassifikationen entwickelt. Eine Einteilung der Rechner ist notwendig, da sie sich nach Prozessoranzahl, Prozessorart, Speicherzugriff, Speicherverteilung und Busbreite unterscheiden. Die gebräuchlichste dieser Einteilungen ist die Taxonomie von Flynn. Weitere Klassifikationen, die hier nicht erläutert werden, sind die Erlanger Klassifikation [Haendler 1977] und die Klassifikation nach Shore [Shore 1973]. Die Klassifikationen basieren normalerweise auf dem Rechnermodell von John von Neumann.

2.6.2.1 von-Neumann-Rechner

Die grundsätzliche Aufbau von heutigen sequentiellen Computern geht auf Arbeiten von John von Neumann zurück. Er veröffentlichte erstmals 1946 zusammen mit Arthur W. Burks und Herman H. Goldstine in der Schrift „Prelimary discussion of the logical design of an electronic computing instrument“ den prinzipiellen Aufbau. Eine Beschreibung ist in [von Neumann 1946] und [Taub 1961] zu finden.

Das von Neumannsche Rechnermodell beschreibt ein Datenverarbeitungs-System, das in seiner Struktur unabhängig von den zu bearbeitenden Problemen ist. Prinzipiell lassen sich mit diesem Modell sämtliche berechenbaren Probleme lösen.

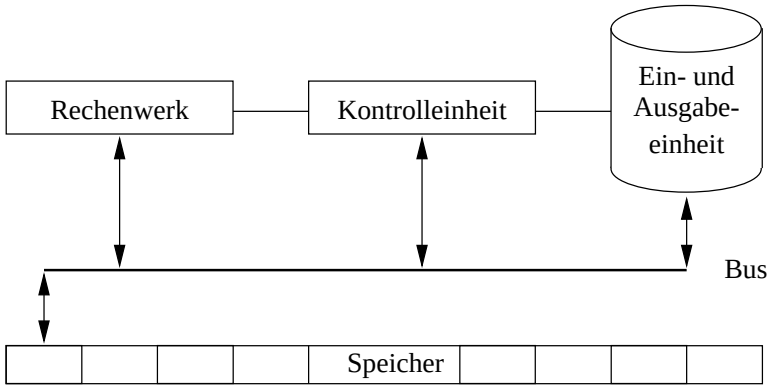


Abbildung 2.21: *von-Neumann-Rechner*

Die Rechnerarchitektur nach John von Neumann besteht aus vier Basiskomponenten:

Komponente 1: Die **Kontrolleinheit**, auch Leitwerk genannt, ist Teil des Hauptprozessors. Sie dient zur sequentiellen Abarbeitung des Anweisungsstroms. Hierdurch übernimmt sie die Steuerung des Rechners. Die verschlüsselten Anweisungen werden von der Kontrolleinheit aus dem Speicher geholt, entschlüsselt und der Rest der Rechners entsprechend instruiert.

Die Operationssteuerung kann auf zwei verschiedene Arten durchgeführt werden. Zum einen gibt es Prozessoren mit festverdrahtetem Befehlssatz. Diese Prozessoren werden RISC-Prozessoren[1] genannt und haben im Vergleich zu den CISC-Prozessoren[2] einen reduzierten Befehlssatz. CISC-Prozessoren haben ein Mikroprogrammsteuerwerk.

Der Vorteil des festverdrahteten Schaltwerks ist die deutlich höhere Geschwindigkeit. Allerdings ist man gezwungen, mit einem kleinen, nicht sehr mächtigen, reduzierten Befehlsvorrat auszukommen. Komplexere Befehle werden durch Maschinenprogramme ersetzt, die häufig hohe Anzahlen von Speicherzugriffen benötigen. Speicher waren in den Anfangszeiten der Datenverarbeitung teuer und langsam. Dies führte zur Entwicklung der CISC-Prozessoren, bei denen die komplexen Maschinenbefehle sich aus den Elementaroperationen zusammensetzten.

Komponente 2: Das **Rechenwerk** dient der Abarbeitung arithmetischer, logischer und transformierender Operationen. Die Rechenoperationen werden von der ALU (arithmetic logical unit) realisiert. Sie stellt ein Schaltnetz dar, bei dem der binäre Ergebniswert nicht nur von den Eingangswerten, sondern zusätzlich von internen Zuständen abhängt. Ein Schaltwerk hängt nur von den Eingaben ab. Es ist die Realisierung einer Schaltfunktion, die jeder der möglichen Wertekombinationen der booleschen Variablen einen booleschen Wert zuordnet.

Komponente 3: Das **Speicherwerk**, das auch Hauptspeicher bzw. interner Speicher genannt wird, hat die Aufgabe, Programme und Daten, mit denen aktuell gearbeitet wird, zu speichern. Auf jede Speicherzelle muss ein gleich schneller, direkter Zugriff möglich sein. Deshalb wird der Hauptspeicher auch als RAM[3] bezeichnet. Die Auswahl einer Informationseinheit im Zentralspeicher (Speicherwerk) erfolgt über eine dem Speicherplatz zugeordnete eindeutige Adresse. Unterteilt ist das Speicherwerk in jeweils gleichlang Teile, die Worte genannt werden.

Komponente 4: Die **Ein- und Ausgabeeinheit** verbindet die Zentraleinheit mit peripheren Geräten, wie z.B. Festplatten, Bildschirm oder Tastatur. Das Ein-/Ausgaberegister im von Neumann'schen Rechnermodell übernimmt das Zwischenspeichern von Adressen und Daten.

[1] RISC: Reduced Instruction Set Computer
[2] CISC: Complex Instruction Set Computer
[3] RAM: Random Access Memory

Die Komponenten eines Rechners werden über ein Leitung, den Bus[4], verbunden. Der Bus ist in mehrere Teile geteilt: Der Adressbus überträgt Adressen und der Datenbus Daten zwischen den verschiedenen Komponenten des Rechners. Der Steuerbus überträgt die zur Steuerung notwendigen Signale und Rückmeldungen zwischen Kontrolleinheit und den restlichen Komponenten.

2.6.2.2 Taxonomie nach Flynn

Die Taxonomie nach Flynn [Flynn 1972] klassifiziert die Rechner nach der Anzahl der Instruktionen bzw. Anzahl der Steuerungseinheiten, der Anzahl der Datenströme und der Verteilung des Speichers. Die Beschreibung der Rechnerart besteht aus vier Buchstaben. Die ersten beiden Buchstaben geben an, ob der Rechner von einem einzigen Anweisungsstrom (SI[5]) oder von mehreren (MI[6]) gesteuert wird. Das andere Buchstabenpaar gibt Auskunft über die Anzahl der Verarbeitungseinheiten. Dies kann wieder eine Einheit sein (SD[7]) oder es können mehrere (MD[8]) Einheiten sein.

SISD-Rechner: Der einfachste Rechner nach Flynn ist ein Rechner mit einer Steuerungseinheit, einem Datenstrom und einem fortlaufenden Speicher (siehe Abbildung 2.22). Dieser Rechnern wird SISD-Rechner genannt und arbeitet sequentielle Programme nacheinander ab. Ein SISD-Rechner entspricht dem klassischen von-Neumann-Rechner. Vertreter dieser Klasse von Rechnern sind die Personal Computer und Workstations mit jeweils einem Prozessor.

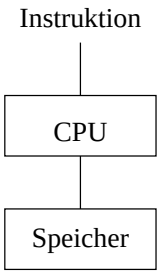


Abbildung 2.22: *SISD Rechner*

SIMD-Rechner: Ein SISD-Rechner kennt nur einen Instruktionsstrom, hat aber mehrere Verarbeitungseinheiten, die den Anweisungsstrom parallel abarbeiten können. Jeder Prozessor kann einen lokalen Speicher besitzen, d.h. der Speicher ist verteilt. In diesem Fall spricht man von einem Distributed-Memory-Rechner, wie er in Abbildung 2.23 abgebildet ist. Der Zugriff der

[4] Bus: lateinisch omnibus = für alle
[5] SI: Single Instruction
[6] MI: Multiple Instruction
[7] SD: Single Data
[8] MD: Multiple Data

einzelnen Prozessoren auf den Speicher kann jeweils unabhängig von den anderen Prozessoren erfolgen. Daten und Signale werden über Links[9] ausgetauscht.

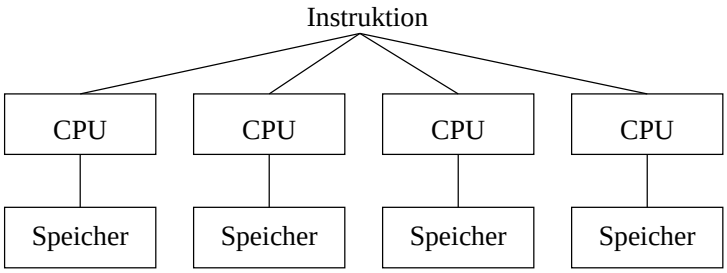


Abbildung 2.23: *SIMD Rechner mit Distributed Memory*

Die verbreitetste Ausführung eines SIMD-Rechners hat einen gemeinsamen Speicher (siehe Abbildung 2.24). Dieser Rechner wird dann als Shared-Memory-Rechner bezeichnet. Der Zugriff auf den Speicher muss in einem solchen Rechner organisiert werden, da jeder Prozessor jeden Teil des Speichers schreiben und lesen darf. Daher kann es zu Zugriffskonflikten kommen. Viele Großrechner der siebziger und achtziger Jahre waren von diesem Typ, der auch als Vektorrechner bekannt wurde. Zu dieser Art von Rechnern sind auch Personal Computer und Workstations mit jeweils mehreren Prozessoren zu zählen.

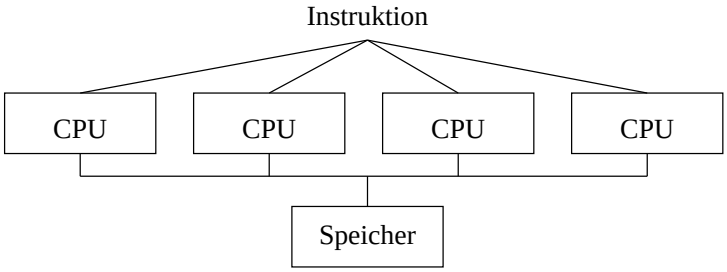


Abbildung 2.24: *SIMD Rechner mit Shared Memory*

MISD-Rechner: Ein MISD-Rechner wendet mehrere Anweisungsströme mit einer Verarbeitungseinheit an. Rechner nach diesem Modell sind unzweckmäßig und werden daher nicht verwendet.

MIMD-Rechner: Bei einem MIMD-Rechner steuern mehrere Anweisungsströme eine gleiche Anzahl von Verarbeitungseinheiten. Diese arbeiten die Anweisungsströme parallel ab. Ein solcher Rechner kann entweder einen verteilten Speicher (Distributed Memory, siehe Abbildung 2.25) oder einen gemeinsamen Speicher (Shared Memory, siehe Abbildung 2.26) besitzen.

[9] Links: engl. Kanäle

In einem MIMD-Rechner mit einem gemeinsamen Speicher können alle Prozessoren auf den gesamten Speicher zugreifen, bzw. alle Daten lesen und schreiben. Daher ist eine Organisation der Schreib- und Leseprozesse notwendig. Ein Austausch der Daten ist nicht erforderlich.

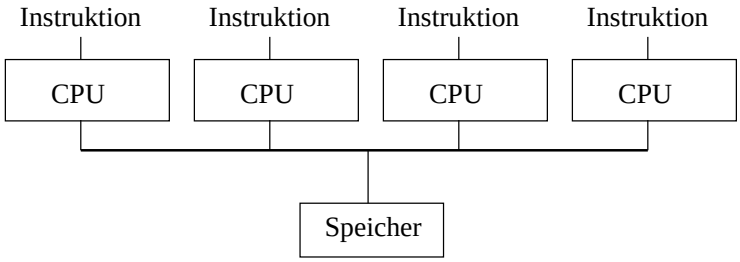


Abbildung 2.25: *MIMD Rechner mit Shared Memory*

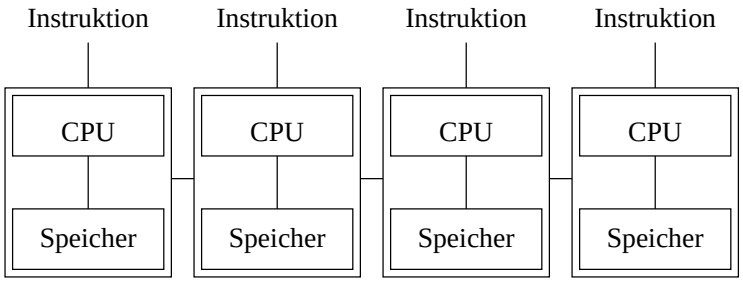


Abbildung 2.26: *MIMD Rechner mit Distributed Memory*

Der Austausch in einem MIMD-Rechner mit verteiltem Speicher wird mit Hilfe von Kanälen realisiert. Der Zugriff auf den Speicher ist nur lokal möglich und muss daher nicht global organisiert werden.

Die leistungsstärksten Rechner sind heutzutage MIMD-Rechner. Vertreter hierfür sind z.B. die Cray T3E, IBM SP Power3 oder die Intel ASCI Red. Einfache Personal Computer oder Workstations, die in einem Netzwerk verschaltet sind, lassen sich als MIMD-Rechner nutzen. Hierbei spricht man von einem Cluster. Da in vielen Firmen ein solches Netzwerk vorhanden ist, bietet sich die Entwicklung von Software für diesen Rechnertyp an.

2.6.2.3 Speicherzugriff

Der Speicherzugriff der verschiedenen Rechnerarchitekturen ist unterschiedlich [Bode 1996]. Die einfachsten Zugriffsmaschinen sind die sogenannten Multicomputer oder NORMA[10]-Maschinen. Hierbei kann der Prozessor nur auf seinen eigenen lokalen Speicher zugreifen. Jegliche Kommunikation wird durch explizites Message Passing (siehe Abschnitt 2.6.4) durchgeführt. Die verbleibenden Klassen von verteilten Speichersystemen haben genau einen einzigen, globalen, physikalischen Adressraum. Diese Maschinen werden oft als Shared-Memory Multiprozessoren

[10]NORMA: No Remote Access

bezeichnet und lassen sich nach dem Daten-Parallelen-Programmierparadigma (siehe Abschnitt 2.6.5) programmieren. Maschinen, die keinen Cachespeicher besitzen und auf den gesamten Speicher zugreifen können, nennt man UMA[11]-Multiprozessoren. Der gemeinsame Speicher kann in Teile zerlegt und den einzelnen Prozessoren zugewiesen werden. Ist das Zugriffsverhalten auf die einzelnen Speicherbereiche unterschiedlich, nennt man die Rechner NUMA[12] Multiprozessoren. Der gesamte Speicher kann auch als verteilter Cache-Speicher interpretiert werden, wobei der Zugriff auf entfernten Speicher durch Kopieren, was sich als Kommunikation interpretieren lässt, realisiert wird. Hierbei spricht man von COMA[13]-Maschinen.

2.6.3 Parallelisierungsparadigma

Das verwendete Speichermodell wirkt sich auf die Programmierung des Rechners aus. Für die verschiedenen Speicher- und Speicherzugriffsmodelle existieren daher verschiedene Programmierparadigmen. Das Programmierparadigma, das besonders für Rechner mit gemeinsamem Speicher entworfen wurde, ist die Daten-Parallel Programmierung. Message Passing, d.h. das Austauschen von Datenpaketen und Signalen, ist das andere Programmierparadigma, das für Rechner mit verteiltem Speicher optimal ist.

Die Programmierung eines Parallelrechners kann auf zwei verschiedene Arten erfolgen. Zum einen können sequentielle Programme unter Zuhilfenahme von Programmbibliotheken erweitert werden. Hierfür sind Aufrufe von Unterprogrammen und Funktionen einzubauen. Die andere Möglichkeit besteht in der Verwendung einer speziellen Programmiersprache für Parallelrechner, die automatisch die Parallelität unterstützt, oder die eines parallelisierenden Compilers.

2.6.4 Message-Passing

Das Programmiermodell Message-Passing wurde speziell für MIMD-Rechner mit verteiltem Speicher entworfen. Die Grundidee dieses Modells liegt in verschiedenen, gleichzeitig ablaufenden Prozessen, die einen voneinander unabhängigen Speicher besitzen. Ein direkter Zugriff auf die Daten eines anderen Prozesses ist nicht möglich [Thole 1996]. Daher ist es notwendig Nachrichten auszutauschen. Diese Nachrichten können zum einen aus Signalen zur Steuerung des Ablaufes und zum anderen aus Daten, die von einem Prozess zu einem anderen gelangen müssen, bestehen. Die Prozesse, die parallel ablaufen, können komplett verschieden sein.

Die wichtigsten Vertreter des Message-Passing sind das Message-Passing-Interface (MPI) und die Parallel Virtual Machine (PVM). Die Parallelisierung eines Programmes geschieht durch Aufteilen des Prozesse in Teilprozesse. Diese Teilprozesse müssen in der Regel miteinander kommunizieren, wofür Aufrufe von Kommunikationsroutinen in das Programm eingefügt werden müssen. Zusätzlich ist eine Verwaltung der Prozesse, die einen gewissen Overhead erzeugen,

[11] UMA: Uniform Memory Access
[12] NUMA: Non-Uniform Memory Access
[13] COMA: Cache Only Memory Architecture

notwendig. Für Cluster aus Workstations oder Personal Computern stellt das Message-Passing das optimale Programmiermodell dar.

Eine Message-Passing Bibliothek muss dem Programmierer eine Reihe von Funktionalitäten zur Verfügung stellen[Thole 1996]:

Starten: Die parallel ablaufenden Prozesse müssen gestartet werden. Dies kann entweder statisch oder dynamisch erfolgen.

Gruppierung: Für eine Reihe von Lösungsansätzen ist es notwendig, Prozessoren zu gruppieren. Hierfür sollte eine Message-Passing-Programmierbibliothek Funktionalitäten liefern.

Benennen: Es ist notwendig, dass jeder Prozess einen Namen für jeden der anderen Prozesse ermitteln kann. Dieser Name kann z.B. eine Nummer sein und ist unter anderem zur Kommunikation notwendig.

Kommunikation: Es müssen Funktionalitäten zum Austausch von Daten und Signalen vorhanden sein. Die wichtigsten Kommunikationsarten sind [Lämmer 1997]:

- Blockierende Kommunikation
- Nichtblockierende Kommunikation
- Mailbox
- Interrupt

Beenden: Beim Beenden eines Prozesses müssen das System und die anderen Prozessoren erkennen, dass der eine Prozess beendet ist.

2.6.4.1 Parallel Virtual Machine

PVM [14] ermöglicht es, verschiedene, vernetzte UNIX-Rechner zu einem virtuellen Parallelrechner zu verschalten[Geist u. a. 1994].

Die Entwicklung von PVM begann im Sommer 1989 am Oak Ridge National Laboratory. Version 1.0 wurde nur intern im Institut verwendet. Version 2.0 wurde in Zusammenarbeit mit dem Institute Emory University, der University of Tennessee, der Carnegie Mellon University und dem Pittsburgh Supercomputer Center entwickelt und im März 1991 veröffentlicht. Eine komplette Neuentwicklung mit einer Reihe von Änderungen kam mit der Version 3.0 im Februar 1993. Die aktuelle Version 3.4.3 wird weltweit eingesetzt und ist für eine große Anzahl verschiedener Rechner verfügbar.

Das Softwaresystem PVM in der Version 3 erlaubt es, ein Netzwerk von heterogenen UNIX-Computern wie einen einzigen MIMD-Rechner mit verteiltem Speicher zu verwenden. Zur Kommunikation der einzelnen Rechner untereinander dient das LAN[15]oder WAN[16]. PVM nimmt

[14]PVM: Parallel Virtual Machine
[15]LAN: Local Area Network
[16]WAN: Wide Area Network

dem Benutzer dabei die explizite Netzwerkprogrammierung und die Datenkonvertierung zwischen verschiedenen Architekturen ab. Die eigentliche Parallelisierung des Algorithmus muss vom Benutzer in C/C++, Fortran oder Java selbst übernommen werden. Zusätzlich ermöglicht PVM eine dynamische Konfiguration der virtuellen Maschine während der Laufzeit sowie die Programmierung dynamischer Prozessgruppen.

PVM erlaubt die gleichzeitige Verwendung der verschiedensten Rechner. Diese Heterogenität erstreckt sich auf folgende Eigenschaften:

Hardwarehersteller: Die verwendeten Rechner können unterschiedliche Hersteller haben und verschiedene Architekturen aufweisen. Zum einen werden Rechner wie Workstations oder Personal Computer unterstützt, zum anderen ist PVM für Parallelrechner wie Intel ASCI Red oder Cray T3E erhältlich.

Betriebssystem: Das verwendete Betriebssystem muss ein UNIX-System sein. Die gleichzeitige Versendung verschiedener UNIX-Systeme auf den verschiedenen Rechnern, wie Solaris, AIX, Linux usw., ist erlaubt.

Datenformate: Die Datenformate der einzelnen Rechner dürfen unterschiedlich sein. Die Konvertierung bei der Kommunikation zwischen den Formaten wird von PVM übernommen.

Geschwindigkeiten: Die Rechengeschwindigkeit der verwendeten Rechner darf stark unterschiedlich sein. Der Programmierer muss dies durch eine dynamische Lastverteilung ausgleichen.

PVM eignet sich durch die beschriebenen Eigenschaften bezüglich der Heterogenität vor allem als System zur Nutzung von Rechner-Clustern als Parallelrechner. Hierdurch ist PVM optimal für den Einsatz in Firmen, die die Anschaffung eines teuren Parallelrechners scheuen und ein Netzwerk von Personal-Computern oder Workstations besitzen. Unterstützt wird dies durch besondere Funktionalitäten zur Steuerung und Überwachung der Prozesse.

Oft werden Rechner verschiedener Leistungsklassen zu einem Parallelrechner zusammengeschaltet. Man sollte dabei sehr große Sorgfalt auf die Erhaltung der Rechenlast-Balance legen. Diese Balance wird häufig durch andere Benutzer gestört, die intensiv von einem Rechner der virtuellen Maschine Gebrauch machen oder große Datenmengen im Netzwerk versenden.

Eine Reihe von Werkzeugen ist entwickelt worden, um dem Programmierer die Arbeit zu erleichtern. Die graphische Oberfläche „xpvm“ visualisiert die Kommunikationen, den Programmablauf, Kommunikationszeiten und Wartezeiten. Debuggen wird durch den Aufruf von Debuggern der Compilerhersteller ermöglicht.

Programmierung: PVM besteht aus verschiedenen Komponenten: einem Dämon, Bibliotheken und der Console. Der PVM-Dämon ermöglicht die Kommunikation zwischen den Rechnern der virtuellen Maschine, steuert und überwacht die Prozesse. Die PVM-Bibliotheken enthalten

die Funktionen und Unterprogramme, die für das Kommunizieren, Starten von Prozessen und zur Koordination gebraucht werden. In das PVM-Programm müssen diese Bibliotheken eingebunden werden. Mit der PVM-Console wird der virtuelle Rechner konfiguriert und überwacht. Die Oberfläche dient auch zum Starten der Prozesse.

2.6.4.2 Message Passing Interface

MPI[17] war der erste Versuch einen Standard für Message-Passing zu definieren. Das erste Treffen zur Schaffung eines Message-Passing Standards war im April 1992. In diesem MPI-Forum waren Rechnerhersteller, Softwareentwickler und Universitäten vertreten. Der erste Standard von MPI wurde im im Mai 1994 veröffentlicht[Forum 1993b]. Bei der Schaffung des Standards sollten die sinnvollsten Grundideen der verschiedenen vorher vorhandenen Systeme übernommen werden. Ziele waren:

- Entwicklung einer Programmierschnittstelle. Hierbei war speziell an Schnittstellen von C und FORTRAN gedacht.
- Eine effiziente Kommunikation sollte möglich sein. Hierfür sollte das Kopieren von Speicher, wie es z.B. bei PVM notwendig ist, weitestgehend vermieden werden.
- Um unnötigen Overhead zu vermeiden, sollte es möglich sein, gleichzeitig zu kommunizieren und zu rechnen.
- Die Kommunikation sollte einfach und sicher sein. Dies bedeutet, dass der Nutzer sich nicht um interne Kommunikationsroutinen des Betriebssystems kümmern muss und die Kommunikation fehlerfrei ausgeführt wird.
- Eine Anlehnung an die bereits bestehenden Systeme wurde angestrebt, um eine Konvertierung alter Programme zum neuen Standard einfach zu gestalten.
- MPI sollte als Bibliotheksfunktionen dem Programmierer bereitgestellt werden.

Es wurde die Entwicklung eines Standards zur einfacheren Portierbarkeit von Software notwendig. Früher ging man davon aus, dass die Hardware länger bestehen würde als die Software. Daher war die Entwicklung von spezieller Software für einen Rechner zulässig. Anfang der neunziger Jahre erkannte man, dass die Software weiterentwickelt wird und meistens noch benutzt wird, während die Hardware schon lange nicht mehr verwendet wird. Hieraus entwickelte sich das Interesse an einer einfachen Portierbarkeit der Software von einem Parallelrechner zu einem anderen.

Entsprechend dem Standard MPI wurde das MPICH von William Gropp und Ewing Lusk am Argonne National Laboratory, USA, als portable Installation des MPI-Standards

[17] MPI: Message Passing Interface

entwickelt[Gropp u. a. 1994]. Hierzu gehören einige Erweiterungen, u.a. die MPE Bibliothek[18], die eine Reihe von zusätzlichen Funktionen enthält, wie z.B. zur Ausgabe über X-Windows, ferner Hilfsprogramme zum Übersetzen, Laden und Starten von MPI-Programmen.

MPI enthält keine Möglichkeiten der Datenkonvertierung während des Austauschens. Daher ist die Verwendung von MPI nur auf speziellen Parallelrechnern und in Rechner-Clustern möglich, die aus Maschinen mit gleicher Architektur bestehen.

Eine dynamische Konfiguration des Netzwerkes oder die Überwachung von Prozessoren durch einen anderen sind in MPI nicht vorgesehen.

Programmierung: MPI in der Variante MPICH besteht aus Bibliotheken mit den Unterprogrammen, die in das ausführbare Programm eingebunden werden müssen.

2.6.5 Daten-Parallele-Programmierung

Das Programmierparadigma des *Daten-Parallelen-Programmierens* wurde speziell zur Programmierung von SIMD-Rechnern und MIMD-Rechnern mit gemeinsamen Speicher entwickelt. Hierbei kontrolliert ein einziges Programm den Ablauf der verteilten Operationen auf allen Prozessoren. Unterstützt werden hierbei hauptsächlich Feld-Operationen. Die Verteilung der Arbeit auf die Prozessoren und das Einfügen von Kommunikationen wird hierbei vom Compiler durchgeführt. Eine vollautomatische Parallelisierung ist allerdings nicht möglich. Die Steuerung des Compilers durch den Programmierer ist in Form von Compiler-Direktiven notwendig.

Das Programmiermodell der Daten-Parallelen-Programmierung hat folgende Eigenschaften:

- Ein Programm kontrolliert den Ablauf der Operationen. Der Programmierer muss sich nur mit einem Anweisungsstrom auseinandersetzen.
- Der Speicher wird als globaler Speicher behandelt, der nur einen Adressraum hat. Der Programmierer muss sich daher nicht um den Austausch von Daten zwischen den einzelnen Speichern der Prozessoren kümmern.
- Eine „lockere Synchronisation“ wird angestrebt, d.h. es ist nicht notwendig, dass zu einem Zeitpunkt alle Prozessoren dieselbe Instruktion ausführen. Notwendig ist die Ausführung desselben Programmabschnittes (z.B. gleiche Schleife) zu einem Zeitpunkt.
- Feld-Operationen werden automatisch parallel durchgeführt. Unabhängigkeiten müssen teilweise durch den Programmierer aufgezeigt werden.
- Zur Parallelisierung sind nur geringe Kenntnisse des Programmcodes notwendig.

Beim Entwurf einer Parallelisierung liegt der Hauptunterschied zwischen Message-Passing und der Daten-Parallelen-Programmierung darin, dass bei Message-Passing das komplette

[18] MPE: Multiple Processing Environment

Programm in datenunabhängige Teile zerlegt werden muss, wobei bei der Daten-Parallelen-Programmierung die Struktur des sequentiellen Programms erhalten bleibt und durch Direktiven erweitert wird.

High Performance Fortran: Der wichtigste Vertreter der Daten-Parallelen-Programmierung ist HPF [19] und stellt eine Erweiterung von FORTRAN 90[Forum 1993a] dar.

Die Entwicklung von HPF begann 1991 mit der Gründung des High Performance Fortran Forum. Detaillierte Arbeiten am Standard HPF wurden von einer Gruppe von etwa 40 Personen, die sich aus Mitgliedern von Hardwarefirmen, Softwarefirmen und Universitäten zusammensetzte, durchgeführt. Der Standard wurde im Januar 1993 verabschiedet. Die Spezifikation 1.0 von HPF wurde im Mai 1993 veröffentlicht.

Ziele von HPF: HPF wurde im Gegensatz zu MPI und PVM nicht als Programmbibliothek, sondern als Compiler konzipiert. Der Programmierer von HPF-Programmen muss das FORTRAN-Programm durch Compiler-Direktiven ähnlich den aus C/C++ bekannten `#define` oder dem `#ifdef` zur Unterstützung des Compilers erweitern. Es sollte ein System erreicht werden, das maximale Performance mit SIMD-Rechnern erreicht. Die Programmentwicklung sollte unabhängig von der Hardware sein. Spezielle Eigenschaften der einzelnen Rechner durften daher nur vom Compiler und nicht vom Programmierer genutzt werden. Die Portierung von bestehenden Programmen sollte möglichst einfach und schnell möglich sein und eine möglichst große Kompatibilität zu anderen Standards (FORTRAN-90) sollte erhalten bleiben. HPF-Programme sollten sich für sequentielle Rechner mit FORTRAN-Compilern direkt ohne Änderungen übersetzen lassen. Eine weitere Forderung war die einfache Implementierung von HPF.

Spracheigenschaften von HPF: In HPF wurde der komplette FORTRAN-90 Sprachumfang integriert. Dies schließt neue Kontrollstrukturen, optionale Funktionsargumente, benutzerdefinierte Datentypen und Pointer ein. Die dynamische Speicherverwaltung ist eine weitere Verbesserung von FORTRAN-90 gegenüber den Vorgängern. Feldanweisungen werden, wenn möglich, parallel ausgeführt. Die Konvertierung von vielen alten FORTRAN-Programmen ist problematisch. In FORTRAN-77 und FORTRAN-90 werden Verknüpfungen von Speicherbereichen mit den Befehlen `COMMON` und `EQUIVALENCE` realisiert. Diese Zuordnungen sind dynamisch und durch HPF nicht auflösbar. Speicherbereiche, die mit den beiden Befehlen bearbeitet werden, können von HPF nicht verteilt werden. Eine Parallelisierung durch HPF ist dann nicht möglich.

Compilerdirektiven: Die Steuerung des Compilers und damit der Parallelisierung geschieht mit Compiler-Direktiven. Direktiven sind Angaben, die den Compiler auf Datenun-

[19] HPF: High Performance Fortran

abhängigkeiten, angestrebte Verteilung von Feldern oder die Prozessorverteilung aufmerksam machen. Die Compilerdirektiven beginnen mit `!HPF$`, `CHPF$` oder `*HPF$`. Bei der Verwendung eines sequentiellen Compilers werden die Direktiven als Kommentare aufgefasst und ignoriert.

2.6.5.1 Java

Verteilte Anwendungen benötigen Kommunikation zwischen den Komponenten, um in Rechnernetzen verteilt arbeiten zu können. Java bietet zwei Mechanismen, die die Kommunikation ermöglichen: RMI (Remote Method Invocation) und JavaIDL über die Common Object Request Broker Architecture (CORBA).

RMI ist Bestandteil von Java seit dem JDK 1.1 und ermöglicht das Starten von „Remote Procedure Calls“ [Wollrath und Waldo 1999]. Hierfür stellt ein Server Methoden zur Verfügung, mit denen Objekte manipuliert werden können. Die hierbei aufgerufenen Unterprogramme auf anderen Rechnern müssen ebenfalls in Java geschrieben sein. Der Aufbau einer verteilten Anwendung unter Benutzung von RMI ist folgendermaßen: Der Server stellt seine Dienste über eine Schnittstelle, einem Java-Interface, das Teil des Servers ist und mittels Namens-Diensten bekannt gemacht wird, zur Verfügung. Hierauf können sowohl der Server als auch der Client zugreifen. Durch Zugriff auf die Namens-Dienste kann der Client Informationen über die Schnittstelle erhalten und Methoden auf dem Server starten. Zur Übertragung der Daten werden die Objekte serialisiert.

Die Architektur von RMI ist in Abbildung 2.27 dargestellt.

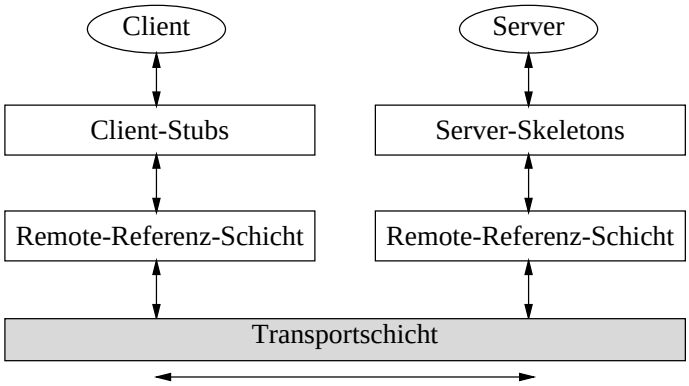


Abbildung 2.27: *Architektur von RMI*

Die einzelnen Teile von RMI arbeiten folgendermaßen zusammen:

- Das auf dem Server laufende Objekt wird durch den Client-Stub auf dem Client vertreten. Alle Aufrufe leitet der Client-Stub an das Server-Objekt weiter. Hierfür werden alle übergebenen Parameter und Objekte serialisiert.

- Auf der Serverseite baut der Server-Skeleton als Gegenstück zum Client-Stub die serialisierten Parameter und Objekte wieder zusammen.
- Der Transport der Daten zwischen Server und Client wird in der Transportschicht durchgeführt. Die Umwandlung der Daten geschieht hierbei in der Remote-Referenz-Schicht.

RMI lässt sich für verteilte Anwendungen leicht einsetzen und kann die Vorteile von Java, wie z.B. die Plattformunabhängigkeit, nutzen. Daten können problemlos zwischen einzelnen Prozessen ausgetauscht werden, solange sie serialisiert werden können. Die Implementation von MPI für Java verwendet daher auch die Funktionalität von RMI. Ein Nachteil von RMI, der bei der Verwendung von Java im Bereich des Hochleistungsrechnens nicht so gravierend ist, ist die Beschränkung auf Java-Objekte. Ein größerer Nachteil bei der Verwendung von Java und RMI im Bereich des Hochleistungsrechnens liegt im Geschwindigkeitsnachteil von Java gegenüber Programmiersprachen wie C++ und FORTRAN. Hier zeigen allerdings die neusten Entwicklungen im Bereich der Compiler für Java, dass eine ähnliche Performance wie unter FORTRAN in den nächsten Jahren erreicht werden kann [Artivagas u. a. 1999] [Maassen u. a. 2000].

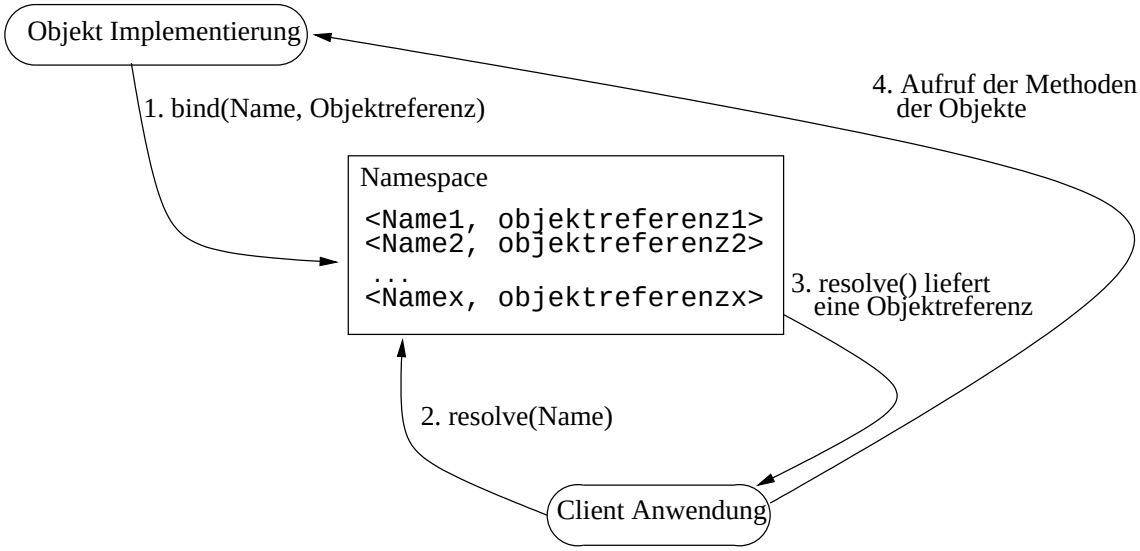


Abbildung 2.28: *Name-Service von CORBA [Inprise 1999]*

CORBA ist die Abkürzung für Common Objekt Request Broker Architecture und ist ein Standard der Open Management Group (OMG). Objekte, die in einer beliebigen Sprache geschrieben sind, können mittels CORBA verwendet werden. Seit dem JDK 1.2 existiert die CORBA-Anbindung in Java in Form von JavaIDL [Inscore 1999]. Die Kommunikation funktioniert ähnlich wie bei RMI. Ein Server stellt eine Schnittstelle zur Verfügung, die bei einem Namensdienst angemeldet (`bind`) wird und dann für Anfragen bereit steht (Abbildung 2.28). Der Client fragt (`resolve`) beim „Name Service“ nach einem Serverobjekt und kann dann Methoden des Servers aufrufen. Die Definition der Schnittstelle geschieht mittels der „Interface Definition

Language“ (IDL) und wird nach Java übersetzt. Der „Object Request Broker“ (ORB) übernimmt die eigentliche Kommunikation zwischen den Objekten.

2.6.6 Leistungsdaten

Die Parallelisierung eines Programms stellt einen zusätzlichen Aufwand gegenüber der Nutzung eines sequentiellen Programms dar. Die Anschaffung eines speziellen Parallelrechners ist in der Regel sehr teuer. Dieser Aufwand sollte durch Zeitgewinn bei der Anwendung eines Programms gerechtfertigt werden. Hierfür sind verschiedene Leistungsdaten entwickelt worden. Die wichtigsten Leistungsdaten einer Parallelisierung sind der Speed-up[20] und die Effizienz.

2.6.6.1 Speed-up

Der Speed-up eines parallelen Programms ist das Verhältnis der Zeit für die sequentielle Berechnung zur Zeit für die parallele Berechnung mit n Prozessoren. Definiert ist der Speed-up als:

$$S(n) = \frac{T_{seq}}{T_n} \tag{2.11}$$

wobei T_{seq} die Zeit bei sequentieller Berechnung ist und T_n die Zeit zur Berechnung mit n Prozessoren. Der optimale Wert des Speed-up ist $S(n) = n$.

2.6.6.2 Effizienz

Eine andere Maßzahl zur Bewertung einer Parallelisierung ist die Effizienz. Diese gibt das Verhältnis zwischen dem Speed-up und der Anzahl der Prozessoren an:

$$E(n) = \frac{\frac{T_{seq}}{T_n}}{n} = \frac{S(n)}{T_n} \tag{2.12}$$

Die Effizienz, deren Optimalwert 1 ist, gibt den Grad der Ausnutzung eins Parallelrechners an.

[20] Speed-up: engl: Beschleunigung

3 Netzgenerierung

Eine Reihe von physikalischen Problemstellungen lässt sich mit Hilfe von Differentialgleichungen beschreiben. Zur numerischen Lösung der Gleichungen sind verschiedene Methoden wie z.B. die Finite-Elemente-Methode, die Finite-Differenzen-Methode oder die Finite-Volumen-Methode entwickelt worden. Diese numerischen Verfahren benötigen eine Diskretisierung, d.h. eine Zerlegung in kleine Elemente, des zu simulierenden Systems.

Die Netzgenerierung ist ein Prozess, bei dem ein System in viele kleine Teilsysteme, die Elemente genannt werden und sich nicht überlappen dürfen, zerteilt wird. Ebene Systeme können in Dreieckelemente oder Viereckelemente, räumliche Gebilde in Tetraeder oder Hexaeder zerteilt werden.

Die Auswahl, ob strukturierte oder unstrukturierte Netze verwendet werden, hängt von der Problemstellung und dem angestrebten Lösungsweg ab. Die Verwendung der Methode der Finiten Differenzen verlangt normalerweise strukturierte Netze, während bei der Methode der Finiten Elemente strukturierte und unstrukturierte Netze verwendet werden können. Der Speicheraufwand zur Generierung von strukturierten Netzen ist normalerweise wesentlich geringer und die Komplexität der Netzgenerierungsalgorithmen ist bei unstrukturierten Netzen höher. Unstrukturierte Netze bieten bessere Möglichkeiten die Geometrie genauer abzubilden. Lokale Verfeinerungen des Netzes sind in unstrukturierten Netzen ebenfalls einfacher zu realisieren. Bei lokal verfeinerten Netzen ist die Anzahl der entstehenden Elemente in unstrukturierten Netzen wesentlich geringer als in strukturierten Netzen. Hierdurch wird die Handhabbarkeit und die Rechenzeit der nachfolgenden Berechnung stark beeinflusst.

3.1 Unstrukturierte Netze

Ein Anwendungsgebiet von unstrukturierten Netzen ist die Finite-Elemente-Methode. Unstrukturierte Netze sind durch die nicht konstante Anzahl von an einen Knoten angrenzenden Elemente definiert. Wesentlicher Vorteil der unstrukturierten Netze im Vergleich zu den strukturierten Netzen ist die Flexibilität der Algorithmen bezüglich der zu vernetzenden Geometrie. Die Steuerung der Elementgröße ist wesentlich einfacher und lokal benötigte Verfeinerungen sind einfacher und mit einer wesentlich geringeren Elementanzahl zu realisieren.

Unstrukturierte Netze können für ein-, zwei- und dreidimensionale Geometrien verwendet werden. Die entstehenden Elemente entsprechen den benötigten, in Abschnitt 2.2.2.1 beschriebenen Finiten-Elementen, wie z.B. Dreiecken, Vierecken, Tetraedern oder Hexaedern.

3.1.1 Netzbasierte Methode

Eine sehr einfache Methode zur automatischen Generierung unstrukturierter Netze ist die Netzbasierte Methode, die auch Überlagerungsverfahren genannt wird. Sie ist sowohl für zwei- als auch für dreidimensionale Geometrien anwendbar [Schneiders 1996a] [Fuchs 1998]. Man kann sowohl Dreieck- und Viereckknetze als auch Tetraeder- oder Hexaedernetze mit der Technik erstellen.

Ein regelmäßiges Netz aus Elementen wird über die Geometrie gelegt und alle aus der Geometrie herausragenden Elemente abgeschnitten oder angepasst. Im Inneren ist die Qualität der Netze sehr gut, die Randelemente sind von schlechterer Qualität. Eine Verbesserung der Qualität dieser Elemente ist mit Glättungsalgorithmen (siehe Abschnitt 3.2) möglich. Lokale Netzverfeinerungen sind in dieser Technik nicht vorgesehen. Eine Vorgabe der Diskretisierung des Geometrierandes bzw. der Geometrieoberfläche ist nicht möglich. Hierdurch wird auch das Zusammenfügen von mehreren Netzteilen verhindert. Der numerische Aufwand des Verfahrens und damit die Zeit zum Generieren eines Netzes ist sehr gering.

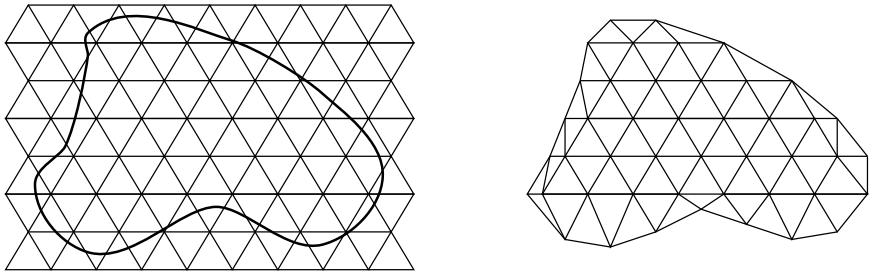


Abbildung 3.1: *Netzbasierte Methode*

Eine Abwandlung des Verfahrens ist die quadtree-basierte Netzgenerierung für 2D Netze sowie die octree-basierte Netzgenerierung für Tetraeder- und Hexaedernetze [Schneiders u. a. 1996a]. Zur Anpassung des regelmäßigen inneren Netzes an die Randgeometrie werden entlang der angestrebten Berandung alle Elemente entfernt. Die entstandene Lücke wird mit der Isomorphie-Technik [Dhondt 1999] vernetzt.

3.1.2 Advancing-Front-Methode

Die Advancing-Front-Methode eignet sich sowohl zur Vernetzung von zweidimensionalen als auch zur Vernetzung von dreidimensionalen Geometrien [Schöberl 1997]. Lokale Verfeinerungen des Netzes sind einfach möglich. Im ebenen Fall können sowohl Dreieckelemente als auch Viereckelemente erzeugt werden. Die Advancing-Front-Methode zur Generierung von Viereckelementen wird *Paving-Algorithmus* genannt. Tetraederelemente können zur Vernetzung dreidimensionaler Gebilde verwendet werden, für Hexaederelemente scheitert zur Zeit das Verfahren häufig beim Vernähen der Restgeometrie [Schneiders 1996a].

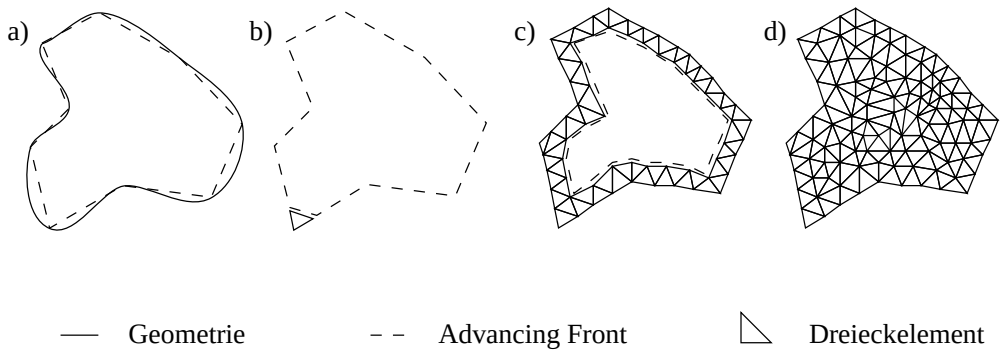


Abbildung 3.2: *Advancing-Front-Methode*

Die Geometrie wird in einem initialen Schritt entsprechend der Dichtefunktion zur Beschreibung der angestrebten Elementkantenlänge diskretisiert (Abbildung 3.2 a)). Die diskretisierte Geometrie stellt die initiale Front dar. Im zweidimensionalen Fall besteht diese initiale Front aus einem Polygonzug, im dreidimensionalen Fall bei der Generierung von Tetraedern aus einem Dreiecknetz und bei der Generierung von Hexaedern aus einem Viereckнetz.

Die Elemente werden im nächsten Schritt der Generierung entsprechend Abbildung 3.2 b) und c) entlang der Front so eingefügt, dass mindestens eine Kante des Elementes auf der Front liegt. Anschließend wird die Front um das neu entstandene Element herum verschoben. Dieser Vorgang wird solange wiederholt, bis die Front nicht mehr vorhanden ist (Abbildung 3.2 d)). Abhängig von der Ausgangsgeometrie können mehrere Fronten vorhanden sein oder zwischenzeitlich entstehen.

Die Advancing-Front-Methode liefert automatisch geglättete Netze. Manuelle Eingriffe zur Steuerung des Algorithmus sind nicht notwendig. Um das Überschneiden von verschiedenen Fronten oder Frontteilen zu vermeiden, ist eine sehr komplexe Suche von Elementen und Knoten in der Nähe der Front nötig. Dadurch ist der Algorithmus sehr langsam.

Bei einer falsch vorgegebenen Dichtefunktion mit zu großen Gradienten kann es möglich sein, dass der Algorithmus scheitert und kein Netz erzeugen kann. Bei der Generierung von Viereckelementen weist der Algorithmus Vorteile gegenüber anderen Verfahren auf. Die Qualität der Netze ist gut und die Anzahl der erzeugten Elemente relativ gering. Irreguläre Punkte, Netzpunkte mit einer von vier abweichenden Anzahl von angrenzenden Elementen (siehe Abschnitt 3.2.2.1), liegen nur im Inneren des Gebietes. Eine Glättung zur Reduzierung der Wirkung der Irregularität ist hierbei möglich.

In der Literatur sind auch Lösungsansätze zur Generierung von Hexaedernetzen mit der Advancing-Front-Methode beschrieben. Diese gehen von einem initialen Viereckнetz auf der Oberfläche der Geometrie aus. Im Inneren der Geometrie bleiben meistens Polyeder übrig, die sich nicht zu Hexaedern vernetzen lassen. Als einfachstes Beispiel hierfür wird unter anderem in [Schneiders 1996a] die Pyramide aus Abbildung 3.3 beschrieben. Das Oberflächennetz aus Vierecken lässt sich beweisbar nicht zu Hexaedern vernetzen.

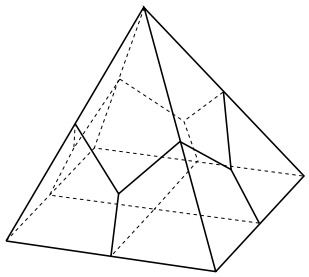


Abbildung 3.3: *Problematische Hexaedervernetzung*

3.1.3 Delaunay-Triangulierung

Im Rahmen dieser Arbeit wird die Delaunay-Triangulierung zur Netzgenerierung von zweidimensionalen Netzen verwendet. Die Erläuterungen zu diesem Verfahren sind daher detaillierter als die Beschreibungen der anderen Verfahren.

Die Delaunay-Triangulierung ist in den letzten Jahren eine sehr populäre Methode zur Generierung von Netzen geworden. Sie existiert zum einen für zweidimensionale Dreiecknetze, lässt sich aber auch auf dreidimensionale Tetraedernetze erweitern [Shewchuk 1997].

Die Delaunay Triangulierung optimiert den Innenwinkel der Dreieckelemente. Für zweidimensionale Netze findet der Algorithmus die optimale Vernetzung mit dem größtmöglichen minimalen Innenwinkel. Die Erweiterung der Delaunay-Triangulierung zum Erzeugen von Tetraedern ist hinsichtlich der Qualität der Elemente nicht so effizient. Delaunay-Netze werden über das Kreis-Kriterium definiert [Delaunay 1934]:

Definition 3 *Ein zweidimensionales Delaunay-Netzwerk besteht aus nicht überlappenden Dreiecken, wobei im Netzwerk kein Eckpunkt vom Umkreis eines beliebigen anderen Dreiecks umschlossen ist.*

Bei der Existenz von nur drei Eckpunkten auf jedem Umkreis ist die Triangulierung eindeutig. Eine eindeutige Delaunay-Triangulierung ist beispielhaft in Abbildung 3.4 a) dargestellt. Der enge Zusammenhang mit den Voronoi-Diagrammen [Voronoi 1908] lässt sich aufzeigen und ist in Abbildung 3.4 b) dargestellt. Voronoi-Diagramme sind definiert durch die Senkrechte auf den Verbindungslinien zwischen zwei Punkten.

Eine Erweiterung der Theorie auf dreidimensionale Gebilde zur Generierung von Tetraedern ist möglich [Fuchs 1998]. Hierbei ist folgende Definition zu beachten:

Definition 4 *Ein dreidimensionales Delaunay-Netzwerk besteht aus nicht überlappenden Tetraedern, wobei im Netzwerk kein Eckpunkt von der umschließenden Kugel eines beliebigen anderen Tetraeders umschlossen wird.*

Zur Generierung der Netze nach den oben beschriebenen Kriterien existieren eine Reihe von Methoden. Im Rahmen dieser Arbeit werden nur Algorithmen zur Generierung zweidimensionaler Netze beschrieben. Die Methoden zur Triangulierung lassen sich in zwei Gruppen teilen: die

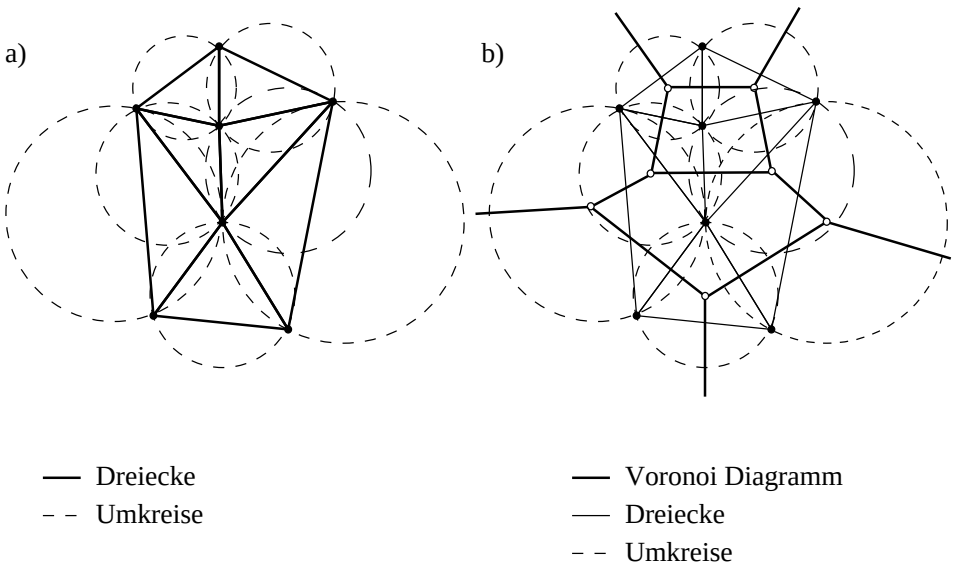


Abbildung 3.4: *Delaunay-Triangulierung: a) Umkreise, b) Voronoi-Diagramm*

statische Triangulierung und die dynamische Triangulierung. Bei der statischen Triangulierung erreicht man im Gegensatz zur dynamischen das Kreis-Kriterium erst, wenn alle vorgegebenen Punkte im Netzwerk berücksichtigt sind.

3.1.3.1 Statische Algorithmen

Recursive-Split-Algorithmus: Dieser Algorithmus wurde zuerst von [Lewis und Robinson 1978] beschrieben und besteht aus folgenden Schritten:

Schritt 1: Der umschließende Polygonzug wird gesucht.

Schritt 2: Rekursiv wird die Geometrie in jeweils zwei Teilgeometrien zerlegt, die möglichst kreisförmig sind (Abbildung 3.5a)-d)). Abgebrochen wird die Rekursion, wenn alle Teilgeometrien nur noch aus drei Punkten bestehen.

Schritt 3: Über alle Kanten wird traversiert. Die beiden durch diese Kante benachbarten Dreiecke werden zu einem gedachten Viereck zusammengefasst. Entlang der kürzeren Kante wird das Viereck dann in zwei Dreiecke zerteilt. Dieser Schritt wird solange wiederholt, bis keine Änderungen mehr auftreten (Abbildung 3.5e)).

Für diesen Algorithmus sind die Laufzeitangaben in der Literatur verschieden. Während [Lewis und Robinson 1978] von einem Laufzeitverhalten von $\mathcal{O}(n \log n)$ schreiben, nehmen [Lee und Schachter 1980] ein Laufzeitverhalten von $\mathcal{O}(n^2)$ an. Beide Angaben sind nicht beweisbar. Die Bisektion zeigt eine direkte Parallelisierbarkeit des Algorithmus auf.

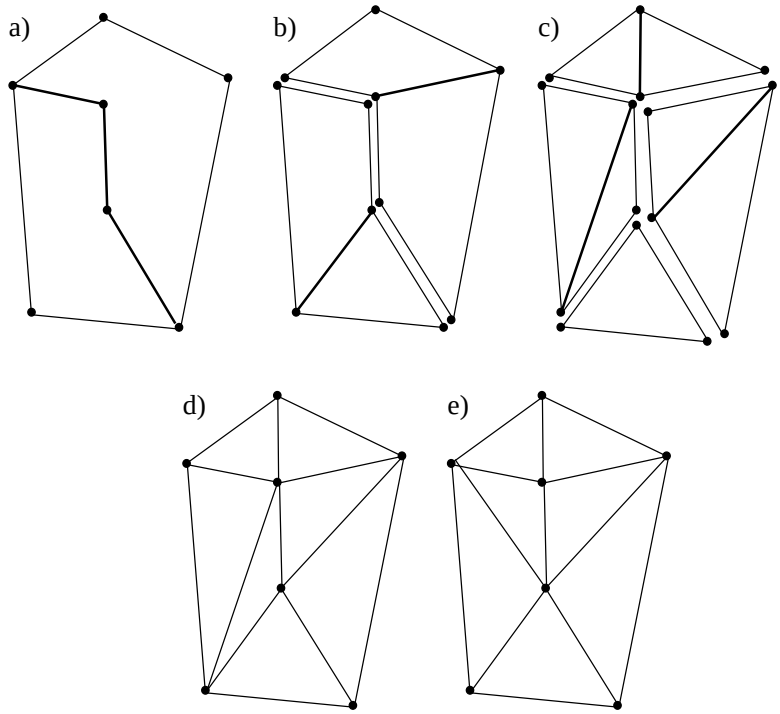


Abbildung 3.5: *Recursive-Split-Algorithmus*

Divide-and-Conquer-Algorithmus: Dieser Algorithmus funktioniert ähnlich wie der Recursive-Split-Algorithmus und besteht aus den folgenden Schritten:

Schritt 1: Die Geometrie wird rekursiv zerteilt, bis jede Teilgeometrie nur noch vier Punkte enthält.

Schritt 2: Aus dem Viereck werden durch Teilen entlang der kürzeren Diagonalen zwei Dreiecke erzeugt.

Schritt 3: Beim Zusammenfügen der Teilgeometrien werden durch Außenkanten der Teilgeometrien benachbarte Dreiecke als Viereck betrachtet und entlang der kürzeren Kante in zwei Dreiecke geteilt.

Die Laufzeit dieses Algorithmus verhält sich mit $\mathcal{O}(n \log n)$ und ist damit der schnellste hier beschriebene Algorithmus. Eine Parallelisierung des Algorithmus zur Nutzung auf Parallelrechnern ist aufgrund der Bisektion denkbar.

Radial-Sweept-Algorithmus: Der Radial-Sweept-Algorithmus ist der am einfachsten zu programmierende Algorithmus und wurde erstmals von [Mirante und Weingarten 1982] vorgestellt. Der Algorithmus besteht aus folgenden Schritten:

Schritt 1: Der dem Mittelpunkt der Geometrie am nächsten liegende Punkt wird ausgewählt und mit allen übrigen Punkten verbunden. Die Entfernung und Richtung zu den restli-

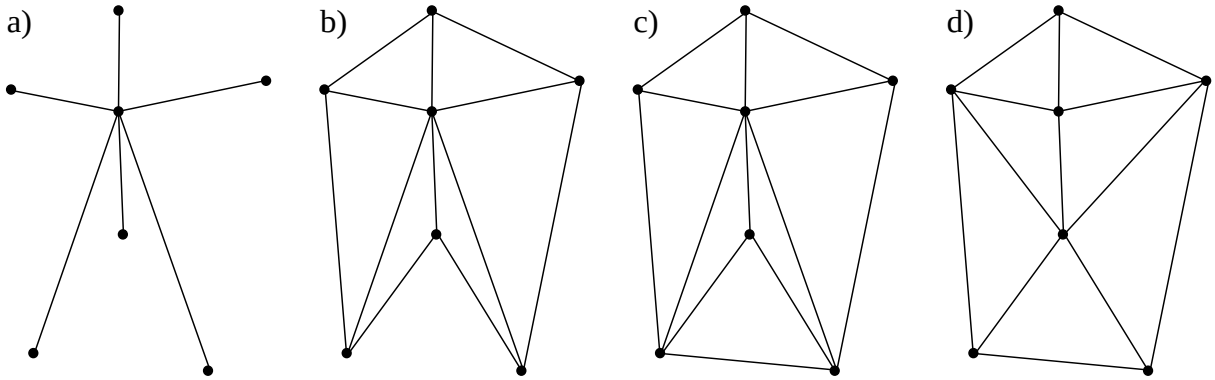


Abbildung 3.6: *Radial-Sweept-Algorithmus*

chen Punkten werden festgestellt und die Punkte nach der Richtung sortiert (Abbildung 3.6a)). Die äußeren Punkte werden entsprechend der Sortierung miteinander verbunden (Abbildung 3.6b)).

Schritt 2: Die Außenkante des Gebiets wird traversiert. Bilden drei aufeinander folgende Punkte ein außenliegendes Dreieck, so sind diese in die Geometrie einzubinden (Abbildung 3.6c)).

Schritt 3: Über alle Kanten wird traversiert. Die beiden durch diese Kante benachbarten Dreiecke werden zu einem gedachten Viereck zusammengefasst. Entlang der kürzeren Kante wird das Vierecke in zwei Dreiecke zerteilt. Dieser Schritt wird solange wiederholt bis keine Änderungen mehr auftreten (Abbildung 3.6d)).

Die beschriebene Vorgehensweise hat Nachteile im Laufzeitverhalten. Das Bestimmen der Richtungen der benachbarten Punkte ist sehr zeitaufwendig. In der Literatur gibt es hierfür Angaben über das Laufzeitverhalten von $\mathcal{O}(n \log n)$. Ebenfalls ungünstig wirkt sich das Traversieren aller Kanten auf die Laufzeit mit $\mathcal{O}(n^2)$ aus. Eine Parallelisierung der Richtungsbestimmung der Punkte sowie der Traversierung der Kanten ist denkbar, in beiden Fällen sind jedoch aufgrund der notwendigen Kommunikationen keine guten Effizienzen zu erwarten.

Step-by-Step-Algorithmus: Der Step-by-Step-Algorithmus ist der wohl am häufigsten verwendete Algorithmus zur Erzeugung von Delaunay-Triangulierungen. Er wird unter anderem von [McCullagh und Ross 1980] beschrieben und verdeutlicht durch die Art der Generierung schon das Kreis-Kriterium.

Schritt 1: Eine initiale Dreieckkante am Rand der Geometrie wird gesucht. Im Kreis mit dem Mittelpunkt in der Mitte zwischen den beiden Punkten der Kante sollte kein weiterer Punkt liegen (Abbildung 3.7 a)).

Schritt 2: Entlang der Mittelsenkrechten der Kante wird der Kreis verschoben und dabei der Radius angepasst. Der erste Punkt, der innerhalb bzw. auf dem Kreis liegt, erfüllt das

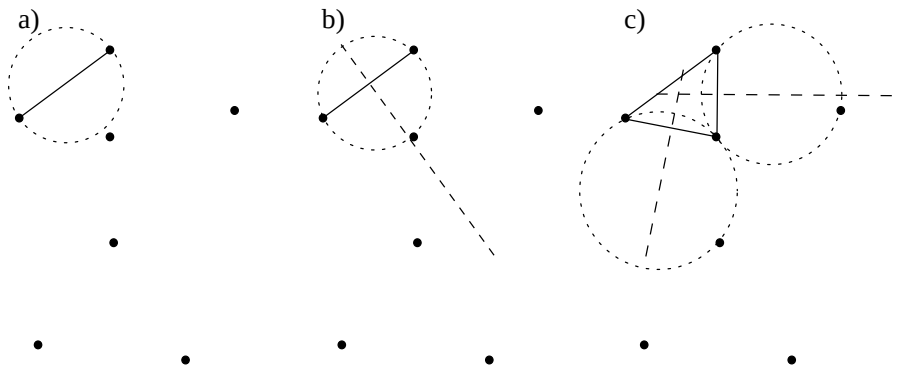


Abbildung 3.7: *Step-by-Step-Algorithmus*

Kreis-Kriterium (Abbildung 3.7 b)). Der gefundene Punkt und die beiden Punkte der initialen Kante werden die Eckpunkte des neuen Dreiecks.

Schritt 3: Die neu eingefügten Kanten sind die initialen Kanten und der Schritt 2 wird rekursiv wiederholt.

Das Suchen der Punkte in der Nähe der initialen Kante erweist sich als ungünstig für das Laufzeitverhalten. Die Komplexität dieses Suchens ist mit $\mathcal{O}(n \log n)$ anzugeben, die Generierung der Dreiecke selbst nur mit $\mathcal{O}(n)$. Eine Parallelisierung dieses Algorithmus wird in [Saxena u. a. 1990] vorgestellt.

Modified-hierarchical-Algorithmus: Dieser Algorithmus wird von [Floriani u. a. 1984] vorgestellt. Der Algorithmus ist eine Mischung aus dem Radial-Sweept-Algorithmus und dem Incremental-Algorithmus.

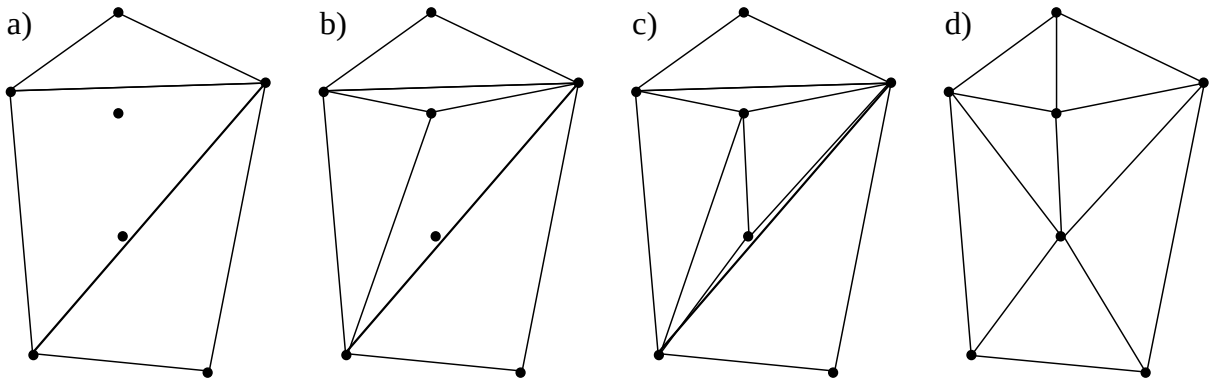


Abbildung 3.8: *Modified-hierarchical-Algorithmus*

Schritt 1: Eine initiale grobe Vernetzung wird erzeugt z.B. durch Triangulierung des umschließenden Polygonzugs (Abbildung 3.8 a)).

Schritt 2: Jeder in einem Dreieck liegende Punkt wird in das Netz eingebaut, indem das Dreieck durch drei neue Dreiecke, bestehend aus dem einzubindenden Punkt und den alten Punkten des Dreiecks, erzeugt wird (Abbildung 3.8 b), c)).

Schritt 3: Über alle Kanten wird traversiert. Die beiden durch eine Kante benachbarten Dreiecke werden zu einem gedachten Viereck zusammengefasst. Entlang der kürzeren Kante wird das Viereck in zwei Dreiecke zerteilt. Dieser Schritt wird solange wiederholt, bis keine Änderungen mehr auftreten (Abbildung 3.8 d)).

Die Komplexität dieses Algorithmus hat die schlechteste Performance mit $\mathcal{O}(n^2)$. Schritt 3 dieses Algorithmus ist parallelisierbar, indem die initialen Dreiecke den einzelnen Prozessoren zugeordnet werden. Durch die möglicherweise ungleiche Anzahl von in das Dreieck einzubindenden Punkten sind schlechte Effizienzen zu erwarten.

3.1.3.2 Dynamische Algorithmen

Dynamische Algorithmen zur Generierung von Delaunay-Triangulierungen erfüllen das Kreis-Kriterium nach dem Einfügen jedes Punktes.

Incremental-Algorithmus: Dieser Algorithmus wurde unter anderem von [Lee und Schachter 1980] vorgestellt.

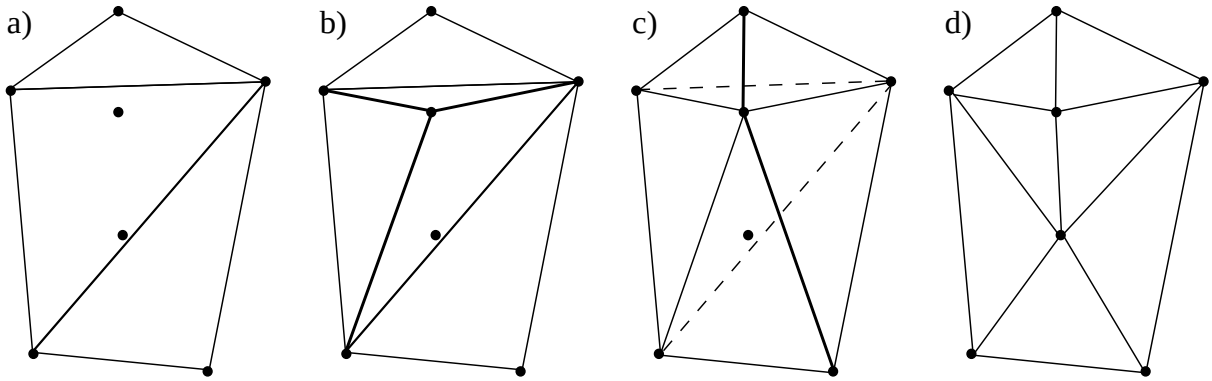


Abbildung 3.9: *Incremental-Algorithmus*

Schritt 1: Es muss eine grobe initiale Triangulierung erzeugt werden. Dies kann z.B. durch die Triangulierung des umschließenden Polygonzugs geschehen (Abbildung 3.9 a)).

Schritt 2: Ein Punkt wird eingefügt, indem das Dreieck, in dem der Punkt liegt, durch drei neue Dreiecke, bestehend aus dem einzubindenden Punkt und den alten Punkten des Dreiecks, ersetzt wird (Abbildung 3.9 b)).

Schritt 3: Alle Vierecke, die sich aus Dreiecken, die Kanten des alten aufgelösten Dreiecks besitzen, bilden lassen, müssen überprüft und möglicherweise muss die diagonale Kante getauscht werden (Abbildung 3.9 c)).

Für diesen Algorithmus sind die Angaben über das Laufzeitverhalten in der Literatur unterschiedlich. Während die meisten Angaben hierfür $\mathcal{O}(n^2)$ lauten, ist in [Heller 1990] ein Laufzeitverhalten von $\mathcal{O}(n \log n)$ angegeben.

Watson's Algorithmus: Der hier beschriebene Algorithmus ist auch bekannt unter dem Namen Incremental Delete-and-build-Algorithmus. Vorgestellt wurde der Algorithmus zuerst von [Watson 1981].

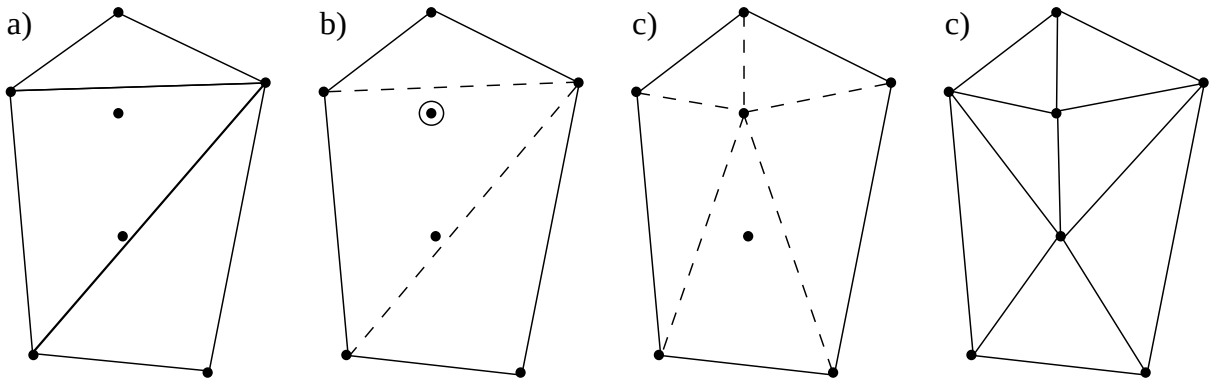


Abbildung 3.10: *Watson's Algorithmus*

Schritt 1: Es muss eine grobe Triangulierung erzeugt werden. Dies kann z.B. durch die Triangulierung des umschließenden Polygonzugs geschehen (Abbildung 3.10 a)).

Schritt 2: Es wird der umgebende Polygonzug des Punktes festgestellt. Dieser Polygonzug ist dadurch gekennzeichnet, dass der neue Punkt innerhalb der Umkreise der Dreiecke liegt. Alle Dreiecke innerhalb des Polygonzuges werden gelöscht (Abbildung 3.10 b)).

Schritt 3: Innerhalb des Polygonzuges werden neue Dreiecke erzeugt. Jedes neue Dreieck besitzt zwei Punkte auf dem Polygonzug und den neu eingefügten Punkt als Eckpunkte.

Der Watson's Algorithmus ist durch die Suche nach dem Polygonzug vom Laufzeitverhalten her ungünstig. Die Komplexität wird mit $\mathcal{O}(n^2)$ angegeben.

3.1.3.3 Einfügen von Punkten

Die oben beschriebenen Verfahren zur Delaunay-Triangulierung können nur vorgegebene Punktmengen optimal zu Dreiecknetzen verbinden. Bei der Vorgabe der Außengrenze, von Fixpunkten und Löchern durch den Anwender ergeben sich nur sehr grobe, für die Finite-Elemente-Methode

unbrauchbare Netze. Zum Generieren von Punkten innerhalb des Gebietes existieren eine Reihe von verschiedenen Generierungsalgorithmen. Mit Hilfe dieser ist es möglich Netze zu erzeugen, die der gewünschten Qualität und der angestrebten Elementgröße entsprechen.

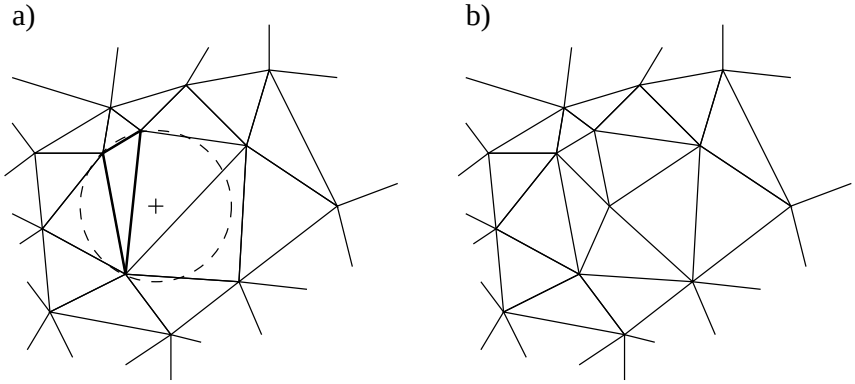


Abbildung 3.11: *Einfügen von Punkten bei einem Dreieck mit schlechter Qualität*

Ziel beim Einfügen von zusätzlichen Punkten in das Netzwerk ist das Eliminieren von Dreiecken mit schlechter Qualität. Bei Dreiecken mit schlechter Qualität liegt der Umkreis-Mittelpunkt weit außerhalb des Dreiecks (siehe auch Abschnitt 3.4). Die beschriebenen Dreiecke werden herausgesucht (Abbildung 3.11 a)). Alle Dreiecke, in deren Umkreis der Mittelpunkt des Umkreises des schlechten Dreiecks liegt, werden gelöscht. Der Umkreis-Mittelpunkt wird als neuer Punkt eingefügt. Aus jeweils zwei Eckpunkten der gelöschten Dreiecke und dem Mittelpunkt werden neue Elemente generiert (Abbildung 3.11 b)).

Chew's erster Algorithmus: [Chew 1989] schlägt einen Algorithmus zum Einfügen von Punkten vor. Als obere Grenze für die Qualität von Dreiecken verwendet er das Umkreisradius-Kriterium (siehe Abschnitt 3.4.2.1) und setzt $\mathcal{B} = 1$ als Maximalwert an. Chew teilt alle Dreiecke auf, deren Umkreisradius größer als die kürzeste Kante im Netzwerk (h_{min}) ist. Hierdurch wird erreicht, dass die maximale Kantenlänge im gesamten Netz $2 \cdot h_{min}$ nicht übersteigt.

Rupperts Algorithmus: Ruppert stellte den Algorithmus in [Ruppert 1993] erstmals vor. In [Ruppert 1995] wird bewiesen, dass der Algorithmus Netze liefert, bei denen die inneren Winkel der Dreiecke nie kleiner als 20.7° sind. Der Algorithmus wird bestimmt durch zwei Regeln:

Regel 1: Wenn sich im Durchmesserkreis einer Kante Punkte befinden, wird in der Mitte der betroffenen Kante ein Punkt eingefügt (siehe Abbildung 3.12).

Regel 2: Spitze Dreiecke werden eliminiert, indem das Kreis-Kriterium mit einer beliebigen Grenze $\mathcal{B}$ entsprechend Abbildung 3.29 und Abschnitt 3.4.2.1 angewendet wird.

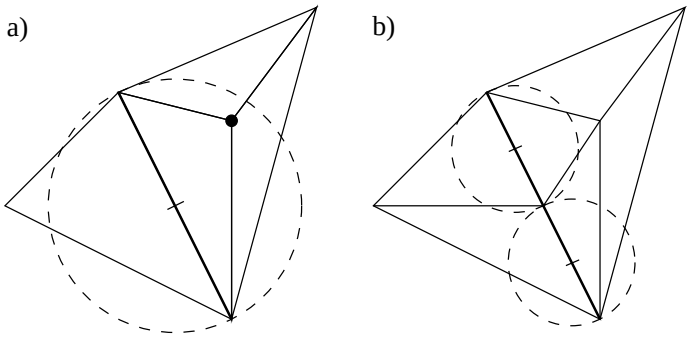


Abbildung 3.12: *Ruppert's Algorithmus, Regel 1, a) Suchen nach einem kritischen Punkt, b) Neuvernetzung*

Chew's zweiter Algorithmus: Dieser Algorithmus wurde erstmals in [Chew 1993] vorgestellt. In [Shewchuk 1997] wird gezeigt, dass der minimale Innenwinkel eines Dreiecks im Netzwerk bei Verwendung dieses Algorithmus 26.5°ist.

Chew zerteilt die Dreiecke unter Verwendung des in Abschnitt 3.4.2.1 vorgestellten Kreis-Kriteriums mit einer Grenze von $\mathcal{B} = 1$. Wenn der Umkreis-Mittelpunkt eines Dreiecks nicht als Knoten eingefügt werden kann, da er z.B. außerhalb der Geometrie liegt, werden alle Punkte im Umkreis der kritischen Kante gelöscht. Ein neuer Punkt wird im Mittelpunkt der kritischen Kante eingefügt. Dieser neue Punkt dient als dritter Punkt für die neu zu erzeugenden Dreiecke.

3.1.4 Generierung von Viereckelementen

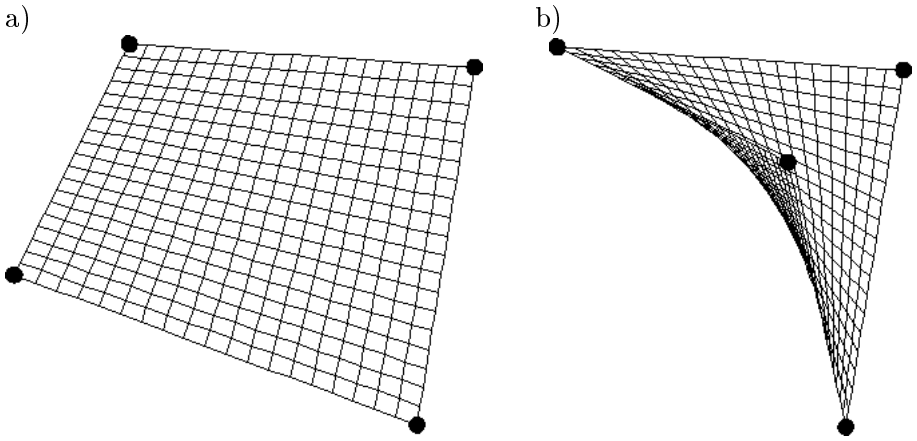


Abbildung 3.13: *a) lokale Koordinaten, b) Überlappung der lokalen Koordinaten eines Viereckelementes mit einem Innenwinkel größer* 180 °

Die Generierung von Netzen aus Viereckelementen ist schwieriger als die Generierung von reinen Dreiecknetzen. Die Ursachen der Schwierigkeiten liegen an Gegebenheiten der Viereckelemente bzw. der Viereckelementnetze:

Ungültige Elementform: Viereckelemente dürfen nicht jede beliebige Form annehmen. So sind Innenwinkel größer oder gleich 180 °unzulässig, da hierbei ein lokales Koordinatensystem nicht mehr eindeutig bestimmbar ist bzw. die lokalen Koordinatenlinien außerhalb des Elementes, wie in Abbildung 3.13 b) dargestellt, liegen. Bei Dreiecken tritt diese Problematik nicht auf, da Dreiecke nur mit Innenwinkeln kleiner 180 °erzeugbar sind.

Vernetzbarkeit: Nicht jede vorgegebene Punktmenge bzw. vorgegebener Polygonzug ist ohne Einfügen von Punkten mit Vierecken vernetzbar. In Abbildung 3.14 sind Beispiele für beide Fälle gegeben. Beliebige Punktmengen oder Randpolygone lassen sich mit gültigen Dreiecken immer vernetzen.

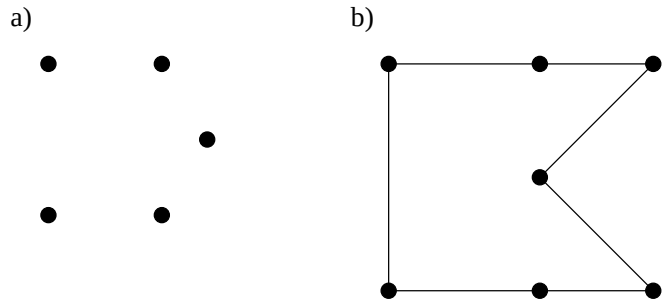


Abbildung 3.14: *a) nicht vernetzbare Punktmenge b) nicht vernetzbarer Polygonzug*

Die beiden dargestellten Gegebenheiten verursachen weitere ungünstige Einschränkungen bei der Vernetzung mit Viereckelementen:

Teilen einer Kante Das nachträgliche Teilen einer Kante in zwei Teilkanten ist nicht ohne weiteres möglich, da hierbei nicht vernetzbare Gebiete entstehen können (Abbildung 3.15 a)).

Einfügen einzelner Punkte Einzelne Punkte können nicht in das Netz eingefügt werden (Abbildung 3.15 b)).

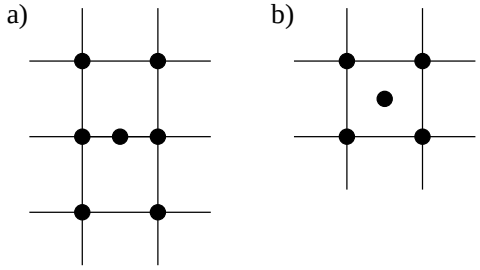


Abbildung 3.15: *a) Teilen einer Kante b) Einbau eines Punktes*

3.1.5 Generierung von Vierecken aus Dreiecken

Ein Teil der hier beschriebenen Algorithmen zur Generierung unstrukturierter Netze kann nur Dreiecke generieren. Zur Generierung von Vierecken ist daher oft ein Umwandeln der Dreiecknetze in Viereckne tze notwendig.

Die Viereckelemente werden aus zuvor generierten Dreieckelementen erzeugt. Hierbei wird angestrebt, aus jeweils zwei benachbarten Dreieckelementen vier Viereckelemente zu erzeugen (Abbildung 3.16 a)). Nicht verteilbare Dreiecke werden in drei Vierecke zerlegt (Abbildung 3.16 b)).

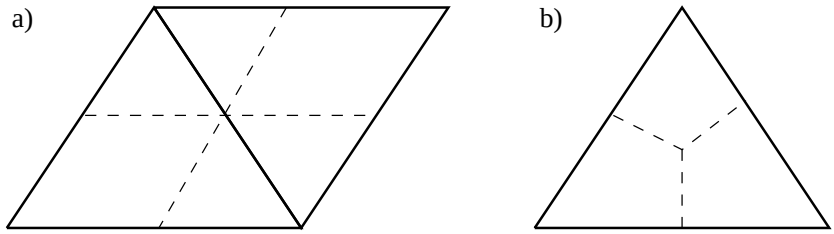


Abbildung 3.16: *Generierung der Vierecke aus Dreiecken*

Zusammenfassen der Dreiecke: Die Zusammenfassung von jeweils zwei Dreiecken zu vier Vierecken wird als Paving-Algorithmus bezeichnet. In der Mitte der Elementkanten der Dreiecke werden Knoten eingefügt. Diese werden entsprechend Abbildung 3.16 a) zur Generierung der vier neuen Viereckelemente verwendet.

Algorithmus I: Das Zusammenfassen der Dreiecke kann mit Hilfe eines abgewandelten Greedy-Algorithmus geschehen. Eine Beschreibung des Greedy-Algorithmus findet man unter anderem in [Farhat 1988]. Basierend auf der Graphentheorie wird ein bestehendes Elementnetz mit dem Greedy-Algorithmus in Teilnetze zerlegt. Hierbei wird jedem Knoten ein Wichtungsfaktor, der der Anzahl der an einen Knoten angrenzenden Elemente entspricht, zugewiesen. Rekursiv werden, ausgehend vom Knoten mit der geringsten Wichtung, einem Teilgebiet Elemente zugeordnet. Die Wichtung der Knoten der verteilten Elemente wird in jedem Rekursionsschritt reduziert.

Um bessere Viereckne tze zu erhalten, wird die Wichtung der Knoten gegenüber dem Standard Greedy-Algorithmus verändert. Die Wichtung der inneren Knoten wird gegenüber den Randknoten erhöht, um nicht zusammenfassbare Dreiecke ins Innere des Gebietes zu verschieben. Hier ist der Einfluss dieser ungünstigen Elemente auf die Netzqualität nach der Glättung geringer.

Zum Zusammenfassen der Dreiecke ist eine aufwendige Datenstruktur des Dreiecknetzes notwendig. In dieser muss jeder Knoten seine Nachbarknoten und die angrenzenden Elemente kennen. Jedes Element muss Zeiger auf seine Knoten sowie die benachbarten Elemente haben.

Algorithmus II: Zum Zusammenfassen wurde ein weiterer Algorithmus entwickelt. Dieser basiert auf dem Greedy-Algorithmus, nutzt aber nicht Knotenwichtungen, sondern Kantenwichtungen. Jedes Dreieckelement hat Zeiger auf die jeweiligen Nachbarelemente. Liegt eine Fixlinie zwischen zwei Dreiecken oder stellt eine Kante die Außengrenze des zu vernetzenden Gebietes dar, wird der entsprechende Zeiger nicht belegt. Knoten auf vorgegebenen Linien, wie z.B. Fixlinien oder Randlinien, erhalten eine Knotenwichtung. Die Summe der Knotenwichtungen ergibt die Elementwichtung. Der Algorithmus sucht nun rekursiv das Element, das die geringste Anzahl von Zeigern auf seine Nachbarelemente hat. Existieren mehrere Elemente mit minimaler Anzahl von Nachbarn, wird unter diesen Elementen das Element mit der größten Elementwichtung gesucht. Dieses wird mit dem Nachbarelement vereinigt, das die geringste Anzahl von Nachbarn und wenn nötig die größte Elementwichtung besitzt.

Teilen eines Dreiecks in drei Vierecke: Nicht zusammenfassbare Dreieckelemente werden in drei Viereckelemente zerteilt. Hierfür wird der Schwerpunkt des Dreiecks bestimmt. Dieser und die Seitenmittenknoten dienen als Eckknoten der neuen Viereckelemente (siehe Abbildung 3.16 b)).

Die Generierung dieser Vierecke ist wesentlich einfacher als das Zusammenfassen von zwei Dreiecken zu vier Viereckelementen. Die Qualität der hierbei entstehenden Vierecke ist sehr schlecht. Netze, die nur aus derart generierten Viereckelementen bestehen, sind durch Glätten nicht verbesserbar.

Eigenschaften der Viereckknetze: Viereckknetze, die aus Dreieckknetzen generiert wurden, haben nachteilige Eigenschaften. Linien und Punkte im Inneren des zu vernetzenden Gebietes, die bei Generierung berücksichtigt werden sollen, müssen schon im Dreieckknetz vorhanden sein. Hieraus folgt, dass die Linien und Punkte nur auf alten Dreieckkanten bzw. -punkten liegen können. Zwischen zwei vorgegebenen Punkten muss daher immer ein nicht vorgegebener Punkt liegen und zwischen zwei parallelen Linien immer eine Punktreihe.

Innerhalb der Netze treten immer Knoten auf, an die nur drei Vierecke angrenzen. Die betroffenen Elemente weisen immer ungünstige Innenwinkel auf, die sich durch Glättung und Netzverbesserung nur bedingt beseitigen lassen.

3.1.6 Generierung von Hexaederelementen

Hexaedernetze lassen sich wie oben beschrieben relativ schlecht generieren. Standard-Techniken zur Netzgenerierung, wie z.B. das Advancing-Front-Verfahren, scheitern bei komplizierteren Geometrien. Lösungen werden hier durch hybride Netze aus Tetraedern, Pyramiden und Hexaedern in [Meyers und Tautges 1998] vorgeschlagen.

Das Überlagerungsverfahren [Dhondt 1999] und die octree-basierte Netzgenerierung können zwar beliebige Geometrien vernetzen, haben aber die Einschränkung, dass die Diskretisierung

der Oberfläche durch den Algorithmus vorgegeben wird und nicht immer den Anforderungen des Nutzers entsprechen.

Das Zusammenfassen von Tetraedern zu Hexaedern analog zur Zusammenfassung von Dreiecken zu Vierecken in Abschnitt 3.1.5 ist auch relativ schwierig und extrem zeitaufwendig. Die gefundene Lösung ist meist von schlechter Qualität.

Die Teilung der Tetraeder in jeweils vier Hexaeder ist für jede Geometrie möglich. Hierbei entstehen Elemente mit einer sehr ungünstigen Topologie. An die Knoten des Tetraedernetzes grenzen normalerweise 24 Elemente an. Entsprechend Tabelle 2.1 sollten an jeden Knoten eines Hexaedernetzes acht Elemente angrenzen. Durch Glätten oder Netzverbesserung lässt sich diese Problematik nicht umgehen. Des weiteren gelten für so entstandene Hexaedernetze die gleichen nachteiligen Eigenschaften wie die in Abschnitt 3.1.5 beschriebenen Eigenschaften für Viereckelementnetze, die aus Dreieckelementnetzen entstanden sind.

3.2 Glättung und Netzverbesserungen

Die Qualität der erzeugten Netze entspricht nicht immer den in Abschnitt 3.4 beschriebenen Anforderungen. Zur Verbesserung der Qualität der Netze gibt es zwei Möglichkeiten. Zum einen kann man das Netz glätten, zum anderen ist eine Verbesserung bei patchweiser Betrachtung durch Hinzufügen und Wegnahme von Elementen möglich.

3.2.1 Glättung

Bei der Glättung wird versucht, jeden Knoten in die Mitte seiner Nachbarknoten zu verschieben. Hierfür wird jeweils ein gewichteter Mittelwert der Nachbarknoten berechnet. Eine Wichtung ist notwendig um die Elementgröße bei der Glättung nicht zu stark zu verändern. Die verwendete Wichtung ist abhängig von der Anzahl der benachbarten Knoten des Nachbarknotens [Schneiders 1996a]:

$$\bar{x} = \frac{1}{\sum n_i} \sum \bar{x}_i \cdot n_i \tag{3.1}$$

Hierbei ist $\bar{x}$ die neue Position den Knotens, $\bar{x}_i$ die Position der Nachbarknoten und n_i die Anzahl der Nachbarknoten des Nachbarknotens. Das Glätten eines Knotens hat Einfluss auf die Glättung der umliegenden Knoten. Daher ist der Glättungsalgorithmus mehrfach für das gesamte Netz durchzuführen. Das Glättungsverfahren konvergiert gut, so dass in den meisten Fällen nur zwei bis drei Durchläufe für das gesamte Netz notwendig sind. Bei der Glättung ist besonders bei Viereckelementen und bei einzelnen Fixpunkten auf die Geometrie sowie auf die Größe der betroffenen Elemente zu achten.

3.2.2 Patchweises Verbessern

3.2.2.1 Irreguläre Punkte

Unter einem irregulären Punkt versteht man einen Punkt, der zu einer von der optimalen Anzahl abweichenden Anzahl von Elementen gehört. Die optimale Anzahl von angrenzenden Elementen entspricht der von strukturierten Netzen und wird in der Qualitätsbeschreibung durch das Topologie-Kriterium (Abschnitt 3.4.1.1) beschrieben. Die optimalen Anzahlen sind der Tabelle 2.1 zu entnehmen. Irreguläre Punkte können durch patchweises Verbessern beseitigt oder in ihrer Anzahl minimiert werden.

3.2.2.2 Patchweises Verbessern von Viereckne tzen

Die patchweise Verbesserung von Viereckne tzen wird von [Kinney 1997] vorgestellt. Zwei relativ einfach zu verbessernde Elementpatche sind in Abbildung 3.17 dargestellt. Die in der Abbildung angegebenen Zahlen entsprechen der Anzahl der an einen Knoten angrenzenden Elemente. Ist diese Anzahl größer oder gleich fünf, ist der Winkel der angrenzenden Elemente kleiner als 90°. Bei einer Anzahl von drei liegt der innere Winkel der Elemente bei etwa 120°.

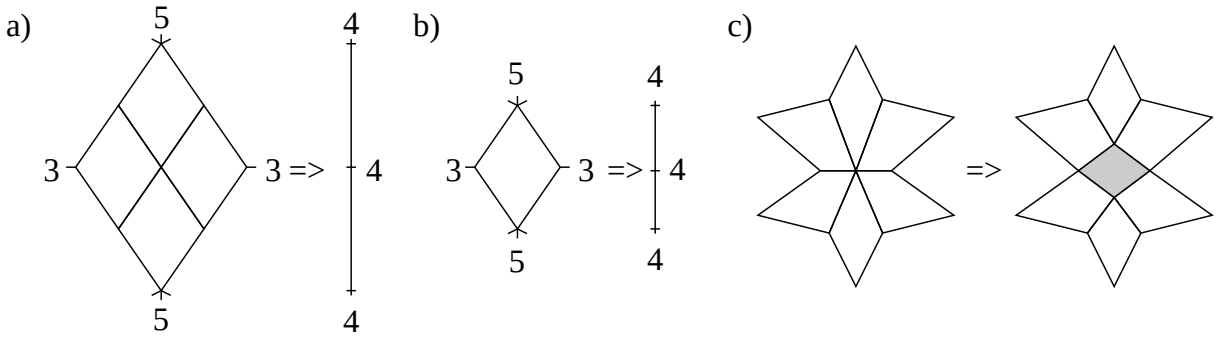


Abbildung 3.17: *Verbesserbare Elementpatche (Zahlen entsprechen der Anzahl der an einen Knoten angrenzenden Elemente)*

Der in Abbildung 3.17 a) dargestellte Patch lässt sich durch Wegnehmen der vier Elemente verbessern. Die beiden Knoten mit der Anzahl 3 fallen hierbei in den mittleren Knoten. Nach dem Löschen der Elemente haben alle Knoten die Anzahl vier.

Treten in einem Netz Elemente mit der gleichen Anzahl von Nachbarelementen auf, wie in Abbildung 3.17 b) dargestellt, ist das Netz durch Wegnehmen des betreffenden Elementes zu verbessern. Nach dem Wegnehmen haben die verbleibenden Knoten vier angrenzende Viereckelemente.

Die in Abbildung 3.17 c) dargestellte Verbesserungsmöglichkeit wurde selbst entwickelt. Hierbei werden Knoten mit sechs oder mehr angrenzenden Elementen aufgetrennt und ein neuer Knoten in der Nähe erzeugt. Mit diesem Knoten wird ein neues Elemente erstellt. Der proble-

matische Knoten sowie der neu erzeugte Knoten haben nach dem Verbessern jeweils die optimale Anzahl darauf referenzierender Elemente.

3.2.2.3 Patchweises Verbessern von Dreiecknetzen

Verschiedene Varianten zur patchweisen Verbesserung von Dreiecknetzen sind unter anderem [Shewchuk 1997] zu entnehmen. In Abbildung 3.18 sind einige leicht zu verbessernde Elementpatche dargestellt. In den Abbildungen 3.18 a) und d) müssen jeweils zwei Dreiecke gelöscht werden. Eine Verbesserung der Qualität der Elemente des Elementpatches in 3.18 c) ist durch Kippen der Diagonalen zu erreichen. Durch Hinzufügen sind die Elementpatche in 3.18 b) und e) zu verbessern.

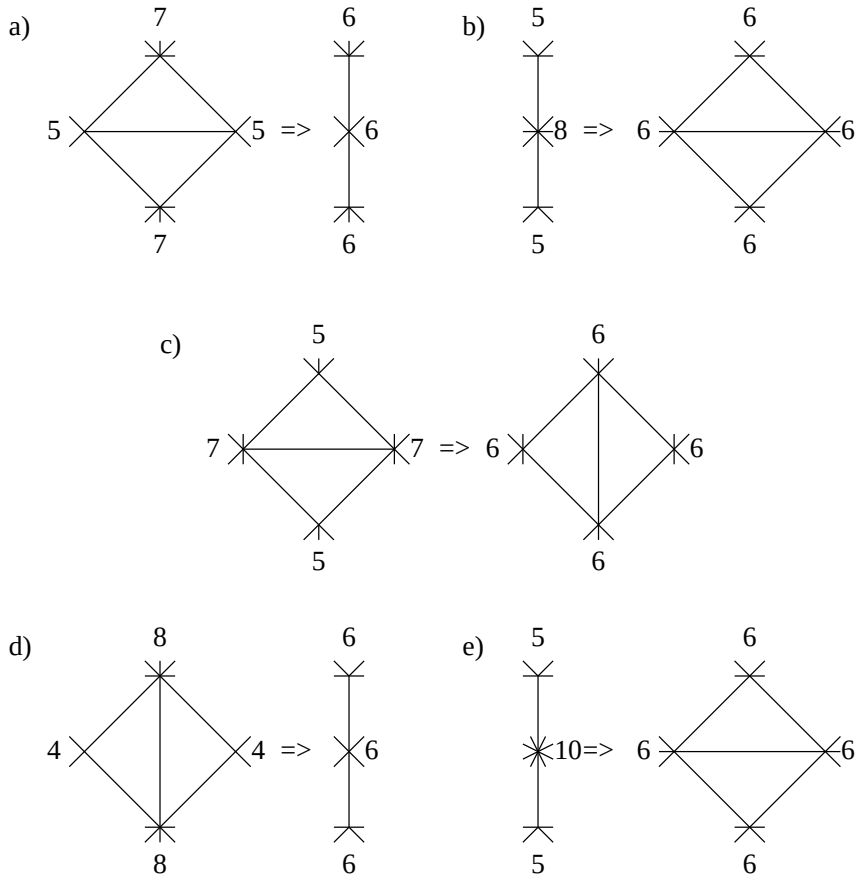


Abbildung 3.18: *Verbesserbare Elementpatche (Zahlen entsprechen der Anzahl der an einen Knoten angrenzenden Elemente)*

3.2.2.4 Patchweises Verbessern von dreidimensionalen Netzen

Für Tetraeder- und Hexaedernetze sind aufgrund der Komplexität bisher noch keine Ansätze für die patchweise Verbesserung gefunden worden.

3.3 Netzverfeinerung

Lokale Netzverfeinerungen sind Regionen, an denen in ein bestehendes Netz feinere bzw. kleinere Elemente eingefügt werden. Bei der Verwendung der Finite-Elemente-Methode geben Fehlerindikatoren Auskunft über notwendige Verfeinerungen des Netzes (siehe [Olden 1998]). Netzverfeinerungen sind hierbei unter anderem an Stellen mit Singularitäten notwendig.

3.3.1 Dreieckelemente

Vorgehensweisen zur Verfeinerung von Dreiecknetzen sind in [Bank und Xu 1996] beschrieben. Eine Verfeinerung kann durch Aufteilen der Elemente in mehrere Elemente geschehen. Die Aufteilung eines Elementes in drei Dreiecke ist in Abbildung 3.19 e) dargestellt. Oft ist eine Unterteilung der Seitenmitten zur Verfeinerung notwendig. Die möglichen Fälle mit einer, zwei und drei zu verfeinernden Kanten eines Dreiecks sind in Abbildung 3.19 a) - d) dargestellt.

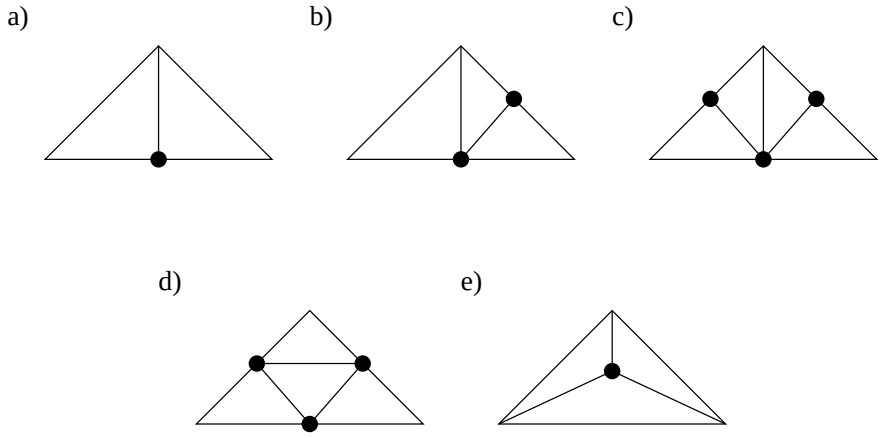


Abbildung 3.19: *Übergangsnetze zur Verfeinerung von Dreiecknetzen*

3.3.2 Viereckelemente

Die Verfeinerung von Viereckelementen ist wesentlich aufwendiger als die Verfeinerung von Dreieckelementen oder Tetraederelementen. Man muss zwei Arten von Verfeinerungen unterscheiden: die zweier-Verfeinerung und die dreier-Verfeinerung.

Die zweier-Verfeinerung von Viereckelementen beschreibt [Kinney 1997]. Sie bedeutet, dass die betroffenen Kanten in der zu verfeinernden Region in jeweils zwei Teile geteilt werden. Bei dieser Art von Verfeinerung ist nur ein Übergangsnetz möglich. Dieses teilt zwei aneinander angrenzende Kanten (Abbildung 3.20 b)). Ein komplett verfeinertes Element ist in Abbildung 3.20 a) dargestellt.

Dieses nur eine mögliche Übergangsnetz bringt starke Einschränkungen für die Verfeinerung der Netze mit sich. Es ist nicht möglich beliebige Verfeinerungen in ein Netz einzubringen, da immer Paare von zu verfeinernden Kanten entlang der Außenkante der Verfeinerung gefunden

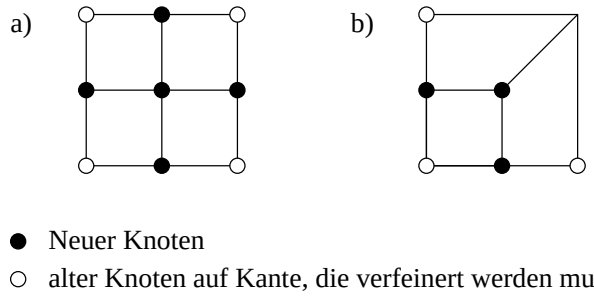


Abbildung 3.20: *Übergangsnetze der zweier-Verfeinerung von Vierecken*

werden müssen. In strukturierten Netzen ist dies oft möglich, in unstrukturierten Netzen sind mögliche Randpolygone mit gerader Anzahl von Polygonkanten nicht immer auffindbar.

Beliebige Verfeinerungen können in unstrukturierte Viereckne tze nur mit Hilfe der dreier-Verfeinerung eingebracht werden. Bei der dreier-Verfeinerung wird jede Elementkante in jeweils drei Teilkanten unterteilt [Schneiders 1996b].

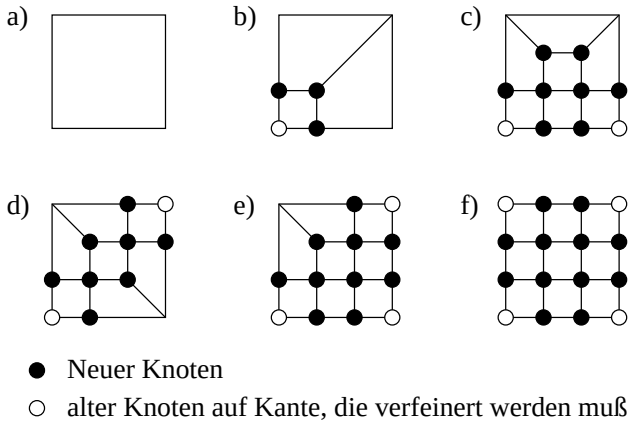


Abbildung 3.21: *Übergangsnetze der dreier-Verfeinerung von Vierecken*

Zum Einbau beliebiger Verfeinerungen in unstrukturierte Viereckne tze sind die in Abbildung 3.21 b) - e) dargestellten Übergangsnetze notwendig. Zur Verfeinerung müssen zuerst alle Knoten der Elemente markiert werden, die komplett (Abbildung 3.21 f)) verfeinert werden sollen. In einem nächsten Schritt müssen die jeweiligen Verfeinerungsnetze (Abbildung 3.21 b) -e)) in Abhängigkeit der Kanten gesucht werden, die am Rand des Verfeinerungsgebietes notwendig sind.

Ein rekursives Vorgehen, das heißt ein mehrfaches Verfeinern an einer Stelle, ist problemlos möglich, indem der Verfeinerungsalgorithmus mehrmals hintereinander auf das Netz angewendet wird. Hierdurch erhält man je nach Rekursionsanzahl verschieden stark verfeinerte Elemente. Bei der verwendeten dreier-Verfeinerung wird ein Element durch maximal neun Elemente ersetzt, nach zwei Rekursionsschritten erhält man schon 81 und nach drei Schritten sind es schon 729 Elemente. Eine mehrfache Verfeinerung an einer Stelle ist immer mit Bedacht anzuwenden, da

die Gesamtelementanzahl rapide steigen kann. In Abbildung 3.23 a) ist ein Netz dargestellt, das an einer Stelle einmal und an einer anderen Stelle zweimal verfeinert wurde. Die Gesamtanzahl der Elemente änderte sich durch diese Verfeinerungen von 48 auf 422.

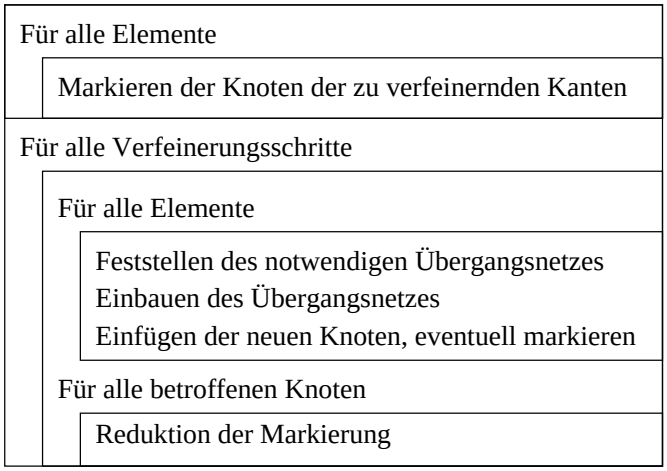


Abbildung 3.22: *Ablauf der lokalen Verfeinerung von Netzen*

Zur rekursiven Verfeinerung sind die in Abbildung 3.22 dargestellten Schritte notwendig. Zuerst werden alle Knoten an den zu verfeinernden Kanten mit der Anzahl der notwendigen Verfeinerungsschritte markiert (Abbildung 3.24 a). Danach sind folgende Schritte für jeden Rekursionsschritt der Verfeinerung notwendig:

Schritt 1: Die passenden Verfeinerungsnetze werden entsprechend Abbildung 3.20 gesucht und eingebaut. Die hierbei entstehenden neuen Knoten werden markiert.

Schritt 2: Bei allen Knoten im Netz wird die Markierung um eins reduziert (Abbildung 3.24 b).

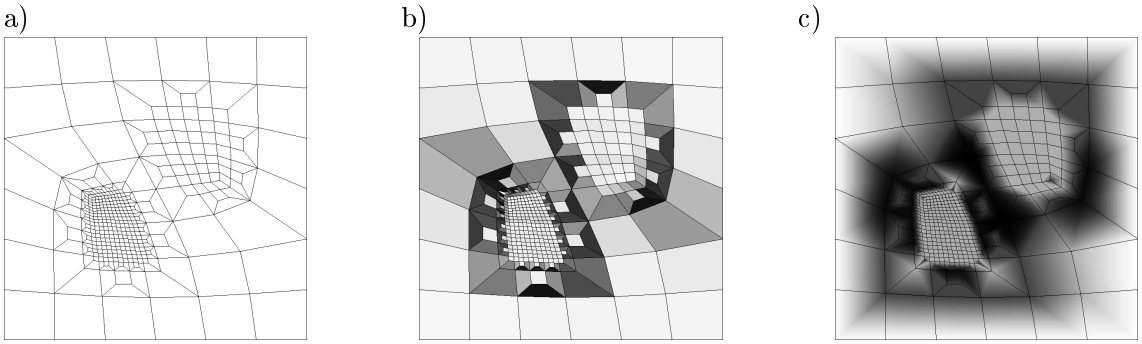


Abbildung 3.23: *Verfeinerung Viereckelemente, a) Netz, b) Form-Kriterium, c)Topologie-Kriterium*

Im Bereich des Übergangs von groben zu feineren Elementen ist die Qualität der Elemente relativ ungünstig. Bei einem optimalen Ausgangsnetz, das nur Elemente mit Innenwinkeln von 90° und jeweils gleicher Seitenlänge besitzt, erhält man Elemente mit Innenwinkeln von 45° und

Seitenlängen im Verhältnis 1 : 3. Eine Darstellung der Qualität mit Hilfe des Form-Kriteriums (siehe Abschnitt 3.4) ist in Abbildung 3.23 b) zu sehen. Hierbei zeigen dunkel markierte Elemente eine schlechte Qualität an. Im dargestellten Beispiel sind diese Elemente nur im Bereich der Verfeinerung zu finden. Das Topologie-Kriterium (Abbildung 3.23 c), siehe Abschnitt 3.4), zeigt ebenfalls die Problematik auf. Im Bereich des Übergangs treten Knoten mit einer Anzahl von angrenzenden Elementen ungleich der optimalen Anzahl auf. So ist die Anzahl der an einen Knoten angrenzenden Elemente im Bereich der Verfeinerung von Viereckelementen teilweise sechs oder sieben, was einem Innenwinkel von 60° bzw. $51,4^\circ$ entspricht.

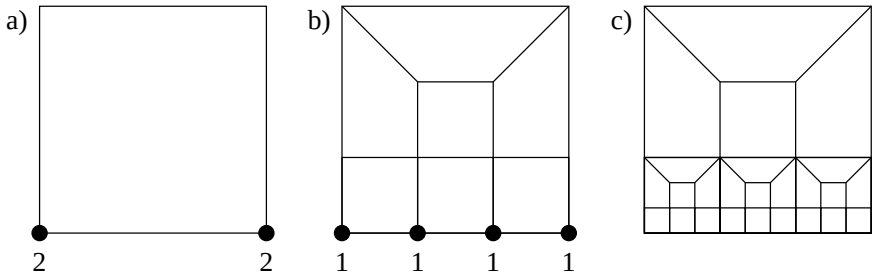


Abbildung 3.24: *Rekursive Verfeinerung*

3.3.3 Tetraederelemente

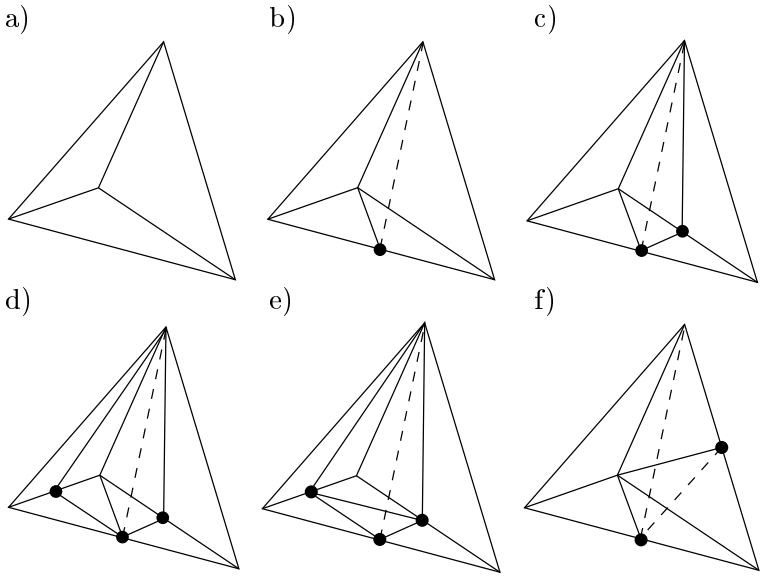


Abbildung 3.25: *Übergangsnetze der Verfeinerung von Tetraedern*

Tetraederelemente können ähnlich wie Dreieckelemente in mehrere Elemente zerlegt werden. Es ist möglich, dass auch einzelne Knoten eingefügt werden. Die Aufteilung eines Tetraeders bewirkt immer, dass mindestens zwei Flächen aufgeteilt werden, sodass hierdurch jeweils min-

destens zwei benachbarte Tetraeder ebenfalls verfeinert werden müssen. In der Abbildung 3.25 ist ein Teil der möglichen Verfeinerungen dargestellt. Weitere Verfeinerungen von Tetraedern ergeben sich aus Kombinationen der dargestellten Verfeinerungen.

3.3.4 Hexaederelemente

Bei Hexaederelementen ist das Verfeinern wesentlich schwieriger als bei allen anderen Elementen. Eine zweier-Verfeinerung ist wie bei den Viereckelementen nur in strukturierten Netzen sinnvoll. Aufgrund des nur einen möglichen Übergangsnetzes (Abbildung 3.26 b)) ist das Finden von verfeinerbaren Regionen schwierig und nicht immer möglich. Dies schränkt die Auswahl des zu verfeinernden Gebietes weiter ein.

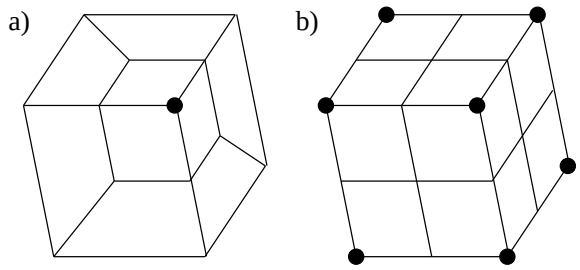


Abbildung 3.26: *Übergangsnetze der zweier-Verfeinerung von Hexaedern*

Eine dreier-Verfeinerung ist bei unstrukturierten Netzen die einzige brauchbare Möglichkeit zur Adaption. Diese ist allerdings nur für konvexe Teilbereiche möglich. Hierfür sind die in Abbildungen 3.27 a) - e) dargestellten Übergangsnetze notwendig. Bei nicht vollständig konvexen Teilgebieten braucht man 22 verschiedene Übergangsnetze [Schneiders u. a. 1996b], die aus sinnvollen Kombinationen der Oberflächen (Abbildungen 3.21 b) - f)) gebildet werden. Von diesen 22 notwendigen Übergangsnetzen lassen sich einige, wie z.B. das in Abbildung 3.27 f) dargestellte, beweisbar nicht generieren [Mitchell 1996]. Rekursives Anwenden zur mehrfachen Verfeinerung ist entsprechend der Vorgehensweise für Viereckelemente möglich.

3.4 Qualitätsbeschreibung

An verschiedenen Stellen innerhalb eines Netzgenerators ist die geometrische Bewertung der Qualität einzelner Elemente, einzelner Knoten oder des gesamten Netzes notwendig. Dies ist unter anderem beim Zusammenfassen von Dreiecken zu Vierecken, bei der Netzverbesserung, beim Glätten oder bei der Bewertung der gesamten Diskretisierung der Fall.

Die Qualitätsbeschreibungen lassen sich in knotenorientierte und elementorientierte Kriterien unterteilen. Das gesamte Netz kann beurteilt werden, indem man die Summe des jeweiligen Indikators über das gesamte Netz bildet:

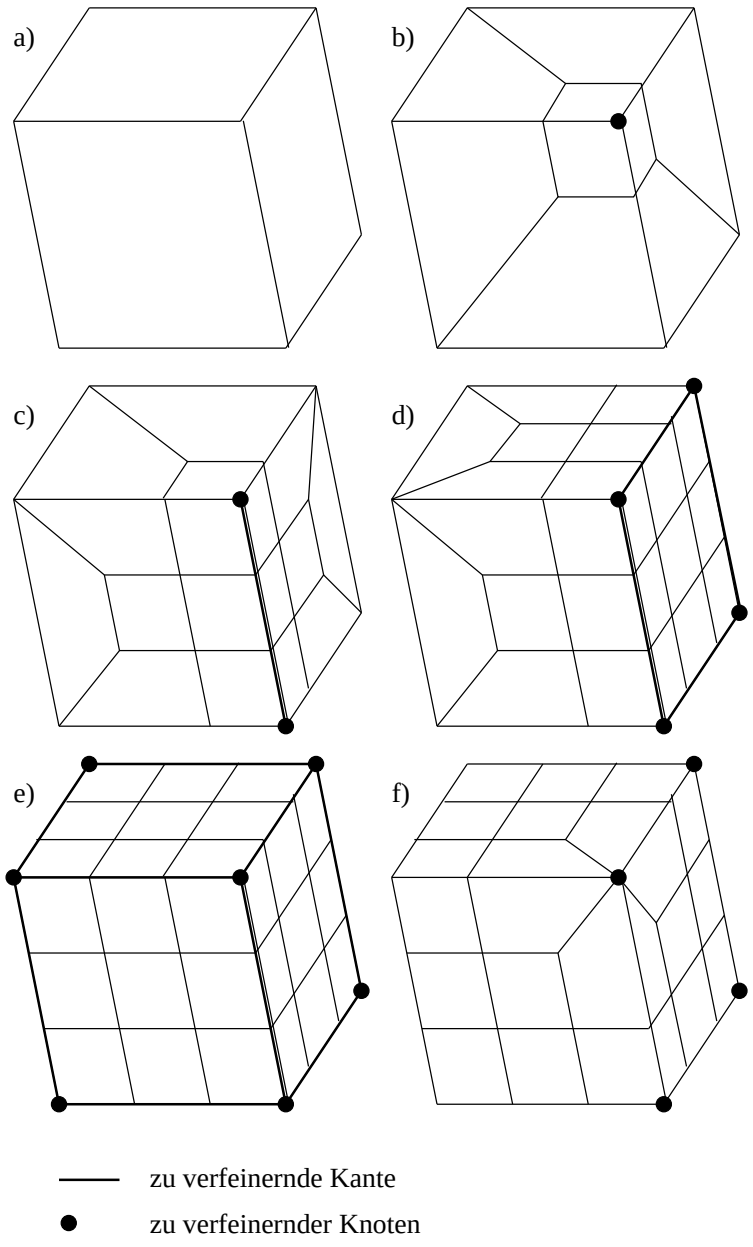


Abbildung 3.27: *Übergangsnetze der dreier-Verfeinerung von Hexaedern*

$$q_{gesamt} = \frac{1}{n} \sum_{i=1}^{n} q_{Element/Knoten} \tag{3.2}$$

wobei q_{gesamt} die Qualität des Netzes, n die Element- bzw. Knotenanzahl und $q_{Element/Knoten}$ die Qualität des einzelnen Elementes bzw. Knotens ist.

3.4.1 Knotenorientierte Kriterien

3.4.1.1 Topologie-Kriterium

[Frey und Field 1991] stellen einen relativ einfach in ein Programm einzubauenden Indikator für die Netzqualität vor, der als Winkelindikator interpretiert werden kann. Definiert ist der Indikator für Dreiecke durch:

$$\epsilon_t = |\delta - 6| \tag{3.3}$$

Eine Anwendung auf Viereckknetze, Tetraeder und Hexaeder ist in der Literatur nicht vorgeschlagen, aber entsprechend der Tabelle 2.1 herleitbar. Für Vierecke:

$$\epsilon_t = |\delta - 4| \tag{3.4}$$

Für Tetraeder:

$$\epsilon_t = |\delta - 24| \tag{3.5}$$

Für Hexaeder:

$$\epsilon_t = |\delta - 8| \tag{3.6}$$

Hierbei ist δ die Anzahl der Elemente an einem Knoten. Ein optimales Netz, d.h. ein Dreiecknetz mit jeweils 6 Dreiecken pro Knoten oder ein Viereckknetz mit jeweils 4 Elementen je Knoten, hat $\epsilon_t = 0$. Die Qualität eines Netzes ist größer, je kleiner ϵ_t ist.

Das Topologie-Kriterium ist ein Kriterium zum Auffinden von Knoten mit einer ungünstigen Anzahl von darauf referenzierenden Elementen. Diese Knoten sind *irreguläre Punkte* und lassen sich durch Topologieverbesserungen entsprechend Abschnitt 3.2.2 verbessern. Eine Verbesserung ist besonders bei Netzen aus Viereckelementen und Hexaederelementen sinnvoll, da dort jede Abweichung von der optimalen Anzahl sofort schlechtere Qualität nach den übrigen Kriterien darstellt. Irreguläre Punkte lassen sich in unstrukturierten Netzen nicht vermeiden. Zum einen sind sie zur Approximation der Geometrie notwendig, zum anderen ist eine lokale Verfeinerung nur durch irreguläre Punkte möglich. Anwendbar ist dieses Kriterium für alle Netze mit einer Elementart unter Berücksichtigung der Tabelle 2.1.

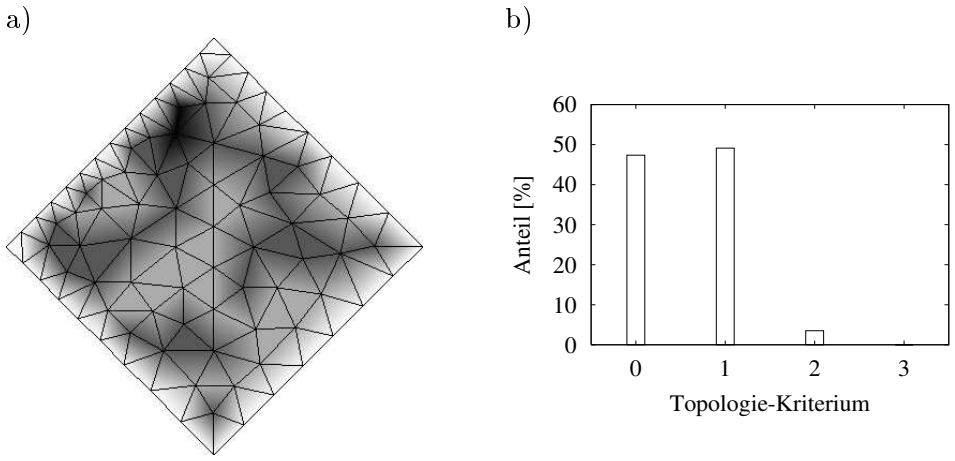


Abbildung 3.28: *Topologie Kriterium a) Netz (grau ≙ gute Qualität, schwarz ≙ schlechte Qualität, weiß ≙ Randknoten), b) Verteilung*

3.4.2 Elementorientierte Kriterien

3.4.2.1 Umkreisradius-Kriterium

Ein Kriterium zum Bewerten der Qualität der Elemente ist das Umkreisradius-Kriterium (circumradius-to-shortest-edge ratio), das in [Talmor u. a. 1995] beschrieben wurde. Definiert wird das Kriterium durch den Quotienten aus dem Umkreisradius und der kürzesten Kantenlänge eines Dreiecks (siehe Abbildung 3.29):

$$\mathcal{B} = \frac{r}{d} \tag{3.7}$$

Dieses Kriterium lässt sich problemlos auch auf Tetraedernetze anwenden. Hierbei ist dann der Umkreisradius durch den Radius der umschließenden Kugel zu ersetzen. Der Qualitätsparameter $\mathcal{B}$ für ein Finite-Elemente-Netz ist eine obere Grenze für alle Dreieckelemente eines Netzes. In [Chew 1989] wird als obere Grenze für Dreiecknetze $\mathcal{B} < 1$ vorgeschlagen. Für gleichseitige Dreiecke ergibt das Umkreisradius-Kriterium den Wert 0, für ungünstige Dreiecke einen kleineren Wert, der als untere Schranke 1 hat.

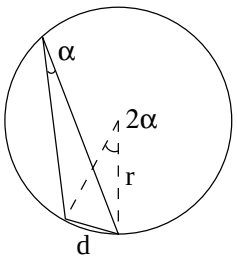


Abbildung 3.29: *Umkreisradius-Kriterium*

Der Qualitätsparameter $\mathcal{B} = \frac{r}{d}$ lässt sich in ein Winkelkriterium überführen. Abbildung 3.29 ist zu entnehmen, dass $d = 2r \sin\alpha$ ist. Umgeformt würde dies für den kleinsten Innenwinkel im Netzwerk bedeuten:

$$\alpha = \arcsin \frac{1}{2\mathcal{B}} \tag{3.8}$$

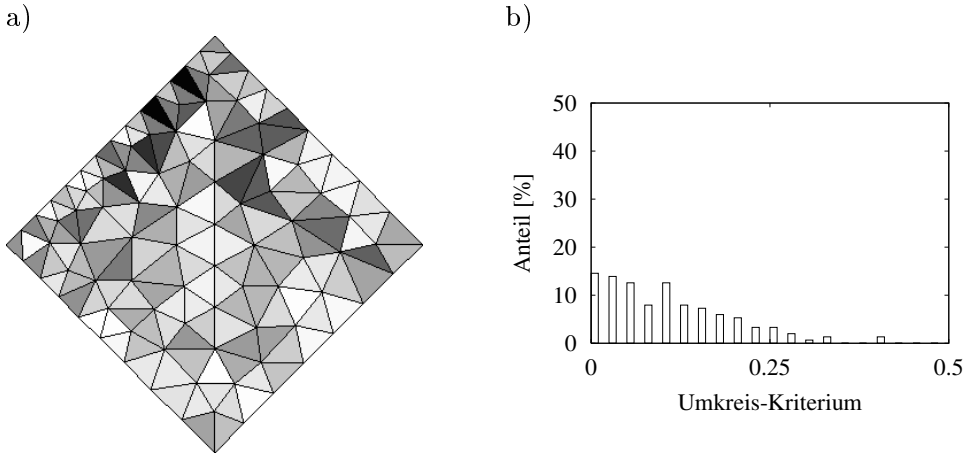


Abbildung 3.30: *Umkreisradius-Kriterium a) Netz (weiß $\hat{=}$ gute Qualität, schwarz $\hat{=}$ schlechte Qualität), b) Verteilung*

Bei einer Vorgabe von $\mathcal{B} < 1$ ergibt sich daraus für den kleinsten Innenwinkel, dass er größer als 30°sein muss . Dieses Kriterium ist vorteilhaft bei der Delaunay-Triangulierung zur Identifikation von ungünstigen Dreiecken anzuwenden, da bei der Delaunay-Triangulierung der Umkreisradius normalerweise bekannt ist. Die Verteilung der Qualität der Elemente ist entsprechend Abbildung 3.30 b) meist linear bis quadratisch, wodurch auch relativ schlechte Elemente noch gut aufgefunden werden können.

Das Umkreis-Kriterium sucht entsprechend der Gleichung 3.8 nach den Elementen, die den kleinsten Innenwinkel haben. Elemente mit zwei kleinen Innenwinkeln werden als schlecht identifiziert.

Der Wertebereich von $\mathcal{B}$ liegt zwischen 0 und 1, wobei 0 für Elemente mit dem kleinsten Innenwinkel von 60°und 1 für Elemente mit dem kleinsten Innenwinkel von 30°steht. Eine Anwendung des Kriteriums ist nur bei Dreieck- und Tetraederelementen möglich. Für andere Elementarten wie Vierecke oder Hexaeder ist der kleinste Innenwinkel ein ungünstiger Indikator für die Qualität.

3.4.2.2 Form-Kriterium

Das Form-Kriterium α für Dreiecke wurde von [Cavendish 1974] vorgeschlagen. Definiert ist es als Quotient aus der Dreieckfläche und den Summen der Quadrate der Kantenlängen sowie einem Normierungsfaktor:

$$\alpha = \frac{4 \cdot \sqrt{3} \cdot A}{\sum l_i^2} \tag{3.9}$$

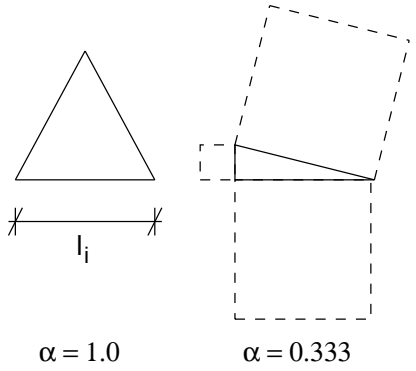


Abbildung 3.31: *Form-Kriterium*

Hierbei ist A die Fläche des Dreiecks und l_i sind die Längen der Kanten. Die Qualität eines Dreiecks ist um so besser, je größer der Wert α ist. Die Obergrenze ist $\alpha = \frac{4 \cdot \sqrt{3} \cdot \sqrt{3}}{12} = 1.0$ für ein gleichseitiges Dreieck.

Einige Algorithmen zur Generierung von Finite-Elemente-Netzen benötigen zum Erreichen des Delaunay-Kreiskriteriums ein Wechseln der Diagonale zwischen zwei benachbarten Dreiecken (siehe Abbildung 3.31). Zum Aufzeigen eines notwendigen Wechsels ist das Form-Kriterium ein brauchbares Maß.

Das Form-Kriterium wird im Rahmen dieser Arbeit auf Vierecke, Tetraeder und Hexaeder übertragen. Für Vierecke ergibt sich:

$$\alpha = \frac{4 \cdot A}{\sum l_i^2} \tag{3.10}$$

Im Dreidimensionalen zieht man für den Vergleich den Quotienten des Volumens zu der Summe der Volumen der Würfel über den Kantenlängen heran. Für Tetraeder ergibt sich:

$$\alpha = \frac{12 \cdot \sqrt{2} \cdot V}{\sum l_i^3} \tag{3.11}$$

Für Hexaeder ergibt sich:

$$\alpha = \frac{12 \cdot V}{\sum l_i^3} \tag{3.12}$$

Für optimale Elemente ergibt das Form-Kriterium den Wert 1, für ungünstige Elemente einen kleineren Wert, der als untere Schranke 0 hat.

Das Form-Kriterium sucht nach Gleichung 3.9 nach den Elementen, bei denen eine Kante eine deutlich andere Länge als die anderen Kanten hat. Hierdurch werden unter anderem gute Elemente mit einem oder zwei spitzen bzw. einem stumpfen Innenwinkel gefunden. Das Kriterium liefert Werte im Bereich von 0 und 1, wobei 1 für Elemente mit gleich langen Kanten und 0 für Elemente mit stark unterschiedlich langen Kanten steht.

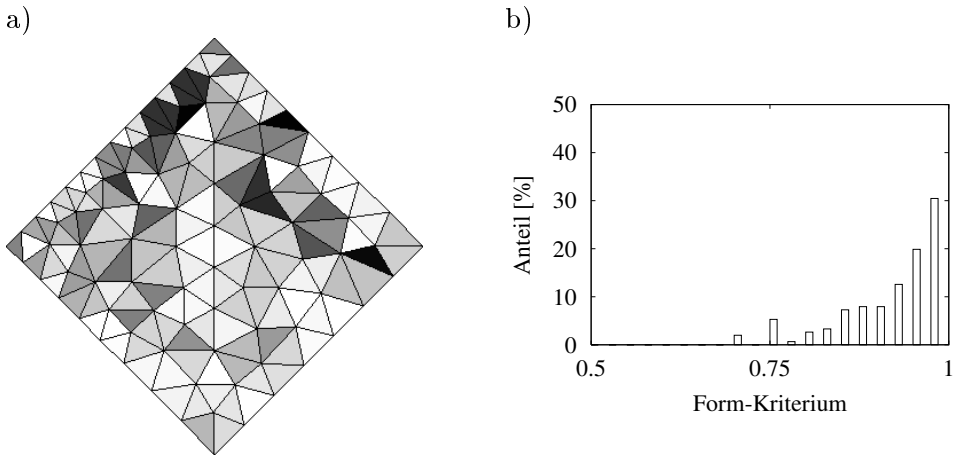


Abbildung 3.32: *Form-Kriterium a) Netz (weiß ≙ gute Qualität, schwarz ≙ schlechte Qualität), b) Verteilung*

Die Verteilung der Qualität der Elemente ist entsprechend Abbildung 3.32 b) meist linear bis quadratisch, wodurch auch relativ schlechte Elemente noch gut gefunden werden können. Eine Anwendung des Form-Kriteriums ist für beliebige Elemente möglich und sinnvoll.

3.4.2.3 Geometrie-Kriterium

[Shimada u. a. 1997] schlagen zum Bewerten von Netzen ein Geometrie-Kriterium vor. Dieses ist definiert durch:

$$\epsilon_g = \frac{1}{m}\sum_{i=1}^{m}(0.5 - \frac{r_i}{R_i}) \tag{3.13}$$

Das Geometrie-Kriterium teilt den Innenkreisradius durch den Umkreisradius. Die Erweiterung des Kriteriums ist problemlos für Tetraeder möglich. Hierbei müssen die Radien der Kreise nur durch die Radien der Kugeln ersetzt werden.

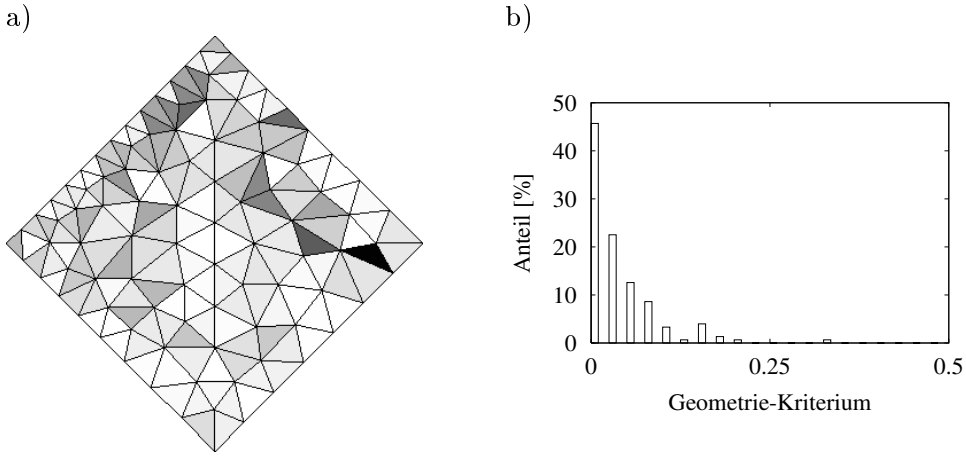


Abbildung 3.33: *Geometrie-Kriterium a) Netz (weiß ≙ gute Qualität, schwarz ≙ schlechte Qualität), b) Verteilung*

Dieses Verhältnis des Innenkreises zum Umkreis ist für gleichseitige Dreiecke 0.5 und für stumpfwinklige Dreiecke strebt es gegen 0. Je kleiner der Indikator ϵ_g ist, desto regulärer ist das jeweilige Element. Als obere Grenze kann der Indikator den Wert 1 annehmen. Dreiecke mit einem spitzen Innenwinkel werden schlecht erkannt.

Die Verteilung der Qualität der Elemente ist entsprechend Abbildung 3.33 b) meist quadratisch bis logarithmisch, wodurch auch relativ schlechte Elemente kaum aufgespürt werden können. Der Wertebereich ist normiert. Eine Anwendung des Indikators ist nur für Dreiecke und Tetraeder möglich.

3.4.2.4 Gemischter Indikator

Ein gemischter Indikator wird von [Schöberl 1997] vorgestellt. Dieser Indikator kann zum einen ungünstige Elemente aufspüren, aber auch die Qualität eines gesamten Netzwerkes beurteilen. Die Qualität eines Dreiecks wird mit:

$$E(T) = \frac{\sqrt{3}}{36}\frac{(\sum_{i=1}^{3} l_i)^2}{A} + \sum_{i=1}^{3}(\frac{l_i}{h_i} + \frac{h_i}{l_i} - 2) \tag{3.14}$$

und die Qualität eines Tetraeder mit:

$$E(T) = \frac{1}{6^4\sqrt{2}}\frac{(\sum_{i=1}^{3} l_i)^3}{V} + \sum_{i=1}^{3}(\frac{l_i}{h_i} + \frac{h_i}{l_i} - 2) \tag{3.15}$$

berechnet. Hierbei sind l_i die Kantenlängen und h_i die zugehörigen Höhen, A die Dreieckfläche und V das Volumen. Der Optimalwert von $E(T)$ ist 1, ungünstige Triangulierungen bzw. Elemente haben größere Werte. Eine obere Schranke kann nicht angegeben werden.

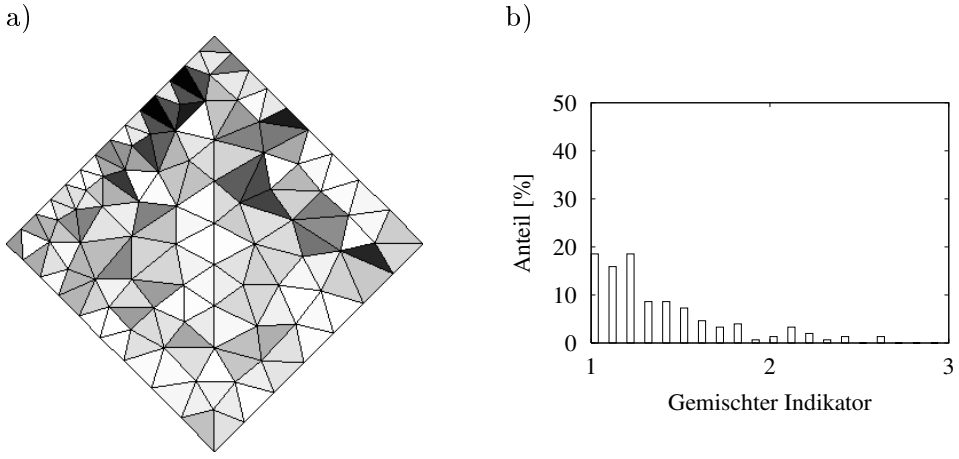


Abbildung 3.34: *Gemischter Indikator a) Netz (weiß ≙ gute Qualität, schwarz ≙ schlechte Qualität), b) Verteilung*

Der in Gleichung 3.14 dargestellte Indikator besteht aus zwei Teilen. Der erste Teil $\frac{\sqrt{3}}{36}\frac{(\sum_{i=1}^{3} l_i)^2}{A}$ sucht nach stumpfwinkligen Elementen, der zweite Teil $\sum_{i=1}^{3}(\frac{l_i}{h_i} + \frac{h_i}{l_i} - 2)$ nach Elementen mit kleinem Innenwinkel.

Die Verteilung der Qualität der Elemente ist entsprechend Abbildung 3.34 b) meist linear bis quadratisch. Eine Normierung des Wertebereichs ist nicht vorhanden, wodurch das Aufspüren relativ schlechter Elemente erschwert wird. Das gleichzeitige Bewerten nach zwei Kriterien hat Vor- und Nachteile. Eine Optimierung nach mehreren Kriterien ist allgemein immer sinnvoll. Durch die Mischung ist nicht festzustellen welches Teilkriterium verletzt wird. Eine Teilung des Kriteriums in zwei Teilkriterien, für die jeweils die Qualität bestimmt wird, erscheint sinnvoll. Eine Anwendung des Indikators ist nur für Dreiecke und Tetraeder möglich.

3.4.3 Bewertung der Kriterien

Bei der Bewertung der Kriterien ist auf verschiedene Bedingungen zu achten. Zum einen sollte ein Indikator für beliebige Elemente anwendbar sein und zum anderen sollte der Indikator nicht nur das schlechteste Element, sondern auch Elemente mit relativ schlechter Qualität aufzeigen. Eine Normierung des Wertebereichs ist für eine Implementierung in Programmen vorteilhaft.

Das Umkreisradius-Kriterium, das Form-Kriterium und der gemischte Indikator zeigen gut ungünstige, zu verbessernde Elemente auf und sind daher für Dreieckelemente gleichwertig. Wegen der oben beschriebenen Nachteile ist das Geometrie-Kriterium für Dreiecknetze nicht zu empfehlen. Das Form-Kriterium wurde im Rahmen dieser Arbeit den anderen Kriterien zur Bewertung der Qualität von Elementen und Elementnetzen vorgezogen, da es auf beliebige Elementarten anwendbar ist.

Ein Vergleich des Topologie-Kriteriums mit den anderen Kriterien ist schlecht möglich und nicht sinnvoll, da dieses knoten- und nicht elementorientiert ist. Aufgrund seiner einfachen Anwendbarkeit auf beliebige Elementnetze erscheint es als guter Indikator für Netze.

4 Parallele Netzgenerierung

Bei der Berechnung großer mechanischer Systeme mit Hilfe der Methode der Finiten-Elemente hat sich die Verwendung von Parallelrechnern als vorteilhaft erwiesen. Für eine Berechnung analog der in Abschnitt 2.3 beschriebenen Substrukturtechnik ist eine nicht überlappende Gebietsaufteilung notwendig. Bei sehr großen Systemen kann es vorkommen, dass das Finite-Elemente-Netz nicht in den Hauptspeicher eines Rechners geladen werden kann. Hier bietet sich die parallele Netzgenerierung an, da diese jeweils nur Teilnetze auf jedem einzelnen Prozessor speichert. Ein angenehmer Nebeneffekt der parallelen Netzgenerierung ist, dass eine zeitaufwendige Partitionierung des Elementnetzes entfällt.

Die hier verwendete Vorgehensweise zur parallelen Netzgenerierung eignet sich sowohl für zweidimensionale als auch für dreidimensionale Netze, wobei die in Abschnitt 3 beschriebenen Probleme der Hexaedervernetzung in diesem Bereich Einschränkungen machen. Die Form der umschließenden Geometrie kann beliebig sein, sie kann sogar aus mehreren Teilen bestehen. Lokale Verfeinerungen des Netzes sind möglich.

4.1 Vorgehensweise

Der im Rahmen dieser Arbeit verwendete Ansatz zur Parallelisierung eines Netzgenerators arbeitet wie folgt:

Teilen der Geometrie: Die Geometrie wird, wie in Abbildung 4.1 dargestellt, rekursiv mit der Schwerachsenmethode [Farhat und Lesoinne 1993] in n Schritten in 2^n Teilgebiete geteilt. Eine Teilung in zwei etwa gleich große Teilgebiete wird in jedem Rekursionsschritt erreicht. Die Qualität der Teilung ist problemabhängig und wird durch die Dichtefunktion, die die Elementgrößen beschreibt, gesteuert. Die Teilung läuft während des Programmablaufes parallel für alle Prozessoren ab, wobei jeder Prozessor sein eigenes Teilgebiet berechnet. Für die Gebietszerlegung ist daher keine Kommunikation zwischen den Prozessoren notwendig.

Die Diskretisierung des Randes wird entsprechend der Verteilungsfunktion durchgeführt. Danach wird der Schwerpunkt der Geometrie mit den Koordinaten x_s, y_s und im dreidimensionalen Fall zusätzlich mit z_s bestimmt und eine Koordinatentransformation des Ursprungs in den Schwerpunkt vorgenommen. Die Eigenvektoren der Flächenträgheitsmomente I_x, I_y und I_z sowie der Deviationsmomente I_{xy}, I_{yz} und I_{xz} ergeben die Schnittachsen- bzw. Schnittebenenrichtung. Die Geometrie wird nun entlang der Schnittachse geteilt, wobei der Rand nur an vorher vorhandenen Knoten geschnitten werden darf. Hierdurch kann es zu Knicken in der Schnittlinien kommen.

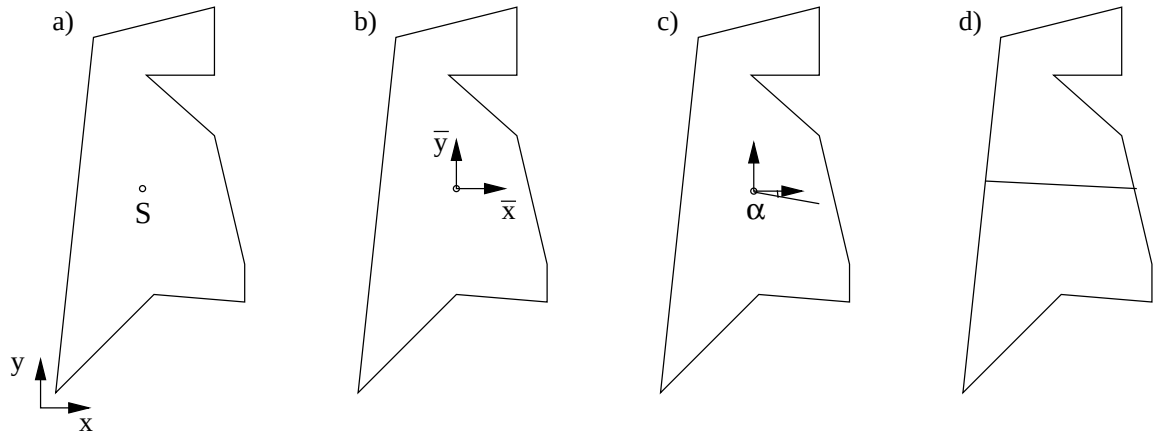


Abbildung 4.1: *Rekursives Teilen mit der Schwerachsenmethode*

Festlegung der Koppelknoten: Nach jeder Teilung der Geometrie wird die Diskretisierung auf den Koppelrändern festgelegt, sodass während der Netzgenerierung keine Kommunikation notwendig ist. Das ist auch der Grund, warum eine Anwendung der Methode bei der Erzeugung von dreidimensionalen Hexaedernetzen nicht möglich ist. Hexaedernetze lassen sich nicht mit einer vorgegebenen Randdiskretisierung erstellen.

Generierung der Teilnetze: In einem letzten Schritt wird nun parallel auf allen Prozessoren ein Netz für das Teilgebiet mit einem sequentiellen Netzgenerator erzeugt. Durch die vorgegebene Randdiskretisierung ist eine Kompatibilität der Elemente automatisch gegeben.

4.2 Beschreibung der Geometrie

Im Rahmen der Arbeit wird die Parallelisierung für zweidimensionale und zweieinhalbdimensionale Netze realisiert. Für die Netzgenerierung ist eine eindeutige Beschreibung der zu vernetzenden Geometrie notwendig.

Geometrie: Eine für zweidimensionale Probleme übliche Beschreibung wird verwendet, die aus berandenden Polygonzügen, Löchern, inneren Linien und Fixpunkten bestehen kann. Jedem Punkt können Höheninformationen zugeordnet werden. Die Notwendigkeit mehrerer berandender Polygonzüge ergibt sich aus dem Ansatz der Parallelisierung, der auf der Teilung der Geometrie basiert und unter Umständen einen Polygonzug in mehrere Teilpolygonzüge zerteilt (Abbildung 4.2).

Punkt: Ein Punkt ist ein geometrischer Ort, der durch seine dreidimensionalen Koordinaten beschrieben wird. Jeder Punkt kann einen Wert für die angestrebte Kantenlänge der Elementkanten in der Nähe des Punktes sowie weitere z-Koordinaten aufweisen.

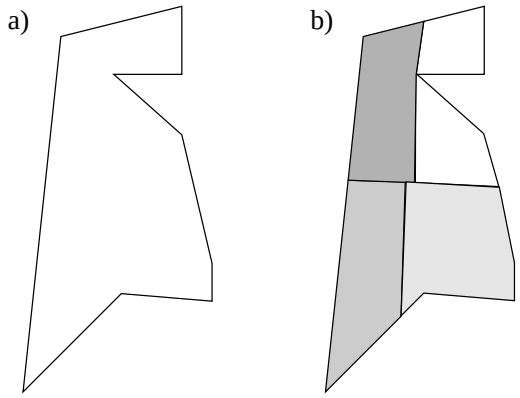


Abbildung 4.2: *a) Ausgangspolygon, b) mehrere Polygonzüge im weißen Teilgebiet auf einem Prozessor nach Teilung für vier Prozessoren*

Polygonzug: Die berandenden Polygonzüge sind geschlossene Polygone, die jeweils aus mehreren Kanten bestehen, die auf jeweils zwei Punkte zeigen. Durch die Richtung der Polygonzügen kann man einfach festlegen, ob ein Punkt außerhalb oder innerhalb eines Polygonzuges liegt. Daher wird definiert, dass die Gebietsberandung im mathematisch negativen Drehsinn angegeben werden muss.

Linie: Eine Linie ist die direkte Verbindung zwischen zwei Punkten. Während der Netzgenerierung muss darauf geachtet werden, dass kein Element eine Linie überdeckt. Linien werden vor allem zum Einbau der Randbedingungen in das Finite-Elemente-Netz benötigt.

Fixpunkt: Fixpunkte sind Punkte innerhalb einer Geometrie, die der Nutzer aus verschiedenen Gründen benötigt. So kann es z.B. notwendig sein, an einem Punkt ein Ergebnis zu erhalten oder eine Randbedingung an eine bestimmte geometrische Stelle zu legen.

Loch: Ein Loch wird durch ein geschlossenes Polygon beschrieben, wobei hier der Umfahrungssinn entgegengesetzt der der Randpolygone ist. Durch Teilungen der Geometrie kann es vorkommen, dass aus dem Polygonzug eines Loches ein Randpolygon wird.

4.3 Teilung der Geometrie

Beim Teilen der Geometrie sind einige Besonderheiten zu beachten. In Abbildung 4.3 a) ist eine Geometrie dargestellt, die aus einem Polygonzug, einem Loch, einer Linie und einem Punkt besteht. Die Berandung besteht aus einem Polygonzug, dessen Umlaufsinn mathematisch negativ ist. Das Loch ist ein Polygonzug in mathematisch positiver Richtung.

In Abbildung 4.3 b) ist die Diskretisierung der Polygonzüge und Linien sowie die erste Schnittlinie dargestellt. Die Schnittlinie schneidet die Polygonzüge immer an Knoten.

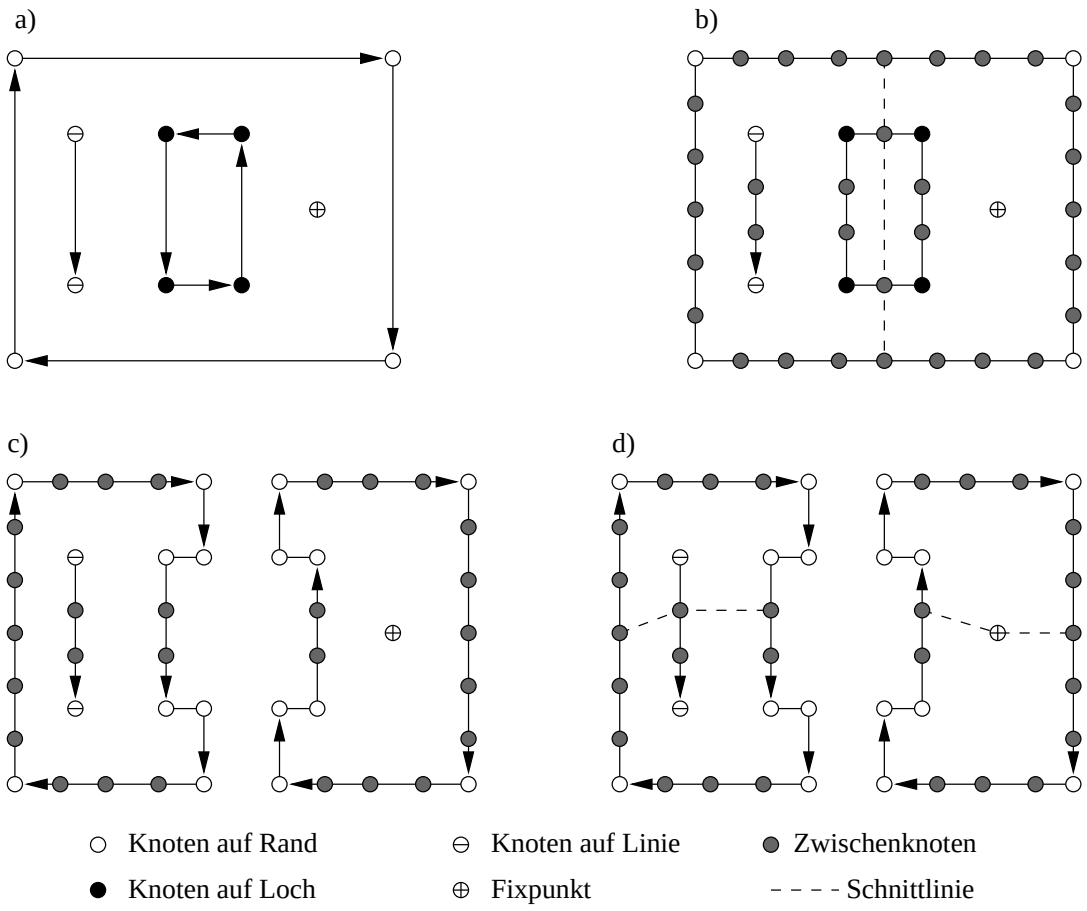


Abbildung 4.3: *Teilung einer Geometrie*

Beim Teilen der Geometrie wird der Randpolygonzug geteilt (Abbildung 4.3 c)). Falls Löcher geteilt werden müssen, werden diese in das Randpolygon integriert. Unter Umständen kann es bei der Teilung vorkommen, dass die Richtung von Teilen der Polygonzügen umgedreht werden muss.

Linien dürfen nur an Knoten geschnitten werden. Schneidet eine Schnittlinie eine Linien, ist die Schnittlinie durch einen Knoten der Linie zu legen. Die Schnittlinie kann danach einen Knick aufweisen (Abbildung 4.3 d), links). Liegt ein Fixpunkt näher als eine Elementkantenlänge entfernt von einer Schnittlinie, ist die Schnittlinie ebenfalls durch den Fixpunkt zu legen (Abbildung 4.3 d), rechts).

4.4 Geometrieverarbeitung

Beim Teilen der Geometrie ist es notwendig festzustellen, welche Teile der Originalgeometrie weiter verwendet werden. Hierfür muss überprüft werden, auf welcher Seite des Teilungspolygons ein Punkt, eine Kante oder ein Loch liegen. Die hierfür notwendigen geometrischen Überprüfungen werden im Folgenden dargestellt:

Punkt auf Punkt: Zur Überprüfung, ob zwei Punkte an der selben Stelle liegen, ist es notwendig dem zu überprüfenden Punkt P eine ϵ-Umgebung zuzuweisen, innerhalb derer alle weiteren Punkte als geometrisch gleich angesehen werden. In Abbildung 4.4 ist die ϵ-Umgebung des Punktes P dargestellt. Punkt A wird als geometrisch gleich angesehen, da er in der ϵ-Umgebung liegt, der Punkt B nicht. Bei der Überprüfung wird das Quadrat des Abstands der Punkte mit Hilfe des Satzes von Pythagoras bestimmt und mit dem Quadrat von ϵ verglichen. Die Ermittlung eines Wertes für ϵ muss in Abhängigkeit der Abmessungen der Geometrie geschehen.

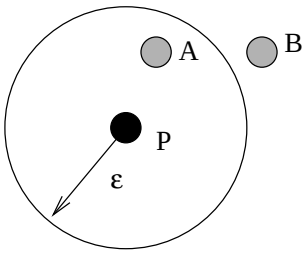


Abbildung 4.4: *ϵ-Umgebung eines Punktes P*

Punkt auf Linien: Die Überprüfung, ob ein Punkt auf einer Linie liegt, kann nur unter Zuhilfenahme einer ϵ-Umgebung der Linie erfolgen. Diese Umgebung muss, wie in Abbildung 4.5 dargestellt, aus einer ϵ-Umgebung der Kante sowie den beiden ϵ-Umgebungen der Punkte bestehen. Die Überprüfung ist daher zweigeteilt. Zum einen wird der Punkt-auf-Punkt Test für beide Endpunkte durchgeführt (Punkt D). Zum anderen wird der Lotfußpunkt des zu untersuchenden Punktes auf der Linie bestimmt (Punkt C). Falls dieser zwischen den Endpunkten liegt, wird das Quadrat des Abstands des Punktes und des Lotfußpunktes mit dem Quadrat von ϵ verglichen. Die Festlegung eines Wertes von ϵ kann entweder in Abhängigkeit der Gesamtgeometrie oder in Abhängigkeit der Länge der Kante geschehen.

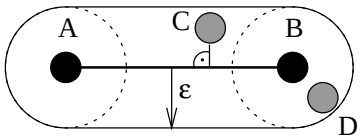


Abbildung 4.5: *ϵ-Umgebung einer Linie*

Punkt in Polygon: Ob ein Punkt innerhalb eines Polygonzuges liegt, ist wesentlich aufwendiger festzustellen als die Durchführung der beiden oben dargestellten Tests. Zur Überprüfung wird ein Punkt P im Unendlichen erzeugt (siehe Abbildung 4.6). Von dem zu überprüfenden Punkt wird ein Strahl zum Punkt P gezogen und die Anzahl der geschnittenen Kanten ermittelt. Ist die Anzahl wie bei Punkt A ungerade, liegt der Punkt innerhalb

des Polygons. Ist die Anzahl gerade wie bei Punkt B, liegt der Punkt außerhalb des Polygons. Der Strahl von Punkt C zum Punkt P geht durch einen Eckpunkt des Polygons. Hier ist eine Sonderbehandlung notwendig, da festgestellt werden muss, ob der Strahl den Polygonzug nur berührt oder ihn an dem Eckpunkt schneidet.

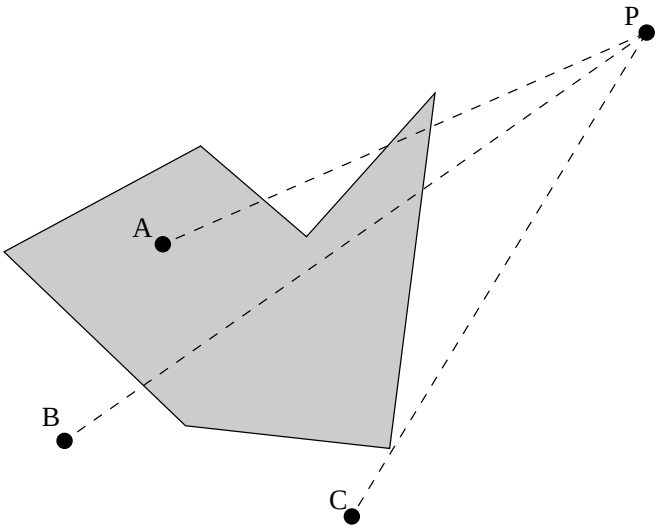


Abbildung 4.6: *Überprüfung Punkt in Polygon*

4.5 Lastverteilungsfunktion

Um auf allen Prozessoren eine gleich große Rechenlast zu erreichen, wird eine Lastverteilungsfunktion für die Geometrie eingeführt. An Stellen, an denen kleinere Element entstehen sollen, hat diese Funktion größere Werte, an Stellen, an denen größere Elemente angestrebt werden, kleinere Werte. Diese Lastverteilungsfunktion wird bei der Berechnung des Schwerpunktes bzw. der Trägheitsmomente berücksichtigt und bewirkt eine Verschiebung des Schwerpunktes des Gebietes in Richtung der kleineren Elemente. Zur Berechnung des Schwerpunktes ergeben sich folgende Gleichungen:

$$x_s = \frac{\int x \cdot f(x,y)dA}{\int f(x,y)dA} \tag{4.1}$$

$$y_s = \frac{\int y \cdot f(x,y)dA}{\int f(x,y)dA} \tag{4.2}$$

Die Trägheitsmomente bzw. Schwerachsen berechnen sich nach der Koordinatentransformation dann wie folgt:

$$I_{\bar{x}} = \int \bar{y}^2 \cdot f(\bar{x},\bar{y})dA \tag{4.3}$$

$$I_{\bar{y}} = \int \bar{x}^2 \cdot f(\bar{x},\bar{y})dA \tag{4.4}$$

$$I_{\bar{x}\bar{y}} = -\int \bar{x} \cdot \bar{y} \cdot f(\bar{x},\bar{y})dA \tag{4.5}$$

Die Lastverteilungsfunktion ist abhängig von der anzustrebenden Kantenlänge der Elemente und wird für zweidimensionale Netze folgendermaßen berechnet:

$$f(\bar{x}, \bar{y}) = \frac{1}{w^2} \tag{4.6}$$

wobei w die angestrebte Kantenlänge an der Stelle x, y ist. Für dreidimensionale Netze ergibt sich:

$$f(\bar{x}, \bar{y}) = \frac{1}{w^3} \tag{4.7}$$

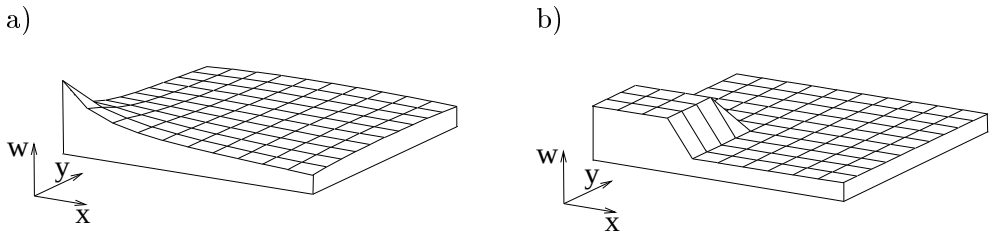


Abbildung 4.7: *Lastverteilungsfunktion von Verfeinerungen*

In Abbildung 4.7 a) ist die Lastverteilungsfunktion für ein Viereck mit feineren Elementen in einer Ecke dargestellt. Das resultierende FE-Netz mit Viereckelementen ist in Abbildung 4.8 a) dargestellt.

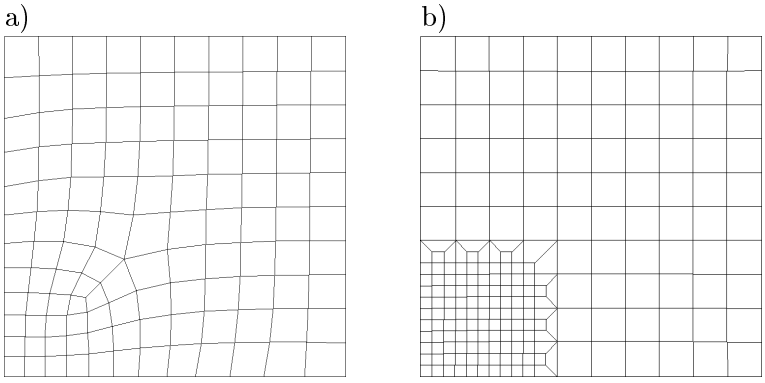


Abbildung 4.8: *FE-Netz mit Verfeinerungen*

Verfeinerungen, die mit der dreier-Verfeinerung nach Abschnitt 3.3.2 und 3.3.4 in ein Netz eingebaut werden sollen, benötigen eine spezielle Lastverteilungsfunktion. In 4.7 b) ist die Funktion zur Verfeinerung einer Ecke dargestellt. In der Ecke mit der Verfeinerung ist die Elementkantenlänge $\frac{1}{3}$ so lang wie im Rest des Netzes. Daher muss der Wert der Lastverteilungsfunktion hier 9 bzw. 27 mal so groß sein. Das resultierende FE-Netz mit Viereckelementen ist in Abbildung 4.8 b) dargestellt.

Die Lösung der Integrale wird mit Hilfe der Gaußintegration durchgeführt. Hierfür ist eine Zerlegung der gesamten Geometrie in grobe Dreiecke notwendig. Diese Triangulierung verwendet nur die vorher schon vorhandenen Knoten und erzeugt keine neuen.

4.6 Beispiel

An einem Beispiel soll nun der Parallelisierungsansatz demonstriert werden. Eine Darstellung der Geometrie des Problems ist in Abbildung 4.1 und 4.2 zu sehen. Anhand von zwei verschiedenen Diskretisierungen der Geometrie sollen Grenzen des Ansatzes gezeigt werden.

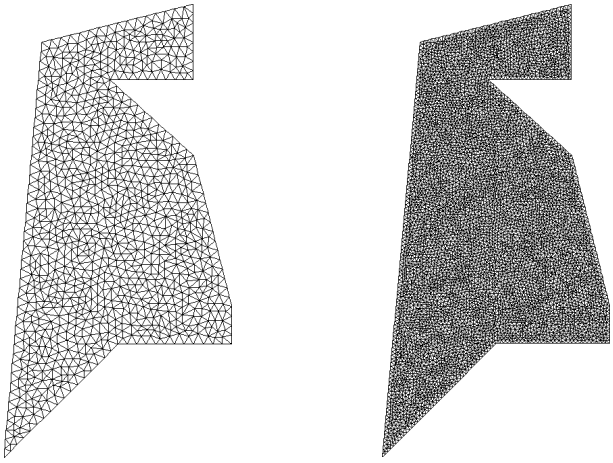


Abbildung 4.9: *Netz mit Diskretisierung 1 und Diskretisierung 2*

Die sequentiell erzeugten Diskretisierungen mit Dreieckelementen sind in Abbildung 4.9 dargestellt. Diskretisierung 1 hat ca. 1900 Elemente, Diskretisierung 2 ca. 12500 Elemente.

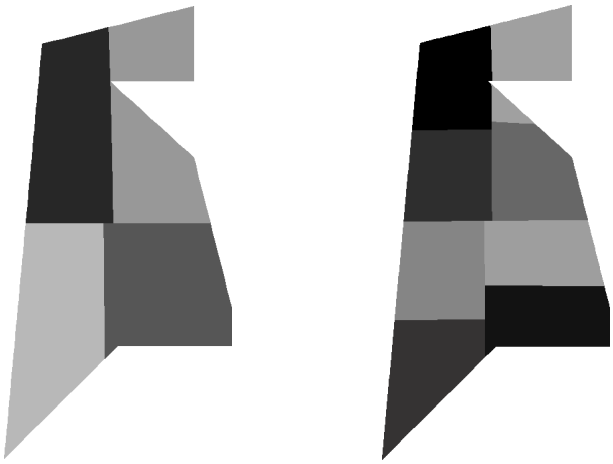


Abbildung 4.10: *Partitionierung mit vier und acht Prozessoren*

Die Partitionierung der Diskretisierung 1 in vier Teilgebiete ist in Abbildung 4.10 a) dargestellt. In Abbildung 4.10 b) ist die in acht Teilgebiete aufgeteilte Diskretisierung 2 zu sehen. Unterschiede in der Teilung sind unter anderem an der einspringenden Ecke zu erkennen. Bei der Diskretisierung 1 mit der geringeren Anzahl der Elemente wird die Ecke genau getroffen, während bei der Diskretisierung 2 die Schnittlinie rechts von der Ecke verläuft. Dieser Unterschied liegt an der feineren Auflösung des Randes bei der Diskretisierung 2, wodurch die Schnittlinien näher an den Schwerachsen liegen können.

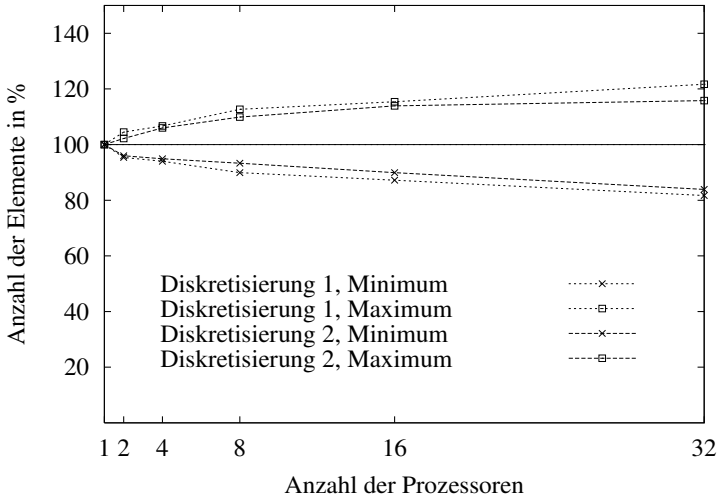


Abbildung 4.11: *Partitionierung*

Die durchschnittlichen, minimalen und maximalen Elementanzahlen der Partitionen der beiden Diskretisierungen sind in Tabelle 4.1 und 4.2 zu sehen. In Abbildung 4.11 sind für beide Diskretisierungen der Quotient aus Minimum bzw. Maximum der Elementanzahl und der durchschnittlichen Elementanzahl dargestellt. Zu erkennen ist, dass die Diskretisierung mit der höheren Elementanzahl und damit den feineren Elementen geringere Abweichungen der Minima und Maxima in den Elementanzahlen gegenüber dem Durchschnitt hat. Dies liegt an der feineren Diskretisierung des Randes, wodurch die Schnittlinie näher an der Schwerachse liegen kann.

4.6.1 Laufzeitverhalten

Die Gesamtlaufzeit der parallelen Berechnung setzt sich aus mehreren Teilen zusammen. Ein Teil ist die Netzgenerierung, ein anderer Teil die Geometrieaufteilung. Der restliche Teil setzt sich unter anderem aus dem Einlesen der Daten, Bestimmen der Netzqualität und Ausgeben der Ergebnisdaten zusammen. In Abbildung 4.12 sind die Zeiten für beide Diskretisierungen für jeweils 2 und 32 Prozessoren dargestellt. Das sequentiell ablaufende Teilen der Geometrie dauert bei einer größeren Anzahl von Elementen absolut gesehen länger. Da die Gesamtzeit bei einer höheren Prozessorenanzahl geringer ist, wirkt sich dieser Anteil relativ gesehen noch stärker aus.

Anzahl der Prozessoren	Elementanzahl Durchschnitt	Elementanzahl Minimum	Elementanzahl Maximum
1	1924	1924	1924
2	960	916	1004
4	474	446	506
8	228	205	257
16	117	102	135
32	60	49	73

Tabelle 4.1: *Elementanzahlen der Netze der Diskretisierung 1*

Anzahl der Prozessoren	Elementanzahl Durchschnitt	Elementanzahl Minimum	Elementanzahl Maximum
1	12716	12716	12716
2	6370	6115	6525
4	3150	2992	3339
8	1606	1498	1766
16	788	709	898
32	409	343	474

Tabelle 4.2: *Elementanzahlen der Netze der Diskretisierung 2*

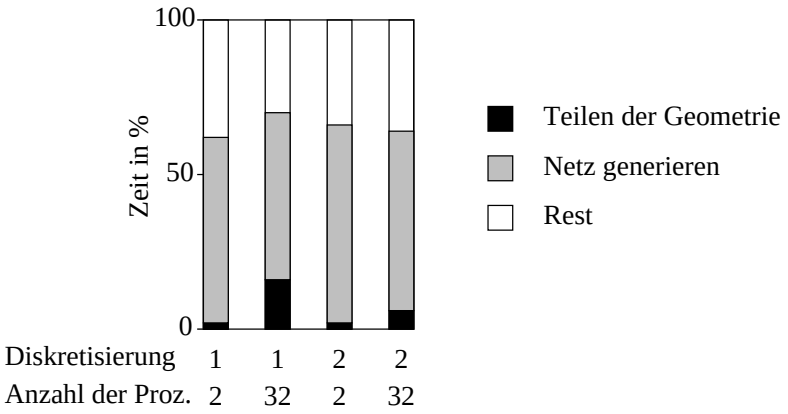


Abbildung 4.12: *Zeiten*

Die zu vernetzende Fläche ist bei höheren Elementanzahlen kleiner. Hierdurch verringert sich die Zeit für das parallel ablaufende Vernetzen bei größerer Prozessorenanzahl.

Die dargestellten Zeitmessungen wurden auf einem Linux-Cluster mit Pentium-II 350 MHz Prozessoren mit jeweils 128 MB Speicher und einem 100 MBit Switch Netzwerk durchgeführt.

Für Diskretisierung 1 beträgt der Zeitanteil für das Teilen der Geometrie 1.89 % bei zwei Prozessoren und 19.19 % bei 32 Prozessoren. Das parallel ablaufende Vernetzen braucht 75.8 % bzw. 67.6 % der Laufzeit. Hierdurch ändert sich das Verhältnis „Netz generieren“ zu „Geometrie teilen“ von 1 : 40.1 auf 1 : 3.5. Für die Diskretisierung 2 beträgt der Anteil für das Teilen 1.45 % sowie 6.92 % der Zeit und der Anteil für das Netzgenerieren 81.0 % bzw. 72.2 %. Das Verhältnis *Netzgenerieren* zu *Geometrieteilen* ist daher 1 : 55.8 sowie 1 : 10.4.

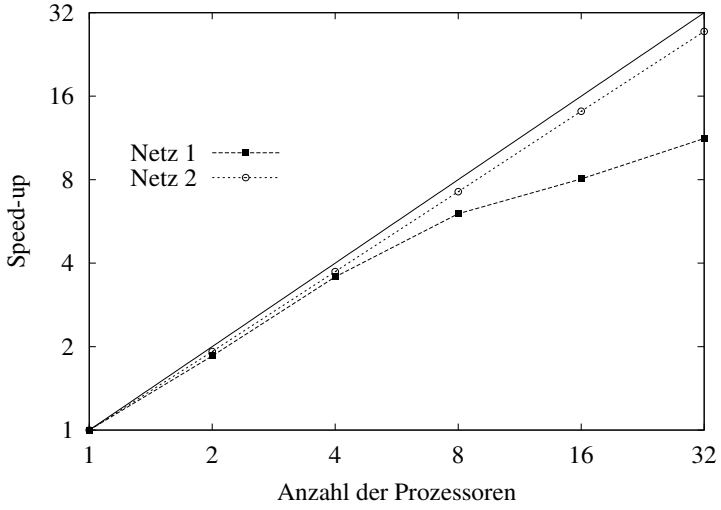


Abbildung 4.13: *Speed-up*

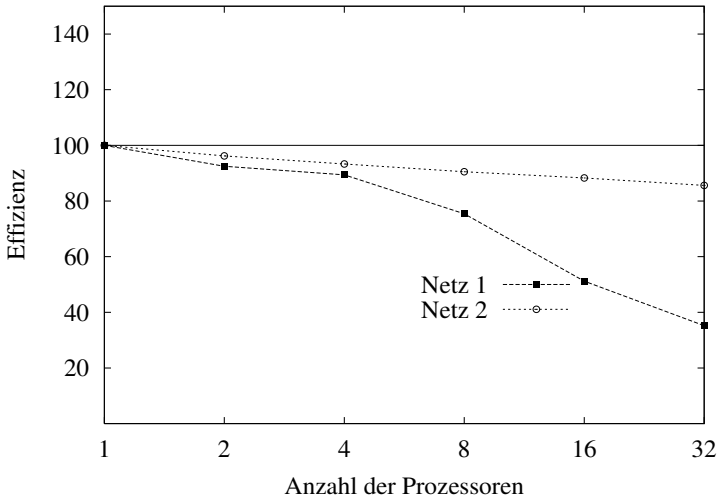


Abbildung 4.14: *Effizienz*

In der Effizienz (Abbildung 4.14) und im Speed-up (Abbildung 4.13) schlagen sich diese Verhältnisse nieder. Die Effizienz der Diskretisierung liegt bei zwei Prozessoren bei 96.2 % und bei 32 Prozessoren bei 85.6 %. Bei der Diskretisierung 1 sieht man, dass eine parallele Berechnung des Netzes mit mehr als vier Prozessoren ungünstig ist. Bei mehr als vier Prozessoren fällt die Kurve der Effizienz stark ab, da hier der sequentielle Anteil durch Teilen bzw. Einlesen und Rausschreiben einen zu großen Anteil an der Gesamtzeit beansprucht. Die Effizienz bei 32 Prozessoren liegt daher bei 35.2 %, was einer Leistung von nur 11.2 Prozessoren entspricht.

4.6.2 Netzqualität

In Abbildung 4.16 ist die Verteilung des Form-Kriteriums für die Diskretisierung 1 für eine Partition und für 32 Partitionen dargestellt. Zu erkennen ist, dass der im Verhältnis deutlich größere Anteil von Elementen am Rand keinen wesentlichen Einfluss auf die Qualität der Elemente hat. Das Form-Kriterium für die einzelnen Elemente ist in Abbildung 4.15 abgebildet. Entlang der Teilungskanten befinden sich Elemente mit besserer Qualität, im Inneren des Gebietes befinden sich die Elemente mit schlechterer Qualität (Abbildung 4.16).

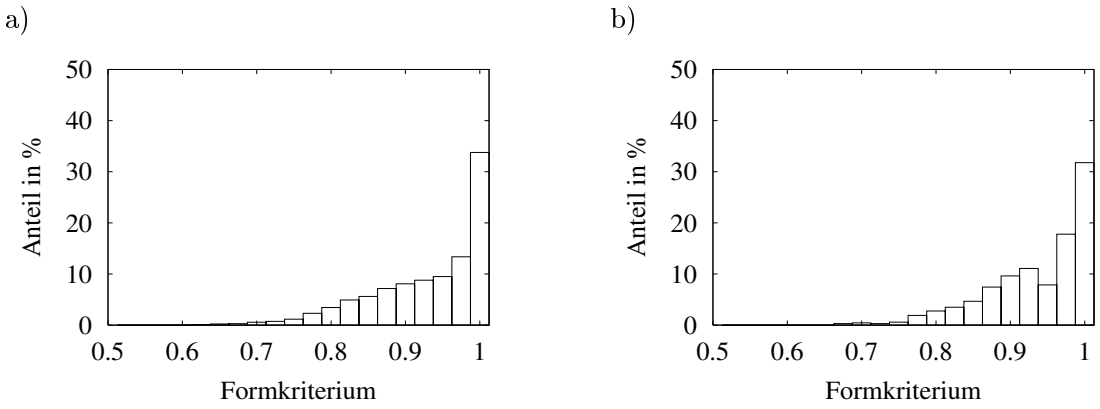


Abbildung 4.15: *Form-Kriterium a) eine Partition b) 32 Partitionen*

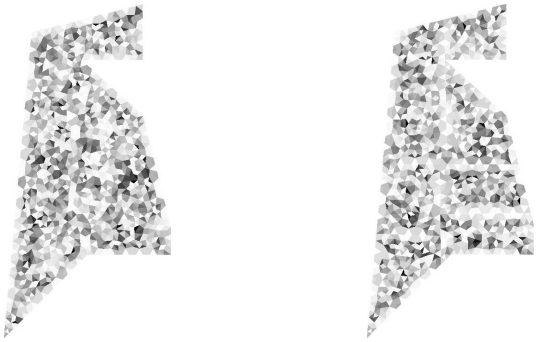


Abbildung 4.16: *Form-Kriterium (weiß $\hat{=}$ gute Qualität, schwarz $\hat{=}$ schlechte Qualität) a) eine Partition b) acht Partitionen*

5 Netzgenerierung geotechnischer Systeme

5.1 Motivation

Zur Simulation der Vorgänge in einer Baugrund-Tragwerk-Struktur hat sich die Methode der Finiten-Elemente als geeignet herausgestellt. Mit ihr ist es möglich, sowohl das komplexe Materialverhalten des Feststoffes abzubilden als auch die zeitabhängigen Setzungs- und Fließvorgänge zu simulieren. Die Beschreibung des Verhaltens des Bodens kann unter Verwendung der „Theorie der Porösen Medien“ die Interaktion der drei Phasen des Bodens, des Korngerüsts, des Fluids und des Gases, unter Verwendung der „Mischungstheorie“ und des „Konzept der Volumenanteile“ geschehen [Hartmann und Meißner 1987] [Ehlers 1998]. Die Simulation der zeitabhängigen Vorgänge der räumlichen Baugrund-Tragwerk-Struktur ist auf Grund der Komplexität und der Problemgröße für praxisrelevante Problemstellungen auf einem seriellen Rechner nicht mehr sinnvoll durchführbar. Eine parallel ablaufende Finite-Elemente-Simulation ermöglicht die Handhabung größerer Systeme.

Baugrund-Tragwerk-Strukturen lassen sich unter Berücksichtigung des komplexen geotechnischen Systems sowie der zugehörigen Lasten und Randbedingungen aufgrund ihrer Komplexität nur schwer manuell diskretisieren. Eine Automatisierung der Vernetzung ist hier daher ebenso wie eine Partitionierung der resultierenden Diskretisierung erforderlich. Im Rahmen dieser Arbeit wird daher untersucht, wie man durch eine parallele Netzgenerierung ein automatisch partitioniertes Finite-Elemente-Netz einer Baugrundstruktur erhält.

5.2 Projektionstechnik

Zur Finite-Elemente-Simulation der Baugrund-Tragwerk-Struktur wird unter anderem das Programm FEAP [Taylor 1998] mit der in [Ehlers 1998] beschriebenen Implementierung der Mehrphasentheorie verwendet. Aufgrund der numerischen Vorzüge werden hier Hexaederelemente den Tetraederelemeten vorgezogen. Wie schon in Abschnitt 3.1.6 beschrieben, existieren zur Zeit noch keine allgemeingültigen Algorithmen zur Erzeugung von Hexaedernetzen. Das Erzeugen der Baugrunddiskretisierung wird daher mit der Projektionstechnik durchgeführt. Die Projektionstechnik ist eine Vorgehensweise zur Erzeugung von Prisma-Netzen, die auf einer $2\frac{1}{2}D$-Darstellung der Geometrie basiert und aus den folgenden Schritten besteht:

Schritt 1: Alle geometrischen Gegebenheiten des CAD-Modells (Abbildung 5.1 a)), wie z.B. die geotechnischen Konstruktionselemente, die Gebietsberandung, die Bohrprofile sowie

die Fixpunkte werden auf eine Ebene in Form von Linien, Polygonzügen und einzelnen Punkten projiziert. Für eine Platte mit vier Pfählen ist die Projektion in Abbildung 5.1 b) dargestellt.

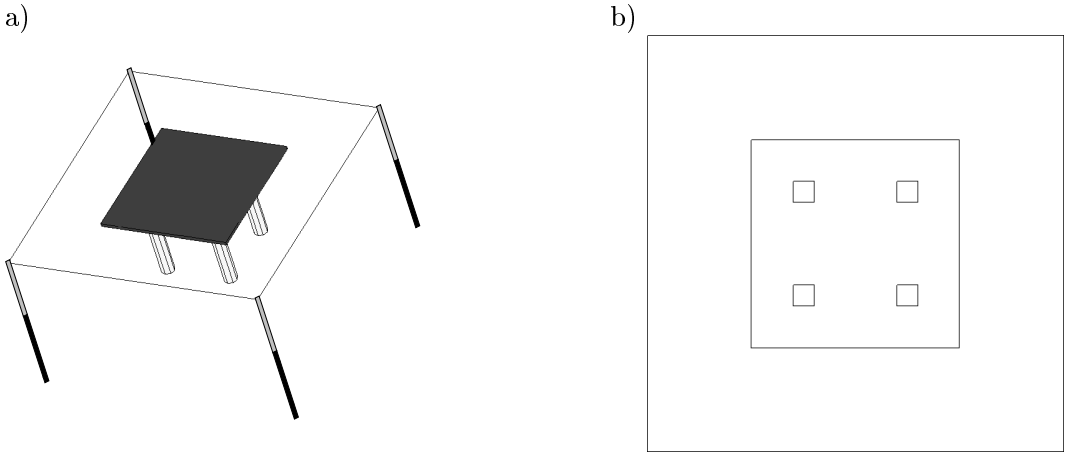


Abbildung 5.1: *Projektionstechnik a) CAD-Modell, b) Linien*

Schritt 2: Auf Basis der Linien, Polygonzüge und Punkte wird ein Dreiecknetz erstellt (Abbildung 5.2 a)). Die hierfür verwendeten Netzgeneratoren sind in [Olden 1998] bzw. in [Shewchuk 1997] beschrieben. Aus diesem Netz wird unter Verwendung der in Abschnitt 3.1.5 beschriebenen Algorithmen ein Netz aus Viereckelementen erzeugt (Abbildung 5.2 b)).

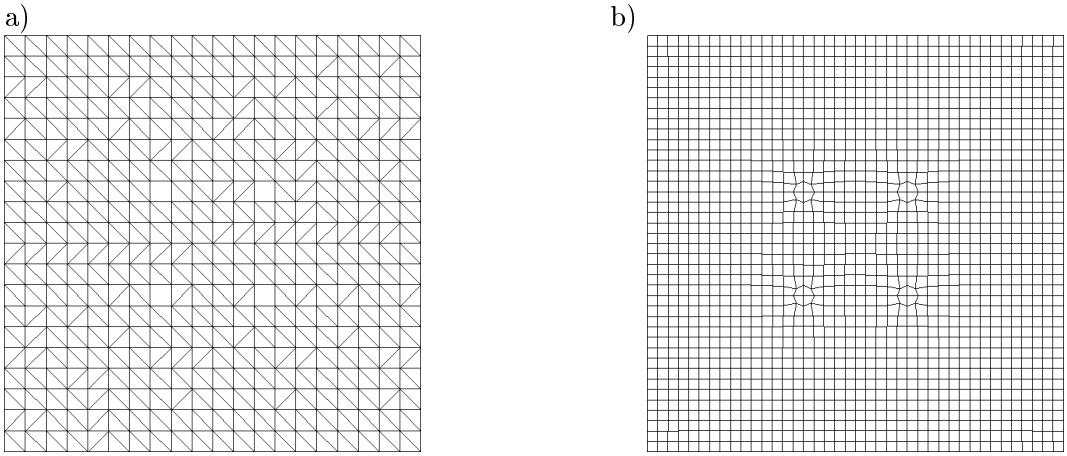


Abbildung 5.2: *Projektionstechnik a) Dreieckelementnetz b) Viereckelementnetz*

Schritt 3: Die Viereckelemente werden in die Tiefe projiziert, wodurch man Hexaederelemente erhält. Hierfür muss zuerst die notwendige Gesamtanzahl von Elementschichten ermittelt werden. Diese ergibt sich aus der Anzahl der Bodenschichten, deren Unterteilungen und den jeweiligen Konstruktionselementen innerhalb der Bodenschichten. Die untereinander liegenden Knoten werden an Stelle der Knoten des 2D-Netzes erzeugt. Die Hexaederele-

mente werden nun als nächstes unter Verwendung der untereinander liegenden Knoten und der Viereckelemente generiert (Abbildung 5.3 und 5.4) und jedem Element ein Material zugewiesen.

a) b)

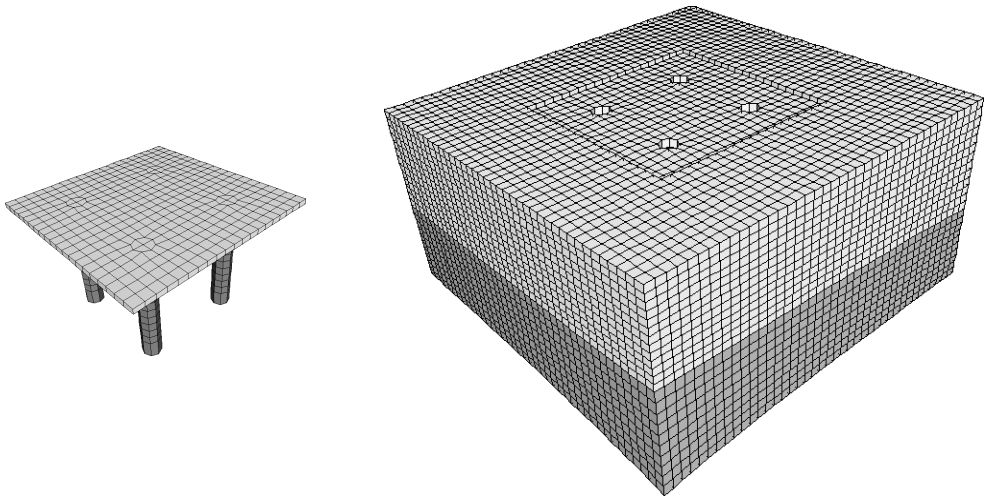


Abbildung 5.3: *Hexaeder-Elemente-Netz a) Gründung b) Baugrund*

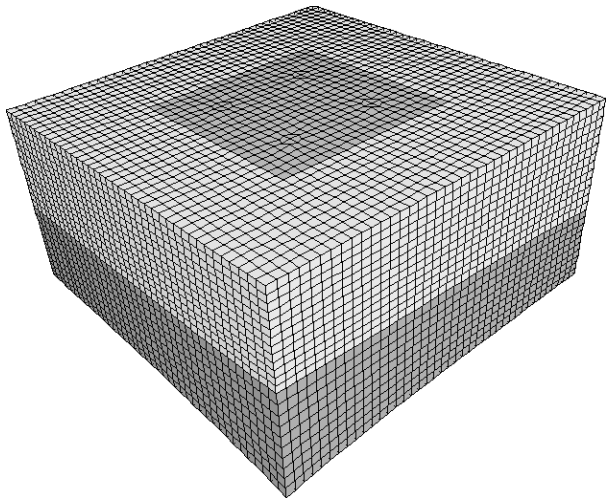


Abbildung 5.4: *Hexaeder-Elemente-Netz*

5.3 Verbessertes Erzeugen der Bauteil-Elemente

Der Einbau von geotechnischen Konstruktionselementen ist mit der Projektionstechnik nicht immer möglich. In einigen Fällen entstehen Netze mit schlechter Qualität. Ein Problem stellen Wände dar. Durch die Vorgabe eines Rechtecks als Berandung im 2D-Dreiecknetz (Abbildung 5.5 a)) entstehen im Viereckнetz durch die oben dargestellte Umwandlung zwei nebeneinander

liegende Elementreihen für jede Wand (Abbildung 5.5 b)). Hierdurch erhält man im Hexaedernetz unnötig viele Elemente.

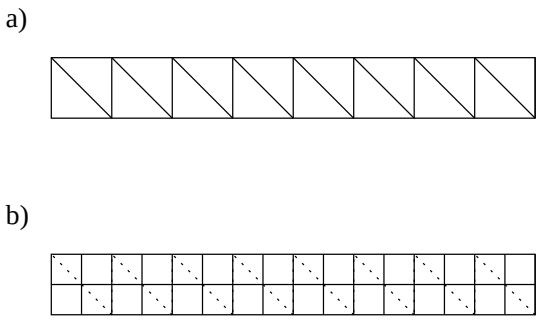


Abbildung 5.5: *a) Dreiecknetz einer Wand, b) resultierendes Viereckenetz*

5.3.1 Wände

Bei wandartigen Bauteilen, wie z.B. Spundwänden, Schlitzwänden oder einfachen Wänden, handelt es sich um Bauteile, bei der die Dicke wesentlich kleiner ist als die Kantenlängen in den anderen Richtungen. In Richtung der Dicke der Bauteile ist es daher sinnvoll, nur eine Elementlage innerhalb des Bauteils zu generieren. Bei der Erzeugung der Viereckelemente aus Dreieckelementen kann es, wie in Abbildung 5.5 dargestellt, zu mindestens zwei Elementschichten innerhalb der Wand kommen.

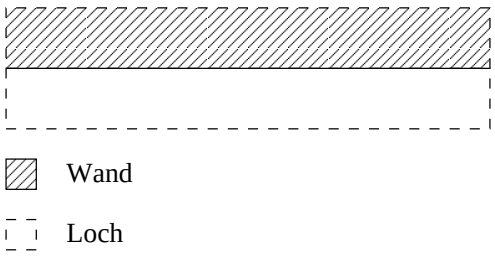


Abbildung 5.6: *Vergrößertes Wandloch in Dreiecknetz*

Die Anzahl der Elementlagen in Richtung der Dicke des Bauteils kann durch eine neu entwickelte Vorgehensweise auf ein Element reduziert werden. Hierbei wird für das Dreiecknetz jeweils ein Loch an der Stelle der Wand vorgegeben. Dieses Loch hat die doppelte Breite der Wand (Abbildung 5.6). Nach der Generierung wird das Loch im Hexaedernetz mit einem regelmäßigen Netz gefüllt. Die Kante des zusätzlichen Loches wird geglättet und verschwindet so.

Die Festlegung des zusätzlichen Loches ist recht aufwendig. Hierbei ist immer die Interaktion mit anderen Bauteilen zu berücksichtigen. Beim Aufeinandertreffen von zwei oder mehreren Wänden sind nicht nur die Wände miteinander zu verschneiden, sondern auch die zusätzlichen Wandlöcher. Diese können durch das Verschneiden größer oder kleiner werden (siehe Abbildung

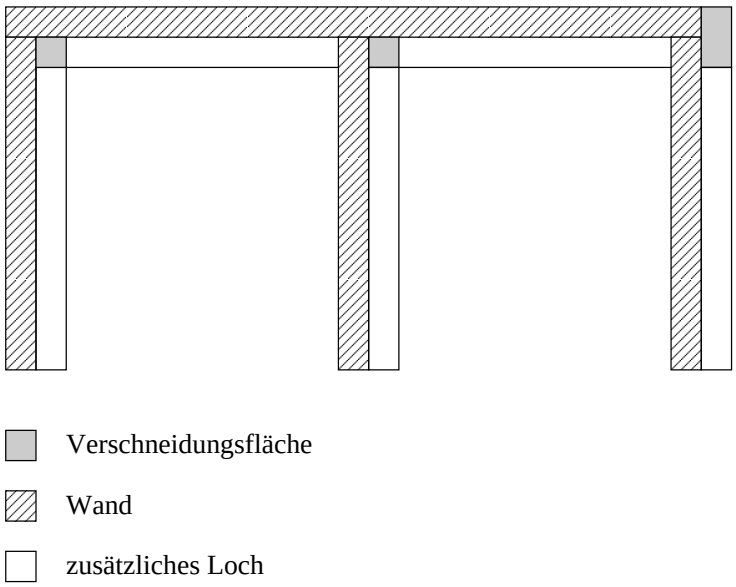


Abbildung 5.7: *Verschneiden der Wände*

5.7). Falls direkt neben der Wand auf einer Seite innerhalb des möglichen zusätzlichen Loches ein Bauteil liegt, das durch das Loch verloren gehen würde, muss das Loch auf der anderen Seite der Wand erzeugt werden.

5.3.2 Pfähle

Pfähle sind runde Bauteile. Bei der Verwendung der Mehrphasen-Theorie, bei der die Geometrie nur durch das lineare 8-Knoten-Hexaederelement angenähert wird, ist die Abbildung der Rundung nur durch die Annäherung als n-Eck möglich. Die Anzahl der Elemente innerhalb des Pfahls kann variabel gehalten werden.

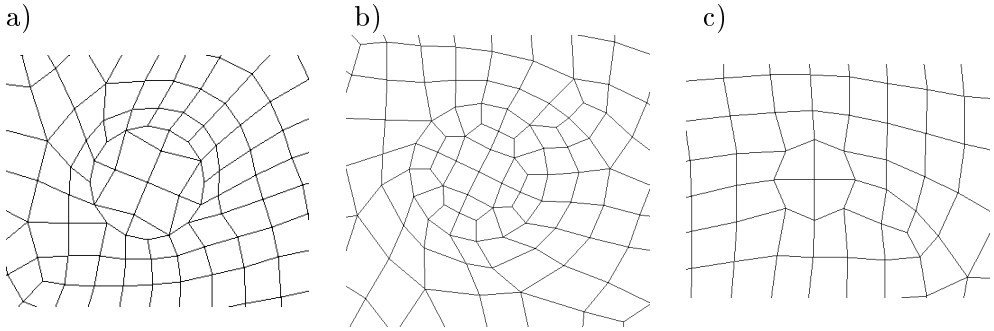


Abbildung 5.8: *a),b) dicker Pfahl, c) dünner Pfahl*

Bei der Generierung der Pfähle wird zwischen dünnen und dicken Pfählen unterschieden. Ob ein Pfahl als dünn oder dick gilt, wird durch das Verhältnis des Pfahldurchmessers zur angestrebten Elementgröße in der Umgebung bestimmt.

Bei dünnen Pfählen werden in jeder Elementschicht des Pfahls nur vier Elemente generiert. Hieraus folgt, dass der Pfahl auch außen nur maximal acht Kanten haben kann und somit als Achteck generiert wird (Abbildung 5.8 c)). Im Dreiecknetz wird ein Loch in der Form eines Quadrates erzeugt. Nach der Generierung der Pfahlelemente werden die mittleren Knoten auf den Kanten des vorherigen Loches auf die Kreisfläche gezogen, wodurch aus dem viereckigen Querschnitt ein achteckiger Querschnitt entsteht. Ein Glätten des umgebenden Netzes ist anschließend notwendig.

Dicke Pfähle werden ebenfalls durch das Herausschneiden eines Loches erzeugt. Im Gegensatz zu den dünnen Pfählen wird hier das Loch im Netz immer als Achteck oder 16-Eck erzeugt. Die Hexaederelemente innerhalb der Pfähle werden nach vorgegebenen Mustern dreidimensional erzeugt. Eine anschließende Ausrundung des Loches wird durchgeführt. Außerdem wird das Netz in der Umgebung und unterhalb des Pfahls geglättet. Durch die Vorgabe von verschiedenen Mustern ist es möglich, recht einfach die Qualität der Diskretisierung des Pfahls zu ändern. Bei einfacheren Anforderungen an das Elementnetz ist die Diskretisierung nach Abbildung 5.8 a) ausreichend, bei höheren Anforderungen ist z.B. die Verwendung des Netzes in Abbildung 5.8 b) möglich.

5.4 Schneiden von Bauteilen

Beim Teilen der Geometrie zur parallelen Netzgenerierung nach Abschnitt 4 ist bei Bauteilen wie Wänden und Pfählen auf eine sinnvolle Teilung zu achten.

5.4.1 Wände

Wände sollen nur rechtwinklig zur Wandachse geschnitten werden. Bei schrägen Schnitten, wie z.B. bei dem in Abbildung 5.9 a) dargestellten Schnitt, ist das Generieren von Hexaedern mit brauchbarer Qualität unmöglich. Bei der Implementierung des Schnittalgorithmus muss daher das schräge Schneiden der Löcher für Wände verhindert werden. Erlaubt ist nur das Schneiden der Wände senkrecht zur Wandachse, wie in Abbildung 5.9 b) dargestellt. Ein Finite-Elemente-Netz einer geschnittenen Wand ist in Abbildung 5.11 dargestellt.

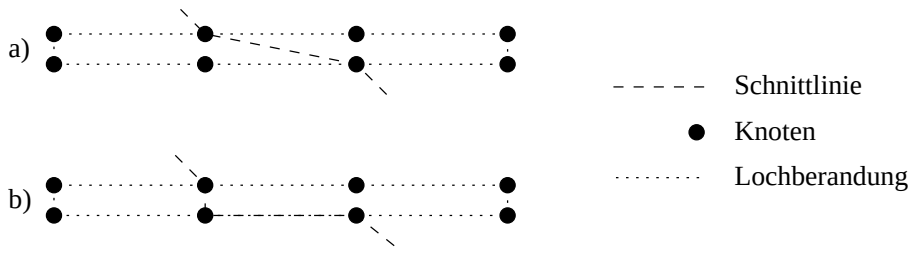


Abbildung 5.9: *Schnitt durch Wand*

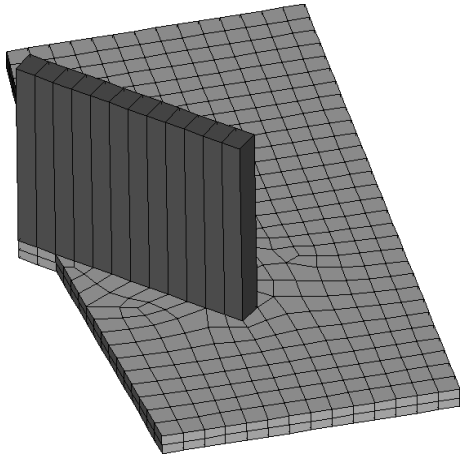


Abbildung 5.10: *Finite-Elemente-Netz mit Schnitt durch Wand*

5.4.2 Pfähle

Beim Schneiden von Pfählen ist darauf zu achten, dass der Innenwinkel der Geometrie nach dem Schnitt nicht zu spitz ist. In Abbildung 5.11 a) wir der Pfahl sehr nah am Rand geschnitten. Der zum rechten Netz gehörende Teil des Pfahls muss Elemente mit extrem spitzen Winkeln bekommen. Sinnvoller ist hier eine Schnittführung um den Pfahl herum (Abbildung 5.11 b)) oder durch den Mittelpunkt des Pfahls (Abbildung 5.11 c)).

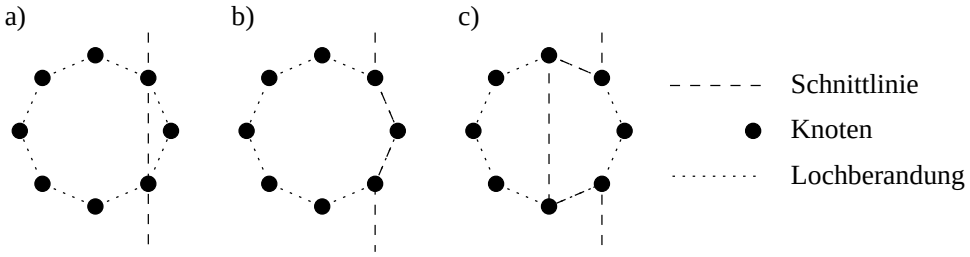


Abbildung 5.11: *Schnitt durch Pfahl*

5.5 Randbedingungen

In dreidimensionalen unstrukturierten Netzen ist wegen des Zeitaufwandes ein manuelles Erzeugen von Randbedingungen für die Finite-Elemente-Berechnung fast unmöglich [Ruben 2000]. Basierend auf der Beschreibung der Randbedingungen wird eine automatische Zuordnung und Interpolation zu den Knoten entwickelt.

5.5.1 Natürliche Randbedingungen

Die Interpolation der natürlichen Randbedingungen für die Knoten ist aufgrund der Mehrphasentheorie komplex. In der verwendeten Implementation der Mehrphasentheorie im Programm FEAP ist die Anzahl der Freiheitsgrade in den Knoten eines Elementes verschieden (Abbildung 5.12). Die Festkörperverschiebungen sind in allen Knoten der Elemente vorhanden und haben einen quadratischen Ansatz. Der Ansatz für den Porenfluiddruck ist linear und der zugehörige Freiheitsgrad ist nur in den Eckknoten der Elemente vorhanden. Die Kantenmittenknoten der Elemente haben daher jeweils einen Freiheitsgrad weniger.

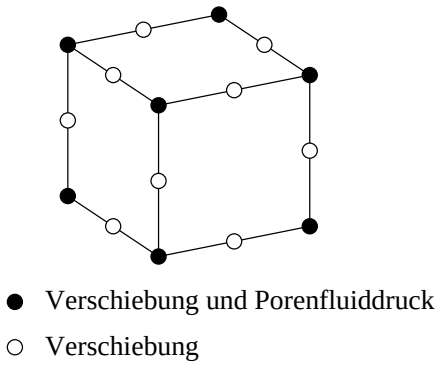


Abbildung 5.12: *Freiheitsgrade eines Hexaederelementes*

Natürliche Randbedingungen auf Linien Für die Berechnung des Zuflusses aus einer Linie ist folgendes Integral zu lösen:

$$Q = \int_{\Gamma_v} \bar{v}_{Linien} \cdot N \; d\Gamma_v$$

Hierbei sind als Formfunktionen N die linearen Formfunktionen entsprechend Abbildung 5.13 einzusetzen. $\bar{v}$ ist der Volumenstrom senkrecht zum Rand und Γ_v die Linie, längs der der Zufluss auftritt.

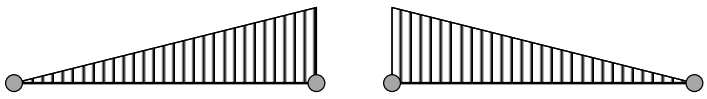


Abbildung 5.13: *Lineare Formfunktion*

Die Ersatzlasten in den Knoten, die aus Linienlasten entstehen, lassen sich durch das folgende Integral beschreiben:

$$F = \int_{\Gamma_\sigma} \tilde{p} \cdot N \; d\Gamma_\sigma$$

Hierbei sind $\tilde{p}$ die Linienlast und Γ_σ die Linien, auf der die Last einwirkt. Zur Interpolation müssen hier die quadratischen Formfunktionen entsprechend Abbildung 5.14 verwendet werden.

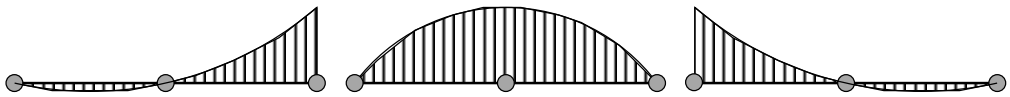


Abbildung 5.14: *Quadratische Formfunktion*

Natürliche Randbedingungen auf Flächen Der Volumenstrom, der flächig einwirkt, muss folgendermaßen auf die Knoten verteilt werden:

$$Q = \int\int_{A_v} \bar{v} \cdot N \; dA_v$$

Als Formfunktionen N sind hier die linearen Formfunktionen für Viereckelemente nach Abbildung 5.15 zu verwenden.

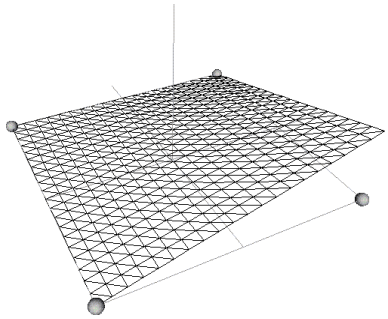


Abbildung 5.15: *Lineare Formfunktion*

Die Verteilung der Flächenlasten wird durch das Integral:

$$F = \int\int_{A_\sigma} \tilde{q} \cdot N \; dA_\sigma$$

beschrieben. Hierbei ist für die Formfunktion das Serendipity-Element anzusetzen (Abbildung 5.16).

a) b)

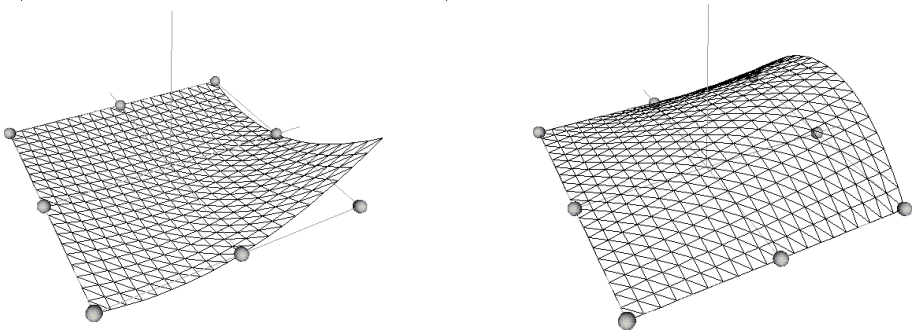


Abbildung 5.16: *Quadratische Formfunktionen*

5.5.2 Wesentliche Randbedingungen

Eine Zuordnung der wesentlichen Randbedingungen zu den Knoten wird automatisiert. Dies können zum einen verhinderte Verschiebungen bzw. der zu Null gesetzte Porenfluiddruck und zum anderen vorgegebene Verschiebungen bzw. ein vorgegebener Porenfluiddruck sein. Hierfür ist die Anwendung der in Abschnitt 4.4 dargestellten Geometrieoperationen notwendig.

5.6 Kontaktproblematik

5.6.1 Pfähle

Zur Übertragung der Lasten eines Bauwerkes in tieferliegende Schichten verwendet man Tiefgründungen, die meistens aus Pfählen bestehen. Die Kraftübertragung des Pfahls geschieht durch die Pfahlfußkraft sowie durch die Pfahlmantelkraft. Die zulässige Pfahlkraft ergibt sich daher aus [Rodatz 1992]:

$$q_{Zul} = \frac{1}{\eta}(Q_r + Q_s) \tag{5.1}$$

wobei η der Sicherheitsbeiwert, $Q_r = A_m \cdot \tau_m$ die Pfahlmantelkraft und Q_s die Pfahlfußkraft ist. An der Mantelfläche des Pfahls geschieht die Kraftübertragung durch Reibung des Bodenkörpers gegen den Pfahl. Die Größe der Mantelreibung hängt von der Oberflächenbeschaffenheit des Pfahls und der Scherfestigkeit des Baugrundes ab und wächst daher mit der senkrecht zur Pfahloberfläche wirkenden effektiven Normalspannung.

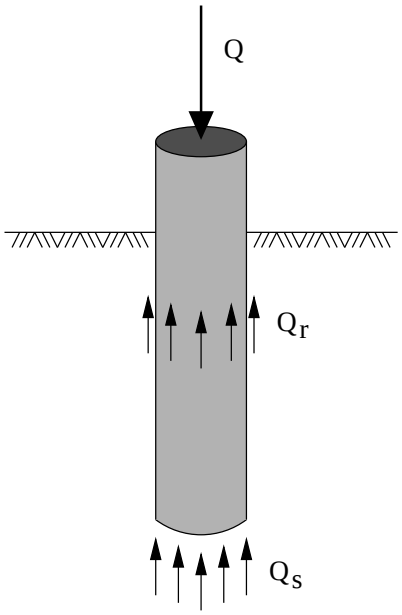


Abbildung 5.17: *Kraftübertragung am Pfahl*

5.6.2 Kontaktflächen

Im Kontaktbereich zwischen Baugrund und Bauteilen können Relativbewegungen der Oberflächen auftreten, wodurch Reibkräfte zwischen Baugrund und Bauteil aktiviert werden. Eine starre Koppelung der Finiten-Elemente ist hier nur im Ausnahmefall zulässig. In der in [Wriggers 1998] beschriebenen Implementierung des Kontakts in das Finite-Elemente-Programm FEAP [Taylor 1998] werden zur Koppelung des Finite-Elemente-Netzes im Bereich der Kontaktzone Kontaktflächen verwendet, die zur Beschreibung der sich aneinander bewegenden Körper dienen. In Abbildung 5.18 sind die Kontaktflächen für den Pfahl dargestellt. Zur Verdeutlichung wurde der Pfahl in der Abbildung im Durchmesser geschrumpft.

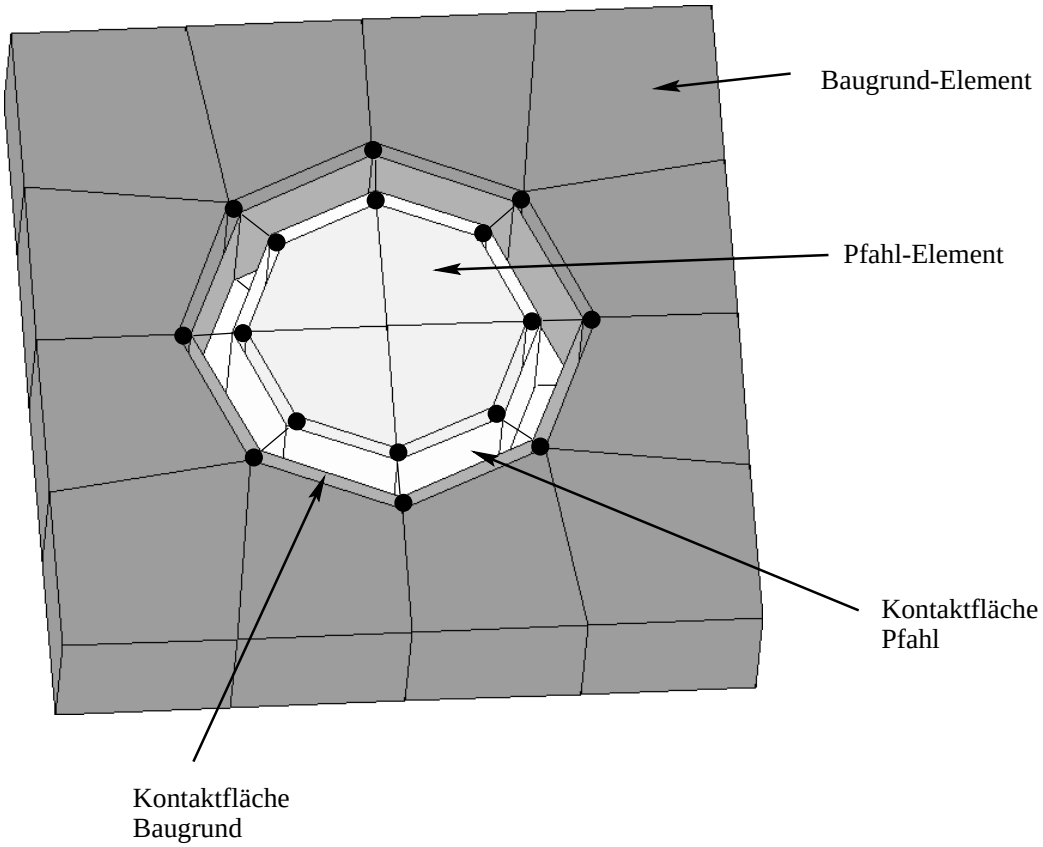


Abbildung 5.18: *Kontaktfläche am Pfahl*

Zur Generierung der Kontaktflächen wird das Netz entlang des Pfahlmantels aufgeschnitten. Für alle Knoten im Bereich des Pfahlmantels werden doppelte Knoten an der geometrisch gleichen Stelle erzeugt, sodass für den Baugrund als auch für den Pfahl jeweils ein Finite-Elemente-Knoten vorhanden ist. Sowohl für den Boden als auch für den Pfahl wird jeweils eine Kontaktfläche erzeugt, die den Pfahlmantel beschreibt und aus Viereckelementen besteht.

In Abbildung 5.19 sind zwei Finite-Elemente-Netze abgebildet, mit denen Belastungsversuche von Pfählen mit dem Programm FEAP nachgerechnet wurden. Zur Reduzierung des numeri-

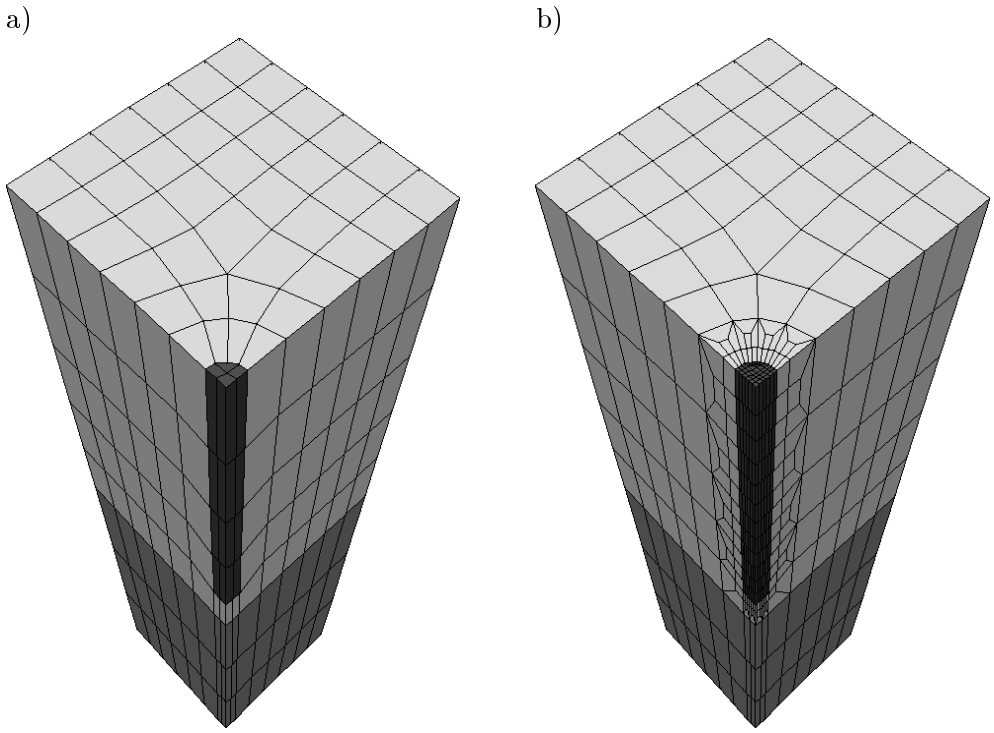


Abbildung 5.19: *Geschnittener Pfahl a) Ausgangsnetz b) mit lokaler Verfeinerung*

schen Aufwandes wurden Symmetrien ausgenutzt. Zur Verbesserung der Berechnungsergebnisse der Berechnung mit dem in Abbildung 5.19 a) dargestellten Netz wurde eine Verfeinerung im Bereich der Kontaktzone mit der in Abschnitt 3.3.4 beschriebenen dreier-Verfeinerung für Hexaederelemente durchgeführt (Abbildung 5.19 b)).

6 Objektorientierte Modellierung

Bei der beschriebenen Vorgehensweise zur Parallelisierung von Netzgeneratoren muss man zwischen drei Teilmodellen unterscheiden. Das eine Teilmodell verwaltet die Beschreibung der zu vernetzenden Problemstellung in Form eines CAD-Modells. Das Geometrie-Modell dient der Teilung des Kontinuums und der zweidimensionalen Netzgenerierung. Als Ergebnis erhält man ein Finite-Elemente-Modell, in dem auch Randbedingungen vorhanden sind.

6.1 CAD-Modell

Zur Beschreibung und Verwaltung der CAD-Daten der Gründung eines Bauwerkes und des umgebenden Baugrundes wurde das CAD-Modell entwickelt. Dieses besteht aus drei Teilen: dem Baugrundmodell, dem Bauzustandsmodell sowie dem Konstruktionselemente-Modell.

6.1.1 Baugrund-Modell

Das Baugrundmodell dient der Verwaltung des Baugrundes mit allen Parametern. In diesem Modell wird der Baugrund selbst beschrieben. Hierfür ist zum einen die Beschreibung der Geländeoberfläche und zum anderen eine Beschreibung der Schichtung und der Materialien des Bodens erforderlich. Zur Beschreibung der Oberfläche werden Digitale-Gelände-Modell-Punkte (DGM-Punkte) verwendet. Die Informationen über die Schichtung des Bodens erhält man in der Regel aus Bohrprofilen. Bohrprofile geben Auskunft über die Reihenfolge, die Dicke und die Materialien der Schichten im Baugrund an jeweils einer Stelle im Gebiet und sind daher Teil des Baugrundmodells.

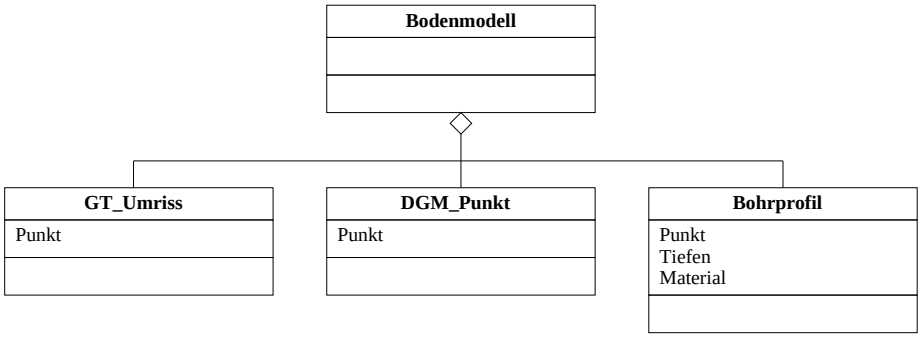


Abbildung 6.1: *Klassendiagramm Baugrund-Modell*

Die Begrenzung des Gebietes in Form eines Umrisses ist ein weiterer Teil des Baugrund-Modells. Der Umriss besteht aus einem geschlossenen Polygonzug und beschreibt die äußere

Grenze des Gebietes, die auch als Berechnungsgrenze für eine Finite-Elemente-Simulation verwendet wird.

6.1.2 Bauzustandsmodell

Die Modellierung von komplexen Baugrund-Tragwerk-Systemen ist oft nur unter Berücksichtigung des Bauablaufes möglich. Hierbei ist es erforderlich, Nachweise für verschiedene Bauzustände und den Endzustand des Bauwerkes durchzuführen.

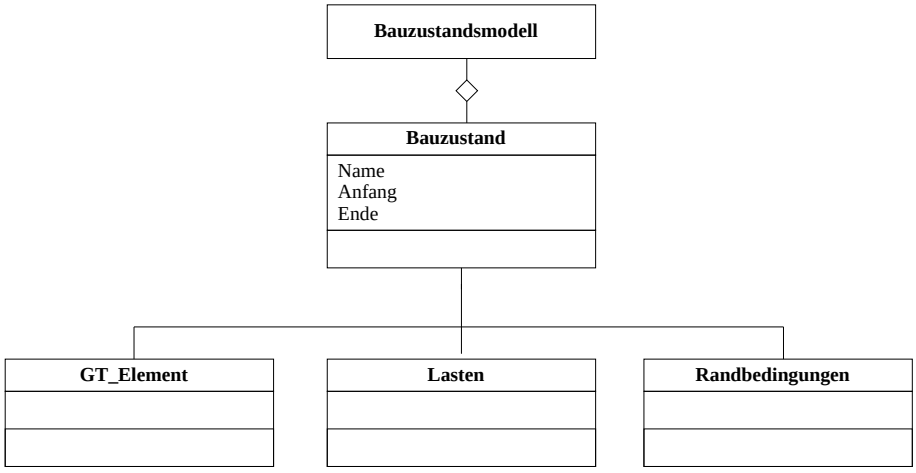


Abbildung 6.2: *Klassendiagramm Bauzustands-Modell*

Jeder Bauzustand besteht aus einer Liste von Elementen, die zu dem Bauzustand vorhanden sind. Des weiteren sind Lasten und Randbedingungen abhängig vom Bauzustand. Daher besteht jeder Bauzustand, wie in Abbildung 6.2 dargestellt, aus eine Liste von Bauzuständen, einer Liste von Lasten und einer Liste von Randbedingungen.

6.1.3 Konstruktionselemente-Modell

Geotechnische Konstruktionselemente lassen sich grob in zwei Arten unterteilen. Die einen Elemente dienen der Gründung eines Bauwerkes und werden daher als Gründungselemente bezeichnet, die anderen Elemente dienen dem Verbau einer Baugrube und werden daher Verbauelemente genannt.

In Abbildung 6.3 ist das Klassendiagramm für das Konstruktionsmodell dargestellt. Gründungselemente sind Platten, Einzelfundamente, Streifenfundamente und Pfähle. Verbauelemente sind Spundwände und Schlitzwände sowie Aushubelemente, die zur Beschreibung des entfernten Bodens verwendet werden.

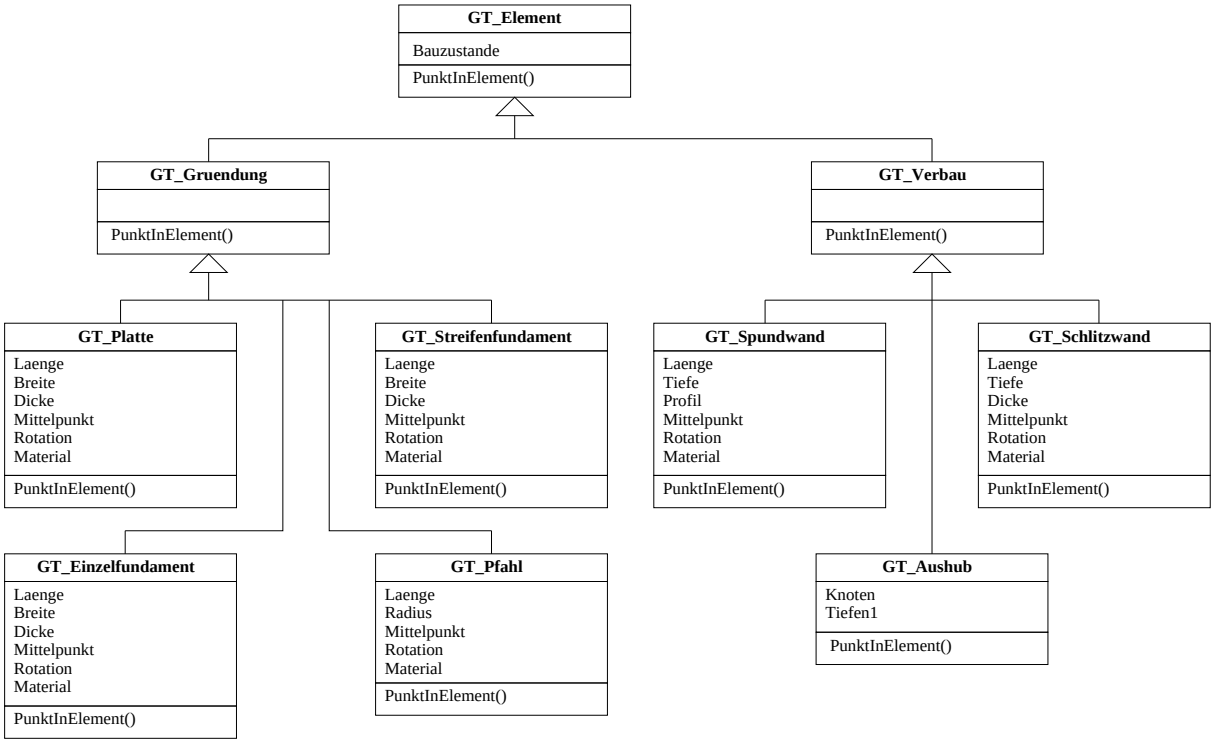


Abbildung 6.3: *Klassendiagramm Konstruktionselemente-Modell*

6.2 Geometrie-Modell

Die Beschreibung der Geometrie ebener Tragwerke ist durch die Verwendung von Polygonzügen, Linien, Löchern und Punkten möglich. Hierbei stellen z.B. Polygonzüge die Außenkanten der zu vernetzenden Platte dar, Linien die Stellen der Lagerung oder Belastung durch Wände sowie Punkte die Stellen an denen Stützen die Platte punktuell unterstützen. Löcher werden unter anderem für Schächte oder Treppen benötigt. Bei der verwendeten Projektionstechnik dienen die Polygonzüge nach der Projektion auf die zweidimensionale Ebene der Beschreibung des Gebietsrandes. Linien und Löcher werden zum Einbau von geotechnischen Konstruktionselementen benötigt. DGM-Punkte und Bohrprofile sind als Punkte auf der Ebene vorhanden.

Zweieinhalbdimensionale Strukturen können ähnlich wie ebene Tragwerke beschrieben werden. Als zusätzliche Informationen kommen Koordinaten in der 3. Richtung hinzu. Bei der Implementierung des Netzgenerators für zweieinhalbdimensionale Netze wurde das Augenmerk auf geotechnische Problemstellungen gelenkt. Die 3. Koordinate ergibt sich in diesen Fällen zum einen aus den Schichtinformationen, die sich aus Bohrpfählen gewinnen lassen, und zum anderen aus den Tiefen der Ober- und Unterkanten der verschiedenen Gründungs- und Verbauelemente.

Eine Geometrie muss daher aus einem Randpolygon, Linien, Löchern und Punkten bestehen. Bei der Teilung eines Polygonzuges können aus einem Polygonzug zwei Polygonzüge, d.h. zwei Geometrien, entstehen. Für die rekursive Teilung der Geometrie werden von allen geometrischen

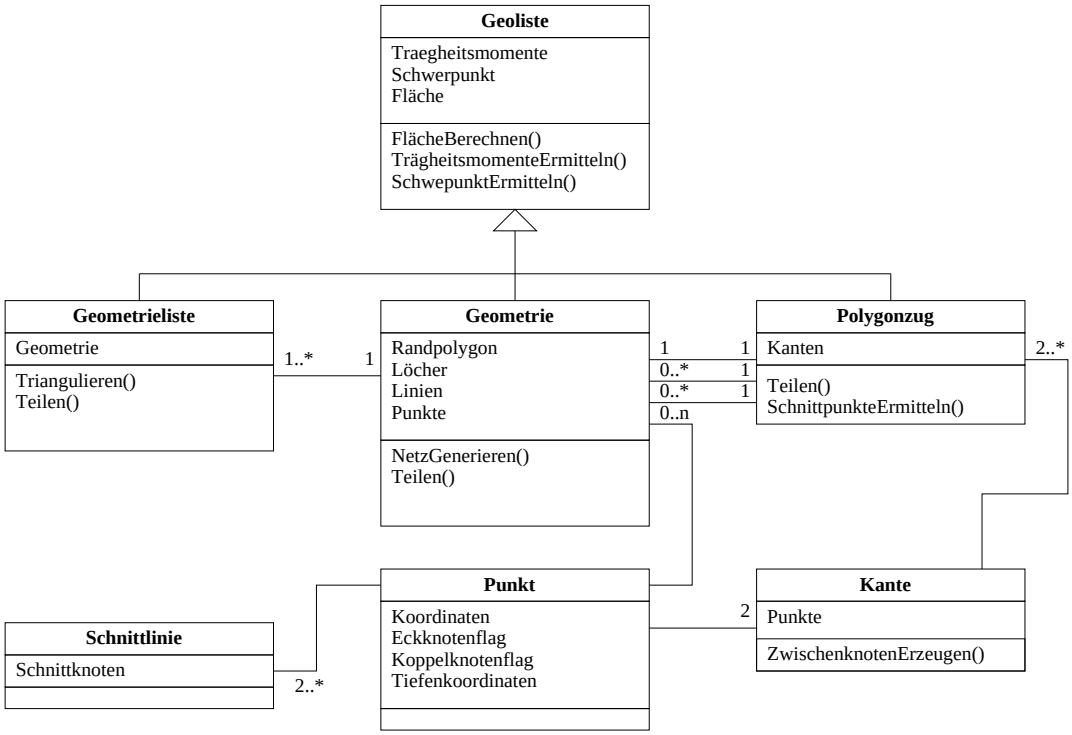


Abbildung 6.4: *Klassendiagramm Geometrie*

Elementen die Trägheitsmomente und der Schwerpunkt benötigt. Die Teilung wird durch eine Schnittlinie beschrieben, wobei die geometrischen Elemente jeweils nur an vorhandenen Punkten geteilt werden dürfen.

In Abbildung 6.4 ist das Klassendiagramm für die Verwaltung der Geometrien dargestellt. Als Basisklasse mit den grundlegenden Eigenschaften von Geometrien, wie z.B. Fläche oder Schwerpunkt, wurde die Klasse `Geoliste` entwickelt. Von ihr werden die Klassen `Geometrieliste`, `Geometrie` und `Polygonzug` abgeleitet. Mit der Klasse `Polygonzug` werden Randpolygone, Linien und Löcher von ebenen Geometrien verwaltet. Die Klasse `Geometrie` fasst die durch einen Randpolygonzug umrandeten Teile einer Geometrie zusammen. Mehrere Objekte der Klasse `Geometrie` werden mit der Klasse `Geometrieliste` verwaltet bzw. bei der Berechnung von Schwerpunkt oder Trägheitsmomenten zusammengefasst.

Eine Geometrie sowie ein Polygonzug bestehen aus Objekten der Klasse `Punkt`, die zur Beschreibung eines geometrischen Punktes dienen. Zur rekursiven Teilung der Geometrie besteht die Klasse `Schnittlinie`. Diese beschreibt den Kantenzug, entlang dem eine Geometrie geteilt wird.

6.3 Finite-Elemente-Modell

Dieses Teilmodell beschreibt das bei der Netzgenerierung entstehende Finite-Elemente-Netz und besteht aus verschiedenen Finiten-Elementen sowie aus den zugehörigen Knoten.

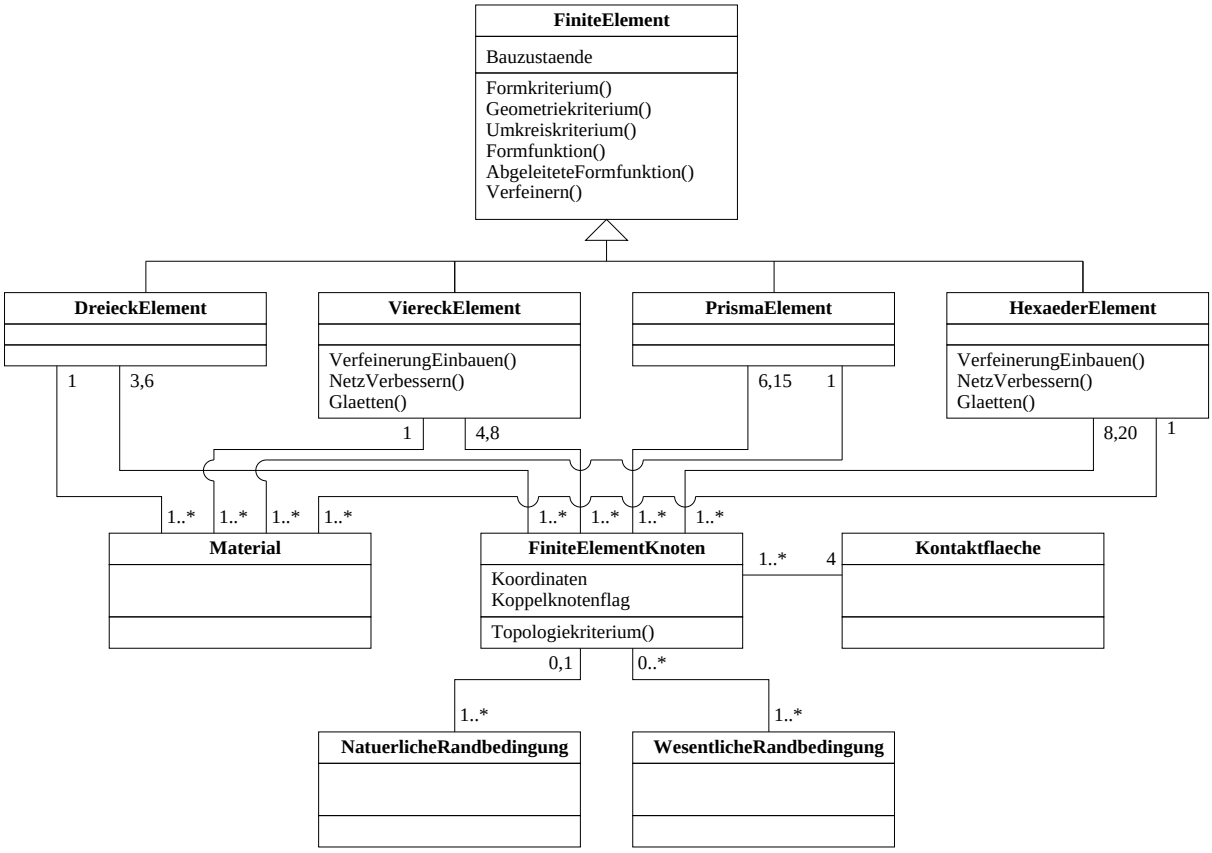


Abbildung 6.5: *Klassendiagramm Finite-Elemente-Modell*

Als Basisklasse für die verschiedenen Finite-Elemente ist die Klasse `FiniteElement` entstanden. Diese Klasse beinhaltet zum einen allgemein den Zeiger auf die Knoten, aber auch virtuelle Methoden, wie z.B. die Methoden zur Berechnung der Qualität der Elemente, die Formfunktionen oder die Netzverfeinerung. Des weiteren enthält jedes Element Informationen über die Bauzustände, zu denen das Element vorhanden ist. Die verschiedenen Finiten-Elemente leiten sich von dieser Klasse ab. Bei den ebenen Elementen sind dies die Dreieckelemente, die mit der Klasse `DreieckElement`, und die Viereckelemente, die mit der Klasse `ViereckElement` beschrieben werden. Die Klasse `ViereckElement` besitzt im Gegensatz zu `DreieckElement` noch Methoden zur Netzverbesserung nach Kapitel 3.2.2. Als dreidimensionale Elemente sind zum einen ein Prisma mit dreieckiger Grundfläche mit der Klasse `Prismaelement` und zum anderen ein Hexaeder mit der Klasse `Hexaederelement` implementiert.

Die Knoten des Finite-Elemente-Netzes werden mit der Klasse `FiniteElementKnoten` dargestellt. Diese enthält zum einen die Koordinaten des Punktes, aber auch ein Koppelknotenflag,

anhand dessen ein möglicher Partnerknoten bei der Kommunikation gefunden werden kann. Knotenorientierte Qualitätskriterien zur Beschreibung der Netzqualität, wie z.B. das Topologie-Kriterium, sind ebenfalls in der Klasse `FiniteElementKnoten` implementiert.

6.4 Geometrisch sortierte Datenstrukturen

Zur Verwaltung der Finiten-Elemente und deren Knoten gibt es verschiedene Herangehensweisen. In vielen älteren Finite-Elemente-Programmen werden hierfür meistens Felder, in neueren Programmen oft auch Listen verwendet. Für Finite-Elemente-Programme ist dies auch in der Regel ausreichend, da ein geometrisches Suchen nach Elementen oder Knoten nicht notwendig ist. Für Netzgenerierungsprogramme sind Felder und Listen allerdings unzureichend, da hier häufig geometrisch nach Knoten oder Elementen, wie z.B. bei der Einarbeitung von Randbedingungen, gesucht werden muss.

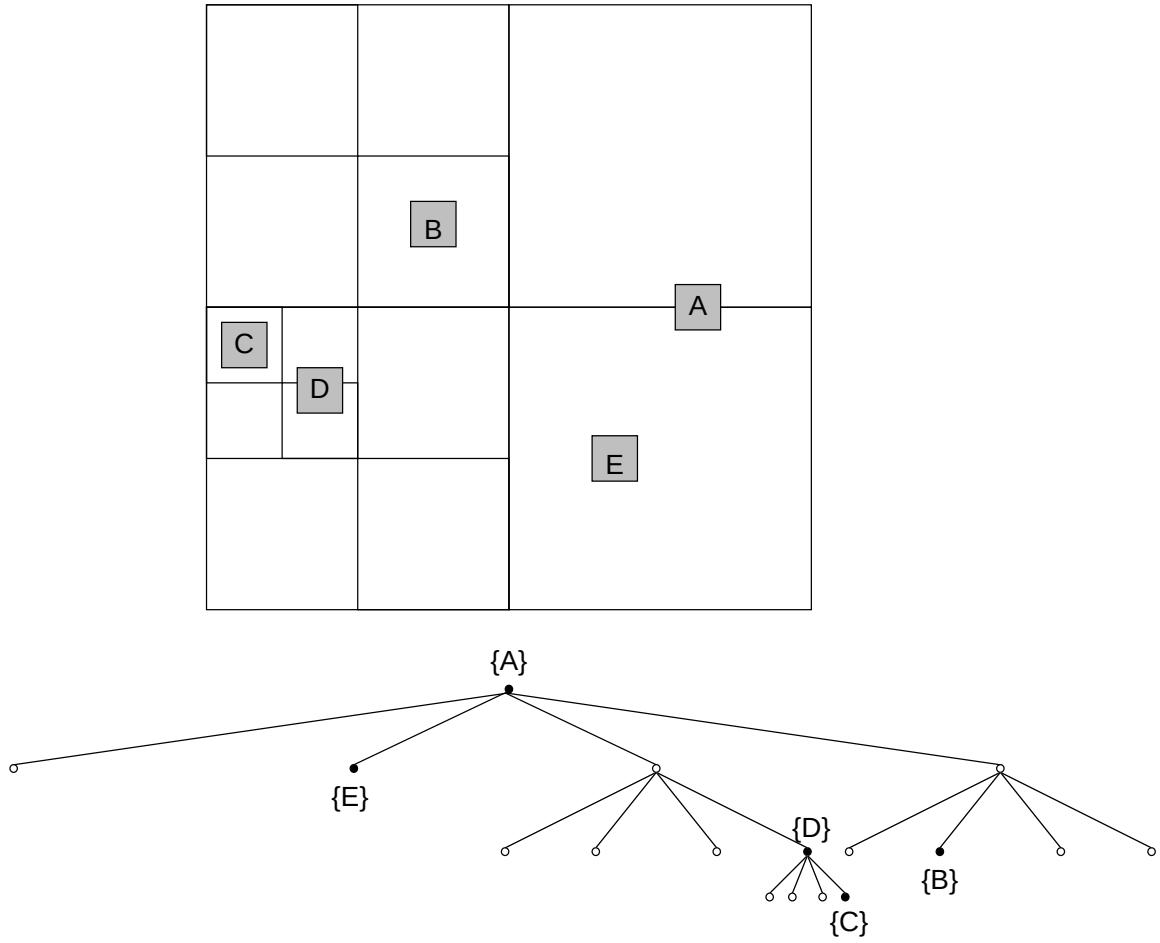


Abbildung 6.6: *Quadtree-Struktur*

Geometrisch sortierte Datenstrukturen sind im eindimensionalen Fall Binärbäume, im zweidimensionalen Fall Quadtree-Strukturen und im dreidimensionalen Fall Octree-Strukturen. Am

Beispiel der Quadtree-Struktur wird nun die Funktionsweise dieser geometrisch sortierten Datenstrukturen demonstriert. Binärbäume und Octree-Strukturen funktionieren ihrer Dimensionalität entsprechend.

In Abbildung 6.6 sind einzelne Finite-Elemente eines Netzes und der zugehörige Quadtree dargestellt. Für das Gebiet, in dem sich die Elemente befinden, wird ein umschließendes Rechteck mit achsenparallelen Kanten gesucht. Dieses Rechteck wird rekursiv in jeweils vier gleich große Rechtecke geteilt, solange Elemente in eins der jeweiligen Teilrechtecke passen. Jedes Viereck besitzt eine Liste mit Elementen sowie vier Zeiger auf Vierecke, die Teile dieses Vierecks sind. Elemente, die nicht vollständig in eines der unteren Vierecke passen, werden in der Liste abgelegt, die anderen den jeweiligen Vierecken zugeordnet und dort rekursiv weiter sortiert. So ist in Abbildung 6.6 das Elemente A nur der obersten Ebene zuzuordnen, während das Element C in der 3. Unterebene einzusortieren ist. Beim geometrischen Suchen in einem Quadtree ist jeweils nur ein Ast der Baumstruktur abzusuchen, weshalb sich gerade bei großen Datenmengen enorme Zeitvorteile ergeben.

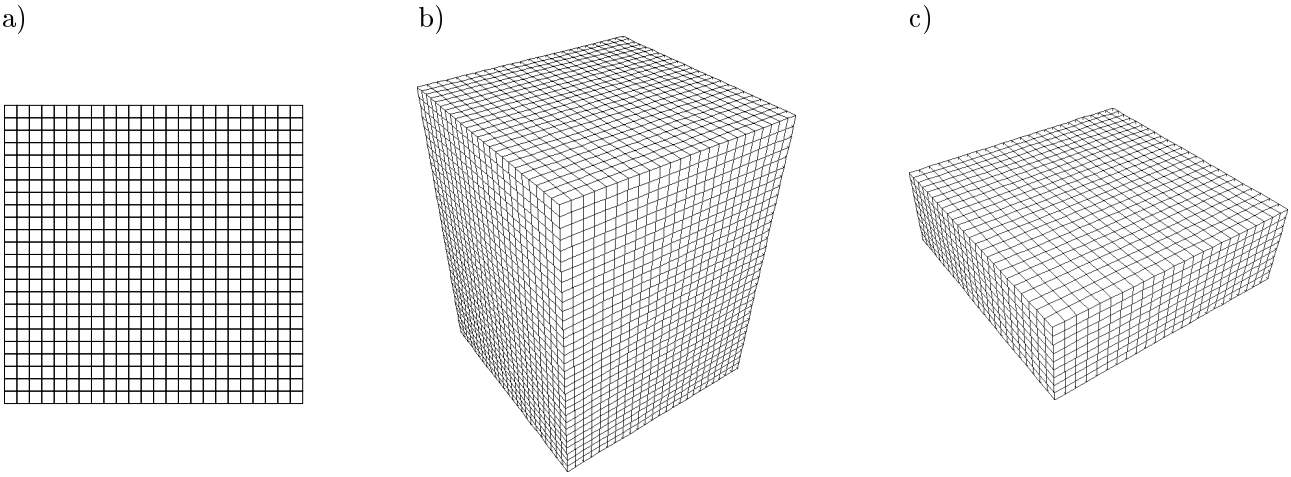
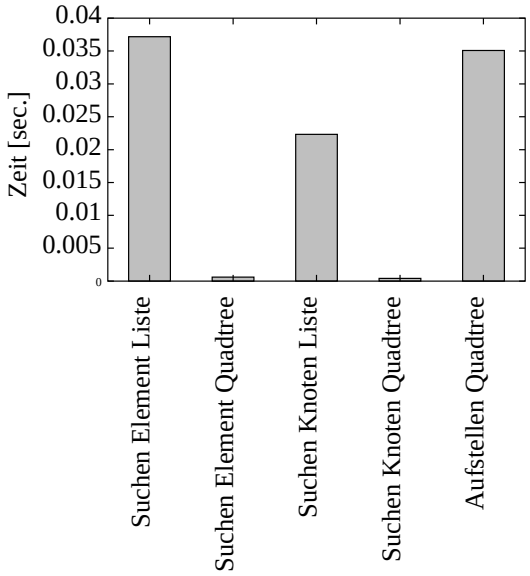


Abbildung 6.7: *Netze zum Vergleichen der Zugriffszeiten*

Die Anwendbarkeit der geometrisch sortierten Datenstrukturen wurde an verschiedenen Beispielen untersucht. Am Beispiel eines Quadrates mit 24×24 Elementen (Abbildung 6.7 a)), eines Quaders mit $24 \times 24 \times 33$ Elementen (Abbildung 6.7 b)), eines Quaders mit $24 \times 24 \times 10$ Elementen (Abbildung 6.7 c)) und des Baugrundmodells des TREPTOWERS in Berlin aus Abschnitt 7.2 mit 49442 Elementen soll die Anwendbarkeit gezeigt werden.

Als Abbruchkriterium beim Aufbauen der Quadtree-Struktur für Finite-Elemente ist festgelegt, dass sie komplett innerhalb des Vierecks liegen müssen, um hinterher auch wieder gefunden werden zu können. Knoten können in beliebig kleine Gebiete einsortiert werden um wieder gefunden zu werden. Als Abbruchkriterium ist hier daher festgelegt, dass nicht mehr als 10 Knoten in einem Viereck vorhanden sein dürfen.

Das zeitliche Verhalten für das Suchen nach einem Viereckelement für das 1. Beispiel ist in Abbildung 6.8 dargestellt. Gesucht wurde jeweils 100 mal nach einem Knoten bzw. nach



Datenstruktur	Element	Knoten
Liste	0.037173	0.022337
Quadtree	0.000612974	0.00041604

Abbildung 6.8: *Zeiten zum Suchen in der Datenstruktur für ein Quadrat mit* 24×24 *Elementen*

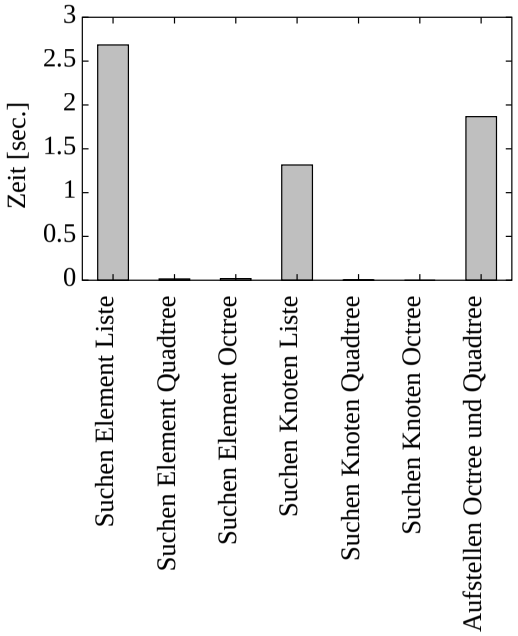
einem Element. Der Vorteil der Quadtree-Struktur ist deutlich zu erkennen. Die Suchzeiten für Elemente im Quadtree betragen nur $\frac{1}{60}$ der Suchzeit der Elemente in der Liste. Für Knoten ergibt sich das Verhältnis $\frac{1}{53}$.

In Abbildung 6.9 sind die Zeiten für 100 mal Suchen in Listen-, Quadtree- und Octree-Struktur für das 2. Beispiel dargestellt. Die Suche nach einem Element ist mit der Quadtree-Struktur am schnellsten, gefolgt von der Octree-Struktur. Die Verhältnisse zur Suchzeit in der Liste sind $\frac{1}{191}$ bzw. $\frac{1}{154}$. Für die Knoten ist die schnellste Suche in der Octree-Struktur möglich. Die Verhältnisse zur Liste sind hier: $\frac{1}{778}$ und $\frac{1}{2232}$.

Finite-Elemente-Modelle von Baugrundstrukturen haben in der Regel eine geringe Tiefe im Vergleich zu den restlichen Ausdehnungen des Modells. Dies wird mit dem Beispiel 3 nachgebildet. Das Verhältnis der Zugriffszeiten (Abbildung 6.10) für die Suche nach Elementen ist für den Quadtree $\frac{1}{204}$ und für den Octree $\frac{1}{43}$ im Vergleich zu den Zeiten in der Liste. Für die Suche nach Knoten sind die Verhältnisse $\frac{1}{532}$ und $\frac{1}{726}$. Hierbei sieht man deutlich, dass die Suche im Octree trotz seines dreidimensionalen Zugriffs nicht schneller ist als das Suchen im Quadtree.

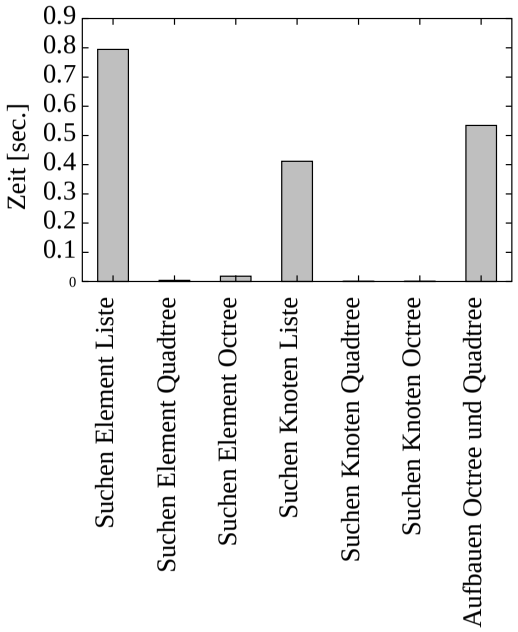
Das Beispiel Treptower zeigt ähnliche Verhältnisse für die Zugriffszeiten (Abbildung 6.11). Die Zeit zum Suchen nach einem Knoten im Quadtree beträgt $\frac{1}{28}$ der Zeit für die Liste, die Zeit für den Octree ist $\frac{1}{19}$. Die Suche nach einem Knoten ist auch für dieses Beispiel deutlich schneller mit dem Octree. Hierbei ist das Verhältnis zur Zeit der Liste $\frac{1}{7937}$. Das Verhältnis Suchzeit im Quadtree zur Suchzeit in der Liste ist $\frac{1}{4949}$.

Ein Zeitvorteil der Octree-Struktur gegenüber der Quadtree-Struktur kann nur für Knoten, aber nicht für Elemente gezeigt werden. Erklären lässt sich dies durch die Abmaße der Geometrie der Beispiele. Bei würfelförmigen Geometrien und würfelförmigen Elementen, wie in Beispiel 2, stellt der Octree eine Alternative zur Liste für die Suche nach Elementen dar. Sind die Abmes-



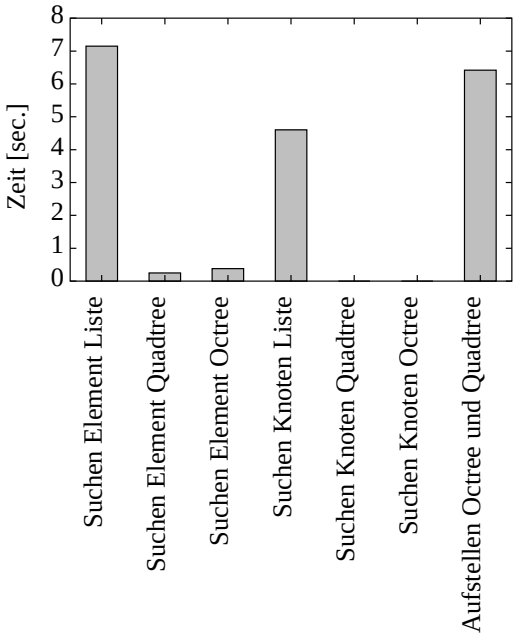
Datenstruktur	Element	Knoten
Liste	2.68537	1.31468
Quadtree	0.014055	0.00165892
Octree	0.017451	0.000589013

Abbildung 6.9: *Zeiten zum Suchen in der Datenstruktur für einen Quader mit* $24 \times 24 \times 33$ *Elementen*



Datenstruktur	Element	Knoten
Liste	0.794503	0.411391
Quadtree	0.00389004	0.000772953
Octree	0.018182	0.000566959

Abbildung 6.10: *Zeiten zum Suchen in der Datenstruktur für einen Quader mit* $24 \times 24 \times 10$ *Elementen*



Datenstruktur	Element		Knoten	
	Zeit	Ebenen	Zeit	Ebenen
Liste	7.147	1	4.60331	1
Quadtree	0.248	9	0.00093	10
Octree	0.379	7	0.00057	8

Abbildung 6.11: *Zeiten zum Suchen in der Datenstruktur für ein Baugrundmodell mit 49442 Elementen*

sungen der Geometrie nicht würfelförmig, sondern ist eine Kante des umschließenden Quaders groß oder klein gegenüber den anderen Kanten, haben die Teilquader ebenfalls unausgewogene Seitenverhältnisse. Die Octree-Struktur kann dann nur mit einer sehr geringen Anzahl von Ebenen erzeugt werden, da die Elemente in einer Richtung schon nach wenigen Rekursionen nicht mehr in die Teilquader passen. Der gleiche Effekt ist bei Elementen zu finden, die in einer Richtung lang oder kurz gegenüber den anderen Richtungen sind. Die Elemente lassen sich schlecht in untere Ebenen eine Octrees einordnen, da sie in einer Koordinatenrichtung in der Regel nicht in einen Teilquader passen. In Abbildung 6.11 sind die Anzahlen der Ebenen für die Quadtree-Struktur und die der Octree-Struktur für das Beispiel TREPTOWERS dargestellt. Die Quadtree-Struktur hat zwei Ebenen mehr, wodurch die Anzahl der Elemente, die in der untersten Ebene überprüft werden müssen, sich auf etwa $\frac{1}{16}$ der Anzahl der Elemente, die sich in der Octree-Struktur befinden, reduziert. Bei Finite-Elemente-Modellen aus dem Bereich der Geotechnik, die mit einer Projektionstechnik erstellt wurden, ist die Tiefe der Modelle meist klein gegenüber den übrigen Ausmaßen. Oft haben die Elemente auch eine säulenartige Form. Hierbei hat sich der Quadtree gegenüber dem Octree als deutlich bessere Datenstruktur zur Verwaltung der Elemente erwiesen.

6.5 Paralleler Ablauf

Der parallele Ablauf der Netzgenerierung ist schematisch in Abbildung 6.12 für die Vernetzung mit vier Prozessoren abgebildet. Zu Beginn liest ein Prozessor das CAD-Modell ein. Anschlie-

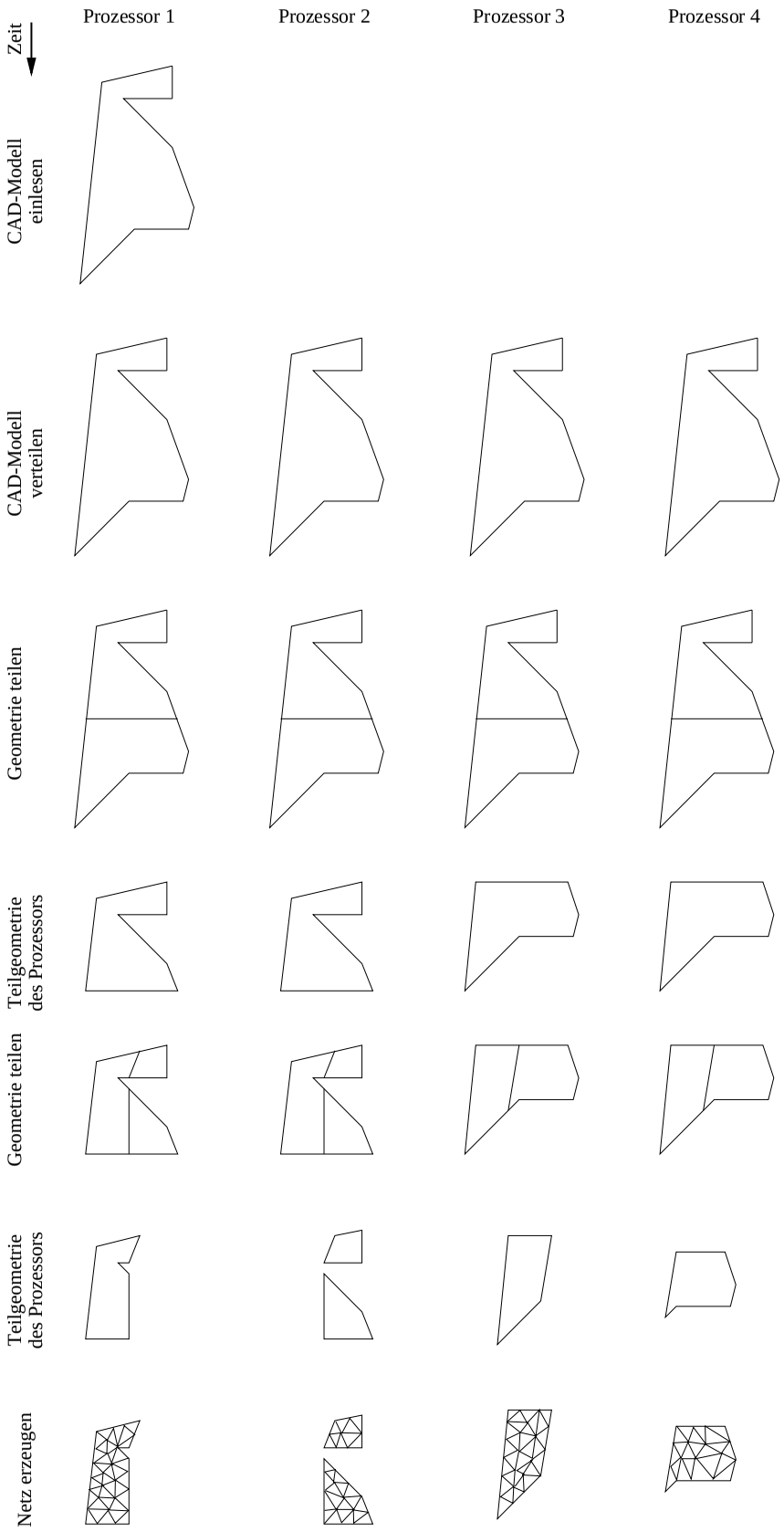


Abbildung 6.12: *Schematische Darstellung des parallelen Ablaufes mit vier Prozessoren*

ßend werden die Daten durch eine Kommunikation den anderen Prozessoren zur Verfügung gestellt. Jeder Prozessor diskretisiert als nächstes den Gebietsrand sowie Linien und Löcher entsprechend der Verteilungsfunktion der Netzdichte. Parallel wird anschließend eine Teilungslinie unter Verwendung der Schwerachsenmethode ermittelt. Die Geometrie wird geteilt, wobei der nicht benötigte Teil der Geometrie gelöscht und die Teilungslinie als Kante in die Geometrie eingebaut werden. Die Diskretisierung der Teilungslinie ist anschließend erforderlich. Rekursiv wird das Teilen der Geometrie und das Diskretisieren der Teilungslinie solange wiederholt, bis jeder Prozessor nur noch die Teilgeometrie hat, die er vernetzen muss. Anschließend wird das Netz parallel generiert.

Das Zustandsdiagramm in Abbildung 6.13 stellt die einzelnen Schritte der parallelen Netzgenerierung auf einem Prozessor genauer dar. Zuerst wird die Grundgeometrie aufgebaut. Wie schon beschrieben, ist es notwendig, eine Diskretisierung auf allen Kanten der Geometrie vorzunehmen, um die Kommunikation zwischen den Prozessoren zu vermeiden. Dies geschieht, indem auf allen Kanten Knoten im Abstand entsprechend der Wichtungsfunktion eingefügt werden. Als nächstes wird ein grobes Dreiecknetz erzeugt, bei dem sich die Eckpunkte der Dreiecke in den Eckpunkten der Geometrie befinden. Unter Verwendung dieses groben Netzes können die Flächenintegrale für den Schwerpunkt und die Trägheitsmomente berechnet werden.

In den nächsten Schritten wird zuerst der Ursprung der Geometrie in den Schwerpunkt verschoben. Unter Verwendung der Hauptachsen wird die Schnittlinie ermittelt und die Geometrie daran geteilt. Anschließend ist entlang des Schnittes eine Diskretisierung der Kanten notwendig. Falls eine weitere rekursive Teilung notwendig ist, d.h. wenn die Rekursionsanzahl kleiner ist als die Dimension des zur Verfügung stehenden Hypercubes des Parallelrechners, folgt eine erneute Vernetzung mit Dreiecken, um die Flächenintegrale für die aktuelle Geometrie zu berechnen. Ist eine weitere Teilung nicht notwendig, wird direkt ein Dreiecknetz erzeugt.

Für Viereckelemente und Hexaederelemente ist als nächstes eine Umwandlung des Dreiecknetzes in ein Viereckne tz notwendig. Dies wird auf die in Abschnitt 3.1.4 beschriebene Art durchgeführt. Eine Glättung und Verbesserung des Netzes nach Abschnitt 3.2 unter Berücksichtigung der Qualitätskriterien ist anschließend notwendig.

Als nächstes werden für Prismaelemente und Hexaederelemente die Knoten in der Tiefe unter Berücksichtigung aller geometrischen Bedingungen erzeugt. Unter Verwendung der neu entstandenen Knoten werden aus den ebenen Elementen dreidimensionale Elemente erzeugt.

Anschließend ist der Einbau einer lokalen Verfeinerung nach Abschnitt 3.3 möglich, bevor die Randbedingungen für die Finite-Elemente-Berechnung den Netzknoten zugewiesen werden. Als Abschluss der Netzgenerierung wird das Netz entweder in eine Datei geschrieben oder die Daten werden direkt einem anderen Programm zur Verfügung gestellt.

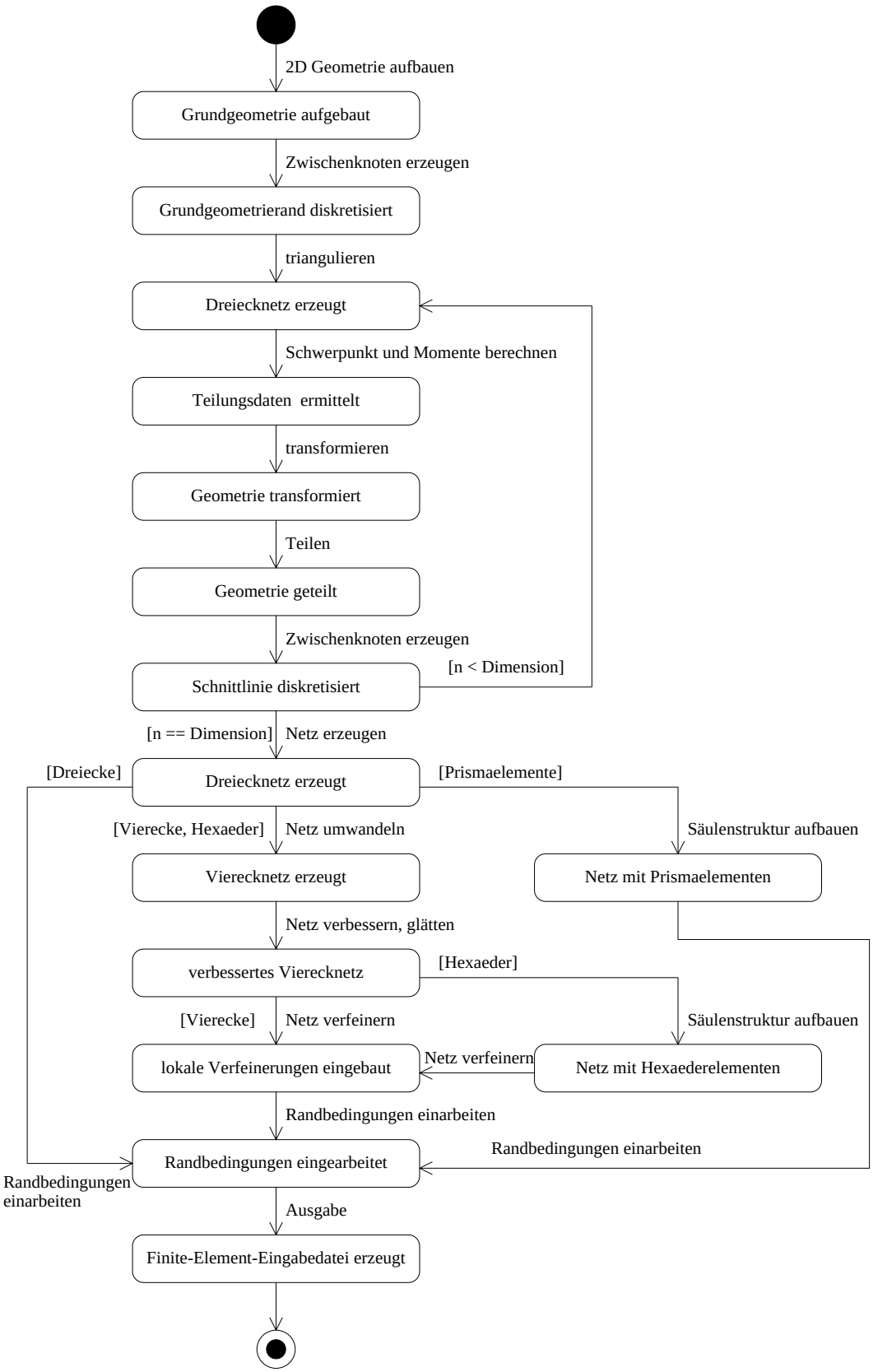


Abbildung 6.13: *Zustandsdiagramm*

6.6 Implementierung

Der Prototyp ist in der Programmiersprache C++ implementiert. Die Verwaltung der Daten geschieht unter Verwendung der Standard Template Library (STL) und den in Abschnitt 6.4 beschriebenen geometrisch sortierten Datenstrukturen. Zur Kommunikation der Prozesse wird, um eine möglichst große Portabilität zu erhalten, sowohl das Message-Passing Interface (MPI) als auch die Parallel Virtual Machine (PVM) verwendet.

Die Wahl der Programmiersprache C++ geschah aus verschiedenen Gründen. Zum einen gibt es nur stabil laufende Netzgeneratoren, zu denen man auch den kompletten Quellcode erhält, in der Programmiersprache C. Des weiteren steht MPI bis heute nur in den Programmiersprachen C und FORTRAN zur Verfügung. PVM lässt die Wahl offen, ob FORTRAN, C oder JAVA verwendet wird.

Durch die Wahl von PVM als auch von MPI wird eine möglichst große Portabilität des Programmcodes angestrebt. Die Wahl der Bibliothek geschieht durch Präprozessordirektiven.

Ausführbare Programme existieren für Sun Workstationcluster, den Parsytec PowerXPlorer als auch für Linux-Netzwerke. Zur Übersetzung des Codes können sowohl Compiler der Hardwarehersteller als auch Compiler aus dem GNU-Projekt verwendet werden. Die unterschiedlichen Hardwaretopologien sowie die leicht unterschiedlichen Implementierungen der Bibliotheken und Compiler ergeben für die gleichen Problemstellungen teilweise sehr unterschiedliche Rechenzeiten. Die resultierenden Finite-Elemente-Netze sind die gleichen.

7 Anwendungsbeispiele

Basierend auf den in den vorigen Kapiteln beschriebenen Algorithmen und dem in Kapitel 6 beschriebenen objektorientierten Entwurf wurde ein Prototyp entwickelt. Anhand dieses Prototyps soll an zwei Beispielen aus dem Bauingenieurwesen die Leistungsfähigkeit der Algorithmen und des Entwurfs gezeigt werden.

7.1 Hochbauplatte - Liederhalle Stuttgart

Zur Demonstration des Prototyps an einem zweidimensionalem Beispiel wird eine Deckenplatte der Liederhalle in Stuttgart verwendet. Das Modell ähnelt in den Proportionen der realisierten Platte, bei der Lagerung, der Belastung und den Materialien wurden jedoch Annahmen getroffen. Der Grundriss der Platte ist in Abbildung 7.1 dargestellt. Die Platte hat mehrere Löcher sowie eine große Aussparung im Bereich des Zuschauerraums der Halle. Gelagert ist die Platte zum einen auf Wänden und zum anderen auf punktförmigen Stützen. Als Belastung wurde eine Gleichflächenlast auf der gesamten Platte angenommen.

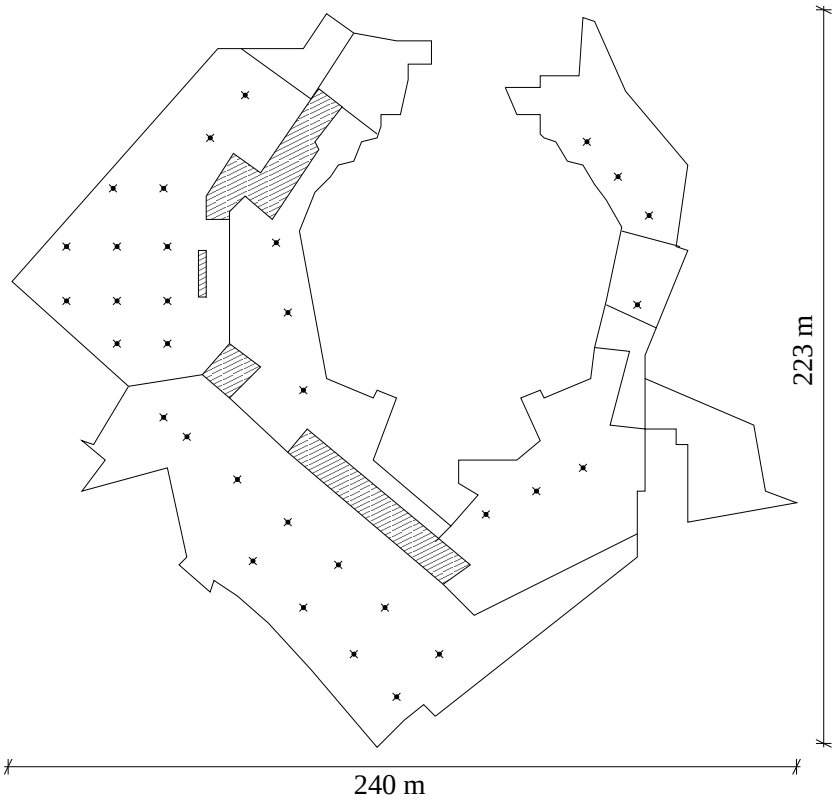


Abbildung 7.1: *Liederhalle: Grundriss der Platte*

Die Vernetzung der Platte mit Viereckelementen wurde mit verschiedenen Netzdichten durchgeführt. Im Rahmen dieser Arbeit werden hiervon zwei Diskretisierungen vorgestellt. Bei der ersten Diskretisierung wird als Netzdichte 2 Meter für die angestrebte Kantenlänge vorgegeben. Das resultierende Netz hat ca. 3.500 Elemente. Für die zweite Diskretisierung wird eine Kantenlänge von 1 Meter angestrebt, wobei das resultierende Netz ca. 16.000 Elemente aufweist.

7.1.1 Diskretisierung 1

Die Diskretisierung 1 hat die maximale Kantenlänge, um die Geometrie der Platte mit konstanter Elementgröße abbilden zu können. Das resultierende Netz ist in Abbildung 7.2 dargestellt. Die Partitionierungen für Berechnungen mit vier und acht Prozessoren sind in den Abbildungen 7.3 und 7.4 zu sehen. Deutlich erkennbar ist hierbei, dass die Partitionierung für acht Prozessoren durch die Teilung der Gebiete der Partitionierung für vier Prozessoren entsteht. Die Elementanzahlen der partitionierten Netze sind Tabelle 7.1 zu entnehmen.

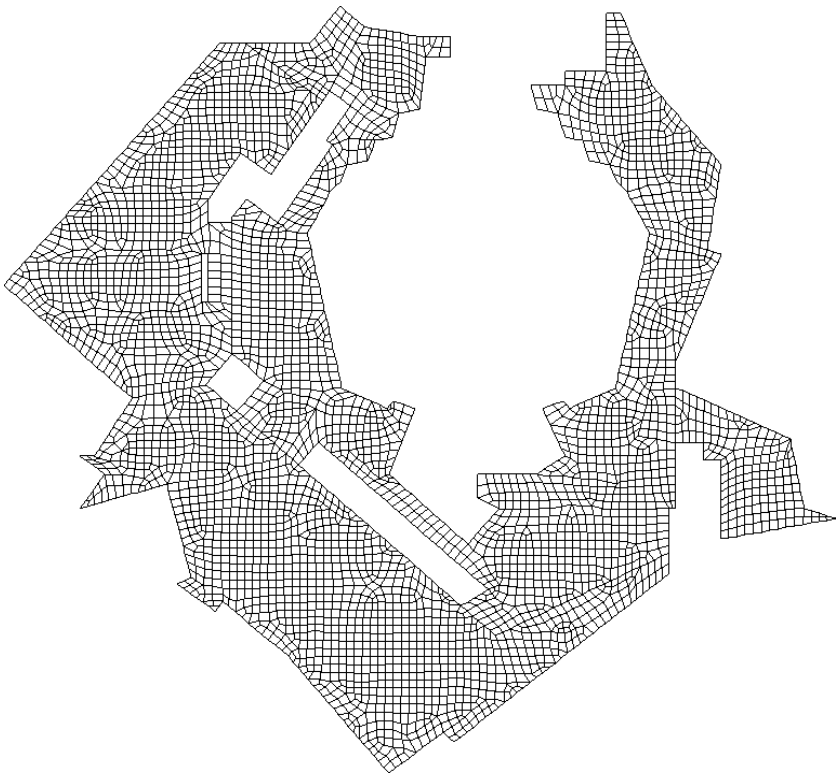


Abbildung 7.2: *Liederhalle: Elementnetz Diskretisierung 1*

Durch die außermittige große Öffnung in der Platte kommt es bei der ersten Teilung zu einer ungleich großen Teilung der Platte. Daher sind die Elementanzahlen bei der Berechnung mit zwei Prozessoren unterschiedlich und der eine Prozessor hat hierbei sogar zwei Teilgebiete. Bedingt wird dies durch die Schwerachsenmethode, bei der die Geometrie mit einer Linie durch den Schwerpunkt geteilt wird. Die große Öffnung bewirkt, dass Teile der Geometrie mit kleiner Fläche

Anzahl der Prozessoren	Elementanzahl Durchschnitt	Elementanzahl Minimum	Elementanzahl Maximum
1	3484	3484	3484
2	1747	1537	1956
4	852	728	997
8	434	449	518
16	224	162	272

Tabelle 7.1: *Liederhalle: Elementanzahlen der Netze der Diskretisierung 1*

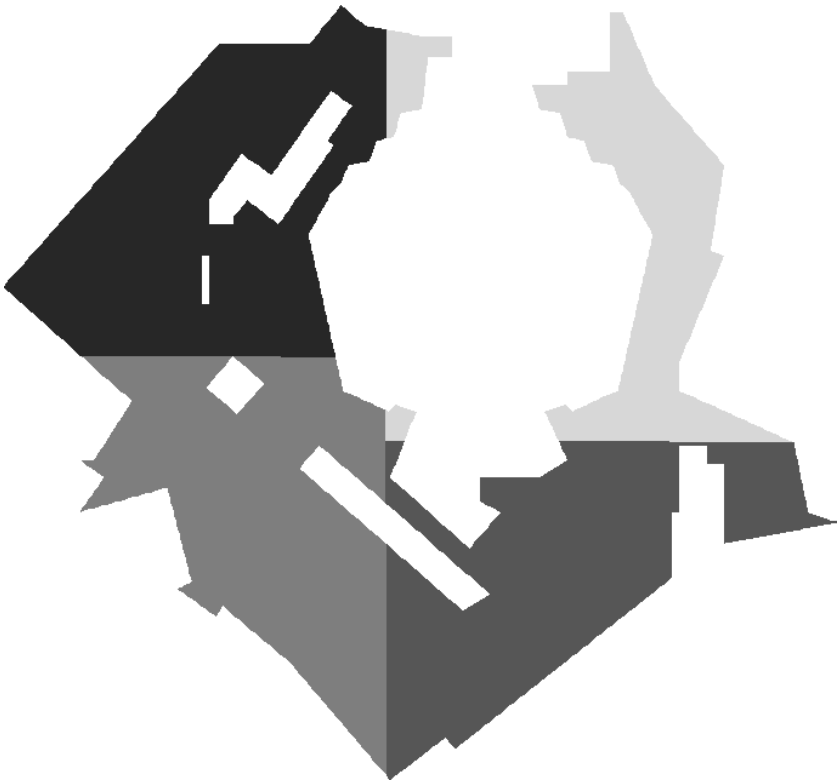


Abbildung 7.3: *Liederhalle: Partitionierung mit vier Prozessoren*

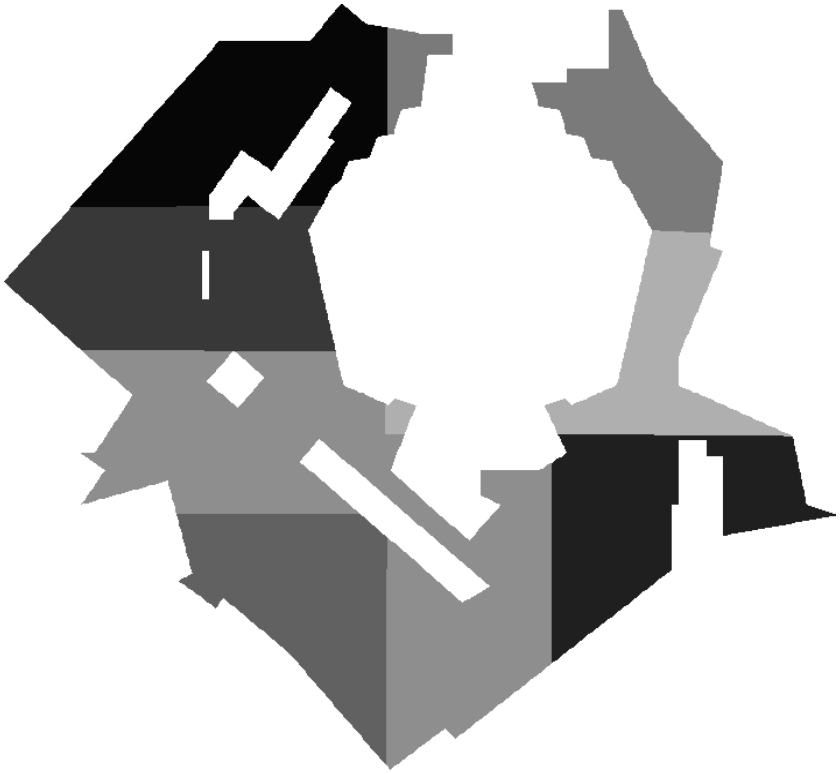


Abbildung 7.4: *Liederhalle: Partitionierung mit acht Prozessoren*

und großem Hebelarm den gleichen Einfluss haben wie Teile der Geometrie mit großer Fläche und kleinem Hebelarm. Daher ist die erste Teilung der Geometrie problematisch und bewirkt, dass die eine Teilgeometrie 44 % der Fläche und die andere 56 % der Fläche bekommt. Die weiteren Teilungen sind ausgewogener. Durch die rekursiven Teilungen hat nach vier Teilungen der Prozessor mit der größten Elementanzahl daher fast 70 % mehr Elemente als der Prozessor mit der geringsten Elementanzahl.

7.1.2 Diskretisierung 2

Mit der Diskretisierung 2 wird eine wesentlich feinere Auflösung des Netzes angestrebt. Die Kantenlängen betragen etwa 1 Meter, woraus sich eine Gesamtanzahl von etwa 16500 Elementen ergibt. Die Diskretisierung für einen Prozessor ist in Abbildung 7.5 dargestellt. Die Partitionierungen für mehrere Prozessoren entsprechen mit minimalen Abweichungen aufgrund einer anderen Randdiskretisierung den Partitionierungen der Diskretisierung 1.

Bei der rekursiven Teilung der Platte ergibt sich wieder das Problem der großen Öffnung. Die feinere Diskretisierung am Rand aufgrund der feineren Netzdichte bewirkt eine etwas günstigere Teilung der Geometrie bei der ersten Teilung als bei der Diskretisierung 1. Das Verhältnis der Elementanzahlen für zwei Prozessoren ist hier 54% zu 46 %. Die weiteren Teilungen sind wiederum günstiger als die erste. Nach vier Teilungen hat das kleinste Teilgebiet 67% der Ele-

Anzahl der Prozessoren	Elementanzahl Durchschnitt	Elementanzahl Minimum	Elementanzahl Maximum
1	16350	16350	16350
2	8274	7612	8935
4	4054	3577	4646
8	2106	1752	2369
16	1046	858	1280

Tabelle 7.2: *Liederhalle: Elementanzahlen der Netze der Diskretisierung 2*

mente des größten Teilgebietes. Die feinere Diskretisierung und damit die höhere Auflösung des Geometrierandes bewirken ein exakteres Treffen der Schwerachsen. Die Teilung der Geometrie ist dadurch besser und die Elementanzahlen in den Teilgebieten ausgeglichener.

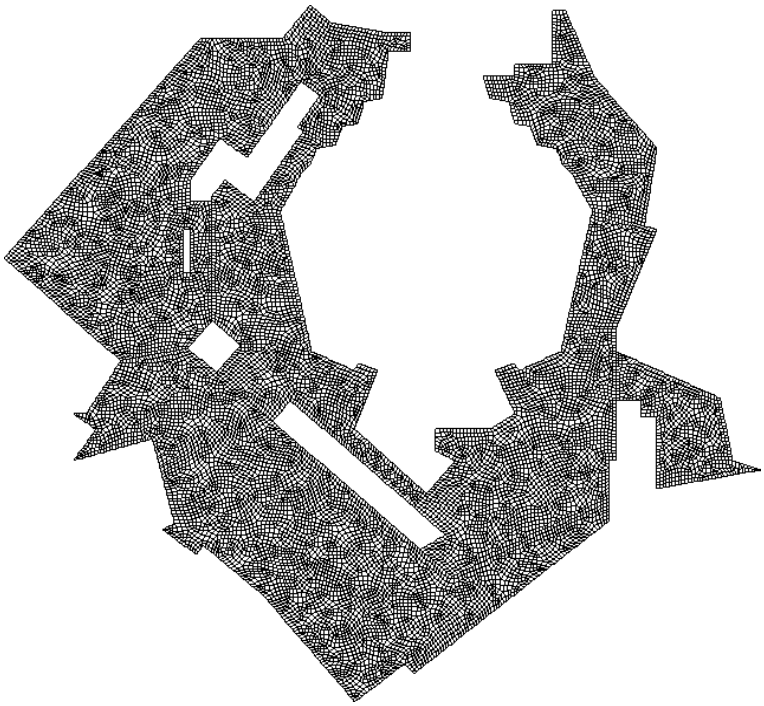


Abbildung 7.5: *Liederhalle: Elementnetz Diskretisierung 2*

7.1.3 Effektivität der Parallelisierung

Die Effektivität der Parallelisierung des Netzgenerators ist im Speed-up-Diagramm in Abbildung 7.6 und im Effizienz-Diagramm in Abbildung 7.7 dargestellt. Die ungleichmäßige Teilung im ersten Rekursionsschritt, bedingt durch die große Öffnung für den Zuschauerraum in der Platte, wirkt sich auf die Effizienz stark aus. Die weiteren ungleichmäßigen Teilungen führen zu einer Effizienz von 59,6 % für die Diskretisierung 1 und von 53,1 % für die Diskretisierung 2 bei der Verwendung von 16 Prozessoren.

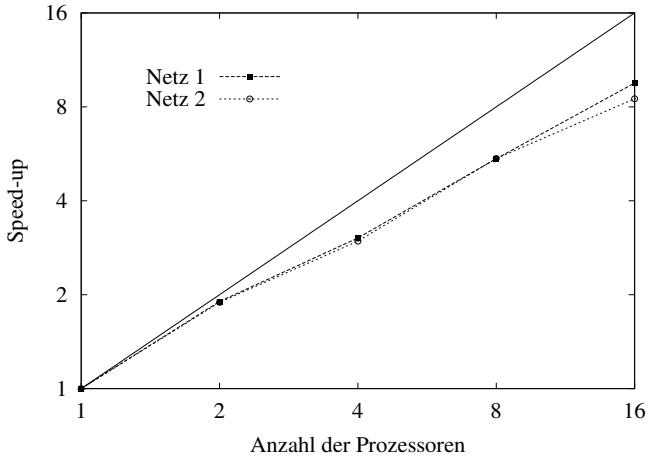


Abbildung 7.6: *Liederhalle: Speed-up*

Die Effektivität der feineren Diskretisierung ist immer schlechter als die der groberen. Dies liegt daran, dass für die feinere Diskretisierung der sequentielle Aufwand geringer ist, da die Randdiskretisierung weniger Knoten hat. Hierdurch ist der Aufwand für die Triangulierung des Gebietes zum Lösen der Integrale sowie das Lösen der Integrale selber auch wesentlich geringer.

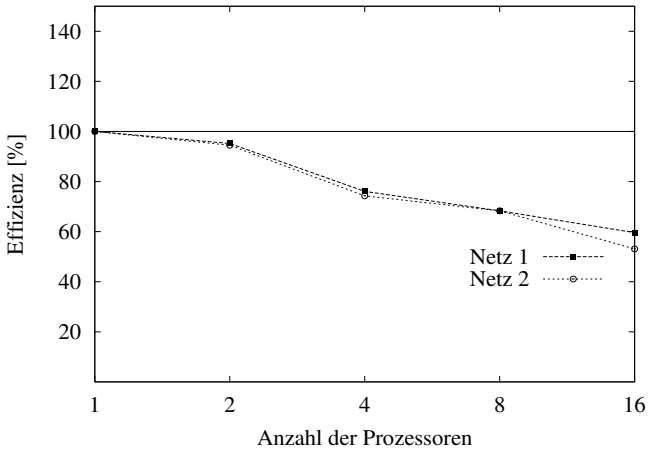


Abbildung 7.7: *Liederhalle: Effizienz*

7.2 Gründung - TREPTOWERS Berlin

Als reales Demonstrationsbeispiel für eine zweieinhalb-dimensionale Baugrundstruktur wird die Gründung des Hochhauses TREPTOWERS in Berlin-Treptow ausgesucht. Das Hochhaus steht in der Nähe der Spree auf dem ehemaligen Gelände des VEB Kombinats Elektro-Anlagen-Werke und ist mit 31 Obergeschossen insgesamt 121.5 m hoch. Gegründet ist das Gebäude mit einer Kombinierten-Pfahl-Platten-Gründung mit einer 37,1 m × 37,1 m großen Fundamentplatte (Abbildung 7.8). Insgesamt 54 mantelverpresste Bohrpfähle mit einer Länge zwischen 12,5 m und 16 m befinden sind unterhalb der Platte.

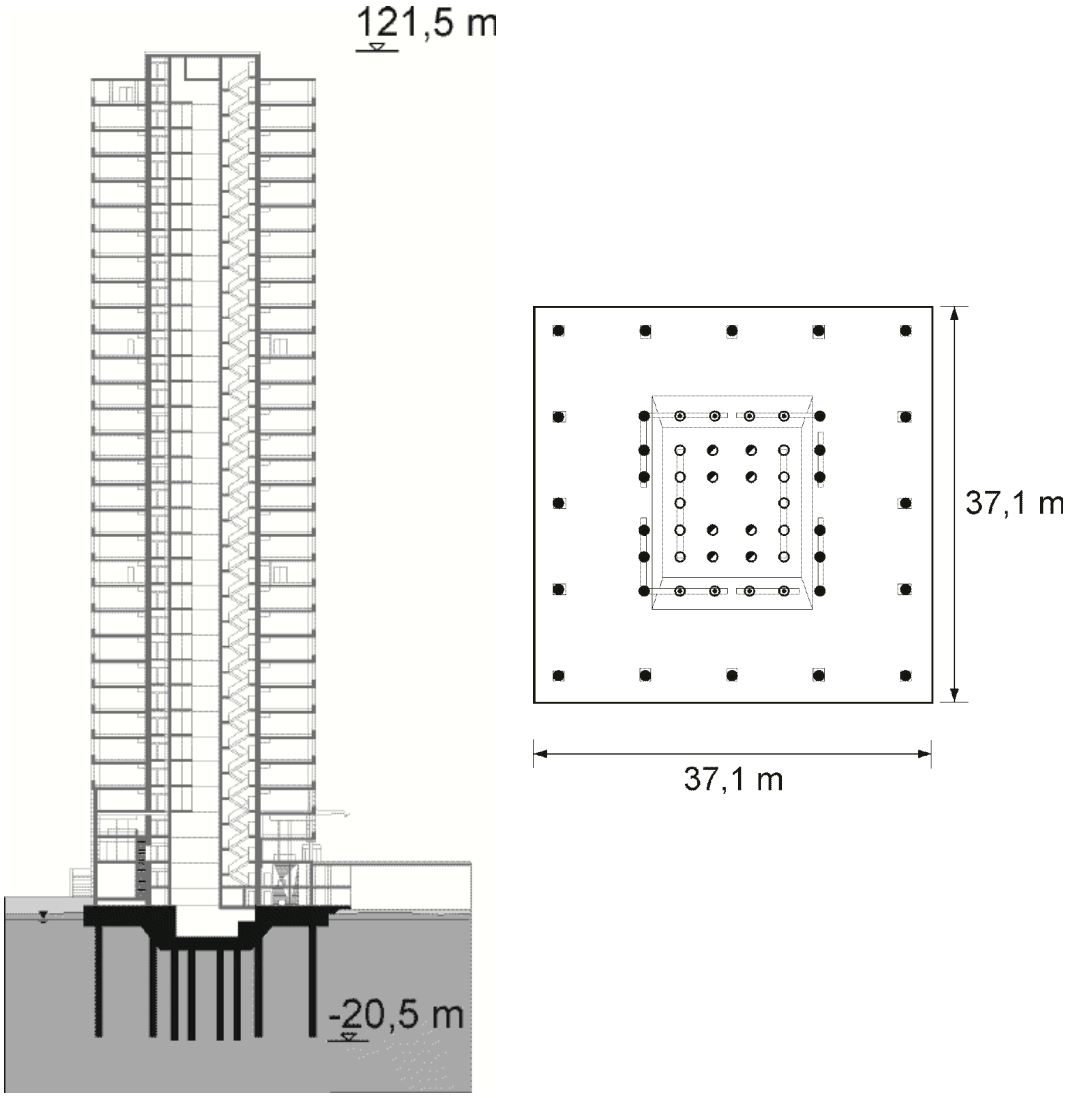


Abbildung 7.8: *TREPTOWERS: Schnitt und Grundriss*

Der Baugrund besteht aus einer ca. drei Meter dicken Schicht aus nicht tragfähigen organischen Böden, unter der eine etwa 32 m dicke Schicht aus wechselgelagerten pleistozänen Sanden

und Kiesen ansteht. Unterhalb dieser Schicht befindet sich eine pleistozäne Geschiebemergelschicht.

7.2.1 Bauablauf

Die Gründungssohle befindet sich fünf Meter unterhalb des Grundwasserspiegels. Zur horizontalen Abdichtung der Baugrube wurde eine umschließende Spundwand eingebracht. Zur Verhinderung des Ausspülens des Bodens in der Baugrundsohle wurde eine Weichgelsohle verwendet, die erst nach dem Bohren der Pfähle erstellt wurde. Anschließend wurde der Aushub des Bodens durchgeführt und die Bodenplatte erstellt.

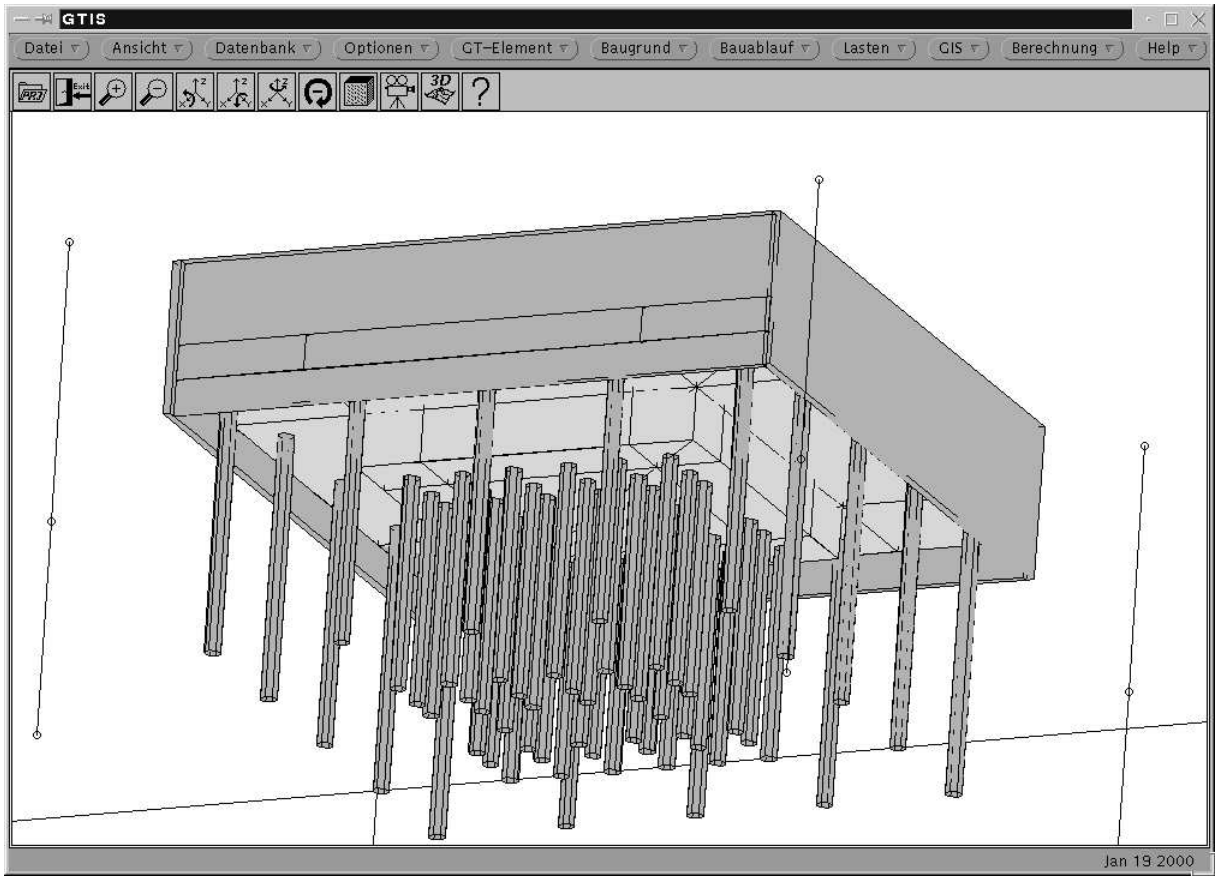


Abbildung 7.9: *TREPTOWERS: Modellierung mit GTIS*

Die Modellierung des Systems wurde mit dem Modellierungswerkzeug GTIS [Diaz 1998], [Schönenborn 1999] entsprechend dem Bauablauf durchgeführt (Abbildung 7.9). Als Modellierungsgebiet wird eine rechteckige Grundfläche angenommen, deren Begrenzung 50 m entfernt parallel zum Baugrubenrand liegt. Unterhalb der Pfähle wurde der Boden bis in eine Tiefe der zweifachen maximalen Pfahllänge modelliert.

Die Last auf die Gründung durch das Eigengewicht des Gebäudes steigert sich während des Bauablaufes in mehreren Schritten auf insgesamt 560 MN und wird als Gleichflächenlast auf

der gesamten Plattenoberfläche angenommen. Als Randbedingung wird angenommen, dass die Verschiebung am Rand des Modellierungsgebietes senkrecht zum Modellierungsrand nicht mehr vorhanden ist. Des weiteren werden die Spundwand als auch die Baugrubensohle auf Grund der Weichgelsohle als undurchlässig angenommen. Die Netzdichte ist am Rand des Gebietes relativ grob mit einer Kantenlänge von 5,0 m gewählt. In der Nähe der Baugrube wird eine feinere Netzdichte mit einer Kantenlänge von 2,5 m vorgegeben. Um eine brauchbare Qualität des Netzes im Bereich der Baugrube mit den viele Pfählen zu ermöglichen, wird hier eine Kantenlänge von 1,0 m gewählt.

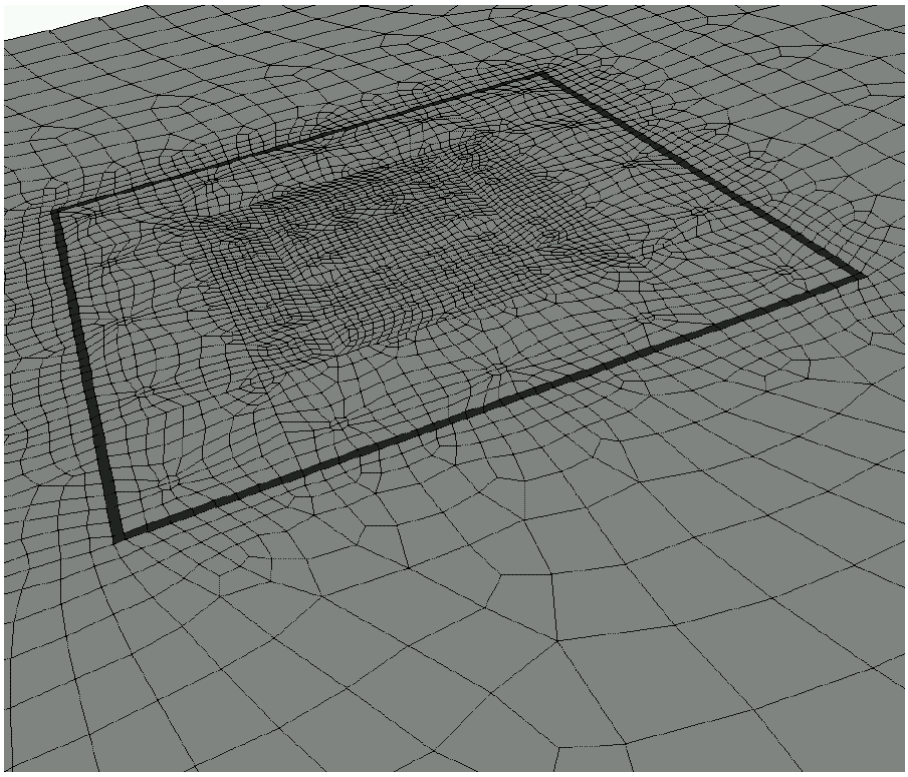


Abbildung 7.10: *TREPTOWERS: Bauzustand nach dem Einbringen der Spundwände*

Der Bauzustand des Bodens nach dem Einbringen der Spundwände ist in Abbildung 7.10 dargestellt. Erkennbar sind im Finite-Elemente-Netz die Linien, die für die später einzubringenden Bauteile erforderlich sind.

Abbildung 7.11 zeigt den Bauzustand der Baugrube nach dem Aushub auf die endgültige Baugrubentiefe. Im Boden befinden sich zu diesem Zeitpunkt auch schon die mantelverpressten Bohrpfähle. Das Finite-Elemente-Netz der fertiggestellten Gründung mit dem Baugrubenverbau ist in Abbildung 7.12 dargestellt.

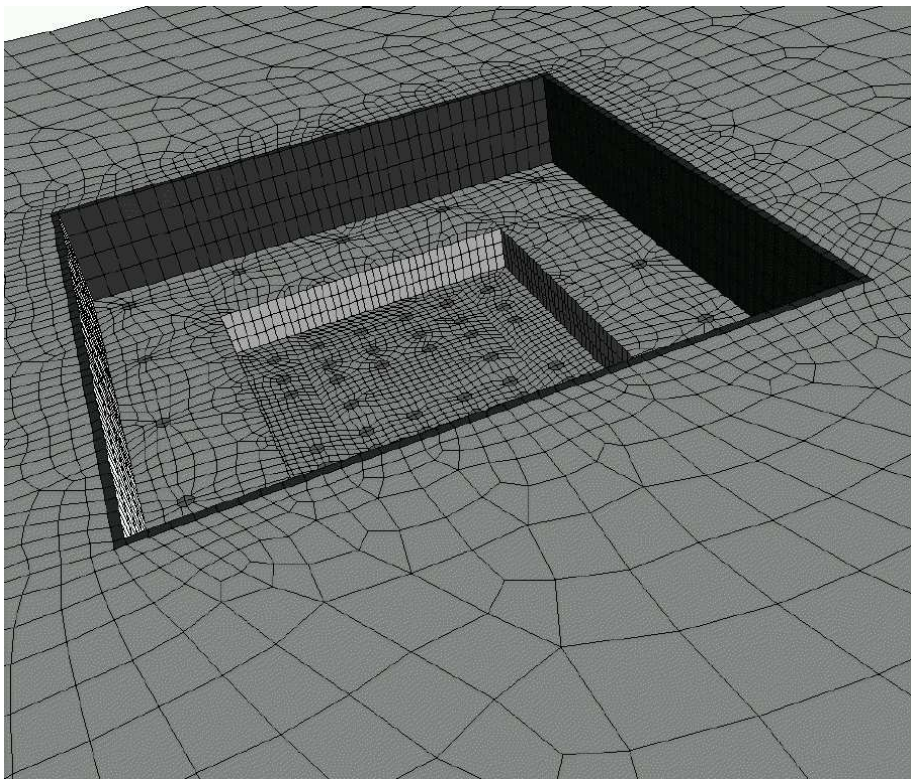


Abbildung 7.11: *TREPTOWERS: Bauzustand nach Aushub*

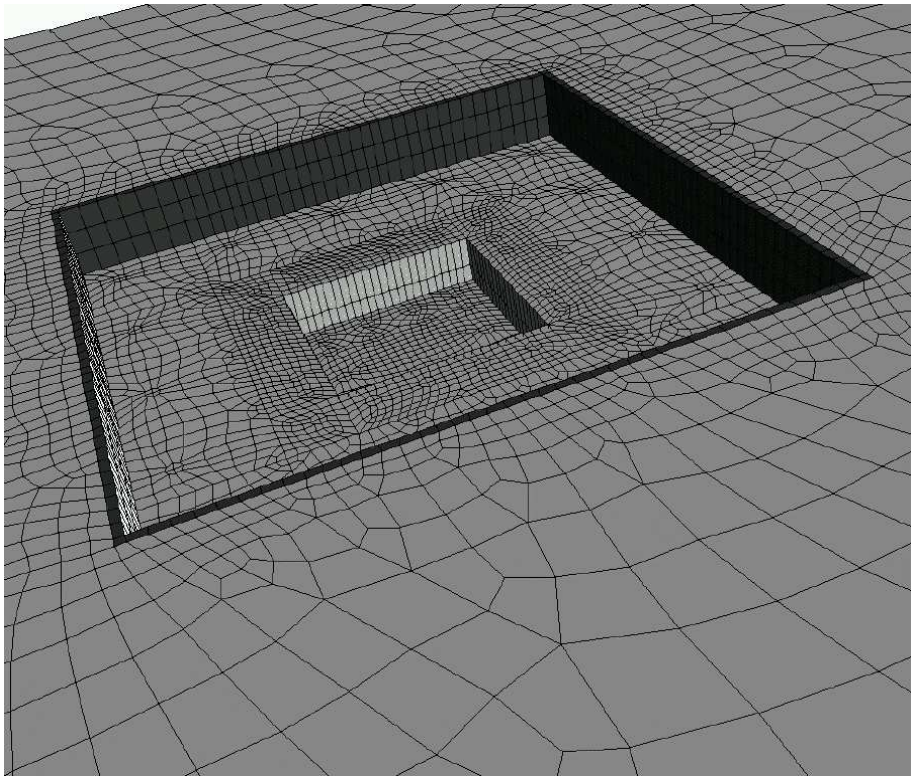


Abbildung 7.12: *TREPTOWERS: Bauzustand nach Fertigstellen der Gründung*

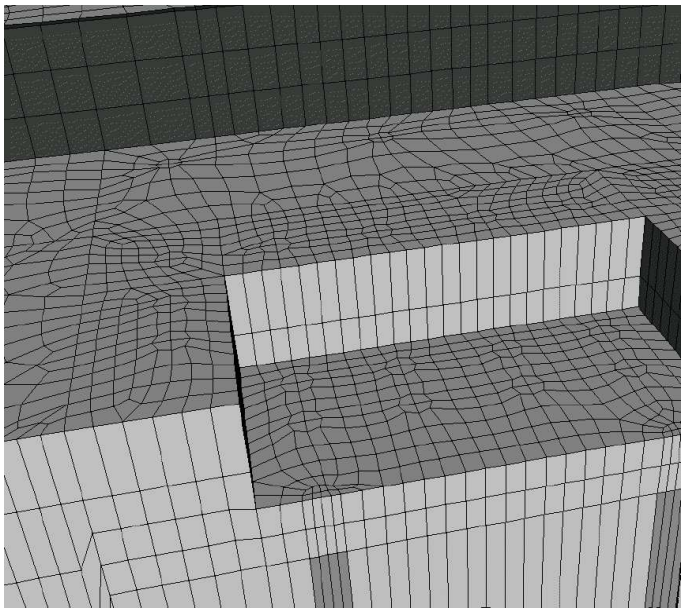


Abbildung 7.13: *TREPTOWERS: Schnitt durch die Baugrube*

Die Diskretisierung der Gründung und des Baugrubenverbaus ist in Abbildung 7.14 zu sehen. Die zwei verschiedenen Sohltiefen der Baugrube sowie der Baugrubenverbau sind zu erkennen. Des weiteren sind die 56 Bohrpfähle mit den verschiedenen Längen zu sehen.

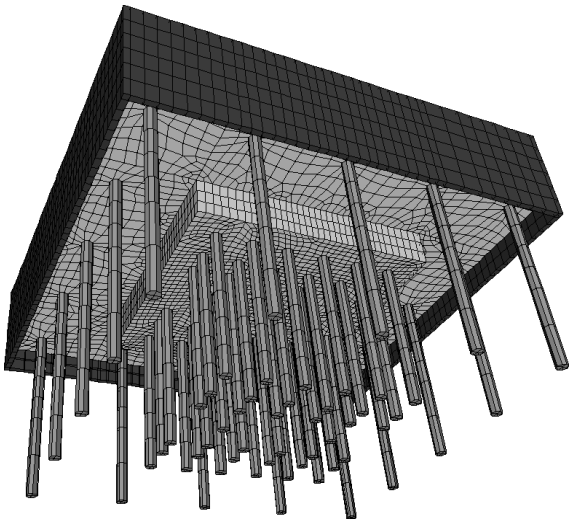


Abbildung 7.14: *TREPTOWERS: Finite-Elemente-Netz der Gründung*

7.2.2 Parallele Netzgenerierung

Die parallele Netzgenerierung wird für das Beispiel durchgeführt. Bei der Steuerung der Partitionierung durch die Lastverteilungsfunktion werden die angestrebte Elementgrößenverteilung für den Endzustand angenommen und alle Verbau- und Gründungselemente berücksichtigt. Nicht berücksichtigt wird die Wegnahme von Elementen im System durch den Aushub der Baugrube. Ein Ausgleich der unterschiedliche Elementanzahlen nach dem Aushub wird nicht durchgeführt.

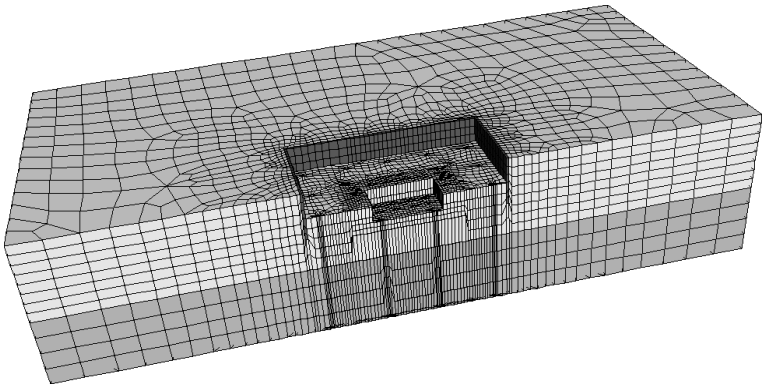


Abbildung 7.15: *TREPTOWERS: Netz eines Prozessors bei der Generierung mit zwei Prozessoren*

In Abbildung 7.15 und 7.16 sind die Teilnetze bei der Berechnung mit zwei bzw. vier Prozessoren für den Bauzustand nach dem Erstellen der Bodenplatte dargestellt.

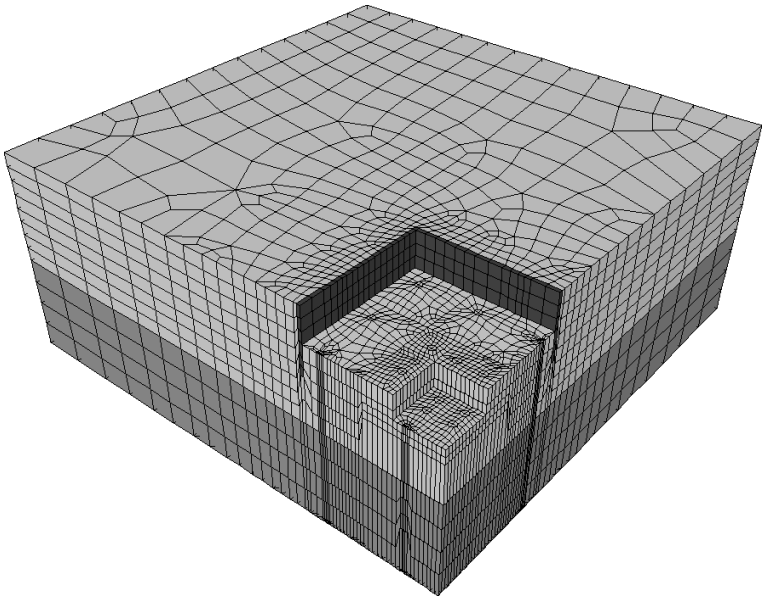


Abbildung 7.16: *TREPTOWERS: Netz eines Prozessors bei der Generierung mit vier Prozessoren*

Die Partitionierung der Baugrundstruktur ist in Abbildung 7.17 als Draufsicht für vier und 16 Prozessoren dargestellt. Der Algorithmus erstellt bei dieser doppelt symmetrischen Problemstellung eine doppelt symmetrische Partitionierung des Gebietes.

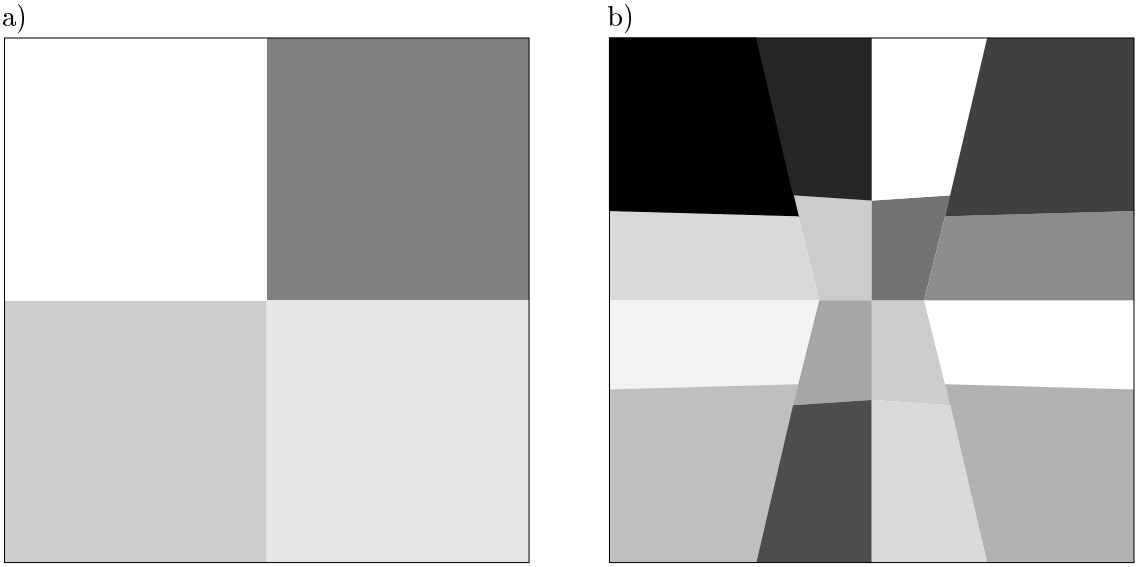


Abbildung 7.17: *TREPTOWERS: Partitionierung mit vier und 16 Prozessoren*

Die kleineren Elemente im Bereich der Baugrube bewirken, dass ein Prozessor, dessen Gebiet im Bereich der Grube liegt, eine kleinere Fläche vernetzen muss als ein Prozessor, dessen Gebiet am Modellrand liegt. Dieser Lastausgleich gelingt durch die vorgegebene Lastverteilungsfunktion relativ gut, so dass die Abweichungen der Elementanzahlen der einzelnen Prozessoren nur gering gegenüber dem Durchschnitt sind. In Tabelle 7.3 sind die durchschnittlichen, die minimalen und die maximalen Elementanzahlen der Diskretisierung für die Partitionierungen mit maximal 32 Prozessoren dargestellt.

Anzahl der Prozessoren	Elementanzahl Durchschnitt	Elementanzahl Minimum	Elementanzahl Maximum
1	45583	45583	45583
2	23089	22817	23361
4	11540	11317	11937
8	5761	5466	6210
16	2885	2621	3247
32	1457	1196	1688

Tabelle 7.3: *TREPTOWERS: Elementanzahlen*

Die Lastverteilung funktioniert für diese Beispiel wesentlich besser als für das Beispiel der Hochbauplatte. Die Anzahl der Elemente im Teilgebiet mit der kleinsten Elementanzahl beträgt

bei der Berechnung mit 16 Prozessoren 80 % der Anzahl der Elemente des Teilgebietes mit der größten Anzahl von Elementen und bei 32 Prozessoren immerhin noch 70 %.

7.2.3 Effektivität der Parallelisierung

Die Beschleunigung der parallelen Berechnung gegenüber der sequentiellen Berechnung ist in Abbildung 7.18 in einem Speed-up-Diagramm dargestellt. Die hierfür notwendigen Zeitmessungen wurden auf einem Linux-Cluster mit Pentium-II Prozessoren mit 350 MHz Taktfrequenz und 128 MB Speicher pro Prozessor durchgeführt. Verwendet wurde ein 100 MBit Netzwerk mit einem Switch als Verteiler.

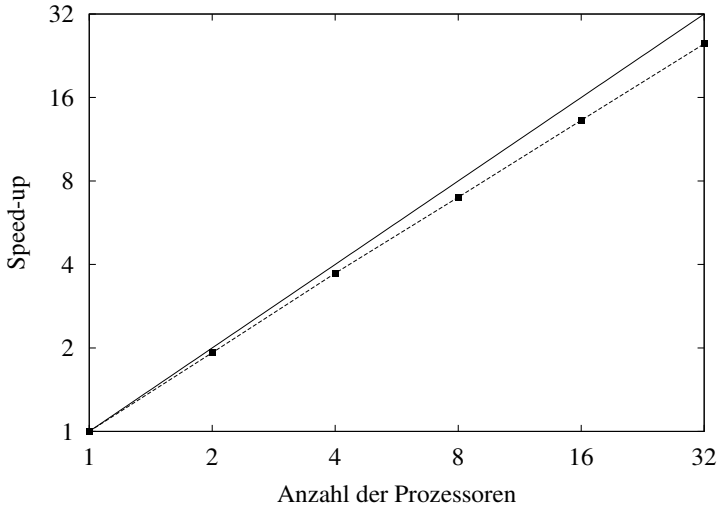


Abbildung 7.18: *TREPTOWERS: Speed-up*

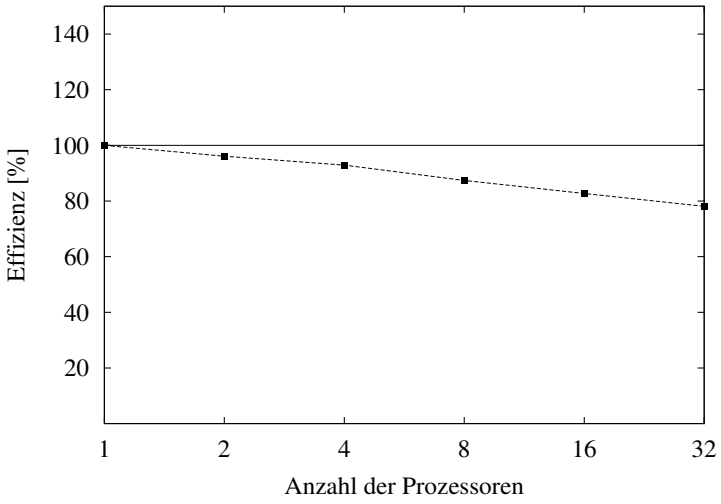


Abbildung 7.19: *TREPTOWERS: Effizienz*

Das in Abbildung 7.19 dargestellte Effizienz-Diagramm zeigt deutlich die gute Effizienz der Parallelisierung. Für Prozessoranzahlen von bis zu vier Prozessoren, bei denen die Elementanzahlen auf den Prozessoren nur gering vom Durchschnitt abweichen, wird eine Effizienz von jeweils über 90 Prozent erreicht. Der schlechtere Lastausgleich und der höhere sequentielle Aufwand bei höheren Prozessoranzahlen spiegeln sich auch im Effizienz- und Speed-up-Diagramm wieder. So liegt die Effizienz bei 16 Prozessoren nur bei 82,7 % und bei der Berechnung mit 32 Prozessoren wird die Rechenleistung von 24,9 Prozessoren erreicht, was einer Effizienz von 78,1 % entspricht.

8 Zusammenfassung und Ausblick

8.1 Durchgeführte Arbeiten und Ergebnisse

Zur Durchführung einer Finite-Elemente-Berechnung auf einem Parallelrechner ist eine partitionierte Diskretisierung erforderlich. Die vorliegende Arbeit zeigt einen alternativen Ansatz, bei dem nicht zuerst eine komplette Diskretisierung oder ein grobes Hintergrundnetz erzeugt werden muss. Ausgehend von einer CAD-Beschreibung des Systems wird eine Partitionierung der Geometrie vorgenommen und die Diskretisierung auf den Koppelrändern festgelegt. Anschließend wird eine Vernetzung der Teilgeometrien durchgeführt.

Für zweidimensionale und für räumliche Problemstellungen wurde der Algorithmus angewendet und mit Hilfe der objektorientierten Modellierung umgesetzt. Im zweidimensionalen Fall wurde am Beispiel einer Hochbauplatte der Liederhalle in Stuttgart die Leistungsfähigkeit demonstriert. Baugrund-Tragwerk-Strukturen wurden als räumliches Anwendungsgebiet verwendet. Hierfür war die Entwicklung von Algorithmen zur kompatiblen Vernetzung der Baugrundstruktur, der Gründung und des Baugrubenverbaus mit Hexaederelementen notwendig. Für das Hochhaus TREPTOWERS in Berlin wurde die Leistungsfähigkeit der parallelen Netzgenerierung dargestellt.

Verschiedene geometrische Qualitätsbeschreibungen für Diskretisierungen wurden untersucht und bewertet sowie Algorithmen zur Glättung und Netzverbesserung vorgestellt. Herangehensweisen zur lokalen Verfeinerung von Diskretisierungen wurden dargestellt und angewendet und Lösungen für spezielle Anforderungen der Verfeinerungen an die Lastverteilung entwickelt.

Die vorgestellte Vorgehensweise zur Parallelisierung von Netzgeneratoren hat Vor- und Nachteile gegenüber anderen Herangehensweisen. Nachteilig im Vergleich zu der sequentiellen Vernetzung mit anschließender Partitionierung ist zum einen die Ausgewogenheit der Partitionierung. Bei der Verwendung graphenbasierter Algorithmen zur Partitionierung des Netzes erhält man Teilgebiete mit jeweils der annähernd gleichen Anzahl von Elementen. Durch die Partitionierung der Geometrie mit dem vorgestellten Algorithmus kann es aufgrund der verwendeten Schwerachsenmethode zu sehr ungleich großen Teilgebieten kommen, besonders wenn die Anzahl der Teilgebiete groß ist oder, wie bei dem Beispiel der Hochbauplatte der Liederhalle, ein großes Loch im zu vernetzenden Gebiet ungünstige Schnittlinien bewirkt.

Ein weiterer Nachteil des Algorithmus gegenüber der parallelen Netzgenerierung unter Zuhilfenahme eines Hintergrundnetzes liegt in der Komplexität beim Zerteilen der Geometrie, was besonders bei komplizierten geometrischen Verhältnissen, wie z.B. bei vielen vorgegebenen Linien, Kanten oder Löchern im Gebietsinneren, problematisch ist und einen erhöhten Rechenaufwand erfordert. Bei diesen komplizierten geometrischen Gegebenheiten hat der vorgestellte Algorithmus aber auch Vorteile. Diese liegen in der relativ guten Teilung der Geometrie, die bei

der Parallelisierung unter Zuhilfenahme eines Hintergrundnetzes nur durch eine hohe Auflösung des Hintergrundnetzes erreicht werden kann, was dann allerdings auch zu schlechten Effizienzen führt.

Die Partitionierung der Geometrie und nicht des Netzes oder eines Hintergrundnetzes hat besonders Vorteile bei der Generierung extrem großer Netze mit vielen Elementen. Das Netz oder ein Hintergrundnetz muss nie komplett auf einem Prozessor vorliegen. Hierdurch erreicht man, dass Netze generiert werden können, die vom Speicherbedarf her gesehen die Größe des Hauptspeichers eines Prozessors um ein Vielfaches übersteigen.

8.2 Möglichkeiten für die Weiterentwicklung

Im Bereich der Generierung der Diskretisierungen für Baugrund-Tragwerk-Strukturen ist die Entwicklung von Algorithmen zur Vernetzung weiterer Bauteile erforderlich. Hierbei ist an Bauteile wie Dämme, Böschungen, Tunnel oder Anker zu denken. Spezielle Lösungen sind hier auch für die Parallelisierung zu suchen. So ist z.B. bei Ankern das Problem der Verbindung des Ankerkopfes mit der Verpressstrecke bei der parallelen Berechnung schlecht auf mehrere Prozessoren verteilbar.

Bei zeitabhängigen Problemstellungen, wie z.B. der Simulationsrechnung der Konsolidierung einer Gründung, können Änderungen im System aufgrund von Bauzuständen auftreten. Hierfür ist die Entwicklung einer Simulationssteuerung erforderlich [Pahl u. a. 2000], die den einzelnen Zeitschritten eine Diskretisierung zuordnet und eine Interpolation der Systemgrößen von einer Diskretisierung auf die nächste übernimmt.

Die Partitionierung des Systems wird in der vorliegenden Arbeit statisch durchgeführt. Nachträgliche Änderungen in der Diskretisierung, wie z.B. durch verschiedene Bauzustände oder notwendige nachträgliche Verfeinerungen, um Fehler in der Berechnung zu minimieren, können zu Veränderungen in der Rechenlast der Teilgebiete führen. Eine automatisierte Umverteilung von Elementen wäre hier denkbar.

Literaturverzeichnis

[Al-Nasra und Nguyen 1991] Al-Nasra, M. ; Nguyen, D.J.: An algorithm for domain decomposition in finite element analysis. In: *Computers & Structures* 39 (1991), S. 227–289

[Artivagas u. a. 1999] Artivagas, P. V. ; Gupta, M. ; Midkiff, S. P. ; Moreira, J. E.: High Performance Numerical Computing in Java: Language and Compiler Issues. In: *Proceedings of the 12'th Workshop on Language and Compilers for Parallel Computers, Aug. 4-6, 1999, San Diego*, 1999

[Bank und Xu 1996] Bank, R. E. ; Xu, J.: An algorithm for coarsening unstructured meshes. In: *Numerische Mathematik* 73 (1996), S. 1–36

[Banks 1995] Banks, David W.: Hybrid Structured/Unstructured Grid Generation: An Object-Oriented Approach. (1995)

[Barnes und Hoffmann 1982] Barnes, E.R. ; Hoffmann, A.J.: Partitioning, spectra and linear programming. / IBM T.J. Watson Research Center. 1982 (RC 9511 (No. 42058)). – Forschungsbericht

[Bathe 1990] Bathe, Klaus-Jürgen: *Finite-Elemente-Methoden.* Berlin [u.a.]: Springer, 1990

[Belkhale und Prithviraj 1990] Belkhale, K.P. ; Prithviraj, P.: Recursive partitions on multiprocessors. In: *5th Distributed Memory Computing Conference*, 1990, S. 930–938

[Betten 1997] Betten, J.: *Finite Elemente für Ingenieure.* Springer, 1997

[Bode 1996] Bode, Arndt: Parallele und verteilte Architekturen für numerische Anwendungen. In: Nagel, W. E. (Hrsg.): *Partielle Differentialgleichungen, Numerik und Anwendungen* Forschungszentrum Jülich GmbH, 1996, S. 195–204

[Bokhari 1981] Bokhari, S.H.: On the mapping problem. In: *IEEE Transactions on Computers* 3 (1981), S. 207–213

[Bronstein 1997] Bronstein, I.N.: *Taschenbuch der Mathematik.* Frankfurt [u.a.]: Deutsch, 1997

[Burghardt 1995] Burghardt, M.: *Visualisierung der Ergebnisdaten von parallelen FE-Berechnungen auf Workstations*, TH Darmstadt, Diplomarbeit, 1995

[Cavendish 1974] Cavendish, J. C.: Automatic triangulation of arbitrary planar domains for the finite element method. In: *Int. J. Mum. Meth. Eng.* 8 (1974), S. 679–696

[Chew 1989] Chew, L. P.: Guaranteed-Quality Triangular Meshes. In: *Technical Report TR-89-983, Department of Computer Science, Cornell University* (1989)

[Chew 1993] Chew, L. P.: Guaranteed-Quality Mesh Generation for Curved Surfaces. In: *Proceedings of the Ninth Annual Symposium on Computational Geometry (San Diego, California), Association for Computing Machinery* (1993), S. 274–280

[Chrisochoides u. a. 1994a] Chrisochoides, N. ; Houstis, E. ; Rice, J.: Mapping algorithms and software environment for data parallel PDE iterative solvers. In: *Parallel and Distributed Computing* 21 (1994), Nr. 1

[Chrisochoides u. a. 1994b] Chrisochoides, N. ; Mansour, N. ; Fox, G.: Performance evaluation of load balancing algorithms for parallel single-phase iterative PDE solvers. In: *Scalable High-Performance Computing Conf. 1994*, IEEE Computer Society Press, Mai 1994b, S. 764–772

[Delaunay 1934] Delaunay, B.: Sur la sphere vide. In: *Bulletin of Academy of Sciences of the USSR,* (1934), S. 793–800

[Dhondt 1999] Dhondt, Guido D.: Unstructured 20-Node Brick Element Meshing. In: *8th International Meshing Roundtable.* Sandia National Laboratory, Albuquerque, NM, 1999. – http://www.cfd.sandia.gov/8imr.html

[Diaz 1998] Diaz, J.: Objektorientierte Modellierung geotechnischer Systeme. In: *Dissertation, TU-Darmstadt, Fachbereich Bauingenieurwesen, Bericht 2/98 des Instituts für Numerische Methoden und Informatik im Bauwesen, Technische Universität Darmstadt* (1998)

[Diekmann u. a. 1994] Diekmann, R. ; Meyer, D. ; Monien, B.: Parallele Partitionierung unstrukturierter Finite Elemente Netze auf Transputernetzwerken. In: R. Flieger, R. G. (Hrsg.): *Parallele Datenverarbeitung aktuell: TAT '94*, IOS Press, 1994, S. 317–326

[Ehlers 1998] Ehlers, W.: Ein Mehrphasen-Stoffmodell für Böden mit Übergang auf Interface-Gesetze. In: *Arbeitsbericht der DFG-Forschergruppe Baugrund-Tragwerk-Interaktion der Technischen Universität Darmstadt für den Zeitraum vom 1.7.96 bis 31.12.97* (1998)

[Farhat 1988] Farhat, C.: A simple and efficient automatic FEM domain decomposer. In: *Computers & Structures* 28 (1988), Nr. 5, S. 579–602

[Farhat und Lesoinne 1993] Farhat, C. ; Lesoinne, M.: Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. In: *International Journal for Numerical Methods in Engineering* 36 (1993), S. 745–764

[Fiedler 1973] Fiedler, M.: Algebraic connectivity of graphs. In: *Czech. Math. J.* 98 (1973), Nr. 23, S. 298–305

[Fiedler 1975] Fiedler, M.: A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory. In: *Czech. Math J.* 25 (1975), Nr. 100, S. 619–633

[Filipiak 1996] Filipiak, Mark: *Mesh Generation.* Edinburgh Parallel Computing Centre, 1996

[Floriani u. a. 1984] Floriani, L. D. ; Falcidieno, B. ; C. Pienovi, G. N.: A hierarchical structure for surface approximation. In: *Computer and Graphics* 8 (1984), S. 183

[Flynn 1972] Flynn, M. J.: Some computer organizations and their effectiveness. In: *IEEE Transactions on Computers* C-21(9) (1972), S. 948–960

[Forum 1993a] Forum, High Performance F.: *High Performance Fortran Language Specification Version 1.0.* Scientific Programming, 1993a

[Forum 1993b] Forum, Message Passing I.: Document for a Standard Message-Passing Interface / University of Tennessee. 1993 (CS-93-214). – Forschungsbericht. Available on **netlib**

[Frey und Field 1991] Frey, W. H. ; Field, D. A.: Mesh relaxation: A new technique for improving triangulations. In: *Intl. J. Numer. Meth. Eng.* 31 (1991), S. 1121–1133

[Fuchs 1998] Fuchs, Alexander: Automatic Grid Generation with Almost regular Delaunay Tetrahedrons. In: *Proceedings 7th International Meshing Roundtable* (1998), S. 133–147

[Geist u. a. 1994] Geist, Al ; Beguelin, Adam ; Dongarra, Jack ; Jiang, Weicheng ; Manchek, Robert ; Sunderam, Vaidy ; Kowalik, Janusz (Hrsg.): *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, Scientific and Engineering Computation, 1994

[Girkmann 1954] Girkmann, Karl: *Flächentragwerke, Einführung in die Elastostatik der Scheiben, Platten, Schalen und Faltwerke.* Wien: Springer, 1954

[Gropp u. a. 1994] Gropp, W. ; Lusk, E. ; Skjellum, A.: *Using MPI.* MIT Press, 1994

[Haendler 1977] Haendler, W.: The impact of classification schemes on computer architectures. In: *Proc. of the 1977 Int. Conf. on Parallel Processing* IEEE, 1977, S. 7–15

[Hartmann und Meißner 1987] Hartmann, T. ; Meissner, U.: About the Numerical Analysis of Dynamics in Multi-Component-Continua. In: Reichel, D. (Hrsg.): *Groundwater Flow and Quality Modelling Series C* Bd. 224, Pup. Comp. Dodrecht, 1987

[Heller 1990] Heller, M.: Triangulation algorithms for adaptive terrain modeling. In: *Proceedings of the 4th International Symposium on Spatial Data Handling* (1990), S. 163–174

[Inprise 1999] Inprise: *Inprise Corporation.* VisiBroker - Naming and Event Services, 1999

[Inscore 1999] Inscore, Jim: The Java Tutorial, A practical guide for programmers: IDL. In: *http://www.javasoft.com/docs/books/tutorial/idl/index.html* (1999)

[Kernighan und Lin 1970] Kernighan, B. ; Lin, S.: An efficient heuristic procedure for partitioning graphs. In: *Bell System Technical Journal* 29 (1970), S. 291–307

[Khan und Topping 1996] Khan, A.I. ; Topping, B.H.V.: *Parallel Finite Element Computations.* Saxe-Coburg Publications, 1996

[Kinney 1997] Kinney, Paul: Clean-Up: Improving Quadrilateral Finite Element Meshes. In: *Proceedings of the sixth annual international meshing roundtable* (1997)

[Lämmer 1997] Lämmer, L.: *Parallelisierung von Anwendungen der Finite-Element-Methode im Bauingenieurwesen.* Habilitation, Technische Hochschule Darmstadt, Bericht 1/97 des Instituts für Numerische Methoden und Informatik im Bauwesen, Technische Hochschule Darmstadt, 1997

[Lee und Schachter 1980] Lee, D. T. R. ; Schachter, B. J.: Two algorithm for constructing a Delaunay triangulation. In: *International Journal of Computer and Information Science* 9 (1980), S. 219–242

[Lewis und Robinson 1978] Lewis, B.A. ; Robinson, J.S.: Triangulation of planar regions with applications. In: *The Computer Journal* 21 (1978), S. 324–332

[Maassen u. a. 2000] Maassen, J. ; Nieuwpoort, R. van ; Veldema, R. ; Bal, H. ; Kielmann, T. ; Jacobs, C. ; Hofman, R.: Efficient Java RMI for Parallel Programming. In: *Vrije Universiteit Amsterdam, Faculty of Sciences* (März 2000)

[McCullagh und Ross 1980] McCullagh, M. J. ; Ross, C.G.: Delaunay triangulation of a random data set for isarithmic mapping. In: *The Cartographic Journal* 17 (1980), S. 93–99

[Meißner u. a. 1996] Meissner, U. ; Lämmer, L. ; Olden, J. ; Burghardt, M.: *Abschlußbericht zum von der DFG geförderten Forschungsvorhaben Me 690/5-2: Entwicklung und Implementierung von Berechnungs- und Entwurfsprozessen im Bau- und Maschinenwesen auf massiv-parallelen Rechnersystemen.* Institut für Numerische Methoden und Informatik im Bauwesen, Technische Hochschule Darmstadt, 1996

[Meißner und Menzel 1989] Meissner, U. ; Menzel, A.: *Die Methode der finiten Elemente.* Springer-Verlag, 1989

[Meißner und Wibbeler 1990] Meissner, U. ; Wibbeler, H.: A Posteriori Errors of Finite Element Models in Groundwater and Seepage Flow. In: *Computational Methods in Subsurface Hydrology* (1990)

[Meyers und Tautges 1998] Meyers, Ray J. ; Tautges, Timothy J.: The Hex-Tet Hex-Dominant Meshing Algorithm as Implemented in CUBIT. In: *Proceedings 7th International Meshing Roundtable* (1998)

[Mirante und Weingarten 1982] Mirante, A. ; Weingarten, N.: The radial sweep algorithm for constructing triangulate irregular networks. In: *IEEE Computer Graphics and Applications* 2 (1982), S. 11–21

[Mitchell 1996] Mitchell, S. A.: A characterization of the quadrilateral mesh of a surface which admit a compatible hexahedral mesh of enclosed volume. In: *Proc. STACS' 96 Grenoble* (1996)

[Mitchell und Vavasis 1997] Mitchell, Scott A. ; Vavasis, Stephen A.: Quality Mesh Generation in Higher Dimensions. In: *6th International Meshing Roundtable.* Sandia National Laboratory, Albuquerque, NM, 1997

[von Neumann 1946] Neumann, John von: The Principles of Large-Scale Computing Machines. (1946)

[Nour-Omid u. a. 1987] Nour-Omid, B. ; Raefsky, A. ; Lyzenga, G.: Solving finite element equations on concurrent computers. In: Noor, A.K. (Hrsg.): *Parallel computations and their impact on mechanics* Bd. 86. ASME, New York, 1987, S. 209–228

[Olden 1998] Olden, J.: Finite-Element-Analyse von Plattentragwerken durch adaptive Software-Techniken. In: *Dissertation, Technische Universität Darmstadt, Fachbereich Bauingenieurwesen, Bericht 1/98 des Instituts für Numerische Methoden und Informatik im Bauwesen, Technische Universität Darmstadt* (1998)

[Owen 1999] Owen, Steven J.: Constrained Triangulation: Application to Hex-Dominant Mesh Generation. In: *Proceedings 8th International Meshing Roundtable*, 1999

[Pahl u. a. 2000] Pahl, P. J. ; Damrath, R. ; Enseleit, J. ; Laabs, A. ; Huhnt, W.: Objektorientierte Analyse und Visualisierung zeitabhängiger physikalischer Zustände dreidimensionaler Körper. In: Hartmann, Dietrich (Hrsg.): *Objektorientierte Modellierung in Planung und Konstruktion*, Wiley-VCH, Weinheim, 2000, S. 141–170

[Pothen u. a. 1990] Pothen, A. ; Simon, H.D. ; Liou, K.-P.: Partitioning sparse matrices with Eigenvectors of graphs. In: *SIAM J. Matrix Appl.* 11 (1990), Nr. 3, S. 430–452

[Powers 1988] Powers, D.: Graph partitioning by eigenvectors. In: *Lin. Alg. Appl.* 101 (1988), S. 121–133

[Price und Armstrong 1997] Price, M. A. ; Armstrong, C.G.: Hexahedral Mesh Generation by Medial Axis Subdivision: Solid width Flat and Concave Edges. In: *Int. Jou. Num. Meth. Eng.* 40 (1997), S. 3335–3359

[Price u. a. 1995] Price, M. A. ; Armstrong, C.G. ; Sabin, M.A.: Hexahedral Mesh Generation by Medial Axis Subdivision: Solid width convex edges. In: *Int. Jou. Num. Meth. Eng.* 38 (1995), S. 3335–3359

[Rodatz 1992] Rodatz, Walter: *Vorlesungsumdruck Grundbau und Unterirdisches Bauen.* Institut für Grundbau und Bodenmechanik, TU-Braunschweig, 1992

[Ruben 2000] Ruben, Jochen: *Automatische Einarbeitung von Randbedingungen in einen Finite-Element-Netzgenerator.* Technische Universität Darmstadt, 2000

[Ruppert 1993] Ruppert, J.: A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation. In: *Proceedings of the Fourth Annual Symposium on Discrete Algorithms, Association for Computing Machinery* (1993), S. 83–92

[Ruppert 1995] Ruppert, J.: A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. In: *Journal of Algorithm* 18 (1995), S. 548–585

[Rypl und Bittnar 1997] Rypl, D. ; Bittnar, Z.: Parallel 3D Mesh Generator. In: *Proceedings of the International Conference on Mathematical and Computer Modeling and Scientific Computation held in Washington DC* (1997)

[Saxena u. a. 1990] Saxena, S. ; Bhatt, B. C. P. ; Prasad, V. C.: Efficient vlsi parallel algorithm for delaunay triangulation in orthogonal tree network in two and three dimensions. In: *IEEE Transctions on Computers* 39 (1990), S. 400–404

[Schneiders 1996a] Schneiders, R.: A grid-based algorithm for the generation of hexahedral element meshes. In: *Engineering with Computers 12* (1996), S. 168–177

[Schneiders 1996b] Schneiders, R.: Refining Quadrilateral and Hexahedral Element Meshes. In: *Proceedings NUMIGRID 96* (1996), S. 679–689

[Schneiders u. a. 1996a] Schneiders, R. ; Schindler, R. ; Weiler, F.: Octree-based Generation of Hexahedral Element Meshes. In: *Proceedings 5th International Meshing Roundtable, Pittsburgh, USA (1996)* (1996)

[Schneiders u. a. 1996b] Schneiders, R. ; Schindler, R. ; Weiler, F.: Octree-based Generation of Hexahedral Element Meshes. In: *Proceedings 5th International Meshing Roundtable, Pittsburgh, USA* (1996)

[Schöberl 1997] Schöberl, J.: NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules. In: *Comput.Visual.Sci.* 1 (1997), S. 41–52

[Schönenborn 1999] Schönenborn, Ingo: Prozeßorientierter Entwurf einer Boden-Tragwerk-Struktur. In: *Dissertation, Technische Universität Darmstadt, Fachbereich Bauingenieurwesen, Bericht 1/99 des Instituts für Numerische Methoden und Informatik im Bauwesen, Technische Universität Darmstadt* (1999)

[Schwarz 1991] Schwarz, H. R.: *Methode der Finiten Elemente.* B. G. Teubner Verlag, Stuttgart, 1991

[Shaw 1999] Shaw, J. A.: Hybrid Meshes. In: Thompson, Joe F. (Hrsg.) ; Soni, Bharar K. (Hrsg.) ; Weatherhill, Nigel P. (Hrsg.): *Handbook of Grid Generation*, CRC Press, Boca Raton, FL, 1999

[Sheffer u. a. 1998] Sheffer, A. ; Etzion, M. ; Rappoport, A. ; Bercovier, M.: Hexahedral Mesh Generation using the Embedded Voronoi Graph. In: *Proceedings 7th International Meshing Roundtable* (1998)

[Shewchuk 1997] Shewchuk, J. R.: *Delaunay Refinement Mesh Generation.* School of Computer Science, Pittsburgh, Phd-Thesis, 1997

[Shimada u. a. 1997] Shimada, Kenji ; Yamada, Atsushi ; Itoh, Takayuki: Anisotropic Triangular Meshing of Parametric Surfaces via Close Packing of Ellipsoidal Bubbles. In: *Proceedings of the sixth annual international meshing roundtable* (1997)

[Shore 1973] Shore, J.E.: Second thoughts on parallel processing. In: *Computers and Electrical Engineering* 1 (1973), Nr. 1, S. 95–109

[Simon 1991] Simon, H.D.: Partitioning of unstructured problems for parallel processing. In: *Computing Systems in Engineering* 2 (1991), Nr. 2/3, S. 135–148

[Talmor u. a. 1995] Talmor, G. L. Millerand D. ; Teng, S.-H. ; Walkington, N.: A Delaunay Based Numerical Method for Three Dimensions: Generation, Formulation and Partition. In: *Proceedings of the Twentyseventh Annual ACM Symposium on Theory of Computing (Las Vegas, Nevada)* (1995)

[Taniguchi u. a. 1996] Taniguchi, Takeo ; Goda, Tomoaki ; Kasper, Harald ; Zielke, Werner: Hexahedral Mesh Generation of Complex Composite Domain. In: *Proceedings 5th International Conference on Numerical Grid Generation in Computational Field Simulations*, Mississippi State University, 1996, S. 699–707

[Taniguchi und Katagiri 1996] TANIGUCHI, Takeo ; KATAGIRI, Hiroki: New Concept of Hexahedral Mesh Generation For Arbitrary 3D Domain - Block Generation Method. In: *Proceedings 5th International Conference on Numerical Grid Generation in Computational Field Simulations*, Mississippi State University, 1996, S. 671–678

[Taub 1961] TAUB, A. H. (Hrsg.): *John von Neumann: Collected Works.* Pergamon Press, Oxford,U.K., 1961

[Taylor 1998] TAYLOR, Robert L.: FEAP – A Finite Element Analysis Program / Department of Civil and Environmental Engineering, University of California at Berkley. 1998. – Forschungsbericht

[Thole 1996] THOLE, Clemens-August: Programmiersprachen für Höchtleistungsrechner: MPI und HPF. In: NAGEL, W. E. (Hrsg.): *Partielle Differentialgleichungen, Numerik und Anwendungen* Forschungszentrum Jülich GmbH, 1996, S. 69–82

[Thompson und Soni 1999] THOMPSON, David S. ; SONI, Bharat K.: Generation of Quad and Hex Dominant Semistructured Meshes Using an Advancing Layer Scheme. In: *Proceedings 8th International Meshing Roundtable*, 1999

[Törnig und Spellucci 1988] TÖRNIG, W. ; SPELLUCCI, P.: *Numerische Mathematik für Ingenieure und Physiker.* Springer Verlag, 1988

[Voronoi 1908] VORONOI, M. G.: Nouvelles applications des parametres continus a la theorie des formes quadratiques. In: *Journal Reine und Angewandte Mathematik* (1908), Nr. 134, S. 198–287

[Watson 1981] WATSON, D. F.: Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. In: *The Computer Journal* 24 (1981), S. 167–172

[Wilson und Topping 1996] WILSON, J. ; TOPPING, B.H.V.: Parallel and Distributed Adaptive Tetrahedral Mesh Generation. In: *Advances in Computational Structures Technology, Civil-Comp Press, Edinburgh* (1996), S. 327–342

[Wollrath und Waldo 1999] WOLLRATH, Ann ; WALDO, Jim: The Java Tutorial, A practical guide for programmer: RMI. In: *http://www.javasoft.com/docs/books/tutorial/rmi/index.html* (1999)

[Wriggers 1998] WRIGGERS, P.: Formulierung und Simulation der Kontaktvorgänge bei der Interaktion Boden-Bauwerk. In: *Arbeitsbericht der DFG-Forschergruppe Baugrund-Tragwerk-Interaktion der Technischen Universität Darmstadt für den Zeitraum vom 1.7.96 bis 31.12.97* (1998)

[Zienkiewicz und Zhu 1991] Zienkiewicz, O. C. ; Zhu, J. Z.: Adaptivity and Generation. In: *International Journal for Numerical Methods in Engineering* (1991), S. 783–810

[Zienkiewicz 1975] Zienkiewicz, Olgierd C.: *Methode der finiten Elemente.* München [u.a.]: Hanser, 1975