Entwurf eines integrierten Systems zur Visualisierung von Ergebnissen numerischer Berechnungsverfahren für massiv parallele Rechnerarchitekturen

Von der Fakultät für Energietechnik der Universität
Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von
Hans-Ulrich Schlageter
geboren in Lörrach

Hauptberichter: Prof. Dr.-Ing. R. Rühle
Mitberichter: Priv.-Doz. Dr.-Ing. habil. F. Schmidt
Tag der Einreichung: 16.03.1999
Tag der mündlichen Prüfung: 24.03.2000

Anwendungen der Informatik im Maschinenbau (AIM),
Fakultät für Energietechnik
der Universität Stuttgart
2000

Meinem Vater
Dr. phil. Albrecht Rudolf Schlageter
1927 – 1999

Sobald durch Strebsamkeit der Weise
Die Trägheit überwunden hat
Erklommen hat den Turm der Einsicht
Blickt sorgenfrei der Held hinab
Auf die gequälte Welt der Toren
Gleichwie von einem Berg ins Tal

aus Milindapañha, der Berg

Zwei Worte vorweg

Die vorliegende Dissertation entstand im wesentlichen während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Computeranwendungen der Universität Stuttgart in den Jahren 1990 bis 1996. Zu ihrer Entstehung haben viele Personen in ganz unterschiedlicher Weise beigetragen. Ich möchte an dieser Stelle auch alljenen danken, die ich nicht nennen werde.
Herrn Prof. em. Dr. J. Argyris, Institut für Computeranwendungen, danke ich für die freundliche Aufnahme am Institut und das anhaltende Interesse an meiner Arbeit.
Ebenso danken möchte ich Herrn Prof. Dr. G. Wittum, jetzt Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, Universität Heidelberg, der mir die Fortführung der Arbeiten am Institut ermöglichte.
Besonderen Dank schulde ich Herrn Prof. Dr. R. Rühle, Rechenzentrum Universität Stuttgart, für die Unterstützung meiner Arbeit und die Übernahme des Hauptberichts.
Herrn Priv. Doz. Dr. habil. F. Schmidt, Institut für Kernenergetik und Energiesysteme, danke ich für die freundliche Übernahme des Mitberichts.
Herrn Dr. U. Lang, Höchstleistungs-Rechenzentrum Stuttgart, gilt mein Dank für die Durchsicht des Manuskripts sowie seine wertvollen Anregungen und seine generelle Unterstützung.
Ferner danke ich allen Mitgliedern des Instituts für Computeranwendungen für die entspannte wissenschaftliche Arbeitsatmosphäre.
Insbesondere danke ich Herrn G. Frik für seine jahrelange freundschaftliche Betreuung.
Herzlich danke ich auch Frau Dr. M. Haase für ihren guten Rat und ihre Vermittlung.
Auf keinen Fall vergessen möchte ich meine ehemaligen Institutskollegen Frau G. Beddies und die Herren Dr. M. Eggers, G. Faust, H. Friz, Dr. A. Hopf, Dr. B. Huurdeman, Dr. S. Nölting, R. Handel, M. Schimmler, Dr. J. Urban. Sie alle haben auf ihre Weise zum Gelingen der Arbeit beigetragen, durch ihre fachliche Zusammenarbeit und ihren Humor, vor allem aber durch die 'strömungssimulatorischen Anforderungen', die sie an die Visualisierung zu stellen wußten.
Nicht zuletzt danke ich meiner Frau Gabriele und meinen Kindern Tara und Tillman für ihre Geduld, ihr Verständnis und vor allem für die mir gewährten Freiräume.

Trossingen, im März 2000 Hans-Ulrich Schlageter

Kurzfassung

Der verbreitete Einsatz von Rechnern mit verteiltem Speicher in der Numerischen Strömungsmechanik CFD hat einen Engpass bei der Interpretation der anfallenden riesigen Datenmengen verursacht, da trotz dieser Entwicklung die Visualisierung üblicherweise immer noch auf sequentiellen Workstations durchgeführt wird.
Mit dem Ziel, den Simulationsprozeß als Ganzes zu beschleunigen, wird in dieser Arbeit ein integriertes Visualisierungssystem vorgestellt, das sowohl aus interaktiven als auch parallelen Komponenten besteht. Das Programmsystem erlaubt es, zahlreiche unabhängige Software-Einheiten bzw. Moduln individuell zu gruppieren und zu einer benutzerdefinierten Anwendung zusammenzubinden.
Die Software-Einheiten umfassen Eingabe, Ausgabe, Export und Import von Daten genauso wie eine Bandbreite an Visualisierungstechniken zum sequentiellen und parallelen Gebrauch. Zusätzlich zu den Standardtechniken wie z.B. Isolinien, Konturflächen, Isoflächen, Vektoren und Stromlinien werden Methoden für die Verfolgung von Objekten angeboten. Dies schließt neben der Partikelverfolgung Kugeln und Flächen ein, die sich während des Verfolgungsvorgangs aufgrund verschiedener Geschwindigkeiten an den betrachteten Oberflächenpunkten deformieren.
Das in AVS [45] eingebettete Modulpaket läuft auf einer schnellen Graphik-Workstation und gestattet die Anbindung eines Parallelrechners als Backend, zum einen, um durch Verteilen den Berechnungsaufwand zu verringern und zum anderen, um visuelle Zwischenergebnisse während des Laufs einer parallelen CFD-Simulation zu erhalten. In der einen, hier vorgestellten Lösung bilden parallele Analyse [84][66] und Visualisierung ein hybrides System mit interaktivem Charakter. Dabei werden aufwendige Berechnungen der Visualisierung in die Analyse integriert und die auf diese Weise erhaltenen Polygondaten über das Netzwerk an eine assoziierte Graphik-Workstation zum Rendern übermittelt [7]. In einer weiteren parallelen Anwendung werden Analyse–Ergebnisdaten, die bereits auf einer Festplatte gespeichert vorliegen, im sequentiellen Postprocessing aufbereitet, wobei jedoch zeitraubende Berechnungen einem Parallelrechner übertragen werden, beispielsweise wenn ein Stromlinenbündel berechnet oder Partikel verfolgt werden sollen.
Darüber hinaus bleibt das Visualisierungssystem nicht auf parallele Anwendungen beschränkt, sondern erfüllt viele andere Ansprüche [6], da die Mehrheit der Benutzer die Visualisierung in der konventionellen Weise als Pre- und Postprocessing-Instrument einsetzt.
In vielen Anwendungen ist das Management des Datenflusses zwischen den Hauptkomponenten des Systems und den verschiedenen beteiligten Rechnern wichtig. Verbindungen werden interaktiv auf Benutzeranforderung aufgebaut. Der Austausch von Daten und Anweisungen über das Netzwerk wird über eine neuentwickelte, auf Sockets basierende Transmitter–Software abgewickelt, die zur Laufzeit Punkt–zu–Punkt–Verbindungen anbieten kann, die nicht bereits im Voraus spezifiziert bzw. erstellt werden müssen. Nicht mehr benötigte offene Verbindungen werden im Zuge der Beendigung der parallelen Berechnung automatisch abgebaut. Auf diese Weise stehen in einem Mehrplatzbetrieb Prozessoren und ebenso Endpunkte der Kommunikation wieder anderen Benutzern zur Verfügung.

Abstract

The wide use of computers with distributed memory in Computational Fluid Dynamics caused a bottleneck in interpreting vast quantities of data, since usually visualization is still effected on sequential workstations despite this evolution.
In order to accelerate simulation process as a whole, an integrated visualization system is presented in this work comprising both parallel and interactive components. The system allows numerous independent software entities called modules to be individually grouped and linked together to user-specific applications.
The software entities comprise facilities which manage the input, output, import and export of data as well as a wide range of visualization techniques for both sequential and parallel use. In addition to the standard techniques like isolines, contour planes, isosurfaces, vectors or streamlines etc. methods for the tracing of objects are offered. Apart from tracing particles this includes spheres and planes, which during the tracing do deform due to the different velocities in each considered point of the surface.
The module package runs on a powerful graphics workstation under AVS [45] and permits the connection of parallel machines as the back end in order to distribute computational effort and to receive visual output during the run of a parallelized CFD simulation.
One solution which will be proposed here is to let parallelized analysis [84, 66] and visualization form a hybrid system with a strongly interactive character. The aim is achieved by integrating the time-intensive calculations of the visualization into the analysis and transporting thus obtained polygon data over the network to an associated graphics workstation for rendering purposes [7].
A further parallel application sequentially post-processes data which are already stored on a hard disk, but performs time-consuming calculations in parallel, e.g. if a bundle of streamlines is to be computed or particles are to be tracked.
Besides, the visualization system is not limited to parallel applications but features many other requirements [6], since the majority of the users will still use visualization in the conventional way as a pre- and post-processing tool.
A very important point in many applications is the management of the data flow between the main components of the system and different machines. Point-to-point connections need to be established interactively whenever requested by the user. The exchange of data and instructions over the network is effected directly by a newly developed transmitter software able to offer point-to-point connections which need be neither specified nor set up in advance. As soon as not needed any further, connections are dismissed, together with the termination of a parallel computation. Thus, in parallel multiuser systems, for instance, processors and endpoints of communication likewise are at other user's disposal again.

Inhaltsverzeichnis

Symbolverzeichnis

n_p	Prozessorzahl
S	Speed-up
E	Effizienz
Sc	Scale–up
D	Problemgröße
μ	normierter serieller Anteil
$n_{p_{max}}$	maximaler Parallelitätsgrad
fw	Farbwert
fs	Streckfaktor
w_{kn}	Knotenwert
w_{el}	Elementwert
n_{el}	Zahl angrenzender Elemente
x_{min}	Wertebereichsminimum
x_{max}	Wertebereichsmaximum
Δ_{inc}	Inkrement
x_{iso_i}	i-ter diskreter Isowert
w_{iso}	interpolierter Isowert
E_{xy}	Ebene XY
E_0	Grundebene
$\vec{a}$	Vektor des Ebenenaufpunkts
$\vec{x}_s$	Koordinaten des Schnittpunkts
$\vec{e}_s$	Ergebnisvektor an einem Schnittpunkt
$\vec{p}_i$	Koordinaten des Punktes i
d_i	Abstand i

$\vec{\omega}$	Wirbelstärke
$\vec{\omega}_i$	Wirbelstärke Teilfläche i
Γ	Zirkulation
Γ_i	Zirkulation Teilfläche i
$\vec{n}_{0_i}$	Normalenvektor Teilfläche i
A_i	Inhalt der Teilfläche i
U_i	Umgebungspunkt i
$\vec{m}$	Kreismittelpunkt
φ_{max}	maximaler Segmentwinkel
φ	Winkelinkrement
dr	radialer Richtungsvektor
r_i	Innenradius
r_a	Außenradius
ds_0	Anfangsschrittweite
$\vec{x}_0$	Koordinaten des Startpunkts
$\vec{x}_1$	Koordinaten des Zielpunkts
v_0	Geschwindigkeit im Startpunkt
v_1	Geschwindigkeit im Zielpunkt
Ma_∞	Machzahl im Freistrom
ρ_∞	Dichte im Freistrom
T_∞	absolute Temperatur im Freistrom
Pr	Prandtl-Zahl
Ra	Rayleigh-Zahl

Kapitel 1

Einleitung

1.1 Die Visualisierung als Bindeglied in der numerischen Simulation

Der Einzug der Parallelrechner in technisch–wissenschaftliche Bereiche hat in den letzten Jahren einen Innovationsschub ausgelöst, der sich vor allem in der numerischen Behandlung komplexer, nichtlinearer physikalischer Problemstellungen niederschlägt. In allen Schlüsselforschungsgebieten, – sei es die Klimamodellierung, die Simulation einer Schadstoffausbreitung im Erdreich, aber auch die Nachbildung einer chemisch reagierenden hypersonischen Strömung beim Wiedereintritt eines Raumflugkörpers in die Erdatmosphäre, – kommt der Angewandten Numerik somit eine Vorreiterrolle zu, und der Vorstoß der Parallelrechner in die kommerzielle Nutzung beginnt sich bereits abzuzeichnen.
Eine numerische Simulation gliedert sich im wesentlichen in die drei für sich stehenden Teilgebiete Preprozessing, Prozessing und Postprozessing, denen die Netzgenerierung, die numerische Analyse und die graphische Ergebnisaufbereitung weitestgehend entspricht.
Lag der Schwerpunkt der Parallelisierung neben der Erzeugung paralleler Teilnetze [66, 17, 5] bisher vor allem auf der numerischen Analyse und hier besonders auf den Lösungsalgorithmen [8, 83], wird man in der Konsequenz der Entwicklungen nicht umhin können, die Simulation einschließlich der Visualisierung als einen Gesamtprozeß zu begreifen und folglich letztere an den Parallelisierungskonzepten zu beteiligen [65, 59]. Die Visualisierung wird dann sogar das entscheidende Bindeglied der Simulationskette sein, weil sie neben ihrer klassischen Bestimmung reinen

Ergebnisauswertens Funktionen der Datenreduzierung [69] übernehmen und in die Aufgabe der Integration aller Komponenten hineinwachsen muß.
Datenreduktion und Integration sind sogar ganz wesentliche Aspekte, nicht zuletzt, weil es im Zuge der Berechnungen immer speicherintensiverer Probleme schon heute in immer kürzerer Zeit enorme Datenmengen graphisch auszuwerten gilt, die sich kaum mehr überschauen und nur unter direkter Mitwirkung des Anwenders am Ort ihres Entstehens eindämmen lassen.
Obwohl in jüngster Zeit vermehrt dazu übergegangen wird, auch die Visualisierung in die Parallelisierung mit einzubeziehen [36, 79, 41, 58, 23], kann doch behauptet werden, daß eine graphische Auswertung bis heute im allgemeinen nur seriell vorgenommen wird, was es mit sich bringt, daß die Daten vor der Weiterverarbeitung zusammengefaßt und zwischengespeichert werden müssen.
Der Engpaß liegt hierbei zum einen bei der Visualisierung selbst, zum anderen aber auch in der bereits angesprochenen Koordinierung bzw. Kopplung und Abstimmung der einzelnen Schritte der Simulationskette miteinander.

Aus dem Zuvorgenannten lassen sich zunächst drei Problemfelder formulieren, die zugleich als Aufgabenfelder für die Visualisierung genommen werden können: Das Problem der interaktiven Zugänglichkeit aller Teilschritte einer Simulation, das Problem der Datenreduktion bzw. der Bewältigung der ernormen Datenmengen im Vorfeld der Auswertung, sowie das Problem der Integration, d.h. der Abstimmung der einzelnen Arbeitsschritte aufeinander, die mit der Optimierung der Datenflüsse und der Verwaltung gemeinsamer Speicherressourcen einhergeht.
Als vierter Punkt kommt die eigentliche Aufgabe hinzu, nämlich die Visualisierung selbst, und als eine fünfte Vorgabe kann gelten, daß alle diese Aufgaben durch ein einziges zentrales Programmsystem bewältigt werden, u.a. damit ein Anwender nicht gezwungen ist, mit vielen verschiedenen Systemen zu operieren.
Es ist offensichtlich, daß die hier angesprochenen Problempunkte sich nicht klar voneinander abgrenzen lassen. So ist eine parallele Datenreduktion etwa mit parallelen Datenfilterungs- und Mappingalgorithmen eng verknüpft, wie das Beispiel einer parallelen Berechnung von Isoflächensegmenten zeigt [62]. Gleiches trifft für die Zugänglichkeit zu, die mit den Aufgaben der Integration und Steuerung in Zusammenhang steht. Dennoch ist an Hand dieser Punkte die eigentliche Problemstellung bereits charakterisiert.

1.2 Zielsetzung

Seit einigen Jahren zeichnet sich ein grundlegender Wandel in den Methoden der computergestützten numerischen Simulation physikalischer Vorgänge ab. Dies ist jedoch nicht alleine leistungsfähigeren Prozessoren und höheren Rechnerkapazitäten zuzuschreiben, sondern der Einführung der Parallelrechner und der Vernetzung verschiedener Rechner miteinander. Dadurch eröffneten sich Möglichkeiten, die ganz neue Konzepte und Programmiermodelle erfordern. Im selben Maße sind auch die Anforderungen und Wünsche seitens der Anwender gestiegen.
Die Aufgaben eines Visualisierungsprogrammsystems der kommenden Generation dürfen somit nicht ausschließlich auf das klassische Einsatzgebiet einer reinen Ergebnisaufbereitung im Sinne des Postprozessings beschränkt bleiben, sondern es müssen neue Aufgabengebiete erschlossen werden. Hierzu einen Beitrag zu leisten, hat sich die vorliegende Arbeit zum Ziel gesetzt.

Aufgabe war es, ein Visualisierungsprogrammsystem speziell für die numerische Strömungsmechanik zu konzipieren und zu entwickeln, das den veränderten Bedingungen Rechnung trägt, in dem es nicht nur Aufgaben des Postprozessings wahrnimmt, sondern überdies mit einer numerischen Simulation bzw. numerischen Finite Element Analyse, sei sie sequentiell oder parallel, zu einem Hybridsystem verbunden werden kann. Werden zusätzlich noch Aufgaben der Steuerung verschiedener beteiligter Prozesse und die Regelung des Datenaustauschs zwischen Komponenten übernommen, ist das Ziel einer Beschleunigung des gesamten Simulationsprozesses erreicht und der Weg zu einer Echtzeitsimulation eingeschlagen.

1.2.1 Ziele aus der Perspektive der Simulation

Wenn man den Simulationsprozeß als Ganzes betrachtet, so wird diesen Zielen der im vorigen Abschnitt aufgestellte Forderungskatalog gerecht, der die Aufgabenfelder Zugänglichkeit, Integration und Visualisierung abdeckt:

- **Interaktive Zugänglichkeit der einzelnen Phasen der Simulation**
  Der interaktive Zugang, physikalisch wie zeitlich, erfordert einen streng modularen Aufbau des Systems, damit sich beispielsweise Teilgebiete besser koordinieren lassen und einzelne Abschnitte unter veränderten Bedingungen wiederholt werden können. Hier ist eine interaktive Kontrolle des Gesamtablaufs erwünscht.
- **Integration der verschiedenen Komponenten des Systems**
  Eine Integration bzw. Kopplung einzelner Komponenten miteinander sollte

durch Verwendung ähnlicher, auf einander abgestimmter Datenstrukturen erreicht werden, sowie durch die Ermöglichung eines direkten Datenaustauschs zwischen aktiven Prozessen, die gegebenenfalls auf verschiedenen Rechnern ablaufen (*distributed processing*).
Es muß gelingen, Performance–Verluste innerhalb des Hybridsystems möglichst gering zu halten, die vorwiegend auf den nicht optimierten Datentransport zwischen den einzelnen Komponenten zurückzuführen sind, sowie auf Kommunikationsmängel, die durch die Verschiedenheit beteiligter Systeme bedingt sind.

- **Datenreduktion**
  Eine Datenreduktion wird erzielt, indem verschiedene Berechnungsalgorithmen jeweils nur Teilaspekte der Ausgangsdaten herausarbeiten, oder indem nur eine beschränkte Auswahl aus dem Spektrum der Daten berücksichtigt wird (Filtermethoden, Bereichseinschränkung).

- **Visualisierung der Daten**
  Die Visualisierung begreift sich als eine Serie von Transformationen, infolge derer aus den rohen Simulationsdaten ein darstellbares Bild erzeugt wird. Die einzelnen Schritte dieser "Graphics Pipeline" werden üblicherweise "Data Enrichment and Enhancement", "Visualization Mapping" und "Visualization Rendering" bezeichnet [24] und sollen in der Folge knapp umrissen werden.
  Während des Schrittes "Datenanreicherung und Erweiterung" (siehe Abbildung 1.1) werden die Ursprungsdaten (oder ein Auszug aus denselben) für die speziellen Belange der Darstellung aufbereitet. Vielleicht enthalten die Daten ein Rauschen und müssen geglättet und gefiltert werden, oder eine weitere physikalische Größe soll berechnet werden. Der resultierende erweiterte Datensatz wird abgeleitende Daten genannt.
  Bei dem sich anschließenden Visualization Mapping, kurz Mapping, werden aus den abgeleiteten Daten imaginäre Objekte erzeugt, sogenannte Abstrakte Visuelle Objekte (AVO). Dabei werden die abgeleiteten Daten auf die Attributfelder abgebildet, die das AVO beschreiben, wobei die Attributfelder neben geometrischen Daten Farbwerte, Transparenz, Oberflächentextur und dergleichen mehr enthalten können. Die Abbildung (Mapping) der abgeleiteten Daten in die AVO-Felder wird von mehr oder weniger komplexen Transferfunktionen übernommen.
  Das Visualization Rendering, kurz Rendering, produziert schließlich auf Basis des AVO das eigentliche darstellbare Bild. Hier kommen die Standardtechniken der Computergraphik zum Einsatz wie z.B. die Translation, Rotation, und Skalierung von Objekten, Clipping, Shading, das Entfernen versteckter Flächen (*hidden surface removal*), Antialiasing, perspektivische Darstellung

und vieles mehr.
Die Vorgehensweise in dieser Arbeit ist im wesentlichen dieselbe wie eben angesprochen, jedoch ist die Aufteilung eine andere (Abbildung 1.1), um der Gesamtkonzeption zu entsprechen, die einen vielseitigen Einsatz, in Ein- und Mehrprozessorsystemen, aber auch in einem verteilten Rechnernetzwerk verlangt.

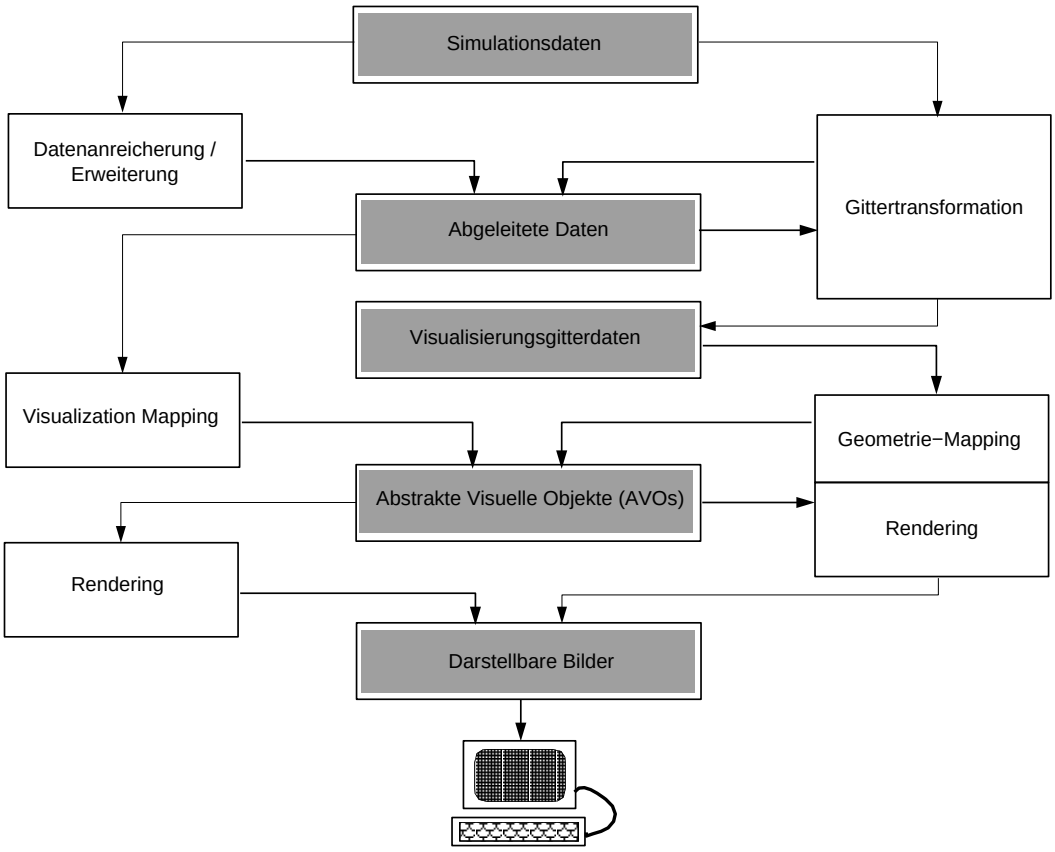


Abb. 1.1: Der Visualisierungsprozeß

Somit werden die Verarbeitungsschritte zwei großen Blöcken zugeordnet, einzelne Schritte gegebenenfalls untergliedert und die Teilschritte verschiedenen Blöcken zugewiesen. Beispielsweise gliedert sich das Mapping einmal in einen Berechnungsteil, der die Daten für die Attributfelder generiert, und zum anderen in die Erzeugung des eigentlichen AVO, bei der weitere Attributfelder hinzukommen können, etwa durch das Color Mapping.

Der Berechnungsteil bildet den Hauptteil des ersten Blocks, in den auch der Schritt der Datenerweiterung fällt. Da dieser Block sich eng an der vorgegebenen Gitterstruktur orientiert und speziell für unstrukturierte Gitter geeignet ist, soll er Gittertransformator genannt werden. Zum zweiten Block gehören das Geometrie–Mapping und das Rendering. Beide Blöcke sollen an dieser Stelle kurz charakterisiert werden.

Gittertransformation
Im ersten, hier "Gittertransformation" genannten Block werden durch unabhängig von einer Graphik– oder Geometriebibliothek arbeitende Berechnungsalgorithmen die für die Visualisierung repräsentativen Gitterdaten (kurz Visualisierungsgitterdaten) erzeugt. Die Berechnungen erfolgen entweder sequentiell oder parallel, weitgehend jedoch in Anlehnung an vorhandene Datenstrukturen aus einem assoziierten FE–Analyseprogrammsystem. Ferner werden die Berechnungen entweder im Postprozessing durchgeführt oder auf Anforderung durch den Benutzer auch innerhalb des Zeitschrittverfahrens einer FE–Analyse. Die Visualisierungsgitterdaten bilden die Basis für den zweiten Schritt, die Erzeugung Abstrakter Visueller Objekte (AVO). Strenggenommen gehört dieser Schritt mit zum Mapping, wurde hier jedoch unter der Bezeichnung Geometrie–Mapping der Darstellung zugeschlagen, womit eine klare Trennungslinie zwischen in parallelisierbaren und nicht parallelisierbaren Graphik– bzw. Geometriebibliotheken enthaltenden Anteilen gezogen ist.

Geometrie–Mapping und Rendering
Die eigentliche Darstellung erfolgt im zweiten Block, dem "Geometrie–Mapping und Rendering", auf der Grundlage der berechneten Gitterdaten. Hier sind Werkzeuge zu entwickeln, welche die Gitterdaten, die in einem festgelegten Schnittstellenformat vorliegen, vor dem eigentlichen Rendern mittels einer Geometriebibliothek in Abstrakte Graphische Objekte fassen (Geometrie–Mapping). Im engeren Sinne sind dies Polygonobjekte, da wie erwähnt eine Gitterstruktur wie in der FE–Analyse im wesentlichen beibehalten wurde. Die Ausgestaltung der Objekte Betreffendes – Objektauswahl, Farbmodelle, Oberflächenausprägungen etc. – wird dabei noch vor dem eigentlichen Rendern interaktiv vom Anwender festlegt.
Der zweite Block wird in die Parallelisierungskonzepte hier nicht mit einbezogen, obwohl zumindest ein verteiltes Geometrie–Mapping und Rendern verschiedener Polygonobjekte auf unterschiedlichen Rechnern heute sehr wohl denkbar ist, sofern die gleichen Geometriebibliotheken dort zur Verfügung stehen [59].

1.2.2 Ziele aus der Perspektive der Visualisierung

Wird die Visualisierung für sich betrachtet, so richten sich die Aufgaben nach der Art und Weise, wie das Visualisierungsprogrammsystem eingesetzt werden soll. Zwei generelle Betriebsweisen sollen unterschieden werden, ein abhängiger und ein unabhängiger Betrieb des Berechnungs- bzw. Mappingteils (Gittertransformator), der jeweils wiederum sequentiell oder parallel ausgeführt werden kann. Um die Vorgaben zu verwirklichen, sind die den Gittertransformator gestaltenden Berechnungsalgorithmen zu entwickeln, die unter dem Gesichtspunkt einer vielseitigen Verwendbarkeit in seriellen, parallelen, gekoppelten und standalone Anwendungen die wesentlichen Visualisierungstechniken herausarbeiten.

- **Gekoppelter Betrieb**
  Im gekoppelten Betrieb wird der Berechnungsteil der Visualisierung, die Gittertransformation, in ein FE–Analyseprogrammsystem integriert. Hier versteht sich die Visualisierung auch als ein Instrument der Überwachung und Steuerung einer laufenden numerischen Simulation, beispielsweise, um jederzeit direkten Einfluß auf Unzulänglichkeiten in der Entwicklung der Lösung nehmen zu können, wie etwa einer zu langsamen Konvergenz oder falsch angenommener Randbedingungen.
- **Standalone Betrieb**
  Im Standalone Betrieb wird die Visualisierung entkoppelt eingesetzt, entweder als Debugging–Instrument zur Verifizierung des Rechengitters vor der Aufnahme einer Simulation (*pre-processing*), oder zur Nach- und Aufbereitung bereits vorliegender Ergebnisse (*post-processing*).
- **Sequentieller Betrieb**
  Obwohl die Visualisierung hier rein sequentiell, – entweder standalone oder integriert in eine sequentielle FE–Analyse, – betrieben wird, können dennoch Berechnung und Darstellung auf verschiedenen Rechnern ablaufen, die in einem lokalen Netzwerk miteinander verbundenen sind.
- **Paralleler Betrieb**
  Sowohl im Standalone–Betrieb als auch im gekoppelten Betrieb wird zur Beschleunigung des gesamten Visualisierungsprozesses die Berechnung des Visualisierungsgitters parallel vorgenommen.
  Im Standalone–Betrieb wird der Parallelrechner aus der seriellen Anwendung heraus initialisiert, empfängt seine Datenbasis über das lokale Netz und übermittelt am Ende als Ergebnis die Visualisierungsgitterdaten zur Objektbildung und zum Rendern an die Graphik–Workstation zurück.
  Im gekoppelten Betrieb hingegen erfolgt die Berechnung auf den Prozessoren

der FE–Analyse. Die zugehörigen Berechnungsalgorithmen werden von der FE–Analyse aufgerufen und auf den aktuell verfügbaren Daten ausgeführt. Das Visualisierungsgitter wird zur Objektbildung und zum Rendern wiederum an eine Graphik–Workstation geschickt.

1.2.3 Eingesetzte Software und Hardware

Es ist nicht die Aufgabe, in Konkurrenz zu einem der kommerziellen Graphikpakete wie z.B. AVS [45], AVS–Express [52], Khoros [49] oder Data–Explorer [46] zu treten, die sich als generelle *application builder* verstehen, sondern vielmehr eine offengelassene Lücke zu schließen, in dem das entstehende Programmsystem enger an den gesamten Ablauf einer numerischen Simulation gekoppelt und vielseitig einsetzbar ist.
Somit kann hier einem Mangel der kommerziellen Produkten begegnet werden, nämlich dem, daß sie für sich stehende, schwer in Gesamtkonzepte integrierbare Programmsysteme sind, die möglichst vielen Anwendern aus vielen Fachgebieten gerecht werden möchten.
Da die Entwicklung eines komplett neuen Visualisierungsprogrammsystems zu umfangreich ist und sich auf die wesentlichen Neuerungen beschränkt werden soll, werden die sequentiellen Programmkomponenten unter die Verwaltung des kommerziellen Visualisierungsprogrammsystems AVS gestellt, in dem sie ein weitgehend eigenständiges Untersystem bilden, das sich der Vorteile des AVS gerade in Hinblick auf ein strukturiertes Modulnetzwerk, visuelles Programmiermodell und verschiedener Rendering–Techniken bedient [6].
Trotz seiner Verknüpfung mit AVS ist das neue Programmsystem jedoch so allgemein auszugelegen, daß Komponenten leicht in andere Systeme übertragbar sind. Graphikspezifische Sprachkonstrukte sollten demzufolge nur in klar abgegrenzten Software–Blöcken enthalten sein, die sich prinzipell durch andere Blöcke ersetzen lassen, die sich auf Standard–Graphikbibliotheken stützen. Ferner ist eine Abgrenzung der Aufgabengebiete vorzunehmen, unter Festlegung aller Schnittstellen gerade auch zu den Bereichen, die durch die Migration in AVS nicht neu entwickelt werden müssen.
Durch Verwendung von Standards soll die zu entwickelnde Visualisierungssoftware bis auf kleineren Portierungsaufwand auf allen UNIX–Rechnern einsetzbar sein. Bei der Entwicklung von Software für den Datenaustausch zwischen vernetzten Rechnern wird daher auf der Interprozeß–Kommunikation (IPC) [54] aufgebaut, die auf sogenanten Sockets basiert, Endpunkten der Kommunikation. Das Software–Paket AVS ist heute auf allen gängigen Plattformen verfügbar. Bei der Programmierung

wird auf ANSI Fortran und C zurückgegriffen.
Die Parallelisierung der Berechnung ist, stellvertretend für MIMD–Rechner, für zwei maßgebliche Parallelrechnerarchitekturen zu realisieren: einen Parsytec–Multi–Cluster MC3 und eine Intel–Paragon.

1.3 Gliederung der vorliegenden Arbeit

In den vorangehenden Abschnitten wurde in die Thematik der numerischen Visualisierung eingeführt und eine Zielsetzung formuliert. Die weiteren Kapitel der Arbeit sind nun folgendermaßen gegliedert:
Kapitel 2 steckt den Rahmen für die Visualisierung betreffend Numerik und Parallelisierung ab und behandelt einige Grundlagen, die mit der Visualisierung in unmittelbarer Beziehung stehen. Abschnitt 2.2 gibt unter diesen Voraussetzungen einen Einblick in den Ablauf einer numerischen Simulation, während Abschnitt 2.3 sich mit den Grundbegriffen der Parallelisierung auseinandersetzt. Der Architektur eines Parallelrechners ist dort ebenso ein Abschnitt eingeräumt wie den häufigsten Modellen paralleler Programmierung und einigen wesentlichen Aspekten der Leistungsmessung bzw. der Bewertung paralleler Pogramme.
Abschnitt 2.4 untersucht, in wie weit Parallelisierungskonzepte bislang in die numerische Visualisierung Eingang gefunden haben.
In Kapitel 3.1 wird der Aufbau des Visualisierungssystems dargelegt und die Systemkomponenten im Überblick vorgestellt. Im einzelnen sind das Gittertransformation, Geometrie–Mapping und Rendering, Datenhandhabung sowie eine Benutzeroberfläche, die alle diese Komponenten integriert. Kapitel 3.2 führt in diesem Kontext in die zugrundeliegende Software des *application builders* AVS [82] ein, dessen Verwaltungsstruktur die sequentiellen Systemteile mehrheitlich untergeordnet sind.

Die folgenden Kapitel vertiefen die Aussagen von Kapitel 3, indem sie die Schritte und Verfahrensweisen bei die Realisierung aufzeigen.
So widmen sich Kapitel 4 und 5 dem Mapping, der Komponente des Gesamtsystems, in der die verschiedenen Methoden zur Erzeugung Abstrakter Visueller Objekte enthalten sind. In Kapitel 4, das einen Schwerpunkt innerhalb der Arbeit bildet, wird zunächst auf die grundlegende Datenstruktur eingegangen, die von der numerischen FE–Analyse übernommen wird und für die Belange der Visualisierung gegebenenfalls um spezielle Behelfslisten zu erweitern ist, je nachdem, welcher Berechnungs- bzw. Mappingalgorithmus zum Einsatz kommt. Auf dieser Basis werden in den weiteren Abschnitten dieses Kapitels die sequentiellen Berechnungsalgorithmen erarbeitet und Charakteristisches an Hand von Beispielen aus der numerischen Strömungsme-

chanik illustriert.
In Kapitel 5 steht die Parallelisierung des Berechnungs- und Mappingschrittes (Gittertransformator) an. Der parallele Programmaufbau wird erläutert und auf die einzelnen Phasen des Programmablaufs eingegangen. Neben der rechnerabhängigen Initialisierung kommen auch die Bereitstellung der Datenbasis, Werkzeuge für interne und externe Kommunikation, die Ausgabe der Ergebnisdaten und die parallelen Berechnungsalgorithmen zur Sprache.

Mit dem Datenaustausch zwischen den einzelnen Systemkomponenten, die auf verschiedenen Rechnern aktiv sein können, und der interaktiven sowie halbautomatischen Datenflußsteuerung ist Kapitel 6 befaßt. Abschnitt 6.1 führt in diesem Zusammenhang in die auf Sockets basierende Inter–Prozeß–Kommunikation IPC [54] ein, welche der in Abschnitt 6.2 vorgestellten Transmitter–Technik für eine Punkt–zu–Punkt–Kommunikation die Grundlage liefert. Abschnitt 6.3 präsentiert exemplarisch einige typische Anwendungen, bei denen die Verteilung der Aufgaben und der Datenaustausch über Transmitter erfolgt.
In Kapitel 7 schließlich werden serielle und parallele Anwendungen exemplarisch vorgestellt, Möglichkeiten der Einflußnahme auf den Ablauf erörtert und die Leistungsfähigkeit der hier eingesetzten Parallelprogramme bewertet. Kapitel 8 faßt noch einmal zusammen.

Kapitel 2

Ausgangsbasis für die Visualisierung

Die Visualisierung ist nicht nur in erster Linie der Versuch einer Nachbildung bzw. - in der wörtlichen Bedeutung des Ausdrucks - Abbildung der Natur in ein sinngewohntes Bild, sondern sie ist überdies ein unverzichtbares Hilfsmittel, sehr Abstraktes dort zu veranschaulichen, wo das menschliche Vorstellungsvermögen im allgemeinen überfordert ist. Das ist beispielsweise der Fall, wenn es sich um reine Abbilder mathematischer Sachverhalte wie Funktions- oder Tensorräume handelt, oder um die Darstellung künstlicher Größen wie etwa zeitliche Änderungen von Iterationsfehlern bei einer numerischen Simulation.
Mathematisch abstrakte Zusammenhänge werden dabei auf ein Bild reduziert bzw. in eine bildliche Form gebracht, die dem Vorstellungsvermögen des Betrachters bekannt erscheint und somit sein Verständnis für den Sachverhalt erschließt.

2.1 Eingrenzung der Aufgaben der Visualisierung

Computergraphik und Visualisierung bilden heute ein weites informationswissenschaftliches Feld, das zahlreiche Nischen und Unterbereiche beherbergt. Stellvertretend für alle anderen seien nur die verschiedenen Verfahren für photorealistische Computergraphik wie Ray-Tracing-Techniken, Forward- und Backward-Mapping angeführt, Verfahren zur Volumenvisualisierung oder seit einiger Zeit virtuelle Realitäten als wegeweisendes Gebiet.
Die Komplexität der wenigen genannten Sparten zeigt die Notwendigkeit einer Ein-

grenzung hinsichtlich der Ansiedlung dieser Arbeit, was die Visualisierung hier zu leisten vermag und welche Bereiche außen vorgelassen werden müssen.

Die Visualisierung, wie sie sich in dieser Arbeit darstellt, lehnt sich eng an die Verhältnisse numerischer Simulationsverfahren an, d.h. durch deren Gegebenheiten und Vorgaben liegen die Aufgabenbereiche und Entwicklungsgebiete im Prinzip fest. Oberstes Ziel der Visualisierung bleibt es, dem Anwender möglichst rasch Aufschluß über die enormen Datenmengen zu geben, die bei einer numerischen Simulation anfallen, und das nicht bloß in der Form, in der sie erzeugt wurden, sondern so aufbereitet, wie sie für eine Interpretation jeweils am besten zuträglich sind. Dabei ist es nicht selten der Fall, daß gerade mehrere Techniken nebeneinander gezeigt ein physikalisches Phänomen erst aufzulösen vermögen.
Die Visualisierung setzt also auf der Datenbasis (Finite–Elemente–Gitter) auf, wie sie die numerische Analyse liefert, und löst ihre spezifischen Aufgaben auf der vorgegebenen Datenstruktur. So lag denn ein Schwerpunkt auf der Entwicklung serieller und paralleler numerischer Algorithmen zur Berechnung des für die Visualisierung maßgeblichen Gitters, aber auch auf der Erstellung von Software sowohl für die Erzeugung von Abstrakten Visuellen Objekten (AVO) als auch für einen effizienten Datenaustausch zwischen allen beteiligten Komponenten.
Dabei liegt jedem AVO ein berechnetes Visualisierungsgitter zugrunde. Die AVOs werden mit Hilfe spezieller Graphikbibliotheksfunktionen generiert, die das Application Visualization System AVS bereithält. Prinzipiell können diese Funktionen auch durch die anderer Graphikbibliotheken ersetzt werden.
Da nicht in Konkurrenz zu einem kommerziellen Visualisierungsprogrammsystem getreten, sondern vielmehr auf die Vorteile und Techniken, die ein solches bietet, unmittelbar zurückgegriffen werden sollte, wurde das System in ein anderes, in diesem Fall das AVS, eingebunden und sich auf die wesentlichen Neuerungen konzentriert. Andere Aufgaben wie z.B. der Aufbau von Windows und Viewports, das Rendern aber auch die Bereitstellung der Grundstrukturen für eine Ereignissteuerung (*event handling*) und dergleichen mehr konnten so dem übergeordneten System überlassen werden. Selbstverständlich wurden jedoch verschiedene Schnittstellen geschaffen bzw. die notwendigen Anpassungen vorgenommen, um ebendies zu ermöglichen, aber auch, um sich eine gewisse Eigenständigkeit und Abgeschlossenheit zu bewahren. In dieser Hinsicht stellt das hier entwickelte System eine Erweiterung zu AVS dar und ist in diesem Rahmen portierbar und nutzbar.

2.2 Die numerische Simulation

An dieser Stelle soll die numerische Simulation kurz beleuchtet werden, weil sie zum einen der Visualisierung die Datengrundlage liefert und zum anderen mit Bereichen der Visualisierung gekoppelt betrieben werden kann. Stellvertretend für alle anderen Fachgebiete, in denen in ähnlicher Weise Simulationen durchgeführt werden, wurde die Strömungsmechanik CFD (*Computational Fluid Dynamics*) gewählt, in dem das hier entwickelte Visualisierungssystem vorzugsweise eingesetzt wird.
Bei der numerischen Behandlung eines Strömungsproblems wird im Prinzip stets nach demselben Schema verfahren, dessen Schritte hier grob skizziert werden.

Problemstellung
Ausgangspunkt ist die Beschreibung eines physikalischen Problems, wie beispielsweise eines Verbrennungsvorgangs in einem Gasbrenner, durch ein mathematisches Modell, durch das dieses, je nach Komplexität der Modellierung, mehr oder weniger exakt charakterisiert ist. Das mathematische Modell, das sich durch ein System linearer oder häufiger nichtlinearer partieller Differentialgleichungen darstellt, ist bislang nur in wenigen einfachen Fällen analytisch lösbar, weshalb sich numerisch durch eine Diskretisierung beholfen werden muß.

Diskretisierung
Bei der Diskretisierung des mathematischen Problems wird zum einen die Diskretisierung des Berechnungsgebietes unterschieden und zum anderen die Diskretisierung der Gleichungen. Im ersteren Fall wird ein Gitter erzeugt, welches das Kontinuum durch eine begrenzte Anzahl von Gitter- bzw. Knotenpunkten ersetzt. An diesen Knotenpunkten werden die Lösungen der linearisierten Gleichungen berechnet, die durch Diskretisierung des Systems partieller Differentialgleichungen erhalten werden, bei der dieses durch ein geeignetes Verfahren auf ein nichtlineares bzw. lineares algebraisches Gleichungssystem zurückzuführen ist. Ein nichtlineares algebraisches Gleichungssystem kann beispielsweise mit Hilfe des Newtonverfahrens [10] linearisiert werden. Die wichtigsten Diskretisierungsverfahren sind

- die Finite Differenzen Methode (FD),
- die Finite Element Methode (FE) und
- die Finite Volumen Methode (FV).

Während die Finite Differenzen Methode bis Ende der Achtzigerjahre die größte Verbreitung genoß, wird heute mehr und mehr den beiden anderen Methoden der Vorzug gegeben, nicht zuletzt dadurch, daß komplexere Geometrien nicht akademischer Probleme wesentlich flexibler gehandhabt werden können.

Das der Visualisierung zugrundeliegende Programmsystem FEPS [84] basiert auf der Methode der Finiten Elemente, weswegen die weiteren Ausführungen diesem Ansatz folgen werden.

Randbedingungen
Ein wesentlicher Punkt bei der mathematischen Modellierung ist die Festlegung der physikalischen wie numerischen Randbedingungen, durch die ein Problem erst richtig definiert ist, bzw. bei deren korrekter Wahl sinnvolle Ergebnisse erst zu erwarten sein werden.
Als Randbedingung einer Strömung kommen z.B. Dirichlet'sche, Neumann'sche oder gemischte Randbedingungen in Frage. Bei Dirichlet bedeutet das die feste Vorgabe eines Wertes z.B. die Temperatur an einer Wand, bei Neumann die feste Vorgabe eines Gradienten z.B. die Ausströmrandbedingung. Diesen problemabhängigen Bedingungen wird bei der Erstellung der Topologie durch die Wahl geeigneter Randelemente Rechnung getragen.

Erzeugung des Gitters und der Topologie
Bei der Gittergenerierung werden zum einen alle Gitterknoten erzeugt und zum anderen die Elemente, die diese Knoten als Eckpunkte haben. Bei der Generierung muß unter anderem darauf geachtet werden, daß keine hängenden Knoten entstehen, das also die Elemente über gemeinsame Knoten lückenlos miteinander verbunden sind, sprich die Konnektivität des Rechengebietes nicht verletzt ist. Die der Visualisierung zugrundeliegenden Gitter wurden alle entweder mit dem Gittergenerator in PATRAN [19] oder dem parallelen Gittergenerator PAGE [66, 17, 5] erzeugt.
Form und Typ der Finiten Elemente oder Finiten Volumen werden entsprechend den Anforderungen des Problems festgelegt. Typ und Form definieren z.B. die Freiheitsgrade, die Art ihrer Diskretisierung des Strömungsfeldes (Ansatzfunktionen), im Falle eines Randelements die Randbedingungen und einiges mehr.

Lösungsverfahren
Wurden die Gleichungen diskretisiert und die Randbedingungen spezifiziert, können konkrete Probleme gerechnet werden, indem das algebraische Gleichungssystem auf dem Gitter gelöst wird. Allerdings muß, um daraus eine konkrete Situation zu simulieren, die Situation noch auf das Modell abgebildet werden. Das geschieht über Daten, die das Lösungsgebiet und seine Ränder im Sinne des diskretisierten Modells beschreiben.
Zur Herleitung der Gleichungen werden als physikalische Grundlagen die Erhaltungssätze von Masse, Impuls und Energie herangezogen. Das allgemeingültige mathematische Modell zur Beschreibung eines strömenden Mediums stellen die Navier-Stokes Gleichungen dar. Die Eulergleichungen werden aus der allgemeingültigen Form erhalten, wenn Reibungskräfte und Wärmeleitungen unberücksichtigt bleiben,

d.h. ihre Therme aus den Gleichungen gestrichen werden. Eine gute Einführung in die Problematik kann bei [2, 71, 35] nachgelesen werden.
Die Zahl der Unbekannten in einem zu lösenden linearisierten Gleichungssystem geht heute schon in die Millionen, weshalb spezielle Beschleunigungstechniken entwickelt wurden, um der Komplexität der Probleme Herr zu werden. Parallelisierung, adaptive Verfahren und schnelle Löser heißen hier die Schlagwörter.
Bei der Parallelisierung ist man bemüht, die Rechenlast möglichst gleichmäßig auf verschiedene Rechner oder Rechenknoten umzuverteilen und gleichzeitig den Kommunikationsbedarf zwischen ihnen zu minimieren, der notwendig ist, weil die Teilgebiete untereinander gekoppelt sind, und die Beiträge der durch die Gebietsaufteilung enstandenen inneren Ränder aufeinander abzustimmen sind. Anschaulich heißt das, die Strömung wird über die inneren Gebietsränder transportiert.
Eine gleichmäßige Lastverteilung wird im günstigsten Fall durch eine Aufteilung des Strömungsgebietes in gleichgroße zusammenhängende Untergebiete (*subdomains*) erreicht.
Ein Lastenungleichgewicht entsteht z.B. dann, wenn die Gitter während der Rechnung fortwährend angepaßt werden, d.h. lokal eine Verfeinerung oder Vergröberung je nach Qualität der örtlichen Lösung vorgenommen wird. Hierbei ist der geschätzte Fehler der Lösung das Kriterium, nach dem in einem adaptiven Verfahren entweder Unbekannte eingebracht oder herausgenommen werden [8].
Eine Konvergenzbeschleunigung der Lösung kann beispielsweise durch Einsatz eines Mehrgitterverfahrens erzielt werden, bei dem die Fehleranteile verschiedener Frequenzen auf den entsprechenden feineren oder gröberen Gittern aufgelöst bzw. herausgefiltert werden [31].
Letztendlich gibt es viele verschiedene Strategien und Verfahren, die alle versuchen, ein Optimum an Zeit und Speicherersparnis, ohne wesentliche Einbußen in der Qualität der Lösung herauszuwirtschaften.

Datenausgabe
Am Ende der numerischen Simulation wird die Auswertung der Ergebnisse vorgenommen. Dabei werden alle wesentlichen Strömungsgrößen aus dem Erhaltungsvektor entweder direkt erhalten oder indirekt berechnet. Auf diesen Daten setzt die nachbereitende Visualisierung auf. Eine Visualisierung während des Laufs der Analyse hingegen kann, wie in dieser Arbeit gezeigt wird, zum Beispiel durch die Integration von Auswerteprozedur und Berechnung des Visualisierungsgitters in das Iterationsverfahren realisiert werden, unter Einbezug diverser Datenhandhabungs- und Steuermechanismen [7].
Ein Beispiel für ein vollständiges numerisches adaptives Verfahren ist in Abbildung 2.1 zu sehen.

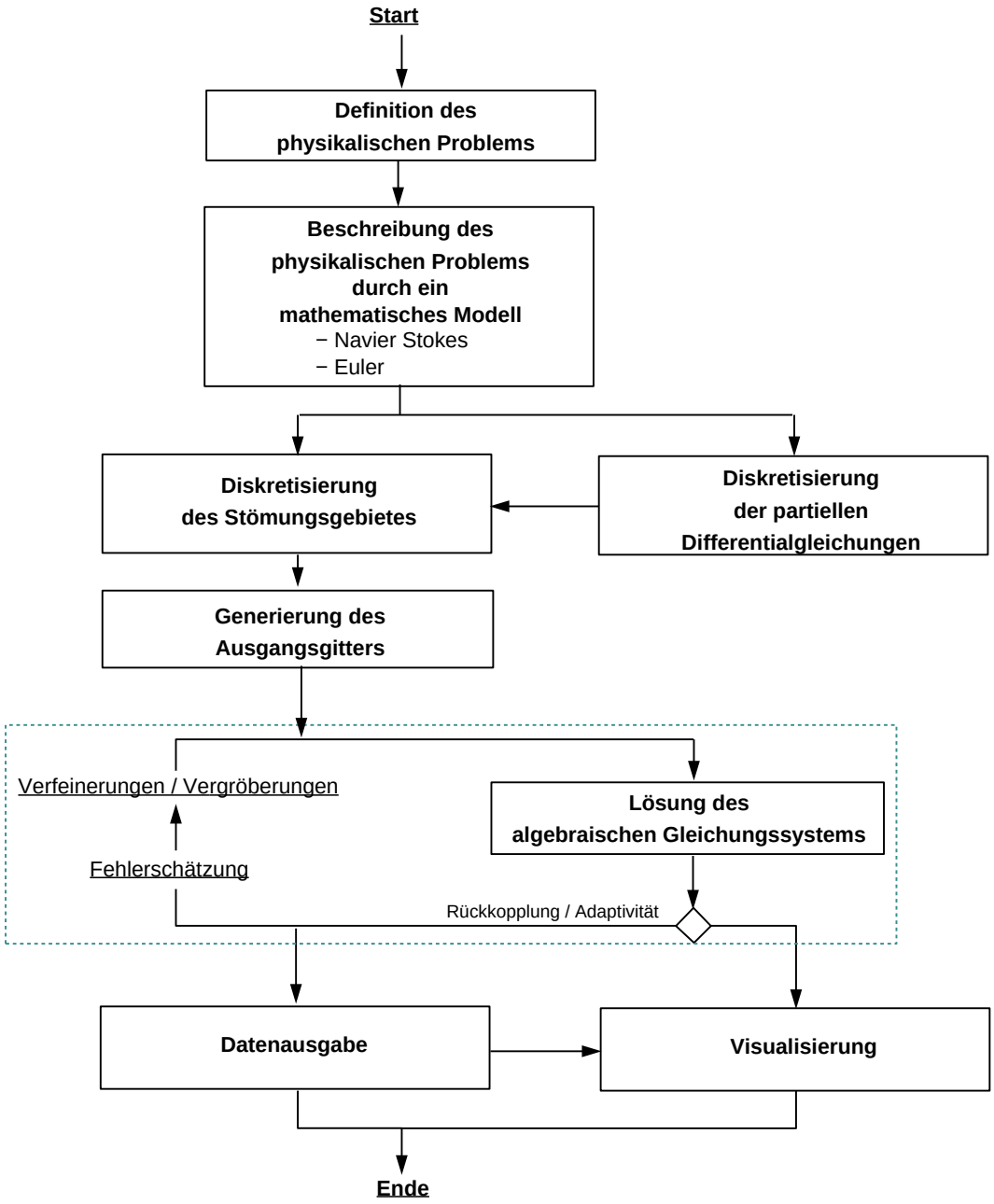


Abb. 2.1: Ablaufschema eines adaptiven Verfahrens

2.3 Grundlagen der Parallelisierung

Vorbemerkungen

Die gewaltigen Fortschritte, die bei numerischen Simulationsverfahren gerade in den vergangenen Jahren erzielt wurden, haben neue parallele Algorithmen und Programiermodelle hervorgebracht und einen Engpass bei der Visualisierung bewirkt; einerseits, weil vor kurzem noch undenkbare Problemgrößen mit mehreren 10 Millionen Gitterpunkten jetzt parallel behandelt werden können, und es dementsprechend ein zigfaches dieser Menge an Ergebnisdaten auszuwerten gilt, und andererseits, weil es enorme Kosten verursacht, geeignete Einprozessormaschinen, respektive Workstations, mit schnellen Graphikprozessoren zu beschaffen, die solche Datenmengen überhaupt zu verarbeiten in der Lage sind.

Ein Nachziehen der Visualisierung in Sachen Parallelisierung ist also erforderlich. Es ist jetzt schon absehbar, daß am Ende der Entwicklungen eine vollständig parallelisierte Visualisierung stehen wird, die ein paralleles Mappen und Rendern vermittels paralleler Graphikbibliotheksroutinen einbegreift. Da heute jedoch Visualisierungssoftware erst ansatzweise den Parallelprozessoren direkt zur Verfügung steht, was zum Zeitpunkt der Aufnahme der vorliegenden Arbeit Anfang 1990 keineswegs der Fall war, ist es naheliegend, zumindest die aufwendige Berechnung der für die Visualisierung repräsentrativen Gitterdaten und die mit dieser Gittertransformation zumeist einhergehende Datenreduktion parallel vorzunehmen. Hier kann sich zunutze gemacht werden, daß, wie das in der Einleitung schon angesprochen wurde, eine klare Trennlinie zwischen Gittertransformation hier und Mapping und Rendering dort gezogen werden kann.

Über parallele Visualisierung und parallele Anwendungen wird in den Kapiteln 5 bis 7 zu berichten sein, während in den verbleibenden Abschnitten dieses Kapitels ein kurzer Streifzug in die Thematik des parallelen Rechnens unternommen wird, der in diesem groben Rahmen oberflächlich und unvollständig bleiben darf. Als weiterführende Literatur zur Thematik parallelen Rechnens sei das Werk von Heiss [32] empfohlen.

2.3.1 Parallelrechnerarchitekturen

2.3.1.1 Klassifizierungen

Es gibt verschiedene Möglichkeiten, Parallelrechner hinsichtlich ihres Aufbaus einzuteilen. Die wohl bekannteste Klassifikation, aus der sich gut der historische Werdegang moderner Parallelcomputer studieren läßt und die auch weiterhin am Ver-

breitesten ist, wurde schon 1972 von Flynn [21] vorgeschlagen.
Flynn befaßte sich mit der Organisation und der Effektivität von Computern und stellte fest, daß bei Rechenabläufen stets Folgen von Operationen auf Folgen von Daten angewendet werden. Danach unterschied er Parallelität nach Einfachheit oder Vielfachheit von Befehlsströmen und Datenströmen, woraus sich theoretisch die vier Kombinationsmöglichkeiten SISD, MISD, SIMD und MIMD ergaben, die in Abbildung 2.2 abgebildet sind.

		Datenströme	
		singulär	multipel
Instruktions–ströme	singulär	**SISD** von–Neumann–Rechner	**SIMD** Vektor–, Feldrechner
	multipel	**MISD** unbesetzt	**MIMD** Multiprozessorsysteme Verteilte Systeme

Abb. 2.2: Die Grobklassifikation nach Flynn

Unter die SISD-Klasse fallen die konventionellen von-Neumann-Rechner, bei denen ein einzelner Prozessor eine Folge von Befehlen auf einem Satz Datenelemente abarbeitet. Die MISD-Klasse ist bisher unbesetzt geblieben [56], weshalb für die Parallelrechner nur die Klassen SIMD und MIMD verbleiben.

SIMD-Rechner:
Beim SIMD-Prinzip wird auf einer Vielzahl von Daten jeweils genau eine Operation ausgeführt. Kontrollorgan ist hier ein zentraler Steuerprozessor, der den verschiedenen Arbeitsprozessoren jeweils identische Befehle zur synchronen Abarbeitung schickt. Da genaugenommen nur ein Programm ausgeführt wird, ähnelt das SIMD-Prinzip noch deutlich dem der konventionellen, sequentiellen Rechner.

MIMD-Rechner:
Bei einem MIMD-Rechner erhält jeder Prozessor sein eigenes Programm, das er auf seinen spezifischen Datenelementen ausführt. Die Prozessoren können also völlig unabhängig voneinander programmiert werden. Die große Mehrheit der heute verfügbaren Parallelrechner fällt unter diese Klasse. Das Prinzip ist so allgemein formuliert, daß beispielsweise ein Verbund beliebiger Rechner, wenn man ihn als Einheit betrachtet, einen MIMD-Rechner darstellt.
Da diese Klassifikation heute nur noch bedingt tauglich ist, bedurfte es weiterer Ordnungsmerkmale.

Einteilung nach der Organisation des Speichers:
Flynns einfache Taxonomie ist auf verschiedene Arten erweitert worden. Häufig werden Parallelrechner daraufhin klassifiziert, wie sie ihren Speicher organisieren; ob die Speichermodule den Prozessoren individuell oder in ihrer Gesamtheit zugeordnet sind. Im ersteren Fall spricht man von Multirechnersystemen. Zu jedem Prozessor gehört hier ein privates Speichermodul, auf das nur er allein zugreifen kann. Typische Vertreter dieser Kategorie sind Message-Passing-Systeme und Lose-Gekoppelte-Systeme. Der Datenaustausch mit den anderen Prozessoren findet über das Verbindungsnetzwerk statt.
Unter Multiprozessorsysteme fallen Rechner, deren Prozessoren über ein gemeinsames Kommunikationsnetz auf einen gemeinsamen Hauptspeicher zugreifen. Zu den Vertretern dieser Kategorie gehören Shared-Memory-Systeme und Fest-Gekoppelte-Systeme. Ein Vorteil dieser Systeme ist, daß Programme, die auf mehreren Prozessoren simultan abgearbeitet werden, sehr effizient kommunizieren können, da sie nur einmal im Speicher gehalten werden müssen. Außerdem dürfen die Prozessoren kleiner und zahlreicher sein und sind entschieden billiger in der Herstellung.
Je nach Art der Erreichbarkeit der Speichermodule kann man Parallelrechner weiter nach einheitlichem oder uneinheitlichem Speicherzugriff aufschlüsseln. Man spricht von UMA- und NUMA-Architekturen nach *uniform* bzw. *nonuniform memory access*. Im Gegensatz zu der NUMA-Gruppe, ist bei UMA-Rechnern die Zugriffszeit auf ein Speichermodul von der Adresse des Speichermoduls oder des Prozessors unabhängig. Überdies existieren Mischformen, also Rechner mit sowohl verteiltem als auch gemeinsamem Hauptspeicher, die sich je nach Anteil beider Speicherarten unterscheiden.
Ein Nachteil von Rechnern mit gemeinsamen Hauptspeicher sind sicherlich Zugriffskonflikte, die entstehen, wenn mehrere Prozessoren gleichzeitig ein Datum beschreiben wollen. Dieses Problem kann nur seriell angegangen werden, wodurch sich ein zeitlicher Überhang (*overhead*) einstellt, der mit der Zahl der Prozessoren überproportional anwächst. Der Skalierbarkeit dieser Systeme sind daher enge Grenzen gesetzt, so daß ohne wesentliche Eingriffe in die Hardware Prozessoranzahlen größer 32 wenig sinnvoll erscheinen.
Da Multirechnersysteme demgegenüber keine Speicherkonflikte kennen, bestimmt bei ihnen allein der Kommunikationsaufwand die Grenze der Skalierbarkeit. Es gilt also, die Kommunikationsstruktur eines Parallelprogramms optimal an das Verbindungsnetz anzupassen, was gelingt, wenn die Daten so aufgeteilt werden, daß Zugriffe überwiegend lokal erfolgen, d.h. möglichst wenig über Prozessorgrenzen hinweg kommuniziert werden muß.

2.3.1.2 Verbindungsnetze

Sowohl bei Multirechnern als auch bei Multiprozessoren ist die Art der Verbindungsnetze von entscheidender Bedeutung für eine Bewertung des Parallelrechners. Neben der Leistung, die sich durch die Menge der Daten ausdrückt und die Geschwindigkeit, mit der übertragen werden kann, ist vorallem die Erweiterbarkeit des Systems, beispielsweise durch die Möglichkeit einer inkrementellen Hinzunahme von Prozessoren, ein Kriterium nach dem beurteilt werden muß; fernerhin ausschlaggebend sind die Funktionalität, die Zuverlässigkeit wie etwa die Ausfalltoleranz und letztendlich die Kosten, die mit der Leistung einhergehen.
Grundsätzlich können statische und dynamische Verbindungsnetze unterschieden werden.

Statische Verbindungsnetze
Statische Netze oder Punkt-zu-Punkt-Netze sind für Multirechnersysteme typisch. Da alle Verbindungen zwischen Prozessoren und Speicher unveränderbar festliegen, ist die Topologie für die Qualität des Netzes maßgeblich. Ein erster Anhaltspunkt für die Bewertung einer Topologie ist der Knotengrad d.h. die Anzahl der direkten Nachbarn eines Knotens. Es gilt, diesen möglichst konstant und niedrig zu halten, der Kosten für die Hardware wegen.
Typische Vertreter statischer Netze sind Ring, Binärbaum, Hyperwürfel und Gitternetz. An ihnen lassen sich sehr anschaulich die Grenzen der Zuverlässigkeit und Skalierbarkeit studieren, und das an Hand weiterer Kriterien für eine Topologie wie Durchmesser und Konnektivität. Der Duchmesser bezeichnet die maximale Weglänge zwischen zwei Knoten eines Netzes und ist ein Maß für Verzögerungen, gerade wenn man beliebige Knoten des Netzes miteinander kommunizieren läßt. Bei einer Ringtopologie z.B. wächst der Durchmesser linear mit der Zahl der Prozessoren. Im selben Maße steigt die Ausfallwahrscheinlichkeit.
Die Konnektivität gibt das Verhältnis von Kanten zu Knoten wieder; günstig sind auch hier niedrige Werte. Um die Zuverlässigkeit zu steigern und sich gegen Ausfall zu wappnen, werden im allgemeinen zusätzliche Verbindungen eingeführt. Einer vollständigen Vermaschung mit einer Ausfallwahrscheinlichkeit gegen Null stehen die enormen Kosten im Wege und Grenzen in der Skalierbarkeit. Es gilt also, durch wenige gezielte Verbindungen die Ausfalltoleranz zu erhöhen bei möglichst konstant niedrigem Knotengrad. Bei einem Binärbaum wird dies durch zusätzliches Verbinden aller Knoten einer Stufe miteinander erreicht (X-Baum).
Als ein weiteres Bewertungsmerkmal seien abschließend die parallelen Programme selbst genannt. Durch eine Ausrichtung der Kommunikationsstruktur der Algorithmen an der Verbindungsstruktur des Rechners nämlich kann hier eine optimale Leistung erzielt werden. Als Beispiel seien nur Gitter- und Torus-Topologie angeführt,

auf welche sich viele aktuelle Probleme abbilden lassen. Eine aufwendige numerische Behandlung von Differentialgleichungen mit unzähligen Matrizenoperationen ist hier besonders geeignet.

Dynamische Verbindungsnetze

Dynamische Verbindungsnetze kommen vorwiegend in Multiprozessorsystemen zur Anwendung. Hier besitzt jeder Prozessor Verbindungen zum Netz, die je nach Bedarf geschaltet werden können. Da jeder Prozessor dynamisch mit jedem verbunden werden kann, bleiben Konflikte nicht aus, z.B. wenn ein angesprochener Prozessor schon eine andere Verbindung auf dem selben Kanal handhabt. Die Güte des Netzes hängt also wesentlich von der Art der Schaltung ab bzw. unter welchem Aufwand es gelingt, Zugriffskonflikte und somit Ausfälle zu mininieren. Danach können drei Hauptklassen unterschieden werden, als da sind: Busartige Netze, Schaltermatrizen und mehrstufige Netzwerke.

Busse sind am einfachsten und am preiswertesten, bieten jedoch keinen Schutz vor Ausfällen, was ihren Einsatz in Netzen mit mehr als 100 Prozessoren fragwürdig macht. Eine Verwendung mehrerer paralleler Busse (Multibus) verbessert hier die Situation.

Eine Alternative ist die Schaltermatrix, über die im Prinzip jeder Prozessor mit einem anderen verbunden werden kann, sofern dieser nicht gerade belegt ist. Der Vorteil der Schaltermatrix ist, daß sie ohne Blockierung arbeitet und in ihrer Leistung skalierbar ist. Allerdings wachsen mit der Anzahl der zu verschaltenden Komponenten Kosten und Aufwand quadratisch an, weswegen es sich anbietet, mehrere kleinere Matrizen miteinander hierarchisch zu koppeln.

Das führt direkt zu den mehrstufigen Netzwerken, bei denen viele 2 x 2-Schaltermatrizen als Basiselemente in mehreren Stufen hintereinander geschaltet werden, so daß eine matrixförmige Anordnung entsteht. Aufwand und Kosten reduzieren sich hierbei von $n * n$ auf $n * log\ n$, während sich die Übertragungszeiten jedoch erhöhen. Die einzelnen Basiselemente kennen zwei bis vier Schaltzustände, was komplizierte Vernetzungen ermöglicht. Geschicktes Einbringen zusätzlicher Elemente erhöht die Pfadredundanz bei ungünstigen Konstellationen und senkt somit die Wahrscheinlichkeit, daß Blockierungen auftreten. Die wohl bekanntesten Vertreter dieser Netzwerke sind Omega-Netze, Banyan- und Waksman-Netze.

2.3.2 Modelle paralleler Programmierung

Bei der parallelen Programmierung geht es in erster Linie darum, Modelle für die Ausschöpfung der theoretischen Leistungsfähigkeit eines Parallelrechners zu entwickeln und allgemeingültige Kriterien zu finden, nach denen diese in die Programmierpraxis umzusetzen sind. Erreicht wird dieses prinzipell durch eine Abbildung des parallelen Programms auf die parallele Maschine, wobei den durch die jeweilige parallele Hardware bedingten Vorgaben Rechnung zu tragen ist, und das bei möglichst geringem zusätzlichen Programmieraufwand.
Zunächst einmal drängt es sich auf, die Flynnsche Klassifizierung der Hardware auch für die Programmiermodelle zu übernehmen, also SIMD- und MIMD-Modelle zu unterscheiden. Da nicht von Instruktionen sondern von Programmen die Rede ist, wurden die Begriffe SPMD und MPMD geprägt.

SPMD-Modelle
SPMD-Modelle haben zumindest in der Numerik keine weitreichende Verbreitung gefunden, weil sie wenig programmstrukturelle Entwicklungsmöglichkeiten bieten. Ihre Parallelität beschränkt sich darauf, auf jedem der Parallelprozessoren synchron ein und dasselbe Programm auszuführen, wobei jeder Prozessor seine privaten Daten zugewiesen bekommt. Man nennt dieses Datenparallelismus. Solche Modelle lassen sich entsprechend einfach unter Berücksichtigung einiger Regeln, die denen der Vektorisierung nicht unähnlich sind, programmieren. Die Effizienz einer Anwendung hängt zum einen ganz wesentlich von der Gleichmäßigkeit ab, mit der sich die Daten auf die Prozessoren umverteilen lassen (*load balancing*) und zum anderen unmittelbar von der Art und Größe des zu bearbeitenden Problems.

MPMD-Modelle
Ganz anders verhält es sich da mit den MPMD-Programmiermodellen. Hier bearbeiten verschiedene Algorithmen bzw. Programme zur gleichen Zeit gemeinsam eine Problemstellung. Die gängigen Begriffe algorithmischer oder Kontrollflußparallelismus für diese Art der Programmierung leiten sich daraus ab.
Klassisches Beispiel für die Verwendung eines reinen MPMD-Modells ist sicherlich die fließbandartige Bearbeitung einer Folge unterschiedlicher Arbeitsschritte durch veschiedene Prozessoren. In der Praxis jedoch ist man zumeist mit Gemischttypen konfrontiert, also Kopplungen algorithmisch-paralleler mit datenparallelen Aufgaben. Lösungen sind dort ausschließlich mit MIMD-Rechnern zu erhalten. Dies ist mit ein Grund, warum im Gegensatz zu SPMD-Modellen eine Effizenz bei MPMD-Modellen nicht so einfach bewertbar ist. Faktoren wie die Art des zu lösenden Problems und der verwendete Algorithmus spielen hier zudem stark mit hinein. Außerdem ist von vornherein die Organisation des Speichers bei der Erstellung eines Mo-

dells zu berücksichtigen, woraus sich analog zur Klassifizierung der MIMD-Hardware in ihren Grundzügen verschiedene Modelle ergeben, Shared-Memory- und Message-Passing-Modelle, für die im wesentlichen das unter 2.3.1.1 gesagte zutrifft.

2.3.3 Aspekte der Leistungsmessung

Die Leistung eines Parallelrechners kann hinreichend mit den Begriffen Anzahl der Prozessoren, verfügbarer Speicherplatz und Kommunikationsbandbreite umrissen werden. Die erzielbare Leistung einer parallelen Anwendung hängt jedoch nicht nur von der Leistung des Parallelrechners ab sondern natürlich auch von der Fähigkeit des Programms, die theoretische Leistungsfähigkeit des Rechners auszunutzen. Dies ist bestimmt durch den Parallelisierungsgrad des Programms, durch Wartezeiten und den Kommunikationsaufwand, der mit der Anzahl der Prozessoren steigt.

Bei der Bewertung paralleler Programmen und Algorithmen werden im allgemeinen die Benchmarks Speed–up, Effizienz und Scale–up herangezogen. Oft werden Benchmark-Messungen jedoch nicht auf das Gesamtproblem angewendet, sondern bleiben auf Teilprobleme beschränkt. So wird zum Beispiel gerne die aufwendige Generierung von parallelen Gitternetzen bei der Bewertung außer acht gelassen, wodurch das Gesamtergebnis schöngefärbt wird.
Im Folgenden werden kurz die wichtigsten Begriffe der Leistungsmessung definiert und erläutert: Zur Berechnung des Geschwindigkeitsfaktors (Speed–up) eines parallelen Algorithmus dient als Vergleichsmaß die Zeit t_1, die ein serieller Algorithmus, der ohne Kommunikation auskommt, auf einem Parallelprozessor zur Ausführung benötigt. Entsprechend wird mit t_p die Ausführungszeit eines parallelen Algorithmus unter Verwendung von n_p Prozessoren gemessen, der die gleichen Ergebnisse liefert wie der serielle Algorithmus. Der Speed-up S ist nun ein Maß für die Beschleunigung und ergibt sich zu

$$S(n_p) := t_1/t_p \tag{2.1}$$

Der theoretisch maximal erzielbare Speed–up ist demnach gleich der Anzahl der Prozessoren. Durch Warte- und Kommunikationszeiten sowie serielle Anteile des Algorithmus bleibt dieser Wert theoretisch. Die Effizienz relativiert den Speed–up auf die Anzahl der verwendeten Prozessoren und ist ein Maß für die Ausnutzung Parallelrechners. Die Effizienz E ist also der normierte Geschwindigkeitsfaktor im Intervall $[0, 1]$.

$$E(n_p) := S(n_p)/n_p \tag{2.2}$$

Ein weiteres Maß für die Qualität eines Parallelprogramms stellt der Problemgrößenfaktor (Scale–up) dar. Hier wird nicht von einem Problem konstanter Größe ausgegangen, das in mit wachsender Anzahl von Prozessoren in immer kürzerer Zeit gelöst wird, sondern es wird eine konstante Zeit vorgegeben und die Problemgröße entsprechend skaliert bzw. angepaßt. Die Fragestellung lautet hier: Um wieviel größer darf mein Problem werden, wenn mir für dieselbe Rechenzeit mehr Prozessoren zur Verfügung stehen? Setzt man die für n_p Prozessoren erreichbare Problemgröße $D(n_p)$ in Beziehung zu der auf einem Einprozessorsystems $D(1)$ erzielbaren, so gewinnt man den Scale–up aus folgender Beziehung:

$$Sc(n_p) := \frac{D(n_p)}{D(1)} \tag{2.3}$$

In der Praxis ist auch dieser Wert unrealistisch, weil die Probleme nicht beliebig skalierbar sind. Ferner kommt hinzu, daß ein paralleler Programmcode sowohl serielle wie auch parallele Anteile aufweist. Zu den seriellen Anteilen werden auch Kommunikationszeiten gerechnet und Wartezeiten, die mit einer unausgeglichenen Lastverteilung in Zusammenhang stehen. Überdies sei darauf hingewiesen, daß Datenabhängigkeiten eine Parallelisierung erschweren, indem durch sie eine Reihenfolge in der Abarbeitung erzwungen wird.

Es ist versucht worden, serielles Verhalten in die Leistungsbewertung mit einzubeziehen. Die wohl bekannteste Bewertung geht auf Amdahl zurück. Amdahl teilte die Ausführungszeit $t(1)$ in einen seriellen Anteil t_s und einen parallelisierbaren Anteil t_p auf und ging davon aus, daß der serielle Anteil konstant bleibt [1].

$$t(1) = t_s + t_p \tag{2.4}$$

Der normierte serielle Anteil sei nun definiert mit

$$\mu := t_s/(t_s + t_p), \qquad (0 < \mu < 1). \tag{2.5}$$

Für den Idealfall erhält man so für die parallele Laufzeit $t(n_p)$:

$$t(n_p) = t_s + \frac{t_p}{n_p} = \mu\, t(1) + \frac{(1-\mu)\, t(1)}{n_p} \tag{2.6}$$

Es ist dies Amdahls Gesetz. Der Ansatz von Amdahl führt zu einer Neubewertung von Speed–up S und Effizienz E:

$$S(n_p) = \frac{n_p}{1 + \mu\,(n_p - 1)} = \frac{1}{\mu + \frac{1-\mu}{n_p}} \tag{2.7}$$

$$E(n_p) = \frac{1}{1 + \mu\,(n_p - 1)} \tag{2.8}$$

Eine weitere Relativierung der Leistungsbewertung geht auf Gustafson zurück. Er trug den von Amdahl vernachlässigten Kriterien Rechnung, daß mit steigender Prozessorzahl nicht nur der Kommunikationsaufwand steige, sondern die Problemgröße in der Praxis nicht konstant gehalten würde. Bei der Motivation für den Einsatz eines Parallelrechners, so stellte er fest, überwiege die Aussicht, wesentlich größere Probleme berechnen zu können, häufig die einer Zeitersparnis.
In seinen Annahmen ging Gustafson davon aus, daß bei einer Problemvergrößerung der serielle Anteil absolut gesehen nahezu konstant bleibt, während der parallele Anteil infolgedessen wächst. Prozentual gesehen sinkt also der serielle Anteil an der Gesamtausführungszeit [30].
Dieser Ansatz geht von einem auf 1 normierten parallelen Anteil $t(n_p)$ an der Ausführungszeit aus. Mit $t(n_p) = 1 = \mu + (1 - \mu)$ ergibt sich die Bearbeitungszeit auf einem Einprozessorsystem, das auch den parallelen Anteil seriell ausführen muß, zu

$$t(1) = \mu + (1 - \mu)\, n_p. \tag{2.9}$$

Wird $t(1)$ in Beziehung zu der Ausführungszeit $t(n_p)$ auf n_p Prozessoren gesetzt, so erhält man den Problemgrößenfaktor (Scale–up) nach der Gleichung

$$Sc(n_p) = t(1) \,/\, t(n_p) = \mu + (1 - \mu)\, n_p = \mu - (\mu - 1)\, \mu. \tag{2.10}$$

Ein weiterer interessanter Ansatz folgt der Amdahlschen Betrachtung, – die weiterhin bei Problemen Gültigkeit besitzt, in denen es alleinig auf eine Zeiteinsparung ankommt, – und mündet in eine Grenzbetrachtung für sehr hohe Prozessorzahlen. Da nicht davon ausgegangen werden kann, daß der parallelisierbare Anteil eines Programms sich tatsächlich beliebig parallelisieren läßt, kann für jedes Programm ein maximaler Parallelitätsgrad $n_{p_{max}}$ angegeben werden, bei dem die Effizienz gegen 0 geht. Somit stellt $n_{p_{max}}$ eine Schranke dar, oberhalb der eine Hinzunahme zusätzlicher Prozessoren nicht mehr sinnvoll ist. Für die Bearbeitungszeit gilt demnach die Beziehung

$$t(n_p) = \begin{cases} t_s + \frac{t_p}{n_p} & \text{für} \quad n_p < n_{p_{max}} \\ \\ t_s + \frac{t_p}{n_{p_{max}}} & \text{für} \quad n_p \geq n_{p_{max}}. \end{cases} \tag{2.11}$$

Für $n_p \to \infty$ ergibt sich daraus

$$t(\infty) = \lim_{n_p \to \infty} (t_p) = t_s + \frac{t_p}{n_{p_{max}}}. \tag{2.12}$$

Abschließend zu dieser Betrachtung sei erwähnt, daß es einige weitere verfeinerte Ansätze zur Abschätzung serieller und paralleler Anteile gibt, beispielsweise solche, die die Kenntnis eines realen Parallelitätsprofil voraussetzen, d.h den Verlauf des Parallelitätsgrades über die Ausführungszeit [18].

2.3.4 Parallele Plattformen

Die im Rahmen dieser Arbeit entstandenen Konzepte und Programme wurden weitestgehend unabhängig von der Struktur eines Zielrechners entwickelt, sind jedoch für den Einsatz auf MIMD-Maschinen vorgesehen. Eine Implementierung sowie Testläufe wurden zunächst auf einer Parsytec SC256 durchgeführt und später auf einem Parsytec MC–3 Cluster sowie einer Intel Paragon XP/S. Die beiden letztgenannten Parallelrechner sollen nun an Hand ihrer unterschiedlichen baulichen Konzeption vorgestellt und in ihrer Leistungsfähigkeit miteinander verglichen werden.

Parsytec MC-3

Das *MultiCluster–3 DE* wurde von der Firma Parsytec entwickelt und ist ein Parallelrechner, dessen Architektur auf einem Transputer–Netzwerk basiert. Ein Transputer ist ein *Single–Chip*–Mikroprozessor, der eigens für parallele Anwendungen entwickelt wurde. Der im MC–3 eingesetzte Transputer T805 enthält neben einem 32–bit–Mikroprozessor (CPU) mit assoziiertem Hauptspeicher zusätzlich eine Floating–Point–Unit (FPU) und ist mit vier, auf dem Chip integrierten bidirektionalen Kommunikationslinks ausgestattet. Seine theoretische Leistungsfähigkeit beträgt 1.8 MFlops.
Die Topologie des MC–3 ist ein zweidimensionales Gitter, das fest verdrahtet und somit im Prinzip beliebig skalierbar ist. Von Nachteil ist allerdings, daß die Kommunikation streng der Gitterstruktur folgt, das heißt, wenn zwei Prozessoren, die nicht direkte Nachbarn sind, miteinander Daten austauschen, müssen Nachrichten von dazwischen liegenden Prozessoren weitergeleitet werden. Das bewirkt einen deutlichen Geschwindigkeitsverlust, vor allem, wenn zwei Prozessoren extrem weit auseinanderliegen, sprich der Durchmesser der Topologie sehr groß ist. Hier kann bei der Erstellung einer virtuellen Kommunikationstopologie durch ausgeklügeltes Einbinden der Prozessoren unter dem Gesichtspunkt einer Optimierung der Weglängen einiges erreicht werden.

Auf das Transputer–Netzwerk kann nur über eine angekoppelte Unix-Workstation zugegriffen werden, die als Hostrechner fungiert. Der Host stellt Betriebssystem, Entwicklungswerkzeuge und Compiler bereit und sorgt für den Zugang zu anderen

Rechnern und sekundären Speichermedien. Das Betriebssystem PARIX (PARallel extensions to unIX) enthält die notwendigen sprachlichen Erweiterungen für einen parallelen Betrieb, darunter unter anderem Bibliotheken für Message Passing, parallele Programmiermodelle und numerische Mathematik (Blas-Routinen).
Jeder Knoten im Netzwerk führt das identische Programm aus, das bei der Initialisierung geladen wird. Individuelles Verhalten wie Verzweigungen im Programm und die Wahrnehmung unterschiedlicher Aufgaben entsteht alleine durch eine knotenabhängige Programmierung. Hierbei wird sich zunutze gemacht, daß jeder Prozessor über Systembefehle Basisinformation abrufen kann, wodurch er z.B. seine Identität feststellen kann oder die Anzahl der an der Anwendung beteiligten Prozessoren.
Die Kommunikation erfolgt über eine sogenannte virtuelle Topologie, in der Prozessoren über virtuelle Links beliebig miteinander vernetzt werden können. Mehrere virtuelle Topologien können nebeneinander existieren und namentlich identifiziert werden (*top_id*). So kann für verschiedene Aufgaben die jeweils passende Topologie eingesetzt werden. Virtuelle Topologien werden vom Anwender entweder selbst dynamisch erstellt oder als vordefinierte Topologie aus einer Bibliothek aufgerufen.

Intel Paragon

Die Intel Paragon XP/S–5 wurde von der Firma Intel als ein Rechner mit verteiltem Hauptspeicher (MIMD) entwickelt, der mit bis zu 2000 Rechenknoten aufrüstbar ist. Jeder Einzelknoten kann als ein eigenständiger Rechner angesehen werden und ist mit zwei Prozessoren des Typs i860XP ausgerüstet, von denen der Applikationsprozessor für die Bereitstellung der Rechenleistung sorgt und der Kommunikationsprozessor den Zugang zum Netzwerk schafft.
Jeder Knoten verfügt, neben einem lokalem Hauptspeicher von 16 bis 192 MByte, über ein Netzwerk–Interface und zwei sogenannte *transfer engines*, die den Zugang zu den Kommunikationskanälen ermöglichen. Jeder der vier bidirektionalen Kommunikationslinks eines Knotens erreicht eine Übertragungsleistung von 200M Byte/s.
Im Gegensatz zur Parsytec ist die Paragon ein hostfreier Rechner und demgemäß direkt adressierbar. Einige Knoten sind als IO–Knoten (SCSI–, VME– oder HIPPI–Knoten) ausgelegt, haben also Anschluß an das Ethernet bzw. FDDI–Verbindung. Somit kann jeder Knoten transparent auf externe Medien wie Festplatte, Bandlaufwerkgeräte oder das Internetz zugreifen.
Die Rechenknoten der Paragon sind in einen zweidimensionalen Gitter angeordnet. Diese Netzwerktopologie hat neben dem Vorteil einer theoretisch beliebigen Skalierbarkeit und einer inkrementellen Erweiterbarkeit der Hardware allerdings den Nachteil langer Kommunikationswege. Im Gegensatz zur Parsytec können jedoch

durch die großen Durchmesser bedingte hohe Latenzzeiten durch schnelle Routing–Chips in Kombination mit Routing–Strategien zumindest stark gemindert werden.
Das Betriebssystem der Paragon OSF/1 ist eine Version des UNIX der Open Software Foundation, die um Elemente für eine parallele Datenverarbeitung bereichert wurde. Dadurch, daß die Paragon nicht im Front–End/Back–End–Modus betrieben wird, ist sie flexibler einsetzbar als die meisten Parallelrechner.
Eine weitere Eigenart ist die Aufteilung zu Verfügung stehender Knoten in sogenannte Partitionen, in denen die gleichzeitige Abwicklung mehrerer Anwendungen kontrolliert werden. Unterschieden wird die Service–Partition, die für die Programmentwicklung ideal ist, und die Compute–Partition, in der die Abarbeitung paralleler Programme auf vom Benutzer allokierten Knoten erfolgt. Solche frei wählbaren Aufteilungen kann der Benutzer jederzeit modifizieren oder wieder löschen. Wo Partitionen verschiedener Benutzer sich überlappen, wird nach Zeitplan (*time sharing*) gefahren (*gang scheduling*).
Ferner werden von der Paragon eine Reihe Programmiermodelle unterstützt wie *shared virtual memory*, *data and worksharing* und vor allem *message passing*. Für das Message Passing können neben dem NX der Paragon auch die Quasistandards MPI (*message passing interface*) und PVM (*parallel virtual memory*) eingesetzt werden.

Bewertung der verwendeten Zielsysteme

Als Parallelrechner standen für diese Arbeit eine Intel Paragon XP/S–5 mit 72 Prozessoren zu je 32 Mbyte Hauptspeicher sowie ein Parsytec MultiCluster MC3–DE mit 44 Transputern des Typs T805 zu je 8 Mbyte Hauptspeicher zur Verfügung. Die Parsytec, der eine Sun Sparc 10 Workstation als Host vorgelagert war, diente auch als Prüfstein für die Konzeption und Entwicklung der parallelen Software, die für die Paragon nur angepaßt und entsprechend modifiziert wurde.

Die Software wurde auf beiden Parallelrechnern ausgiebig getestet. Dabei wurde im Vorfeld der Untersuchungen die generelle Leistungsfähigkeit der Parallelrechner an Hand simpler Testfälle bewertet, welche die Rechenzeit und die Übertragungsgeschwindigkeit messen.
Zur Ermittlung der Rechenleistung wurden einfache Fließkomma–Operationen, z.B. die Addition und Multiplikation zweier einfachgenauer Zahlen, wiederholt ausgeführt und die Anzahl der Operationen auf eine Sekunde normiert.
Zur Ermittlung der Kommunikationsleistung wurden Latenzzeit und Sendezeit gemessen, also einerseits die Zeit, die zum Aufbau der Verbindung verstreicht (*startup time*) und andererseits die Zeit, die es zum Verschicken eines Wortes bedarf.
Den Messungen zufolge schnitt die Paragon durchweg um eine Größenordnung besser ab. Die Rechenleistung war bei der Paragon mit 5.9 MFlops (0.17 μs/Byte) um

den Faktor 12.3 höher als bei der Parsytec die es auf 0.48 MFlops (2.1 μs/Byte) brachte. Die Latenzzeit betrug 44 μs gegenüber 70 μs beim MC–3 bei unmittelbaren Nachbarn. Während bei der Paragon infolge schnellen Routings eine Zunahme der Latenzzeit mit dem Durchmesser kaum zu verzeichnen war, nahm sie beim MC–3 mit jedem Zwischenschritt durchschnittlich um 12 μs zu.
Für die Kommunikationsleistung wurde beim MC–3 Werte um 2.9 μs/Byte zwischen direkten Nachbarn gemessen und Werte von 5.0 bis 5.2 μs/Byte bei längeren Laufwegen, d.h. wenn zwischen Sender und Empfänger mehrere Knoten liegen. Bei der Paragon schwankten die Leistungswerte zwischen 0.29 und 0.38 μs/Byte.
Das Verhältnis von Rechenleistung zu Kommunikationsleistung gilt als ein Maß für die Ausgewogenheit des Systems. Hier wurde der Idealwert von 1 von der Intel Paragon nahezu erreicht. Die Parsytec liegt mit 1.8 noch in innerhalb des Bereichs akzeptabler Werte.

2.4 Parallele Architekturen in der Visualisierung

In diesem Abschnitt soll eine möglichst allgemeine Architektur für einen Visualisierungsprozeß entworfen werden, der paralleles und verteiltes Rechnen mit einschließt. An Hand zu entwickelnder Konzepte soll aufgezeigt werden, in wie weit diese im Rahmen dieser Arbeit umgesetzt werden konnten, und welche Punkte offen bleiben mußten.
Zuerst aber sollen die Architekturen einiger wichtiger Visualisierungssoftwarepakete gerade in Hinblick auf paralleles und verteiltes Rechnen untersucht werden. Unter den heute führenden Paketen, IBM-Data-Explorer [46], Khoros [49], AVS–Express [52], AVS 5.0 [82], SGI-Explorer [51] und COVISE [44], bieten bisher nur IBM-Data-Explorer, Khoros und COVISE Funktionalität für verteiltes und/oder paralleles Rechnen an. In gewissen Umfang kommt durch sein Konzept vom fernen Modul (*remote module*) auch AVS 5.0 in Betracht, während bei AVS–Express diesbezügliche funktionale Erweiterungen erst in Planung sind [53].
Allen Visualisierungspaketen ist zunächst gemein, daß sie über ein visuelles Programmiermodell verfügen, mit dessen Hilfe der Anwender seine individuellen Applikationen aufbaut. Generell entsteht eine lauffähige Anwendung durch die visuelle Vernetzung von Softwarebausteinen bzw. derer Symbole oder Ikonen, die auf einer Arbeitsfläche positioniert wurden. Mit der Verknüpfung dieser graphischen Elemente, die sich nach den ihnen eigenen Regeln interaktiv manipulieren lassen, ist neben der zeitlichen Ausführung der einzelnen unterlegten Prozesse (*scheduling*) auch geregelt, woher diese ihre Daten beziehen. AVS 5.0 beispielsweise verwendet zu diesem Zweck "*shared memory*" und "*remote procedure calls*" (RPC).

Unterschiede gibt es in der jeweiligen Terminologie. So werden "Visuelles Programm" und "Modulnetzwerk" weitgehend synonym gebraucht, ebenso "Glyphen und Ikonen", sowie "Visueller Programm Editor (VPE)" und "Netzwerkeditor".
Es folgt eine Übersicht einiger Konzepte zur parallelen Verarbeitung in oben erwähnten Visualisierungspaketen. Soweit es einem besseren Verständnis dient wird am Rande auch auf den grobstrukturellen Aufbau und das Datenflußmodell des jeweiligen Systems eingegangen werden.

2.4.1 Remote Moduln in AVS 5.0

- **Allgemeines**
  AVS 5.0 besteht aus zwei Hauptteilen, einmal der Hauptanwendung (*main application*), der auch der AVS–Kernel angehört, und zum anderen die Moduln, die Softwarebausteine, die zu Datenflußnetzwerken zusammengebunden werden können. Der AVS–Kernel enthält die Editoren für Netzwerk– und Control Panel Layout, den Benutzerschnittstellen-Code (*user interface code*) und Funktionen, die die Executive bilden, d.h. für die Ausführung des Datenflußnetzwerks verantwortlich sind.

- **Ferne Moduln**
  Benutzerdefinierte Moduln werden als separate UNIX–Programme implementiert, die mit dem AVS–Kernel via Sockets (Berkley UNIX) aber auch über gemeinsamen Speicher (*shared memory*) kommunizieren. Im allgemeinen werden dazu die lokalen UNIX–Sockets verwendet und INET–Sockets bzw. TCP–Sockets nur dann, wenn Moduln auf fernen Maschinen laufen [43].
  AVS 5.0 bietet also grundsätzlich die Möglichkeit, Moduln in ein Modulnetzwerk mit einzubinden, deren Rumpf auf einer fernen Workstation angesiedelt ist. Die Kommunikation zwischen den lokalen und fernen Prozessen eines Moduls erfolgt dabei über RPC bzw. die Externe Daten–Repräsentation XDR (*external data representation*), die zusammen mit RPC zu NFS gehört.
  Ein solches Modul wird am geeignetsten als ein Coroutine–Modul implementiert, das, im Gegensatz zu einem Subroutine–Modul, vom übrigen Datenfluß–Scheduling abgekoppelt ist und immer nur dann ausgeführt wird, wenn es einen Impuls von außen erhält, etwa wenn aktuellere Eingangsdaten vorliegen.

- **Bemerkung**
  In der in dieser Arbeit vorgestellten Visualisierungssoftware, deren Verwaltungsstruktur ja größtenteils AVS 5.0 unterstellt wurde, wurden ferne Moduln nicht eingesetzt, zum einen wegen Plattformabhängigkeiten wie die unterschiedliche Datenrepräsentation beteiligter Rechner (*little endian* versus *big*

endian), und zum anderen mangels Möglichkeiten, direkt mit einem Typus Parallelrechner zu kommunizieren, dem eine Hostmaschine vorgeschaltet ist, etwa die Parallelrechner von Parsytec. Nicht zuletzt auch, um unabhängig zu bleiben, wurde ein eigenes Konzept der Kommunikation mit fernen Einprozessor- wie Multiprozessorrechnern entwickelt und umgesetzt, eine Software, die auf der Basis der Sockets funktioniert und durch ihren nur gerinfügigen Portierungsaufwand relativ plattformunabhängig ist. Näheres zu diesem Konzept der Transmitter ist in Kapitel 6, weiteres über AVS 5.0 in Kapitel 3.2. nachzulesen.

2.4.2 Das Visualisierungsprogrammsystem IBM Data Explorer

- **Allgemeines**
  Die Systemstruktur des Data Explorer wurde als Client–Server Modell ausgelegt. Die Architektur umfaßt Systemkomponenten wie zum Beispiel TCP/IP, X–Windows–System, Motif und Sockets. Der Client ist die Benutzeroberfläche, während der Server das ausführende Organ ist, die Executive. Diese kann auf derselben Plattform ablaufen, auf einer anderen, oder sich sogar über mehrere Plattformen verteilen. Der Data Explorer erlaubt es also, Serversoftware einzubeziehen, die auf verschiedenen Hardware–Plattformen beheimatet ist.
  Der Data Explorer kann als System mit vier Schichten verstanden werden, von der jede ihre eigene Datenschnittstelle besitzt. Die vier Schichten heißen "*Graphical user interface*", "*Executive*", "*Modules*" und "*Data management*".
  Die graphische Benutzerschnittstelle (*graphical user interface*) erlaubt dem Benutzer das Erstellen und Kontrollieren seines Visualisierungsprozesses. Während Visuelle Programme bzw. Modul–Netzwerke mit Hilfe des Visual Program Editors (VPE) erstellt werden, erfolgt die Kontrolle des Ablaufs über die zu den einzelnen Moduln gehörigen Control Panels, welche die verschiedenen Eingangsparameter zugänglich machen.
  Die Executive basiert auf dem Datenflußkonzept und hat die Aufgabe, die einzelnen Visualisierungsfunktionen eines visuellen Programms abzuarbeiten. Darunter fällt auch der Zeit- und Ablaufplan (*scheduling*) für die einzelnen Prozesse und das Datenflußmanagement. Für die Visualisierungsfunktionen steht ein reichhaltiges Modulpaket zur Verfügung. Moduln können auf verschiedene Weise verfügbar gemacht werden. Wo ihre Ikonen in einem Visuellen Programmier–Netzwerk verwendet werden, fungieren sie als Knoten. In der Scripting Language dienen sie als Funktionsaufrufe, lassen sich aber auch in in-

tegrierten Anwendung verwenden, dort als Teile der Visualisierungsbibliotheks–Programmierschnittstelle (*visualization library programming interface*).
Das Datenmanagement schließlich verschafft den Moduln den Zugang zum Datenmodell. Ferner übernimmt es das Erzeugen und Bearbeiten von Datenobjekten, stellt aber auch allgemeine Systemdienste zur Verfügung sowie Routinen, mit denen sich neue Moduln via Anwendungsprogrammierschnittstelle (*application programming interface*) API einbringen lassen [47].

- **Externe asynchrone Datenquellen**
  Innerhalb des "Execution Model" wird ein Netzwerk für eine direkte Schnittstelle zu einer externen Datenquelle zur Verfügung gestellt. Damit eignet es sich besonders für die Echtzeit–Simulation von Daten, die asynchron von einem Prozeß dieser Art erzeugt wurden.
  Eine solche externe Datenquelle wird an den Data Explorer angebunden, indem ein kommunikationsfähiges Modul eingebunden wird, das Daten von der externen Quelle, vorzugsweise über einen Socket bezieht und das resultierende Datenobjekt an den Modulausgang legt. Diese Vorgehensweise ist mit dem fernen Modul in AVS 5.0 vergleichbar. Ist das Kommunikatonsmodul ein Outbord–Modul (*outbord module*), werden es und seine Abkömmlinge immer nur dann ausgeführt, wenn die externe Datenquelle anzeigt, daß neue Daten verfügbar sind. Das Outbord–Modul entspricht dabei dem Coroutine–Modul in AVS, das ein unabhängiger Prozeß ist. Das Äquivalent zum Subroutine–Modul in AVS ist das Inbord–Modul, welches natürlich auch extern asynchron kommunizieren kann, jedoch nur dann neue Daten importiert, wenn der Benutzer das Modulnetzwerk ausführt.

- **Parallelismus**
  Parallelismus wird in den Data Explorer auf zwei Arten eingeführt, einmal durch verteiltes Rechnen (*distributed processing*) auf verschiedenen Rechnern, wie z.B. einem Workstation–Cluster, und zum anderen durch den Einsatz einer Multiprozessormaschine, also eines echten Parallelrechners mit gemeinsamem Speicher.

  Verteiltes Rechnen
  Beim verteilten Rechnen wird die Ausführung eines visuellen oder eines mittels der Skriptsprache (*scripting language*) erstellten Programms auf mehrere Workstations in einem Netzwerk (NOW) aufgeteilt. Verschiedene Abschnitte der Visualisierung lassen sich so simultan ausführen. Auf diese Weise werden zum Beispiel erweiterte Ressourcen für rechenintensive Aufgaben genutzt, oder es werden Algorithmen direkt auf fernen Maschinen auf dort lagernde Daten angewandt, womit sich zudem die Aufwände für den Datentransfer reduzieren.

Paralleles Rechnen

Parallelrechnen ist mit dem Data Explorer nur in seiner SMP–Version auf einer SMP–Workstation möglich, einem Multiprozessorsystem. Allerdings unterscheidet sich das Konzept von dem weit verbreiteten Clustermodell, wie es Maschinen wie IBM–SP2, Intel–Paragon, Cray T3D oder Parsytec MC verwenden, die Multirechnersysteme sind. Die Parallelisierung beruht hier nämlich auf dem Modell eines gemeinsamen Speichers (*shared memory model*), d.h. die Daten sind logisch in einem einzigen Adreßraum angesiedelt, auf den jeder Parallelprozessor zugreift.

Die Parallelisierung besteht nun in der Verteilung der Aufgaben (Tasks) und der Aufteilung der gemeinsamen Daten des Adreßraumes in Partitionen. Jeder Prozessor bearbeitet seine eigene Partition.

Zu diesem Zweck stellt der Data Explorer SMP das Standardmodul "Partition" bereit. Zunächst werden die Daten, – zumeist sind es Datenobjekte vom Typ "Field"–, in lokale unabhängige Gebiete aufgeteilt. Das Ergebnis ist ein zusammengesetzes Feld, dessen Mitglieder die Partitionen sind. Da nun jedes Interpolationselement eines Eingangsfeldes einer ganz bestimmten Partition zugeordnet ist, nehmen die sich ergebenden Partitionen exakt und ohne Überlappungen den Raum des Ursprungsfeldes ein.

Jedoch müssen, um über die gemeinsamen Grenzen zweier Partitionen hinweg interpolieren zu können, temporäre Überlappungen generiert werden. Den Auf- und Abbau dieser Nachbarschaftsregionen übernehmen die Routinen *grow* und *shrink*, und zwar bevor die individuellen, den einzelnen Partitionen entsprechenden Subtasks erzeugt werden [48].

Hinzuzufügen bleibt nur noch, daß der Data Explorer SMP dem Software–Entwickler die komplizierte parallele Programmierung erleichtert, indem er einfache Parallelisierungsmodelle bereitstellt, die auf dem Prinzip der Verzweigung und Zusammenführung (*fork-join*) beruhen, wodurch eine grobkörnige Parallelisierung erreicht wird. Durch die Verwendung von Automatismen wird hier weitgehend vermieden, daß Anwendungen blockieren, bzw. kommunikationsbedingt in einen Totpunkt geraten (*deadlock*).

2.4.3 Das Visualisierungsprogrammsystem Khoros

- **Allgemeines**
  Unter den Visualisierungsprogramsystemen stellt Khoros mit seiner Philosophie eines Software–Baukastens die Ausnahme dar. Im Gegensatz zu den anderen hier beschriebenen Application–Builder–Systemen bietet Khoros keine fertig einsetzbaren Anwendungen sondern versteht sich in erster Linie als eine Software–Integrations– und –Entwicklungsumgebung. Das System stellt eine Reihe von Entwicklungswerkzeugen (*Toolboxes*) und Prozedurensammlungen (*Programming Services*) zur Verfügung, mit denen ein Entwickler sich individuelle Anwendungen erstellen und sogar sein eigenes Software–System entwerfen kann. Die Entwicklungswerkzeuge decken die Bereiche Bildverarbeitung, Datenmanipulation, Matrix–Verarbeitung, und wissenschaftliche Visualisierung ab. Daneben existiert die visuelle Programmierumgebung "Cantata", die es dem Benutzer erlaubt, Anwendungen aus den zuvorgenannten Bereichen zusammenzubringen und zusammenarbeiten zu lassen.

- **Toolbox Programming**
  Mit dem Khoros–System kann auf einer oder auf mehreren Ebenen interagiert werden. Jede Ebene bietet einen anderen Programmierzugang, beispielsweise dem reinen Anwendungsprogrammierer, der einfach nur einen Datensatz visualisieren möchte, ebenso wie dem Toolbox-Programmierer, der vielleicht eine neue Datenverarbeitungs- oder eine Bildgenerierungsroutine implementieren will.
  Jede Programmbibliothek innerhalb des Khoros–Systems ist mit einer der Toolboxen verknüpft. Eine Toolbox ist eine thematische Sammlung von Programmen und Bibliotheken, die als eine Einheit verwaltet wird. Eine Toolbox zwingt seinem Inhalt eine vordefinierte Verzeichnisstruktur auf, damit die Software später konsistent und konfigurierbar wird.
  Grob gliedern sich die das System ausmachenden Toolboxen in Installationshilfen, das Software–Entwicklungssystem, das Visualisierungssystem, Datenverarbeitungsroutinen, Bildverarbeitungsroutinen, spezielle 3D–Visualisierungsprogramme, Verarbeitungsroutinen für lineare Algebra und Matrizen sowie Migrationsroutinen für die Einbindung von Software aus einer älteren Khoros–Version.

- **Programming Services**
  Die Khoros–Programmierdienste genannt *Programming Services* sind eine Untermenge einzelner Toolboxen und setzen sich aus Gruppen von Bibliotheken zusammen, die dem Software–Entwickler eine große Auswahl von Programmierschnittstellen zur Verfügung stellen. Die drei wichtigsten Dienste

der Software–Infrastruktur, deren Namen bereits ihre Aufgaben verraten, sind *Foundation Services*, *Data Services*, und *GUI & Visualization Services*. Die Foundation Services gliedern sich noch einmal in *Basic Services*, *Math Services*, *Expression Services*, *Operating System Services*, *Software Services* und *User Interface Services*.

Foundation Services

Die *Foundation Services* sind ein Sammelbecken für alle Programmierdienste, die weder mit Datenverarbeitung (*data processing*) noch mit Datendarstellung (*data display*) etwas zu tun haben.

Die *Basic Services* bieten dem Software–Entwickler eine weite Bandbreite an Grundfunktionalität für Speicherallokierung, Manipulation und Analyse (*parsing)* von Zeichenketten, Message Reporting und vieles mehr.

Die *Math Services* stellen maschinenunabhängige Implementierungen allgemeiner und erweiterter mathematischer Operationen zur Vefügung, die sehr effizient und weitgehend portierbar sind.

Die *Expression Services* enthalten einen symbolischen Ausdrucksanalysierer (*expression parser*) für die Evaluierung mathematischer Gleichungen und Funktionen.

Die *Operating System Services* koppeln Khoros vom Betriebsystem ab und bieten zusätzliche Funktionalität, um beispielsweise ein verteiltes Rechnen zu ermöglichen.

Die *Software Services* ermöglichen dem Toolbox–Programmierer eine kohärente Sicht der verschiedenen Komponenten seiner Software. Auf diese Weise wird die Produktivität bei der Software–Entwickling erhöht, z.B. dadurch, daß ein Satz von Dateien zusammen mit einem Programm, einer Bibliothek oder einem Skript als ein einziges Software–Objekt assoziiert werden kann.

Die *User Interface Services* schließlich geben eine Basisunterstützung für die Benutzung der Befehlszeilen-Benutzerschnittstelle (*command line user interfaces*) CLUI des Khoros–Programmes. Des weiteren ermöglichen sie eine automatische Code-Generierung für die graphische Benutzerschnittstelle (*graphical user interface*) GUI sowie das Erstellen einer Programmdokumentation.

Data Services

Die *Data Services* stellen ein leistungsfähiges System für den Datenzugang und die Datenmanipulation zur Verfügung. Der Anwendungsprogrammierer soll,
– unabhängig von Dateiformaten oder physikalischen Merkmalen wie Länge und Datentyp, – auf seine Daten zugreifen und sie manipulieren können, sei es bei der Bild- und Signalverarbeitung, der Geometrie–Visualisierung oder der Numerischen Analyse.

GUI & Visualization Services
Die *GUI & Visualization Services* stellen alle Dienste zur Verfügung, die mit der graphischen Darstellung mit Hilfe von X–Toolkit und X–Windows zu tun haben. Dies schließt das interaktive Generieren und Modifizieren einer graphischen Benutzeroberfläche ebenso ein wie den Aufbau einer graphischen Benutzerschnittstelle. Als Werkzeug dazu dient eine abstrakte Widget–Bibliothek, die transparent und simultan auf einigen weitverbreiteten Widget-Sets aufsetzt (Athena, OLIT oder Motif). Des weiteren ist den Diensten ein umfangreiches Paket von Visualisierungstools inbegriffen das z.B. für die Darstellung und Bearbeitung von Bildern, Farbtabellen, 2D- und 3D–Plotten, Oberflächen–Rendering und einigem mehr geeignet ist.

- **Cantata**
  Verbindendes Glied für alle Systemkomponenten ist "Cantata", die visuelle Programmierumgebung, mit der sich u.a. visuelle Hierarchien aufbauen lassen, Kontrollstrukturen, beispielsweise für Verzweigungen und Zusammenführungen des Datenflusses, aber auch abgekoppelte Programme, die später unabhängig von der Umgebung als eigenständige Anwendungen ablaufbar sind. In dem visuellen Programmier-Toolkit werden die einzelnen Programme des Khorossystems durch visuelle Objekte repräsentiert, die Glyphen genannt werden. Um nun ein visuelles Programm zu erstellen, wählt der Anwender die gewünschten Programme aus einem Vorrat aus und positioniert die korrespondierenden Glyphen auf der Arbeitsfläche. Indem er Glyphen miteinander verbindet, zeigt er den Fluß der Daten von Programm zu Programm an. Somit entsteht auf der Arbeitsfläche ein Netzwerk.

- **Parallelismus**
  Gegenwärtig ist Khoros auf Parallelrechnern nicht einsetzbar, sehr wohl aber für verteiltes Rechnen geeignet. Innerhalb der Programmierdienste stellt das Paket Betriebsystemdienste (*Operating System Services*) dazu neben Bibliotheken für "Datentransport" und "Prozeßausführung" auch eine Bibliothek für verteiltes Rechnen zur Verfügung. Mit Hilfe dieser Bibliotheken kann der Entwickler sich verteilte Anwendungen zusammenprogrammieren, die dann von den spezifischen Transportmechanismen des Betriebsystems isoliert bleiben.

  Datentransport und Verteiltes Rechnen
  Der Mechanismus des Verteilten Rechnens innerhalb der Betriebsystemdienste enthält sowohl Prozeß- als auch eine Transportabstraktionen, die mehrere Maschinen zu einem Netzwerk vereinen, und den Ort der Daten und Rechenressourcen den Khoros-Anwendungen transparent gestalten. Auf jedem beteiligten Rechner muß zu Beginn der Daemon "Phantomd" gestartet werden, die

distributed computing program utility, die für die Interprozeßkommunikation und den Datentransport zwischen den Rechnern verantwortlich ist. Auf lokaler Ebene unterstützt Khoros die Transportmechanismen Shared Memory, Standard Unix File, Standard Pipes, und Named Stream/Pipe, für den Transport auf fernen Rechnern Socket Transport, Memory Mapped und Sys V Transport Layer Interface (tli), immer jedoch vorausgesetzt, daß die Architektur des verwendeten Rechners die Mechanismen verfügbar macht.

2.4.4 Die Software–Umgebung COVISE

- **Allgemeines**
  COVISE ist eine erweiterbare verteilte Software–Umgebung, die mit dem Ziel geschaffen wurde, Funktionalitäten der Simulationsrechnung, des Postprocessing und der Visualisierung zu integrieren. So steht denn auch das Kürzel COVISE für **CO**llaborative **VI**sualization and **S**imulation **E**nvironment. COVISE wurde speziell entwickelt, um eine Zusammenarbeit zwischen Ingenieuren und Wissenschaftlern zu ermöglichen, die über eine Netzwerk–Infrastruktur miteinander verbunden sind. Das gelingt mit COVISE durch Zerlegung einer Anwendung in mehrere Verarbeitungsschritte, die durch Moduln repräsentiert werden. Die Moduln werden als alleinstehende Prozesse implementiert, die, je nach den Anforderungen, auf verschiedenen heterogenen Plattformen verteilt sein können. Bei einer Zusammenarbeit über mehrere Plattformen hinweg ist eine hohe Verfügbarkeit und Performance unabdingbar. Hier bietet COVISE gerade hinsichtlich des Einsatzes leistungsfähiger Parallel- und Vektorrechner in Verbindung mit schnellen Netzwerken Techniken zur Integration und Optimierung an.

- **Datenmodell**
  Der MapEditor bildet den Hauptteil der Benutzerschnittstelle und ist ein auf Motif basierendes Programm. Die Bedienoberfläche enthält eine Arbeitsfläche, in der die Moduln plaziert werden, die über einen Point&Klick–Mechanismus zu einem Flußdiagramm verbunden werden. Damit ist der Datenfluß zwischen den Moduln einer Anwendung festgelegt.
  Neben den Moduln der Kategorien IO, Filter, Mapper und Renderer stellt COVISE die Kategorien Tools, Tracer und Simulations zur Verfügung. So stellt die Kategorie IO für die Datenein– und –ausgabe ein breites Spektrum an Moduln zur Verfügung und deckt beispielsweise gängige Datenformate wie die der Simulationspakete ProEngineer, Star–CD, Patran, und FIDAP ab. Neben den Standard–Visualisierungstechniken für strukturierte und unstrukturierte

Gitter, die durch eine Vielzahl von mathematischen Tools und Filtertechniken unterstützt werden, enthält das Paket z.B. auch Moduln für serielles und paralleles Particle–Tracing und virtuelle Realität. In diesem Zusammenhang sei das Renderer–Modul für virtuelle Realität COVER (COVISE Virtual Environment) erwähnt, das ein integraler Bestandteil von COVISE ist und aufwendige technisch–wissenschaftliche Anwendungen wie z.B. Crash–Simulationen unterstützt. Um die Rendering–Hardware von SGI–Workstastions optimal ausnutzen zu können, basiert COVER auf dem IRIS–Perfomer von Silicon Graphics.
Darüber hinaus kann, bedingt durch die modulare Struktur, eigener Simulations– und Rendering–Code leicht in COVISE integriert werden. Dieser kann dann als neues Modul mit bestehenden Moduln kommunizieren.

- **Verteilte Anwendungen**
  COVISE eignet sich sowohl für die Steuerung einer verteilten Simulationsberechnung, als auch für die Zusammenarbeit mehrerer Benutzer über Rechnergrenzen hinweg.
  Eine COVISE–Sitzung wird üblicherweise auf einer lokalen Workstation gestartet. Die Bedienoberfläche erscheint und der Controller startet. Alle anderen Prozesse, ob lokale oder ferne, werden von hier aus erzeugt. Zusätzlich kann der Benutzer Host–Rechner für eine ferne Modulabarbeitung *(remote module execution)* in seine Sitzung mit einbeziehen. Auf jeder Maschine existiert ein gemeinsamer Datenraum der aus gemeinsamem Speicher besteht.
  In einer gemeinsamen Sitzung gibt es ebensoviele Benutzerschnittstellen (UI) wie Teilnehmer, wobei immer ein Benutzer der Master ist, der die gesamte Kontrolle der Umgebung innehat, während die restlichen Benutzer die Slaves sind. Ein Slave ist im Prinzip passiv und kann als einzige Aktivität die Masterrolle anfordern.

2.4.5 Gründe für die Auswahl von AVS als graphische Basismaschine

In dieser Arbeit wurde als graphische Basismaschine AVS der Vorzug vor COVISE gegeben. So hatte zumindest zum Zeitpunkt der Entscheidung Anfang der Neunzigerjahre AVS bereits einen leistungstarken Renderer und ein kompaktes graphisches Gesamtkonzept, in das sich Eigenentwicklungen in den Programmiersprachen Fortran und C bestens eingliedern ließen (siehe Kap. 3.2).
Allerdings muß auch darauf hingewiesen werden, daß mit der Verwendung von AVS eine gewisse Steuerproblemtik in Kauf zu nehmen ist, was bei COVISE nicht der Fall

ist. Bei Anwendungen, die in Teilbereichen auf entfernten Rechnern ablaufen, hat die entfernte Seite (z.B. ein Parallelrechner) keinen Zugriff auf die Kommunikationsschale von AVS (siehe dazu Kapitel 6.3). Im Gegensatz zu AVS erlaubt das visuelle Programmiermodell von COVISE durch seine Technik der Netzwerk–Infrastruktur jedoch eine Steuerung auch der parallelen bzw. der entfernten Prozesse.
Bei AVS erhält man infolgedessen unkonnektierte Moduln, d.h. es ist dem Anwender überlassen, den Ablauf richtig zu steuern bzw. die Moduln in der richtigen Reihenfolge auszuführen.

2.4.6 Entwurf einer allgemeinen Architektur für die Visualisierung

Nachdem in den vorigen Unterabschnitten ein kurzer Streifzug durch die Welt kommerzieller Visualisierungspakete unternommen wurde, soll in diesem Unterabschnitt ein Konzept für eine möglichst allgemeine Visualisierungsarchitektur entworfen werden, die neben konventioneller sequentieller Verarbeitung sowohl verteiltes als auch paralleles Abarbeiten einzelner Aufgaben vorsieht. Im Gegensatz zum IBM Data Explorer bezieht sich Parallelität hier auf ein Multirechnersystem (Message Passing).

Benutzeroberfläche
Zentrales Element des Systems ist die Benutzeroberfläche, über die ein Anwender mit den einzelnen Programmsystemkomponenten kommuniziert. Die Benutzeroberfläche und die mit ihr assoziierten Verwaltungsprogramme stellen den Client dar (im Idealfall ein dünner Client), der das Bild eines homogenen Programms vermittelt bzw. emuliert. Im allgemeinen sieht der Anwender also nur seine Benutzeroberfläche, die ihm den Zugang auch zu fernen Komponenten und Rechnern verschafft. Die fernen Rechner verstehen sich als die Server, auf denen die verschiedenen Serverprogramme bzw. Dienstleister installiert sind, derer sich ein Client bedient.
Merkmal der zu entwickelnden Software sollte zudem sein, daß nicht auf jeder Servermaschine das darunterliegende Programmsystem, – im Falle der zu entwickelnden Software das AVS 5.0, – implementiert werden muß, was nicht zuletzt auch eine Kostenfrage, sicher jedoch eine Frage der Hardware ist und außerdem nicht den vorgefundenen Gegebenheiten entsprach.

Dienstleister
Auf den Servermaschinen kann nun jeweils ein Programmblock installiert werden, als ein Dienstleister, der jederzeit von außerhalb ansprechbar ist, beispielsweise via TCP/IP. Wird ein solcher Dienstleister gestartet, erhält er einen Auftrag übermit-

Abb. 2.3: Netzwerk mit Dienstleistern

telt, den er dann entweder auf lokalen Daten abarbeitet oder auf Daten, die er im Zuge des Auftrages erst noch transferiert bekommt.
Dabei kommunizieren Client und Server über Sockets miteinander. Der Dienstleisterprozeß terminiert, sobald er das Signal dazu erhält, bzw. wenn er seinen Auftrag erledigt hat.
Der Dienstleister besteht nun aus einem Paket aus Dienstleistungsprogrammen und Routinen, die daten- und anweisungsabhängig abgearbeitet werden. Im Falle eines Parallelrechners enthält das Programmpaket zusätzlich zu den Routinen für externen Datenaustausch ein Bündel von internen Kommunikationsroutinen, d.h. Routinen, die für den Informations- und Datenaustausch zwischen den einzelnen Parallelprozessoren verantwortlich sind. Die Kommunikationsroutinen sind ganz auf die jeweilige Rechnerarchitektur (z.B. Parsytec, Intel–Paragon) und das jeweilige parallele Betriebsystem (z.B. PARIX, OSF/1) zugeschnitten.
Ein Dienstleisterprogramm wird aus einem Modul der Benutzeroberfläche heraus über RPC initialisiert und bekommt dann aus gleicher Quelle seinen Auftrag vermittelt. Einem Auftrag liegt ein vereinbartes Datenformat zugrunde, daß allen beteiligten Programmen bekannt ist. Gleichsam bekannt sein müssen jedem Teilnehmerrechner die TCP/IP-Adressen der anderen beteiligten Rechner, die in einer Datei

abgelegt sind (z.B. *.rhosts* für *remote hosts*).
Einen solchen losen Rechnerverbund zeigt nun Abbildung 2.3. Der Anwender sitzt an seinem Client–Rechner und steuert über seine zentrale Oberfäche verschiedene Dienstleister an, um seine spezifischen Aufgaben gelöst zu bekommen. Dabei kann er auch gezielt einen Parallelrechner einsetzen, auf dem parallele kommunikationsfähige Dienstleisterprogramme installiert wurden.

Verteilte und parallele Visualisierung
Das Dienstleisterkonzept läßt sich nun auf konkrete Anwendungen übertragen. In der vorliegenden Arbeit stand die Visualisierung von Ergebnisdaten einer Finite Element Analyse im Vordergrund. An diesem Unterabschnitt geht es vor allem darum, eine grobe Einteilung des parallelen bzw. verteilten Ablaufs einer Visualisierung vorzunehmen. Einzelheiten der Umsetzung des Konzepts betreffend Programmaufbau, Visualisierungsalgorithmen, Kommunikationsstrukturen und parallele Anwendungen werden in den folgenden Kapitel dargelegt.
Abbildung 2.4 zeigt das Schema einer Architektur für die Visualisierung. Das Schema folgt im wesentlichen dem in Kapitel 1.3 angesprochenen allgemeinen Ablauf des Visualisierungsprozesses (vgl. Abbildung 1.1), dessen einzelne Phasen durch die Programmkomponenten Datenimport, Gittertransformation, Geometrie–Mapping, und Rendering widergespiegelt werden.

Datenimport
Der Import der Daten aus einer FE–Analyse in das Programmsystem erfolgt entweder durch Leseroutinen oder durch direktes Empfangen von einem fremden Prozeß, mit dem über Sockets eine Verbindung besteht.
Datensätze können entweder sequentiell vorliegen oder parallel. Für den parallelen Fall bedeutet das, daß jeder Prozessor simultan mit den übrigen Prozessoren seine individuellen Daten einliest bzw. empfängt. Wo Datensätze nur sequentiell vorliegen, können diese auch mit Hilfe geeigneter Distributionsalgorithmen von einem Prozessor aufgespalten und zur parallelen Weiterverarbeitung an die anderen beteiligten Parallelprozessoren umverteilt werden.

Gittertransformation
Der nächste Schritt ist die Gittertransformation, in der auf selektiven Daten, die durch Reduktion oder Anreicherung erhalten wurden, Algorithmen angesetzt werden, die das für die Visualisierung repräsentative Gitter erzeugen.
Die Verarbeitung erfolgt wiederum sequentiell oder parallel, wobei erforderliche Information zwischen den Prozesoren via Message Passing ausgetauscht wird.

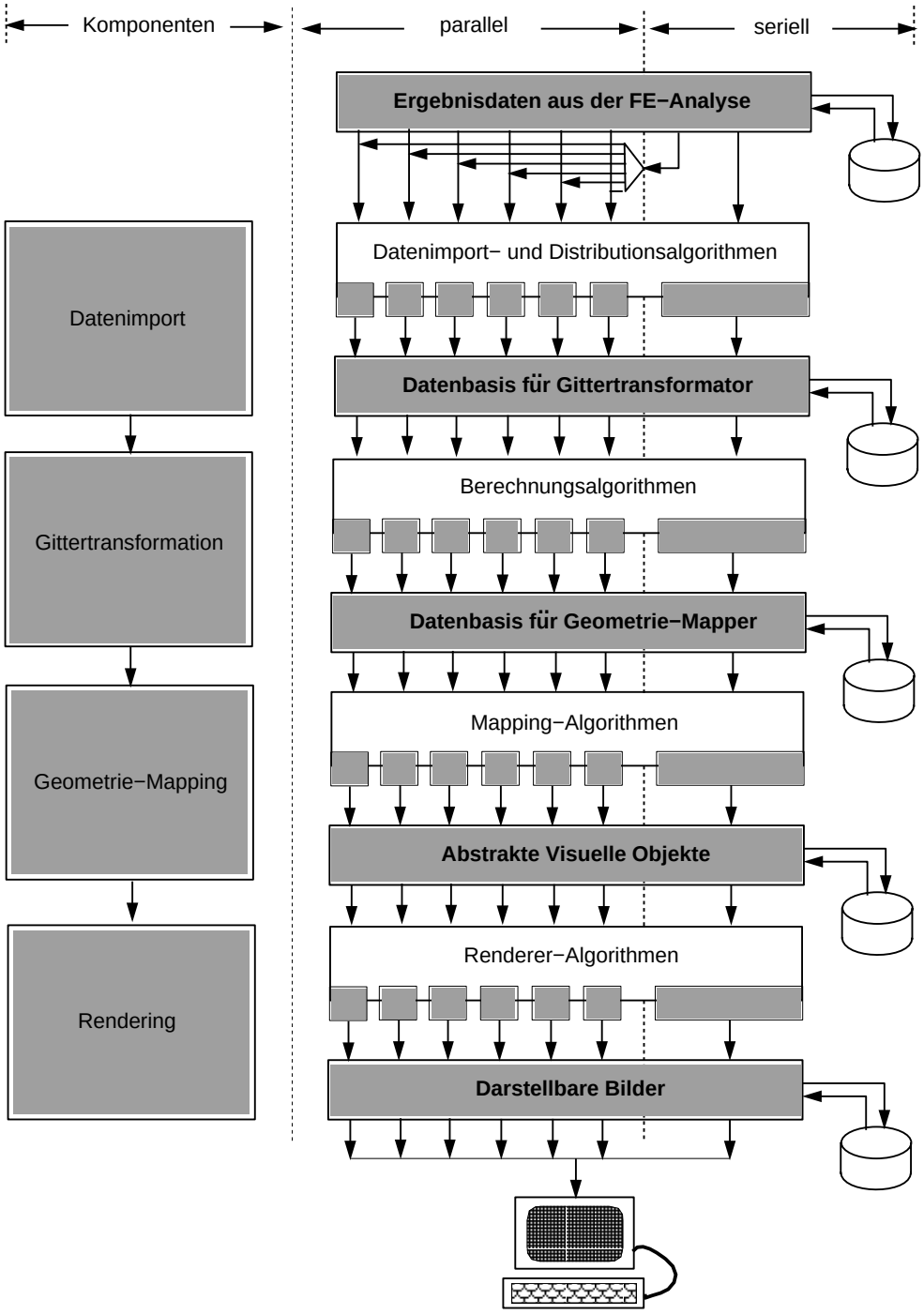


Abb. 2.4: Die Visualisierungsarchitektur

Geometrie–Mapping

Das sich anschließende Geometrie–Mapping kann im Prinzip parallel erfolgen, wenn auf jedem der Parallelprozessoren die für die Erzeugung eines abstrakten Visuellen Objekts bzw. Teilobjekts erforderliche Graphikbibliothek zur Verfügung steht. Für

das sequentielle Mapping hingegen muß vor dem Schritt eine Zusammenführung der Daten der einzelnen Teilobjekte erfolgen.

Rendering
Abschließend folgt das Rendering der AVOs, die eigentliche Darstellung auf einem Bildschirm. Da heute noch keine Methoden für paralleles Rendern zur Verfügung stehen, wird die Paralleltät sich hier darauf beschränken müssen, daß jeder Parallelprozessor seine Teil-AVOs an den Renderer schickt, der sie nacheinander empfängt und sequentiell darstellt.
Entsprechendes gilt für ein verteiltes Rechnen, infolge dessen verschiedene AVOs auf verschiedenen Dienstleistungsrechnern berechnet wurden.
Im Übrigen sei gesagt, daß für jeden der vier Schritte Zwischenergebnisse zum Zwecke einer späteren Weiterverarbeitung zwischengespeichert werden können.

Beschränkungen
In dieser Arbeit beschränkt sich das verteilte Rechnen auf den Zugriff auf einen einzigen fernen Rechner, einen von der Graphikworkstation getrennten Parallelrechner, auf dem diverse Jobs ausgeführt (*remote job execution*) und Daten hin– und hertransportiert (*remote data transport*) werden müssen.
Außerdem schließt eine Parallelisierung die Schritte Geometrie–Mapping und Rendering aus, dies u.a. deswegen, weil AVS nur auf einer einzigen seriellen Workstation zur Verfügung stand.

Kapitel 3

Aufbau des integrierten Visualisierungssystems

Das im Rahmen dieser Arbeit konzipierte und entwickelte multifunktionale Visualisierungssystem erlaubt es dem Wissenschaftler, viele verschiedene Anwendungen unter ein und derselben graphischen Benutzeroberfläche nebeneinander abzuarbeiten [7, 72]. So wird neben dem üblichen rein seriellen graphischen Pre- und Postprozessing sowohl eine parallele Berechnung der für die Visualisierung relevanten Gitterdaten (Gittertransformation) ermöglicht, als auch die direkte Kopplung mit einer seriellen oder parallelen Simulationsrechnung, was die Visualisierung während des Ablaufs einer CFD–Analyse [84] gestattet.
Das Visualisierungsprogrammsystem deckt demnach verschiedene Arten der Anwendung ab, für die der Anwender jedoch nicht verschiedene Systeme zu erlernen und zu bedienen hat. Vielmehr kann er sich seine Anwendung bedürfnisgerecht zusammenstellen, individuell und zentral aus einer graphischen Benutzeroberfläche heraus. Dazu steht eine Vielzahl von Softwarebausteinen bzw. Moduln zur Verfügung, was durch die Verknüpfung des Systems mit dem *Application Builder* AVS [45] gewährleistet ist, auf den in Abschnitt 3.2 näher eingegangen wird. Das Kürzel AVS steht für *Application Visualization System.*
Weiterer Vorteil der Einbindung des Systems in einen kommerziellen Application Builder ist die Verwendung eines für eine Hochleistungs–Graphik–Workstation optimierten Hardware–Renderers, sowie die Möglichkeit, seitens AVS zur Verfügung gestellte Softwarekomponenten mit den hier entwickelten in Anwendungen zu kombinieren.
Ein besonderer Akzent wurde bei der Konzipierung des Systems auf die Interak-

tivität gelegt, was neben der Zugänglichkeit einzelner Phasen der Visualisierung (insbesondere der Gittertransformation und dem Geomtrie–Mapping) implizierte, daß alle Berechnungen in einem für den Anwender verträglichen Zeitrahmen zu erfolgen haben.
Obwohl Aufgaben, wie etwa die Steuerung des Gesamtsystems, die interaktive Kommunikation mit den einzelnen Moduln, sowie das Ansprechen und die Integration eines assoziierten Parallelrechners, in das Visualisierungssystem AVS eingegliedert sind, wurde dennoch darauf Wert gelegt, die Programmstruktur so zu gestalten, daß sich AVS–spezifischer Programmcode leicht von AVS–unabhängigem herauslösen läßt. Damit wird eine Einbindung der Bausteine der Visualisierungssoftware in andere Oberflächen erleichtert, z.B. deren Verwaltung und Steuerung durch eine Oberfläche, die auf Motiv und X–Windows [70] basiert, einem Unterfangen, das jedoch nicht Gegenstand dieser Arbeit sein konnte.

3.1 Bausteine des Systems

Wie bereits erwähnt besteht das Programmsystem aus zahlreichen modularen Komponenten, die sich grob sechs Hauptaufgabenbereichen zurechnen lassen: einer **Gittertransformation**, dem **Geometrie–Mapping**, dem **Rendering**, einer allgemeinen **Datenhandhabung**, einer **Schnittstelle zur Einbindung von Simulationsprogrammen** sowie einer **Benutzeroberfläche** (Abb. 3.1).
Bedingt durch die Struktur des AVS ist letztere in die einzelnen AVS–Moduln integriert, d.h. letztendlich geschieht die Verwaltung und Steuerung des Systems mittels der in der AVS–Umgebung entwickelten Moduln.
Die Visualisierung selbst wurde in zwei voneinander unabhängige Teilgebiete untergliedert, einen Gittertransformator und einen Geometrie–Mapper und Renderer. Die Aufgabe des Geometrie–Mappers ist es u.a., unter Verwendung einer Graphikbibliothek die Ergebnisse aus der Gitterberechnung in geometrische Primitive zu mappen und abstrakte graphische Objekte zu erzeugen, während der Renderer die Darstellung und die Manipulation der Objekte am Bildschirm übernimmt. Hiervon entkoppelt ist die Berechnung des für die Visualisierung relevanten Gitters, mit der der Gittertransformator betraut ist. Dadurch ist sichergestellt, daß die Berechnungen im Gittertransformator unabhängig von jeder Graphikbibliothek, auf leistungsstarken seriellen oder parallelen Rechnern, entweder autark oder integriert in ein FE–Analysepaket, erfolgen können. Darüber hinaus ist eine Kopplung an fremde Renderersysteme zum reinen Mappen und Rendern möglich.
Die Datenhandhabung schließlich umfaßt, neben diversen Ein- und Ausgabeformaten, Kommunikationsroutinen für verschiedene Parallelrechnerarchitekturen, sowie

eine Software zur Regelung des Datenaustausches, wenn verteilt auf mehrere Rechenknoten im Internet (WAN) oder einem lokalen Netzwerk (LAN) gerechnet wird. Erwähnt sei in diesem Zusammenhang die Möglichkeit der Einbindung fremder Softwarekomponenten wie etwa eines FE–Analyseprogramms [84], eines Netzgenerators [66] oder eines Fremd–Renderer. Deren Steuerung ist allerdings bisher nur in Grundzügen verwirklicht worden, nämlich soweit, wie es für die Ankopplung der Visualisierung unerläßlich ist.
Im Folgenden sollen Aufgaben und Funktionsweise der Hauptkomponenten und ihr Zusammenwirken mit anderen Komponenten näher erläutert werden. Abbildung 3.1 gibt eine erste Übersicht über die Hauptkomponenten des Systems, deren wichtigste Unterkomponenten sowie die damit assoziierten Begriffe und Zuständigkeiten.

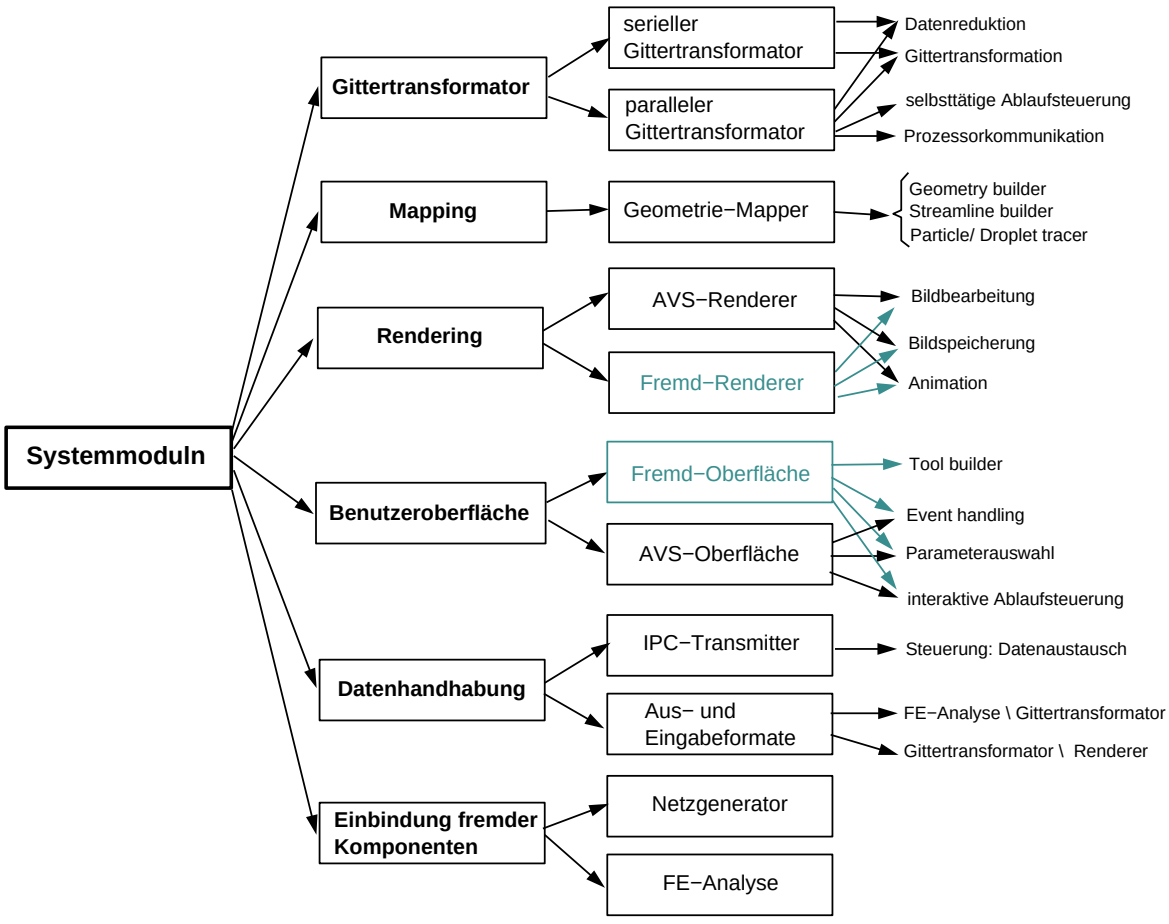


Abb. 3.1: Komponenten des parallelen, interaktiven, integrierten Visualisierungsprogrammsystems

3.1.1 Die integrierte Benutzeroberfläche

3.1.1.1 Erstellung individueller Anwendungen

Eine primäre Benutzeroberfläche ist bereits durch das AVS selbst gegeben. Diese übergeordnete Schicht soll hier jedoch nicht weiter beschrieben werden. Vielmehr ist diejenige untergeordnete Oberfläche von Bedeutung, die sich der Anwender selbst, durch Gruppieren von Moduln zu einem Modulnetzwerk, individuell erzeugt.
Jedes AVS–Modul, bzw. jedes in AVS eingebundene Modul, enthält einen Teil einer Oberfläche als ein Set aus Reglern, Schiebern und anderen interaktiven Eingabegeräten, das hier als ein Untermenü des Modulnetzwerks bezeichnet werden soll. Die höchste Schicht dieser individuellen Oberfläche ist das Modulnetzwerk selbst, in dem jedes einzelne Modul gesondert ansteuerbar ist. Die Namen der Moduln wurden so gewählt, daß Zweck und Ablauf der Anwendung aus den Namen ersichtlich werden.
Die Bezeichnung "integrierte Benutzeroberfläche" bezieht sich also zum einen auf ihre Integration in AVS, zum anderen und hauptsächlich jedoch auf die integrierende Funktion, die ihr zufällt: die gekoppelten Anwendungen unter einer Schale zu beherbergen, diese zentral zu verwalten und ihren Ablauf zu steuern.

3.1.1.2 Generelle Aufgaben einer individuellen Benutzeroberfläche

Je nach individuell komponierter Anwendung ist die Benutzeroberfläche mit verschiedenen Aufgaben betraut, die sich mit den Begriffen Initialisierung, Steuerung, Datenflußregelung, Datenauswahl und Parameterwahl umschreiben lassen.

- **Initialisierung:** Die Initialisierung erfolgt interaktiv aus der Oberfläche heraus. Sowohl beteiligte entfernte Rechner werden initialisiert bzw. aktiviert, als auch Programme, die für die Abwicklung des Datenaustausches über das Internet verantwortlich sind, ferner die Anwendung selbst, oder Teilbereiche davon, sofern sie unter direkter AVS–Verwaltung stehen.
- **Steuerung:** Die Steuerung des Ablaufs einer Anwendung oder einzelner Komponenten derselben verläuft ebenfalls interaktiv oder aber halbautomatisch. Der interaktiven Steuerung obliegt z.B. das wiederholte Ausführen von Programmsequenzen unter veränderten Bedingungen (Parameterauswahl).
  Interaktive Steuerung beinhaltet aber auch, daß man, um eine aus mehreren AVS–Moduln bestehende Anwendung einmal vollständig abarbeiten zu lassen, jedes der Moduln separat ansprechen muß, und zwar nacheinander in der

Reihenfolge, wie sie die Hierarchie des Netzwerkes vorschreibt. Anders ausgedrückt heißt das, daß die Abarbeitung der Richtung des Datenflusses folgt, wobei hier übergeordnete Moduln ihre Daten nachgeordneten Moduln via verbundene Anschlüsse (Modulports) übertragen.
Das System wurde so konzipiert, daß jedes Modul auf einen interaktiven Startimpuls des Benutzers wartet, um es zu initialisieren (Startbit).
Um jedoch ein nachgeordnetes Modul halbautomatisch auf Basis seiner aktuellen Parameter abarbeiten lassen zu können, ohne es interaktiv aktivieren zu müssen, wurde ein Überbrückungsparameter eingeführt. Dieser Parameter wird in einem hierarchisch übergeordneten Modul gesetzt, und über die verbundenen Modulports an das nachgeordnete weitergereicht, wo es eine Umgehung des Startbits bewirkt.
Ebenfalls halbautomatisch wird ein assoziierter Parallelrechner gesteuert, der nach Empfang seines Auftrages selbständig arbeitet, jedoch an bestimmten Schnittstellen im Programm für eine vorgegebene Zeitspanne auf Anschlußaufträge wartet. Fehlen weitere Anweisungen, reagiert das Parallelprogramm nach voreingestellten Werten (Defaults), die in einer gesonderten Datei, für den Anwender jederzeit änderbar, abgelegt sind.

- **Datenflußregelung:** Einerseits wird die Datenflußregelung dort, wo verschiedene Moduln in einem Modulnetzwerk über ihre Modulports miteinander verbunden sind, AVS–intern abgewickelt. Andererseits werden Regelungen, die etwa das Einlesen von Daten, ihre End- oder Zwischenlagerung auf Speichermedien, sowie ihren Transport über das lokale Netzwerk bzw. das Internet betreffen, interaktiv von speziellen, unter AVS entwickelten Moduln übernommen.
- **Datenauswahl:** Eine Datenauswahl erfolgt unter Gesichtspunkten der Datenreduktion und natürlich der Art der Darstellung. Letzteres ist beispielsweise durch die Auswahl einer darzustellenden physikalischen Größe aus vielen möglichen gegeben oder in der Beschränkung der Darstellung auf zwei Dimensionen. Eine Datenreduktion ist bereits zum Zeitpunkt der Dateneingabe möglich, sei es durch Einlesen von selektierten Daten oder durch die direkte Übernahme nur ausgewählter Ergebnisdaten aus einem angekoppelten Analyseprogrammsystem.
- **Parameterwahl:** Eine Vielzahl von Parametern und Steuergrößen, die alle genannten Bereiche betreffen, werden vom Benutzer gehandhabt.

3.1.1.3 Steuerung integrierter Anwendungen mittels Benutzeroberfläche

Als Beispiel für eine individuelle Benutzeroberfläche soll an dieser Stelle ein Netzwerk dienen, das serielle und parallele Komponenten integriert und somit eine Reihe von Anwendungen ermöglicht. In Abbildung 3.2 ist vereinfacht das Schema für diese in einem Netzwerk zusammengefaßten Anwendungen dargestellt. Programmeinheiten, die auf dem Parallelrechner betrieben werden, sind nicht in die graphische Benutzeroberfläche integriert.

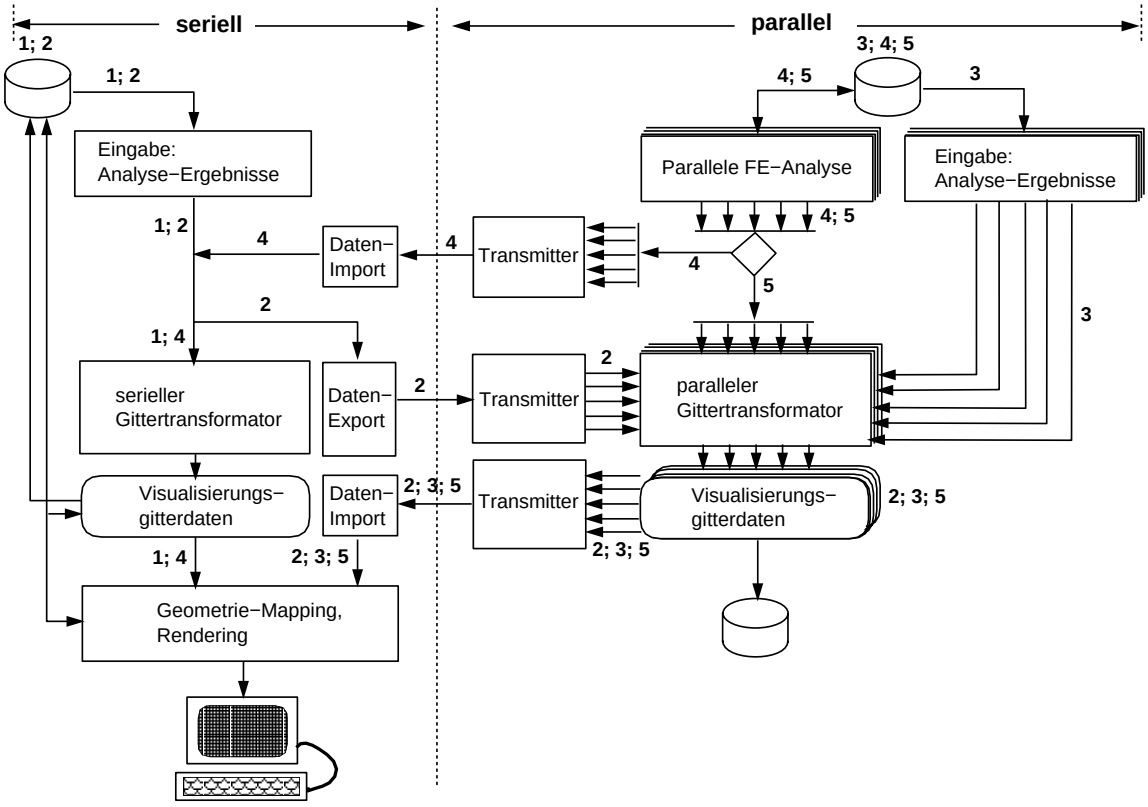


Abb. 3.2: Komponenten des parallelen, interaktiven und integrierten Visualisierungsprogrammsystems in einer Anwendung

Um die Abläufe besser nachvollziehbar zu machen, sind die einzelnen Anwendungen an Hand ihres Datenflusses jeweils durch Ziffern gekennzeichnet wie folgt:

- **serielle Visualisierung (1):** Auf der Basis sequentiell vorliegender Analyseergebnisdaten wird das Visualisierungsgitter vollständig sequentiell berechnet gemappt und gerendert (Standard–Graphik).
- **parallelrechnerunterstützte Visualisierung (2):** Sequentiell vorliegende Analyseergebnisdaten werden eingelesen, zerlegt und zur parallelen Gitter-

transformation auf einen assoziierten Parallelrechner übertragen. Anschließend werden die erhaltenen Gitterdaten an den seriellen Rechner zum Geometrie–Mapping und Rendern zurückübertragen.

- **parallele Visualisierung (3):** Der Parallelrechner liest verteilt vorliegende Analyseergebnisdaten ein, das Visualisierungsgitter wird parallel berechnet, und an den seriellen Rechner zum Geometrie–Mapping und Rendern übertragen.
- **semiparallele Simulation (4):** Die parallele FE–Analyse überträgt, auf Anfrage oder auch fortwährend, Analyseergebnisse direkt an den seriellen Gittertransformator. Das seriell erzeugte Visualisierugnsgitter wird seriell gemappt und gerendert.
- **parallele Simulation (5):** Der in eine parallele FE–Analyse integrierte parallele Gittertransformator überträgt, auf Anfrage oder fortwährend, Graphikergebnisse direkt an den seriellen Rechner zum Mappen und Renderer.

Weitere Beispiele für komplette Anwendungen mit Einzelheiten, die Steuerung der Komponenten und des Ablauf betreffend, finden sich in Kapitel 6 und 7.

3.1.2 Der Gittertransformator

Der Gittertransformator ist das Herzstück des Programmsystems, da hier die eigentliche Berechnung des Visualisierungsgitters vorgenommen wird. Der Gittertransformator, der unabhängig von jeder Graphikbibliothek arbeitet, versteht sich als ein modulares Paket, das sowohl aus seriellen und parallelisierten Transformationsalgorithmen besteht wie aus Routinen und Prozeduren für Internet- und Interprozessorkommunikation, Datenzerlegung bzw. Datenverteilung und Parametersteuerung. Kurz umrissen werden hier die Ergebnisdaten, wie sie die CFD–Analyse liefert, in Gitterdaten transformiert, die für die Visualisierung relevant sind. Zur endgültigen Darstellung (Erzeugung geometrischer Primitiven, Bildung abstrakter graphischer Objekte und Rendering) werden die Gitterdaten über eine Schnittstelle an den Geometrie–Mapper und Renderer weitergereicht.

3.1.2.1 Datenübertragung

Zwei Übertragungswege wurden festgelegt, um zum einen die Datenübertragung zwischen der CFD–Analyse und dem Gittertransformator zu gewährleisten und zum anderen den Datentransfer vom Gittertransformator an den Mapper und Renderer

sicherzustellen. Zwischen verschiedenen Rechnern erfolgt der Datenaustausch am schnellsten mittels der auf Sockets basierenden Interprozeßkommunikation (siehe Kap. 6), jedoch wird nach wie vor der konventionelle Weg über das Speichermedium Platte unterstützt, der beispielsweise unumgänglich ist, wenn Datensätze archiviert oder von Fremdsystemen übernommen werden sollen. Für diesen Zweck wurden spezielle Formate definiert, wie im Unterkapitel Datenhandhabung näher beschrieben wird.

3.1.2.2 Integration in parallele CFD–Analyse

Der Programmteil Gittertransformator gewährleistet einen von der CFD–Analyse und dem Geometrie–Mapper und Renderer völlig unabhängigen Betrieb. So kann der Gittertransformator gleichzeitig mit der CFD–Analyse auf demselben Rechner aber getrennten Prozessoren oder auf einem anderen Parallelrechner ausgeführt werden.
Daneben können die Moduln auch als Unterprogrammeinheiten von einer seriellen oder parallelen CFD–Analyse gerufen werden. Allerdings muß dann auf eine Neuaufteilung des Netzes verzichtet werden. Dies gilt selbst dann, wenn die optimale Auslastung der beteiligten Prozessoren für die Visualisierung nicht mehr gegeben sein sollte. Bei der Gittertransformation wird die CFD–Analyse vorübergehend in einen "Park–Modus" übergeführt und die entsprechenden Berechnungsroutinen von jedem der an der Simulationsrechnung beteiligten Prozessoren ausgeführt. Am Ende der Visualisierungsprozedur, z.B. wenn das Visualisierungsgitter zur Darstellung an einen anderen Rechner übertragen wurden, erfolgt dann die Reaktivierung der CFD–Analyse.
Ein wesentlicher Vorteil dieser Konfiguration besteht neben einem geringeren zusätzlichen Speicherbedarf darin, daß es keiner zeitraubenden Neuzerlegung des Netzes mehr bedarf. Es kann auf denselben Prozessoren weitergerechnet werden. Der Zeitverlust durch das Parken fällt indes nicht allzusehr ins Gewicht, bedenkt man, daß keine zusätzlichen Prozessoren für Visualisierungszwecke reserviert werden müssen, und der Analyse somit von vornherein eine höhere Anzahl Prozessoren zur Verfügung gestellt werden kann.

3.1.2.3 Steuerung der gekoppelten Systeme CFD–Analyse und Visualisierung

Betrachtet wird weiterhin der Fall, daß die Visualisierung mit einer CFD–Analyse gekoppelt auf einem Rechner betrieben wird. Die Steuerung der Visualisierung wird

von der CFD–Analyse übernommen, indem diese zu gewissen Zeitpunkten, beispielsweise nach jedem Zeitschritt, einen kleinen Parameterblock abfragt bzw. einliest.
Ist ein bestimmtes Visualisierungsbit gesetzt, wird der Prozeß des Gittertransformators in Gang gesetzt und durchgeführt. Die Initialisierungsroutine empfängt dann die weiteren Anweisungen entweder direkt über das Netz oder liest eine Parameterdatei, die sich der Anwender zuvor mit der Benutzeroberfläche erstellt hat. Ist keine solche Datei vorhanden, bzw. fehlen weitere Anweisungen (*timeout*), wird der Visualisierungsprozeß beendet und die FE–Simulationsrechnung wieder aufgenommen.
Der Parameterblock enthält Informationen über den Typ des Transformations- bzw. Berechnungsalgorithmus, ob zum Beispiel Isolinien berechnet werden sollen, Schnitte oder Stromlinien. Weiter sind unter anderem nähere Angaben zur Anzahl der Isolinien oder zur physikalischen Größe, die auf einem Schnitt dargestellt werden soll, Darstellungsbereich und vieles mehr enthalten.

3.1.2.4 Parallelisierung der Algorithmen des Gittertransformators

Abhängig von der Zerlegung der Daten des Ausgangsgitters werden vier Arten der Parallelisierung unterschieden. Parallelisiert werden kann entweder nach zusammenhängenden Gebieten (*domain decomposition*), nach elementbezogenen Daten, knotenbezogenen Daten oder nach Anzahl bestimmter Objekte oder Einheiten (z.B. Stromlinien).
Die Mehrzahl der Algorithmen hier basiert auf der Zerlegung nach elementbezogenen Daten. Zur Wahrung des Lastengleichgewichts wird jedem Prozessor möglichst dieselbe Zahl an Elementen zugewiesen (*load balancing*), wobei jedes Finite Element unabhängig von seinem Ort und seinen Nachbarelementen für sich bearbeitet werden kann, beispielsweise wenn Schnittflächen- oder Isoliniensegmente elementweise berechnet werden.
Die willkürliche Umverteilung der Elemente auf die Prozessoren geht jedoch mit dem Verlust lokal zusammenhängender Gebiete einher. Somit ist eine Erstellung lokaler Nachbarschafts- und Elementkantenlisten nicht mehr sinnvoll. Soll keine Neuaufteilung erfolgen, werden die Teilgebiete (*subdomains*) der CFD–Analyse übernommen, die ja wiederum aus einzelnen Elementen bestehen. Zu den elementbezogenen Algorithmen gehört die Berechnung von kontinuierlichen Darstellungen von Knoten– und Elementergebnissen, Isolinien, Konturflächen, Isoflächen sowie Schnittflächen.
Bei der Darstellung von Vektorfeldern werden knotenbezogene Daten auf die Prozessoren verteilt. Jeder Vektor greift an einem Gitterpunkt an, die Information der Verbindung der Knoten über Elemente wird nicht benötigt.
Eine weitere Methode ist die Aufteilung nach Anzahl von Objekten. Diese wurde bei der Stromlinienberechnung und dem Partikel Tracing verwendet. Jeder Prozessor

erhält das gesamte Strömungsgebiet und berechnet eine anteilige Anzahl von Stromlinien. Diese Methode versagt jedoch bei allzugroßen Strömungsproblemen, da die Speicherkapazität eines Prozessors begrenzt ist, und nicht mehr das Gesamtproblem halten kann.
Eine zusätzliche Möglichkeit ist die Stromlinienberechnung bzw. Partikelverfolgung auf der Grundlage von Gebietszerlegungen. Jeder Prozessor berechnet denjenigen Anteil einer Stromlinie, der durch sein Gebiet verläuft. Dabei müssen Informationen über den Ein- und Austritt der Stromlinien bzw. Partikeln in und aus dem Gebiet mit den angrenzenden Gebieten ausgetauscht werden.
Prinzipiell werden mit steigender Prozessorzahl die Gebiete immer zahlreicher, zugleich aber auch immer kleiner, d.h. der Kommunikationsaufwand steigt in unerwünschtem Maße. Es läßt sich eine Mindestgebietsgröße abschätzen, die nicht unterschritten werden sollte. Somit entsteht jedoch ein Überhang an Prozessoren.
Hier wäre eine Verbesserung zu erzielen, wenn die Methode der Gebietszerlegung mit der der Aufteilung nach Partikeln bzw. Stromlinien sinnvoll kombiniert, und überhängende Prozessoren mit anderen lastenausgleichenden Aufgaben betraut würden.

3.1.2.5 Datenaufteilung

Zu Beginn der Berechnung des Visualisierungsgitters wird jedem beteiligten Prozessor ein eigener Datensatz zugewiesen. Arbeitet der Gittertransformator auf denselben Prozessoren wie die CFD–Analyse, liegen die Daten bereits aufgeteilt vor. Im Falle einer Neuaufteilung jedoch empfängt ein Masterprozessor die gesamten Daten oder liest diese von einem Speichermedium ein. Dieser Prozessor nimmt dann auch die Datenzerlegung und die Umverteilung auf die übrigen Prozessoren vor.
Mit Hilfe von speziellen Kommunikationsroutinen können die Prozessoren untereinander Daten austauschen oder weiterleiten. Vor der Initialisierung der parallelen Transformationsalgorithmen muß die Datenzerlegung abgeschlossen sein. Während des Transformationsprozesses werden bei der Mehrzahl der Algorithmen keine Daten mehr zwischen Prozessoren ausgetauscht. Eine Ausnahme wird der Stromlinienalgorithmus sein, der mit Gebietszerlegung arbeitet.
Zum Ende der Berechnungen werden die erhaltenen Visualisierungsgitterdaten an den Geometrie–Mapper und Renderer übermittelt. Dabei werden die Daten entweder prozessorweise vom Masterprozessor eingesammelt, oder jeder Prozessor transferiert seine lokalen Daten direkt nach dem Prinzip des Token Ring.
Zur späteren Weiterverarbeitung oder Zwischenspeicherung besteht außerdem die Möglichkeit, jeden Prozessor seine eigene individuelle Ausgabedatei erzeugen zu lassen, bzw. den Master eine Gesamtdatei.

3.1.3 Der Geometrie–Mapper und Renderer

Die eigentliche Visualisierung wird im Geometrie–Mapper und Renderer vorgenommen und erfolgt in drei Schritten: Übernahme der von den Berechnungsalgorithmen des Gittertransformators gelieferten Ergebnisse, Weiterverarbeitung zu abstrakten visuellen Objekten (Geometrie–Mapping) und Darstellung am Bildschirm (Rendern). Entsprechend den drei Schritten waren Schnittstellen–Moduln, Visualisierungs–Moduln und Rendering–Moduln zu entwickeln. Allerdings konnte für das Rendering der AVS–eigene *geometry viewer* verwendet werden.

3.1.3.1 Schnittstellen–Moduln

In der seriellen Anwendung unter AVS werden die Moduln des Gittertransformators direkt mit den Visualisierungsmoduln des Geometrie–Mappers verknüpft. Das bedeutet, daß hier auf spezielle Schnittstellenmoduln verzichtet werden kann. Anders liegt der Fall in parallelen Anwendungen oder immer dann, wenn der Gittertransformator von AVS entkoppelt betrieben wird.
Für diese Anwendungen existieren Schnittstellenmoduln für verschiedene Belange, die alle die für die Visualisierung repräsentativen Gitterdaten einlesen bzw. über das Netz importieren. So liest eine Modulgruppe beispielsweise das allgemeine Polygondatenformat CVP, das von der Mehrzahl der seriellen und parallelisierten Transformationsalgorithmen erzeugt wird, während andere Moduln das STR–Format lesen, das eine Weg–Zeit–Beschreibung durch ein Vektorfeld wandernder Partikeln enthält (siehe auch Kapitel 3.1.4).
Moduln beider Gruppen können sowohl einzelne als auch mehrere Dateien verarbeiten. Letzteres geschieht über eine sogenannte Hauptdatei (*header file*, die die Namen aller zu lesenden Dateien enthält. Mehrere Dateien müssen etwa dann verarbeitet werden, wenn jeder Parallelprozessor seine individuelle Ergebnisdatei ausgibt.
Eine weitere Modulgruppe ermöglicht den direkten Datentransfer zwischen einzelnen beteiligten Komponenten über das Internet bzw. Intranet, so daß der Umweg über ein Speichermedium entfällt.

3.1.3.2 Visualisierungs–Moduln

Die Weiterverarbeitung der Visualisierungsgitterdaten unter Verwendung geometrischer Primitiven, sowie die Erzeugung eines oder mehrerer abstrakter visueller Objekte (AVO), bis hin zur Erstellung ganzer Animationszyklen, wird mit Hilfe von Graphikbibliotheksfunktionen in den Visualisierungsmoduln vorgenommen.

Als Standard–Visualisierungsmodul wurde für diese Belange der *geometry builder* entworfen, ein multifunktionales Modul, welches das am häufigsten verwendete Polygondatenformat (CVP) verarbeitet. Die Moduln *streamline builder*, *particle tracer* und *droplet tracer* hingegen verwerten die Ergebnisse aus der Berechnung von Stromlinien bzw. einer Teilchen- oder Tropfenverfolgung (STR–Format).
Mit Visualisierungs–Moduln nimmt der Benutzer direkt Einfluß auf die Auswahl und die Gestaltung der zu erzeugenden abstrakten visuellen Objekte. Einzelne Optionen erlauben beispielsweise die schattierte Darstellung oder Präsentation als Drahtmodell, die Erzeugung der für Beleuchtungseffekte erforderlichen Oberflächennormalen sowie deren Richtungsumkehrung, während durch das Heraufsetzen eines Objektzählers sich mehrere Objekte gleichen Namens darstellen lassen, ohne daß nachfolgende Objekte ihre Vorgänger überschreiben.
Sind Daten für mehrere Objekte vorhanden, können diese Objekte entweder einzeln, in ihrer Gesamtheit, aber auch in Form eines Animationszyklus betrachtet werden. Ferner besitzen die Moduln einen speziellen Datenausgang, an den das ebenfalls neuentwickelte Modul *colour table* angeschlossen werden kann, das zur Erstellung von Farbtafeln und Legenden verschiedener Ausprägungen dient. Die im AVS–Geometrieformat erzeugten AVOs werden schließlich einem der AVS–Standard–Renderer–Moduln, vorzugsweise dem *geometry viewer*, übergeben, der für die Darstellung der Objekte verantwortlich ist. Außerdem können AVOs zur Wiederverwendung in diesem speziellen AVS-Geometrieformat auf Festplatte gespeichert werden.

3.1.3.3 Einsatz auf einer Hochleistungs–Graphik–Workstation

Das Programmsystem Geometrie–Mapper und Renderer wird am geeignetsten auf einer Hochleistungs–Graphik–Workstation installiert, die bereits in die Hardware integrierte optimierte Darstellungstechniken wie Beleuchtung, Reflexion, Perspektive, Depthcueing und vieles mehr besitzt.
Obwohl hier das AVS eingesetzt wird, können prinzipiell auch andere Programmsysteme zugänglich gemacht werden. Dazu sind die Schnittstellen– und Visualisierungsmoduln entsprechend anzupassen.

3.1.4 Die Datenhandhabung

Die Datenhandhabung umfaßt allgemeine genormte Schnittstellenformate für das Ein- und Auslesen von Daten zwischen den einzelnen Programmkomponenten und interne Arbeitsformate, die die Form von Datenfeldern und Strukturen besitzen und

die programminterne Datenübertragung zwischen Moduln ermöglichen. Zudem sind interne Arbeitsformate am Datenaustausch zwischen Parallelprozessoren einerseits und zwischen Kommunikationsendpunkten im Internet andererseits beteiligt.

Die beiden wichtigsten Arbeitsformatgruppen betreffen den Gittertransformator. Die verschiedenen Lesemoduln oder das Analyseprogrammsystem selbst stellen die Eingangsdaten für den Gittertransformators im GT–Format (GT für *grid transformation*) zur Verfügung. Dieses Format, das der Gitterberechnung die Datenbasis liefert, findet auch innerhalb des Gittertransformators Verwendung, etwa bei Mehrfachberechnungen (Hintereinanderschalten zweier Moduln des Gittertransformators), wenn beispielsweise die Ergebnisse aus einer Schnittberechnung durch einen soliden Körper oder ein räumliches Strömungsgebiet als Ausgangsbasis für die Berechnung von Isolinien auf diesem Schnitt genommen werden.

Die zweite Arbeitsformatgruppe beschreibt die Schnittstelle zu den Moduln des Geometrie–Mappers und Renderers. Während das interne STR–Format (STR für *Stream*) alle Ergebnisse aus Berechnungen von Stromlinien und Partikelverfolgungen in Form von Weg–Zeit–Beschreibungen verwaltet, sind die Ergebnisse aus allen übrigen Berechnungen im CVP–Format (CVP für *Coordinates-Values-Polygons*) zusammengefaßt.

Äquivalent zu den internen Arbeitsformaten existieren, wie bereits ausgeführt, genormte Schnittstellenformate gleichen Namens für das Ein- und Auslesen von Ergebnissen verschiedener Berechnungsstadien. Datensätze können sowohl im ASCII–Format vorliegen als auch unformatiert (binär).
Im Gegensatz zu den genormten Formaten enthalten die internen Arbeitsformate jedoch zusätzlich zu den Gitterdaten einen reservierten Bereich für gewisse Steuergrößen, Parameter und Wegvariablen, die für nachfolgende Moduln (stromabwärts) bestimmt sind und diese über Ursprung, Art und Umfang der Daten in Kenntnis setzen, sowie Aufträge an sie vermitteln. Diese Zusatzdaten sind also für den reibungslosen Dialog zwischen Komponenten und Moduln verantwortlich. Das schließt das Erkennen von Fehlverbindungen von Moduln und die interaktive Weiterleitung der Fehlermeldungen an den Benutzer mit ein.

3.2 Die Software–Umgebung AVS

Als Basis für das entwickelte Programmsystem wurde eine Visualisierungssoftware gewählt, die sich durch ihre modulare Struktur und Portabilität auszeichnet und die für eine Weiterentwicklung geradezu prädestiniert ist: das "Application Visualization Systems" AVS [43, 82]. AVS wurde unter Verwendung der Graphikstandards X–Windows [67] und PHIGS [25, 57], sowie einer objektorientierten Version des C–Compilers für Graphikworkstations unter dem Betriebsystem UNIX entwickelt und ist auf allen gängigen Plattformen verfügbar.
Ausschlaggebend für die Wahl war zum einen der zur Verfügung stehende ausgereifte Renderer, und zum anderen der Netzwerk–Editor, das zentrale Werkzeug innerhalb von AVS, das eine separate graphische Oberfläche ist, unter der sich Programmbausteine aus Eigenentwicklungen mit von AVS bereitgestellten zu individuellen, ablauffähigen Anwendungen verknüpfen lassen, die als AVS–Netzwerke bezeichnet werden.
Mit Hilfe eines Netzwerk–Editors können Moduln aus einer Modulpalette ausgewählt, in einem Arbeitsfenster positioniert, visuell und somit logisch aneinander gekoppelt werden. Die Kontrolle der Modulparameter erfolgt über ein Netzwerk–Kontroll–Panel, das beim Erstellen des Netzwerks automatisch, entsprechend den eingefügten Moduln, miterzeugt wird.
Im übrigen wählt der Benutzer aus einer Vielzahl von Programmbausteinen nur diejenigen aus, die er speziell für seine Anwendung benötigt. Auf diese Weise entstehen individuelle Netzwerke, die man samt voreingestellter Modulparameter abspeichern lassen kann, um sie jederzeit abrufbereit zur Verfügung zu haben.
Ferner erlaubt die Modularisierung eigener Programme, die mit der Verwendung typischer AVS–interner Strukturen einhergeht, beispielsweise den interaktiven Zugang zu allen wesentlichen Schnittstellen einer Anwendung. Dadurch, daß die einzelnen Komponenten des Netzwerks unabhängig voneinander angesprochen werden können, ist ein Wiedereinstieg im Ablauf der Anwendung oder ein abschnittweises Abarbeiten, ja selbst eine Wiederholung einzelner Teilschritte mit veränderten Parametern jederzeit möglich.
Eine interaktive Modifikation einer Applikation wird durch Hinzunahme neuer Bausteine oder durch Eliminierung nicht mehr benötigter Komponenten erzielt, ganz nach den Erfordernissen des Benutzers.
Ein weiterer Punkt ist eine Datenverzweigung auf Basis der Verwaltung gemeinsamen Speicherplatzes (*shared memory*), die es erlaubt, daß nur einmal bereitgestellte Daten nebeneinander in verschiedenen Moduln verarbeitet werden, um sie z.B. zusammen mit bereits fertiggestellten Graphiken in unterschiedlichen Fenstern am Bildschirm darzustellen.

Für den Programmentwickler ist letzten Endes die Wiederverwendung bzw. die erleichterte Integration von Programmcode in einmal erstellte Strukturen interessant. Dazu steht eine Fülle von Bibliotheken und Routinen zur Entwicklung eigener Programmeinheiten bzw. Moduln zur Verfügung [43].
Eine Vielzahl der Manipulationsmöglichkeiten im Renderer ist von der Hardwareseite her geregelt, wie zum Beispiel geometrische Transformationen (verschieben, drehen, skalieren), Beleuchtung, Schattierung, Lichtreflexion, Antialiasing, Perspektive, Depthcueing, Front/Back Clipping, das Ein- und Ausblenden von Objekten und einiges mehr. Vorrichtungen betreffend die Ausgabe der Graphik auf Video–Recordern, Postscriptdruckern und Hardcopy–Geräten etc. sind vorhanden.

In den verbleibenden Abschnitten dieses Kapitels wird auf die modulare Grundstruktur des in AVS integrierten Visualisierungssystems eingegangen, der Bau eines Moduls erörtert, sowie die Erzeugung abstrakter visueller Objekte unter Verwendung spezieller Graphikbibliotheken behandelt werden.

3.2.1 Das modulare Konzept der Visualisierung innerhalb von AVS

Die Basis des entwickelten Visualisierungsprogrammsystems bildet eine abgeschlossene Modulwelt innerhalb von AVS, deren Abstraktion und Funktionsweise an dieser Stelle erläutert werden soll [6]. Wo die Berechnung des Visualisierungsgitters außerhalb von AVS erfolgt, wie das beispielsweise für parallele Anwendungen der Fall ist, übernehmen gesonderte Moduln die Aufgaben der Fremdrechnersteuerung sowie des Imports und Exports von Daten. Einzelheiten betreffend Datenaustausch und Steuerung einer parallelen Anwendung sind dem Kapitel 6 zu entnehmen.

Die Visualisierung unter AVS, wie sie für diese Arbeit relevant ist, läßt sich grob in drei sequentiell voneinander abhängige Bereiche fassen, die im vorigen Abschnitt bereits ausführlich vorgestellt wurden:

- Die Datenbereitstellung, bei der die Ausgangsdaten (FE–Gitter) eingelesen oder direkt empfangen werden und in ein einheitliches Format gebracht werden.
- Der Gittertransformator, der mit der Berechnung des Visualisierungsgitters betraut ist und seine Ergebnisdaten ebenfalls in ein einheitliches Format packt.
- Der Geometrie–Mapper einerseits, der mittels Graphikbibliotheksfunktionen die Erzeugung der AVOs vornimmt und andererseits natürlich der Renderer selbst.

Zu jedem Bereich existieren eine oder mehrere Modulgruppen, die hier nur stichworthaft aufgelistet werden:

- IO–Moduln zur Handhabung externer Schnittstellen (Formate) z.B. mit fremden Programmsystemen.
- GT–Moduln für den Gittertransformator
- GM–Moduln für das Geometrie–Mapping und das Rendering
- IE–Moduln für Import und Export von Daten über ein lokales bzw. entferntes Netzwerk
- CT–Moduln zur Ablaufsteuerung von Parallelanwendungen

Die Steuerung der Moduln obliegt dem Benutzer und erfolgt durch interaktive Eingabegeräte.

3.2.1.1 Der Aufbau eines Moduls

Das Modul ist der fundamentale Programmbaustein innerhalb von AVS. Moduln erhalten Input und generieren Output. Es wird zwischen Subroutine- und Coroutine–Moduln unterschieden. Während bei einem Subroutine–Modul der ausführende Programmteil (*computation function*) immer dann gerufen wird, wenn sich die Eingangsdaten oder ein Parameter geändert haben, startet ein Coroutine–Modul davon unabhängig durch einen AVS–internen Anstoß. Da die Mehrheit der Moduln Subroutine–Moduln sind, soll an dieser Stelle näher auf sie eingegangen werden.
Ein Subroutine–Modul gliedert sich grundsätzlich in einen beschreibenden und einen ausführenden Teil.
Im beschreibenden Teil (*description function*) ist alles definiert und festgelegt, was Aussehen und Gestaltung des Moduls betrifft: Der Name des Moduls, seine Anschlüsse für Ein- und Ausgang von Daten (*input ports, output ports*) sowie alle interaktiven Parameter (*widgets*). Ferner ruft die beschreibende Funktion die ausführende Funktion und übergibt ihr alle Definitionen in der Reihenfolge ihrer Spezifizierung.
Im ausführenden Teil des Moduls wird auf Basis der obigen Beschreibung die Berechnung d.h. die eigentliche Aufgabe des Moduls vorgenommen und die Daten für den vordefinierten Outputport erzeugt. Diese Daten werden in einem der von AVS unterstützten Datentypen formatiert. Neben den Primitiv–Datentypen wie *byte, integer, real* und *string* existieren die sogenannten Aggregat–Datentypen, wie *field, colourmap, geometry, unstructured cell data (UCD)* und andere [45].
In den unter AVS entwickelten Moduln wurden vornehmlich die Datentypen *geometry* und *field* verwendet. Obwohl für Field–Datentypen uniforme, rectilineare und

irreguläre Felder jeglicher Dimension in Frage kommen, wurden ausschließlich uniforme Felder der Dimension 1, 2 und 3 definiert. Der Speicherplatz wird diesen Feldern innerhalb der ausführenden Funktion dynamisch und abhängig von den Eingangsdaten und den gewählten Parametern zugewiesen. Der Speicherbedarf für weitere interne Felder wird hingegen abgeschätzt (*worst case*) und ebenfalls dynamisch allokiert.
Uniforme Felder dienen hierbei zur Vereinheitlichung der Schnittstellen zwischen CFD–Analyse und Gittertransformator einerseits, sowie zwischen Gittertransformator und Geometrie–Mapper und Renderer andererseits. Abstrakte Visuelle Objekte werden im Datentyp *geometry* erzeugt und können mit AVS–Standard–Renderermoduln wie beispielsweise dem *geometry viewer* weiterverarbeitet werden.

3.2.1.2 Datenübertragung zwischen AVS–Moduln und Ablaufsteuerung

Moduln können untereinander Daten austauschen, sobald zugehörige In- und Outputports visuell miteinander verbunden wurden. Den unter AVS neu entwickelten Moduln ist allen ein reservierter Speicherbereich in den obengenannten Ausgabe–Datenfeldern zu eigen, der Basisinformationen für Moduln stromabwärts enthält, wie spezielle Steuergrößen, Kennziffern und Kennungsparameter, die auch zur sekundären Steuerung dienen. Auf diese Weise können nachfolgende Moduln Kenntnis von der Herkunft und Art ihrer Daten erlangen, sowie von Besonderheiten und speziellen Aufgaben, die nicht ein weiteres Mal über einen interaktiven Parameter vermittelt werden sollen. Außerdem werden etwaige Fehlverbindungen erkannt und dem Benutzer zur Korrektur interaktiv über Messagetafeln mitgeteilt.
Mit jedem Anklicken eines Widgets ruft die *description function* automatisch die *computation function* [45], was in den Standard–AVS–Moduln eine komplette Ausführung des Moduls und der Nachgeordneten nach sich zieht.
Um diese Abarbeitungskette zu unterbinden, also Moduln nicht automatisch ausführen zu lassen, wenn immer ein Anstoßimpuls erfolgte, – sei es interaktiv, oder durch ein anderes Modul, das Daten schickt, – wurde jedes hier entwickelte Modul mit einem Startknopf *start* ausgerüstet, so daß nunmehr das Modul seinen Auftrag erst dann ausführt, wenn dieser Knopf betätigt wird. Der Benutzer kann also in Ruhe seine Modulparameter setzen und ändern, und das entsprechende Modul führt den eigentlichen Auftrag erst dann aus, wenn er es wünscht.
Der Widget *start through* wiederum umgeht den Widget *start* und läßt nachfolgende Moduln durchstarten, ohne daß sie erst interaktiv angestoßen werden müssen.

3.2.2 Erzeugung Abstrakter Visueller Objekte

Die hier verwendeten Abstrakten Visuellen Objekte bestehen aus Polygonen und geometrischen Primitiven und werden im AVS–Datenformat *geometry* erzeugt. Im Folgenden wird ein kurzer Blick auf die verwendeten Datenstrukturen in diesem Format geworfen. Eine ausführliche Beschreibung der Makros und Aufrufe der *geometry library* findet sich im AVS Developer's Guide [43].
Der Aufbau der Geometrie beginnt mit der Spezifikation eines Objekts. Aus den zur Verfügung stehenden Objekttypen wie *mesh, polyhedron, polytriangle, sphere* und *label* wurde hier – bis auf Farbtabellen, die als *label object* erzeugt werden, – der Objekttyp Polyhedron gewählt, dem sämtliche Polygonkoordinaten, die Farbwerte an den Polygonknoten, die zugehörigen Normalenrichtungen sowie die Vorschriften zur Bildung der Polygone (Konnektivität) und ein Name beigeordnet werden. Für jede dieser Zuordnungen stehen AVS–Makros zur Verfügung, die auf den Graphikbibliotheken PHIGS und Figaro aufsetzen.
Die Makrostrukturen für ein Objekt des Typs Polyhedron stellen sich wie folgt dar:

GEOMcreate_obj(GEOM_POLYHEDRON,GEOM_NULL)

GEOMadd_vertices_with_data
(obj,vert,GEOM_NULL,icol,nvert,GEOM_COPY_DATA)

GEOMadd_polygon(obj,nop,ipoly,GEOM_CONVEX,0)

GEOMgen_normals(obj,0)

GEOMcvt_polyh_to_polytri(obj,GEOM_SURFACE)

Zunächst wird das Objekt kreiert. In den folgenden Makros werden die Polygondaten spezifiziert. *Add_vertices* übernimmt das Koordinatenfeld (vert) und die Integerfarbwerte (icol), die zu jedem der *nvert* Knoten aus den RGB–Werten gemäß der nachstehenden Beziehung erhalten werden:

$$\begin{aligned} ir &= int(255 * r) \\ ig &= int(255 * g) \\ ib &= int(255 * b) \\ icol(i) &= (ir * 256 + ig) * 256 + ib \end{aligned} \tag{3.1}$$

Mit jedem Aufruf *add_polygon* wird dem Objekt ein einzelnes Polygon zugewiesen. *Nop* bedeutet die Anzahl der Knotenpunkte des Polygons, während in das Feld *ipoly* die Identitäten der Knoten, die das Polygon bilden, eingetragen sind.
Gen_normals generiert automatisch Normalen für das Objekt. *Cvt_polyh_to_polytri* schließlich ist eine Optimierungsfunktion für *polytriangle strips* mit der Option *SURFACE* oder *WIREFRAME* für schattierte Darstellung bzw. die Präsentation als Drahtmodell.
Das Polyhedronobjekt wird unter einem Objektnamen in eine *editlist* transferiert, über die es durch eines der Renderermoduln direkt auf dem Bildschirm darstellbar wird. Die Editlist ist ein besonderer AVS–Datentyp, der die Kommunikation zwischen einem geometrieproduzierenden Modul und dem AVS–Renderer–Modul *geometry viewer* bewerkstelligt. Die Makrostrukturen hierfür sehen folgendermaßen aus:

GEOMinit_edit_list(output)

GEOMedit_geometry(output,obj_name,obj)

GEOMdestroy_obj(obj)

Diese Makros initialisieren und erstellen für das erzeugte AVO eine Editlist und transferieren seine Ausgabedaten an den zugehörigen vordefinierten Outputport vom Datentyp *geometry* des Moduls. Das ursprüngliche Objekt vom Typ Polyhedron wird hiernach zur Freigabe des belegten Speicherplatzes eliminiert.

3.2.3 Ablaufschema einer Visualisierung unter AVS

Der Ablauf der Visualisierung, wie er sich in dieser Arbeit innerhalb von AVS dem Anwender darstellt, soll nun erörtert werden. Eine Visualisierung folgt im Prinzip der bereits in Abbildung 1.1 dargestellten Visualisierungspipeline.
Ein Modulnetzwerk für eine einfache Anwendung, die diesem Schema gehorcht, zeigt Abbildung 3.3. Mit den Moduln *nodal results* und *configuration* wird hier ein FE–Problem (Finite–Element–Gitter und Ergebnisse an den Gitterknoten) eingelesen, dessen Daten im Neutralformat von PATRAN vorliegen.
Genannte Moduln gehören zur Gruppe der für den Aufbau Datenbasis für die Visualisierung verantwortlichen Moduln. Mehrheitlich sind dies Lesemoduln für die Formate PATRAN, FEPS und MOVIE.BYU, aber auch Moduln, die Startpunkte für Stromlinien oder Referenzebenen für Schnittflächen erzeugen. Auf diese Weise aufbereitete Ergebnisdaten werden zusammen mit einigen Steuer- und Kennungspa-

rametern im AVS–Format *field*, das einer ganzen Palette von Moduln als genormte Schnittstelle dient, an die entsprechenden Outputports der Lesemoduln gebracht.

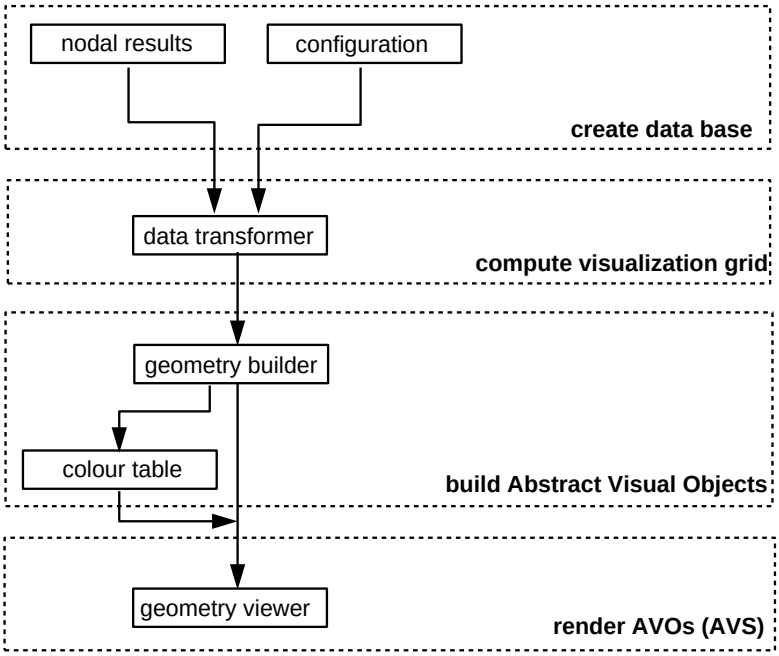


Abb. 3.3: Beispiel für ein Ablaufschema einer Visualisierung

Das Modul *data transformer* gehört einer Gruppe von Moduln an, deren Aufgabe es ist, das Gitter für die Visualisierung zu berechnen. Der Data-Transformer ist ein Vielzweckmodul, mit dem sich auf Basis der zur Verfügung gestellten Gitterdaten und Knoten- aber auch Elementergebnisdaten neben der einfachen Darstellung einer Variation der Ergebnisse auf dem Gitternetz auch Isolinien und Konturflächen berechnen lassen, sowie im dreidimensionalen Fall sogar Isoflächen. Eine Fülle von Optionen steht zur Wahl, angefangen von der Auswahl eines Ergebnisses über Wertebereichseinschränkungen bis hin zur Berechnung von ganzen Animationsfolgen.
Das zur Mappergruppe gehörende Modul *geometry builder* ist ebenso multifunktional in seiner Aufgabe der Verarbeitung der Ergebnisdaten aus der Gittertransformation in Abstrakte Visuelle Objekte (AVOs) verschiedener Ausprägungen, sowie der Erstellung von Animationszyklen.
Das Modul *colour table* dient zur Erstellung und individuellen Ausgestaltung von Farbtafeln und Wertetabellen. Anzahl der Farbwerte, Schriftgröße und Formatparameter wie Mantissenlänge, Gleitkomma- oder Exponentmodus und Tabellenmaße sind frei wählbar.
Das von AVS zur Verfügung gestellte Renderermodul *geometry viewer* schließlich übernimmt die Darstellung der AVOs in einem oder mehreren Fenstern am Bildschirm.

Kapitel 4

Entwicklung serieller Darstellungsalgorithmen

Vorbemerkungen

Bei der Berechnung des Visualisierungsgitters wird zunächst die vorgegebene geometrische Struktur der numerischen Simulation beibehalten, die sich aus Finiten Elementen oder Finiten Volumen zusammensetzt. Die Gitterstruktur spielt hierbei keine Rolle; sowohl strukturierte als auch völlig unstrukturierte Gitter sind mit den in diesem Kapitel vorgestellten Berechnungsmethoden handhabbar. Für die Darstellung wird die geometrische Struktur entweder direkt verarbeitet, wie das bei der Darstellung des Netzes bzw. von Elementgruppen der Fall ist, oder es werden Operationen auf Basis ihrer Information ausgeführt, beispielsweise für die Berechnung einer Schnittfläche durch räumliche Elemente. Auch bei einer Partikelverfolgung oder Stromlinienberechnung bedient man sich der Elementstruktur, um den Aufenthaltsort der Teilchen zu ermitteln und deren Geschwindigkeit aus den bekannten Daten ihrer unmittelbaren Umgebung zu interpolieren.

Die entwickelten Algorithmen beschränken sich auf ganz bestimmte Elementgrundtypen: Linienelemente, Dreiecks- und Viereckselemente im Zweidimensionalen, sowie Tetraeder, Pentaeder und Hexaeder im Dreidimensionalen. Mittelpunkte von Kanten, Seitenflächen und die Mittelpunkte des Raumes werden ignoriert, sofern keine Aufteilung in Unterelemente erfolgen soll (siehe Abschnitt 4.3.2.).

Außerdem ist es für die visuelle Bearbeitung der numerischen Ergebnisdaten Voraussetzung, daß diese entweder in den Knotenpunkten der Elemente ausgegeben werden oder aber über ein ganzes Element konstant bleiben. Die Berechnungsverfahren sind also indirekt. Grund dafür ist, daß verschiedene Analyseprogrammsysteme wie FI-

DAP, FLOW3D, STAR–CD über die verschiedensten Finite Element- bzw. Finite Volumen–Ansatzfunktionen linearer oder höherer Ordnung verfügen [35]. Diese Ansatzfunktionen arbeiten mit inneren Integrationspunkten, die dem Gittertransformator weder bekannt sind noch zur Verfügung stehen und die sogar bei Elementen gleichen Typs variieren können. Somit kann die Visualisierung unabhängig von jeglicher Diskretisierung der Gleichungen arbeiten. Direkte Verfahren, die die Kenntnis der Modellierung benötigen, werden von vorneherein ausgeschlossen.
Weiter wird in Hinblick auf eine spätere einfache Parallelisierung bei der Mehrheit der Algorithmen elementweise verfahren. Jedes Element wird für sich alleine betrachtet, ohne daß die Kenntnis seiner Nachbarn notwendig ist. Der Nachteil dieser Methode ist, daß entstehende Segmente und Teilflächen nur lose miteinander verbunden sind. Es entstehen mehrdeutige Koordinaten, die aufgrund nicht mittelbarer Normalen nur einfache Schattierungstechniken wie *flat shading* anstelle von *Gouraud-* oder *Phong-Shading* erlauben. Es wird dabei bewußt in Kauf genommen, daß die Oberfläche eines dreidimensionalen Körpers facettenförmig erscheint, da an den Knoten die Normaleninformation der Nachbarelemente unberücksichtigt bleibt. Abhilfe schaffen Algorithmen, die spezielle Kantenreferenzlisten verwenden. Deren Erstellung ist komplizierter und kostet zunächst einmal zusätzlichen Speicherplatz. Dies wird aber durch schnellere Algorithmen und geringeren Speicherbedarf für die Ausgabe der für die Visualisierung repräsentativen Gitterdaten wieder aufgewogen.
In den folgenden Abschnitten werden die grundlegenden Darstellungsalgorithmen entwickelt und an Hand von Anwendungsbeispielen erörtert. Einige Details und Wesensmerkmale, die die Handhabung und Leistung des Systems herausstellen, können dabei nur im Zusammenhang mit den Moduln erläutert werden, mit denen sie in einer Anwendung unmittelbar in Beziehung stehen. So ist z.B. für die Verfolgung von Partikeln ein Partikelgenerator und die Erzeugung einer Weg–Zeit–Beschreibung der Partikeln Grundlage und Voraussetzung.
Darüberhinaus arbeiten die Algorithmen, die ausschließlich die Berechnung des Visualisierungsgitters vornehmen (Gitterpunkte, Gitterelemente), unabhängig von jeder Graphikbibliothek, was für eine spätere Parallelisierung der Algorithmen sowie eine etwaige Vergabe der Darstellung an fremde Renderer bedeutsam ist.
Im wesentlichen werden folgende Algorithmen unterschieden: Kontinuierliche Darstellung (Fringe), Isolinien, Konturflächen, Isoflächen mit Datenmapping, beliebige Schnitte durch Körper, Darstellung von Volumenausschnitten, Vektoren, Stromlinien, die Verfolgung von Partikeln und von zusammenhängenden Flächen. Für eine Kombination verschiedener Visualisierungstechniken wurden interne Schnittstellen implementiert. Auf eine Untergliederung in zwei- und dreidimensionale Algorithmen wird verzichtet, da z.B. Stromlinienalgorithmen für beide Dimensionen ähnlich strukturiert sind und gemeinsam behandelt werden können.

4.1 Datenstrukturen

Die Datenstruktur, die dem ersten Schritt der Visualisierung, nämlich der Gittertransformation, zugrundeliegt, wurde eng an die Datenstruktur des FE–Analyseprogrammsystems FEPS [84] angelehnt, nicht zuletzt, um eine Kopplung beider Systeme zu ermöglichen. Dies ist beispielsweise der Fall, wenn eine Simulationsrechnung Visualisierungsroutinen auf Basis ihrer eigenen Daten ruft. Von Vorteil ist hier, daß keine Analysedaten kopiert werden müssen, was gerade bei parallelen Anwendungen wichtig ist, um Speicherplatz zu sparen.
Zusätzlich zu den Konfigurations- und Ergebnisdaten der Analyse benötigt der Gittertransformator eigene Behelfsfelder und, wie Abbildung 4.1 zeigt, Felder für die Ausgabe der Gitterdaten.

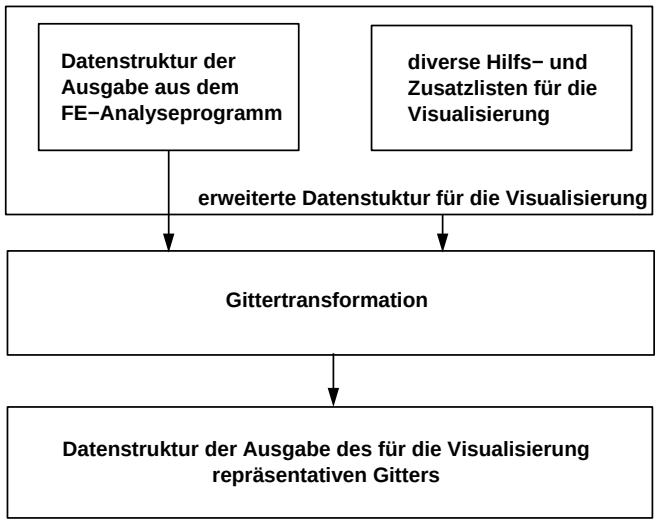


Abb. 4.1: Gittertransformation: Datenstrukturen für Ein- und Ausgabe

Die Datenstruktur des FE–Programms

Selbstverständlich ist die Datenstruktur des FE–Programmsystem ungleich komplexer als hier darstellt. Von Interesse ist jedoch nur eine reduzierte Datenstruktur, insofern sie die Daten enthält, die für die Visualisierung relevant sind.
Die reduzierte Datenstruktur, die auch als Exportdatenformat für fremde Programmsysteme dient, enthält im einzelnen Felder für die Knotenkoordinaten, die Elementtypen, die Knotenreferenzierung für jedes Element sowie die Simulationsergebnisse, die entweder in den Knoten ausgegeben werden oder in den Elementen.

Die wichtigsten Datenfelder:

- Knotenkoordinaten **{float coord(3,1),**
- Elementtyp **long ietyp(1),**
- Knotenreferenzierung **long ldat(ikn,1)),**
- Knotenergebnisse **float xval(ival,1),**
- Elementergebnisse **float eval(kval,1)}**

Die erweiterte Datenstruktur für die Gittertransformation
Die reduzierte Datenstruktur der Analyse muß für die Berechnung des Visualisierungsgitters um zusätzliche interne Listen ergänzt werden. Hierbei handelt es sich um Kantenknoten- Elementkanten-, Nachbarschaftslisten und andere Behelfsfelder, mit deren Aufbau und Funktion Abschnitt 4.2 dieses Kapitels befaßt ist.

Die Datenstruktur der Ausgabe des Visualisierungsgitters
Im Prinzip ähnelt die Datenstruktur der Ausgabe des Visualisierungsgitters der reduzierten Datenstruktur der Analyse. Es existieren zwei generelle Formate, die auch als Schnittstelle zur Außenwelt dienen, das CVP–Format und das STR–Format. Im CVP–Format ist alle Information abgelegt, die der Geometrie–Mapper benötigt, um aus dem berechneten Visualisierungsgitter ein oder mehrere Abstrakte Visuelle Objekte (AVO) zu erzeugen. Enthalten sind die Ergebnisse aus allen Algorithmen, die nichts mit Stromlinien oder Teilchenverfolgung gemein haben: Polygon–Knotenkoordinaten, Polygon–Knotenwerte und Polygon–Verknüpfungsvorschriften (Konnektivität) sowie die Objektzuordnung. Stromliniendaten sind gesondert im sogenannten Strom–Format STR abgelegt, das eine Weg–Zeit–Beschreibung einzelner Partikel enthält.

Das CVP–Format:

- Polygon–Knotenkoordinaten **{float vert(3,1),**
- Polygon–Knotenergebnisse **float xpol(kval,1),**
- Polygontyp **long ntyp(1),**
- Polygon–Knotenreferenzierung **long ndat(ikn,1),**
- Objektzugehörigkeit **long nobj(1)}**

Die Polygontypen stimmen nicht mehr mit den Elementtypen der Analyse überein. Die Typennummer gibt hier nur an, aus wievielen Knotenpunkten ein Polygon besteht. Die Objektzugehörigkeit legt fest, welchem Abstrakten Visuallen Objekt das Polygon angehören wird. Dies ist zum Beispiel bei Animationen wichtig, wenn für

jede Objektnummer genau ein Objekt erstellt und in einen Animationszyklus eingebunden wird.
Als weiteres Beispiel sei lediglich die Ermittlung eines Schnittes durch einen dreidimensionalen Körper angeführt, bei der zwei AVOs berechnet werden, die separat betrachtet werden können, einmal die Schnittfläche selbst und zum anderen der verbleibende Restkörper.

Das STR–Format:

- Koordinaten der Aufenthaltsorte **{float xw(3,1),**
- Zu den Aufenthaltsorten gehörige Zeitwerte **float xt(1),**
- Ergebnisse physikalischer Größen an den Orten **float xph(kval,1),**
- maximale Anzahl von Punkten je Stromlinie **long maxcor(1)}**

Der Speicherbedarf für die Ausgabe des Visualisierungsgitters wie auch für die Behelfsfelder wird erst zur Laufzeit festgelegt, der Speicherplatz dynamisch bereitgestellt. Hierbei wird der maximal zu erwartende Speicheraufwand in Abhängigkeit von der Art der gewählten Visualisierung und der Problemgröße abgeschätzt.

4.2 Behelfslisten für die Gittertransformation

Für die Berechnung des Visualisierungsgitters im Gittertransformator bedarf es über die aus der Analyse zur Verfügung stehenden Daten hinaus gewisser Behelfsdatenfelder und Listen. Nachbarschaftslisten erster und zweiter Ordnung sind beispielsweise für die Stromlinienberechnung erforderlich, wobei jeder neue Entwicklungspunkt einer entstehenden Stromlinie in der Nachbarschaft des unmittelbar zuvor aufgefundenen Punktes gesucht und die Geschwindigkeit am neuen Ort aus den Daten seiner Umgebung interpoliert wird.
Algorithmen zur Berechnung von Isolinien, Isoflächen, Schnitten und anderem greifen auf Kantenknoten- und Elementkantenlisten zurück, um bei der Polygonbildung zugehörige Punkte eindeutig zu erfassen. Dazu bedarf es einiger temporärer Behelfsfelder, um diese Listen aufzustellen.

Erstellung von Kantenknoten- und Elementkantenlisten
Die Erstellung einer Kantenknotenliste $kante()$ sowie einer Elementkantenliste $kael()$ wird hier am Beispiel des dreidimensionalen Falls aufgezeigt. Als zusätzliche temporäre Behelfsfelder werden ein Markierungsfeld $mark()$ benötigt, in dem bereits abgearbeitete Elemente gekennzeichnet werden, sowie die Listen $ip()$ und $list()$, in denen für jeden Elementknoten die Anzahl und Namen der angrenzenden Elemente

abgelegt werden. In Abbildung 4.2 ist vereinfacht der Algorithmus zur Aufstellung der beiden Kantenlisten dargelegt.

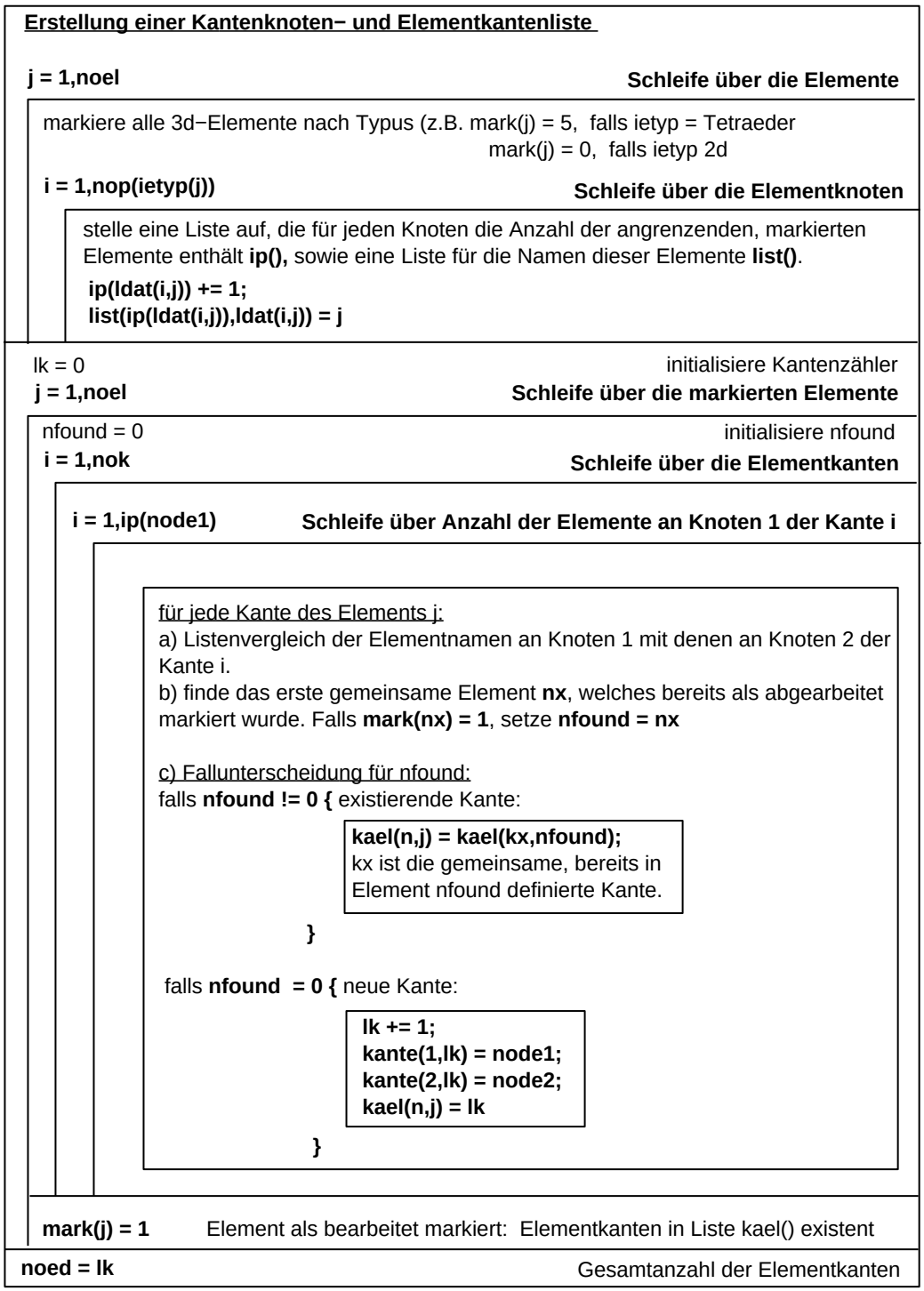


Abb. 4.2: Erstellung einer Kantenknoten- und einer Elementkantenliste

Erstellung von Nachbarschaftslisten

Bei der Generierung von Nachbarschaftslisten werden Nachbarn erster, zweiter und dritter Ordnung unterschieden.

Im zweidimensionalen Fall bezeichnet die erste Ordnung alle diejenigen Elemente, die mit dem betrachteten Element eine Kante gemein haben. Der zweiten Ordnung gehören alle Elemente an, denen mindestens ein Knoten gemein ist (Abb. 4.3). Eine höhere Ordnung existiert im zweidimensionalen Fall nicht.

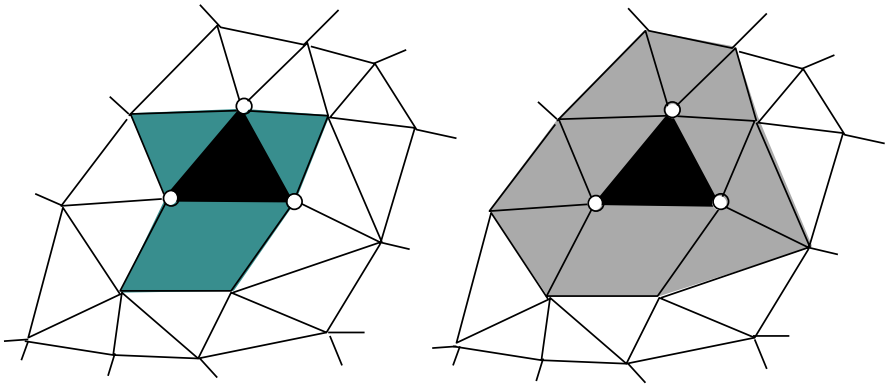


Abb. 4.3: Nachbarschaften erster und zweiter Ordnung um ein 2D–Element

Im dreidimensionalen Fall bezeichnet die erste Ordnung diejenigen Elemente, die mit dem betrachteten Element eine Seitenfläche gemein haben. Die zweite Ordnung bezieht sich auf gemeinsame Kanten, während die dritte Ordnung gemeinsame Knoten meint. Nachbarschaften höherer Ordnung schließen Nachbarschaften niederer Ordnung ein, d.h Nachbarschaften niederer Ordnung bilden Untermengen.

$$N_1 \subset N_2 \subset N_3 \tag{4.1}$$

Bei der Erstellung von Nachbarschaftslisten wird ähnlich verfahren wie zuvor bei den Kantenlisten. In einem ersten Schritt werden zu jedem Knoten alle an ihn grenzenden Elemente ermittelt und in einer Behelfsknotenliste abgelegt. In einem zweiten Schritt wird, abhängig von Dimension und Ordnung, die Nachbarschaftsliste erstellt, durch elementweises Vergleichen der Behelfsknotenlisten an den Elementknoten.

Je nach Ordnung der Nachbarschaftsliste werden verschiedene Kriterien angewendet. So werden zum Beispiel für eine Nachbarschaftsliste höchster Ordnung alle Elemente, die mehrfach genannt werden, eliminiert, während bei geforderter Kantengemeinsamkeit hingegen gerade diese Elemente ausgewählt werden. Bei gemeinsamen Flächen letztendlich muß ein Element mindestens in drei Behelfsknotenlisten vertreten sein, um in die Nachbarschaftsliste dieser Ordnung aufgenommen zu werden.

4.3 Serielle Algorithmen für zwei- und dreidimensionale Darstellung

4.3.1 Kontinuierliche Darstellung des Farbverlaufs (Fringe-Darstellung)

Die Fringe-Darstellung ist die Grundform der graphischen Darstellung, da sie ohne Transformation der Ergebnisse auskommt. Ein Fringeplot wird erhalten, wenn eine aus einem Ergebnisvektor ausgewählte Größe auf die zugehörige Finite-Element-Struktur abgebildet wird.
Das Verfahren der Darstellung richtet sich nach der Art der Ergebnisdaten. Ergebnisse aus einer FE-Analyse können knotenbezogen vorliegen oder elementbezogen, wobei knotenbezogen bedeutet, daß zu jedem Knoten der Struktur ein Ergebnisvektor existiert, im Gegensatz zu elementbezogenen Ergebnissen, die über das gesamte Element konstant sind.

- **Darstellung knotenbezogener Daten:**
  es wird so vorgegangen, daß jedes Finite Element graphisch durch einen oder mehrere geschlossene Polygonzüge dargestellt wird, deren Polygonknoten die zugehörigen Farbwerte besitzen, entsprechend den Ergebniswerten an den Knoten der Struktur (Abb. 4.4).

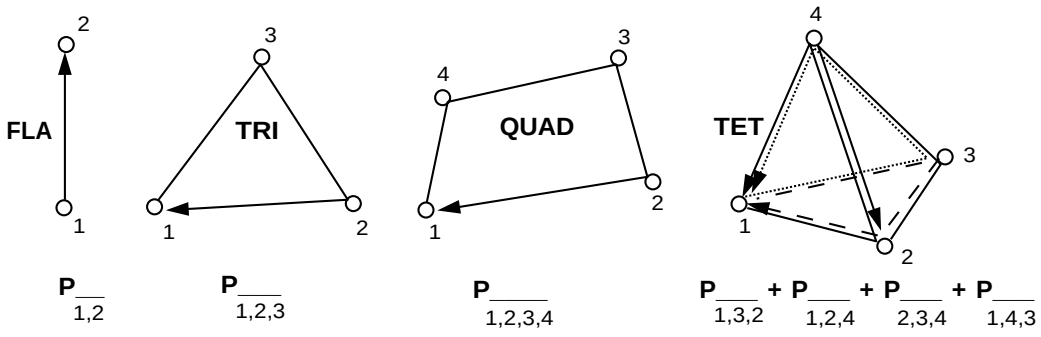


Abb. 4.4: Zur Darstellung Finiter Elemente mit Polygonzügen

Im Fringeplot erscheinen die Elemente der Struktur schattiert, d.h. das Ergebnis variiert über das Element als Funktion des Einflusses der Farbwerte an den Strukturknoten. Die Schattierung erfolgt mittels eines PHIGS-internen multilinearen Interpolationsverfahrens [25].
Unterschiedliche Darstellungsweisen wurden implementiert. Neben der schat-

tierten Darstellung kann die Struktur auch als Drahtmodell präsentiert werden. In diesem Fall werden nur die Kanten der Finiten Elemente gezeichnet, wobei entweder die Farbgebung entlang einer Kante variiert oder alle Kanten einfarbig wiedergegeben werden. Letzteres entspricht der reinen Strukturdarstellung ohne jede Ergebnisse.

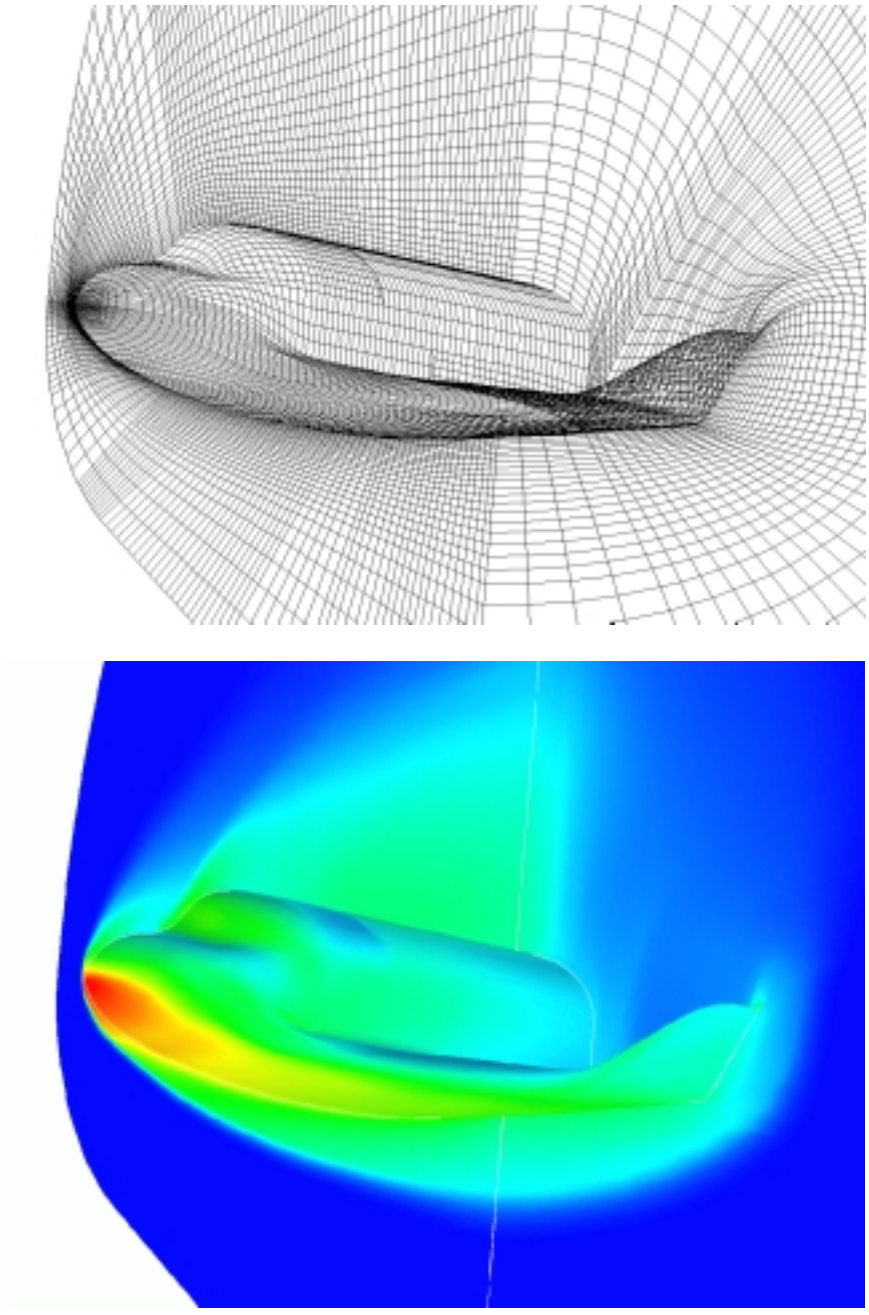


Abb. 4.5: Fringe-Darstellungsarten: Drahtmodell und Knotendaten

In Abbildung 4.5 sind beide Darstellungsweisen an Hand eines praxisorientierten Beispiels dargestellt. Während das obere Bild das Oberflächennetz für die 3D-Berechnung einer hypersonischen Strömung um den europäischen Raumgleiter Hermes zeigt, illustriert das untere Bild das kontinuierliche Temperaturprofil des chemisch reagierenden Mediums in 75 Kilometer Höhe bei einer Anströmmachzahl $Ma = 25$ [3, 4].

- **Darstellung elementbezogener Daten:**
  In der Darstellung elementbezogener Daten erscheint jedes Finite Element, entsprechend dem Ergebniswert im Element, einfarbig. Jeder Knoten eines Polygonzuges erhält denselben Farbwert. Da jedoch die Datenfelder für die Farbwerte der Knoten und deren Koordinaten intern derselben Datenstruktur zugeordnet sind, darf für jedes Koordinatentripel $coord(x, y, z)$ nur ein zugehöriger Farbwert fw existieren, was eine Vervielfältigung der Koordinaten erforderlich macht. Die Zahl der Vervielfältigungen jedes Knotens hängt von der Zahl der Elemente ab, denen der Knoten gemein ist.

  Darüber hinaus können auch Elementergebnisse schattiert dargestellt werden, indem sie auf die Knoten umgerechnet werden. Hierbei werden für jeden Knoten die Elementwerte w_{el}, der dem Knoten angehörigen Elemente aufsummiert und durch die Zahl der angrenzenden Elemente n_{el} dividiert, so daß sich im arithmetischen Mittel für die Knotenwerte w_{kn} ergibt:

$$w_{kn} = \frac{1}{n_{el}} * \sum_{k=1}^{n_{el}} w_{el_k} \tag{4.2}$$

  Der Anwender sollte sich allerdings darüber im Klaren sein, daß mit dieser Methode eine Glättung des Ergebnisses erkauft wird. So erfahren zum Beispiel Wertemaximum und -minimum nicht nur eine Nivellierung hinsichtlich ihres Betrages, sondern auch der Ort ihres Auftretens kann sich verschieben, mit Ausnahme der Orte, an denen alle dem Knoten zugehörigen Elemente denselben Maximalwert besitzen. Diese Darstellungsweise kommt ohne Vervielfältigung von Knotenkoordinaten aus.

4.3.2 Darstellung von Isolinien und Konturflächen

In zweidimensionalen Gebieten wird eine **Isolinie** als die Summe aller Orte definiert, an denen eine physikalische oder beliebige andere Größe einen konstanten Wert besitzt. Das gilt ebenso für nichteuklidische Flächen. So sind z.B. Isothermen Linien gleicher Temperatur.

Unter einer **Konturfläche** wird graphisch diejenige Fläche verstanden, die sich ergibt, wenn dem Bereich zwischen zwei Isolinien ein konstanter Farbwert zugewiesen wird. Durch die Darstellung mehrerer Konturflächen entsteht so ein abgestuftes Bild (siehe Abb. 4.6). Analytisch betrachtet, werden n diskreten Intervallen $[x_{i-1}, x_i]$ n diskrete Werte zugeordnet, beispielsweise stets der obere Grenzwert x_i des i-ten Intervalls oder der untere Grenzwert x_{i-1}, so daß für letzteren Fall gilt:

$$x = x_{i-1} \quad \forall \quad x \in [x_{i-1}, x_i] \qquad \text{mit} \qquad 1 \leq i \leq n \tag{4.3}$$

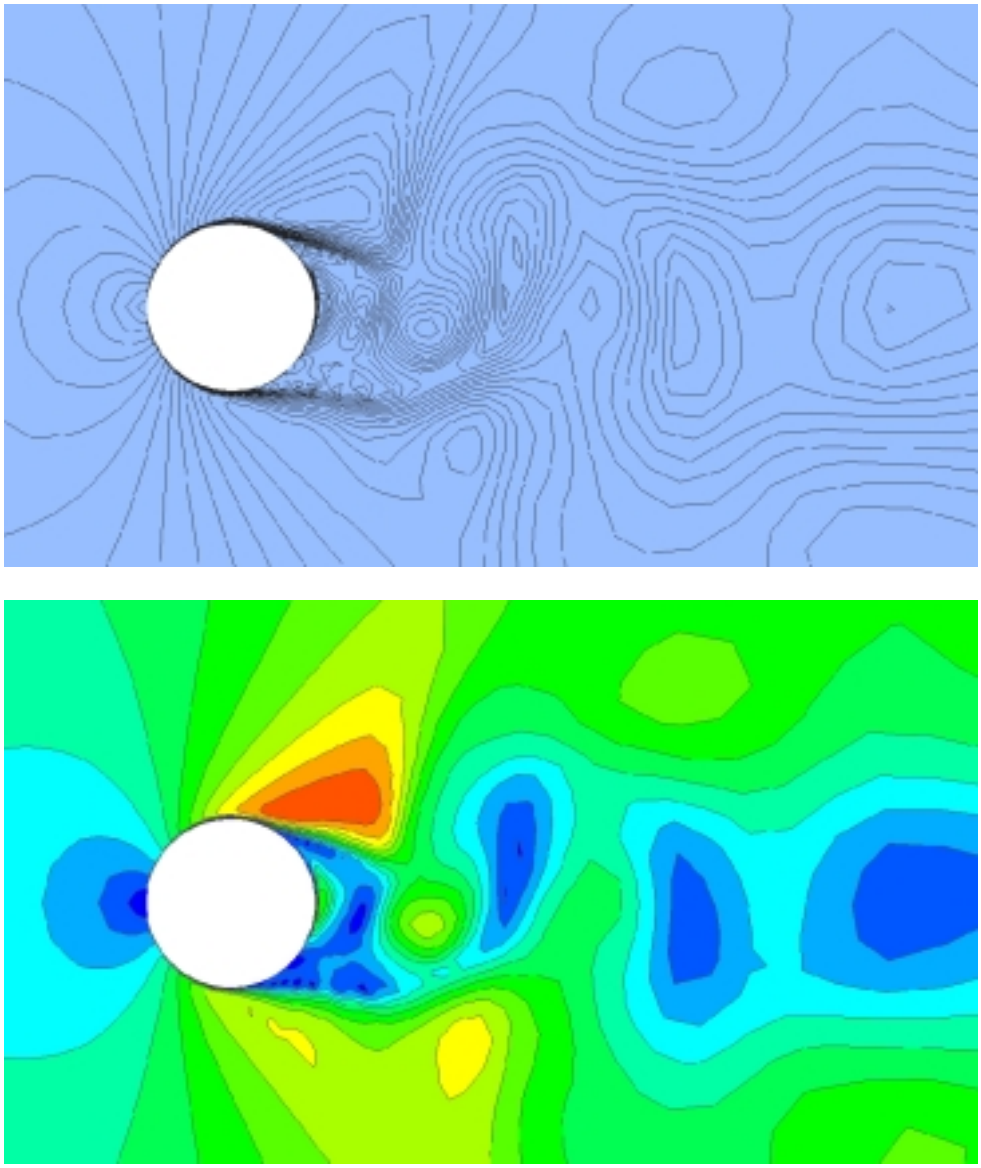


Abb. 4.6: Darstellung von Isolinien und Konturflächen der Geschwindigkeit in einer viskosen Strömung um einen Kreiszylinder (Karmann'sche Wirbelstraße, Re = 100)

4.3.2.1 Einschränkung des Wertebereichs und die Generierung diskreter Isowerte

Interaktiv werden die untere Grenze x_{min} und die obere Grenze x_{max} eines zu untersuchenden Wertebereichs festgelegt, in dem Isowerte betrachtet werden sollen. Isowerte können zum einen durch Vorgabe von Einzelwerten festgelegt werden, und zum anderen durch Generierung einer Wertesequenz nach linearen oder exponentiellen Verfahren. Die Generierung erfolgt entweder durch Vorgabe eines festen Inkrements oder durch Vorgabe einer Anzahl (Abb. 4.7).

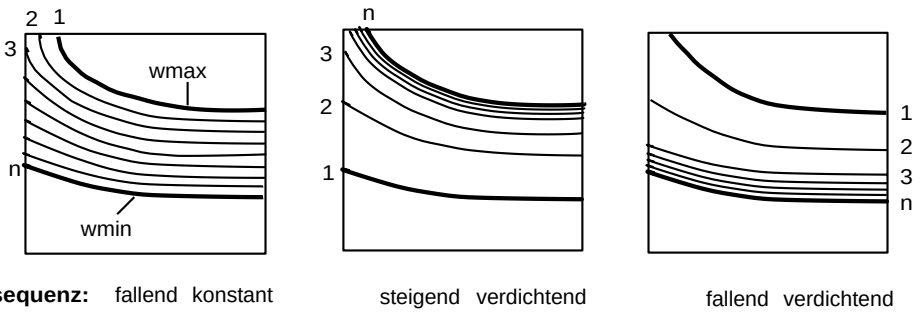


Abb. 4.7: Zu den Verfahren der Isowertegenerierung

Ist ein Inkrement Δ_{inc} vorgegeben, so wird, ausgehend vom Wertebereichsmaximum x_{max}, bei jeder Iteration um das Inkrement vermindert, bis das Wertebereichsminimum x_{min} erreicht ist. Folglich gilt für den i-ten Isowert:

$$x_{iso_i} = x_{max} - (i-1) * \Delta_{inc} \qquad \text{mit} \qquad x_{iso_i} \geq x_{min} \tag{4.4}$$

Ist indes eine Anzahl n zu generierender Werte vorgegeben, werden die einzelnen Isowerte entweder über ein lineares Inkrement nach

$$x_{iso_n} = x_{max} - (n-1) * \Delta_{inc} \tag{4.5}$$

$$\text{mit } \Delta_{inc} = (x_{max} - x_{min})/(n-1) \qquad \text{und} \qquad n > 1 \tag{4.6}$$

bestimmt oder exponentiell aus

$$x_{iso_i} = x_{min} + \Delta x * \sum_{j=1}^{i} 2^{-j} \qquad \text{steigend verdichtend, bzw.} \tag{4.7}$$

$$x_{iso_i} = x_{max} - \Delta x * \sum_{j=1}^{i} 2^{-j} \qquad \text{fallend verdichtend,} \tag{4.8}$$

$$\text{mit} \qquad \Delta x = x_{max} - x_{min} \tag{4.9}$$

Die interne Verwaltung des Isowertefeldes ermöglicht die interaktive Kombination aller Generierungstechniken, einschließlich der Hinzunahme bereits auf Datei vorliegender Isowerte, der Eliminierung einzelner Isowerte bzw. ganzer Isowertegruppen in der Reihenfolge ihrer Generierung und nicht zuletzt der Erstellung einer Isowertedatei.

4.3.2.2 Beschränkung in der Elementauswahl

Für die Berechnung von Isolinien und Konturflächen finden ausschließlich zweidimensionale Drei- und Vierpunktselemente Verwendung und hier die einfachsten Typen, die keine Seiten- und Flächenmittelpunkte aufweisen.
Zur Berechnung werden lineare Interpolationsverfahren herangezogen. Seiten- und Flächenmittelpunkte können jedoch insofern berücksichtigt werden, als eine Aufspaltung dieser Elemente in Unterelemente erfolgt, unter Beibehaltung des Drehsinns des Ursprungselements der sich aus der Nummerierungsfolge ergibt (Abb. 4.8). Jedes Unterelement muß in sich konvex sein, was bedeutet, daß keine Verbindungslinie beliebiger zweier Punkte dieses Elements darf außerhalb der Elementfläche liegen darf.

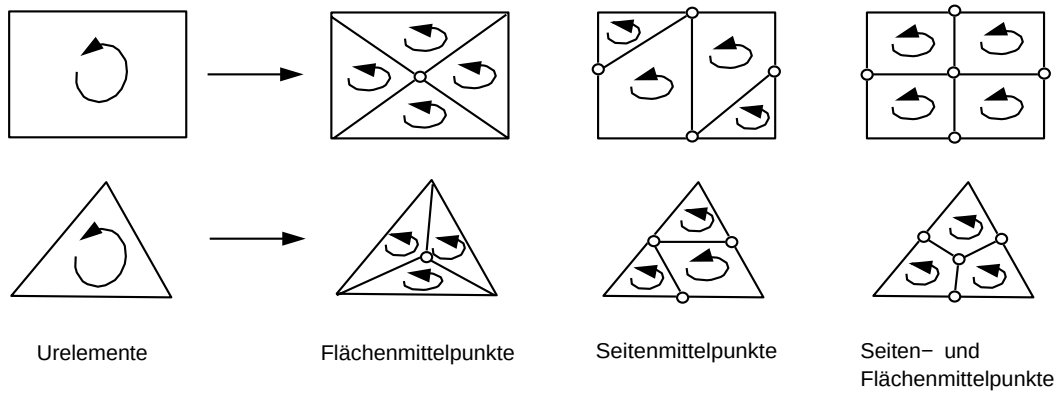


Abb. 4.8: Aufspalten von Elementen mit Seiten- und Flächenmittelpunkten in konvexe Unterelemente

4.3.2.3 Algorithmen zur Berechnung von Isolinien und Konturflächen

In der ersten Generation von Algorithmen wurden elementweise Isoliniensegmente und Konturteilflächen berechnet, wobei jedes Element für sich betrachtet wurde, ohne daß die Kenntnis seiner Nachbarn einfloß, mit denen gemeinsame Kanten bestehen.

Das führt zu schnellen Algorithmen, die ohne zusätzliche Behelfslisten auskommen, dafür jedoch eine Fülle von Isopunkten mehrfach erzeugen, eben weil gemeinsame Kanten nicht berücksichtigt werden.
Eine speicherfreundlichere Lösung, hinsichtlich der Datenausgabe des Gittertransformators, wird mit der zweiten, hier vorgestellten Methode erzielt, bei der jeder Isopunkt nur einmal existiert, dafür aber temporäre Behelfsfelder wie Kantenreferenzlisten und Elementkantenlisten benötigt werden.
Der Vorteil der zweiten Algorithmengruppe zeigt sich nicht nur bei großen Datenmengen, sondern besonders bei gekrümmten Flächen im Raum, die aufgrund der zu einem Netz verknüpften Punkte, und der damit erhaltenen Konnektivität, *Gouraud*- und *Phong*-schattiert werden können [29]. Im Gegensatz dazu können Algorithmen der ersten Gruppe, bei der jede Isoteilfläche bzw. jedes Liniensegment in keinerlei Verbindung zu ihren Nachbarn steht nur flach-schattiert werden. Gekrümmte Flächen erscheinen dort facettenförmig in der Feinheit des ursprünglichen Finite-Element-Gitters.
Algorithmen der ersten Gruppe werden jedoch weiterhin in der parallelen Version des Gittertransformators eingesetzt, da dort eine willkürliche Aufteilung der Elemente auf die Prozessoren erfolgt und folglich keine zusammenhängenden Gebiete mehr existieren, die für die zweite Generation der Algorithmen eine Voraussetzung sind.
Die Berechnung von Isolinien und Konturflächen erfolgt in zwei Schritten. In einem ersten Schritt werden sämtliche Schnittpunkte der Isolinien mit den Kanten der Finiten Elemente ermittelt. Da an beiden Knoten x_i einer Kante die Ergebniswerte w_i bekannt sind, findet man den Isopunkt x_{iso} durch lineare Interpolation. Ein Isopunkt existiert dann auf einer Kante, wenn genau einer der Knotenwerte der Kante w_i größer ist als der Isowert w_{iso} (Abb. 4.9). Der exakte Ort eines Isopunktes auf einer Kante errechnet sich somit aus

$$x_{iso} = x_1 + (x_2 - x_1) * \frac{w_1 - w_{iso}}{w_1 - w_2}. \tag{4.10}$$

Die Berechnung erfolgt kantenweise und isowertweise, angefangen mit dem größten Isowert monoton fallend bis zum kleinsten Wert. Eine Isoreferenzliste $kali(2, j)$ wird angelegt, in der für jede Kante j die Identität $id0$ des ersten Isopunktes und die Anzahl der gefundenen Punkte *kpp* auf der Kante gespeichert wird.
Somit ist in einem zweiten Durchlauf, in dem elementweise die Verbindung der Punkte zu Polygonen festgelegt wird, jeder Isopunkt identifizierbar und unkompliziert zu referenzieren, immer auch unter Berücksichtigung des Drehsinns, der sich aus der Reihenfolge der Knotennummerierung ergibt und mit dem des Urelements übereinstimmen muß (Abb. 4.12).

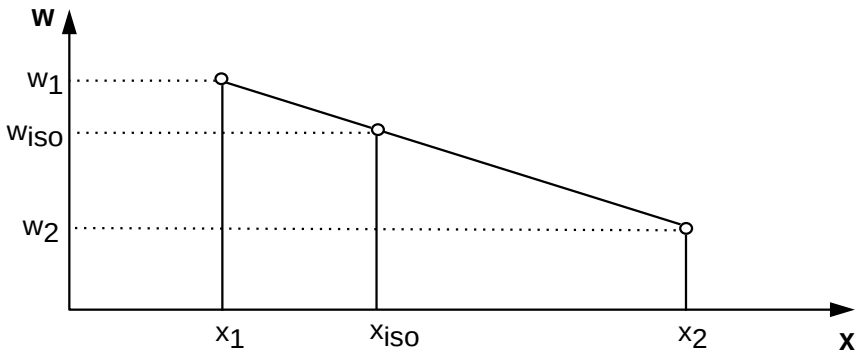


Abb. 4.9: Interpolation eines Isopunktes auf einer Elementkante

Isolinien

Für den Fall der Isolinien ist eine Referenzierung der im ersten Schritt gefundenen Isopunkte noch relativ einfach, da nur Linienversatzstücke entstehen bzw. Zweipunktspolygone, bei denen der Umlaufsinn keine Rolle spielt.
Abbildungen 4.10 und 4.11 zeigen vereinfacht in einem Struktogramm das Prinzip des Algorithmus' auf. Beide Schritte sind getrennt voneinander dargestellt.

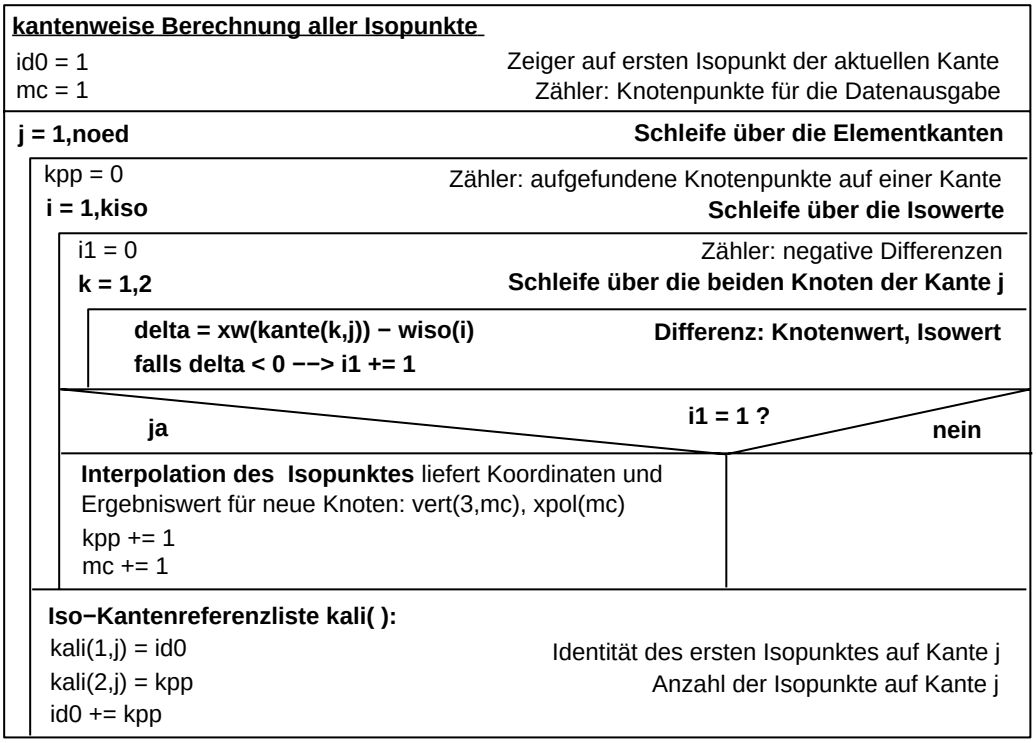


Abb. 4.10: Isopunkteberechnung im Isolinienalgorithmus

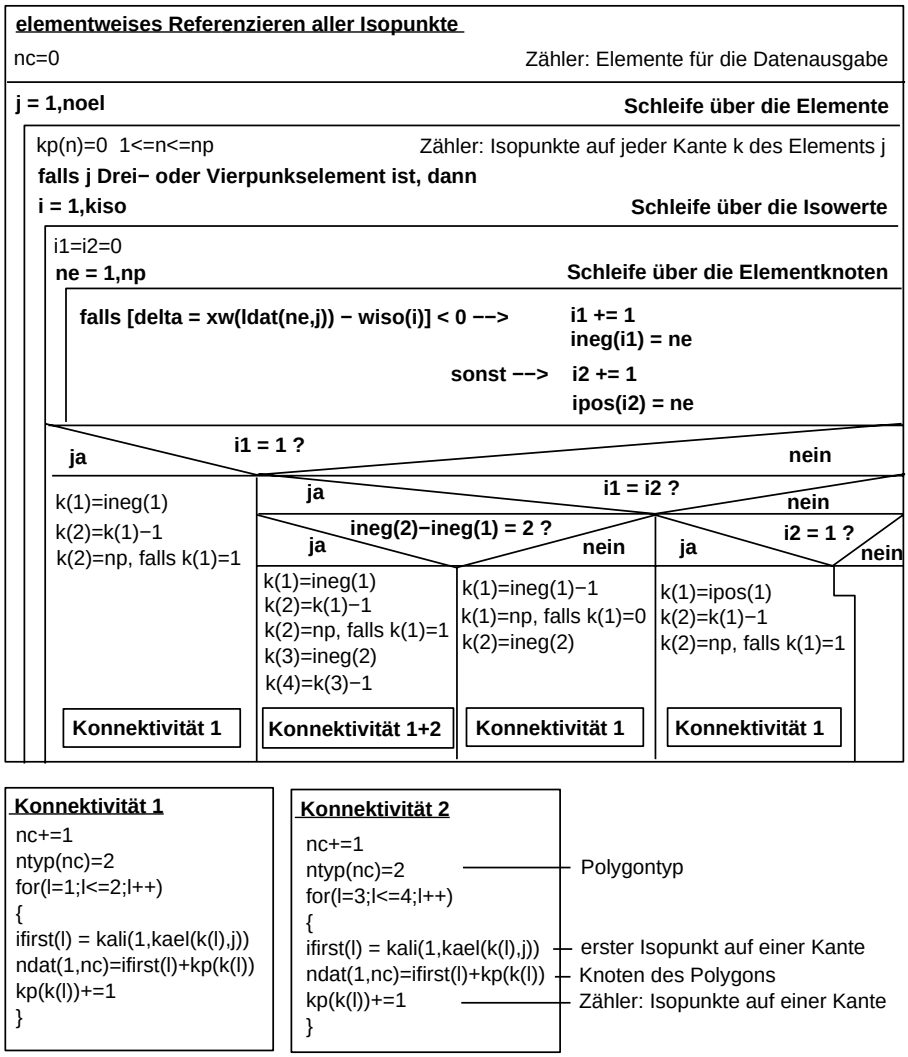


Abb. 4.11: Referenzierung im Isolinienalgorithmus

Konturflächen

Eine Besonderheit im Konturalgorithmus ist der Schritt der Isopunkteberechnung und liegt in der Duplizierung jedes gefundenen Isopunktes (Abb. 4.12). Jede Konturteilfläche in einem Finiten Element wird durch ein Polygon umrandet, das sowohl nur aus Isopunkten bestehen kann, als auch gemischt aus Isopunkten und Elementknotenpunkten.
Jedem Polygonpunkt wird dieselbe Farbe beigeordnet, damit die umspannte Fläche entsprechend dem Isowert einfarbig erscheint. Dabei darf jeder Polygonkoordinate, aufgrund der mit der Graphikbibliothek vorgegebenen Datenstruktur, eindeutig nur ein einziger Farbwert entsprechen. Damit jedoch Isopunkte nicht zu Polygonen verschiedener Farbwerte gehören, wird bereits bei der Generierung der Isopunkte jeder

gefundene Punkt i auf der Kante k dupliziert, und zwar so, daß der verdoppelte Punkt die Identität $i + 1$ erhält.
Im zweiten Schritt, dem Referenzierungsschritt, wird elementweise verfahren und wiederum isowertweise vorgegangen. Für jeden Isowert wird das zugehörige Polygon gebildet, welches sich aus den Isopunkten beteiligter Kanten und Elementknotenpunkten zusammensetzt. Sich ergebende Teilflächen werden über einen Typenparameter in Erstflächen, Zwischenflächen sowie Restflächen unterschieden (Abb. 4.12).

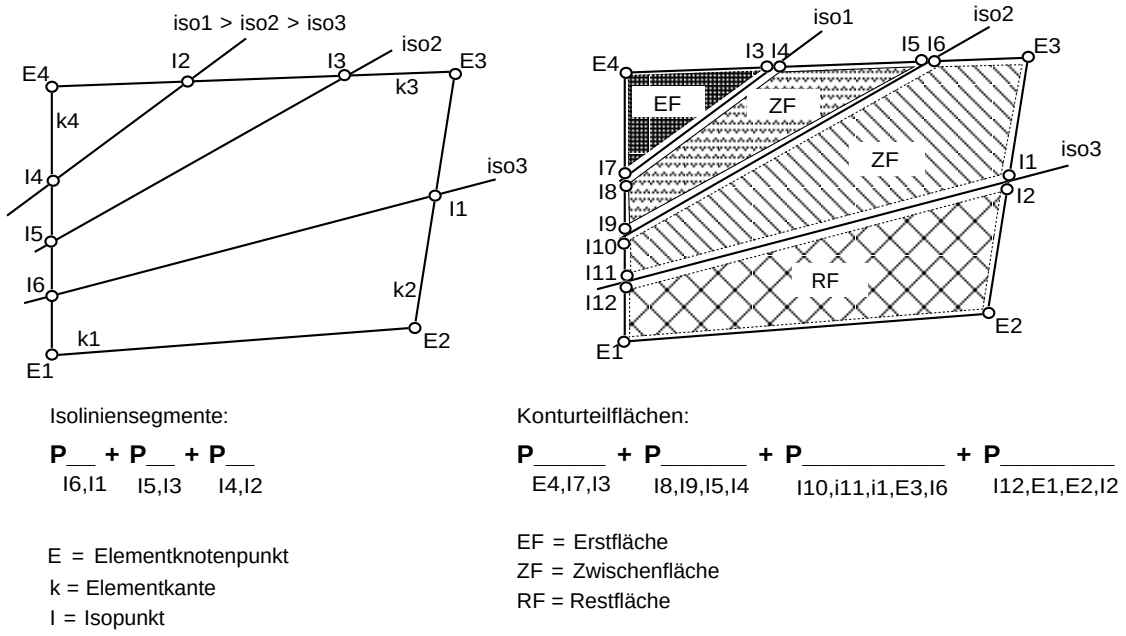
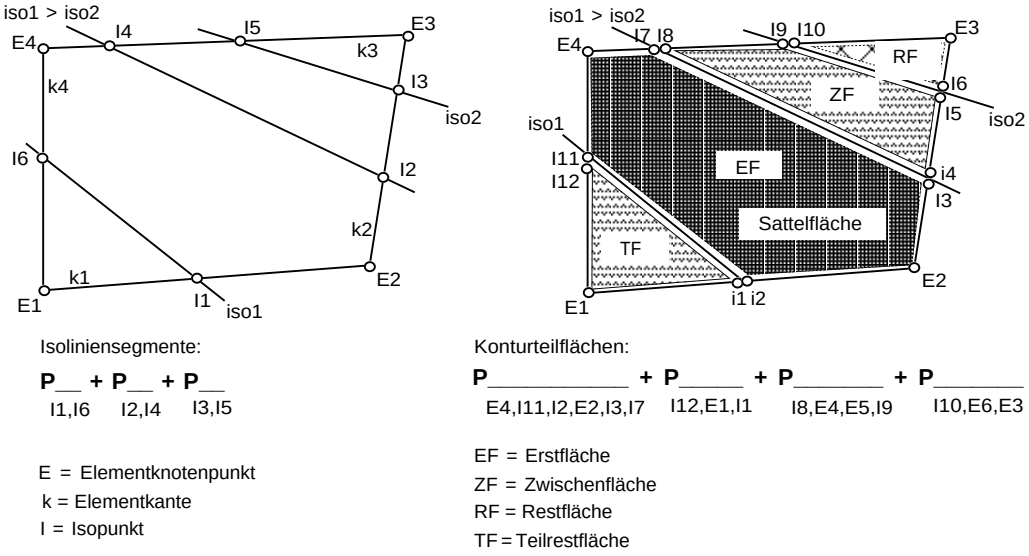


Abb. 4.12: Isoliniensegmente und Konturteilflächen

Die Struktur des Algorithmus zur Ermittlung der Konturflächenpunkte ist in Abbildung 4.14 angegeben. Teilrestflächen ergeben sich für den Spezialfall einer Sattelfläche (Abb. 4.13).
Die Erstfläche ist die erste, im betrachteten Element gefundene Isoteilfläche. Die Restfläche ist die letzte, die übrig bleibt. Die Zwischenflächen liegen, wie der Name sagt, dazwischen. Natürlich weisen Elemente, durch die nur eine einzige Isolinie verläuft, keine Zwischenflächen auf, ebenso wie es für Elemente, die von Isolinien unberührt bleiben, keine Flächenuntergliederung gibt.
Sattelflächen entstehen in Vierpunktselementen, wenn in der Abfolge der Knoten des Elements die Werte an den Knoten alternierend mal größer und mal kleiner als der Isowert sind, d.h. entweder der erste und der dritte Knotenwert des Elements größer als der Isowert ist, oder der zweite und der vierte (Abb. 4.13).
Bei der Erzeugung der Teilflächen muß die Kenntnis über Typ und Gestalt der unmittelbar zuvor in diesem Element gefundenen Teilfläche vorliegen, damit die rich-

tigen Punkte referenziert und zu Polygonen verbunden werden können. Während der Typenparameter angibt, ob es sich um eine Erst- oder Zwischenfläche handelte, sagt der Gestaltparameter über die Fläche aus, ob ein Drei-, Vier-, Fünf- oder gar ein Sechseck vorliegt.



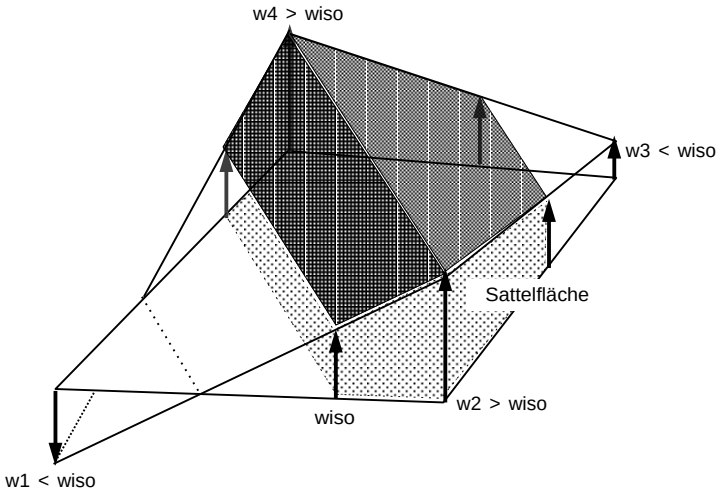


Abb. 4.13: Behandlung von Sattelflächen

Es werden bis zu sechs Hauptfälle unterschieden, abhängig davon, wieviele Differenzen kleiner Null, bzw. größergleich Null sind (Abb. 4.15). Die Mehrheit dieser Fälle bildet wiederum Unterfälle, je nachdem, ob es sich bei den zu berechnenden Konturteilflächen um eine Erst-, Zwischen-, Rest- oder gar um eine Teilrestfläche handelt.

Bei Zwischen-, Rest- und Teilrestflächen werden weitere Fälle unterschieden, je nach

der Gestalt der zuvor im selben Element aufgefundenen Teilfläche, die über den Parameter *ivor* definiert wird.

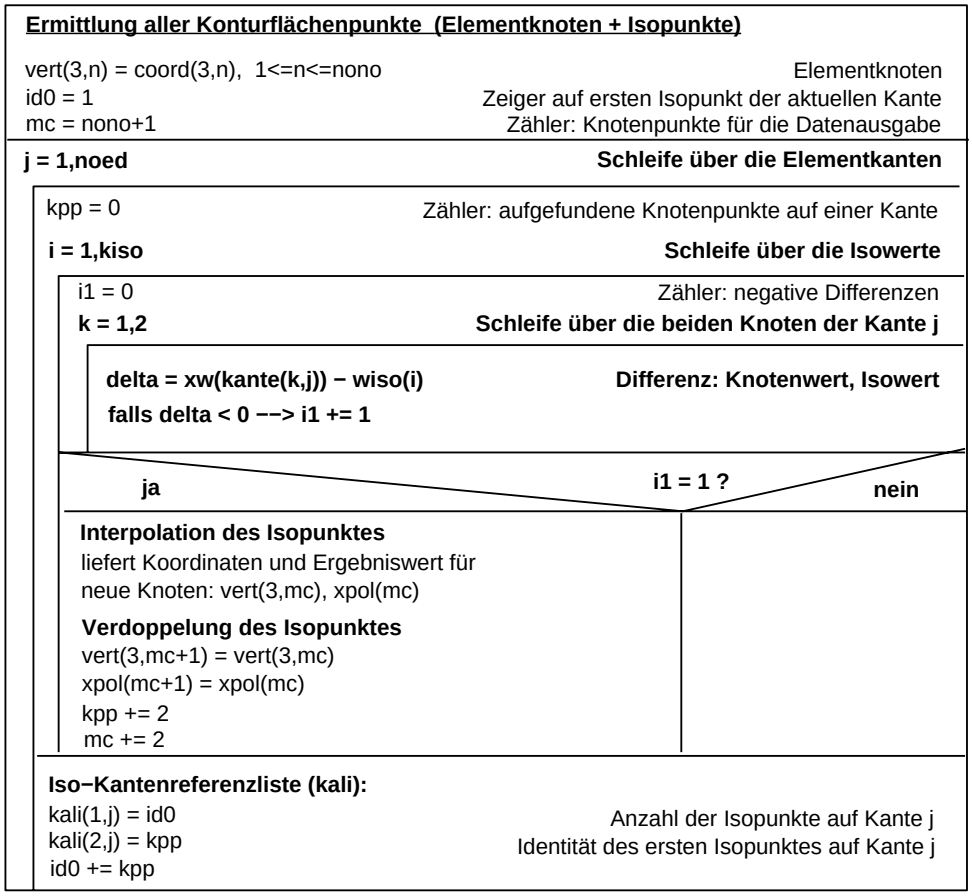


Abb. 4.14: Ermittlung aller Konturflächenpunkte

4.3.2.4 Alternatives Verfahren für Konturflächen

Ein einfaches, häufig in kommerziellen Produkten [19] zu findendes Verfahren, zur Darstellung von Konturflächen, wird zur Reduzierung der Farben eingesetzt.
Um eine Anzahl von n Konturflächen zu erhalten, wird eine Farbtafel *(colour-map)* generiert, die das gesamte Farbenspektrum auf n diskrete Bereiche abbildet bzw. darauf reduziert. Da nun beispielsweise bei einer quadrilinearen Interpolation von Knotenwerten über eine Viereckfläche nur noch wenige diskrete Farbwerte zur Verfügung stehen, erscheint die Fläche diskontinuierlich d.h. farblich abgestuft.
Dieses Verfahren, kommt eigentlich einer kontinuierlicher Darstellung mit reduzierten Farbwerten gleich. Die Hauptschwierigkeit ist hier, eine exakte Farbtafel zu generieren.

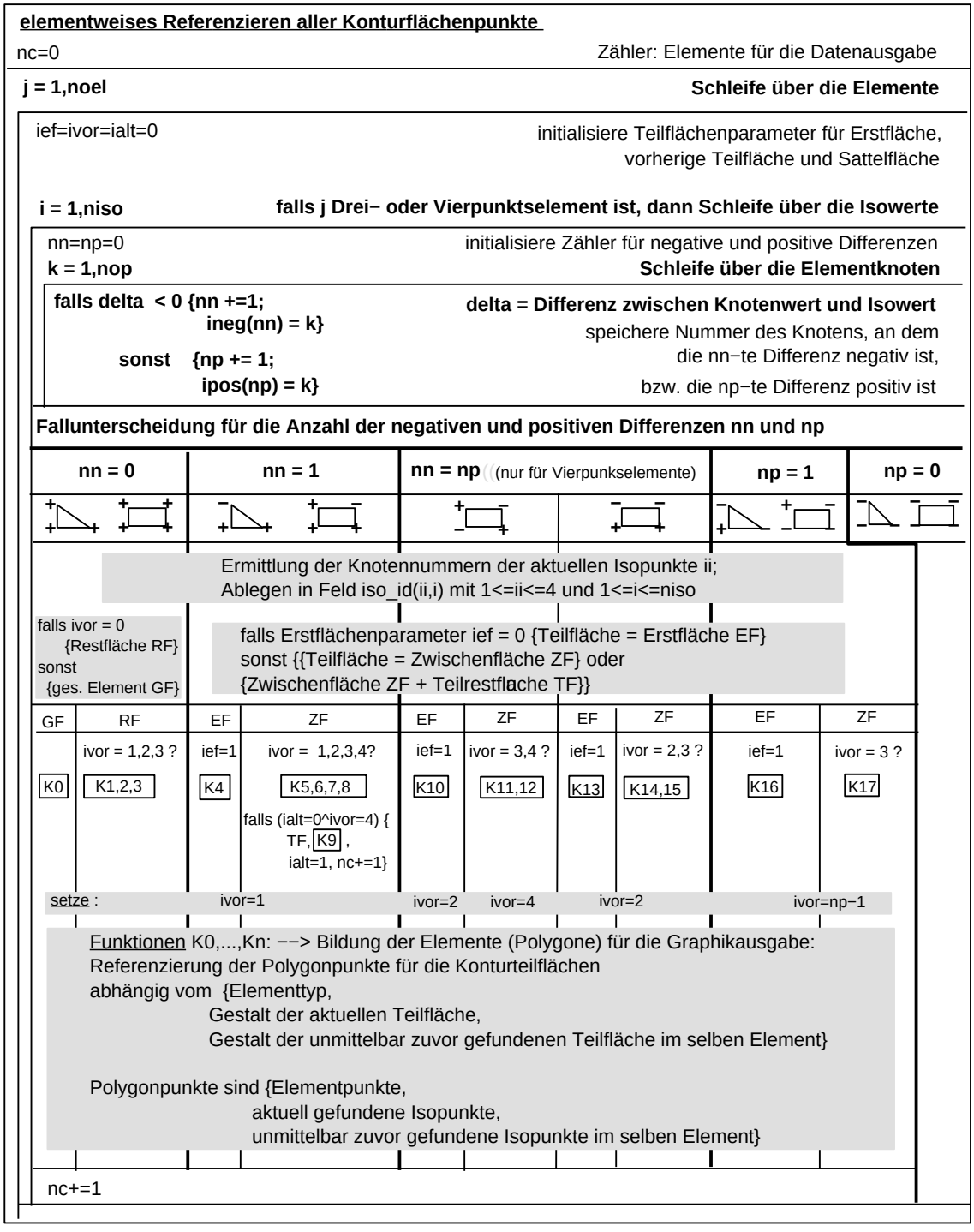


Abb. 4.15: Konturteilflächenbildung durch Referenzierung der Konturflächenpunkte

4.3.3 Die Carpet-Darstellung

Der Begriff Carpet-Darstellung hat sich für eine Quasi-3D-Darstellung eingebürgert, in der Ergebniswerte ebener Probleme zur Veranschaulichung zusätzlich als Höhenwerte abgetragen werden. Für Finite Element- und Finite Volumen-Anwendungen heißt das, daß die Elementknoten aus einem 2D-Problem entlang der dritten Raumachse verschoben werden, wobei die Verschiebung $\vec{v}$ sich nach dem Ergebniswert e jedes Knotens und einem Streckfaktor fs richtet. Somit ergibt sich der neue Ort eines Punktes X_p zu:

$$\vec{x}_{p_1} = \vec{x}_{p_0} + \vec{v} \tag{4.11}$$

mit einem Verschiebungsvektor $\vec{v} = (0, 0, fs * e)$ für ein Problem, das in der Ebene E_{xy} angesiedelt ist.
In Abbildung 4.16 und 4.17 sind zwei Anwendungen der Carpetmethode illustriert. Beiden Beispielen liegt dasselbe Strömungsproblem zugrunde, ein angeströmter Kreiszylinder. Die sich ergebende Karmannsche Wirbelstraße im Nachlauf wurde in Abbildung 4.16 als Konturflächen der Geschwindigkeiten dargestellt, wohingegen in Abbildung 4.17 ein mit Isolinien überlagerter einfarbiger Fringeplot zu sehen ist. Alle Werte wurden mit einem Streckfaktor $fs = 0.5$ in die Höhe projiziert.

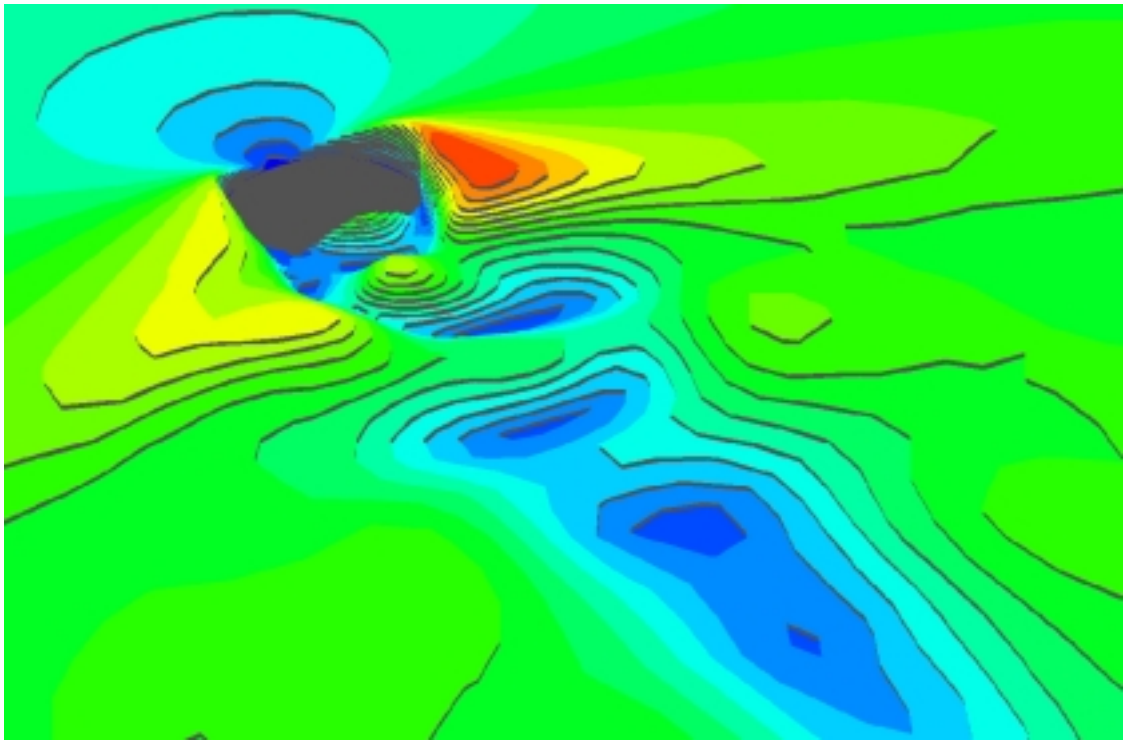


Abb. 4.16: Konturflächen, dargestellt auf einem Geschwindigkeitsgebirge

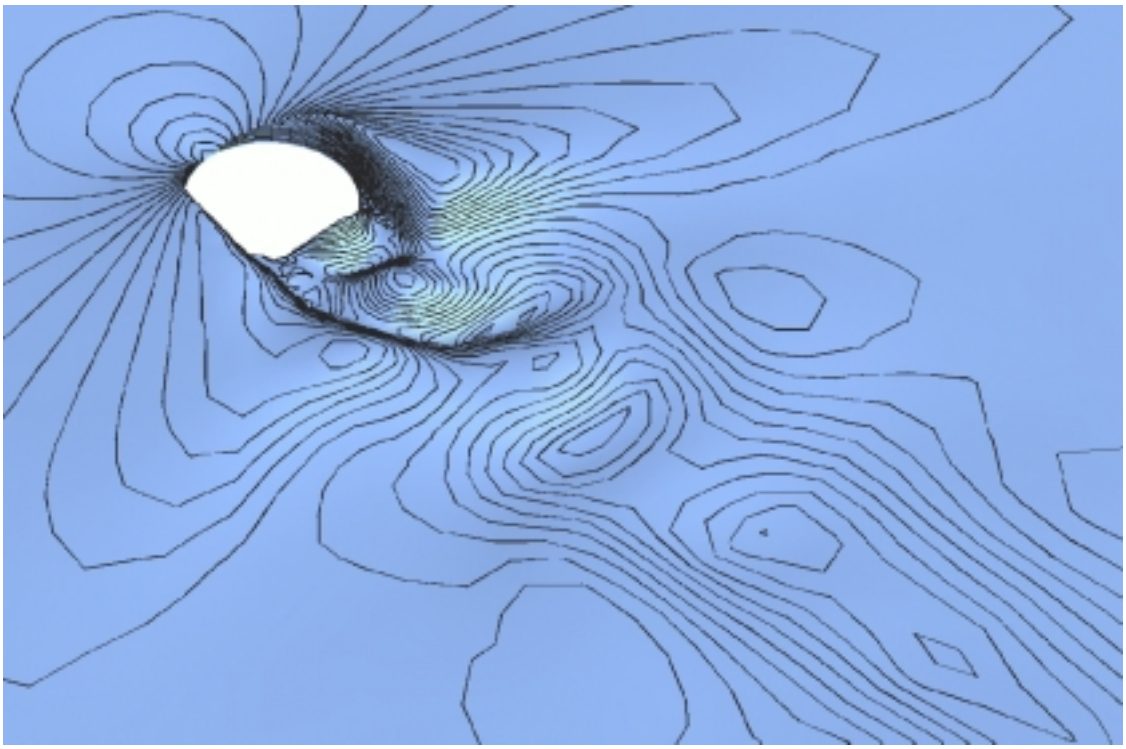


Abb. 4.17: Isolinien, dargestellt auf einem Geschwindigkeitsgebirge

4.3.4 Isoflächen

Wie die Isolinien- und Konturflächenberechnung ist auch die Isoflächenberechnung auf zwei Wegen möglich. Die einfachere Verfahrensweise, die ohne Elementkanten- und Kantenknotenlisten auskommt, wird vorwiegend auf Parallelrechnern eingesetzt, während die elegantere Methode wiederum nur in zusammenhängenden Gebieten anwendbar ist. Da sich beide Algorithmen in ihrer Struktur sehr ähnlich sind, soll an dieser Stelle die Funktionsweise des zweiten Algorithmus erläutert werden. Der Algorithmus ähnelt gerade in seiner ersten Serie von Fallunterscheidungen dem Marching Cube Algorithmus [62], schließt jedoch in seinen weiteren Fallunterscheidungen Mehrdeutigkeiten bei der Flächenbildung aus und liefert überdies in der Knotenreferenzierung den korrekten Drehsinn der Flächen.

Vorgehensweise
Analog der Isolinienberechnung werden zunächst unter Verwendung der entsprechenden Behelfslisten für jede Elementkante die Schnittpunkte mit den Isoflächen ermittelt. Soll eine physikalische Größe auf die Isofläche abgebildet werden (*data mapping*), kann optional zu jedem Isopunkt der Ergebnisvektor mitgeliefert werden.

Den Hauptteil des Algorithmus macht die komplizierte Knotenreferenzierung aus, sowie die Bildung der Teilflächen innerhalb jedes räumlichen Elements. Eine Serie von Fallunterscheidungen wurde getroffen und Regeln vereinbart, nach denen elementweise zugehörige Isopunkte miteinander zu Polygonzügen, die die Isoteilfläche umranden, verbunden werden. Dies geschieht unter Berücksichtigung des Umlaufsinns der Punkte, der die Richtung der Flächennormalen festlegt.

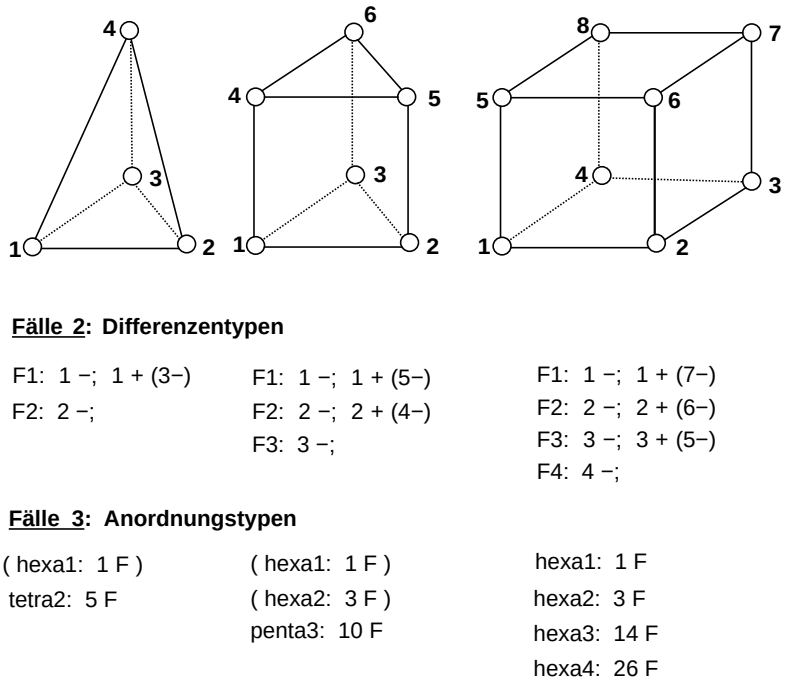


Abb. 4.18: Die dreifache Schachtelung der Fallunterscheidungen für eine Isoteilflächenbestimmung

Gültige Elementtypen für die Berechnung sind Tetraeder, Pentaeder und Hexaeder. Eine Zusatzoption gestattet die Aufspaltung der Hexaeder und Pentaeder in Tetraeder, für die die Teilflächenbildung wesentlich einfacher ist, da es zum einen weniger Fallunterscheidungen gibt und zum anderen als Teilflächen lediglich Drei- und Vierecke auftreten, im Gegensatz zum Hexaeder, bei dem Sechsecke keine Seltenheit sind.

Fallunterscheidungen

Die Fallunterscheidungen sind dreifach ineinander geschachtelt, nach Element-, Differenzen- und Anordnungstypus. Für jeden der Elementtypen (Tetraeder, Pentaeder, Hexaeder) wird in der zweiten Schachtelungstiefe eine Betrachtung der Isowertedifferenzen an den Elementknoten angestellt. Sowohl die Positionen der negativen und positiven Knoten werden ermittelt als auch die Summen positiver und negativer

Vorzeichen. Die Fallunterscheidung selbst richtet sich nach der Anzahl der Knoten mit negativen Differenzenvorzeichen.

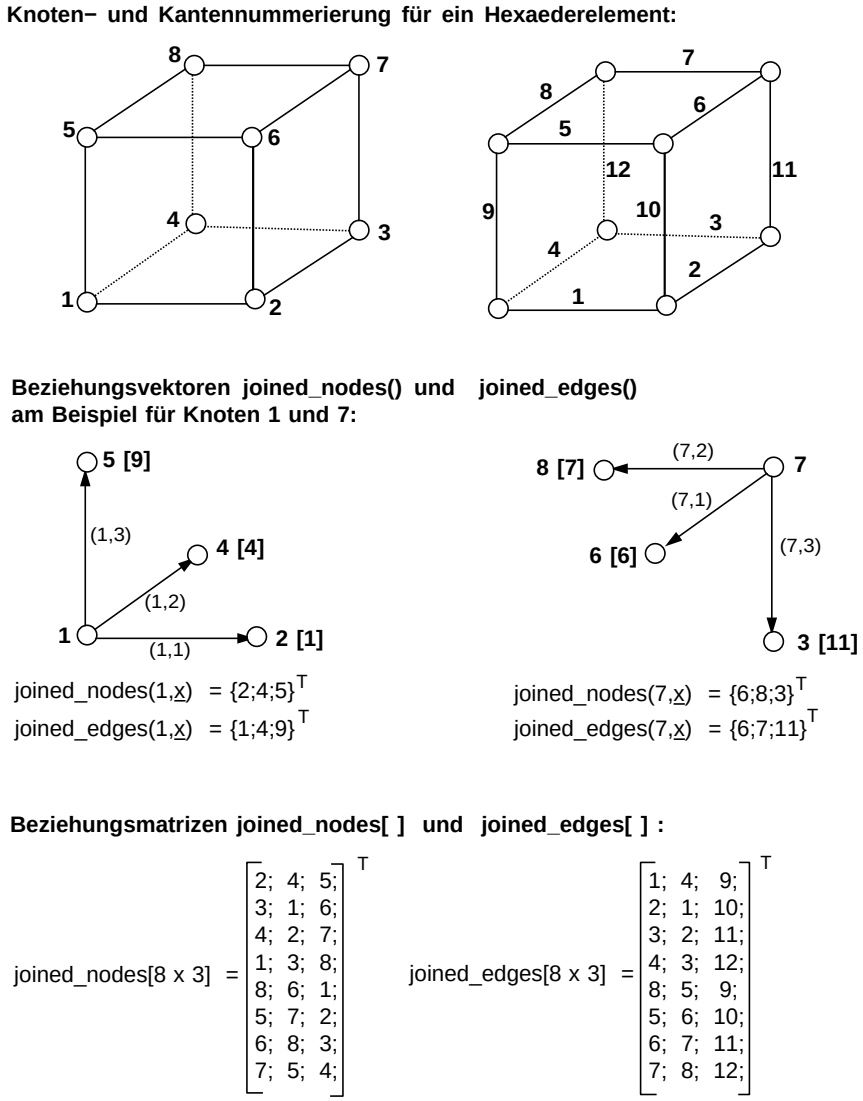


Abb. 4.19: Knoten- und Kantennachbarschaftsbeziehungen am Beispiel Hexaeder

In einer dritten Schachtelung wird die Lage der markierten Knoten zueinander untersucht. Unter Zuhilfenahme zweier Elementbeziehungsmatrizen, - von denen die eine ($joined\_edges[8x3]$) für jeden Elementknoten die angrenzenden Kanten benennt und die andere ($joined\_nodes[8x3]$) die Namen der Knoten an den Enden der angrenzenden Kanten - , wird eine Vielzahl Unterfälle unterschieden. In diesem Schritt findet die eigentliche Referenzierung derjenigen Isopunkte zu Teilflächenpolygonen statt, die auf den zu den markierten Knoten gehörigen freien Kanten sitzen. Als *frei* wird in diesem Zusammenhang eine Kante dann bezeichnet, wenn von den beiden Knoten, die die Kante definieren, maximal einer markiert ist.

In Abbildung 4.19 sind die Beziehungsmatrizen dargestellt während Abbildung 4.18 die drei Schachtelungen der Fallunterscheidungen zeigt. Zur Verdeutlichung sind in den Abbildungen 4.20 und 4.21 zwei Unterfälle explizit dargestellt, einmal die fünf verschiedenen Lagen der Isoteilflächen für einen Tetraeder mit zwei negativen Differenzen (tetra2) und zum anderen alle 26 Unterfälle für ein Hexaederelement mit vier negativen Differenzen (hexa4).

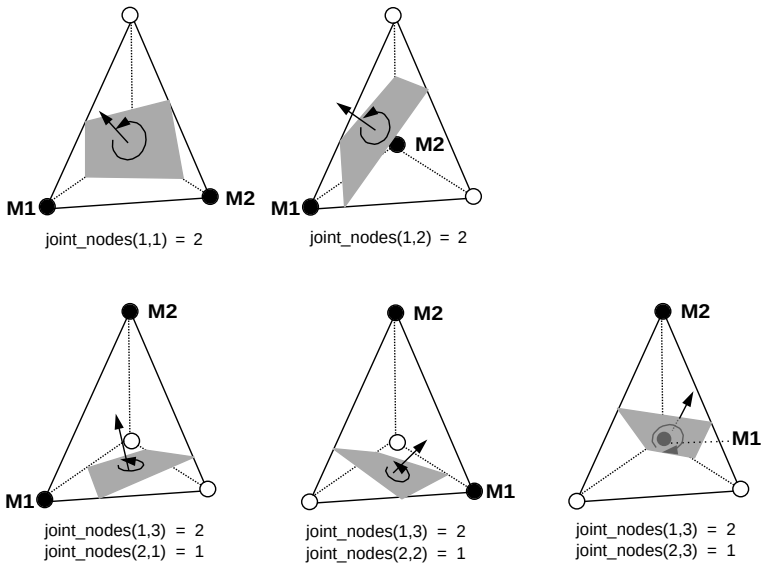


Abb. 4.20: Die fünf Anordnungsfälle für einen Tetraeder mit zwei negativen Isodifferenzen (tetra2), dargestellt mit Flächennormalen und Drehsinn

Im übrigen können Fälle, in denen positive Differenzen überwiegen, wie ihre negativen Pendants behandelt werden, in dem an die entsprechenden Routinen die positiv markierten Knoten übergeben werden sowie ein gesetztes Spinbit, das anzeigt, daß der Drehsinn der referenzierten Isoteilfläche für diese Fälle umzukehren ist.

Referenzierung

Wie gesehen wurde, ergeben sich die an der Isoteilflächenbildung beteiligten Elementkanten aus der Falluntersuchung. In einem letzten Schritt müssen nun die korrekten Isopunkte auf diesen beteiligten Kanten in der richtigen Reihenfolge in diejenigen Datenfelder übertragen werden, in denen die Vorschriften für die spätere Polygonbildung (Konnektivität) abgelegt werden.
In Abbildung 4.22 wird dieser Schritt erklärt. Mit dem gefundenen Anordnungsfall liegt die Reihenfolge der Kanten für die Referenzierung fest. Variable *id_loc*() speichert die lokale Identität der i-ten beteiligten Kante. Die lokale Identität ist die Nummer, die eine Kante *per definitionem* erhält, beim Hexaeder z.B. eine Nummer

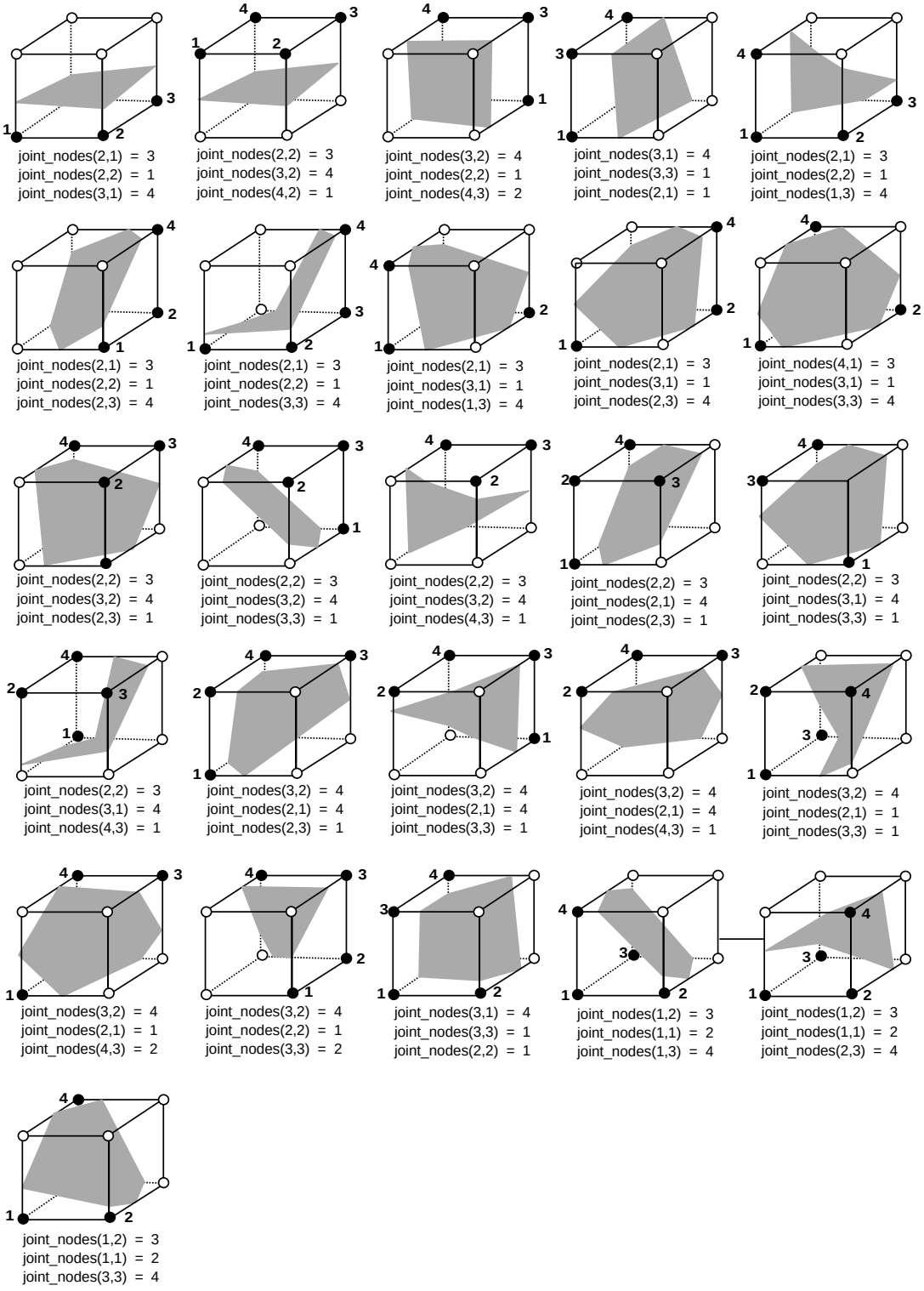
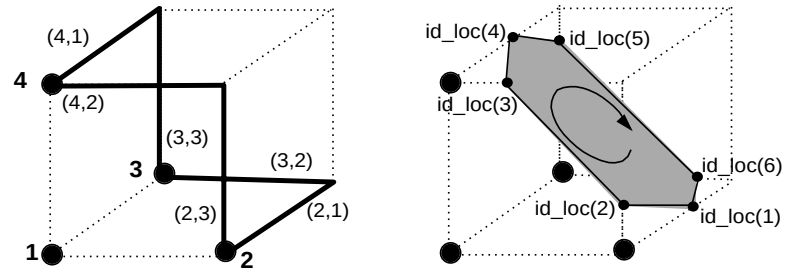


Abb. 4.21: Die 26 Anordnungsfälle für einen Hexaeder mit vier negativen Isodifferenzen

zwischen 1 und 12 (Abb. 4.19). Die lokale Identität eingesetzt in die Elementkantenliste $kael()$ liefert die globale Identität $id\_glob()$ der Kante des betrachteten Elements, eine Nummer zwischen 1 und $noed$.

An der Isoteilflächenbildung beteiligte Elementkanten für Fall 24 aus hexa4



Festlegung der Reihenfolge der Kanten für die Referenzierung

```
id_loc(1) = joined_edges(2,1)
id_loc(2) = joined_edges(2,3)
id_loc(3) = joined_edges(4,2)
id_loc(4) = joined_edges(4,1)
id_loc(5) = joined_edges(3,3)
id_loc(6) = joined_edges(3,2)
n_iso = 6
```

Identitäten der Isopunkte *id_iso(i)* auf den Kanten *id_loc(i)*

```
for (i =1; i < n_iso; i++)
{
    id_glob(i) = kael(id_loc(i),id_element)
    id_first(i) = kali(1,id_glob(i))
    id_iso(i) = id_first(i) +kp(id_loc(i))
    kp(id_loc(i) += 1
 }
```

Funktion *add_polygon*: speichern der Polygondaten

```
iadd += 1
for (i =1; i < n_iso; i++) ndat(i,iadd) = id_iso(i)
ntyp(iadd) = n_iso
nobj(iadd) = id_obj
```

Abb. 4.22: Referenzierung der Isopunkte am Beispiel hexa4, Fall 24

$Id\_first()$ wiederum markiert die Identität des ersten gefundenen Isopunkts auf dieser Kante und wird durch Einsetzen der globalen Identität in die Kantenliste $kali()$ erhalten, nach $id\_first(i) = kali(1, id\_glob(i))$. Der Zähler $kp()$, der für jedes Element neu mit 0 initialisiert wird, gibt an, wieviele Punkte bisher auf der i-ten beteiligten Kante referenziert wurden, so daß sich schließlich die Identität des aktuell gesuchten Isopunkts ergibt zu:

$id\_iso(i) = id\_first(i) + kp(id\_loc(i))$.

Das Verfahren macht sich zu Nutze, daß im zweiten Referenzierungsdurchlauf natürlich die Isowerte nacheinander in derselben Abfolge gefunden werden wie zuvor beim Berechnungsdurchlauf. Das gilt erst recht für die auf jeder Elementkante gefundenen Isopunkte.

Ergebnisbeispiele

In Abbildung 4.23 ist das Ergebnis einer Isoflächenberechnung für den Fall einer Kaltströmung durch eine axialsymmetrische Düse zu sehen. Das berechnete Problem idealisiert die isotherme Luftzufuhr für einen Brenner und besteht aus 6800 Knoten und 7200 Hexaederelementen.
Gezeigt wird die Isofläche für einen Geschwindigkeitsbetrag von 0.031 m/s. Auf der Isofläche wurde eine Dichtewerteverteilung dargestellt, wobei dunkle Farben Bereiche größter Dichte markieren.

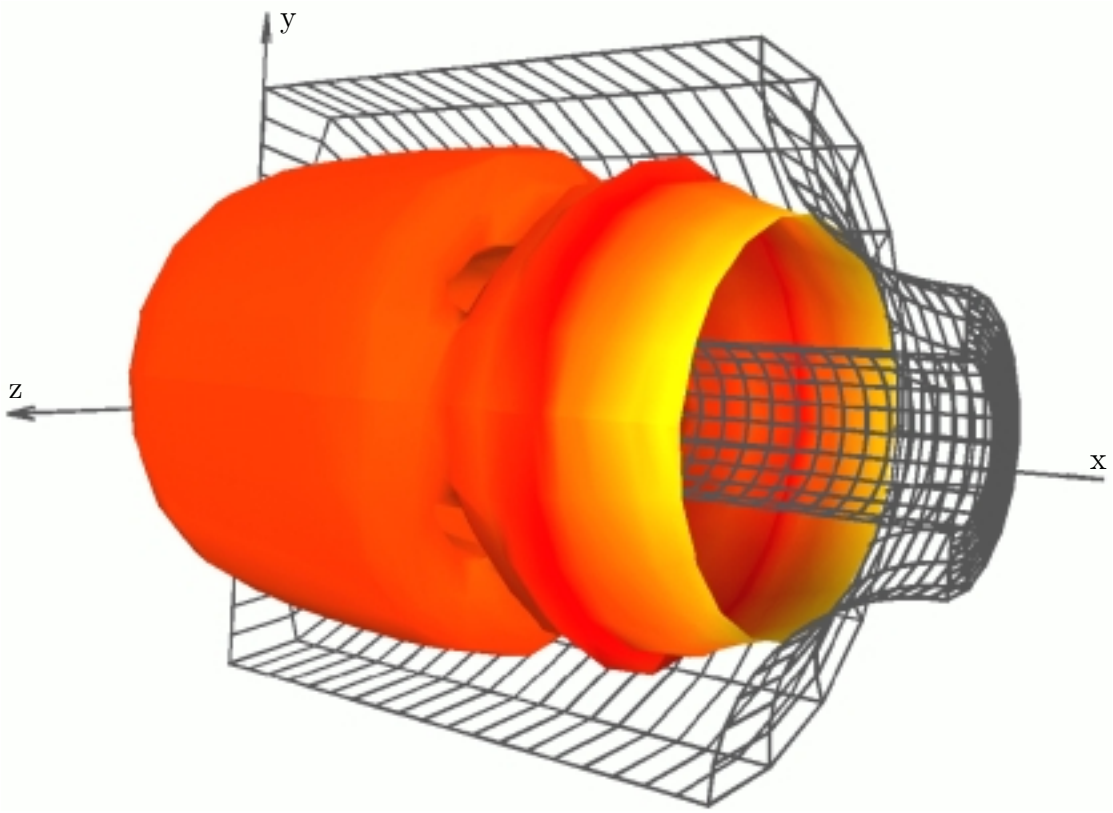


Abb. 4.23: Isofläche einer Geschwindigkeit mit aufprojizierter Dichte in einer kaltdurchströmten Düse

Das zweite, in Abbildung 4.24 dargestellte Beispiel zeigt acht Flächen gleicher Dichte in der chemisch dissoziierten hypersonischen Strömung um den Raumgleiter Hermes beim Wiedereintritt in die Erdatmosphäre. Der Ausschnitt rechts im Bild löst das Gebiet der Schockwelle auf. Das Rechengebiet besteht aus 113000 Knoten und 122000 Elementen, Hexadern und 2D-Oberflächenelementen. Letztere bilden den Raumgleiter selbst und die Ausströmebene ab.

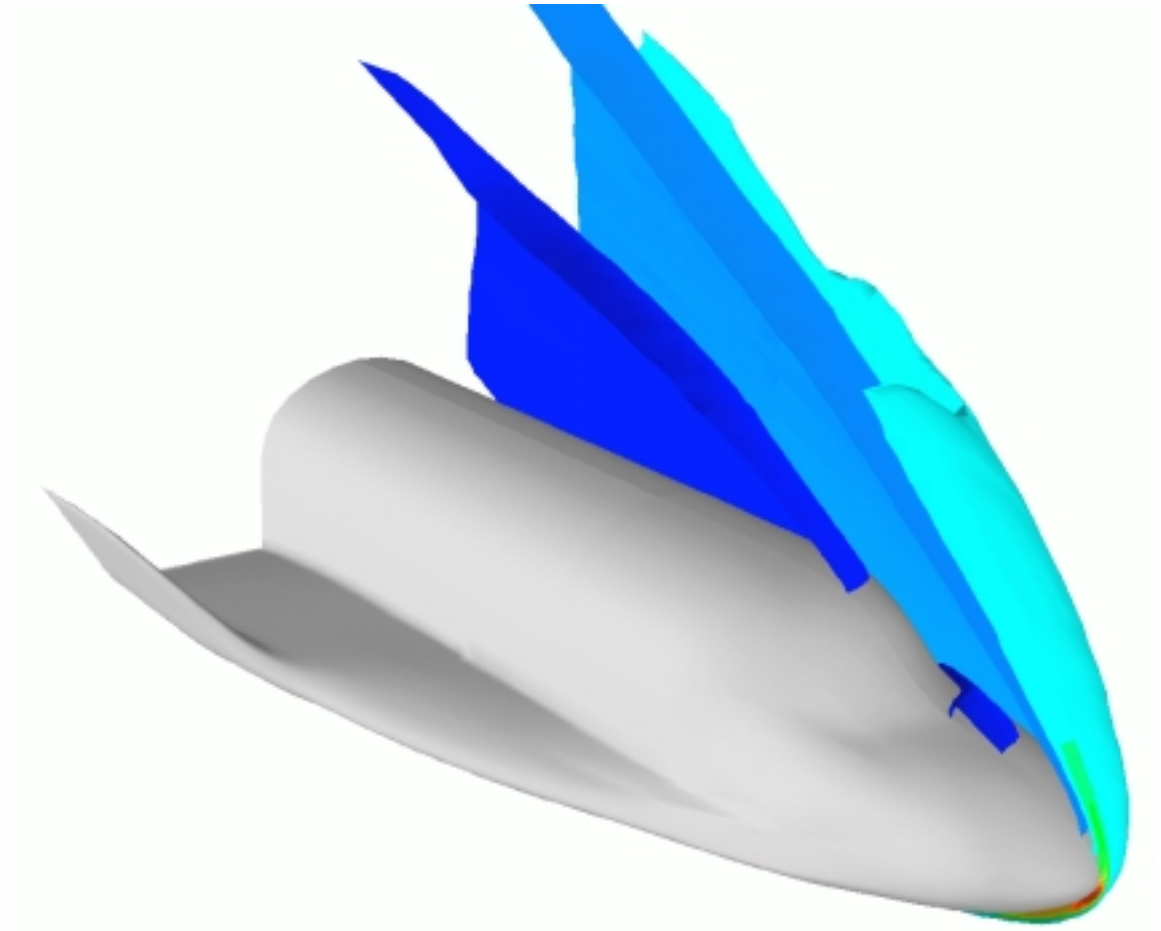


Abb. 4.24: Acht Isoflächen der Dichte um den Raumgleiter Hermes in chemisch reagierender Strömung mit Mach 25.

4.3.5 Schnitte durch dreidimensionale Strukturen

Dieser Abschnitt ist mit der Darstellung von Schnitten durch dreidimensionale Strukturen befaßt. Im einzelnen werden Berechnungsverfahren für Schnittflächen, geschnittene Körper (Restgeometrien), sowie die Darstellung von Ergebniswerten auf beiderlei Strukturen behandelt.
An erster Stelle steht immer die Generierung einer oder mehrerer Schnittebenen mit dem Ebenengenerator, einem eigenständigen Modul, dessen Funktionsweise ein eigener Abschnitt gewidmet ist. Die eigentliche Berechnung der Schnittflächen findet auf Basis der generierten Ebenendaten im Schnittemodul statt. Unter anderem legt der Benutzer hier fest, ob er ausschließlich Schnittflächen zu sehen wünscht oder ob dazugehörige Restgeometrien, d.h. die verbleibenden geschnittenen Objekte, mit dargestellt werden sollen.
Mehrere Schnitte bzw. Restgeometrien lassen sich durch das Modul zu einer Folge von Objekten verarbeiten. Hierbei wird bereits während der Berechnung der Gesamtdaten jedes einzelne Polygon einem ganz bestimmten Objekt zugeordnet. Nur so ist gewährleistet, daß in dem assoziierten Geometrie–Mapper–Modul *geometry builder* für die Darstellung sowohl die Gesamtheit der Objekte ausgewählt werden kann, als auch einzelne Objekte, - seien es Schnitte oder Restgeometrien. Anmerkend sei erwähnt, daß sich in dem Schnittemodul festlegen läßt, ob eine Restgeometrie später einfarbig oder mit Ergebnisdaten dargestellt wird.
Darüber hinaus bestehen Schnittstellen, an die andere Moduln ankoppeln können, z.B., wenn Konturflächen oder Isolinien auf Schnittflächen dargestellt werden sollen.

4.3.5.1 Interaktive Schnittflächengenerierung

Eine Schnittebene kann auf vier verschiedene Weisen eindeutig definiert werden: Einmal durch Wahl dreier verschiedener Ebenenpunkte, die nicht auf ein und derselben Geraden liegen dürfen, zweitens durch Vorgabe zweier Ebenenpunkte und einer Richtung, drittens durch einen Ebenenpunkt und zwei linear unabhängige Richtungen und schließlich durch die Vorgabe eines Sprungvektors bzw. Aufpunkts auf die Ebene sowie eines zum Normalenvektor der Ebene kollinearen Vektors. Die letzte Möglichkeit ist zu bevorzugen, da nicht nur weniger interaktive Eingaben getätigt werden müssen, sondern sie auch dem Vorstellungsvermögen des Benutzers entgegenkommt.

Verfahrensweise
Durch Vorgabe der drei Komponenten des Normalenvektors sowie der Koordinaten des Aufpunktes legt der Benutzer die Schnittebene fest, die dann idealisiert als quadratische Fläche mit dem Aufpunkt als Zentrum gezeichnet wird, mit einer Kantenlänge und in einer Farbe, die ebenfalls vom Benutzer angegeben wird.
Eine solche Ebene kann als Grundebene für eine Serie von Schnitten dienen. Als Schnittserie können z.B. Parallelebenen erzeugt werden oder aber Ebenen, die alle eine gemeinsame Schnittachse haben (Abb. 4.25). Eine Serie von parallelen Schnitten wird durch Wahl eines Inkrements d_{inc} und einer Anzahl zu generierender Ebenen festgelegt, eine Serie gedrehter Schnittebenen hingegen durch Vorgabe eines Drehwinkels ω, der Anzahl der innerhalb dieses Winkels zu generierender Ebenen, sowie der Lage der Drehachse. Als Drehachsen sind ausschließlich Parallelen zu einer der drei kartesischen Raumrichtungen erlaubt. Die Drehachse geht durch den Ebenenaufpunkt A.

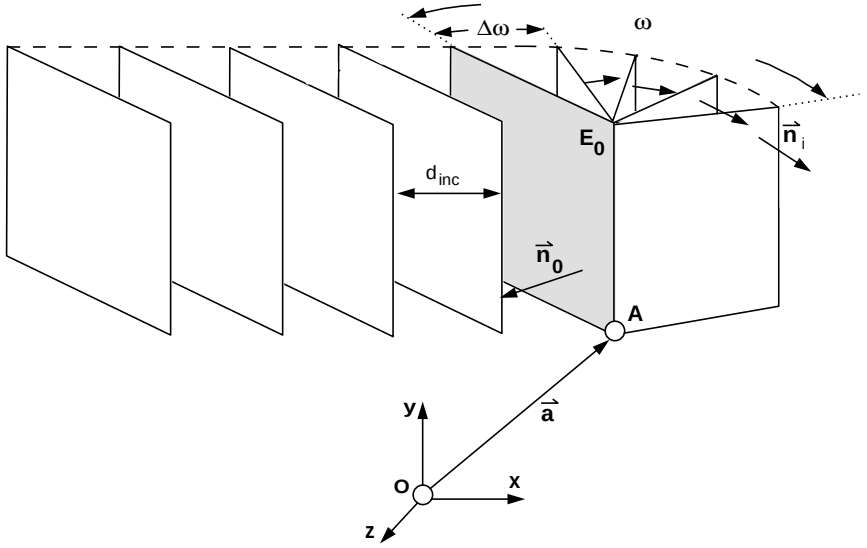


Abb. 4.25: Generierung paralleler und gedrehter Schnitte aus einer Grundebene E_0 heraus

Ein Vorteil des Generators liegt darin, daß sowohl beliebige einzeln generierte Schnittebenen, als auch verschiedene Schnittserien miteinander kombiniert und zu Schnittfolgen verarbeitet werden können. Selbst bereits auf Platte gespeicherte Schnittserien können jederzeit wiedergelesen und dazugenommen werden. Eine solche Folge ist als Animation darstellbar, so daß z.B. der Eindruck eines durch eine Struktur wandernden Schnittes ensteht.

4.3.5.2 Das Verfahren zur Berechnung der Schnitte

Die Berechnung eines Schnittes erfolgt in zwei Schritten. An erster Stelle steht die Ermittlung aller Schnittpunkte mit den Elementkanten sowie die Aufstellung der Listen, die für eine anschließende Referenzierung dieser Punkte erforderlich sind. An zweiter Stelle steht die Referenzierung und Erzeugung der Polygondaten für die Ausgabe der Visualisierungsgitterdaten. Das Verfahren der Referenzierung bleibt, mit dem Unterschied, daß anstelle der Isopunkte Schnittpunkte zugrundeliegen, dasselbe wie bei den Isoflächen. Deshalb kann sich hier die Beschreibung auf die Berechnung der Schnittpunkte beschränken.

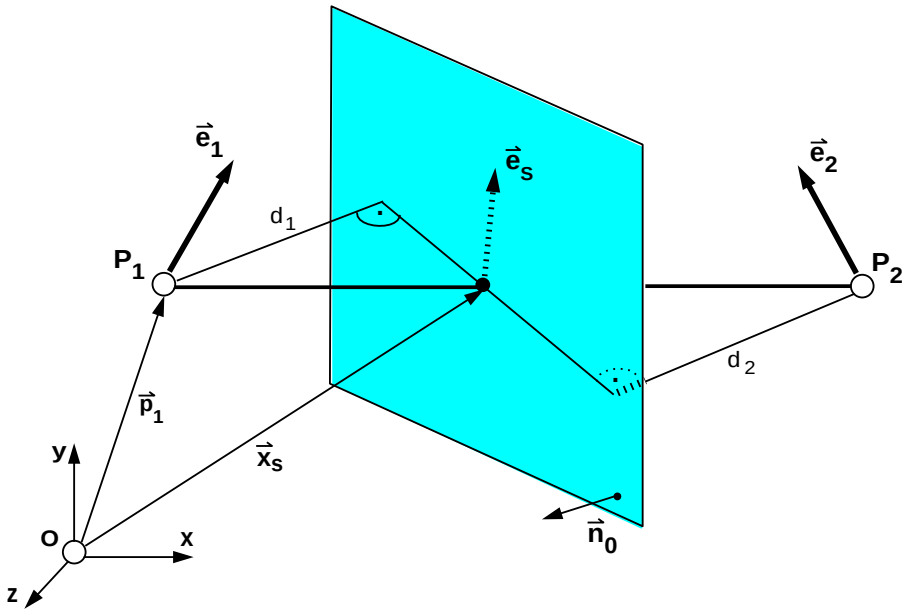


Abb. 4.26: Interpolation eines Schnittpunktes X_s sowie seines Ergebnisvektors $\vec{e}_s$

Zunächst wird ein Datenfeld *dist_node*() angelegt, in das die Abstände sämtlicher Elementknoten zu einer Schnittebene eingetragen werden. Der Abstand d_i eines Punktes $\vec{p}_i$ zu einer Ebene ergibt sich durch Einsetzen seines Koordinatenvektors in die Hessesche Normalform der Ebenengleichung zu

$$d_i = \vec{n}_0 * (\vec{a} - \vec{p}_i), \tag{4.12}$$

wobei $\vec{a}$ der Vektor des Aufpunktes auf die Ebene und $\vec{n}_0$ ihr normierter Normalenvektor ist.
Es folgt die Berechnung der Koordinaten und Ergebnisvektoren für alle Schnittpunkte der Ebene mit den Elementkanten. Kantenweise werden die Abstände der beiden Kantenknoten bezüglich der Ebene auf ihr Vorzeichen hin geprüft. Ist die Summe der negativen Vorzeichen 1 heißt dies, daß die Ebene zwischen den beiden Kno-

ten verläuft und ihre Verbindungslinie, also die Kante, schneidet. Aus den beiden Knotenabständen sowie der Kenntnis der Ergebnisvektoren an den Knotenpunkten lassen sich Ort und Ergebnisvektor des Schnittpunktes interpolieren, wie es aus Abbildung 4.26 zu ersehen ist.
Die Interpolationsgleichung für den Schnittpunkt X_s lautet:

$$\vec{x}_s = \vec{p}_1 - (\vec{p}_2 - \vec{p}_1) * \frac{d_1}{d_2 - d_1}. \tag{4.13}$$

Für den Ergebnisvektor $\vec{e}_s$ an diesem Schnittpunkt gilt:

$$\vec{e}_s = \frac{d_2 * \vec{e}_1 - d_1 * \vec{e}_2}{d_2 - d_1}. \tag{4.14}$$

4.3.5.3 Berechnung der Restgeometrien

Ein Schnitt teilt einen Körper in zwei Stücke. Die verbleibende Struktur, von der die Schnittnormale wegzeigt, ist die Restgeometrie, die andere Struktur wird verworfen. Der Algorithmus bewertet die Elemente nach drei Kategorien: Entweder liegen sie vollständig außerhalb der Restgeometrie, vollständig innerhalb, oder sie werden geschnitten. Demzufolge werden Elemente, die nicht geschnitten werden, unverändert in die Ausgabeliste übernommen, sofern sie negative Knotenabstände aufweisen, also der Restgeometrie angehören.
Die Schwierigkeit besteht in der Behandlung geschnittener Elemente, die zu Elementstümpfen bzw. Restelementen werden. In einfachen Fällen bleiben Tetraeder, Pentaeder oder Hexaeder übrig. Überwiegend hat man es jedoch nicht mit Standardelementen zu tun, sondern mit Restelementen, wie sie in Abbildung 4.27 dargestellt sind.
Es werden nun drei Verfahren vorgestellt, wie diese Restelemente für die Darstellung aufbereitet werden. Definitionsgemäß werden sie hier mit den Begriffen Linien-, Flächen- und Raumverfahren belegt.

- Linienverfahren:
  Im einfachsten Verfahren, dem Linienverfahren, werden die Elemente durch ihre Kanten idealisiert. Da jede Kante als eindimensionales Linienelement übernommen wird, eignet sich das Linienverfahren ausschließlich für eine spätere Darstellung als Drahtmodell.
- Flächenverfahren:
  Dieses Verfahren, bei dem die durch 2D-Elemente idealisierte Oberfläche eines Restelements übernommen wird, ist sehr wohl zur dreidimensionalen Darstellung tauglich, jedoch nicht für die interne Weiterverarbeitung vorgesehen. Eine

nachfolgende Schnittebene beispielsweise, kann gegebenenfalls diese Elemente kein zweites Mal sinnvoll schneiden.

- Raumverfahren:
  Das Raumverfahren ist das einzige Verfahren, das die ursprüngliche Information auch im Bereich geschnittener Elemente erhält. Um die Dreidimensionalität beibehalten zu können, müssen die Elementstümpfe jedoch in Standard-3D-Elemente zerlegt werden. In Abbildung 4.27 ist eine Restkörperzerlegung am Beispiel eines Hexaeders exemplarisch für drei Fälle ausgeführt. Der Normalenvektor $\vec{n}$ deutet die jeweilige Schnittebene an.

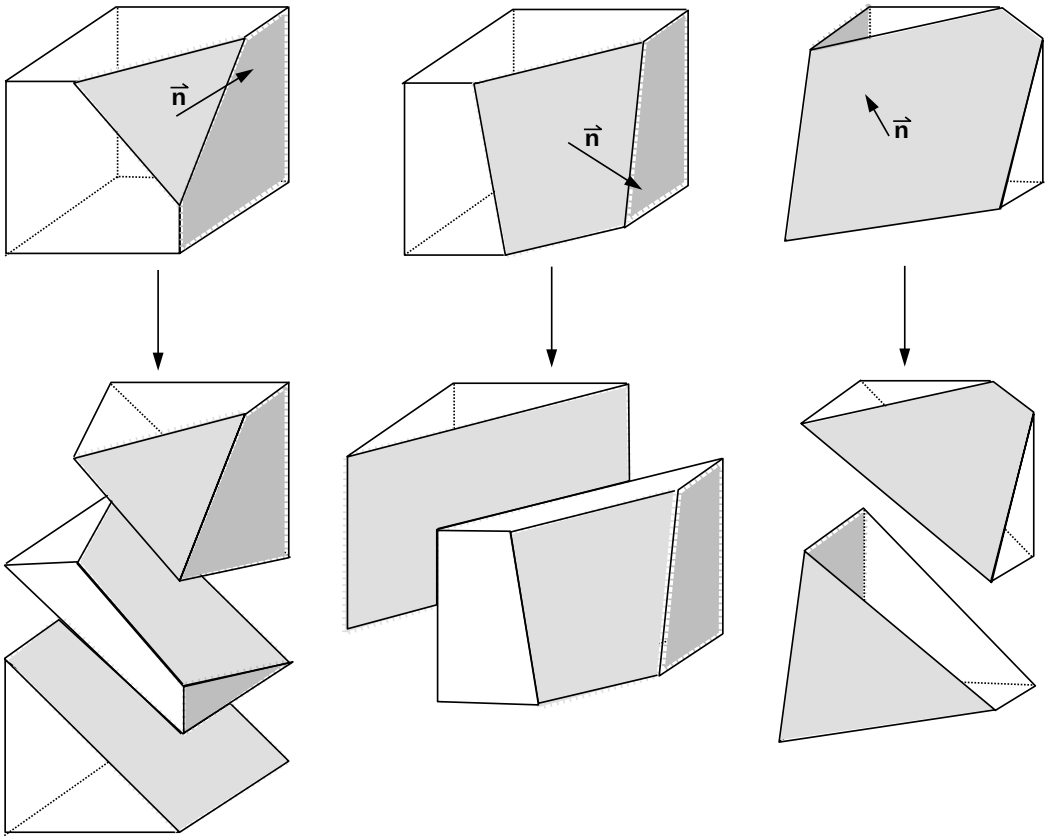


Abb. 4.27: Drei Beispiele für die Zerlegung von Elementstümpfen

Für Restkörper, deren Schnittflächen Fünf- oder Sechsecke aufweisen (Abb. 4.27 rechts), ist eine Zerlegung im allgemeinen mit einer sehr aufwendigen Fallbetrachtung verbunden. In solchen Fällen ist es besser, das ungeschnittene Ausgangselement noch vor dem Schneiden in Tetraeder aufzuteilen und die Tetraeder dann einzeln zu behandeln. Elementstümpfe, die aus Tetraederschneidungen resultieren, sind hingegen stets einfach zerlegbar.

4.3.5.4 Ergebnisbeispiel für Schnitte und Restgeometrien

Den Abbildungen 4.28 und 4.29 liegt derselbe Datensatz einer axialsymmetrischen Düse zugrunde, der bereits in Abbildung 4.23 vorgestellt wurde. Abbildung 4.28 zeigt eine Serie paralleler Schnitte durch die Düsenströmung mit Darstellung von Geschwindigkeitswerten.
Abbildung 4.29 zeigt zweimal denselben Schnitt durch die erwähnte Düse zusammen mit der verbleibenden Restgeometrie. Über eine Option wurde der Restgeometrie im Bild oben eine beliebige Farbe zugewiesen und im Bild unten die Strömungsgeschwindigkeiten dargestellt.

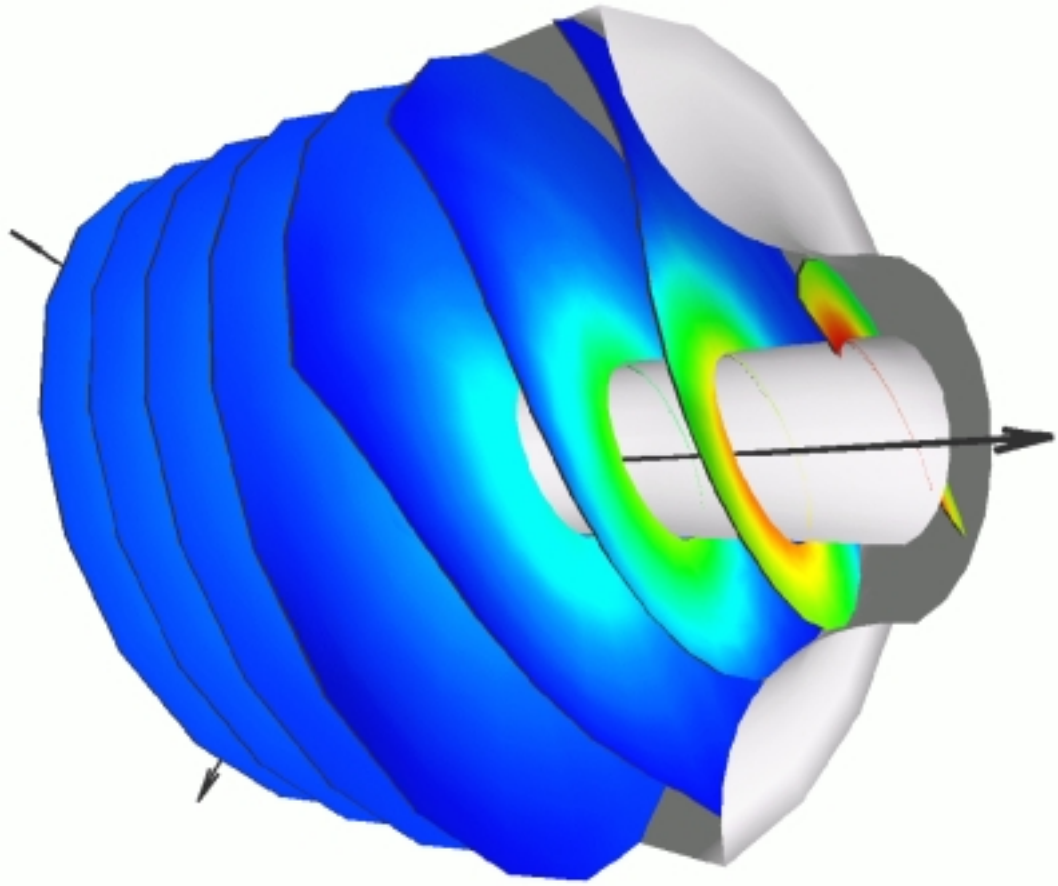


Abb. 4.28: Serie paralleler Schnitte durch eine kaltdurchströmte Düse mit aufprojizierten skalaren Geschwindigkeitswerten

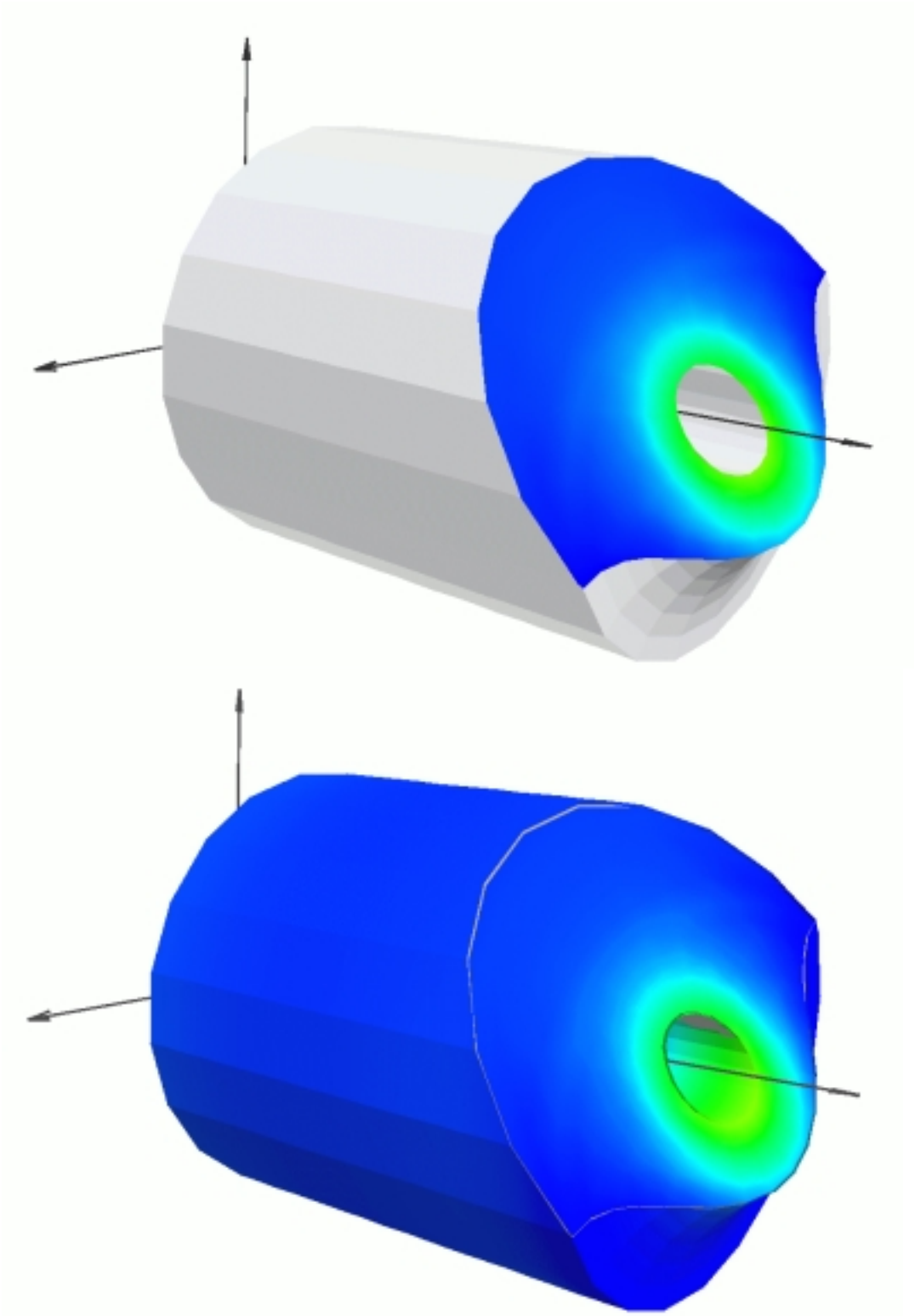


Abb. 4.29: Schnitt durch eine kaltdurchströmte Düse mit Restgeometrie einmal einfarbig dargestellt und einmal schattiert mit skalaren Geschwindigkeitswerten

4.3.6 Volumenausschnitte

Die Darstellung von Volumenausschnitten ergibt sich als Konsequenz aus der vorangegangenen Behandlung von Schnitten und Restgeometrien.
Oftmals sind in einem dreidimensionalen Strömungsgebiet bestimmte Bereiche von besonderem Interesse und sollen daher weiter untersucht und graphisch aufbereitet werden. Andererseits ist bei großen Rechenproblemen mit einigen Millionen Knoten von vornherein eine Reduktion der Daten unumgänglich, einmal um sie überhaupt auf eine Workstation zu bekommen und zum anderen, um große Datenmengen in einem sinnvollen Zeitrahmen darstellen zu können.
Die Darstellung von Volumenausschnitten beginnt mit der Festlegung eines Kontrollvolumens, d.h. eines dreidimensionalen, quaderförmigen Raumes, der durch sechs interaktive Eingaben eindeutig bestimmt ist, nämlich die drei Koordinaten der vorderen, linken, unteren Ecke, sowie die drei Kantenlängen Δx, Δy und Δz. Auf diesem, den Ausschnitt markierenden Quader, wird nun das gesamte Maschennetz reduziert, indem es sechs Mal hintereinander geschnitten wird, mit den Ebenen, die durch die Seitenflächen des Quaders vorgegeben sind.

4.3.6.1 Das Verfahren zur Berechnung eines Volumenausschnitts

In einem ersten Grobdurchlauf werden zunächst alle Elemente mit 1 markiert, sofern sie ganz und mit 2 wenn sie teilweise im Inneren des Quaders liegen. Die Knoten aller markierten Elemente werden mit 1 markiert. Für die markierten Elemente, Knoten und deren Ergebnisvektoren werden neue, reduzierte Datenfelder angelegt. Elemente, die von einer oder mehreren Ebenen geschnitten werden und folglich mit 2 markiert wurden, tauchen dort nicht auf, sondern werden gesondert einer Arbeitsliste zugeführt.
In einem Feindurchlauf findet nun das eigentliche Schneiden statt. Hierbei wird ebenenweise die jeweils aktuelle Arbeitsliste abgearbeitet. Vor jedem Durchlauf werden lokale Elementkanten und Kantenknotenlisten für die Elemente in der Arbeitsliste angelegt. Es folgt die kantenweise Berechnung der Schnittpunkte. Gefundene Schnittpunkte und deren Ergebnisdatenvektoren werden den oben genannten reduzierten Knotendatenfeldern angefügt.
In einem sich anschließenden Referenzierungsschritt wird die Arbeitselementeliste fortwährend aktualisiert, in dem das gerade zum Schneiden bestimmte Element durch das geschnittene Element bzw. das Restelement ersetzt wird. In den Fällen, in denen eine Restelementzerlegung erforderlich ist, um Standard-3D-Elemente zu erhalten (siehe Abschnitt 4.3.4.3), ersetzt nur eines der neu entstandenen Elemente das ursprüngliche Element, während alle übrigen der Arbeitsliste angefügt werden.

Die vollständig aktualisierte Arbeitsliste dient der nächsten Schnittebene als Grundlage und so fort. Zur Verdeutlichung ist der Ablauf dieses Verfahrens in Abbildung 4.30 skizziert.

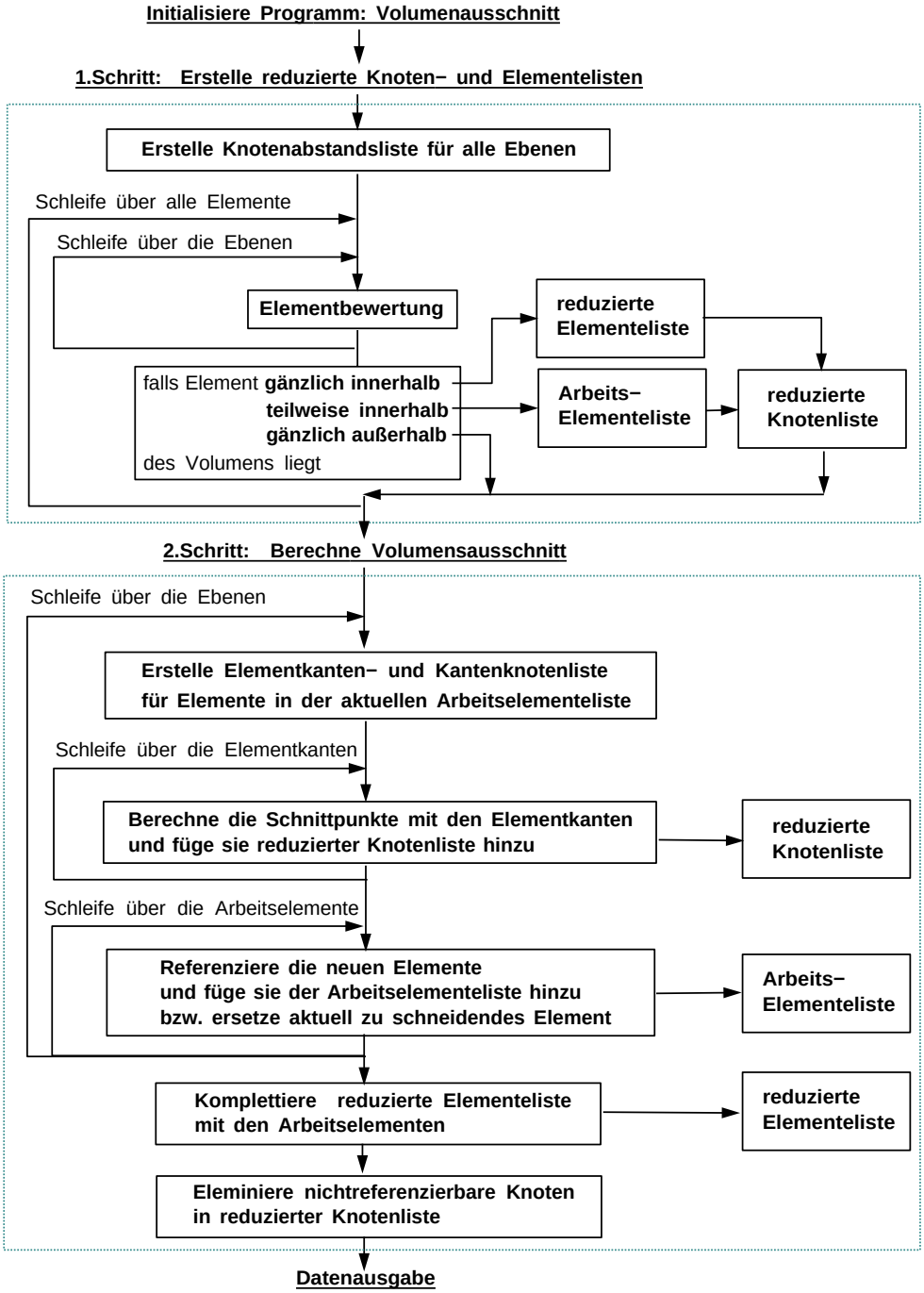


Abb. 4.30: Verfahren zur Berechnung von Volumenausschnitten

Als Ergebnis wird so ein zusammenhängendes, räumliches Gebiet erzeugt, in welchem die Konnektivität erhalten geblieben ist. Allerdings enthalten die Knotenlisten überzählige, nicht referenzierbare Knoten, die jedoch durch geeignete Standardverfahren leicht entfernt werden können.
Auf Wunsch kann die Generierung von 2D-Randelementen auf der Oberfläche des Kontrollvolumens erfolgen, entweder zusätzlich zu den 3D-Elementen oder aber ausschließlich. Ergebnisse einer Volumenausschnittsberechnung können von anderen Moduln weiterverarbeitet werden, z.B. zu Isoflächen.

4.3.6.2 Ergebnisbeispiel für Volumenausschnitte

Abbildung 4.31 illustriert wiederum am Beispiel der axialsymmetrischen Düse die begrenzenden Oberflächen von vier Volumenausschnitten. Auf den Oberflächen wird eine Variation aufprojizierter Druckwerte gezeigt.

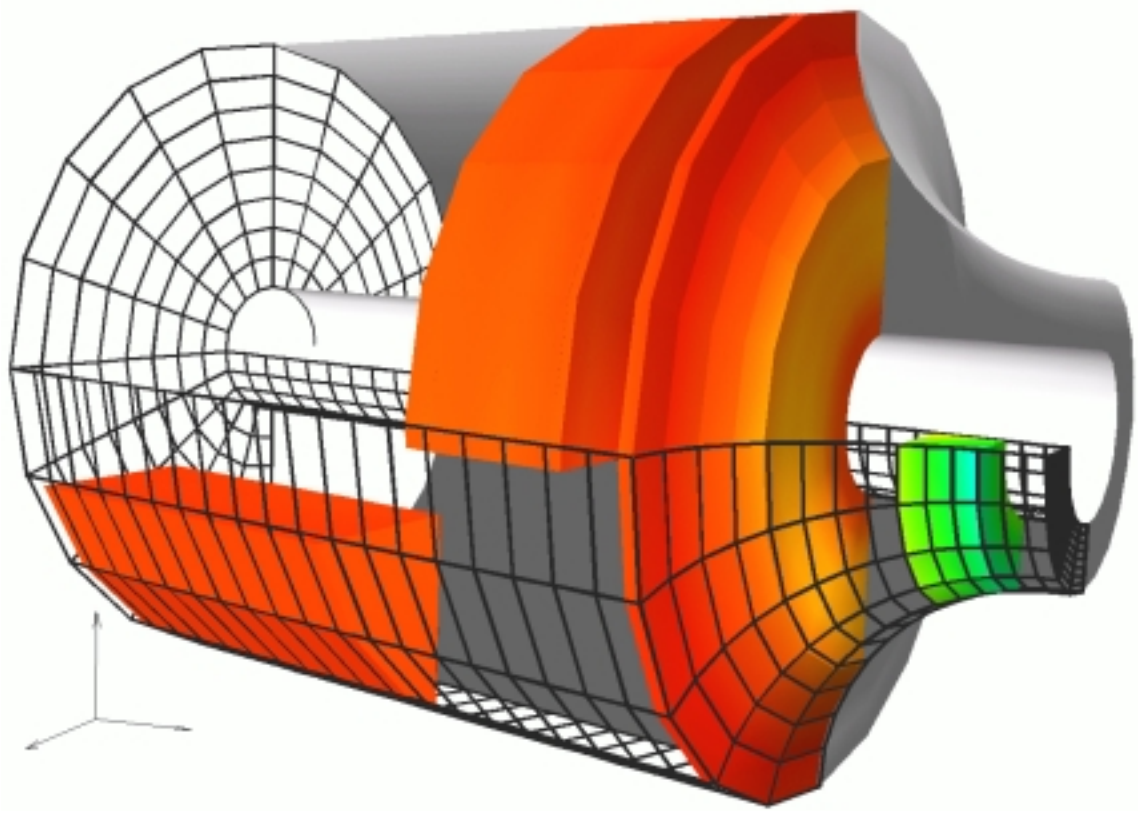


Abb. 4.31: Darstellung von Volumenausschnitten in einer kaltdurchströmten Düse

4.3.7 Vektoren

Als Basis für die Darstellung von Vektoren dient die Ergebnisdatei aus der Simulationsrechnung, in der für jeden Elementknoten die drei Komponenten eines Vektors abgelegt sind. Das Vektormodul berechnet nun für jeden Knoten den dort angreifenden Vektor und liefert dem nachfolgenden Geometrie–Mapper–Modul die Polygondaten für die gewählte Art der Repräsentierung.

4.3.7.1 Repräsentierung eines Vektors

Aus einem Set interaktiver Eingaben stellt sich der Benutzer eine Schablone für einen Vektor her. Die Schablone legt Form und Aussehen eines Referenzvektors in einem lokalen Koordinatensystem fest, abhängig von Gestaltungsparametern wie etwa Pfeilungswinkel, Pfeillänge, Vektordicke, Skalierungsfaktor, Skalierungs- und Darstellungsmodus.
Der Darstellungsmodus bezieht sich auf die dreidimensionale Darstellung eines Vektors, der entweder als flächiger Pfeil, als Kügelchen mit Richtungsweiser oder als Liniensegment mit Kugelkopf repräsentiert werden kann (siehe Abb. 4.32).
Der Skalierungsmodus legt fest, nach welchen Gesichtspunkten sich die Darstellungslänge eines Vektors richtet. Die Skalierung kann auf drei Weisen erfolgen: linear, logarithmisch oder konstant. *Konstant* bedeutet, daß jedem Vektor, ungeachtet seines Betrages, dieselbe Länge zugewiesen wird. Bei *linear* und *logarithmisch* ist der Betrag das Maß für die Länge, die sich entweder linear oder logarithmisch zwischen der Minimal- und der Maximallänge bewegt. Mit dem Skalierungsfaktor wird das Gesamtbild der Vektoren unabhängig vom eingestellten Skalierungsmodus skaliert. Eine weitere Option erlaubt die Projektion einer Ergebnisgröße auf die Vektoren *(data mapping)*.

4.3.7.2 Berechnungsverfahren für Vektoren

Zur Berechnung eines Vektors wird die in lokalen Koordinaten erstellte Vektorschablone mittels einer Beziehungsmatrix auf die globalen Koordinaten transformiert. Die Matrix enthält Angaben über die Lage des Vektors, also seinen Angriffs- und seinen Endpunkt, sowie im Dreidimensionalen eines dritten Punktes, durch den die Ausrichtung der Vektorfläche bestimmt ist. Die beiden, die Vektorfläche aufspannenden Einheitsvektoren, bilden ein orthogonales Rechtssystem und werden im Folgenden Angriffsvektor und Orientierungsvektor genannt (siehe Abb. 4.33).
Anders als im zweidimensionalen Fall, wo nur ein einziger Orientierungsvektor exi-

stiert, stellt sich im Dreidimensionalen generell das Problem der Orientierung einer Vektorfläche im Raum, da für die zweite Richtung eine Mannigfaltigkeit erster Ordnung besteht. Hier gilt es, aus der Menge der Vektoren, die alle in der Angriffsebene, d.h. der Ebene normal zum Angriffsvektor liegen, denjenigen zu ermitteln, der eine sinnvolle graphische Darstellung ermöglicht.
Zwei Möglichkeiten zur Festlegung der Orientierung bieten sich hier an. Bei der einen wird als Orientierungsrichtung die kürzeste Komponente des Angriffsvektors projiziert auf die Angriffsebene genommen. Bei der zweiten wird die Richtung der Wirbelstärke $\vec{\omega}$ am Angriffspunkt berechnet und als Orientierungsvektor verwendet.

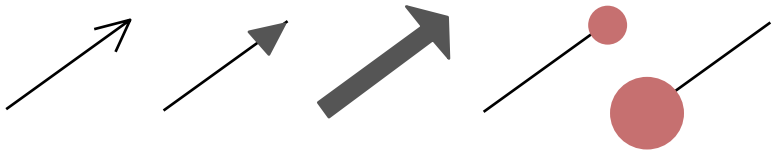


Abb. 4.32: Die unterschiedlichen Darstellungsweisen von Vektoren

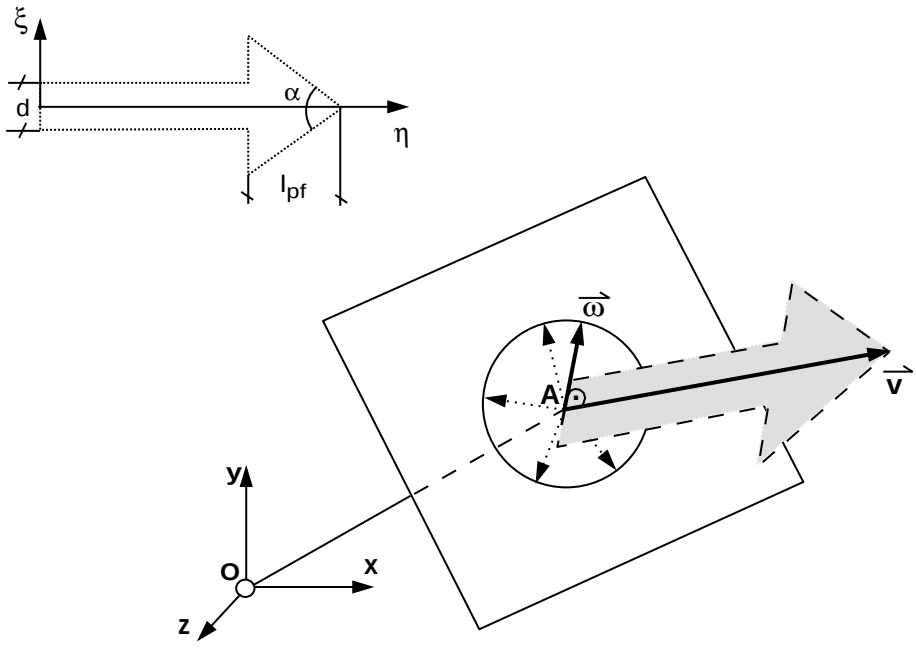


Abb. 4.33: Zur Orientierung einer Vektorfläche im Raum

4.3.7.3 Berechnung der Wirbelstärke als Orientierungsrichtung

Die Berechnung der Orientierung soll hier für die Wirbelstärke $\vec{\omega}$ demonstriert werden, die auch für die Darstellung von Stromlinienbändern benötigt wird.
Die Komponenten der Wirbelstärke werden näherungsweise aus den Geschwindigkeitsdifferenzen in einer räumlichen Umgebung unmittelbar um den Angriffspunkt A des Vektors berechnet. Als Umgebung wird ein Tetraeder gewählt, der durch vier Umgebungspunkte U_i mit $0 \leq i \leq 3$ definiert ist, die auf den drei Achsen eines lokalen kartesischen Koordinatensystems mit U_0 als Ursprung liegen. Als Abstand vom Ursprung wird eine Länge ds gewählt, die 10 Prozent der Länge einer beliebigen, an den Elementknoten grenzenden Kante beträgt. Der vorgegebene Tetraeder besitzt drei aufeinander senkrecht stehende kartesische, sowie eine um 45 Grad zu allen drei Raumachsen geneigte Seitenfläche. Der Umgebungspunkt U_0 wird so gewählt, daß A Schwerpunkt des Tetraeders wird (Abb. 4.34).

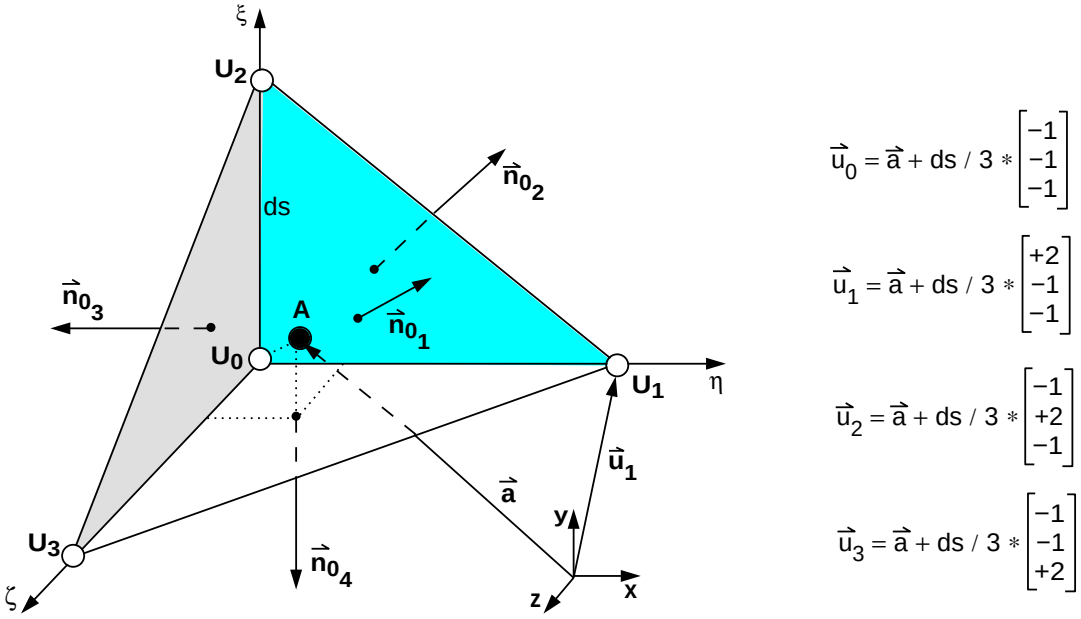


Abb. 4.34: Die tetraederförmige Umgebung um den Angriffspunkt A

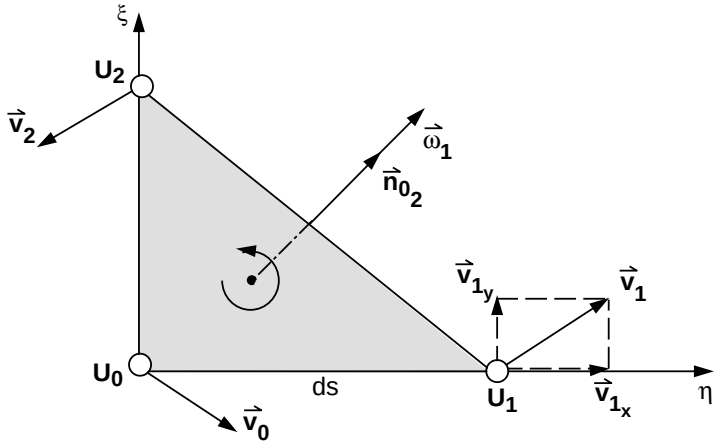
Es gilt nun, den Drehvektor $\vec{\omega}$ für dieses abgeschlossene Volumen zu berechnen. Dazu werden zuerst die Geschwindigkeitsvektoren an den Umgebungspunkten ermittelt, welche aus den Knotendaten derjenigen Elemente in der Nachbarschaft interpoliert werden, die den jeweiligen Umgebungspunkt enthalten. Es folgt für jede der vier Seitenflächen des Tetraeders die Berechnung der Zirkulationen Γ_i, die aus dem Li-

nienintegral der jeweiligen Umrandung nach der Beziehung

$$\Gamma = \oint_S \vec{v} * d\vec{s} \tag{4.15}$$

gewonnen werden. In Abbildung 4.35 ist die Berechnung Γ_2 für die zweite Tetraederseitenfläche skizziert.
Der Drehvektor $\vec{\omega}_i$ jeder Seitenfläche ist parallel zum Normalenvektor $\vec{n}_{0_i}$ der Seitenfläche. Der Betrag des Drehvektors ergibt sich aus dem Stokesschen Zirkulationssatz [80], der besagt, daß die Zirkulation um die Randkurve einer beliebigen räumlichen Fläche gleich dem doppelten Wert des Wirbelstroms durch diese Fläche ist. Der Wirbelstrom bedeutet hier das Flächenintegral über die Drehung.
Allgemein ausgedrückt lautet der Stokessche Zirkulationssatz:

$$\Gamma = \oint_S \vec{v} * d\vec{s} = 2 * \int_A \vec{\omega} * d\vec{A}. \tag{4.16}$$



$$\Gamma_2 = \oint_s \vec{v} * d\vec{s} \cong ds * [v_{0x} - v_{2y} + 1/2 * (v_{1y} - v_{1x})]$$
$$\omega_2 = \Gamma_2 / (2 * A_2)$$
$$\vec{\omega}_2 = \omega_2 * \vec{n}_{02} = \Gamma_2 / (2 * A_2) * \vec{n}_{02}$$

Abb. 4.35: Zur Ermittlung der Zirkulation Γ_2 und des Drehvektors $\vec{\omega}_2$ der Fläche 2

Der Betrag ω_i der i-ten Teilfläche ergibt sich demnach zu

$$\omega_i = \frac{\Gamma_i}{2 * A_i}, \tag{4.17}$$

wobei A_i ihr Flächeninhalt ist. Zuletzt erhält man die Gesamtdrehung $\vec{\omega}$ durch

Addition der einzelnen $\vec{\omega}_i$.

$$\vec{\omega} = \sum_{i=1}^{4} \omega_i * \vec{n}_{0_i}. \tag{4.18}$$

4.3.7.4 Ergebnisbeispiel für Vektoren

Abbildung 4.36 zeigt als Beispiel aus der Thermodynamik eine Bénard-Konvektionsströmung, die sich z.B. dann einstellt, wenn ein abgeschlossenes Volumen, - hier in einem Würfel mit vier kalten und zwei gegenüberliegenden heißen Wänden, - sich selbst überlassen wurde [9]. Im Bild sind die Vektoren des Geschwindigkeitsfeldes in gleicher Farbe und mit konstanter Länge dargestellt.

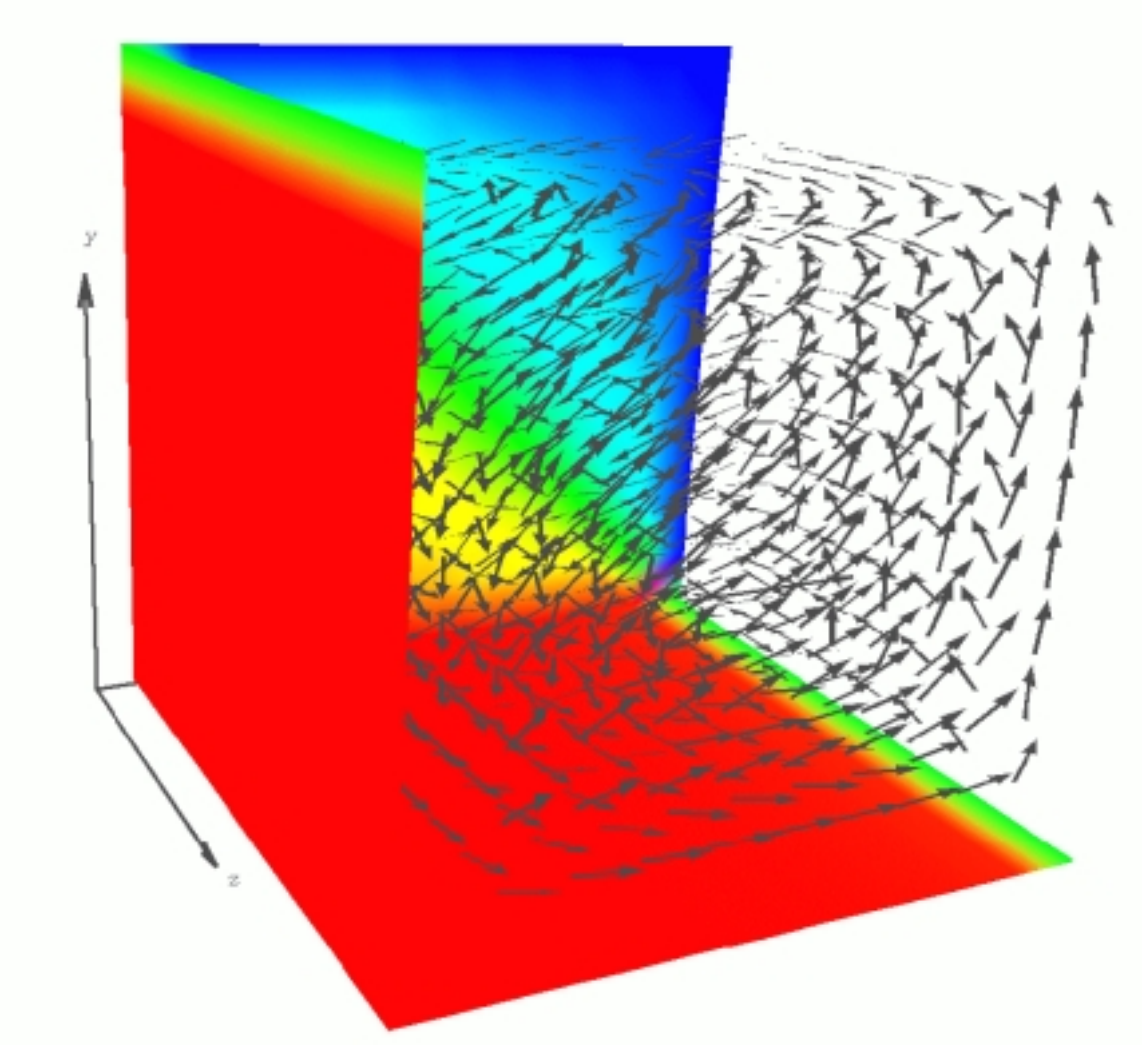


Abb. 4.36: Vektoren in einer Bénard-Konvektionsströmung

4.3.8 Stromlinien und Bahnlinien

4.3.8.1 Definitionen

Eine **Stromlinie** ist zu einem bestimmten Zeitpunkt als diejenige Kurve definiert, deren Richtung in jedem Raumpunkt der dort vorhandenen Richtung des Geschwindigkeitsvektors entspricht. Stromlinien vermitteln sozusagen ein Momentbild des augenblicklichen Strömungszustandes. Es ist dies die Eulersche Betrachtungsweise, die nur bei stationären Strömungen ein korrektes Bild des Strömungsvorganges wiedergibt.
Im instationären Fall spricht man von **Bahnlinien**, die erhalten werden, wenn die Bewegung masseloser Flüssigkeitsteilchen verfolgt wird. Es ist dies die substantielle Betrachtungsweise nach Lagrange, die der zeitlichen Änderung der Geschwindigkeit an jedem Raumpunkt Rechnung trägt. Nur wenn das Geschwindigkeitsfeld zeitlich konstant, die Strömung also stationär ist, fallen Strom- und Bahnlinien zusammen [73].

4.3.8.2 Hintergrund

Bei numerisch behandelten Strömungen handelt es sich nur selten um reine Potentialströmungen; vielmehr treten Unstetigkeiten wie Stöße, Grenzschichten, Trennflächen oder Wirbel auf, was mit sprunghaften Änderungen des Potentials einhergeht. Das Geschwindigkeitsfeld ist also im allgemeinen nicht mehr zirkulationsfrei.
Für die Stromlinienberechnung scheiden deshalb diejenigen Lösungsverfahren von vornherein aus, die die Stromfunktion zu Hilfe nehmen.
Als weiterer Grund hierfür kann angeführt werden, daß die Stromlinienberechnung zumeist im Rahmen des Postprozessings durchgeführt wird, - also entkoppelt vom Strömungslöser, - so daß den Visualisierungsprozessen die Kenntnis über den physikalischen Hintergrund des berechneten Problems fehlt.
Das hier eingesetzte Verfahren ist ganz auf Vektorfelder abgestimmt, deren einzelne Vektoren an den Raumgitterpunkten eines Finite Element- oder Finite Volumen-Netzes angreifen. Im wesentlichen wurde das sogenannte Shoot-Verfahren [81] weiterentwickelt, welches sich eines Vektorfeldes bedient, ohne die Kenntnis der zugehörigen physikalischen Gesetzmäßigkeiten vorauszusetzen.

Nach dem Shoot-Verfahren werden sowohl Stromlinien als auch Bahnlinien berechnet, letztere unter Berücksichtigung der zeitlichen Änderung des Geschwindigkeitsfeldes. Im instationären Fall wurde bei der Programmentwicklung davon ausgegangen, daß für eine konkrete Anzahl von Zeitschritten die zugehörigen momentanen

Strömungszustände als Datensätze vorliegen. Zur Berechnung eines Bahnlinienpunktes wird also die zum jeweiligen Zeitpunkt gehörige örtliche Geschwindigkeit linear aus den einzelnen Momentaufnahmen interpoliert.
Darüber hinaus läßt sich das Verfahren in ein Analyseprogramm integrieren, wobei auch hier nach jedem Zeitschritt der neue Ort $x_1(t)$ der losgelassenen Teilchen aus dem alten Ort $x_0(t-1)$ zu ermitteln ist, unter Berücksichtigung der zugehörigen Geschwindigkeiten $v_0(t,x)$, $v_0(t-1,x)$ und $v_1(t-1,x)$, die durch ein Runge-Kutta-Verfahren zweiter Ordnung die Ortsbestimmung beeinflussen.

4.3.8.3 Interaktive Startpunktegenerierung

Die Berechnung von Stromlinien beginnt mit der Generierung der Startpunkte. Mit dem "*point generator*" steht zu diesen Zweck ein eigenständiges Modul zur Verfügung, mit dem sich, ähnlich wie mit dem Schnittegenerator, verschiedene Generierungstechniken miteinander kombinieren lassen.
Zur Wahl steht hier die Erzeugung einzelner Punkte durch Angabe ihrer Koordinaten und ferner die Generierung einer Punkteschar innerhalb eines abgeschlossenen Volumens, eines Flächenausschnitts oder eines Linienabschnitts. Eine Punkteschar wird bestimmt durch die Angabe der Ecke unten vorne links des Volumens, die Kantenlängen Δx, Δy, Δz, sowie die Anzahl zu generierender Punkte nx, ny, nz in jede kartesische Raumrichtung. Gleichsam können Punkte in einem Flächenausschnitt oder auf einer geraden Linie erzeugt werden, indem man einzelne Kantenlängen bzw. die Anzahl der Punkte in ausgewählten Richtungen zu Null setzt.
Generierte Punktesequenzen lassen sich als Dateien ausgeben, wiederlesen und zu neugenerierten Sequenzen hinzunehmen. In diesem Zusammenhang können Sequenzen und Punkte in der Reihenfolge ihrer Generierung gelöscht werden. Generierte Sequenzen oder Einzelpunkte kann der Benutzer in der Aufbauphase beliebig oft korrigieren und durch Anklicken von "*add points*" schließlich der Punktefolge hinzufügen. Zur visuellen Kontrolle können die Startpunkte zusammen mit der FE-Struktur am Bildschirm dargestellt werden.

4.3.8.4 Das Shoot-Verfahren

Berechnung

Die Berechnung einer Stromlinie erfolgt im Modul *flow server* mittels des oben erwähnten Shoot-Verfahrens. Beginnend mit einer aus den Geometrieabmessungen geschätzten Anfangsschrittweite ds_0, wird vom Startpunkt aus in Richtung des Ortsvektors "geschossen". Auf diese Weise wird der Zielpunkt $\vec{x}_1$ erhalten zu

$$\vec{x}_1 = \vec{x}_0 + \frac{\vec{v}_0}{|\vec{v}_0|} * ds_0. \tag{4.19}$$

Der neue Geschwindigkeitsvektor $\vec{v}_1$ wird aus den Vektoren der unmittelbaren Umgebung des Zielpunktes interpoliert, genau gesagt aus den Vektoren der Knotenpunkte desjenigen Elements, das den Zielpunkt enthält. Als Maß für die Genauigkeit der Approximation an die Stromlinie, wird das Skalarprodukt zwischen dem Zielvektor und dem Ausgangs- bzw. Startvektor herangezogen. Je nach Größe des Skalarproduktes, wird die Schrittweite entweder verkleinert oder vergrößert. Überdies wird immer ein zweites Mal geschossen, mit einer veränderten Anfangsschußrichtung, die mit einem einfachen Runge-Kuttaverfahren aus Start- und Zielvektor gewonnen wird. Auf diese Weise gelingt es, der exakten Stromlinie möglichst nahe zu kommen. Dieser Vorgang wird wiederholt, wobei der alte Zielpunkt neuer Startpunkt ist.
Liegt eine instationäre Strömung in Form von zeitschrittbeschreibenden Dateien vor, gibt der Benutzer ein Zeitinkrement an, gemäß der zeitlichen Änderung des Vektorfeldes an festem Ort. Das Zeitinkrement legt somit die globale Änderungsgeschwindigkeit der Strömung fest.

Abbruchbedingungen

Verschiedene Abbruchbedingungen beenden das Shoot-Verfahren. In diesem Zusammenhang sei erwähnt, daß jeder Zielpunkt in der Nachbarschaft des Startpunktes liegen muß. Damit ist zum einen gewährleistet, daß bei der Ermittlung jedes neuen Partikelortes nicht jedesmal alle Elemente in die Suche einbezogen werden müssen, und zum anderen, daß bei großen Schrittweiten zwar die Skalarproduktbedingung erfüllt ist, jedoch ein Gebiet hoher Gradienten nicht aufgelöst bzw. übersprungen wurde.
Die Nachbarschaft um ein Startelement ergibt sich aus seinen zugehörigen Nachbarelementen erster oder zweiter Ordnung. Die Generierung von Nachbarschaftslisten wurde bereits in Abschnitt 4.2.2 besprochen. Liegt ein Zielpunkt außerhalb dieses Bereichs, wird automatisch die Schrittweite halbiert, dies jedoch nur so oft, bis ein Abbruchkriterium (Mindestschrittweite) erfüllt ist, denn es könnte sich ja um einen Strömungsrand handeln.
Auf ganz ähnliche Weise werden im Zweidimensionalen Stromlinien abgebrochen, die

in einem Wirbel umlaufen. Um das Aufspulen einer Stromlinie zu verhindern, wird die Stromlinie nach einem Umlauf geschlossen. Für die Schließbedingung werden die Nachbarn erster Ordnung herangezogen, so daß die Stromlinie dann geschlossen wird, wenn sie in die Nähe des Anfangsstartpunktes gerät. Dies bedeutet auch, daß die Kurve im Bereich der Anfangs- und Endpunkte geglättet werden muß.
Eine Abbruchbedingung in Wirbeln existiert im dreidimensionalen Fall wie im übrigen für Bahnlinien nicht, da die Teilchen sich hier nicht entlang einer geschlossenen Kurve bewegen, sondern auf einer Spiralbahn, die bestenfalls an einem der Ränder des Gebietes endet.
Schließlich sei die triviale Abbruchbedingung erwähnt, die der Benutzer selbst vorgibt, indem er die Maximalzahl von Shoot-Iterationen wählt.

Zusatzoptionen
Als zusätzliche Wahlmöglichkeiten bietet der *flow server* zum einen die Abbildung einer ausgewählten physikalischen Größe auf Stromlinien und Partikeln an und zum anderen eine Umkehrung des Vektorfeldes, so daß eine Stromlinie von einem beliebigen Ort aus in beide Richtungen verfolgt werden kann. Außerdem kann für eine sich anschließende Darstellung von Strombändern zu jedem Stromlinienpunkt der Drehvektor mitgeliefert werden.

Datenausgabe
Am Ende der Berechnungsprozedur wird ein Datenfeld ausgegeben, in welchem die Weg-Zeit-Beschreibung aller Partikeln vom Startpunkt bis zum Endpunkt abgelegt ist. Dieses STR-Format dient der Weiterverarbeitung durch besondere Geometrie–Mapper–Moduln, beispielsweise dem *streamline builder*, dem *particle tracer* oder dem *droplet tracer*, von denen in den folgenden Abschnitten die Rede sein wird. Desweiteren existieren Lesemodule, die im STR-Format abgelegte Dateien wieder zur Verfügung stellen.

4.3.8.5 Der Stromlinienbauer

Das Standardmodul für die Verarbeitung von Stromliniendaten ist der *streamline builder*, mit dem Stromlinien und Strombänder produziert werden können. Für die Darstellung wird entweder die auf die Stromlinien zu projizierende Größe ausgewählt oder aber die Farbe, in der die Stromlinien erscheinen sollen. Ferner kann die Auswahl einzelner Stromlinien erfolgen und die Zahl von Punkten pro Stromlinie festgelegt werden.

Strombänder

Strombänder dienen zum einen als Hilfsmittel zur visuellen Aufbereitung von Stromlinien, zum anderen können mit einem geeigneten Verfahren Drehungen in einem Vektorfeld sichtbar gemacht werden.

Die Schwierigkeit liegt in jedem Fall in der Ausrichtung der Bandbreite in jedem Stromlinienpunkt. Als Orientierungsrichtung wird vorzugsweise die örtliche Drehung $\vec{\omega}$ verwendet, die nach dem in Abschnitt 4.3.7.3 behandelten Verfahren berechnet wird.

Bereits während der Berechnung der Stromlinien wird optional zu jedem gefundenen Stromlinienpunkt i der normierte örtliche Drehvektor $\vec{\omega}_{0_i}$ mitgeliefert, der die Ausrichtung des Bandes an dieser Stelle bestimmt.

Die entgültige Einstellung der Bandbreite nimmt der Benutzer ebenfalls im Stromlinienbauer über einen Skalierungsfaktor vor. Dabei kann die Bandbreite konstant oder variabel nach dem Betrag des örtlichen Drehvektors respektive der Zirkulation gewählt werden.

4.3.8.6 Ergebnisbeispiele aus der Stromlinienberechnung

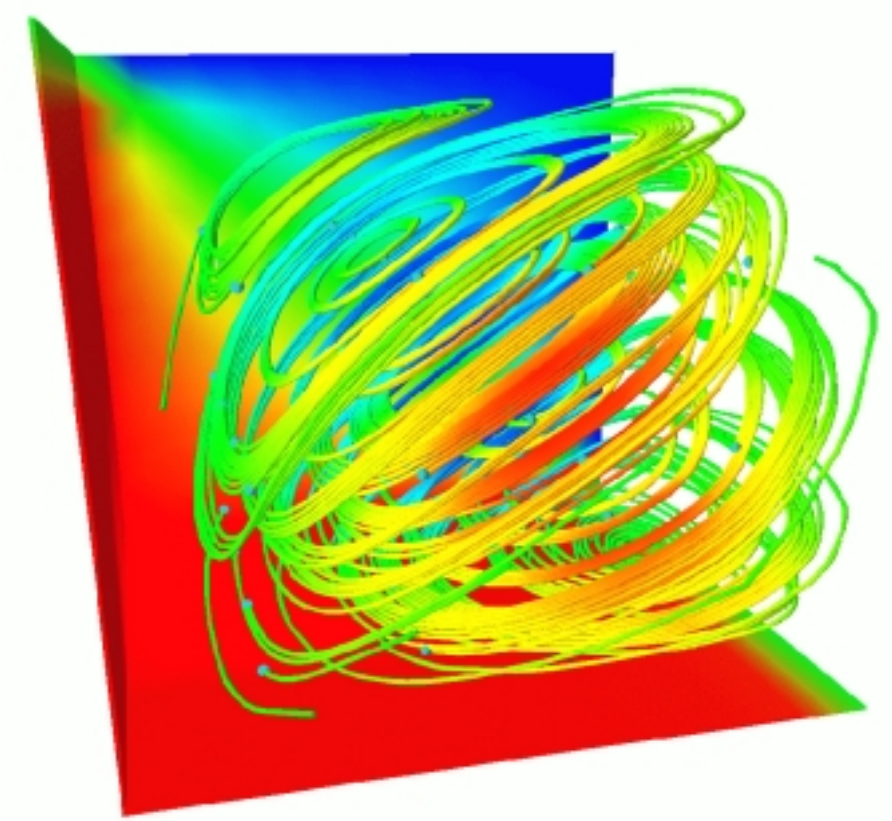


Abb. 4.37: Stromlinienbündel in einer Bénard-Konvektionsströmung

Abbildung 4.37 zeigt am Beispiel der Bénard-Konvektionsströmung stationäre Stromlinienbänder mit aufprojizierten Temperaturwerten. Die Berechnung einer Stromlinie wurde nach maximal 1000 Shoot-Iterationen abgebrochen.

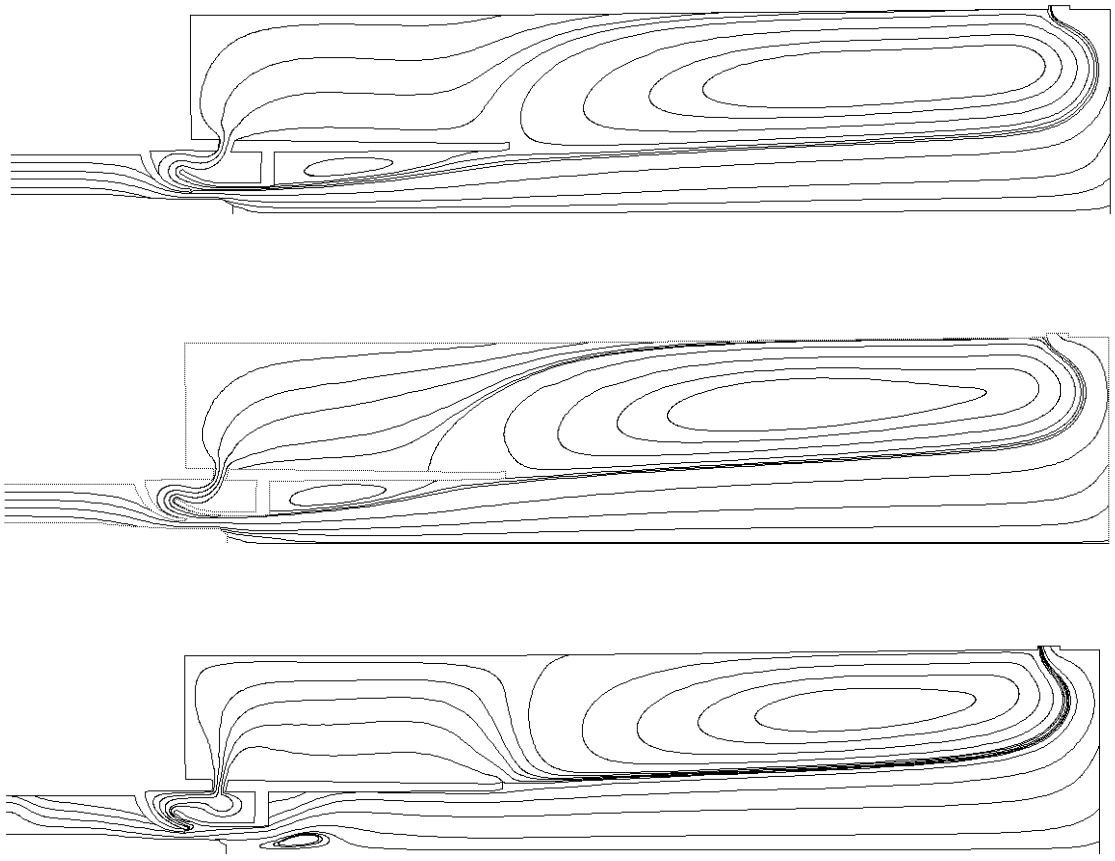


Abb. 4.38: Stromlinien in einem Methan-Luft-Brenner mit Abgasrückführung für die drei Drallzahlen 0, 0.7 und 2

Das zweite Beispiel (Abb. 4.38) zeigt eine axialsymmetrische Rechnung eines idealisierten Haushaltsbrenners mit Abgasrückführung. Die Methan-Luft-Vormischflamme wurde mit verschiedenen Eintrittsdrallzahlen simuliert [42]. Mit zunehmender Drallzahl wird die Rezirkulationszone im Brennraum kleiner, bis - wie aus der dritten Teilabbildung zu ersehen ist, - die Abgasrückführung versagt. Um den Verlauf der Stromlinien durch ganz bestimmte Orte zu erhalten, wurde hier ein Teil der Stromlinien mit Hilfe des *reverse mode* durch Umkehrung des Vektorfeldes in beide Richtungen verfolgt und zusammengesetzt.

Die folgenden Abschnitte 4.3.9 bis 4.3.11 behandeln Anwendungen, die sich unmittelbar aus den Daten einer Weg-Zeit-Beschreibung ergeben: die Verfolgung von Partikeln und die Verfolgung verschiedener Ausgangsgeometrien wie beispielsweise ein ebenes Flächensegment, die Oberfläche einer Kugel oder eines Kubus.

4.3.9 Verfolgung von Partikeln

Das Modul *particle tracing* generiert aus einer stationären oder instationären Weg-Zeit-Beschreibung masseloser Partikeln eine animierbare Bilderfolge, welche die Bewegung der Partikeln entlang zuvor berechneter Bahnen simuliert. Das Modul kann die Daten der Weg-Zeit-Beschreibung sowohl von einem *flow server* Modul beziehen als auch von einem speziellen Einlesemodul namens *read particle description*, das diese Beschreibung dann von einer im ASCII-Format vorliegenden Datei einliest.

Verfahrensweise
Der Benutzer gibt die Anzahl der zu generierenden Bilder bzw. Momentaufnahmen vor. Aus der Gesamtzeit der Bewegung und der Anzahl der Bilder resultiert so ein festes Zeitinkrement. Im Folgenden wird sukzessive zu jedem diskreten Zeitpunkt interpolativ der Ort der Partikeln berechnet und das zugehörige Bild generiert. Als Resultat wird so eine Bilderserie erhalten, die eine synchronisierte Bewegungsbeschreibung der Teilchen wiedergibt, Momentaufnahmen für jeden Zeitschritt.

Darstellungsarten
Verschiedene Darstellungsweisen werden unterstützt. In der einfachsten Form wird nur das Partikel dargestellt, indem an dessen Ort ein geometrisches Primitiv, vorzugsweise eine Kugel mit vorgebbarem Halbmesser, gezeichnet wird.
Eine andere Form ist die Darstellung des Teilchens mit seiner Spur, d.h. seines bis dato zurückgelegten Weges. Hier kann entweder der Gesamtweg oder die letzten k Zeitschritte nachgezeichnet werden. Darüber hinaus kann eine Spur einfarbig erscheinen oder in der Farbgebung variabel gehalten werden, gemäß einer aufprojizierten physikalischen Größe. Die Spur selbst kann als Linie oder als Band einstellbarer Breite dargestellt werden, wobei für die Ausrichtung des Bandes wiederum der lokale Drehvektor maßgeblich ist.
Weitere Optionen betreffen die Vorgabe einer Maximalzeit, die jedoch die Gesamtzeit der zugrundeliegenden Weg-Zeit-Beschreibung nicht überschreiten darf, ferner die Vorgabe einer maximalen Zahl von Stromlinienpunkten, ebenfalls die Weg-Zeit-Beschreibung betreffend, sowie die Auswahl nur bestimmer Partikel für die Darstellung.

4.3.9.1 Ergebnisbeispiel für eine Partikelverfolgung

Als Beispiel für eine Partikelverfolgung ist in Abbildung 4.39 wiederum das axialsymmetrische Problem einer kaltdurchströmten Düse dargestellt. Gezeigt wird der Ort von acht Partikeln zu sechs Zeitpunkten, einschließlich der bis dahin zurückgelegten Wegstrecke, die in Form eines Stromlinienbandes mit aufprojizierten Temperaturwerten dargestellt ist. Die rote Farbe gibt hier das Temperaturmaximum an, blau das Minimum.

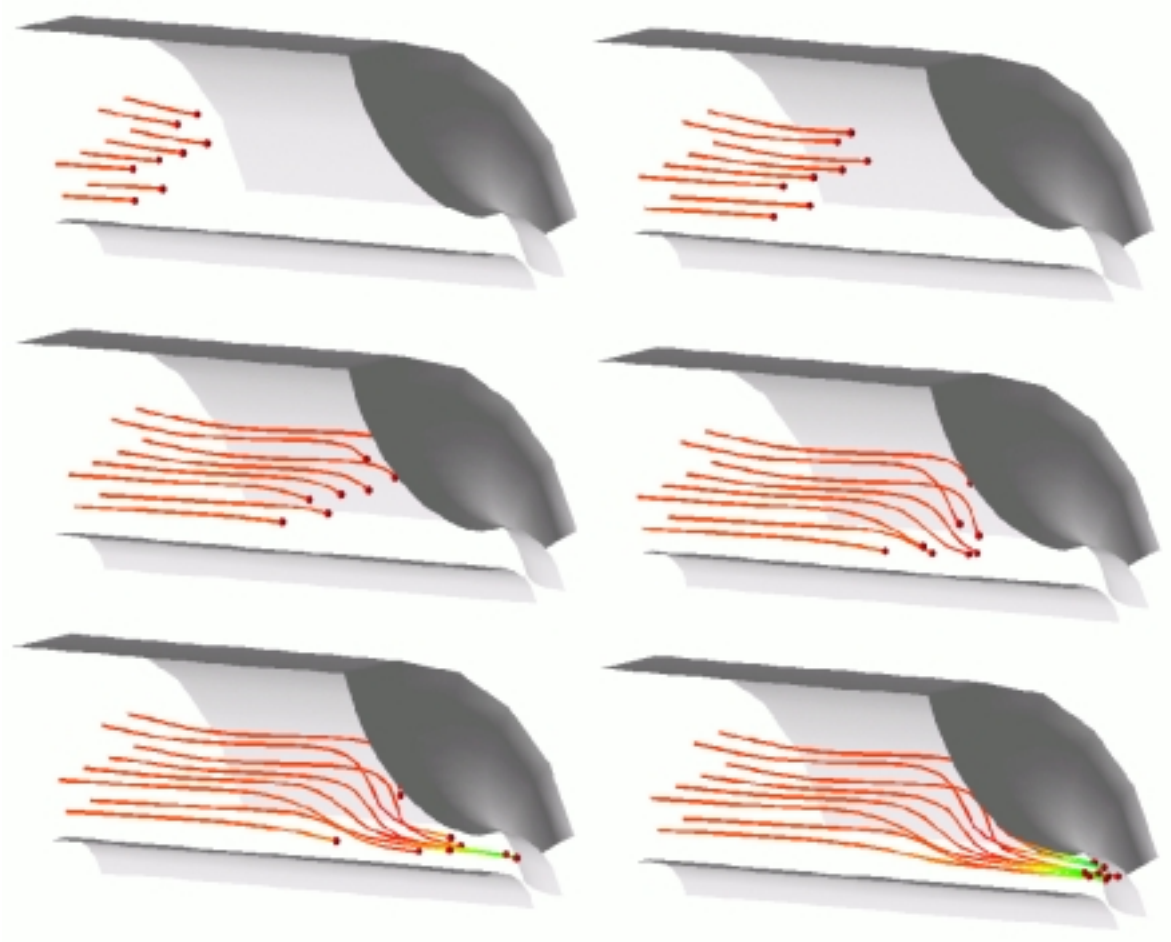


Abb. 4.39: Serie von Momentaufnahmen einer Partikelverfolgung am Beispiel einer kaltdurchströmten Düse

4.3.10 Tropfenverfolgung

Bei der Tropfenverfolgung wird das Schicksal einer Punktemenge untersucht, deren Elemente zur Startzeit alle auf ein und derselben Kugeloberfläche liegen, zugleich aber die Knotenpunkte eines Gitters bilden, das die Kugeloberfläche in Form von Dreiecken approximiert.
Der Begriff Tropfen ist hier nicht im physikalischen Sinne zu verstehen. Gemeint ist lediglich die äußere Form des Gebildes, die einem Tropfen ähnlich sieht. Es handelt sich hierbei also um ein rein visuelles Instrument, mit dem sich eine Partikelschar in optisch angenehmer Weise verfolgen läßt, so daß sich Vorgänge in der Strömung leichter interpretieren lassen, beispielsweise dadurch, daß lokale Geschwindigkeitsänderungen sich in einer Verformung des Tropfens plastisch erfahrbar machen.

Vorgehensweise
Die Vorgehensweise bei der Tropfenverfolgung ähnelt der der Partikelverfolgung. Am Anfang steht die Generierung eines Tropfengitters, die mit dem Modul *droplet generator* in Abhängigkeit der Parameter Feinheitsgrad und Radius an einem ebenfalls interaktiv vorzugebenden Startort erfolgt. Daran schließt sich die Berechnung der Stromlinien für jeden Kugelpunkt, die Synchronisierung der Daten und zuletzt die eigentliche Darstellung an.

Generierung eines kugelförmigen Tropfens
Die Generierung erfolgt durch eine Rasterung von Würfelseitenflächen, abhängig vom gewählten Feinheitsgrad. Es wird ein kartesisches Punktenetz zugrundegelegt, in dem jeder Punkt unitär ist. Die Netzpunkte werden auf eine Kugel mit Radius r und Zentrum $\vec{m}$ im Mittelpunkt des Würfels abgebildet, indem jeder Gitterpunkt denselben Abstand vom Mittelpunkt erhält. Die Kugeloberfläche wird trianguliert (Abbildung 4.40). Anstelle der Kugel- kann jedoch auch die Würfeloberfläche als Ausgangsgebiet für eine Verfolgung dienen.

Berechnung
Jeder einzelne Punkt auf der Kugeloberfläche wird als Startpunkt für eine Stromlinie genommen, deren Berechnung mit dem Modul *flow server* erfolgt. In Anbetracht der großen Zahl der Startpunkte, die es bedarf, um einen Tropfen zu idealisieren, ist eine zeitweilige Auslagerung der Daten für die Berechnung auf einen Parallelrechner angezeigt.

Synchronisierung
Die Synchronisierung der Stromliniendaten auf n Zeitpunkte erfolgt wie bei der Partikelverfolgung.

Tropfendarstellung
Eine Darstellung des Tropfens zum Zeitpunkt $i * \Delta_t$ wird durch Zeichnen der Dreiecksflächen in der Anordnung des Ausgangstropfens erhalten, jedoch mit den zeitlich veränderten Koordinaten.

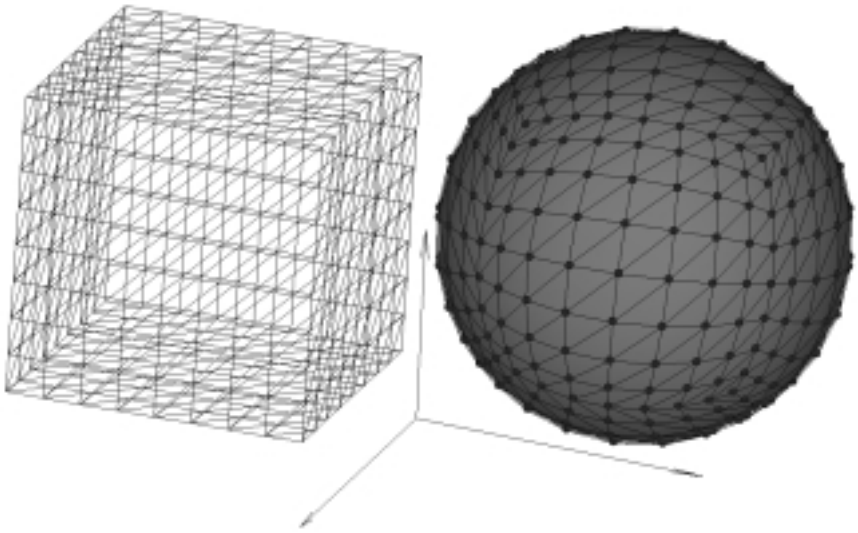


Abb. 4.40: Radiale Projektion einer Würfeloberfläche auf eine Kugeloberfläche (Feinheitsgrad 8)

Darstellungsarten
Zum einen kann dem Tropfen ein beliebiger konstanter Farbwert verliehen werden, zum anderen läßt sich eine physikalische Größe auf der Tropfenoberfläche abbilden, mit variierender Farbschattierung. Auf jeder der berechneten Stromlinien sind die Werte der physikalischen Größe zu jedem Zeitpunkt bekannt und somit der Aufenthaltsort der Partikel zu einem gegebenen Zeitpunkt. Da die Partikel gleichzeitig auch Knotenpunkte des Tropfengitters sind, läßt sich der Tropfen jederzeit darstellen, in dem die Dreiecke nach der ursprünglichen Vorschrift gezeichnet werden zusammen mit Farbwerten, die der physikliaschen Größe an den Knoten bzw. in den Partikeln entsprechen.

Animation
In einem automatisierten Prozeß kann sich der Anwender für jeden Zeitschritt eine Momentaufnahme des Tropfens als Bild auf Platte speichern lassen. Die Summe der Bilder wird in einem Zyklus zusammengefaßt und kann unter AVS animiert werden, so daß die Bewegung der Tropfen durch das Strömungsfeld mit Veränderung der Tropfenoberfläche und der Farbe simuliert werden kann (siehe Abbildung 4.41).

Einschränkungen in der Anwendung
Die Tropfenverfolgung läßt sich sinnvoll anwenden auf lochfreie Strömungsgebiete (Beispiel Strömungverzweigung um ein Hindernis), rotationsfreie Gebiete, sowie Gebiete, in denen im Verlauf die Stromlinien nicht zu weit auseinanderlaufen. Die Punkteschar sollte nicht divergieren, der Tropfenradius nicht zu groß gewählt werden.

4.3.10.1 Ergebnisbeispiel für eine Tropfenverfolgung

Abbildung 4.40 zeigt Momentaufnahmen dreier Tropfen für das Beispiel der axialsymmetrischen Düse mit einer isothermen laminaren Strömung. Für die Ausgangstropfen wurden jeweils der Feinheitsgrad 10 eingestellt, der eine Kugel mit 602 Gitterpunkten bzw. 1200 Dreiecken idealisiert. Ergebnisse für die drei Ausgangsradien ($r_2 = 2 * r_1 = 5 * r_0$) sind illustriert. Zusätzlich wurden Geschwindigkeitswerte berechnet und auf die Oberfläche projiziert.

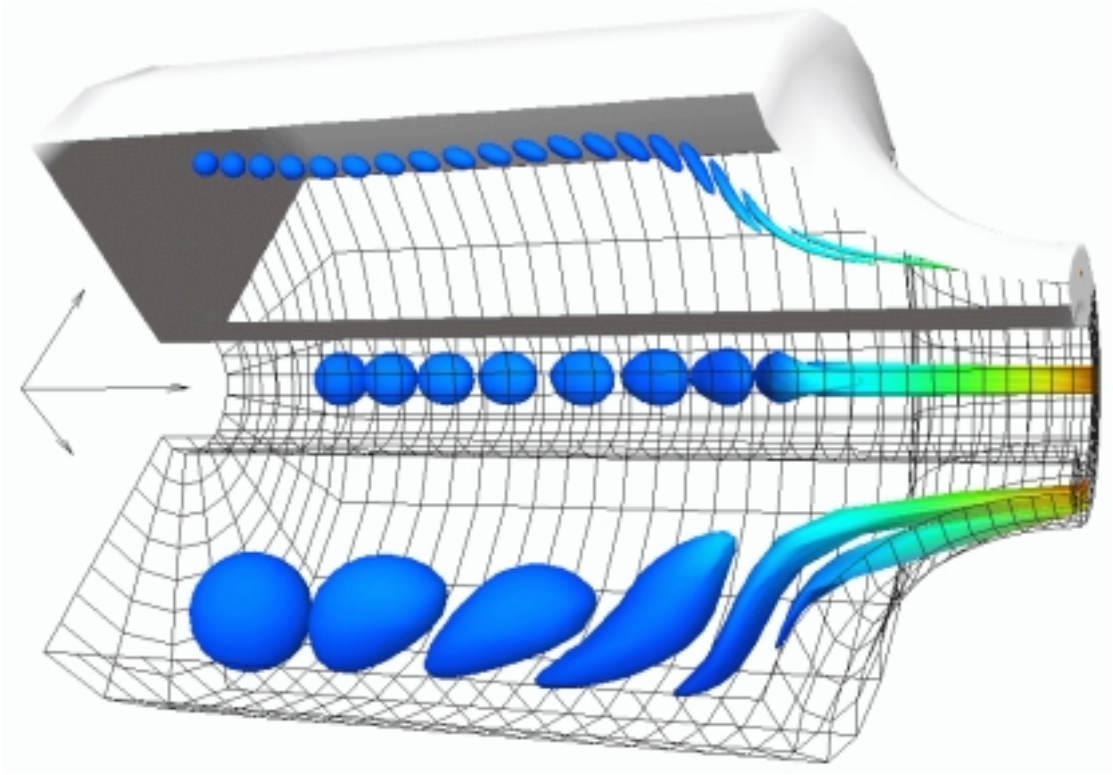


Abb. 4.41: Wanderung dreier Tropfen verschiedener Radien in das Ausströmgebiet einer kaltdurchströmten Düse, dargestellt mit einer Geschwindigkeitswerteverteilung.

4.3.11 Verfolgung ebener Flächensegmente

Bei der Verfolgung ebener Flächensegmente tritt an die Stelle des Tropfengenerators ein Flächengenerator, der verschiedene Kreisflächensegmente im Raum generieren kann (Abb. 4.42). Der Anwender gibt interaktiv die Flächennormale $\vec{n}$ vor, den Kreismittelpunkt $\vec{m}$, die beiden das Segment begrenzende Radien r_i und r_a sowie den maximalen Segmentwinkel φ_{max}. Die Feinheit des generierten Gitters innerhalb des Segments wird durch ein Inkrement in radialer Richtung dr sowie ein Winkelinkrement φ bestimmt. Als Gitterelemente werden Vierpunktselemente eingesetzt.

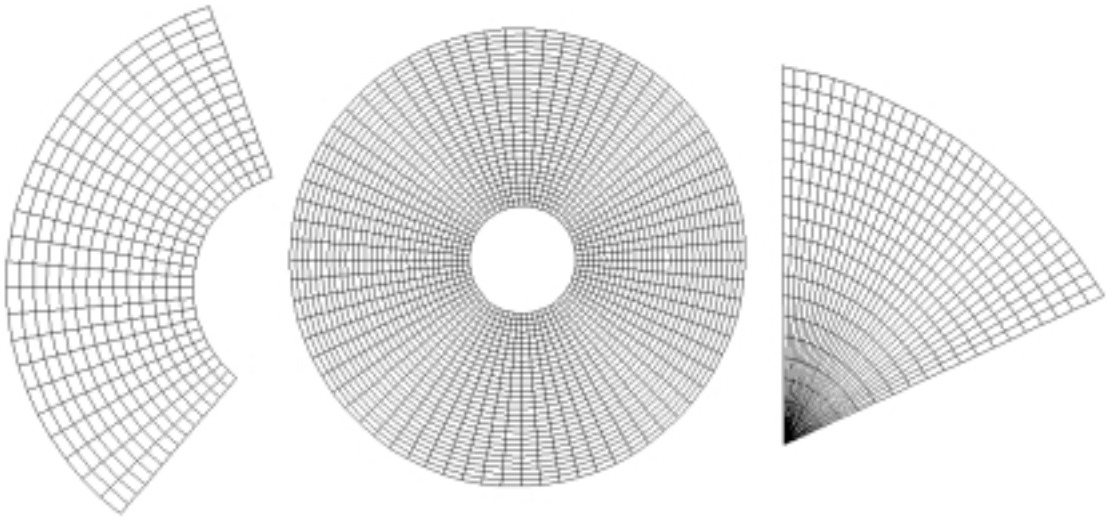


Abb. 4.42: Zur Generierung verschiedener Flächensegmente.

Im Anschluß an die Stromlinienberechnung, in der wiederum jeder Knotenpunkt des Ausgangsgitters als Startpunkt einer Stromlinie angesehen wird, bildet das Modul *surface tracer* zu jedem der diskreten Zeitpunkte der Animationsfolge die Polygone der Gitterelemente, gemäß der ursprünglichen Beschreibung des Gitters.

4.3.11.1 Ergebnisbeispiel für die Verfolgung eines Flächensegments

Abbildung 4.43 zeigt drei Momentaufnahmen einer stromabwärts driftenden ebenen Ausgangsfläche für das Beipiel der axialsymmetrischen Düse. Links im Bild ist das Ausgangsgitter (900 Knoten, 864 Elemente) für die Fläche dargestellt. Auf die Oberflächen wurden Dichtewerte projiziiert. Diese Darstellung verdeutlicht die Zähigkeit der Strömung, die eine Verminderung der Geschwindigkeiten in der Nähe der Innen- und Außenwand bewirkt. Eine andere Darstellung wie etwa ein Längsschnitt einschließlich Vektorkomponenten in Abb. 4.44 vermag die Grenzschicht nicht so plastisch herauszuarbeiten, da damit nur ein Momentanausschnitt gezeigt werden

kann. Bei der Ebenenverfolgung hingegen dokumentiert jede Momentaufnahme auch den zeitlichen Verlauf der Strömung bezüglich des Ortes, an dem die ebene Fläche losgelassen wurde. Die Verfahren der Teilchenverfolgung sind sehr gut für parallele Verarbeitung geeignet. Dadurch wird das Postprozessing erheblich beschleunigt. Die Methodik und der Einsatz der parallelen Algorithmen wird im nächsten Kapitel beschrieben.

Abb. 4.43: Auszüge einer Verfolgung eines Flächensegments am Beispiel der Düse

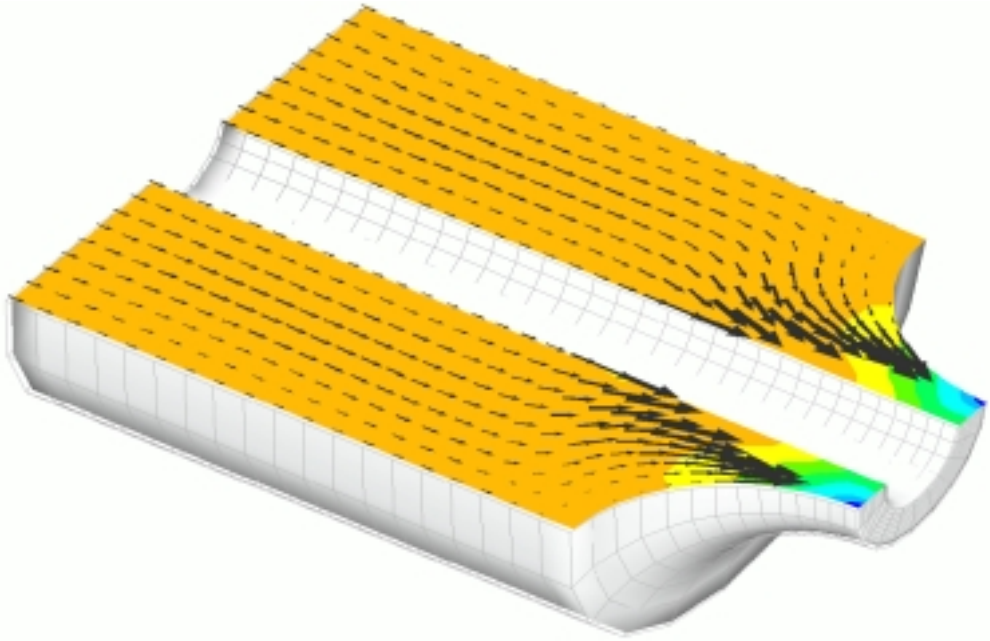


Abb. 4.44: Vektoren in einem Längsschnitt durch die Düse zum Vergleich

Kapitel 5

Parallele Visualisierung

Dieses Kapitel widmet sich der Parallelisierung der im vorangegangenen Kapitel vorgestellten Algorithmen, die weitestgehend durch Einbinden serieller Algorithmen in eine parallele Befehlsstruktur sowie in eine parallele Programmverwaltung zustande kommt. Letzterer kommt dabei neben der Steuerung des gesamten parallelen Ablaufes und der selbständigen Abarbeitung des Auftrages auch die Aufgabe der Bereitstellung bzw. der Übernahme der Datenbasis zu. Ferner ist die Speicherverwaltung und schließlich die Ausgabe der Ergebnisse aus der parallelen Gittertransformation Aufgabe eines entsprechenden Programmsystems.
In diesem Abschnitt wird zunächst auf die Struktur des Programms bzw. auf den Programmaufbau eingegangen bevor einzelne Unterpunkte genauer zur Sprache kommen, wie z.B. Initialisierung und Aufbau einer Topologie, Verwendung von Bibliotheken für interne und externe Kommunikation, Bereitstellung der Datenbasis und des Auftrages, parallele Erzeugung der Visualisierungsgitterdaten oder deren Ausgabe.

5.1 Der parallele Programmaufbau

Grundsätzlich richtet sich der parallele Programmablauf nach der Art und Weise, wie die Visualisierung betrieben werden soll. Im *standalone* Modus arbeitet die parallele Visualisierung unabhängig von einem FE–Analyseprogramm und dient hauptsächlich zur Unterstützung und Beschleunigung des Postprozessings. Im gekoppelten Modus hingegen arbeitet die parallele Visualisierung integriert in einen FE–Analyseprozeß und unterscheidet sich vom standalone Modus zum einen durch

eine von der Analyse abhängige Initialisierung und zum anderen durch Datenbasis und Speicherverwaltung. Dennoch sind bei beiden Schemen ganze Funktionsblöcke identisch (vgl. Abb. 5.1 mit Abb. 5.2).

5.1.1 Programmablauf im Standalone–Modus

Der Programmablauf im Standalone–Modus, wie er in Abbildung 5.1 dargestellt ist, beginnt mit der Initialisierung und dem Aufruf der Funktion *init_pgra0()*. In der Initialisierungsfunktion wird unter anderem rechnerabhängig das Kommunikationsnetz erstellt. Ferner werden die entsprechenden Links aufgebaut und die voreingestellten Parameter eingelesen, nach denen sich der weitere parallele Programmablauf richtet. In der Funktion *get_instruc()* erhält der Masterprozessor den Auftrag und die Pfadnamen für die Datenein- und -ausgabe und gibt diese Information den übrigen Prozessoren bekannt (*broadcast*). Abhängig von den beiden grundlegenden Algorithmustypen, die sich vor allem durch die Ausgabeformate unterscheiden, die sie erzeugen, werden die Funktionen *pre_cvp()* bzw. *pre_str()* aufgerufen. Hier wird die Datenbasis erstellt und genügend Speicherplatz für die Datengrundlage reserviert, die entweder direkt über das lokale Netz empfangen oder von der Festplatte eingelesen wird. Beim Einlesen wird erwartet, daß die Daten verteilt vorliegen, das heißt, daß zu jedem Prozessor eine individuelle Datei gehört. Überdies kann der Masterprozessor, sofern er über ausreichende Speicherkapazität verfügt, das Gesamtproblem einlesen und die Datenaufteilung selbst vornehmen.

Die eigentliche Berechnung und die Ausführung der parallelen Algorithmen erfolgt in den Funktionen *run_cvp()* bzw. *run_str()*. Vor der Berechnung werden wichtige Parameter und globale Größen wie die Extremwerte einer physikalischen Größe ermittelt und kommuniziert oder beispielsweise Startpunkte von Stromlinien auf die Prozessoren umverteilt.
Nach Berechnung des für die Visualisierung repräsentativen Gitters müssen die erhaltenen Gitterdaten zur Ausgabe aufbereitet werden, was in den Funktionen *post_cvp()* und *post_str()* vorgenommen wird. Dabei können die Gitterdaten direkt über das lokale Netz an den Geometrie–Mapper und Renderer transferiert werden, wenn sie nicht zu einer späteren Weiterverarbeitung entweder in einer einzigen globalen Datei oder prozessorweise in individuellen Dateien abgelegt werden sollen.
Mit dem Abschluß der Übertragung bzw. dem Herausschreiben der Daten endet das Programm.

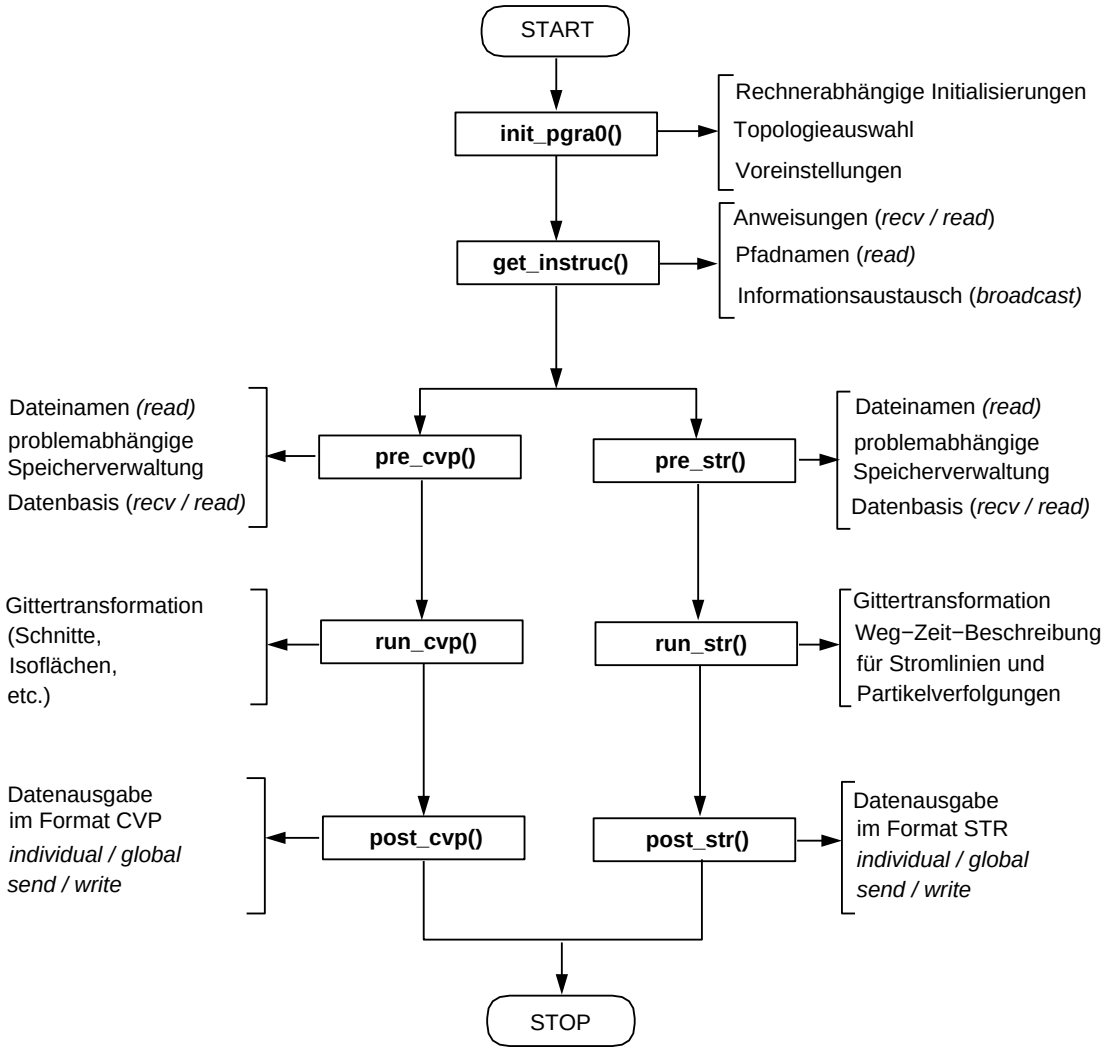


Abb. 5.1: Ablauf einer parallelen Gittertransformation im Standalone Modus

5.1.2 Programmablauf im gekoppelten Modus

Beim Programmablauf im gekoppelten Modus entfällt eine Initialisierung des Prozessornetzwerkes, da das parallele Visualisierungsprogramm abhängig vom Wert einer Visualisierungsmarke aus einer bereits initialisierten parallelen FE-Berechnung (FEPS [84]) heraus aufgerufen wird. Wie aus Abbildung 5.2 entnommen werden kann, geschieht der Aufruf in der Funktion *parsc()* innerhalb der Zeitinkrementierung des Gleichungssystemlösers des Analyseprogrammsystems.
In der Funktion *init_pgra1()* wird zunächst eine Teilauswertung der zum Zeitpunkt vorliegenden numerischen FE-Ergebnisse vorgenommen und der Ergebnisvektor in Abhängigkeit der eingelesenen Visualisierungsparameter gebildet. Der Aufbau der

Datenbasis erfolgt durch Übergabe der Zeiger auf die für die Visualisierung relevanten Analyseergebnisdaten. Ein eindimensionales Datenfeld, das die Elementtypen im PATRAN-Modus benennt, wird jedoch zusätzlich angelegt.

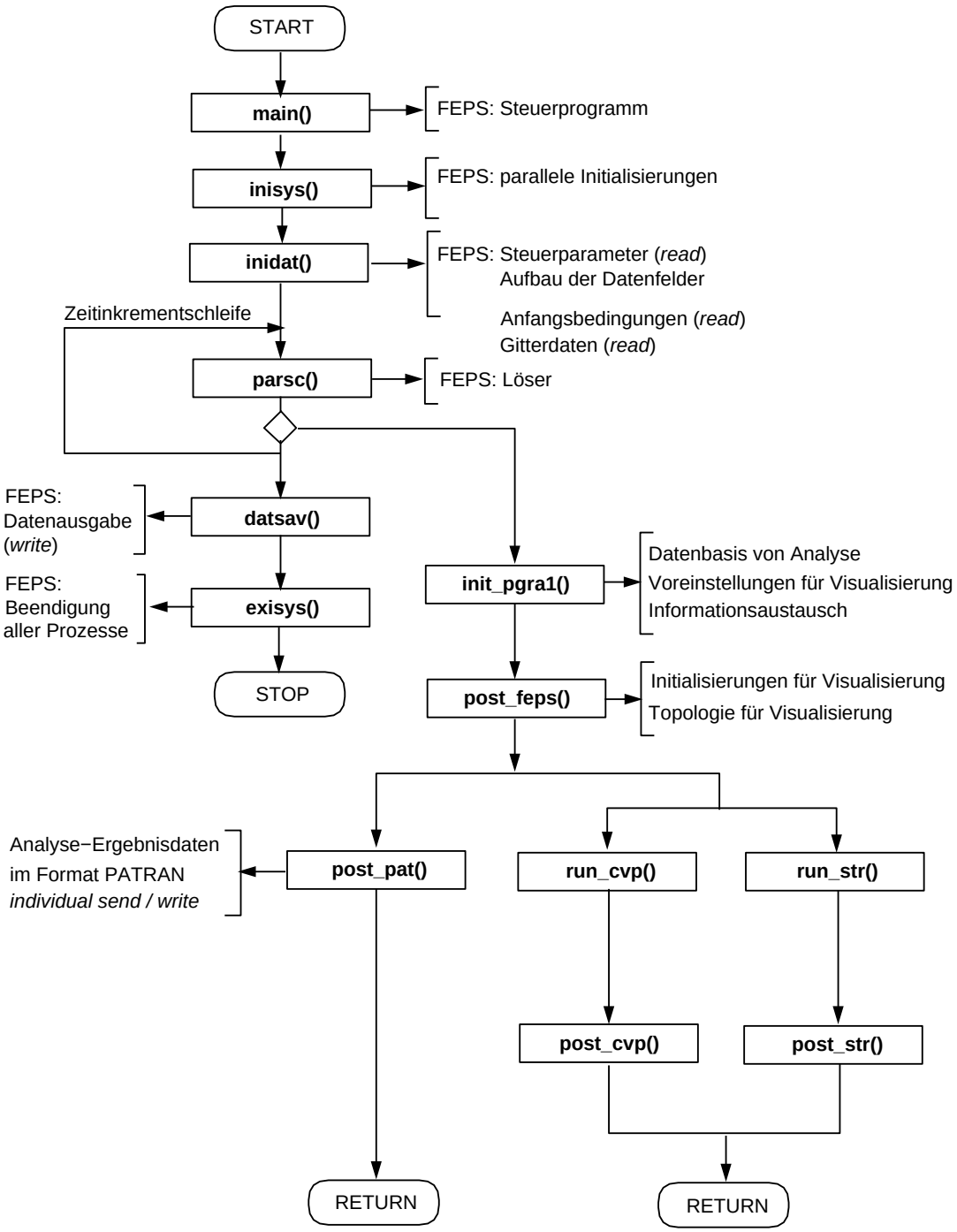


Abb. 5.2: Ablauf einer parallelen Gittertransformation im integrierten Modus

In der Funktion *post_feps()* werden Initialisierungen die Visualisierung betreffend vorgenommen. So wird zu der bestehenden Topologie temporär eine weitere für die Belange der Visualisierung aufgebaut. Je nach Auftrag werden die Funktionen *post_pat()*, *run_cvp()* oder *run_str()* gerufen. Während für die beiden letztgenannten Funktionen das weitere Programm wie im Standalone–Modus abläuft, ist *post_pat()* für die Ausgabe der Analyseergebnisdaten vorgesehen. Diese Ergebnisse werden entweder in Dateien abgelegt (global oder individuell) oder zur seriellen graphischen Auswertung direkt über das lokale Netzwerk an eine Graphik-Workstation übertragen.
In allen drei Fällen endet die Visualisierungsprozedur mit dem Abbau der temporären Topologie und der Wiederaufnahme der FE-Berechnung.

5.1.3 Initialisierung

Nachdem das Programm auf einer Anzahl von Parallelprozessoren gestartet wird, müssen gewisse Voraussetzungen erfüllt sein, damit die einzelnen Prozessoren ihre Aufgaben überhaupt wahrnehmen und relevante Daten miteinander austauschen können. Zum einen muß jeder Prozessor sich selbst identifizieren können, zum anderen muß er die Gesamtzahl der beteiligten Prozessoren kennen wie auch seine direkten Nachbarn, von denen er Daten erhält bzw. an welche er Daten weiterschickt.
In einem Prozessornetzwerk gibt es physikalische Nachbarn, bedingt durch die Hardware (Verdrahtung), und virtuelle Nachbarn, die sich durch die jeweils erstellte Topologie ergeben. In den seltensten Fällen sind physikalische und virtuelle Nachbarn identisch, d.h. die Kommunikationswege werden unterschiedlich lang ausfallen. Im Prinzip lassen sich die Kommunikationswege über die Wahl einer geeigneten Topologie optimieren, die mit der Anordnung der Prozessoren korreliert, immer vorausgesetzt, der physikalische Ort eines jeden der zugewiesenen Prozessoren ist bekannt. Ein aufwendiges Prozessoren-Mapping wurde hier jedoch unterlassen.
Für die oben genannten Initialisierungen existieren maschinenabhängige Befehle und Kommandos, die von denjenigen Programmiersprachen (i.a. FORTRAN oder C) gerufen werden können, die das parallele Betriebssystem (z.B. PARIX) unterstützen.
Das Visualisierungsprogrammsystem kommuniziert entweder mittels einer voreingestellten Ringtopologie (*ringtop*) oder mittels einer Baumtopologie (*treetop*), bei der nicht wie sonst üblich jede Stufe voll besetzt sein muß. Für beide Topologien gibt es keine Einschränkungen bezüglich der Anzahl der Prozessoren. Im gekoppelten Betrieb mit einer parallelen FE–Analyse wird im Vorfeld der Visualisierungsinitialisierung zunächst die Topologie der FE–Analyse weiter verwendet, während der

eigentlichen Gittertransformation jedoch zusätzlich eine visualisierungseigene Topologie aufgebaut.

Das parallele Programm arbeitet mit einem Masterprozessor, der besondere Aufgaben wie z.B. das Einlesen oder direkte Empfangen von Anweisungen wahrnimmt. Ferner obliegt es dem Master, den übrigen Prozessoren die wichtigsten und für den Ablauf relevanten Parameter mitzuteilen (*broadcast*). Beim Einlesen mehrerer globaler Dateien wird der Master- von einem Coprozessor unterstützt.

5.1.4 Interne und externe Kommunikation

Um überhaupt Information zwischen Prozessoren aber auch mit der Außenwelt austauschen zu können, bedarf es einer Kommunikationssoftware. Dazu wurden Bibliotheken sowohl für die interne als auch für die externe Kommunikation eingerichtet.

Interne Kommunikation
Die Bibliothek für interne Kommunikation stellt die wichtigsten Austauschmechanismen für beide verwendete Topologien *ringtop* und *treetop* zur Verfügung. Hier die Grundfunktionen für die Kommunikation zwischen den Prozessoren:

- **broadcast:** Bekanntmachung bzw. Verbreitung einer Information. Ausgehend vom Master werden alle übrigen Prozessoren informiert.
- **sum_up:** Akkumulieren eines Skalars über alle Prozessoren und Bekanntmachen des Ergebnisses.
- **max_min:** Ermittlung der globalen Extremwerte eines Skalars aus den lokalen Extremwerten eines jeden Prozessors, mit anschließender Verbreitung des Ergebnisses.
- **send_once:** Unidirektionales Senden von einem Prozessor zu einem beliebigen anderen.
- **read_once:** Unidirektionales Empfangen von einem anderen Prozessor.

Die Grundfunktionen sind so gestaltet, daß die Initiative im allgemeinen vom Masterprozessor ausgeht. Ausnahmen bilden lediglich *send_once* und *read_once*, die zum einmaligen unidirektionalen Austausch zwischen zwei beliebigen virtuell miteinander verbundenen Prozessoren bestimmt sind.
Ferner wurden diese Grundfunktionen in übergeordnete oder erweiterte Funktionen eingebunden, die häufig verwendet werden. Solche Funktionen können eine oder mehrere Grundfunktionen enthalten und schicken in der Regel genormte Vorabinformationen über die zu versendenden Felder und Variablen, etwa deren Länge in

Bytes und deren Datentyp (*long, float, char*). Das ist unter anderem deshalb wichtig, damit der Empfängerprozessor vor dem Erhalt der Information ausreichend Speicherplatz reservieren kann.
Die Kommunikation erfolgt vorzugsweise synchron. Der Vorteil asynchroner Kommunikation, - nämlich das Beschreiben eines Puffers, den der Zielprozessor dann liest wenn er soweit ist, während der Sendeprozessor in der Zwischenzeit schon weiterarbeiten kann, - kommt durch die Programmstruktur nicht zum Tragen und ist zudem mit einer Unsicherheit behaftet. Es ist nämlich nicht gesichert, ob der Empfänger die Information auch tatsächlich erhält oder ob er mit falschen Daten weiterrechnet, falls ein anderer Prozessor mittlerweile den Puffer überschrieben hat. In jedem Fall gestaltet sich eine Fehlererkennung und -behandlung bei asynchroner Kommunikation schwierig.

Externe Kommunikation
Die Kommunikation mit der Außenwelt erfolgt auf Basis sogenannter Sockets, den Kommunikationsendpunkten der Interprozeßkommunikation [54], von denen in Kapitel 6 ausführlich die Rede sein wird. Wie bei der internen Kommunikation wurden Grundfunktionen für das Senden und Empfangen von Datenfeldern programmiert. Bisher werden Transputer und Intel-Prozessoren unterstützt, und das mit den Funktionen *px_send* und *px_recv* bzw. *pg_send* und *pg_recv*. Die Byte–Anordnung beider Architekturen ist *little endian*, weswegen beim Empfang gleich wie vor dem Schicken von Daten eine Byte–Umkehrung zu erfolgen hat. Dies geschieht mittels der Funktionen *big2lit* und *lit2big*. Nachstehend sind im Überblick die Grundfunktionen für einen Parsytec-Parallelrechner aufgelistet:

- **px_send:** Senden eines oder mehrerer Datenfelder von einem beliebigen Prozessor aus über einen AF_UNIX-Socket [54] an die dem Parallelrechner vorgelagerte Hostmaschine.
- **px_recv:** Empfangen eines oder mehrerer Datenfelder von dito.
- **lit2big:** Daten-Konvertierung von *little endian* nach *big endian*.
- **big2lit:** Daten-Konvertierung von *big endian* nach *little endian*.

Für einen Intel-Paragonrechner lauten die äquivalenten Funktionen:

- **pg_send:** Senden eines oder mehrerer Datenfeldern von einem beliebigen Prozessor aus über einen AF_INET-Socket [54] an die Welt.
- **pg_recv:** Empfangen eines oder mehrerer Datenfelder von dito.
- **lit2big:** Daten-Konvertierung von *little endian* nach *big endian*.
- **big2lit:** Daten-Konvertierung von *big endian* nach *little endian*.

Zusätzlich sei erwähnt, daß im Gegensatz zu einer Intel-Paragon, bei der jeder Prozessor mit der Welt kommunizieren kann, eine Parsytec einen vorgeschalteten Hostrechner benötigt, auf dem auch kompiliert und initialisiert wird.
Die erweiterten Funktionen *px_export* und *px_import* bzw. *pg_export* und *pg_import* wickeln den prozessorweisen Empfang bzw. das prozessorweise Senden von bis zu 6 Datenfeldern nach außen ab.
Maximal 6 Datenfelder sind jeweils erforderlich, um sowohl die Eingangsdaten als auch die Ausgabedaten für die Gittertransformation zu beschreiben (vgl. Kapitel 4.1). Nicht verwendete Felder werden mit Dummy-Variablen belegt. Der Sendeprozeß wird vom Master eingeleitet, der vor den eigentlichen Daten das Kenndatenfeld *iwhat()* vorausschickt, welches globale Angaben über Anzahl der zu versendenden Felder, deren Byte–Zahlen und die Kennung des empfangenden Moduls auf der AVS-Seite enthält.

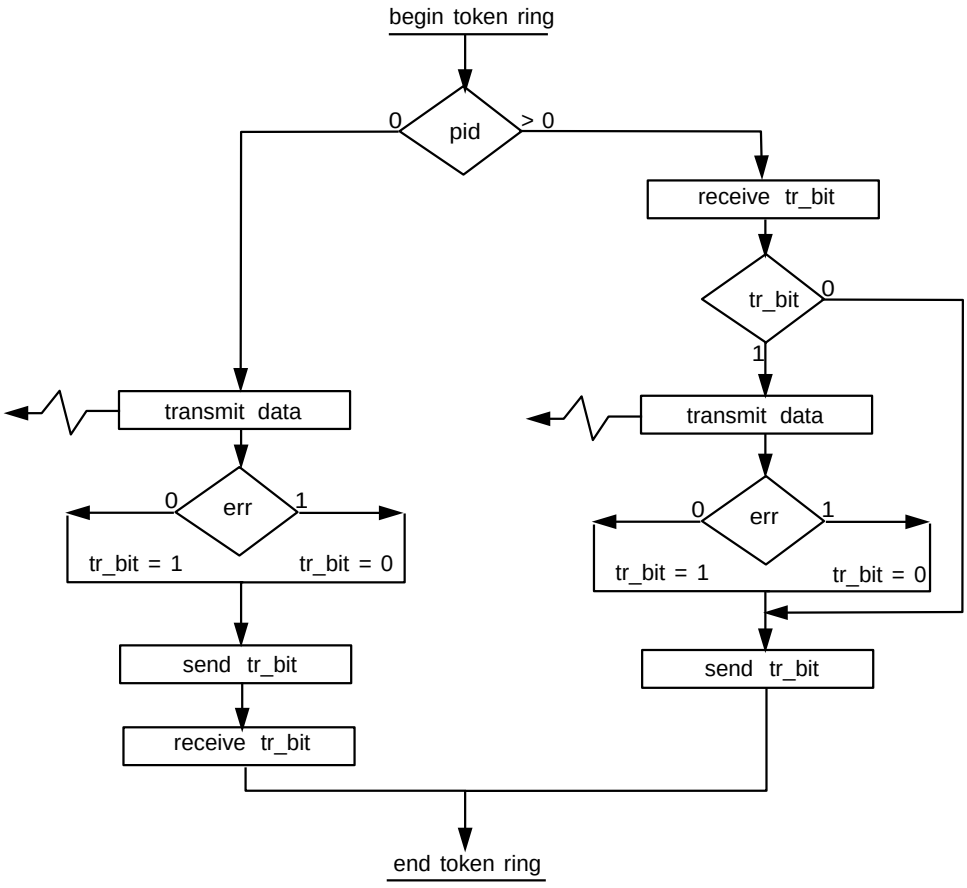


Abb. 5.3: Zum Datentransfer mit Token Ring

Das prozessorweise Vorgehen funktioniert nach Token Ring. Der Token ist hier ein Transferbit, das der aktuelle Prozessor von seinem Vorgänger zusammen mit einem Errorbit erhält. Beide Bits werden nach dem Senden an den Folgeprozessor weiter-

gereicht, bis der Master erreicht ist, von dem die Initiative ausging (Abb. 5.3).
Das Errorbit zeigt den Erfolg bisheriger Sendeprozesse an. Der interne Kommunikationszyklus muß in jedem Fall zu Ende geführt werden, um keinen Abbruch der gesamten Parallelanwendung auszulösen.
Tritt bei der Übertragung der Daten unerwartet ein Fehler auf, schaltet das Errorbit von 0 auf 1, was alle Folgeprozessoren veranlaßt, Transferbit und Errorbit weiterzureichen, ohne jedoch versucht zu haben, ihre lokalen Daten extern abzuliefern. Trotz eines Misserfolgs der Kommunikation nach draußen kann das Parallelprogramm also mit seiner Aufgabe fortfahren, notfalls mit einer standardisierten Voreinstellung.
Für die Baumtopologie gestaltet sich die oben beschriebene Token–Ring–Technik schwierig, da infolge der Verzweigungen auf jeder neuen Stufe Zugriffskonflikte auftreten, da sich auch die Zahl der Tokens jedesmal verdoppelt.

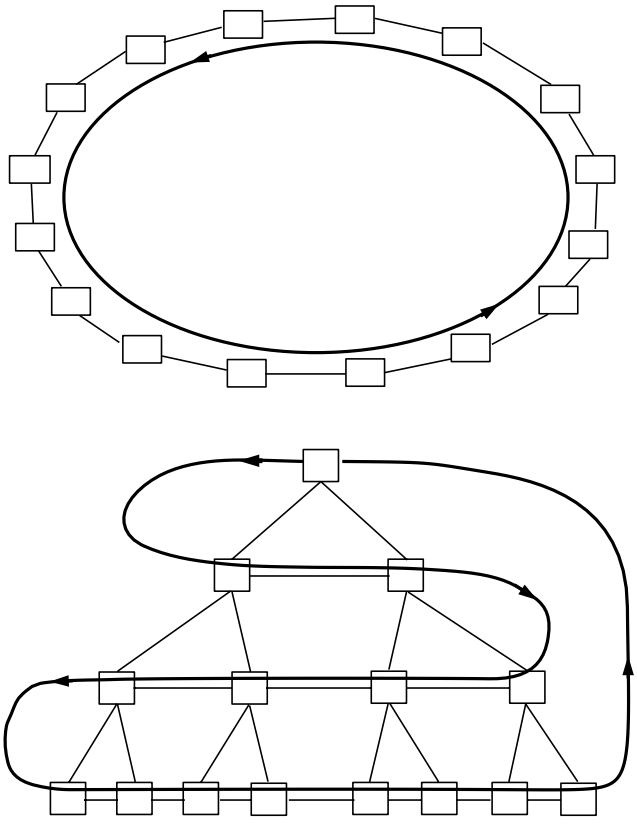


Abb. 5.4: Token Ring auf einer Ring- und einer X-Baum-Topologie

Zeitmessungen haben zudem ergeben, daß sich für Belange der Visualisierung die Verwendung einer Baumtopologie gegenüber der Ringtopologie erst ab einer Prozessorzahl größer 32 merklich positiv auswirkt. Beim Problem des Zugriffs bietet sich als Alternative zum Baum der X-Baum an, eine Topologie, die bisher in diesem Programmsystem nur als Konzept existiert. Der X-Baum kombiniert die Vorteile von

Ring und Baum, weil er beide Kommunikationsstrukturen verwenden kann. Der Vorgang des Token Ring ist in Abbildung 5.4 für eine Ring- und eine X-Baum-Topologie dargelegt.

5.1.5 Auftragsabarbeitung und Steuerung

Ein Auftrag kann zum einen direkt durch einen anderen Rechner über externe Anweisungen erteilt werden und zum anderen durch einzulesende Steuerdateien. Es ist der Master, der empfängt oder einliest. Überdies kann eine parallele Anwendung auch unabhängig nach voreingestellten Werten (Default) betrieben werden, wie das übrigens auch im Falle eines Versagens einer externen Steuerung geschieht, meist verursacht durch Instabilitäten im lokalen Netzwerk.
Der Master gibt den übrigen Prozessoren alle ablaufbestimmenden Parameter bekannt. Jeder Prozessor führt danach seine spezifischen Aufgaben auf seinen spezifischen Daten aus und weiß so, an welchen Stellen im Programm er auf Daten zu warten, und wann er Daten zu verschicken hat.
Ein Auftrag besteht aus drei eindimensionalen Datenfeldern zu je 25x8 Byte. Eines verwaltet die wichtigsten Steuerparameter (*long*) für Datenein- und -ausgabe, während die anderen beiden die Visualisierungsparameter (*long* und *float*) enthalten, die die Algorithmen steuern. Darunter fallen Angaben über die Art des Algorithmus, Anzahl von z.B. Isolinien, die Wahl einer skalaren oder vektoriellen physikalischen Größe, Wertebereichseinschränkungen, Skalierungsfaktoren und einiges mehr. Angaben dieser Art existieren voreingestellt, können aber über Einlesedateien modifiziert werden, oder bequemer extern und interaktiv über Steuermoduln unter AVS übermittelt werden.

5.1.6 Bereitstellung der Datenbasis

Nach der Initialisierung und der Bekanntmachung aller Steuerparameter durch den Masterprozessor wird die Datenbasis für die Gittertransformation aufgebaut. Im Zuge dessen werden zunächst an Hand eingelesener oder direkt empfangener Kenngrößen die Datenfelder allokiert und danach die Daten eingelesen oder empfangen. Aus den Kenngrößen wie die Anzahl der Knoten und der Elemente, der Länge des Ergebnisvektors und des anzuwendenden Algorithmus, wird außerdem der Datenaufwand für die Ergebnisse der Gittertransformation abgeschätzt und die entsprechenden Felder bereitgestellt.
Vor dem Einlesen der Daten liest der Master einen speziellen Datensatz *.parrc*, in

dem die absoluten Pfadnamen sowohl für die Dateneingabe, als auch für die Datenausgabe sowie eventuelle Steuerparameter abgelegt sind.
Das Einlesen der Daten selbst kann entweder verteilt erfolgen, oder ein oder zwei Prozessoren lesen seriell vorliegende globale Dateien ein. Im letzteren Fall liest der Master die Gitterdaten und sein Coprozessor gleichzeitig die Ergebnisvektoren. Das Neutralformat von PATRAN wird hierbei ebenso unterstützt wie das Eingabeformat des Programmsystems FEPS. Die Namen der zu lesenden Dateien erhält der Master über eine Hauptdatei, die in der ersten Zeile das Format von PATRAN oder FEPS enthält und in den folgenden Zeilen die Dateinamen für die Gitter- und Ergebnisdaten.
Es ist die Aufgabe beider Prozessoren, die Daten möglichst lastausgleichend auf die restlichen Prozessoren umzuverteilen, wobei Erfahrungen gezeigt haben, daß für die meisten der hier verwendeten Algorithmen eine annähernd gleiche Zahl von Elementen pro Prozessor ausreichend ist. Allerdings ist es auch naheliegend, daß bei großen Problemen Master und Coprozessor nicht mehr in der Lage sind, das Gesamtproblem in den Speicher zu laden. Dies gilt für die Ausgabe der Ergebnisse aus der Gittertransformation, sofern der Master alle Daten einsammelt und als eine globale Gesamtdatei auf das Speichermedium schreibt.
Im Falle verteilter Datensätze werden die Namen der Dateien allen Prozessoren bekannt gegeben (*broadcast*). Ein dreistelliges, den Prozessor kennzeichnendes Suffix, z.B. *007* für den achten Prozessor, wird dem ersten Namensglied angehängt und die Existenz einer solchen Datei vorausgesetzt. Zu dem eingelesenen Namen *benard.ndat* erwartet der achte Prozessor demnach eine Datei namens *benard007.ndat*.

5.1.7 Datenausgabe

Die Ausgabe der Visualisierungsgitterdaten kann ebenfalls wahlweise seriell durch Abspeichern einer Gesamtergebnisdatei erfolgen, die der Master durch Aufsammeln der Ergebnisse der übrigen Prozessoren erstellt, oder parallel in Form von Einzeldateien, die jeder Prozessor selbständig ausgibt. Ferner kann mit den Funktionen externer Kommunikation *px_export* bzw. *pg_export* die Direktübertragung auf einen Zielrechner, beispielsweise eine Graphik-Workstation, vorgenommen werden.
Bei der individuellen Datenausgabe richtet sich der Name nach dem verwendeten Algorithmus und dem Prozessor. Kam beispielsweise ein Isoflächenalgorithmus zur Anwendung, heißt die individuelle Ausgabedatei des achten Prozessors *ISURF007.dat*. Zusätzlich zu seiner Ergebnisdatei schreibt der Masterprozessor, der die Identität 0 besitzt, eine Hauptdatei auf Platte, *ISURF.head*, die zeilenweise alle Namen der zugehörigen Ergebnisdateien im ASCII-Format enthält, angefangen mit *ISURF000*

bis *ISURF127*, wenn 128 Prozessoren beteiligt sind.
Seitens der Workstation gibt es korrespondierende Lesemoduln unter AVS, die diese Hauptdatei interpretieren und nacheinander alle in ihr aufgeführten Dateien einlesen. Die Daten werden in serielle Felder integriert, um sie den geeigneten Geometrie–Mapper–Moduln verfügbar zu machen.
Für den direkten Empfang über das Netz mittels Sockets existieren gleichsam Moduln für beide Datenformate STR und CVP, sowie für den direkten Empfang von Ergebnisdaten im Format PATRAN aus einer numerischen Simulation. Näheres hierzu ist in Kapitel 6.3 nachzulesen.

5.2 Parallele Algorithmen

Im vorigen Kapitel wurden die seriellen Darstellungsalgorithmen vorgestellt, die neben dem Gesichtspunkt der Effizienz vor allem auch Flexibilität bezüglich ihres Einsatzes aufweisen mußten. Das bedeutet, daß sie im wesentlichen drei Anforderungen zu genügen haben:
Zum einen muß der sequentielle Betrieb, eingebunden in das abgeschlossene Modulpaket innerhalb von AVS, und zum anderen der parallele Einsatz in beiden Betriebsweisen, sowohl im Standalone Modus bei unabhängiger paralleler Visualisierung, als auch im integrierten Modus gewährleistet sein. Im letzteren Fall müssen die Algorithmen ihre Integrationsfähigkeit beweisen, sie sollten also in eine numerische Analyse problemlos und ohne aufwendige Modifikationen eingepaßt werden können und zur Zeit- und Speicherersparnis möglichst auf vorhandene Datenstrukturen zurückgreifen können.
Für die große Mehrheit der Algorithmen lassen sich diese Vorgaben gerade in Hinblick auf die Parallelisierung gut verwirklichen, weil diese Algorithmen auf Elementen basieren, und somit jedes Finite Element für sich betrachtet werden kann. Die Berechnungen liefern Lösungen wie Flächensegmente innerhalb des durch ein Element vorgegebenen Raumes unabhängig von der Kenntnis anderer Elemente und Strukturen.

Die aus den Flächensegmenten resultierende und für ein späteres Geometrie–Mapping erforderliche Polygoninformation wird für alle dem Prozessor verfügbaren Elemente erzeugt, aufgesammelt und in einem geeigneten Ergebnisformat entweder abgespeichert oder direkt transferiert.
Der Parallelisierung kommt das elementweise Vorgehen sehr entgegen, kann zum einen doch die räumliche zusammenhängende Aufteilung (*domain decomposition*) einer FE-Analyse wie z.B. FEPS verwendet werden, zum anderen aber auch im

Postprozessing eine willkürliche Aufteilung der Elemente auf die Prozessoren erfolgen, immer unter dem Aspekt des Lastausgleichs.
Für eine Minderheit der Algorithmen ist dieses Konzept jedoch untauglich, nämlich ausgerechnet für die Berechnung von Stromliniendaten und eine Wanderung von Teilchen oder Flächen in einem Strömungsmedium. Dort ist die Information der unmittelbaren Umgebung und der Zusammenhang der Elemente zwingend, was eine willkürliche Umverteilung der Elemente von vorneherein unsinnig macht. Es muß hier ein anderes Konzept der Parallelisierung erarbeitet werden und dazu kommen in Abhängigkeit vom Betriebsmodus zwei Möglichkeiten in Frage:
Wo das Rechenproblem seriell vorliegt, bietet es sich an, jeden Prozessor auf dieselben Gesamtdaten anzusetzen und die Startpunkte der Stromlinien bzw. der Partikel umzuverteilen. Allerdings ist diese Methode für große Probleme nicht geeignet, da die Speicherkapazität auf den verwendeten Prozessoren zumeist noch unter 32 MByte lag.

Die zweite Lösung existiert bisher nur als Konzeptstudie, soll aber dennoch angeführt werden. Hier liegen die Daten verteilt vor, was bedeutet, daß bei der Problemanalyse jeder Prozessor auf einem zusammenhängenden Teilgebiet (*subdomain*) arbeitet und nach jedem Rechenschritt die Daten der mit den Nachbargebieten gemeinsamen inneren Ränder aktualisiert. Anschaulich und bezogen auf ein strömendes Medium heißt das, die Strömung wird über die inneren Ränder transportiert, die physikalisch nicht existieren dürfen. Dies ist dann gewährleistet, wenn die Beiträge der Knoten und Elemente der inneren Grenzen beider Gebiete Eingang finden.
Bei der Berechnung der Stromliniendaten ist man also auf die Kommunikationsstruktur der parallelen Analyse angewiesen und muß sich ihrer bedienen, sobald Teilchen über innere Gebietsränder wandern.

5.2.1 Einige Parallelisierungskriterien

Für eine Parallelisierung können im Kontext der numerischen Visualisierung generell fünf Klassen unterschieden werden. Vier Klassen orientieren sich dabei an der Datenbasis und eine am Algorithmus selbst. Je nach der Datenbasis können entweder Knotendaten, Elementdaten, zusammenhängende Gebiete (*subdomains)* oder Einheiten (*items*) auf Prozessoren umverteilt werden.
Eine Zuordnung der wichtigsten in Kapitel 4 eingeführten Algorithmen wird unter diesen Aspekten in Abbildung 5.5 vorgestellt. Die Mehrheit der Algorithmen ist an Elementdaten orientiert, also können diese Algorithmen elementweise arbeiten, ohne den geometrischen Zusammenhang der Gitterstruktur zu kennen. Genauso können sie auf zusammenhängende Gebiete angewandt werden, die ja auch aus Elementen

bestehen. Eine Aufteilung nach Knotendaten kommt nur für die Berechnung von Vektoren in Betracht und das für den einfachsten Fall, wenn die Orientierung eines flächig dargestellten Vektors im Raum, wie es in Abschnitt 4.3.7 beschrieben ist, nicht gefordert ist.

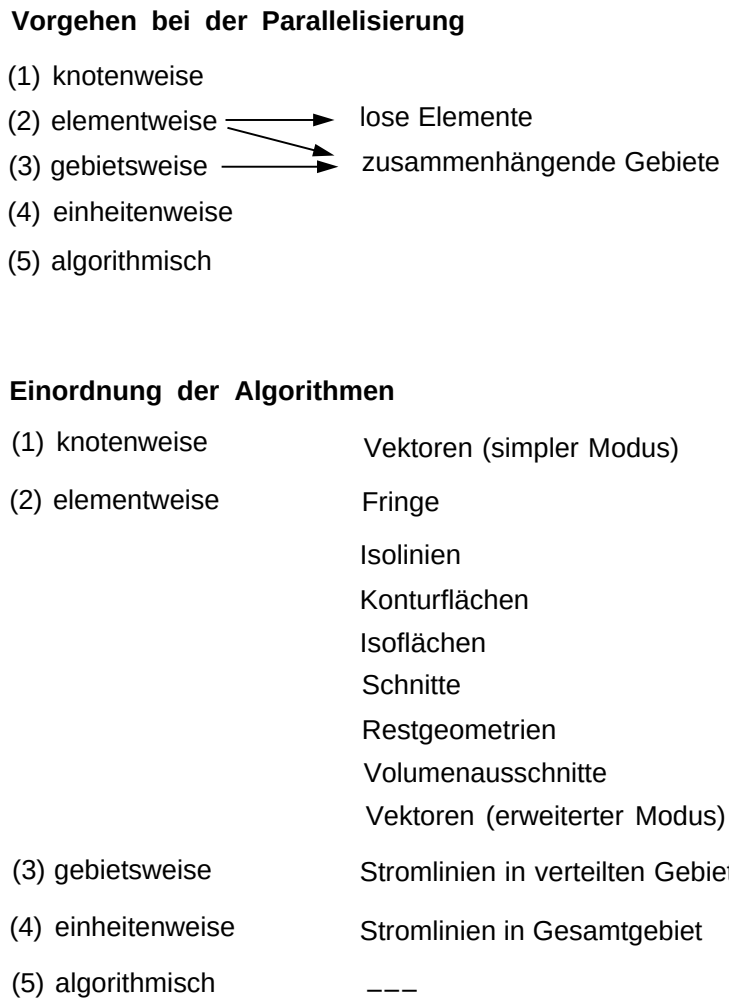


Abb. 5.5: Einteilung der parallelen Algorithmen

Die algorithmische Methode der Parallelisierung, wie sie z.B. bei der Invertierung von großen Matrizen eingesetzt werden kann, war nicht Gegenstand der Arbeit.

5.2.2 Elementparallele Algorithmen

Algorithmen, die elementweise vorgehen, die also beispielsweise Isolinien, Isoflächen, Konturflächen, aber auch Schnittflächen und Volumenausschnitte berechnen, können

ohne große Änderungen aus der seriellen in die parallele Anwendung übertragen werden. Zu Beginn müssen lediglich Maximal- und Minimalwerte physikalischer Größen ermittelt und zusammen mit einigen Visualisierungs- und Steuerparametern jedem Prozessor bekannt gemacht werden. Jeder Prozessor arbeitet daraufhin ohne Kommunikationsbedarf seine Elemente ab und generiert am Ende sein eigenes Datenformat.
Für eine prozessorweise Datenübertragung an eine Graphik-Workstation müssen vom Masterprozessor Gesamtzahlen über Anzahl der Polygone, der Polygonknoten und der Länge des Ergebnisvektors an den Polygonknoten ermittelt werden, damit auf der Rendererseite genügend Speicherplatz allokiert werden kann.

5.2.3 Parallele Stromlinienberechnung

Wie bereits in Abschnitt 4.3.8 ausgeführt wurde, wird im stationären wie im instationären Fall eine Stromlinie numerisch so gewonnen, daß, ausgehend von einem für sie maßgeblichen Startpunkt, iterativ entlang des örtlichen Geschwindigkeitsvektors integriert wird. Somit wird eine Folge von Punkten gewonnen, die allesamt auf der approximierten Strom- bzw. Bahnlinie liegen. Anschaulicher wird der Sachverhalt, wenn man einen Startpunkt als masselosen Partikel betrachtet, der im Laufe der Zeit durch das Strömungsgebiet wandert, wobei der zurückgelegte Weg die Stromlinie kennzeichnet. Aus den verschiedenen Orten und den zugehörigen Zeiten ergibt sich so eine Weg-Zeit-Beschreibung des Partikels.

Eine Weg-Zeit-Beschreibung aller Partikel kann als ein Datenformat begriffen werden, auf dessen Grundlage der serielle Geometrie–Mapper ein oder mehrere Abstrakte Visuelle Objekte berechnet, angefangen von Stromlinien bis hin zur Verfolgung von Partikeln und Oberflächen (siehe Abschnitte 4.3.8 bis 4.3.11). Wurde dieses STR-Format parallel erzeugt, kann es vom Parallelrechner aus entweder direkt an eine Workstation transferiert oder prozessorweise in Datensätzen abgelegt werden, um zu einem späteren Zeitpunkt eingelesen und weiterverarbeitet zu werden.

5.2.3.1 Stromlinien im Gesamtgebiet

Die Methode entspricht vom Konzept her der seriellen. Jeder Prozessor erhält vom Master das gesamte Strömungsgebiet zugewiesen Die Parallelität beruht hier in einer einheitenweisen Aufteilung. Einheiten, in diesem Fall Partikel, werden möglichst gleichmäßig auf die Prozessoren umverteilt. Kleinere Lastenungleichgewichte entstehen hier lediglich durch unterschiedliche Berechnungsdauer einzelner Stromlinien. So

werden zum Beispiel wesentlich mehr Strömungspunkte benötigt, um Gebiete mit hohen Vektorgradienten aufzulösen, d.h. die damit verbundene starke Krümmung der Stromlinie zu approximieren. Wie im seriellen Fall baut jeder Prozessor Nachbarschaftslisten auf und iteriert seine Stromlinien nach dem adaptiven Shoot-Verfahren.

5.2.3.2 Stromlinien in verteilten Gebieten

Bei der Berechnung von Stromlinien in verteilten Gebieten entsteht im Gegensatz zur Gesamtgebietsmethode ein Lastenungleichgewicht durch die Häufung momentaner Aufenthaltsorte von Partikeln in einem Teilgebiet und die daraus resultierende Beschäftigungslosigkeit einiger Prozessoren. Das Verhalten ist dabei in hohem Maße von der Strukturierung der Gebiete abhängig, d.h. wie die Gebiete bezogen auf die Vorzugsrichtung des Strömungsflusses im Gesamtgebiet angeordnet wurden. Zum Vergleich einer Partikelhäufung sind in Abbildung 5.6 zwei mögliche Gebietsaufteilungen fiktiv für eine Kreiszylinderumströmung und einer Aufteilung des Problems in sechs Teilgebiete dargestellt. Während in Situation a) zum Zeitpunkt t_o nur ein Prozessor aktiv ist und zu den Zeitpunkten t_i und t_k immerhin jeweils drei, sind in Situation b) durchweg fünf der sechs Prozessoren aktiv an der Berechnung beteiligt.

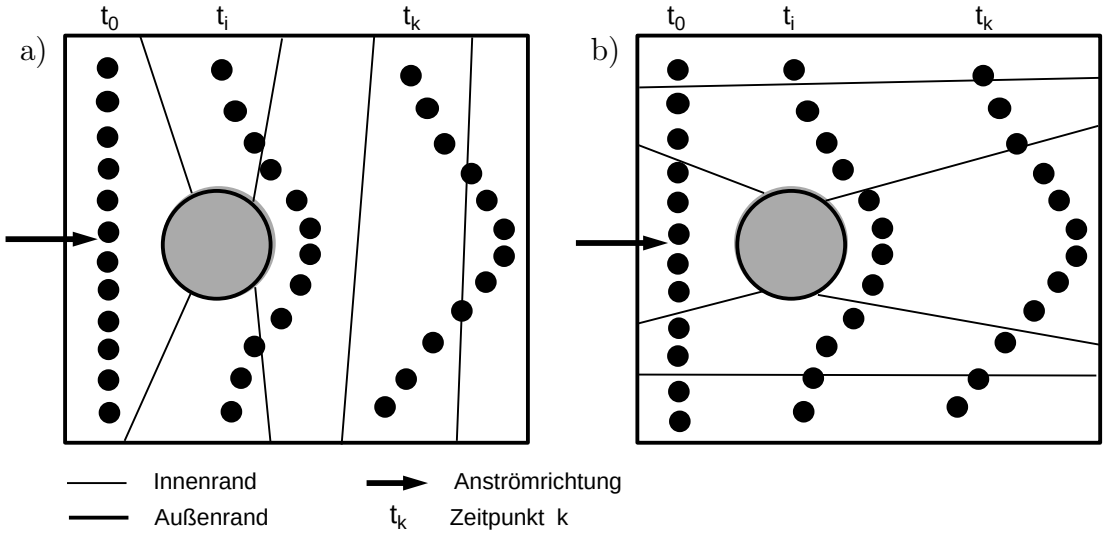


Abb. 5.6: Zur gebietsparallelen Berechnung von Stromlinien

Jeder der Prozessoren berechnet denjenigen Anteil der Stromlinien, der durch sein ihm zugeordnetes Gebiet verläuft. Interessant wird es, wenn Stromlinien an Gebietsränder stoßen. An den Außenrändern, die das Gesamtgebiet begrenzen, wird die Stromlinie iterativ auf die Randlinie treffen und schließlich abbrechen, sobald die Mindestschrittweite unterschritten wurde. Bei Innenrändern jedoch, die durch

die Aufteilung in Gebiete entstehen, liegt der Fall anders. Hier muß geprüft werden, ob das aktuelle Element, in dem sich der Partikel bzw. der momentane Endpunkt der Stromlinie gerade aufhält, ein Innenrandelement ist und ob dieses Element Innenrandknoten besitzt.
Sobald nach einem Shoot-Schritt der neue Ort außerhalb des Teilgebietes liegt, (jedoch nicht außerhalb das Gesamtgebietes, sondern in einem der Nachbargebiete), sendet der Prozessor die Identität und alle anderen Angaben über den bisherigen Verlauf der Stromlinie an den Prozessor, der dieses angrenzende Gebiet besitzt. Die Identität des Empfängerprozessors liefert hier das gemeinsame Innenrandsegment.
Jeder Datenaustausch zwischen Prozessoren erfolgt asynchron, der Sendeprozeß in zwei Schritten. Aktive Prozessoren, also Prozessoren, die momentan Stromlinien berechnen, versuchen alle n Shoot-Schritte von ihren Gebietsnachbarn Daten zu empfangen. Die Größe n ist einstellbar und gegebenenfalls zu optimieren. Zunächst wird abgefragt, ob überhaupt Stromliniendaten übermittelt werden können. Nur wenn dem so ist werden in einem zweiten Schritt die Gesamtdaten der jeweiligen Stromlinie übernommen. Passive Prozessoren hingegen versuchen permanent von ihren angrenzenden Gebieten Daten zu erhalten und übernehmen in gleicher Weise Stromliniendaten, sowie diese vorliegen.
Eine Stromlinie wird immer dann abgebrochen, wenn sie entweder auf einen Außenrand trifft oder die vorgegebene Zahl von Shoot-Iterationen erreicht hat. Beendete Stromlinien werden mit einem Stoppbit gekennzeichnet. Das eindimensionale Feld, das alle Stoppbits enthält, wird mit jedem Datenaustausch mitgeliefert. Sind alle Stoppbits gesetzt, kann jeder Prozessor terminieren, womit sichergestellt ist, daß kein passiver Prozessor endlos auf Daten wartet.
Daten werden über die Innenrandknoten ausgetauscht nach Prinzipien, die bei der parallelen Netzgenerierung und der darauf basierenden FE-Analyse festgelegt werden [17, 83, 66]. Leider lag zu Beendigung der vorliegenden Arbeit die zugehörige Software gerade für dreidimensionale Probleme nicht vor, so daß an dieser Stelle nur das Konzept präsentiert werden kann. Eine parallele Stromlinienberechnung dieser Art funktioniert überdies nur eingebunden in das Analysepaket, da der Datenaustausch, wie eben angesprochen wurde, über die besonderen Kommunikationsstrukturen der parallelen FE-Analyse abgewickelt wird, was neben anderem beinhaltet, daß jeder Prozessor Kenntnis darüber besitzt, welche Knoten entweder allgemeine Knoten, innere oder äußere Randknoten sind.
Anwendungen auf Parallelrechnern, die die in diesem Kapitel aufgezeigten Programmstrukturen verwenden, werden im Anschluß an dieses Kapitel im Zusammenhang mit dem Datenaustausch vorgestellt.

Kapitel 6

Datenaustausch zwischen Systemkomponenten

Dieses Kapitel beschreibt den Datenaustausch zwischen einzelnen Systemkomponenten, der über Transmitterprozesse erfolgt und über diese gesteuert wird. An erster Stelle soll eine Übersicht über den größeren Zusammenhang gegeben werden, in dem dieser Datenaustausch steht, und hier maßgeblich die Architektur bzw. der Schichtenaufbau eines Netzwerks [76, 77]. Ferner wird die Interprozeß–Kommunikation der Transportebene erläutert, welche letztendlich die Basis für das Transmitterkonzept ist. Schließlich wird das Konzept und die Funktionsweise der Transmitter–Software aufgezeigt und der Betrieb von Transmittern sowie deren Steuerung durch AVS–Moduln an Hand typischer Anwendungen exemplarisch vorgestellt.

6.1 Architektur eines Netzwerks

Einen Verbund mehrerer voneinander unabhängiger Computer bezeichnet man als Computernetzwerk. Die Verbindungsleitungen zwischen diesen einzelnen Hostrechnern oder Knoten, seien es Glasfaserkabel oder ein Satellit, werden Kommunikations-Subnetz oder kurz das Subnetz genannt. Um Informationen innerhalb von Computernetzwerken austauschen zu können, bedarf es sowohl einer strukturierten Netzwerk–Architektur als auch einer Netzwerkverwaltung, für welche jedoch der Benutzer selbst verantwortlich ist.
Netzwerke sind hierarchisch organisiert, bzw. in mehrere Schichten oder Ebenen untergliedert, die alle aufeinander aufbauen und in einem gewissen Dienstverhältnis

zueinander stehen. Exakter ausgedrückt, delegieren übergeordnete Schichten Anwendungsarbeit an tiefere Schichten. Jede der Schichten übernimmt festgelegte Aufgaben und kommuniziert mit den benachbarten Schichten, nach unten wie nach oben, über exakt definierte Schnittstellen.

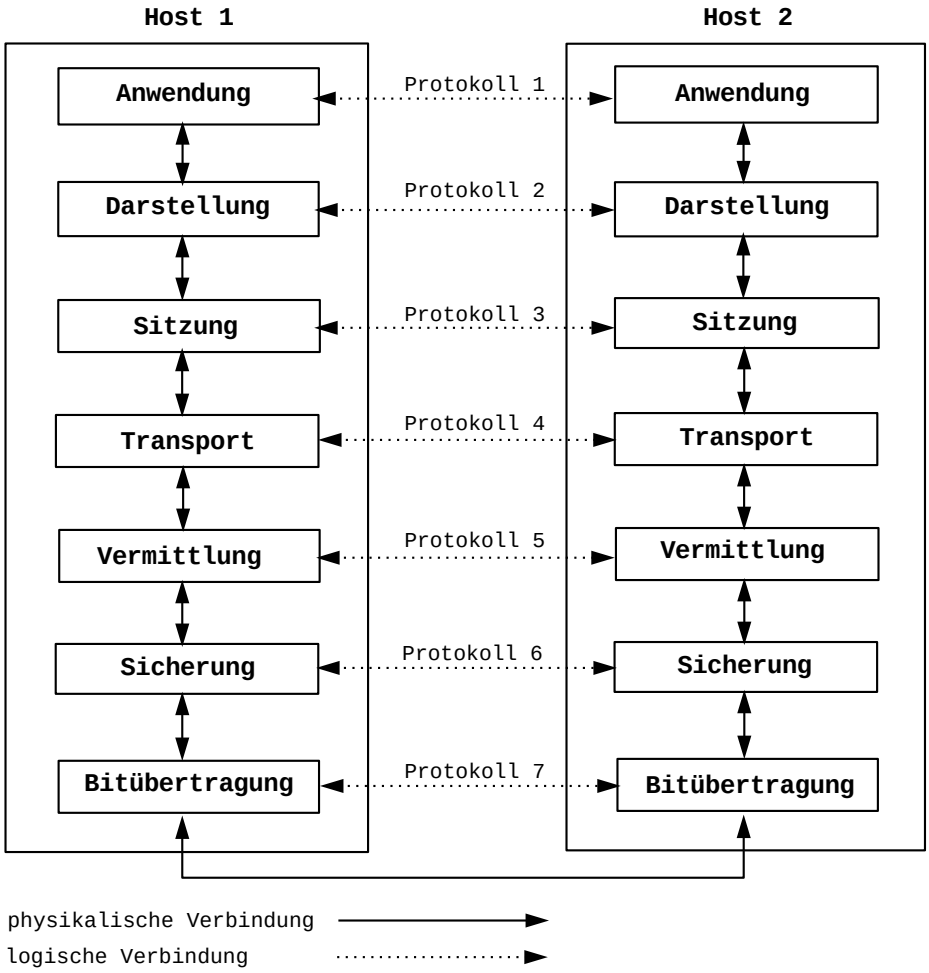


Abb. 6.1: Die Abstraktion der Hierarchie der Schichten im OSI–Referenzmodell

Zum Aufbau der Schichten existieren historisch bedingt verschiedene abstrakte Modelle, die im Prinzip ganz ähnlich strukturiert sind. Am häufigsten finden heute die Netzwerkmodelle ARPA [33] und OSI [13] Verwendung. Wenngleich die hier zu beschreibende Software für den Datenaustausch unter dem von Berkeley–UNIX unterstützten TCP/IP entwickelt wurde, das von vielen Rechnern im ARPA–Netz verwendet wird, soll hier dennoch der Netzwerkaufbau an Hand des OSI–Referenzmodells beschrieben werden, das von der "International Standards Organisation" ISO entwickelt wurde und sich immer mehr durchsetzt. Andere Netzwerke unterscheiden sich z.B. hinsichtlich der internen Schnittstellen und Anzahl der Schichten.

Das OSI–Referenzmodell (OSI für Open Systems Interconnection) wurde eingeführt, um für die Übertragung von Information einheitliche Standards festzuschreiben, welche die Verknüpfung verschiedener Netzwerktypen vereinfachen. Nach diesem heute weit verbreiteten Modell werden sieben Protokollschichten unterschieden, angefangen mit der untersten, – der Bitübertragungsschicht, – der die Sicherungs-, Vermittlungs-, Transport-, Sitzungs-, Darstellungs- sowie als höchste die Anwendungsschicht übergeordnet sind.
Das OSI–Referenzmodell ist in Abbildung 6.1 skizziert. Bei einem Verbindungsaufbau tritt jede Schicht des Senders mit der ihr korrespondierenden Schicht des Empfängers in Kontakt. Die Regeln, Vereinbarungen und Formate, die die Kommunikation auf jeder Schicht bestimmen, werden Protokolle genannt. Protokolle existieren logisch und nicht eigentlich physikalisch, d.h. der tatsächliche physikalische Austausch findet ausschließlich auf der untersten Schicht statt.
Welche Aufgaben fallen nun jeder einzelnen Schicht zu?

- Die **Bitübertragungsschicht** ist mit der Übertragung von rohen Bits über einen Kommunikationskanal befaßt, wobei das Augenmerk hierbei auf mechanischen, elektrischen sowie prozeduralen Schnittstellen liegt, bzw. Eigenschaften beider Seiten (Sender und Empfänger) miteinander in Übereinstimmung gebracht werden müssen, damit überhaupt ein sinnvoller Austausch stattfinden kann.

- In der **Sicherungsschicht** wird der Strom der rohen Bits in kleine Dateneinheiten unterteilt, in sogenannte Datenrahmen gepackt und an die Vermittlungsschicht weitergereicht. Die einzelnen Rahmen werden durch spezielle Anfangs- und Endbitsequenzen kenntlich gemacht. Auf diese Weise können Übertragungsfehler korrigiert und gegebenenfalls beschädigte oder verlorengegangene Rahmen erneut geschickt bzw. zuviel duplizierte Rahmen vernichtet werden. Eine weitere Aufgabe der Sicherungsschicht neben der Datenkennzeichnung ist die Datenflußregelung, welche erforderlich ist, um z.B. eine Datenüberschwemmung respektive Pufferüberläufe zu vermeiden, etwa wenn der Empfänger langsamer arbeitet als der Sender.

- Die **Vermittlungsschicht** ist für die Steuerung des Subnetz–Betriebs verantwortlich. Dies umfaßt das Routing bzw. die Festlegung der Leitwege im Netzwerk vom Sender- zum Empfängerort (statische Wege oder dynamische, die für jedes Paket neu bestimmt werden, um das Netz optimal auszulasten). Weitere Aufgaben sind die Beseitigung von Stauungen bzw. Engpässen wenn sich zu viele Datenpakete zur selben Zeit im Subnetz befinden, sowie die Lösung aller Probleme, die entstehen, wenn mit fremden Netzwerken in Verbindung getreten werden muß, unterschiedliche Adressierung, verschiedene Protokolle

usw. Schließlich sei noch die Kundenabrechnung erwähnt, für die die Anzahl der transferierten Bits oder Pakete gezählt werden muß.

- Die **Transportschicht** stellt den darüberliegenden Schichten eine zuverlässige Endpunkt-zu-Endpunkt-Verbindung zur Verfügung und gewährleistet den Transport der Daten vom Quell- zum Zielrechner. Sie ist die tiefste Schicht, zu der der Benutzer direkten Zugang und Eingriffsmöglichkeiten besitzt, das Netzwerk handzuhaben.
- Die **Sitzungsschicht** bringt eine Verbesserung der Dienste der Transportschicht durch zusätzliche Einrichtungen wie z.B. Datenrettung nach Systemabstürzen. Weitere gehobene Sitzungsdienste, die der Darstellungsschicht zur Verfügung gestellt werden müssen, sind neben anderen Token–Management, Organisation und Synchronisierung des Dialogs zwischen zwei Prozessen.
- Die **Darstellungsschicht** vereinheitlicht die Darstellungsweise von Datenstrukturen und deren Beschreibung. Da verschiedene Computer verschiedene interne Darstellungsweisen kennen, wird in die Standarddarstellung des Netzwerkes konvertiert. Zu den weiteren Aufgaben die Informationsdarstellung betreffend gehören Kodierung, Verschlüsselung und Komprimierung der zu übermittelnden Daten.
- Die **Anwendungsschicht** fungiert als Mittler zwischen korrespondierenden Anwendungsprozessen, die untereinander Informationen austauschen. Dabei handelt es sich um anwendungsspezifische Protokolle wie elektronische Post, Datei–Transfer, Verzeichnisabfragen, virtuelles Terminal und andere. Die Abstraktion virtuelles Terminal wird bespielsweise verwendet, um inkompatible Terminalarten, Editoren und andere Programme einander anpassen zu können.

Nach dieser Kurzübersicht über den Schichtenaufbau von Netzwerken [78, 60] soll nun diejenige Schicht eingehender betrachtet werden, auf der die Programmierung des direkten Datenaustauschs zwischen Prozessen erfolgt, nämlich die Transportschicht.

6.2 Die Transportschicht

Die sieben Schichten des OSI–Referenzmodells lassen sich zwei Gruppen zuordnen, den Transportdienstanbietern und den Transportdienstbenutzern. Zur Benutzergruppe rechnet man die drei oberen Schichten, während die vier unteren die Dienstleister sind.
Die Transportschicht ist deshalb die wichtigste Schicht in einem Netzwerk, da sie

zum einen die unterste Ebene darstellt, die einen verlässlichen Grundservice für einen Datentransfer von Endpunkt zu Endpunkt gewährleisten kann, zum anderen jedoch gleichermaßen die höchste Schicht ist, die Transportdienste anbietet. Somit fällt ihr eine Schlüsselrolle zu, eben weil sie bei der zuverlässigen Datenübertragung die Schnittstelle zwischen Anbieter und Benutzer darstellt, und alle tieferen Schichten des Subnetzes gegen die Benutzer abgeschirmt.
So sorgt sie z.B. dafür, daß Anwendungsprogramme mit einem Standardset von Dienstelementen erstellt werden können, und daß diese auf verschiedenen Netzwerken laufen, ohne daß man sich um die verschiedenen Subnetz–Schnittstellen oder eine zuverlässige Datenübertragung zu kümmern hat.

Heute stehen einige kommerzielle Softwarepakete wie zum Beispiel PVM [63, 26], P4 [63], LINDA [63] und MPI [27] zur Verfügung, mit denen auf Transportebene Daten ausgetauscht werden können. Entscheidendes Manko dieser Pakete ist, daß noch vor Beginn einer Anwendung alle potentiell am Datenaustausch beteiligten Maschinen spezifiziert und das Kommunikationsnetzwerk zwischen ihnen erstellt werden muß. Während der Anwendung ist es nicht mehr möglich, andere Maschinen hinzuzunehmen bzw. nicht mehr benötigte freizugeben. Im Gegensatz dazu wird bei dem hier vorgestellten Prinzip eine Verbindung erst zu dem Zeitpunkt aufgebaut, zu dem es erwünscht wird und mit der Maschine, die interaktiv ausgesucht wurde, was gerade bei Mehrplatzsystemen wünschenswert ist. Solche Verbindungen bleiben nur solange bestehen, wie Daten geschickt bzw. erwartet werden, was über einen Datenkopf geregelt wird, der immer als erstes ausgetauscht wird und als Auftraggeber fungiert.
Für die Programmierung der Software für den Datenaustausch auf Transportebene wurde das Berkeley–IPC zugrundegelegt, da es den Vorteil hat, auf nahezu allen Maschinen verfügbar zu sein. IPC steht für "Inter Process Communication". Das IPC–Paket wurde speziell für C–Shells mit Berkeley–UNIX entwickelt [54]. Es unterstützt TCP/IP, auf das es mit einem Satz von Transportdienstelementen zugreift.

6.2.1 Die Interprozeß–Kommunikation IPC

Das IPC–Paket bietet dem Programmierer verschiedene Möglichkeiten an, Anwendungen zu entwickeln, die kooperierende Prozesse auf derselben oder auf verschiedenen Maschinen verwenden. Im Mittelpunkt steht hierbei der *Socket*.
Sockets sind Endpunkte der Kommunikation zwischen dem Betriebsystem und Prozessen der Benutzer, sozusagen eine temporäre Schnittstelle, die nur solange besteht, wie beide Seiten aneinander gekoppelt sind. Der Systemaufruf *socket* erzeugt eine Datenstruktur innerhalb des Betriebsystems. Die beigeordneten Parameter Adressenformat, Sockettyp und Protokoll definieren die Art der Kommunikation.

Es gibt verschiedene Arten, einen Kommunikationskanal aufzubauen. Die Grundidee der IPC war, Kommunikationskanäle ähnlich der UNIX–Schnittstelle für die Ein- und Ausgabe von Dateien zu gestalten. Ein UNIX–Prozeß besteht immer aus einem Satz von Deskriptoren (wie z.B. *standard input* oder *standard error*), von denen gelesen oder auf die geschrieben wird. Dabei können sich Deskriptoren auf normale Dateien beziehen, auf Geräte wie Terminals, oder aber auf Kommunikationskanäle.
Die Verwendung eines Deskriptors kennt drei Phasen: seinen Aufbau, seinen Betrieb und seinen Abbau. Eine Art von Deskriptoren sind die *Pipes*. Sie erlauben den Transfer von Daten in einer Richtung von einem Prozeß zu einem anderen, wobei jedoch die beiden Prozesse und die Pipe selbst entweder von einem gemeinsamen Vorfahren (*parent*) initialisiert worden sein müssen, oder einer der Prozesse vom anderen abstammt (*parent* und *child*).
Sockets funktionieren im Prinzip ganz ähnlich wie Pipes, haben aber den entscheidenden Vorteil, daß hier kommunizierende Prozesse keinen gemeinsamen Vorfahren benötigen. Das bedeutet, daß sie unabhängig voneinander existieren und somit beispielsweise ein Datenaustausch zwischen zwei Maschinen erst ermöglicht wird.

Domainen (Definitionsbereiche)

Eine Domaine ist, vereinfacht ausgedrückt, ein Namensraum, der neben der Eigenschaft, daß er an einen Socket gebunden werden kann, noch gewisse andere Konventionen enthält.
Die Sockets, die von verschiedenen Programmen aufgebaut wurden, benutzen Namen, um sich aufeinander zu beziehen. Diese Namen werden in Adressen umgewandelt und der Adressenraum einer Domaine zugeordnet. Man unterscheidet verschiedene Domainen für Sockets. Die beiden hier am häufigsten benutzten sind die UNIX–Domaine (auch AF_UNIX) für das Adressenformat UNIX, und die Internet–Domaine (auch AF_INET) für das Adressenformat INET.
In der UNIX–Domaine wird für jeden Socket im Dateiensystem ein Knoten angelegt und ihm ein Pfadnamen innerhalb des Namensraumes gegeben. Andere Prozesse können sich somit auf den Socket beziehen, indem sie den korrekten Pfadnamen angeben. UNIX–Domaine–Namen erlauben die Kommunikation zwischen beliebigen Prozessen auf demselben Rechner.
Die Internet–Domaine ist die UNIX–Implementierung, die auf dem DARPA–Internet-Standard-Protokoll IP, TCP/IP und UDP basiert. Adressen in der Internet–Domaine setzen sich aus Maschinennetzwerkadresse und einer Identifikationsnummer zusammem, die *Port* genannt wird. Internet–Domaine–Namen erlauben die Kommunikation zwischen Prozessen auf verschiedenen Rechnern.

Sockettypen

Es gibt zwei Kommunikationsstile, die mit der Benennung eines Sockettypen festgelegt werden, das verbindungsunabhängige *Datagramm* und der verbindungsorientierte *Stream*.
Stream–Socket–Kommunikation beinhaltet zum einen, daß die Kommunikation über eine Verbindung zwischen zwei Sockets stattfindet und zum anderen, daß diese Kommunikation zuverlässig und fehlerfrei ist, und es – wie bei Pipes üblich – keine Übertragungsgrenzen gibt. Fehlerhafte Nachrichten werden erneut geschickt und Fehlermeldungen dann zurückgegeben, wennimmer ein Prozeß versucht, eine Nachricht zu schicken, nachdem die Verbindung unterbrochen wurde (*broken pipe*).
Die Datagramm–Socket–Kommunikation kommt hingegen ohne Verbindungen aus. Jede Nachricht ist individuell adressiert und wird, wenn die Adresse korrekt ist, im allgemeinen auch erhalten, obwohl das nicht garantiert ist. Datagramme werden oft für Anfragen benutzt, die eine Antwort des Empfängers erfordern. Wurde in einer vernünftigen Zeitspanne keine Antwort erhalten, wird die Anfrage wiederholt. Der Empfänger hält die individuellen Datagramme auseinander d.h. Übertragungsgrenzen werden eingehalten.
Die Benutzung von Datagrammen mag einen Leistungsgewinn bedeuten, doch muß diese mit einer größeren Komplexität des Programms erkauft werden. Außerdem hat man sich jetzt mit verlorengegangenen oder außerhalb der Norm liegenden Nachrichten auseinanderzusetzen.

Protokolle

Ein Protokoll ist ein Satz von Regeln, Datenformaten und Konventionen, die den Datentransfer zwischen den Kommunikationsteilnehmern regulieren. Generell gibt es für jeden Sockettypen, – Stream oder Datagramm, – ein Protokoll innerhalb jeder Domaine, – AF_UNIX oder AF_INET. Das Protokoll eines Sockets wird zusammen mit der Domaine und dem Typ zu dem Zeitpunkt spezifiziert, wenn der Socket aufgebaut wird.

Sockets in der UNIX–Domaine

Socketnamen in der UNIX–Domaine sind Pfadnamen und können wie Dateien entweder absolut oder relativ sein. Da diese Pfadnamen gebraucht werden, um Prozesse miteinander kommunizieren zu lassen, sollten relative Pfadnamen mit Vorsicht gehandhabt werden.
Jedesmal wenn ein Name an einen Namensraum gebunden wird, wird gleichzeitig eine Datei in dem Dateiensystem allokiert. Wird diese Datei nicht mehr deallo-

kiert, existiert der Name selbst dann weiter, wenn der gebundene Socket geschlossen wurde, mit dem Ergebnis, daß bei späteren Programmaufrufen der Name nicht als verfügbar vorgefunden wird. Die Namen werden mit dem Systembefehl *unlink* oder dem Shell–Befehl *rm* gelöscht.
Im Gegensatz zur Internet–Domaine werden Namen der UNIX–Domaine ausschließlich für Rendezvous benutzt und nicht, um Nachrichten zu übermitteln. Deshalb werden ungebundenen Sockets nicht automatisch Adressen mitgegeben.

Sockets in der INET–Domaine

Internet–Adressen spezifizieren eine Hostadresse (eine 32 Bitnummer) und einen *delivery slot*, einen Port auf der Maschine. Diese Ports werden von Systemroutinen gehandhabt, die dazu ein besonderes Protokoll implementieren. Somit stellt eine Internet–Adresse eigentlich ein Tripel aus Protokoll, Port und Maschinenadresse dar.
Eine Assoziation ist eine temporäre oder permanente Spezifizierung eines Paares von kommunizierenden Sockets. Ein Socket wird demnach durch ein Fünftupel identifiziert: Protokoll, lokale Maschinenadresse, lokaler Port, ferne Maschinenadresse, und ferner Port. Eine solche Verbindung kann zeitabhängig sein wenn man Datagrammsockets benutzt. Die Assoziation existiert tatsächlich während des Transfervorgangs.
Das Protokoll für einen Socket wird bei seinem Aufbau gewählt. Die lokale Maschinenadresse für einen Socket kann dabei jede gültige Netzwerkadresse der Maschine sein oder alternativ der *Wildcard*-Wert "INADDR-ANY". Selbst wenn der Wildcard-Wert gewählt wurde, muß ein Programm, das Nachrichten an einen benannten Socket schickt, eine gültige Netzwerkadresse spezifizieren, einfach weil man zwar von Unbekannt empfangen, jedoch nicht an Unbekannt schicken kann.
Die Nummer des Ports der Zielmaschine (*destination port number*) kann als eine Art Postkasten (*mailbox*) verstanden werden, in welche das Protokoll die Nachrichten ablegt. Portnummern von 1 bis 1023 sind für *Daemons* reserviert. Höhere Nummern können von allen Benutzern verwendet werden. Allein der sendende Prozeß muß eine Portnummer spezifizieren. Beispielsweise benutzt das File Transfer Protokoll FTP im allgemeinen Port 20 und 21.
Wird eine Nachricht zwischen zwei Maschinen ausgetauscht, wird sie an die Protokollroutine der Zielmaschine gegeben, welche die Adresse interpretiert, um zu entscheiden, an welchen Socket die Nachricht ausgegeben werden soll. Mehrere verschiedene Protokolle können auf derselben Maschine aktiv sein, werden aber im allgemeinen nicht miteinander kommunizieren. Außerdem ist es verschiedenen Protokollen erlaubt, dieselbe Portnummer zu benutzen.
Im Unterschied zu UNIX–Domainen werden die Internet–Socket–Namen nicht an

das Dateiensystem geknüpft und müssen deshalb auch hinterher nicht entkoppelt werden (*unlink*) wenn ein Socket geschlossen wird.

Dienstelemente der Stream–Socket–Kommunikation

Der Begriff Transport–Dienstelemente versteht sich als eine Gruppe von Systemaufrufen, die es dem Benutzer erlauben, die Dienste der Transportschicht in Anspruch zu nehmen. Es gibt Dienstelemente für den Aufbau, den Betrieb und den Abbau von Sockets. Eine Übersicht der wesentlichen Dienstelemente zeigt Abbildung 6.2. Mit dem Aufruf **socket** wird ein Socket aufgebaut und ihm ein Name und eine Domaine zugewiesen. Der so festgelegte Name wird mit dem Systembefehl **bind** an den Socket gebunden.

socket	Aufbau eines Sockets der definierten Art
bind	Assoziation eines ASCII–Namens zu einem aufgebauten Socket
connect	Verbindungsaufnahme mit einem entfernten Socket
listen	Erstellen einer Warteschlange für eingehende Verbindungsanforderungen
accept	Erwarten oder Entfernen einer Verbindungsanforderung
select	Prüfen einer Gruppe von Sockets auf Sende– und Empfangsbereitschaft
shutdown	Beendigung einer Verbindung auf einem Socket
send, sendto, write	Schicken von Nachrichten an einen entfernten Socket
recv, recvfrom, read	Empfang von Nachrichten vom einen entfernten Socket
close	Abbau eines Sockets

Abb. 6.2: Die gebräuchlichsten Dienstelemente im IPC

Um Daten zwischen Stream–Sockets (Kommunikationsstil Stream) austauschen zu können, muß zwischen ihnen eine Verbindung hergestellt werden. Der Verbindungsaufbau erfolgt asymmetrisch. Hierbei ist einer der Sockets der aktive Endpunkt, der eine Verbindung anfordert, während der andere der passive Endpunkt ist, der Empfangsbereitschaft signalisisert. Mit dem Aufruf **connect** wird ein Socket in einen aktiven Endpunkt umgewandelt, mit dem Aufruf **listen** hingegen in einen passi-

ven Endpunkt, dem genügend Pufferplatz zugewiesen wird, damit er mehrere Verbindungsanforderungen speichern kann. Mit **accept** wird ein Verbindungsanforderung angenommen, falls überhaupt eine zur Verfügung steht. Ist dies nicht der Fall, blockiert der passive Endpunkt bzw. wartet auf eine Anforderung.
Ist die Verbindung schließlich komplettiert, kann mit dem Austausch von Nachrichten begonnen werden. Dienstelemente für das Verschicken von Nachrichten sind **write**, **send**, **sendto**, die für das Empfangen **read**, **recv**, **recvfrom**. Auf die Unterschiede der einzelnen Elemente wird an dieser Stelle nicht eingegangen. Die einzelnen Nachrichten werden nach dem Prinzip der Pipes in der Reihenfolge ihrer Übermittlung empfangen, ohne daß es Übertragungsgrenzen gibt.
Eine Verbindung ist dann zerstört, wenn einer der Sockets geschlossen wurde (**close**). Wenn ein Prozeß nachdem die Verbindung gelöst wurde weiterhin Nachrichten sendet, schickt das Betriebssystem als Fehlermeldung das SIGPIPE–Signal.

6.3 Prinzip und Funktionsweise von Transmittern

Werden beispielsweise in der Visualisierung zur Berechnung und Darstellung verschiedene Maschinen eingesetzt, ist ein effizienter Datenaustausch unabdingbar. Gleiches gilt für den Fall, wenn Berechnung und Darstellung zwar auf derselben Maschine arbeiten, jedoch verschiedenen Prozessen zugeordnet sind.
Umständlich kann ein Datentransfer über Zwischenspeicherung in Dateien erfolgen, mit sich daran anschließendem Dateientransfer mittels FTP und erneutem Einlesen von der Empfängerseite. Eine wesentlich schnellere Methode, die überhaupt erst einen automatisierten Rechenablauf ermöglicht, ist jedoch ein direkter Datentransfer von Anwendung zu Anwendung über das Netz. Im Folgenden soll die hier verwendete, auf den Sockets der Interprozeß–Kommunikation basierende Transmitter–Software beschrieben werden.
Als Ausgangssituation stelle man sich hierzu zwei Prozesse vor, die auf irgendeine Weise Information auszutauschen suchen. Einer der beiden Prozesse wird der Initiator sein, der die Information anfordert, während der andere der Adressat sein wird, an den die Anforderung sich richtet. Um den Austausch anzuregen, setzt der Initiator einen Transmitterprozeß ab, der eine Verbindung herstellen soll. Demnach fungieren Transmitter als Vermittler. Sie können gleichsam auf lokalen wie auf fernen Maschinen initialisiert werden.
Da es, wie im vorangehenden Abschnitt gezeigt wurde, bedingt durch das Prinzip der Sockets ausschließlich Verbindungen zwischen aktiven und passiven Sockets geben kann, und Sockets passiver Natur immer auf eine Verbindung warten, ginge in jedem Fall wertvolle Rechenzeit verloren. Im Extremfall könnte es sogar zum

Blockieren einer Anwendung kommen, wenn aus irgendeinem Grund die aktive Seite keine Verbindung aufnimmt oder anbietet. Da die Socketprozesse als Unterfunktionen einer Anwendung nicht für sich alleine stehen und somit von der Anwendung nicht getrennt eliminierbar sind, zöge dieses den Abbruch etwa einer parallelen Simulationsrechnung nach sich.
Um diesen Eventualitäten vorzubeugen, wurden die Transmitter konzipiert. Zwar kann bei Instabilitäten im Netz (*broken pipe*), selbst hier ein Blockieren eintreten, was sich jedoch durch einfaches Eliminieren (*kill*) des von jeder Anwendung unabhängigen Transmitterprozesses beheben läßt. Jedoch hat dies keinen Abbruch eines der beteiligten Anwenderprozesse zur Folge. Ein weiteres Plus der Transmitter ist, daß bei Bedarf rasch eine Punkt–zu–Punkt–Verbindung hergestellt werden kann, die gerade so lange währt, wie sie auch benötigt wird. Auf diese Weise kann im Mehrbenutzerbetrieb, wo man sich die Zahl der Parallelprozessoren mit anderen Anwendern zu teilen hat, der Rechner interaktiv sehr flexibel eingesetzt werden, ohne daß zu Beginn einer Anwendung Kommunikationskanäle zu reservieren bzw. aufzubauen sind.

Wie arbeitet nun ein Transmitterprozeß? Ein einmal initialisierter Transmitter baut immer zunächst einen Socket auf, der passiver Endpunkt ist und auf eine Verbindung mit einem Adressaten wartet. Erst wenn der Adressat erfolgreich mit diesem Socket verbunden wurde und somit sichergestellt ist, daß ein Datentransfer tatsächlich erfolgen kann, erstellt der Transmitter einen zweiten Socket passiver Natur. Der Initiatorprozeß wird nun seinerseits mit ebendiesem zweiten passiven Endpunkt Verbindung aufnehmen. Wurde die Verbindung komplettiert, kann der Datentransport über den Transmitter erfolgen (Abbildung 6.3).

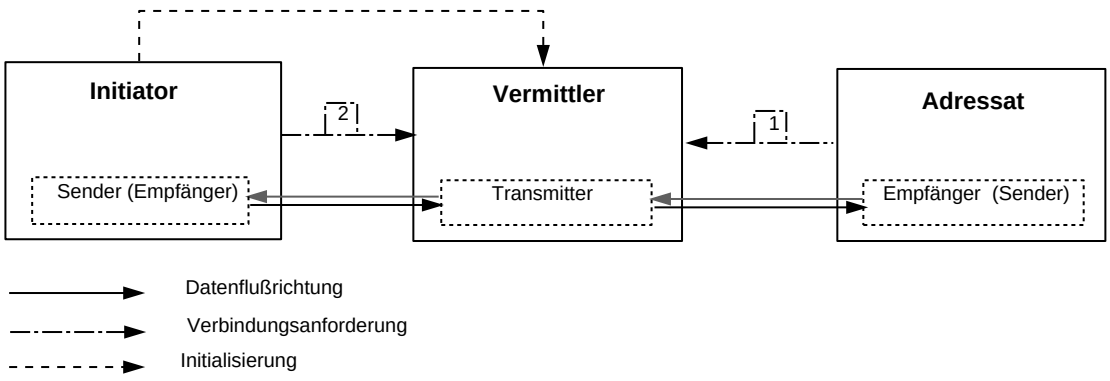


Abb. 6.3: Initiator, Vermittler und serieller Adressat

Dazu sei bemerkt, daß beim Aufbau zugehöriger Kommunikationsendpunkte von vornherein eine bestimmte Adresse (*destination port number*) vereinbart wird. Nur

derjenige Socket kann verbinden, der diese Portnummer verwendet, womit ausgeschlossen sein dürfte, daß ein fremder Socket eines anderen Prozesses im Mehrplatzbetrieb irrtümlicherweise verbindet, es sei denn, er wähle zufälligerweise dieselbe Portnummer.

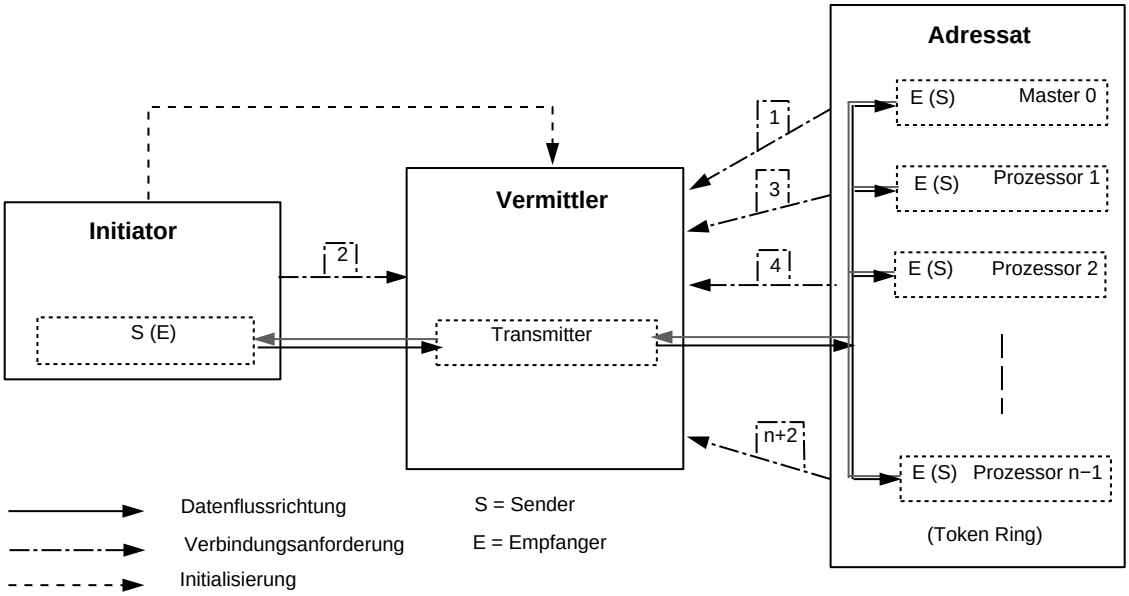


Abb. 6.4: Initiator, Vermittler und paralleler Adressat

Zuallererst empfängt der Transmitter vom Senderprozeß ein spezielles Kenndatenfeld, das die weiteren Instruktionen sowohl für den Transmitter enthält, als auch für den Empfängerprozeß, an den das Feld weitergeleitet wird. Diesen Instruktionen bzw. dem Auftrag zuzurechnen sind Angaben über Anzahl und Art der zu übertragenen Felder, zugehörige Bytezahlen sowie den Zustand des Transmitters nach Beendigung der Übertragung; ob nach Beendigung weitere Übertragungen erfolgen sollen, und wenn ja, in welcher Richtung, oder ob der Transmitter terminieren bzw. abgebaut werden soll. Desweiteren kann optional eine Kennziffer mitgeliefert werden, die seitens der Benutzeroberfläche durch das empfangende Modul mit der Intention interpretiert wird, Fehlsteuerungen des Anwenders zu melden.
Nach dem Auftrag werden die Datenfelder übertragen, und dies paketweise mit einem dazugehörigen Zeiger, der gewährleistet, daß die Pakete auf der Empfängerseite korrekt in Felder rückintegriert werden. Der gleichen Technik wird sich bedient, wenn als Sender n Parallelprozessoren auftreten, der Empfänger jedoch seriell ist und somit eine Anzahl von n Teildatenfelder rückzuintegrieren hat. In Abbildung 6.4 ist dieses illustriert. Nach dem Prinzip des Token Ring verbinden sich die Parallelprozessoren ausgehend vom Master 0 nacheinander mit dem Transmitter und übermitteln ihre Daten. In einer Variante des Datentransfers zwischen paralleler und

serieller Maschine sendet ausschließlich der Master und erhält die Ergebnisdaten der übrigen Prozessoren über das Verbindungsnetz zwischen den Prozessoren.
Die Methode, daß der Master zunächst alle Ergebnisdaten aufsammelt und diese anschließend überträgt, scheitert häufig am begrenzten Speicherplatz eines parallelen Rechenknotens. Generell empfiehlt es sich daher gerade bei großen Datenmengen, die Datenelemente vor dem Austausch zu komprimieren, um so auch die Übertragungszeiten zu minimieren.
Ist ein Auftrag vollständig abgearbeitet werden alle Verbindungen abgebaut, und der Transmitter terminiert selbsttätig.

6.4 Der Transmitter in der Anwendung

Die Anwendung von Transmittern wird hier an Hand zweier Modellfälle entwickelt und die Funktionsweise für zwei Parallelarchitekturen aufgezeigt, einen Parsytec Multicluster und eine Intel Paragon. Darüber hinaus kann der Parallelrechner auch durch einen seriellen Rechner ersetzt werden, z.B. wenn auf ferne Ergebnisdaten zugegriffen oder überhaupt das Visualisierungsgitter fern berechnet wird, und ausschließlich das Geometrie–Mapping und Rendering lokal auf einer Graphik–Workstation stattfindet.

6.4.1 Parallelrechnerunterstützte Visualisierung

Im ersten Fallbeispiel wird der Parallelrechner, wie in Abbildung 6.5 skizziert ist, visualisierungsunterstützend eingesetzt. Visualisierungsunterstützend bedeutet hier, daß rechenintensive Aufgaben, bei denen ein Anwender unzumutbar lange Wartezeiten in Kauf zu nehmen hätte, von einem Parallelrechner übernommen werden. Die Steuerung des gesamten Ablaufs erfolgt dabei interaktiv durch den Benutzer selbst. Aus der seriellen Visualisierungsanwendung heraus, die die Benutzeroberfläche enthält, wird zunächst der Parallelrechner aktiviert bzw. das parallele Gittertransformationsprogramm initialisiert, das sofort einen Auftrag erwartet. Nach den Anweisungen, die das weitere Verhalten des Parallelprogramms bestimmen, werden als Berechnungsgrundlage die Ergebnisse aus der Analyse, FE–Netz und Knotendaten, übermittelt. Auf dieser Basis und den erhaltenen Visualisierungsparametern führt das Parallelprogramm selbständig die Berechnung des Visualisierungsgitters durch. Am Ende werden die erhaltenen Gitterdaten an die serielle Seite zurückübertragen.
Eine Technik wie diese eignet sich beispielsweise zur interaktiven und parallelen Be-

rechnung eines Stromlinienbündels. Einzelheiten zur interaktiven Steuerung sind in Abschnitt 6.5 aufgeführt.

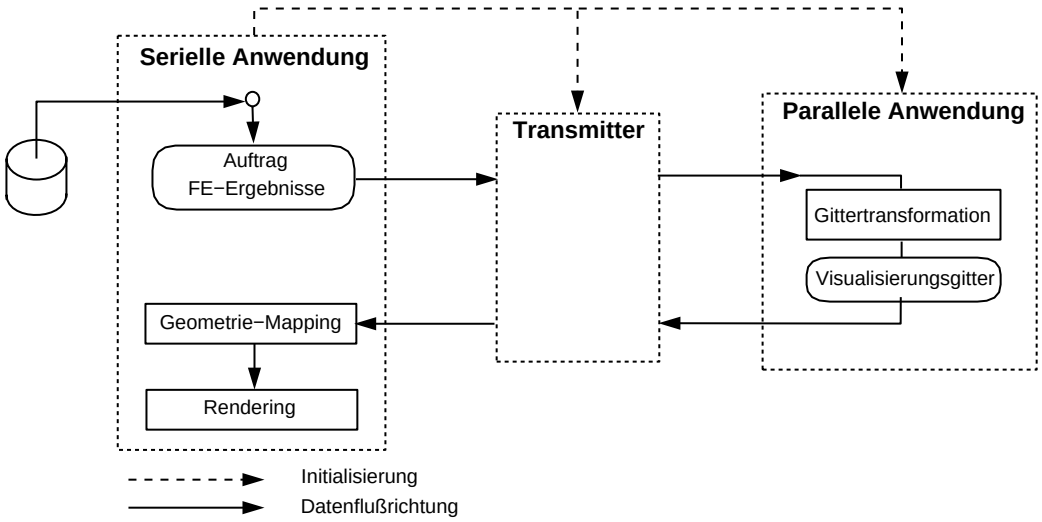


Abb. 6.5: Schema für parallelrechnerunterstützte, interaktive Visualisierung

6.4.2 Visualisierung während laufender FE–Berechnung

Im zweiten Fallbeispiel wird auf dem Parallelrechner eine FE–Simulation durchgeführt und zu einem beliebigen Zeitpunkt – oder auch fortwährend – sollen Daten der laufenden FE–Rechnung visualisiert werden. Zu diesem Zweck sucht das Analyseprogramm – beispielsweise nach jedem Rechenzeitschritt – die Verbindung mit der Visualisierungsanwendung, indem es einen Transmitterprozeß absetzt und versucht, mit dessen korrespondierenden passiven Socket Kontakt herzustellen. Dieser Socket existiert jedoch erst dann, wenn die Empfängerseite auf der Graphik–Workstation ihrerseits durch erfolgte Verbindung ihres aktiven Socket mit dem Transmitter Empfangsbereitschaft signalisiert hat. Steht schließlich die Verbindung, übermittelt die Parallelanwendung nach dem Token–Ring–Prinzip prozessorweise ihre Daten.
Hier gibt es nun zwei Möglichkeiten: Entweder werden FE–Ergebnisse direkt übermittelt und das Visualisierungsgitter dann seriell auf der Workstation berechnet (Abb. 6.6), oder aber die Berechnung des Visualisierungsgitters findet parallel auf den Prozessoren der FE–Analyse statt, wobei dann jeder Prozessor die geeigneten Berechnungsroutinen ruft. Ist dies der Fall, werden am Ende der Berechnung als Ergebnisse die Visualisierungsgitterdaten übermittelt, welche, wie im seriellen Berechnungsfall, anschließend auf der Workstation mit dem Geometrie–Mapper und Renderer polygonisiert und dargestellt (Abb. 6.7).

Nach Abschluß des Datentransfers nimmt das Simulationsprogramm seine Berechnung wieder auf, ebenso wenn keine Verbindung zustandekam.
Dieser zweite Modellfall soll nun am Beispiel zweier Parallelrechnerarchitekturen eingehender betrachtet werden.

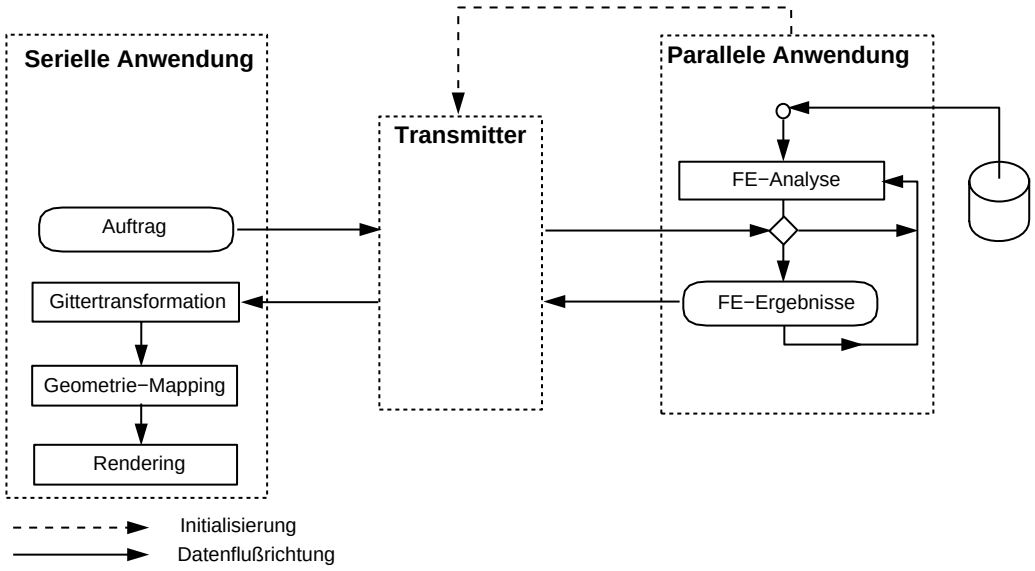


Abb. 6.6: Visualisierung während der FE–Analyse (serielle Gittertransformation)

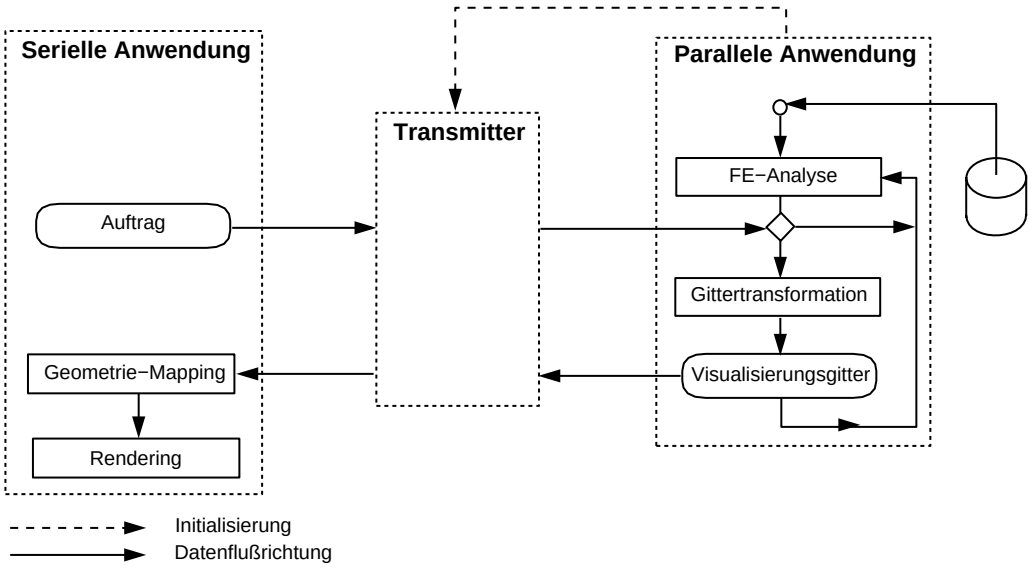


Abb. 6.7: Visualisierung während der FE–Analyse (parallele Gittertransformation)

6.4.3 Übertragungsschema für einen Parsytec–Rechner

Der Datentransfer zwischen serieller Anwendung (Graphik–Workstation) und paralleler Anwendung (Parsytec–Rechner) erfolgt ausschließlich über die an den Parallelrechner gekoppelte Hostmaschine (Abb. 6.8), wobei die Hostmaschine selbst die Graphik–Workstation sein kann (Abb. 6.9). Die Parallelanwendung initialisiert – beispielsweise nach jedem Rechenzeitschritt – auf dem Hostrechner ein Transmitterprogramm, das nacheinander zwei Sockets aufbaut, je einen in den Domainen AF_UNIX und AF_INET bzw. beide in der Domaine AF_UNIX, wenn die Visualisierung direkt auf dem Host läuft.
Zunächst wartet der passive Socket S3 auf eine Verbindung mit der seriellen Anwendung (Socket S4), die durch einen interaktiven Anstoß aus der Benutzeroberfläche heraus zustandekommt. Nur dann, wenn der Kontakt geschlossen wurde, wird auch der passive Socket S2 aufgebaut, der es dem Parallelrechner über seinen Socket S1 ermöglicht, die Verbindung zu komplettieren. womit der Datenaustausch aufgenommen werden kann. Nach Beendigung des Datenaustauschs terminiert das Transmitterprogramm automatisch.

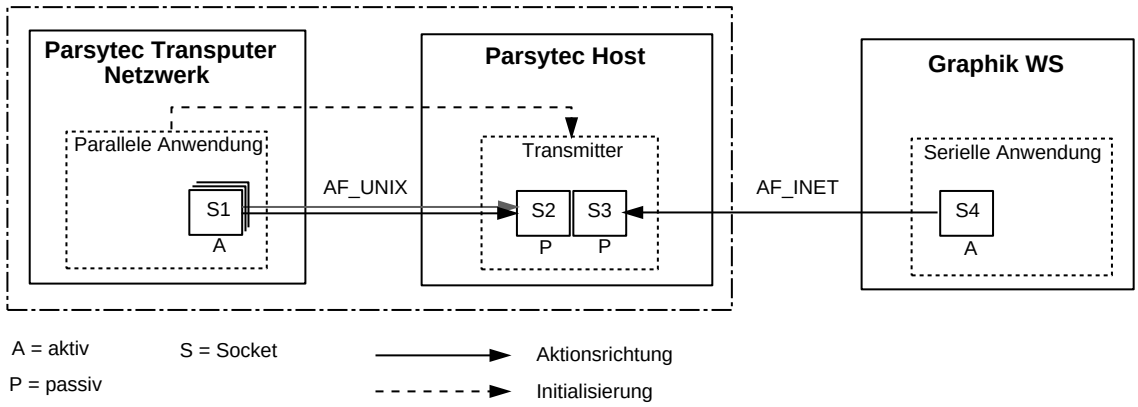


Abb. 6.8: Grobskizzierter Verbindungsaufbau zwischen Parsytec–Rechner und einer Graphik–WS, die nicht Parallel–Host ist

Strenggenommen steht der Socket S1 für eine Reihe von Sockets, die in zeitlicher Abfolge nach dem Prinzip des Token–Ring auf den Parallelprozessoren erstellt werden, und zwar so, daß immer nur ein Socket S1 existiert, der Daten transferiert, die lokal auf dem Prozessor verfügbar sind (siehe hierzu auch Abbildung 6.3). Der Masterprozessor 0 stellt die Urverbindung her und übermittelt dem Transmitter unter anderem auch die Information über die Gesamtzahl der am Datenaustausch beteiligten Prozessoren.
Über eine Ringkommunikation werden als Token zwei Bits von Prozessor zu Prozessor gereicht, das Transferbit und das Errorbit. Ein auf 1 gesetztes Datenaustauschbit

im Datensatz *.parrc* für voreingestellte Parameter (Default) signalisiert eine prinzipielle Bereitschaft zur Datenübertragung. Diese Datei wird vom Masterprozessor nach jedem Rechenzeitschritt gelesen und die Bits den übrigen Prozessoren bekanntgemacht *(broadcast)*. Solange Bereitschaft gezeigt wird, findet nach jedem Rechenzeitschritt eine Kommunikation von Transfer- und Errorbit statt. Hat das Transferbit den Wert 1, bedeutet das, das zur Datenübertragung aufgefordert wird. Ein Errorbit mit Wert 1 führt zum Abbruch des Datenaustauschprozesses. Haben die Bits den Wert 0, geschieht nichts.

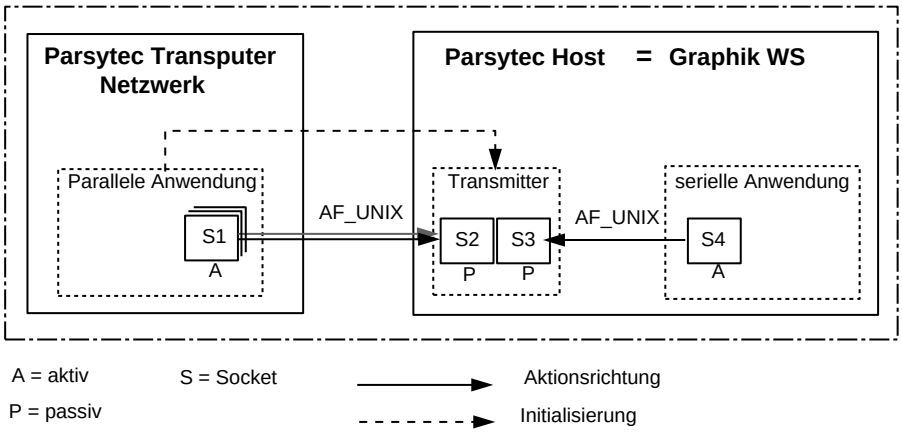


Abb. 6.9: Grobskizzierter Verbindungsaufbau zwischen Parsytec–Rechner und einer Graphik–WS, die gleichzeitig Parallel–Host ist

Da nicht unbedingt nach jedem Rechenschritt von der Benutzerseite her interaktiv eine Verbindung gefordert wird, jedoch jedes Mal ein weiteres Transmitterprogramm abgesetzt wird, können zeitweise mehrere Prozesse dieser Art aktiv sein. Dies ist in sofern nicht von Bedeutung, da, solange ein Transmitter aktiv ist, für nachfolgende Transmitter die Socket–Adresse gesperrt ist, was letzteren automatisch terminieren läßt. Anders ausgedrückt wird stets nur ein aktueller Transmitter existieren, der Verbindungen annehmen kann.
Abbildung 6.10 zeigt nun im Detail den Verbindungsaufbau für den Verbund mit einem Parsytec–Rechner. Betrachtet wird der Fall, daß Ergebnisdaten während der numerischen Simulation visualisiert werden sollen.
Hierbei sind, wie bereits erwähnt, zwei Anwendungen denkbar: In der ersten Anwendung werden Analyseergebnisdaten direkt übertragen, aus denen dann auf der Graphik–Workstation sequentiell durch Moduln des Gittertransformators das für die Visualisierung repräsentative Gitter berechnet wird. In der zweiten Anwendung werden Visualisierungsgitterdaten transferiert, die erhalten werden, wenn auf denselben Prozessoren wie der Analyse die parallelen Berechnungsoutinen des Gittertransformators gerufen werden.

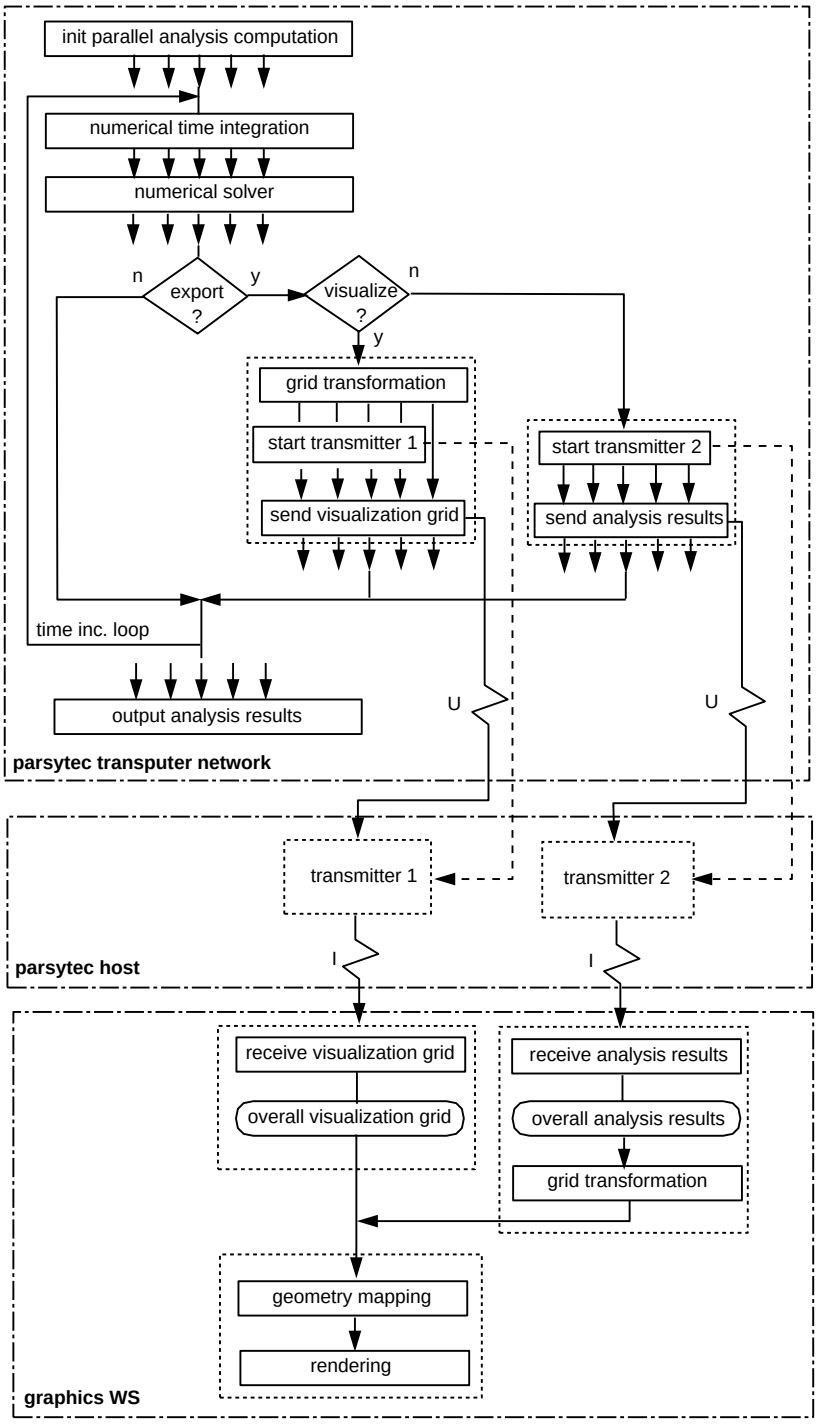


Abb. 6.10: Verbindungsaufbau (detailliert) zwischen einem Parsytec–Rechner, seinem Host und einer fernen Graphik–WS

In beiden Anwendungen jedoch bildet das zusammengeführte Visualisierungsgitter die Datengrundlage für das abschließende Geometrie–Mapping und Rendering auf der Graphik–Workstation. Der Auftrag kann vom Parallelrechner entweder von einer Parameterdatei gelesen oder ihm über einen Transmitter interaktiv vermittelt werden. Beide Möglichkeiten finden nicht Eingang im Bild 6.10, um dieses nicht zu unübersichtlich zu gestalten.

6.4.4 Übertragungsschema für einen Intel–Paragon–Rechner

Auf dem Intel–Paragon–Rechner kann jeder Parallelprozessor für sich genommen unabhängig mit der Außenwelt kommunizieren, ohne daß es eines Hostrechners bedarf. Demnach ist es angezeigt, ein Transmitterprogramm auf der Graphik–Workstation zu initialisieren, und zwar aus der Benutzeroberfläche der seriellen Anwendung heraus (Abb. 6.11). Zu diesem Zweck baut der Transmitter zunächst einen AF_INET–Socket auf (S2), der eine Verbindung von Seiten des Parallelrechners anfordert (S1). Nach erfolgter Verbindung öffnet er den zweiten AF_INET–Socket (S3), auf den die serielle Anwendung bereits wartet. Der Datenaustausch kann beginnen.
Der Masterprozessor 0 stellt hier wiederum die Urverbindung her und übermittelt essentielle Kenndaten, die das Verhalten des Transmitters steuern. Für weitere Einzelheiten der Anwendung gilt das bereits für den Parsytec–Rechner gesagte. Den detaillierten Verbindungsaufbau für den Verbund mit einem Intel–Paragon–Rechner, zeigt schematisch Abbildung 6.12.

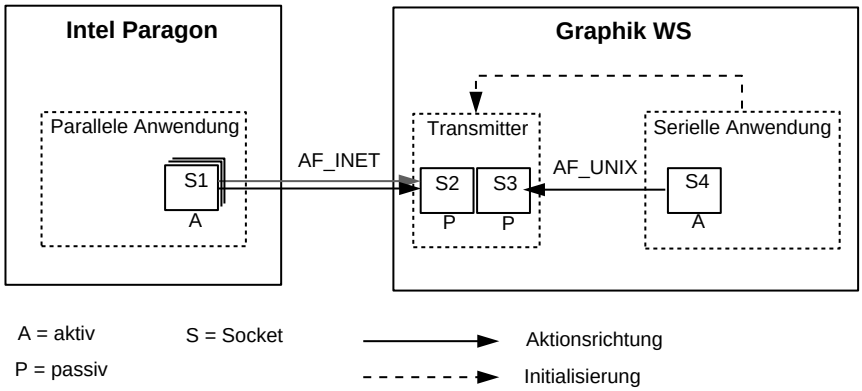


Abb. 6.11: grobskizzsierter Verbindungsaufbau einer Intel Paragon mit einer fernen Graphik–WS

Ein wichtiger Punkt, bei der korrekten Datenübertragung, der bisher noch nicht angesprochen wurde, ist die Art und Weise, wie maschinenintern jedes Byte abgelegt wird. Da beide Parallelrechner *Little–Endian*–Maschinen sind, d.h. jedes Byte in

umgekehrter Reihenfolge gespeichert wird, wie beispielsweise bei Sun–Workstations *(big endian)*, muß seitens der Parallelanwendung sowohl vor dem Verschicken, als auch nach dem Empfang von Daten eine Byte–Konvertierung vorgenommen werden.

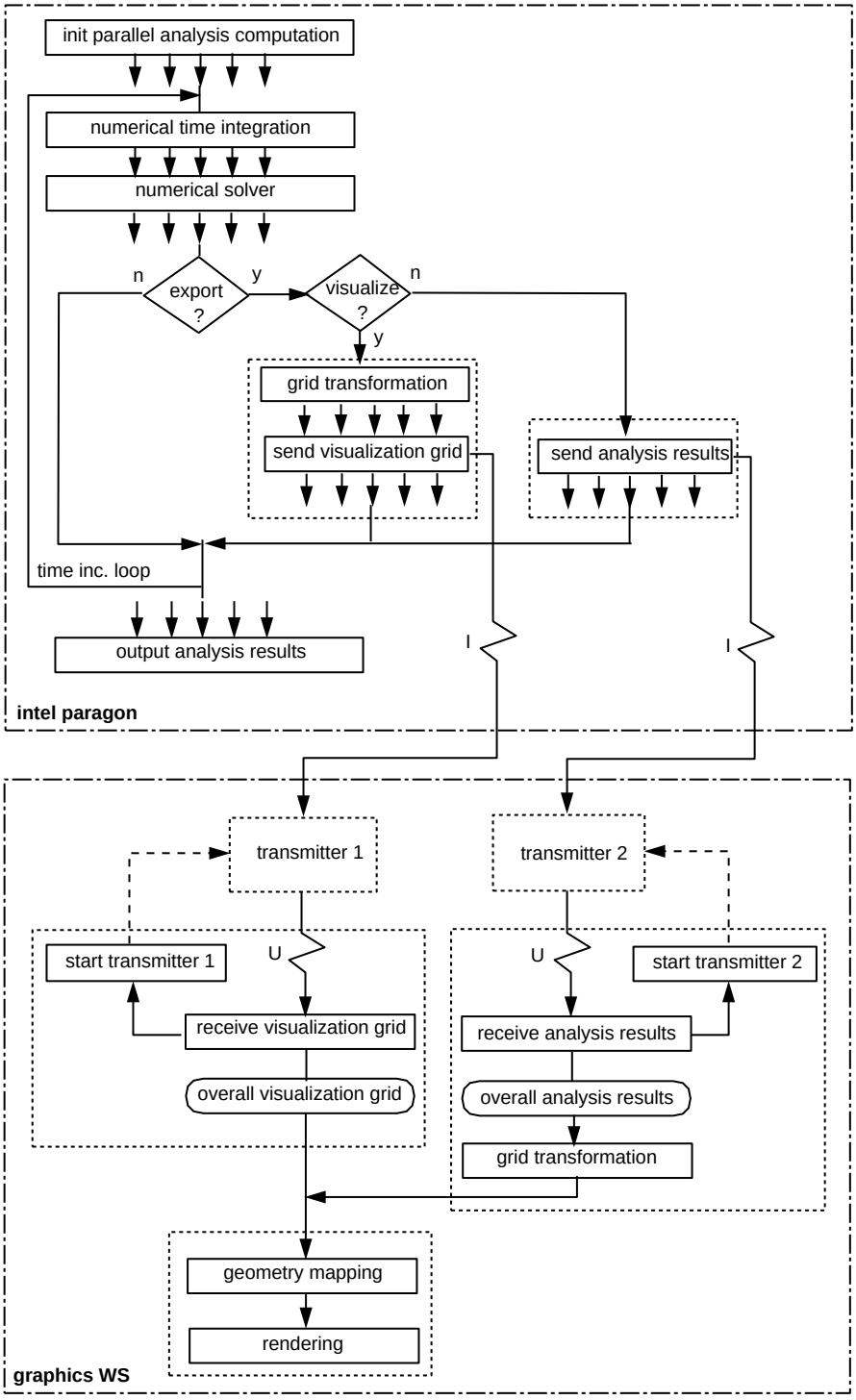


Abb. 6.12: Verbindungsaufbau zwischen Intel–Paragon und Graphik-WS

6.5 Interaktive Steuerung von Transmitteranwendungen unter AVS

Dieses Unterkapitel befaßt sich mit den Möglichkeiten interaktiver Steuerung eines Parallelrechners zur parallelen Visualisierung. An Hand der im vorigen Unterkapitel vorgestellten Modellfälle und einigen Untervarianten wird jeweils das zugehörige Modulnetzwerk beschrieben und Benutzeroptionen und Einflußnahmen auf den Ablauf der Simulation erläutert. Angaben über Benchmarks und Absolutrechenzeiten kompletter Anwendungen sind dem siebenten Kapitel zu entnehmen. Folgende Modellfälle werden betrachtet:

- Parallelrechnerunterstützte Visualisierung
- Visualisierung verteilt vorliegender Datensätze
- Visualisierung während des Ablaufs der FE–Analyse
  - integrierte Simulation mit sequentieller Berechnung des Visualisierungsgitters
  - integrierte Simulation mit paralleler Berechnung des Visualisierungsgitters

Grundsätzlich gibt es ein Modulset, aus dem sich, je nach individuellem Gruppieren der Moduln und Setzen von Modulparametern, ganz verschiedene Anwendungen erzeugen lassen. Das setzt eine gewisse Disziplin und konsistentes Vorgehen des Anwenders voraus, da zum Einen die Anwendungen stark interaktiven Charakter haben bzw. nicht alle Moduln miteinander vernetzt sind, und zum Anderen die parallele Seite bei Nachlässigkeiten wie dem Überschreiten einer intern festgelegten Wartezeit nach Voreinstellung (Default) reagiert. Im Übrigen sind die voreingestellten Werte in dem Datensatz *.parrc* abgelegt, den der Benutzer jederzeit unter Einhaltung eines festgelegten ASCII–Formats ändern darf.
An dieser Stelle soll ein Schwachpunkt des Systems nicht verschwiegen werden, der mit dem Datenfluß in Zusammenhang steht. Die Bausteine, die den Export und den Import der Daten über das lokale Netz bewerkstelligen sind unkonnektiert, d.h. der Ablauf ist, wie zuvor erwähnt, nicht automatisch und somit vom Bediener abhängig. Dadurch wird auch der Ablauf der Verarbeitung, gerade was die Parallelrechnerseite anbelangt, nicht mehr deutlich, denn die parallele Seite hat keinen Zugriff auf die Kommunikationsschale von AVS. Beispielsweise sieht der Anwender deshalb weder den Fortschritt der Verarbeitung, noch wo auf der fernen Maschine gegebenenfalls Fehler auftreten. Die Problematik könnte entschärft werden, wenn die Steuerung

der parallelen Prozesse vollständig in ein Visualisierungsprogrammsystem integriert würde, wie es z.B das visuelle Programmiermodell von COVISE [44] erlaubt.

6.5.1 Parallelrechnerunterstützte Visualisierung

Parallelrechnerunterstützte Visualisierung sieht die zeitweilige Auslagerung aufwendiger Berechnungen von einer Graphik–Workstation auf einen Parallelrechner vor. Hierzu bedarf es einer Bedarfsaufwandsabschätzung, damit der Parallelrechner auch tatsächlich Rechenzeit einspart. Die eingesparte Zeit t_{sp} ergibt sich aus der seriellen Berechnungszeit t_{ser} abzüglich der parallelen Berechnungszeit t_{par} und der Zeiten, die für die Aufteilung t_{dist}, der Übertragung der FE–Daten t_{trans1} und dem Rückholen der Visualisierungsgitterdaten t_{trans2} verloren geht.

$$t_{sp} = t_{ser} - (t_{par} + t_{dist} + t_{trans1} + t_{trans2}) \tag{6.1}$$

Der Aufwand für Aufteilung und Gesamtdatentransfer ist proportional zur Größe des FE–Problems, bzw. der Knoten- und Elementezahl. Der Berechnungsaufwand wächst mit der Problemgröße deutlich stärker an, so daß sich, abhängig vom verwendeten Berechnungsalgorithmus und der eingesetzten Parallelmaschine, eine Mindestproblemgröße schätzen läßt, ab welcher der Aufwand der parallelen Berechnung überhaupt lohnenswert ist.

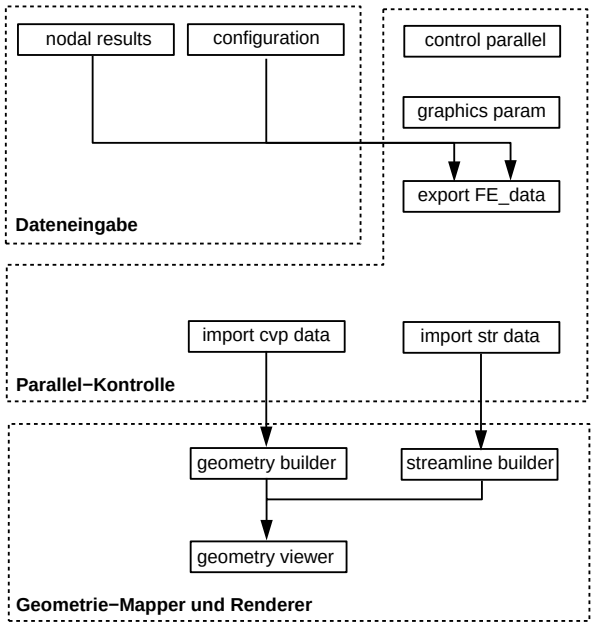


Abb. 6.13: Modulnetzwerk für eine Visualisierungsanwendung mit Parallelrechnerunterstützung

Es folgt eine Beschreibung des interaktiven Ablaufs dieser Anwendung mittels der beteiligten Moduln. Das in Abbildung 6.13 dargestellte Modulnetzwerk unter AVS repräsentiert lediglich die für den Anwender sichtbare Seite; Parallelrechner und Transmitter bleiben ihm verborgen, weshalb zum besseren Verständnis in Abbildung 6.14 schematisch der Gesamtablauf der Anwendung einschließlich ihrer Variationen skizziert ist.
Die Anwendung beginnt mit dem Einlesen der sequentiell vorliegenden Ergebnisdaten aus der FE–Analyse durch die beiden Lesemoduln *nodal result* und *configuration*, die ihre Datenfelder direkt dem Modul *export FE-data* zur Verfügung stellen. Der Parallelrechner wird interaktiv aus dem Modul *control parallel* heraus auf Basis der ausgewählten Parameter (Link, Anzahl der Prozessoren in x und y) gestartet. Parallelrechner, die nicht einem lokalen Netzwerk angehören, müssen hingegen manuell initialisiert werden.
Damit dem Modul *export FE-data* bekannt ist, auf wieviele Prozessoren die Ursprungsdaten aufgeteilt und verschickt werden sollen, legt das Modul *control parallel* den Datensatz *.parctrl* an, der alle wichtigen Kontrollvariablen enthält. Derselbe Datensatz hindert auch ein Modul daran, sich an einen Transmitter zu binden, wenn der Benutzer irrtümlich eine Verbindung anforderte, bzw. die Reihenfolge in der Anwendung mißachtet hat.

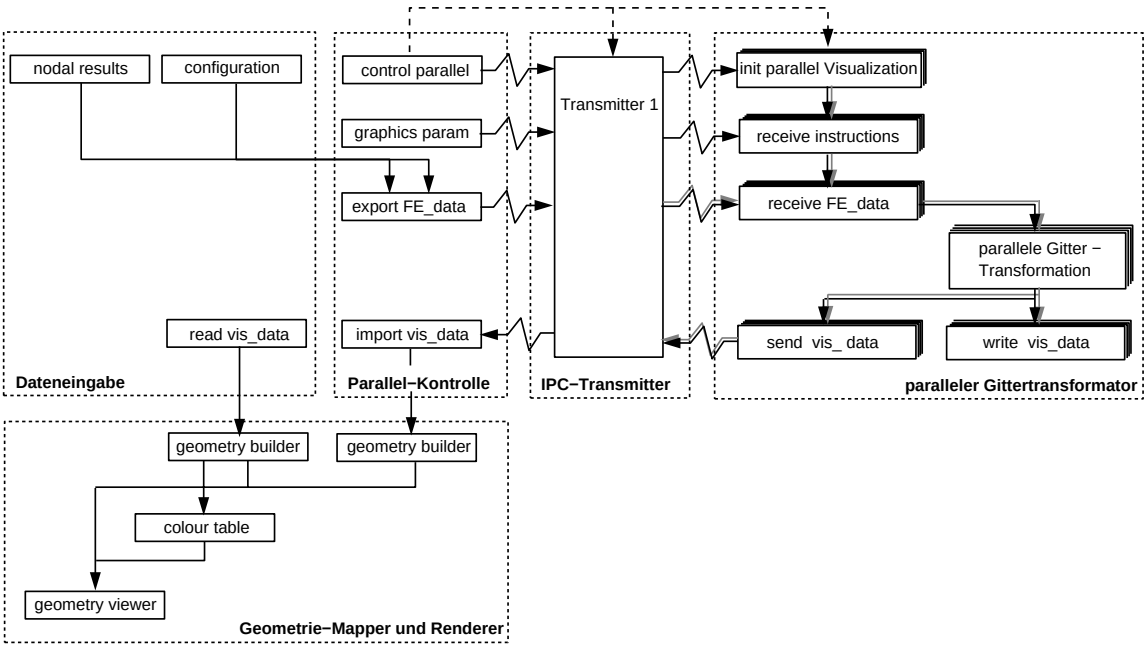


Abb. 6.14: Modulare Vernetzung für eine Visualisierungsanwendung mit Parallelrechnerunterstützung

Nachdem durch das Modul *control parallel* Parallelrechner und Transmitter initialisiert wurden, wird dem Parallelprogramm über das Widget *start computation* der

Auftrag übergeben. Im Auftrag wurde u.a. spezifiziert, an welchen Stellen das Parallelprogramm Empfangsbereitschaft für externe Daten signalisiert, bzw. von welchen Moduln Daten übertragen werden (Abb. 6.15 links).
In dieser Anwendung wurden die Widgets *data graphics, data export* und *data reimport* aktiviert; die Reihe der vom Benutzer auszuführenden Moduln liegt somit fest. Im folgenden Modul *graphics parameters* werden alle relevanten Parameter für die Berechnung des Visualisierungsgitters gesetzt, wie es aus Abbildung 6.15 ersichtlich ist. Jetzt erst werden mit dem Modul *export FE-data* die eigentlichen, die Berechnungsgrundlage bildenden Daten transferiert, nämlich die des FE–Problems. Da die Berechnungsalgorithmen überwiegend elementbasierend sind, folgt hier eine Aufteilung auf die Prozessoren nach Elementen. Jeder Prozessor erhält also in etwa dieselbe Anzahl Elemente zugewiesen.

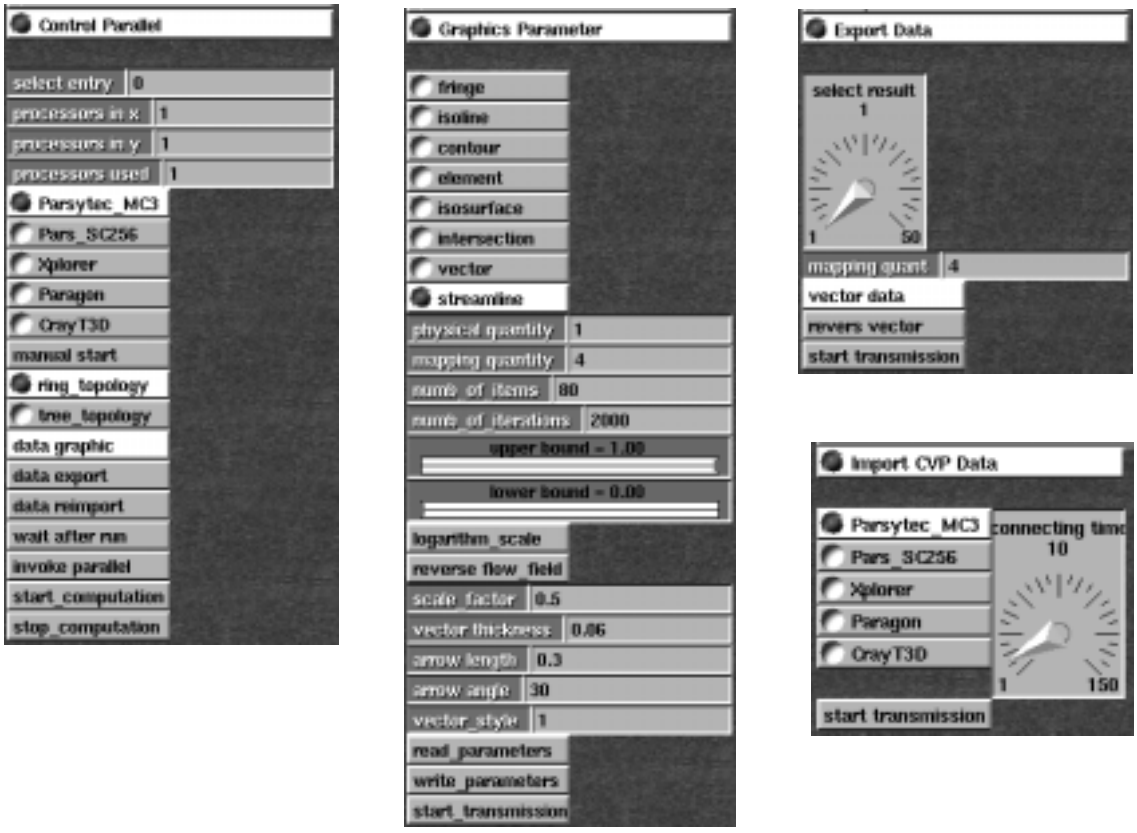


Abb. 6.15: Modulparameter einiger Steuermoduln

Ist jedoch eine Stromlinienberechnung vorgesehen, wird das gesamte FE–Problem an den Master–Prozessor verschickt, der die Daten den übrigen Prozessoren bekanntmacht (*broadcast*). Die Parallelität besteht in diesem Fall in der Aufteilung der Partikel bzw. der Startpunkte. Von Nachteil ist, daß bei dieser Anwendung die Speicherkapazität eines Parallelprozessors sehr rasch erschöpft ist.

Das Modul *import cvp data* sorgt dafür, daß mit Beendigung der parallelen Berechnung die Visualisierungsgitterdaten prozessorweise zurückübertragen und in sequentielle Felder integriert werden. Zu diesem Zweck fordert der Benutzer interaktiv die Daten an, wobei er eine Zeit vorgibt *(connecting time)*, innerhalb der versucht werden soll, eine Verbindung herzustellen.
Um beispielsweise Zeitverluste in einer Simulation zu minimieren, sucht die parallele Seite ihrerseits nur für eine sehr kurze Zeit eine Transmitterverbindung und reagiert bei Fehlschlägen entsprechend der Voreinstellung. Da für verschiedene Parallelrechner verschiedenartige Transmitter aufgebaut werden, wählt der Benutzer hier noch einmal seine Maschine aus. Empfangene Ergebnisse schließlich können mit den geeigneten Geometrie–Mapper und Rendermoduln interaktiv weiterverarbeitet werden (Abb. 6.13). Zur Veranschaulichung des Datenflusses und der interaktiven Einflußnahme ist in Abbildung 6.16 die Steuerung des Datenaustauschs zwischen AVS, Transmitter und Parallelrechner skizziert.

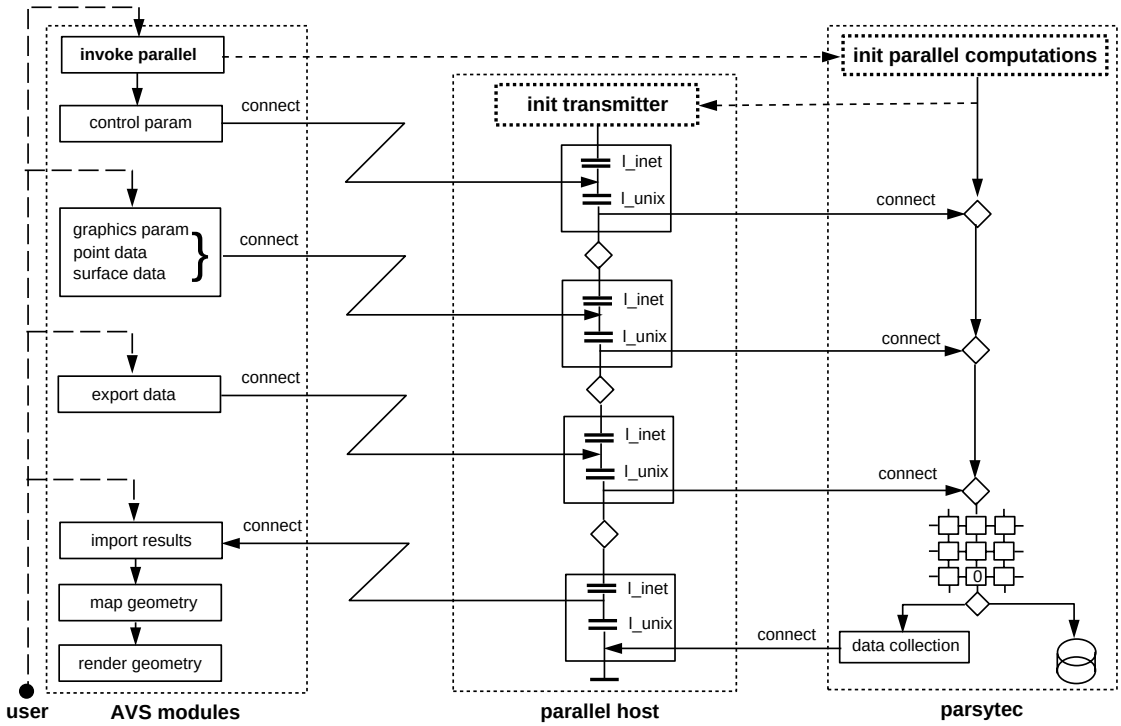


Abb. 6.16: Steuerung des Datenaustauschs zwischen AVS und Parallelrechner

6.5.2 Visualisierung verteilt vorliegender Datensätze

Dieser Fall gleicht dem soeben behandelten bis auf die Art der Bereitstellung der Daten. Auf einer fernen Maschine verteilt vorliegende FE–Ergebnisdaten werden von Parallelprozessoren eingelesen, fiur die Visualisierung aufbereitet, die Ergebnisdaten zum Geometrie–Mapping und Rendering an eine Graphik–Workstation übermittelt. Abbildung 6.17 zeigt die zu dieser Anwendung gehörigen Steuermoduln.

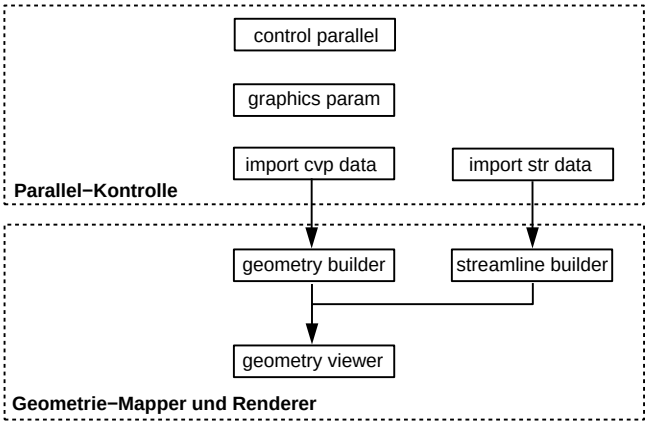


Abb. 6.17: Modulnetzwerk für eine Visualisierung verteilt vorliegender Datensätze

6.5.3 Visualisierung während des Ablaufs der FE–Analyse

sequentielle Berechnung des Visualisierungsgitters:
Bei der sequentiellen Berechnung werden von der Parallelmaschine Analyse–Ergebnisdaten transferiert, d.h. die eigentliche Gittertransformation findet sequentiell auf der Graphik–WS statt. Das zugehörige AVS-Modulnetzwerk zeigt Abbildung 6.18 links. Knotenergebnisse und Konfiguration können von dem Modul *recv patran* empfangen werden. Ist das FE–Netz jedoch nicht adaptiv, empfiehlt es sich, dieses einmalig mit dem Modul *configuration* einzulesen und lediglich über das Modul *recv nodal results* die Knotenergebnisse zu aktualisieren. Das Modul *data transformer* berechnet seriell das Visualisierungsgitter abhängig vom ausgewählten Algorithmus und gesetzten Parametern. Die Erzeugung der Abstrakten Visuellen Objekte und ihre Darstellung schließt sich daran an.

parallele Berechnung des Visualisierungsgitters:
Bei der parallelen Berechnung hingegen nimmt der Parallelrechner auf den Prozessoren der FE–Analyse die Gittertransformation vor und transferiert daran anschließend das Visualisierungsgitter auf eine Graphik–WS. Das zugehörige AVS–

Modulnetzwerk zeigt Abbildung 6.18 rechts. Der Import des parallel berechneten Visualisierungsgitters wird je nach Datenformat (*STR* für Stromliniendaten oder *CVP* für alles andere) durch die entsprechenden Datenimportmoduln vorgenommen.

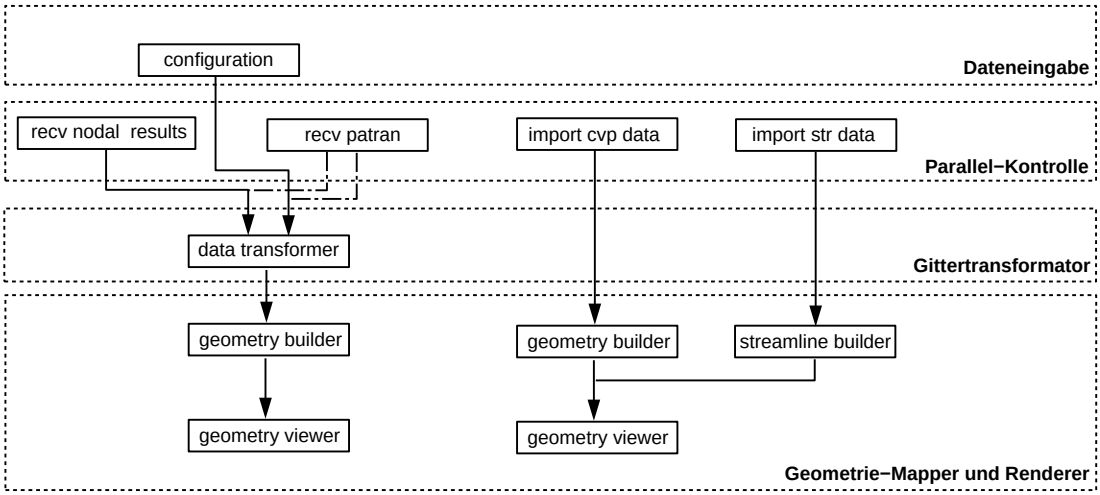


Abb. 6.18: Modulnetzwerk für eine integrierte Simulation mit sequentieller (links) und paralleler Berechnung des Visualisierungsgitters

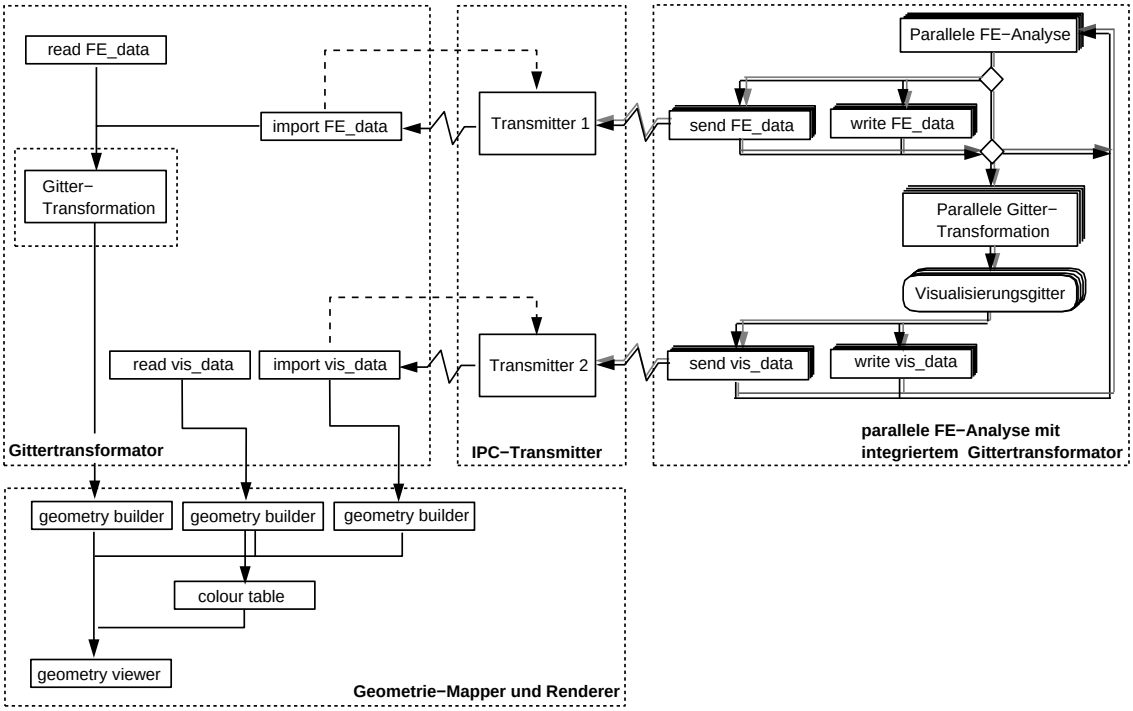


Abb. 6.19: Modulare Vernetzung für eine integrierte parallele Anwendung

In Abbildung 6.19 ist zur Veranschaulichung wiederum der Gesamtablauf dieser Anwendung für beide Variationen dargelegt, einschließlich aller Komponenten, die für den Benutzer unsichtbar bleiben.
Der Vollständigkeit halber sei bemerkt, daß diese Art der integrierten Simulation selbstverständlich auch auf Einprozessorsysteme, d.h. rein sequentielle Anwendungen auf herkömmlichen Vektorrechnern, übertragbar ist. Konkrete Rechenbeispiele gerade für parallele Anwendungen werden inklusive einer Bewertung der Rechenzeiten im nächsten Kapitel vorgestellt.

Kapitel 7

Anwendungen und Ergebnisse

In diesem Kapitel werden sowohl serielle als auch parallele Anwendungen vorgestellt. Soweit es möglich ist, werden zum direkten Vergleich neben parallelen Rechenzeiten auch die einer rein seriellen Rechnung herangezogen. Um das System für den Benutzer in groben Zügen zu dokumentieren, wird zu jeder Anwendung das zugehörige Modulnetzwerk präsentiert und einige der interaktiven Parameter erläutert. Jede Anwendung, ob seriell oder parallel, folgt dabei stets demselben Grobschema, das bereits in Abbildung 1.1 erläutert wurde.
Es sei an dieser Stelle nochmals ausdrücklich darauf hingewiesen, daß alle in dieser Arbeit aufgeführten und präsentierten Moduln neu entwickelt wurden, bis auf das AVS–eigene Renderer–Modul *geometry viewer*. Lediglich die Netzwerkstruktur, der grundlegende modulare Aufbau sowie die Graphikbibliotheksroutinen für die Erzeugung Abstrakter Visueller Objekte wurde dem AVS entlehnt, um die Software optimal an die Gegebenheiten im AVS anzubinden. Prinzipiell ist es ohne größeren Aufwand möglich, die interaktiven Eingaben durch eine Kommunikationsoberfläche basierend auf OSF_Motif zu ersetzen und die Erzeugung von AVOs durch eine andere Graphikbibliothek wie PHIGS [25] oder GL [64] vorzunehmen.

7.1 Serielle Anwendungen

Für die Mehrheit der Benutzer sind serielle Anwendungen ausreichend. Für besonders aufwendige Aufgaben jedoch, wie z.B. die Berechnung hunderter Stromlinien, kann, wie in Abschnitt 7.2 gezeigt wird, durch Einbringen weniger Moduln in das bestehende serielle Modulnetzwerk ein Parallelrechner hinzugezogen werden. Im Folgenden werden die wesentlichen Modulnetzwerke und einige ihrer Variationen vorgestellt.

7.1.1 Anwendungen mit Schnittflächen

Die erste Anwendung, deren Netzwerk in Abbildung 7.1 zu sehen ist, erzeugt aus einem dreidimensionalen Problem beliebige Schnittebenen, auf denen Isolinien, Konturflächen oder auch Vektoren einer ausgewählten Größe abgebildet werden können. Für die wichtigsten Moduln sind in den Abbildungen 7.2 bis 7.4 die interaktiven Parameter (*widgets*) dargestellt.

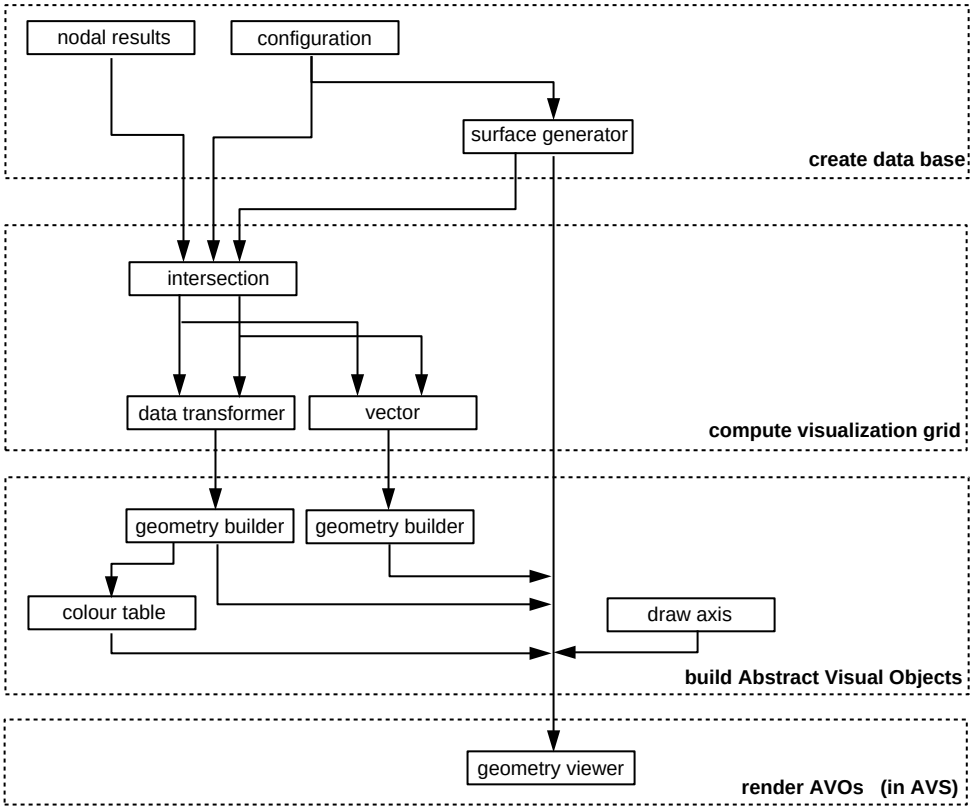


Abb. 7.1: Serielle Modulnetzwerke 1

Die einzelnen Schritte im Netzwerk werden von oben nach unten durchlaufen; die Verbindungslinien zeigen den Fluß der Daten an.
Die Anwendung beginnt mit dem Einlesen der Knotenergebnisdaten und des Finite–Elemente-Netzes im Patran–Neutral–Format [19] durch die Moduln *nodal results* und *configuration*. Mittels eines Browsers wird die gewünschte Datei ausgewählt. Bei den Gitterdaten können entweder alle Elemente eingelesen werden oder wahlweise nur zweidimensionale oder dreidimensionale. Beim Einlesen der Knotendaten gibt es ebenfalls Wahlmöglichkeiten. Hier kann für jeden Knoten entweder der gesamte Ergebnisvektor eingelesen werden oder im Sinne einer Datenreduktion ausgewählte Spalten: ein Skalar, die drei Komponenten eines Vektors, oder Skalar und Vektor. Fernerhin kann über eine Kopfdatei (*read_header*) eine Serie von Dateien verarbeitet werden, beispielsweise für instationäre Simulationen wie Bahnlinienberechnungen oder Animationsfolgen. Sollen keine Ergebnisse eingelesen werden, aber dennoch Moduln benutzt werden, die beide Dateneingänge benötigen (z.B. der *data transformer*), kann für eine ausgewählte Zahl von Knoten ein Dummy–Feld mit Nullwertergebnissen erzeugt werden (Abb. 7.2).

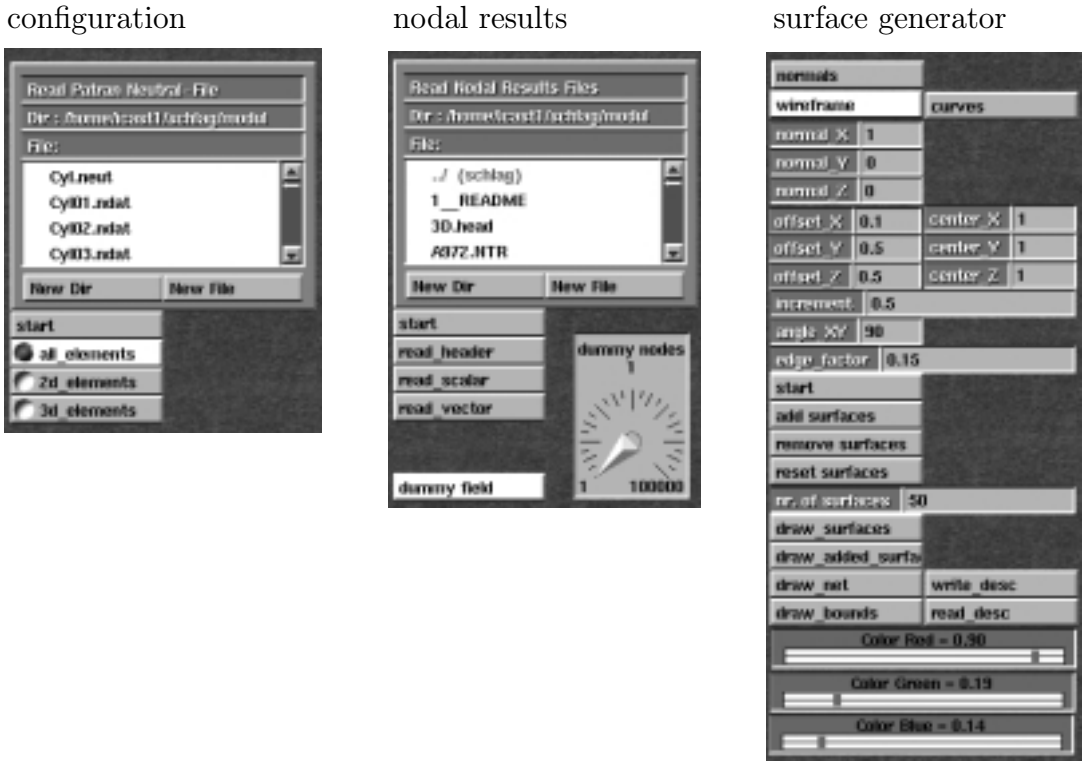


Abb. 7.2: Menüs für den Aufbau der Datenbasis

Das Modul *surface generator* ist ein Ebenengenerator, mit dem sich beliebige einzelne Schnittflächen bzw. Serien von Schnittflächen erzeugen lassen. Das Modul *intersection* berechnet auf der Datenbasis die eigentlichen Schnitte. Für den Ebe-

nengenerator gibt es eine Reihe von Optionen. Die drei Komponenten des Aufpunktes (*offset*) sowie die Komponenten der Normalen legen die erste Referenzebene fest. Ferner wird die Anzahl der zu erzeugenden Ebenen und das zugehörige Inkrement in Normalenrichtung vorgegeben, das den Abstand der Ebenen voneinander bestimmt. Mit dem Faktor (*edge_factor*) kann die Kantenlänge der Referenzebenen für die Darstellung skaliert werden.
Mit *curves* wird anstelle paralleler Ebenen die Generierung einer Serie gedrehter Schnittebenen um ein gewähltes Drehzentrum (*center*) und innerhalb eines Drehwinkel (*angle_xy*) vorgenommen (vergleiche hierzu Abschnitt 4.3.6). Daneben ist auch die Generierung einzelner Ebenen möglich sowie die Kombination aller Generierungsarten miteinander. Die Option *add surfaces* fügt aktuell generierte Ebenen einer internen Liste hinzu; *remove surface* streicht das zuletzt generierte von der Liste, während durch *reset surface* die gesamte interne Liste neuinitialisiert wird. Interne Listen können als Ebenenbeschreibungen auf Platte gespeichert werden *write_desc*, bereits gespeicherte zu aktuellen Listen jederzeit dazugenommen werden *read_desc*. Mit dem Knopf *start* wird eine Ebenenbeschreibung schließlich als eindimensionales Feld an den Outputport gebracht, womit die Daten dem Modul *intersection* verfügbar werden.
In vielen Moduln zeichnet *draw net* alle Elemente des Gitters in einer ausgewählten Farbe, Während *draw bounds* alle vorhandenen Randelemente zeichnet, d.h. 2D–Elemente in dreidimensionalen sowie 1D–Elemente in zweidimensionalen Problemen.

Im Modul *intersection* werden entweder Schnitte (*cut*), Restgeometrien (*body*) oder beides (*body+cut*) berechnet. Eine auf den Geometrien darzustellende Ergebnisgröße wird mit *select result* ausgewählt, ferner, ob eine Restgeometrie einfarbig erscheinen soll (*uncoloured body*) und, ob sie nur zweidimensionale Elemente enthalten soll (*2D body*). Für manche Anwendungen ist nur die Außenlinie eines Schnitts gefragt (*intersect 2D*). *start* initialisiert wiederum die Berechnungen; ebenso verfährt *animate planes* mit dem Unterschied, daß hier für Animationszwecke jede Ebene bzw. Restgeometrie einem gesondertem Objekt zugeordnet wird (Abb. 7.3).

Die berechneten Schnittdaten können direkt mit dem Geometrie–Mapping–Modul *geometry builder* zu einem oder mehreren AVOs verarbeitet werden, um mit dem AVS–Modul *geometry viewer* zuletzt am Bildschirm dargestellt zu werden.
Die mit dem *geometry builder* erzeugten AVOs können entweder einfarbig (*monochrome*) erscheinen oder eine Variation einer Ergebnisgröße zeigen, ferner als Drahtmodell (*wireframe*) generiert werden oder schattiert, ohne oder mit Normaleninformation (*normals*) sowie mit inversen Normalen (*flip normals*). Liegt für ein Objekt ein Ergebnisvektor vor, kann mit *select results* ausgewählt werden.

Objekte können im AVS–Format *geometry* erzeugt und auf Plattenspeicher abgelegt werden. Jedes Objekt erhält einen Namen, der sich von dem Algorithmus ableitet, der für die Berechnung der Objektdaten sorgte, und dem Modul codiert übermittelt wurde. Als Suffix erhält der Objektnamen die zweiziffrige Nummer eines Zählers, der anfänglich auf 01 steht. *Store graphics* setzt den Zähler um eins hoch, damit ein zweites Objekt selben Namens das Vorhergehende nicht überschreibt.

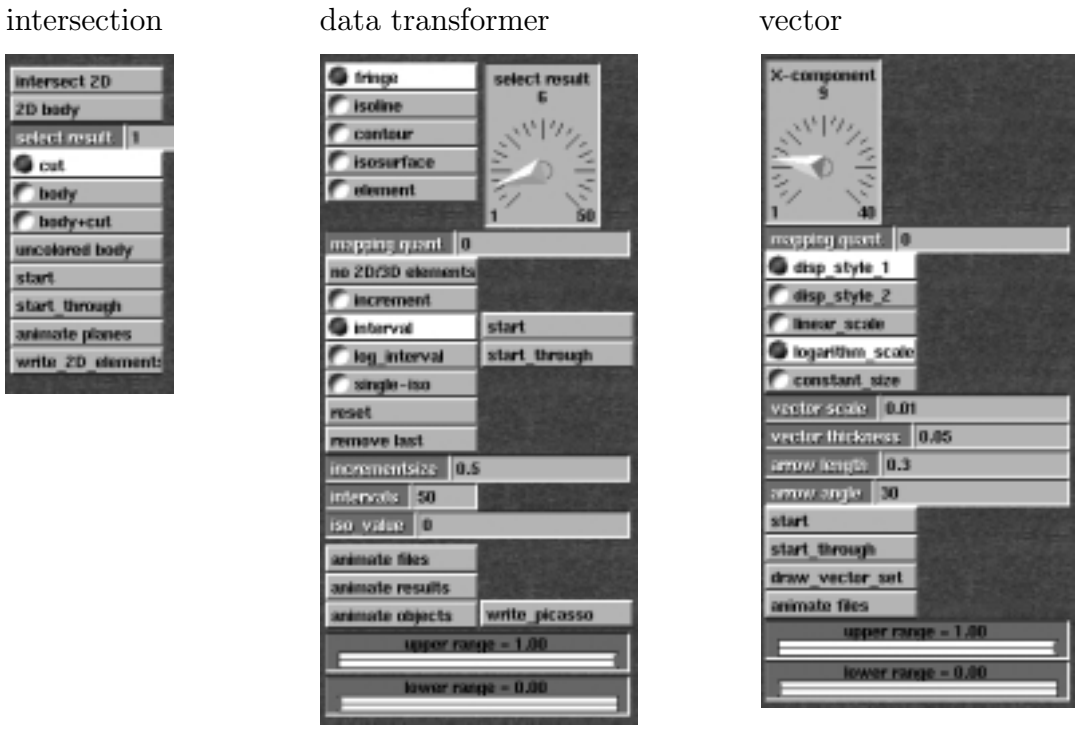


Abb. 7.3: Menüs für die die Berechnung verschiedener Visualisierungsgitter

Es stehen 14 verschiedene Farbmodelle (*colour model*) zur Verfügung, darunter das Sonnenspektrum (0) und Schwarz–Weiß (1). *Upper range* und *lower range* schränken den Wertebereich ein. Ergebniswerte außerhalb des Bereichs erscheinen in der Farbe, die mit den RGB–Schiebern eingestellt wurde.
Start löst den Prozeß der graphischen Objektbildung aus. Generiert wird dasjenige Objekt, das mit *select object* ausgewählt wurde. Liegen Daten für mehrere Objekte vor, kann durch *select object = 0* ein Gesamtobjekt erstellt werden, das alle Objekte umfaßt. Wird anstelle von *start* hingegen *write animation* ausgelöst, wird für jede Objektnummer ein eigenes Objekt erstellt und in einen Animationszyklus eingebunden, der im Hauptmenü *AVS–Geometry Viewer* als bewegte Bildfolge betrachtet werden kann (Abb. 7.4).

Als Alternative zur direkten Verarbeitung der Schnittdaten bietet es sich an, diese zuvor mit dem Modul *data transformer* zu behandeln. Der *data transformer* be-

rechnet auf den ebenen Schnittdaten einen Fringeplot, Isolinien oder Konturflächen eines ausgewählten Ergebnisses. Isolinien wie Konturflächen können sowohl einzeln unter Angabe eines diskreten Wertes generiert werden (*single_iso*), als auch über die Vorgabe eines Inkrements oder einer Anzahl zu generierender Werte in einem Intervall. Alle drei Methoden sind kombinierbar. *Upper range* und *lower range* legen die Grenzen eines Intervalls fest. *Lower range* dient zudem als Startwert für die Inkrementierung.

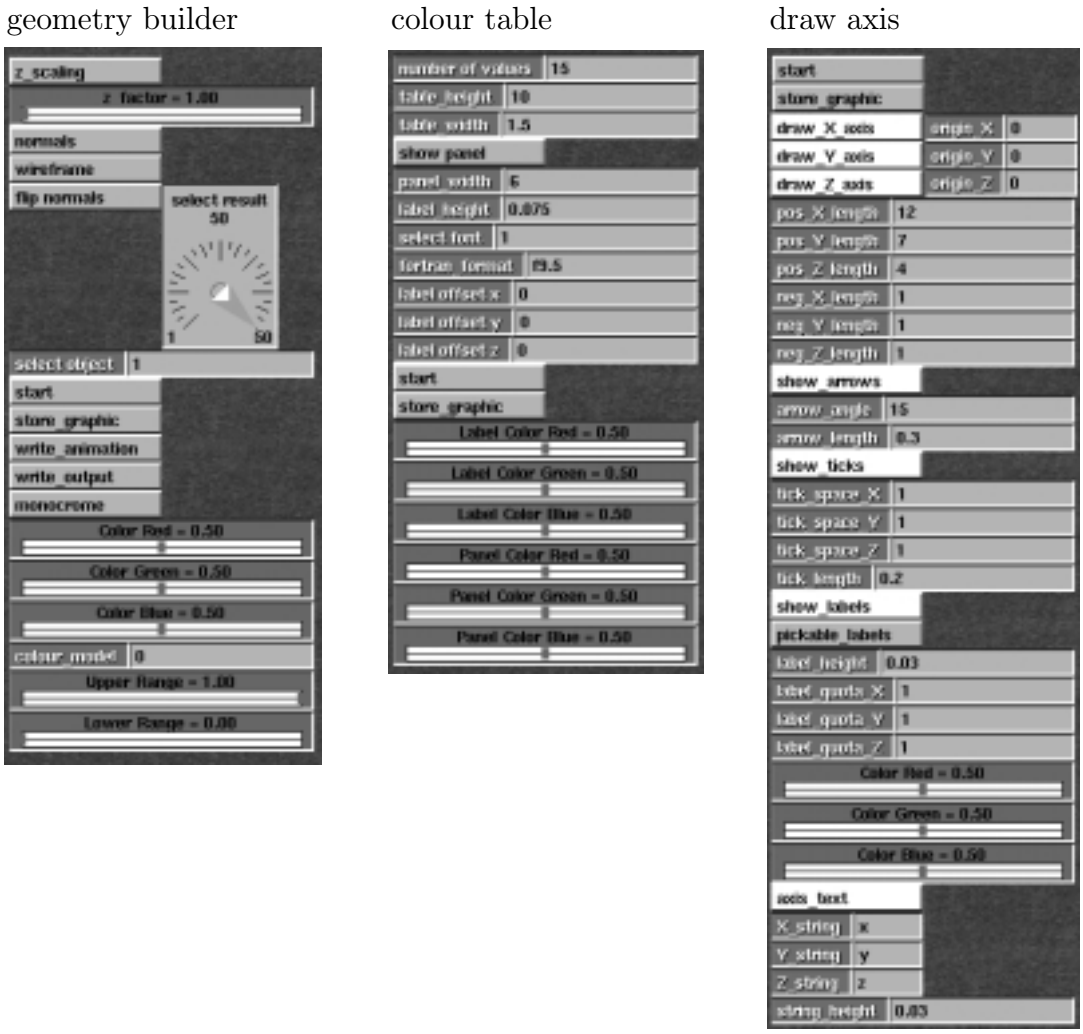


Abb. 7.4: Menüs für die Erzeugung verschiedener Abstrakter Visueller Objekte

Das Modul *vector* generiert auf den Schnittflächen Vektoren. Vektoren können auf unterschiedliche Weise dargestellt werden (*disp_style*), wie aus Abbildung 4.32 in Kapitel 4 zu ersehen ist. Neben einer linearen und logarithmischen Skalierung können Vektoren auch in konstanter Länge dargestellt werden.
Zusätzlich ins Netzwerk eingebracht wurden die beiden Moduln *colour table* und

draw axis, die individuelle Farbwertetabellen erzeugen bzw. beschriftete Koordinatenachsen generieren. Die Widgets dieser Moduln erklären sich selbst (Abb. 7.4).

7.1.2 Anwendungen mit Stromlinien

Die zweite Referenzanwendung, deren Netzwerk Abbildung 7.5 demonstriert, erzeugt zunächst eine Weg–Zeit–Beschreibung interaktiv in einem Strömungsgebiet losgelassener Partikeln. Ergebnisdaten einer Weg–Zeit–Beschreibung sind im Datenformat STR abgelegt und bilden die Grundlage für mehrere Anwendungen, wie Stromlinien, eine Verfolgung von Partikeln, Tropfen oder Flächensegmenten.

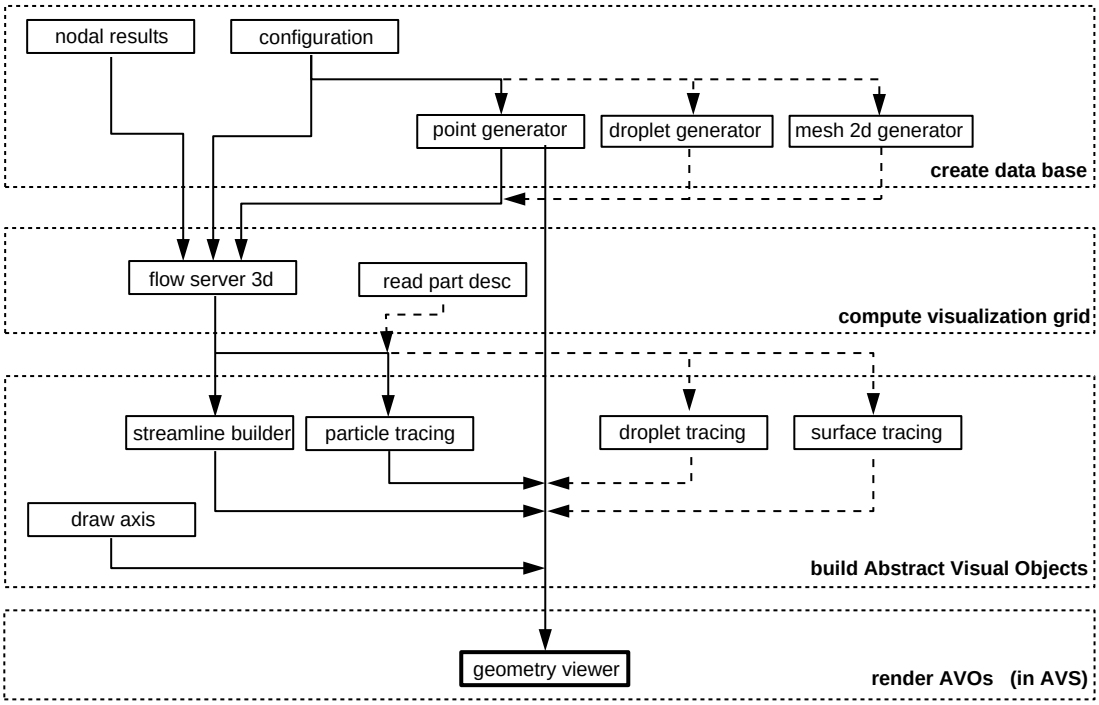


Abb. 7.5: Serielle Modulnetzwerke 2

Nachdem die Problemdaten (*nodal results* und *configuration*) eingelesen wurden, generiert der Benutzer zunächst interaktiv mit dem *point generator* Startpunkte. Punkte können einzeln erzeugt werden (*single_points*) oder in kartesischen Gebieten (*gen_points*), auf Linien, in Flächen gleichwie in Volumen. Generierungsmethoden sind beliebig miteinander kombinierbar.
X_point gibt die X–Komponente eines Einzelpunkts oder Aufpunktes an. *Delta_X* begrenzt den Raum vom Aufpunkt aus in positiver X–Richtung, wohingegen mit *nr. points in X* die Anzahl der zu generierenden Punkte in X–Richtung im Intervall Δ_x vorgegeben wird. Entsprechend wird in Y– und in Z–Richtung verfahren.

Alternativ zum *point generator* kann der *droplet generator* oder der *mesh 2d generator* eingesetzt werden, wenn Tropfen respektive Flächen verfolgt werden sollen. Beim Tropfengenerator wird zunächst das Zentrum bzw. der Mittelpunkt einer Kugel vorgegeben, der Tropfenradius sowie der Feinheitsgrad, in dem die Oberfläche approximiert wird. *Draw droplet* zeichnet in der ausgewählten Farbe (RGB–Schieber) den Tropfen, *draw cube* hingegen einen Würfel, der die Grundlage für die Generierung bildet (vgl. Kap. 4.3.10).

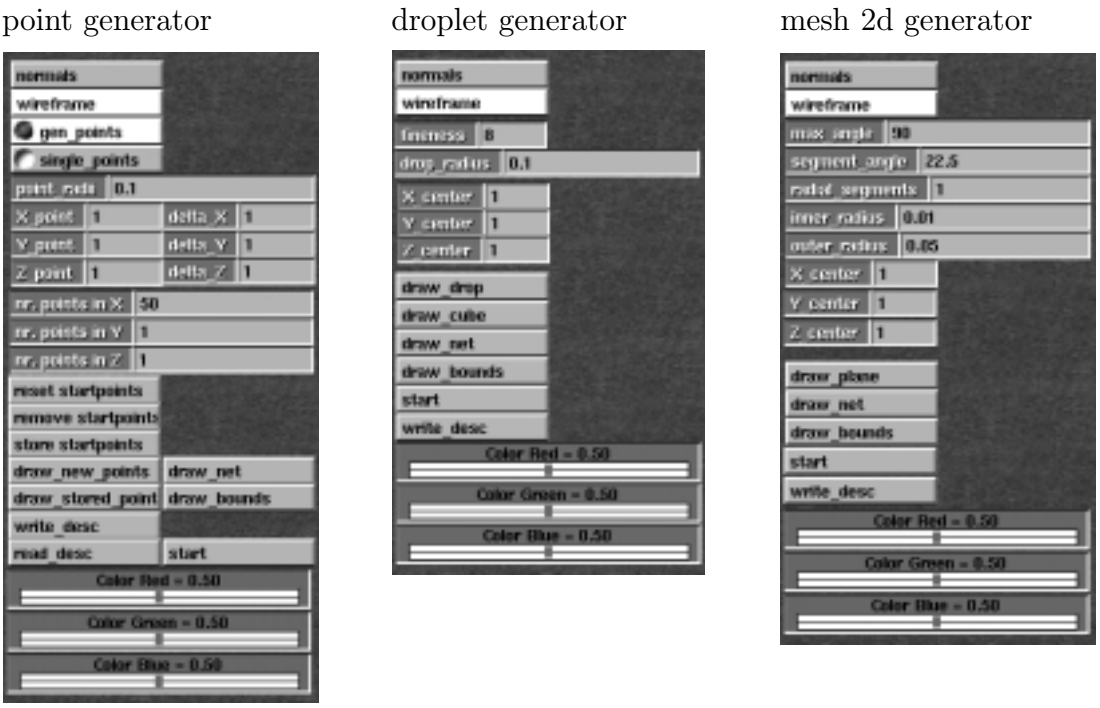


Abb. 7.6: Menüs für den Aufbau der Datenbasis

Bei der Generierung eines Oberflächensegments mit dem *mesh 2d generator* wird ebenfalls ein Zentrum vorgegeben, dazu der äußere und der innere Radius, der Gesamtsegmentwinkel (*max_angle*), der Winkel eines Teilsegments (*segment angle*) sowie die Anzahl der Segmente in radialer Richtung (*radial segments*).
Store startpoints speichert aktuell generierte Punkte in einer internen Liste; *remove startpoints* löscht die letzte Listeneingabe und *reset startpoints* die gesamte Liste.
Draw stored points zeichnet alle in der Liste gespeicherten Punkte in der ausgewählten Farbe, während *draw new points* die aktuellen Punkte in der Komplementärfarbe darstellt. Bei allen Generatormoduln löst der Startknopf die Berechnung und die Erzeugung des Ausgabefeldes aus, wohingegen mittels *write_desc* eine Punkteliste in einer Datei namens *point_desc.dat* abgelegt wird, die mit *read_desc* jederzeit wieder eingelesen werden kann (Abb. 7.8).

Im Berechnungsmodul *flow server 3d* wird für dreidimensionale Probleme eine Weg–Zeit–Beschreibung der produzierten Partikeln berechnet, welche die Datengrundlage für die Geometrie–Mapper–Moduln bildet, wie z.B *streamline builder* und *particle tracing*, in denen AVOs für das AVS–Modul *geometry viewer* erzeugt werden (Abb. 7.7).
Das Modul *flow server 3d* enthält folgende Widgets: Mit *X-Component* wird die X-Komponente des Vektorfeldes gewählt und mit (*shooting iterations* die maximale Anzahl der Iterationsschritte vorgegeben. *Course iteration* verändert die Bandbreiten des Sakalarprodukts, womit weniger exakt entlang einer Stromlinie integriert, die Berechnung jedoch beschleunigt wird. In Vektorfeldern ohne extreme Gradienten erweist sich die Grobiteration als hinreichend exakt. Mit *include vorticity* wird für die Darstellung von Strombändern zu jedem Stromlinienpunkt die Richtung der örtlichen Zirkulation berechnet. *Reverse flowfield* kehrt das Vektorfeld um. Somit kann von einem gewählten Punkt aus eine Stromlinie in beiden Richtungen verfolgt werden.

flow server 3d

streamline builder

particle tracing

Abb. 7.7: Menüs für Berechnung und Objektbildung

Liegen instationäre Daten in Form mehrerer eingelesener Ergebnisdateien vor (*transient problem*), wird mit *trans_time_inc* ein Zeitinkrement gewählt, welches die Abfolge der Datensätze festlegt. Da die Datensätze Momentaufnahmen aller Ortsvek-

toren zu einem bestimmten Zeitpunkt darstellen, ergibt sich ergo die zeitliche Änderung eines Vektorfeldes durch die inkrementweise Abfolge der Datensätze bzw. der Zustände. Während also, anders ausgedrückt, ein Teilchen im Laufe der Zeit durch das Strömungsgebiet wandert, ändern sich die Ortsvektoren, die aus den zu der Absolutzeit gehörigen Datensätzen zu interpolieren sind.

Das Modul *streamline builder* erzeugt auf Basis der vom Modul *flow server 3d* (bzw. *flow server 2d für ebene Probleme*) berechneten Daten ein AVO, das Stromlinien repräsentiert. Mit *select result* wird eine skalare Größe auf den Stromlinien dargestellt, die aber auch einfarbig *monochrome* in einer ausgewählten Farbe erscheinen können.
Gerade wenn sie in Wirbeln umlaufen, müssen Stromlinien nicht in ihrer ganzen Länge gezeichnet werden. Hier wird nach der voreingestellten Länge (*max linepoints*) abgebrochen. *Upper range* und *lower range* geben einen unteren bzw. oberen Schwellenwert an, so daß Stromlinien mit Identifikationsnummern außerhalb dieses Intervalls nicht gezeichnet werden. *Stream_ribbon* sorgt für die Darstellung von Stromlinienbändern, deren Bandbreite je nach Größe der örtlichen Zirkulation variierend oder konstant (*constant_ribbon*) mit einstellbarer Breite (*ribbon_width*) dargestellt werden kann.

Das Modul *particle tracing* verfährt ähnlich wie der Stromlinienbauer und besitzt einige Widgets, die in ihren Funktionen identisch sind. Partikeln werden als Kugeln einstellbaren Durchmessers dargestellt (*particle size*). Mit *nr. of frames* gibt der Anwender die Zahl der Bilder für eine Animationsfolge und mit *time_limit* die maximale Zeit vor, so daß aus diesen beiden Angaben ein konstantes Zeitinkrement berechnet werden kann. Für jedes Bild wird der momentane Aufenthaltsort der Partikeln aus der Weg–Zeit–Beschreibung interpoliert.
Ein Partikel kann auch mit dem bisher zurückgelegten Weg dargestellt werden (*display spoor*), der dann als Linie oder als Band (*ribbon*), einfarbig oder mit einer aufprojizierten skalaren Größe (*mapping quantity*) erscheint. Es kann entweder der gesamte bisherige Weg gezeigt werden, oder das zuletzt zurückgelegte Wegstück bis zu einer Zeit, die sich aus der Momentanzeit minus der durch *spoor_length* eingestellten Zeit ergibt. *Start animation* schließlich löst die Berechnung des Animationszyklus nach den eingestellten Parametern aus.

Die Moduln *droplet tracing* und *surface tracing* sind mit dem Modul *particle tracing* verwandt. Zu jedem Zeitpunkt der Animationsfolge wird die triangulierte Kugeloberfläche bzw. das Flächensegment nach der ursprünglichen Generierungsvorschrift (siehe Generator-Moduln) erneut gezeichnet und als Bild in den Animationszyklus eingereiht.
Im Verlauf der Animation wandert jeder Punkt der betrachteten Fläche entlang ei-

ner Strom- bzw. Bahnlinie, woraus der Eindruck der Bewegung und der Verformung der Oberfläche resultiert.
Im Übrigen können alle drei Verfolgungs–Moduln auch Stromlinien darstellen, doch ist hier *draw streamlines* eher als eine Kontrollfunktion gedacht, mit der sich zum einen der Datensatz verifizieren läßt, und zum anderen Wahlparameter besser vorausgewählt werden können, z.B. die Einschränkung der Gesamtweglängen mit *max. linepoints*.

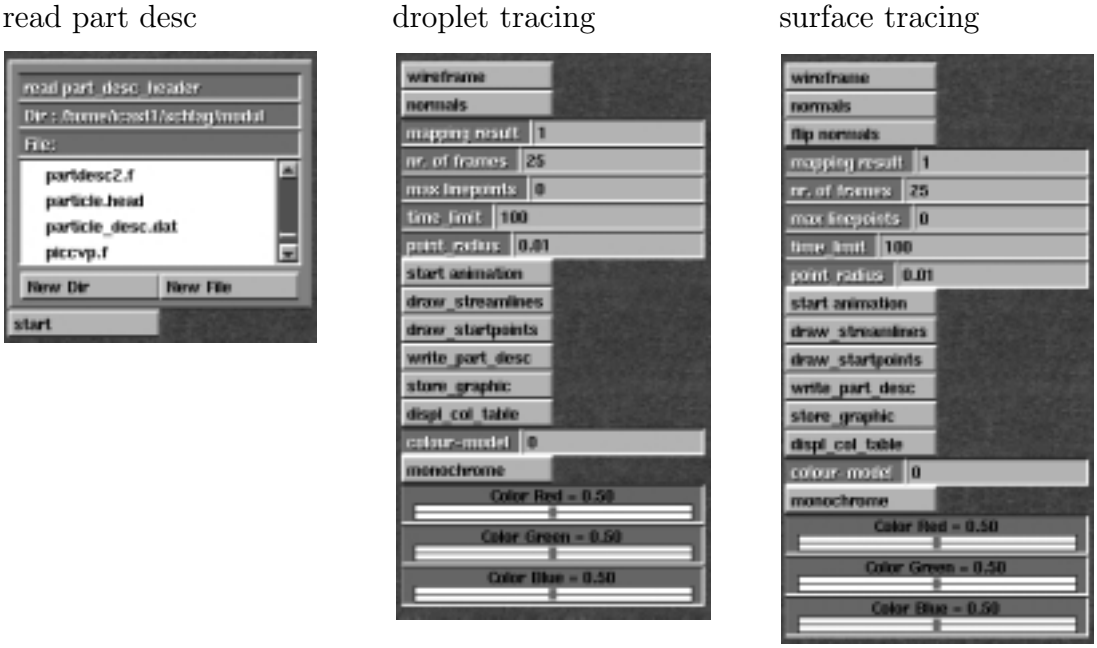


Abb. 7.8: Menüs für die Objektbildung

7.1.3 Anwendungen mit Volumenausschnitten

Mit der dritten seriellen Anwendung, deren Modulnetzwerk in Abbildung 7.9 skizziert ist, lassen sich in einem dreidimensionalen Problem Volumenausschnitte erzeugen, die entweder räumlich oder in ihren Umrissen flächig dargestellt werden, und darüber hinaus anderen Moduln als Datengrundlage dienen können, beispielsweise, wenn Isoflächen in einem ausgeschnittenen Volumen dargestellt werden sollen.
Liegt nur eine Hälfte eines symmetrischen Problems vor, kann mit dem Modul *mirror data* zunächst der Datensatz gespiegelt werden (Abb 7.10). Als Symmetrieebenen kommen hierzu alle drei kartesischen Grundebenen des Raumes (*plane_xy, plane_yz, plane_zx*) in Frage. Mit der Spiegelung wird der Datensatz grundsätzlich verdoppelt, es sei denn, es wären ausschließlich die Bilddaten verlangt (*omit initial data*). Im Zuge einer Spiegelung muß beachtet werden, daß, im Gegensatz zu skalaren Ergeb-

nisdaten, Vektorkomponenten ihr Vorzeichen ändern können. Dem wird Rechnung getragen, in dem Vektorkomponenten dem Programm kenntlich gemacht werden (*vector components, x_, y_, z_component*).
Die Option *reverse orientation* erzeugt eine Umkehrung des Elementorientierungssinns d.h. eine andere Knotennummerierung der Bildelemente, was sich bei der Generierung von Normalen bemerkbar macht.

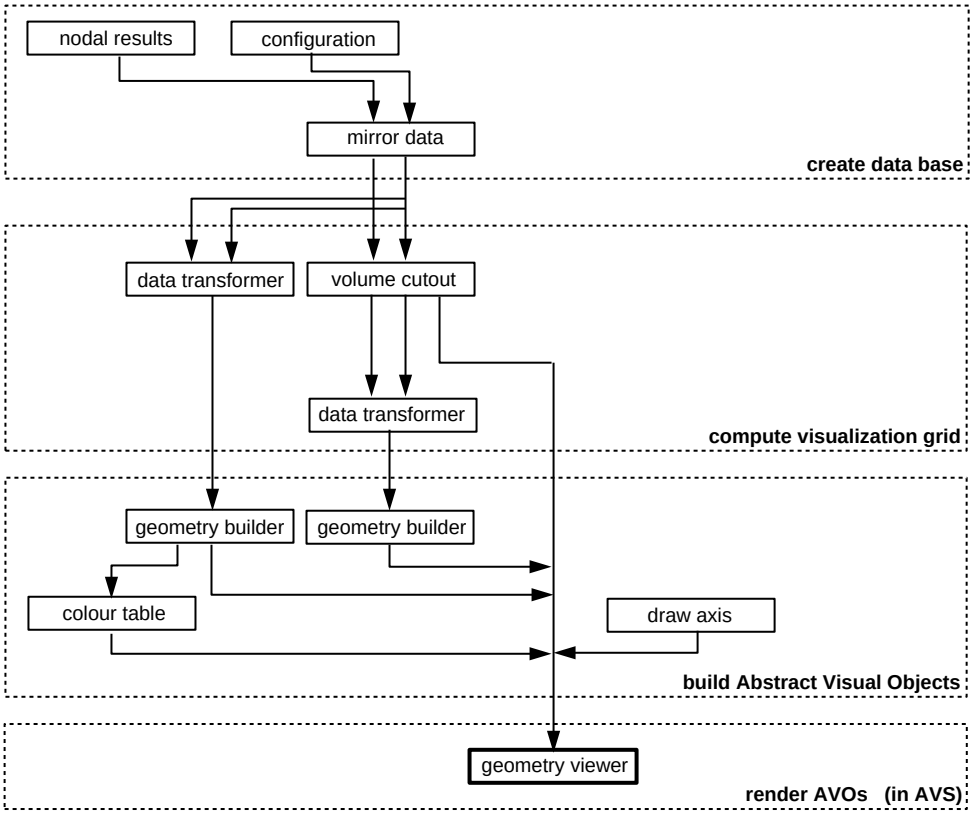


Abb. 7.9: Serielle Modulnetzwerke 3

Im Übrigen erzeugen die Moduln *data mirror* und *volume cutout* beide Ausgabefelder desselben Typs, die mit denen der Moduln *nodal results* und *configuration* korrespondieren und somit derselben Palette nachgeordneter Moduln zur Verfügung stehen.

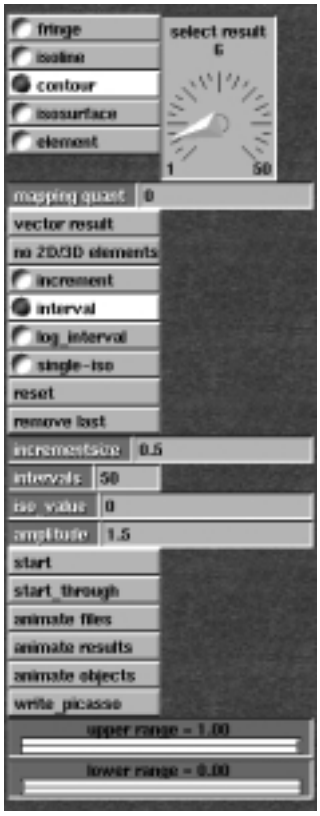
Mit dem Modul *volume cutout* wird zunächst interaktiv ein quaderförmiges Gebiet abgegrenzt, das ausgeschnitten werden soll. Dazu werden mit *x–min, x–max, y–min, y–max und z–min, z–max* die Gebietsgrenzen markiert. Dabei kann der Benutzer sich die Umrisse eines Ausschnitts direkt in Beziehung zu seinem Problem zeichnen lassen (*draw_c_space*) und die Eingaben beliebig modifizieren. Das Speichern der Rahmendaten eines Ausschnitts erfolgt mit *write control data*, das Lesen entsprechend mit *read control data*.

Ferner kann angegeben werden, ob für den Ausschnitt zweidimensionale Randelemente generiert werden sollen (*add_bounds*). Erst wenn der Startknopf betätigt wurde, wird die Berechnung der Elemente in dem Volumenausschnitt vorgenommen und entsprechend die Ausgabefelder erzeugt.

data mirror

nodal + displace

data displacer



volume cutout

Abb. 7.10: Menüs für Aufbau der Datenbasis und Gittertransformation

Visualisierung am Beispiel einer Bénard–Konvektion

Für die drei in den vorangehenden Abschnitten vorgestellten Anwendungen sollen nun verschiedene behandelte Techniken an Hand eines konkreten Rechenbeispiels demonstriert werden. Um einen unmittelbaren Vergleich zu haben, wurde allen drei Anwendungen derselbe Datensatz zugrundegelegt. Es handelt sich um die Ergebnisdaten aus einer FE–Berechnung einer Bénard–Konvektion, die mit den Kenngrößen $Pr = 7$ und $Ra = 19870$ charakterisiert ist [9].
Ein quaderförmiger, mit Wasser gefüllter Behälter mit einem Seitenverhältnis von 4:2:1 wurde mit 6312 Hexaedern diskretisiert. Anfänglich war nur die Unterseite des Behälters heiß; die übrigen Flächen wurden kalt gehalten bis auf die linke und rech-

te Seitenwand, auf der ein lineares Temperaturprofil zwischen beiden Extremwerten ($T_{max} = 273.65K, T_{min} = 273.15K$) gegeben war. Die Analyseergebnisse geben den sich einstellenden quasistationären Zustand wieder [55].

Gemäß der ersten beschriebenen Anwendung wurde eine Serie von Schnitten generiert und auf den Schnitten Konturflächen und Isolinien dargestellt. Zusätzlich wurde auf Schnitten in unmittelbarer Wandnähe Vektoren konstanter Größe abgebildet, um die Strömung zu verdeutlichen 7.11.

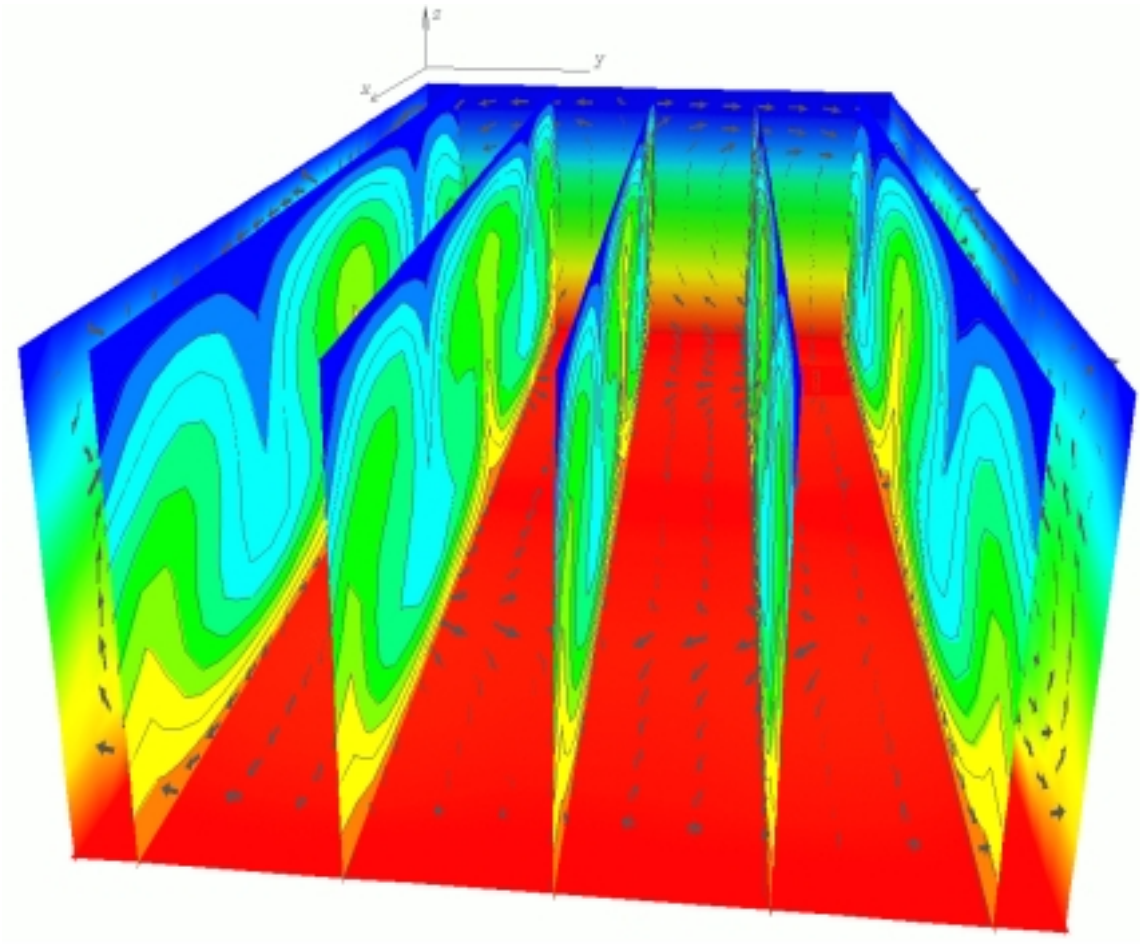


Abb. 7.11: Bénard–Konvektion: Serie von Schnittflächen mit Isowerten

Abbildung 7.12 bezieht sich auf die zweite Anwendung, in der eine, zwischen den beiden mittleren Wirbeln unterbrochene Ausgangsfläche im Vektorfeld losgelassen wird. Der Anfangsabstand der Fläche zu der heißen Bodenfläche beträgt zwei Prozent der Gesamthöhe des Behälters. Die ebene Fläche wird durch 5600 Gitterpunkte und 5584 Vierpunktselemente aufgespannt.
Jeder der 5600 Start- bzw. Gitterpunkte wandert nun entlang der durch sie führenden Stromlinie. Für die Berechnung von Stromlinien und der Verfolgung von Parti-

keln und Flächen wurden die 6312 Hexaeder vom Berechnungsalgorithmus in 25872 Tetraeder zerlegt. Die Berechnung der Weg–Zeit–Beschreibung der Startpunkte wurde nach 150 Iterationen abgebrochen, die Daten synchronisiert und 100 Bilder einer Animationsfolge berechnet, welche die Bewegung bzw. Verformung der Fläche im Strömungsfeld wiedergeben. Abbildung 7.12 zeigt den Zustand der Fläche nach 67 Sekunden. Auf der Fläche schattiert dargestellt sind Temperaturwerte. Durch diese Darstellungsweise werden die Wirbelröhren deutlich erkennbar (vgl. hierzu Abb. 7.19).

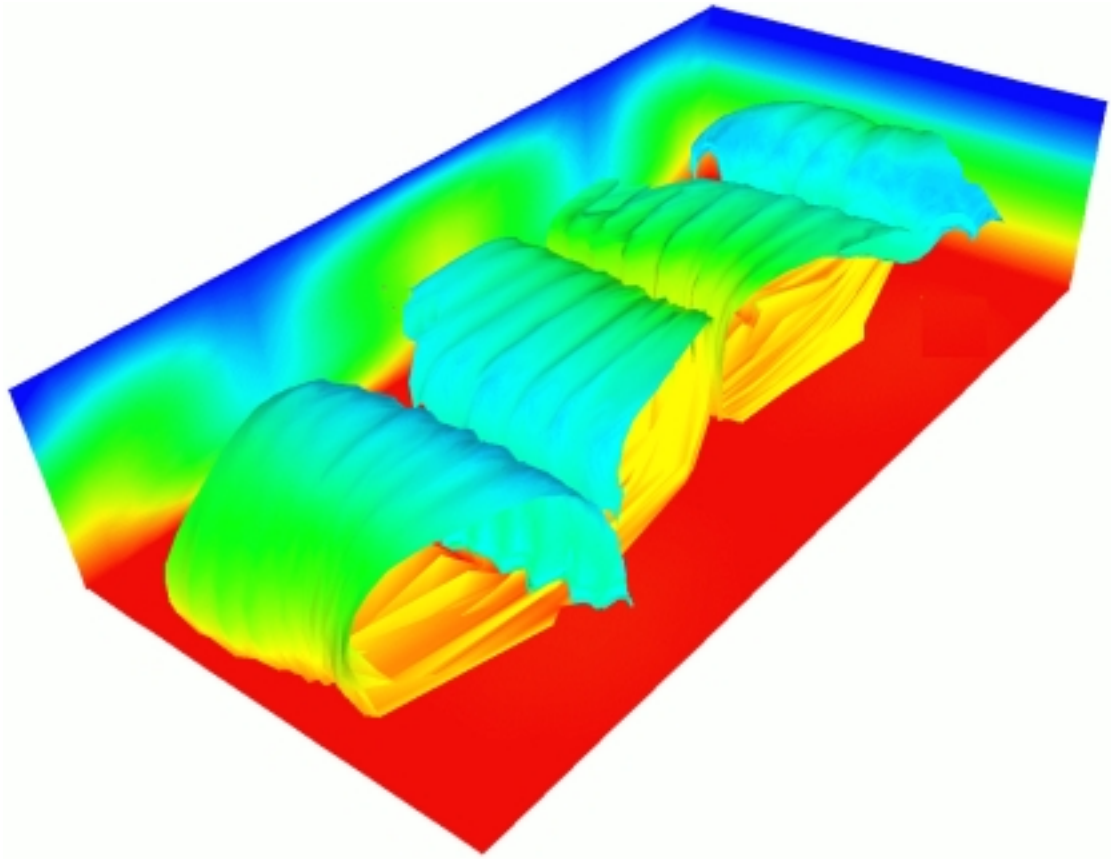


Abb. 7.12: Bénard–Konvektion: Zeitliche Verfolgung einer Fläche im Vektorfeld

Abbildung 7.13 bezieht sich auf die dritte Anwendung und zeigt vier Volumenausschnitte aus dem Konvektionsgebiet. Diese wurden so gewählt, daß jeder Ausschnitt innerhalb einer der vier Wirbelzonen liegt. Auf den Oberflächen der ausgeschnittenen Gebiete sind sowohl Temperaturwerte schattiert dargestellt als auch 25 Isolinien gleicher Temperatur. Dieses Bild verdeutlicht die Temperaturverteilung wie auch das folgende Bild (Abbildung 7.14), in dem sieben Isoflächen der Temperatur illustriert sind.

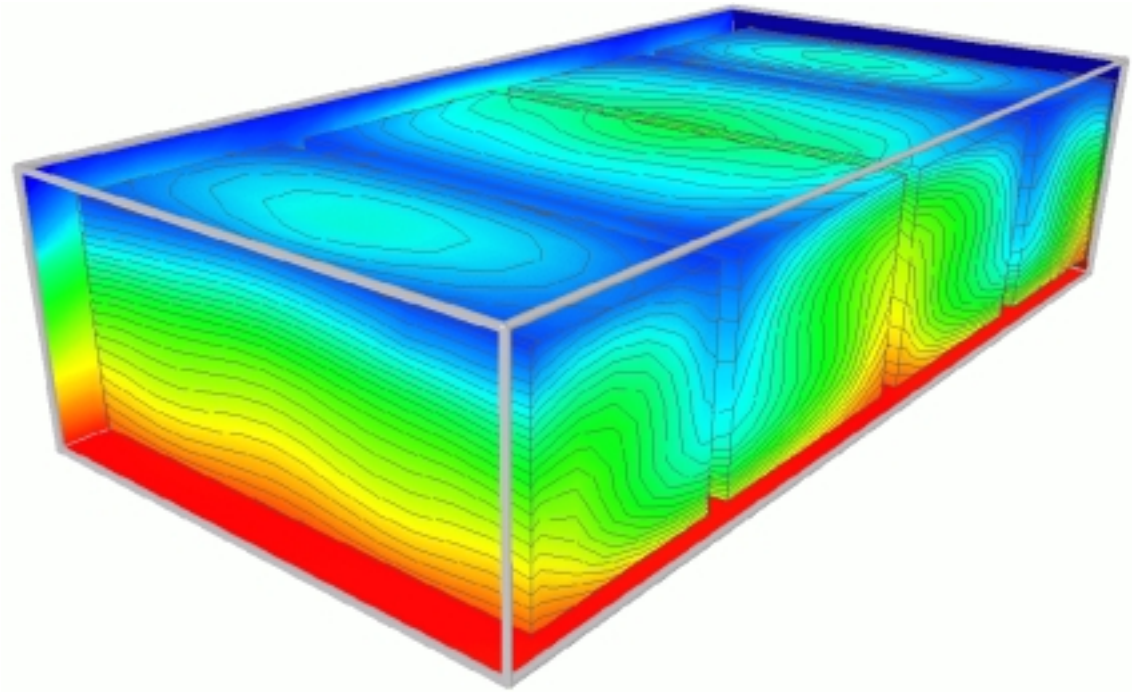


Abb. 7.13: Bénard–Konvektion: Vier Volumenausschnitte mit Oberflächendaten

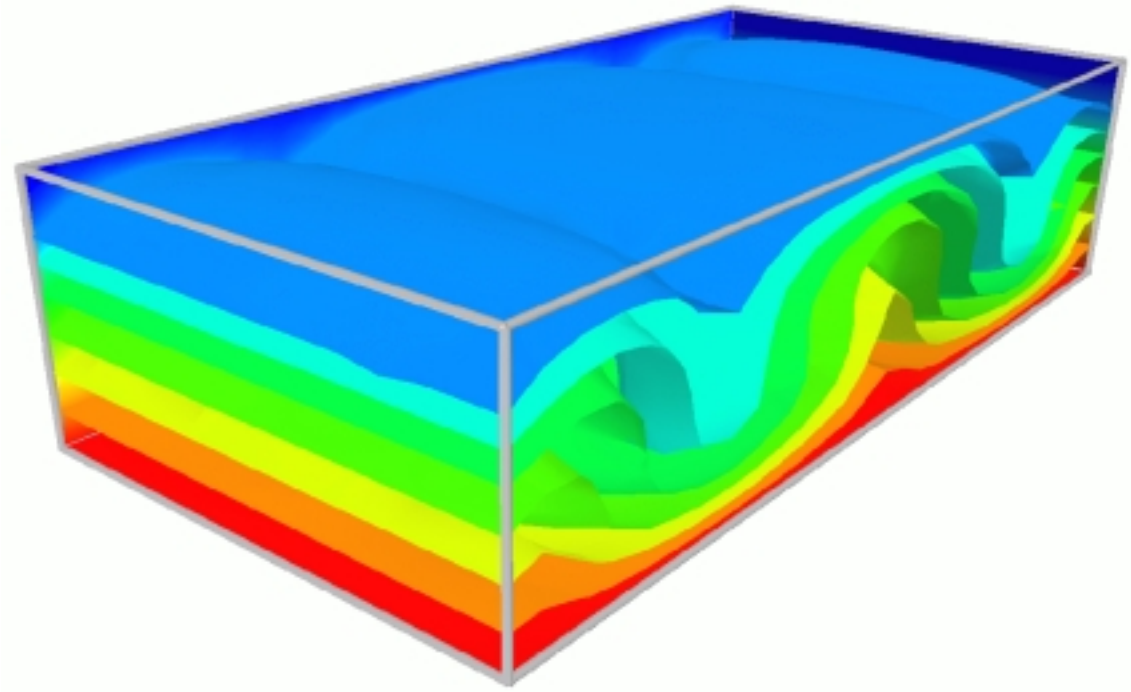


Abb. 7.14: Bénard–Konvektion: Sieben Isoflächen der Temperatur

7.1.4 Anwendungen mit Gitterverschiebungen

Die folgende Anwendung, deren Modulnetzwerk Abbildung 7.15 illustriert, verarbeitet Gitterverschiebungen und ist somit vor allem Problemen aus der Strukturmechanik vorbehalten. In der Anwendung werden neben Knotenergebnisdaten und Gitterdaten zusätzlich Gitterverschiebungen berücksichtigt. Die Verschiebungsvektoren greifen dabei an den Gitterknoten an und werden mit dem Modul *displacements* eingelesen. Die Daten liegen in einem Patranformat vor. Der Ergebnisvektor enthält für jeden Knotenpunkt 6 Spalten, je 3 Freiheitsgrade der Translation und der Rotation.

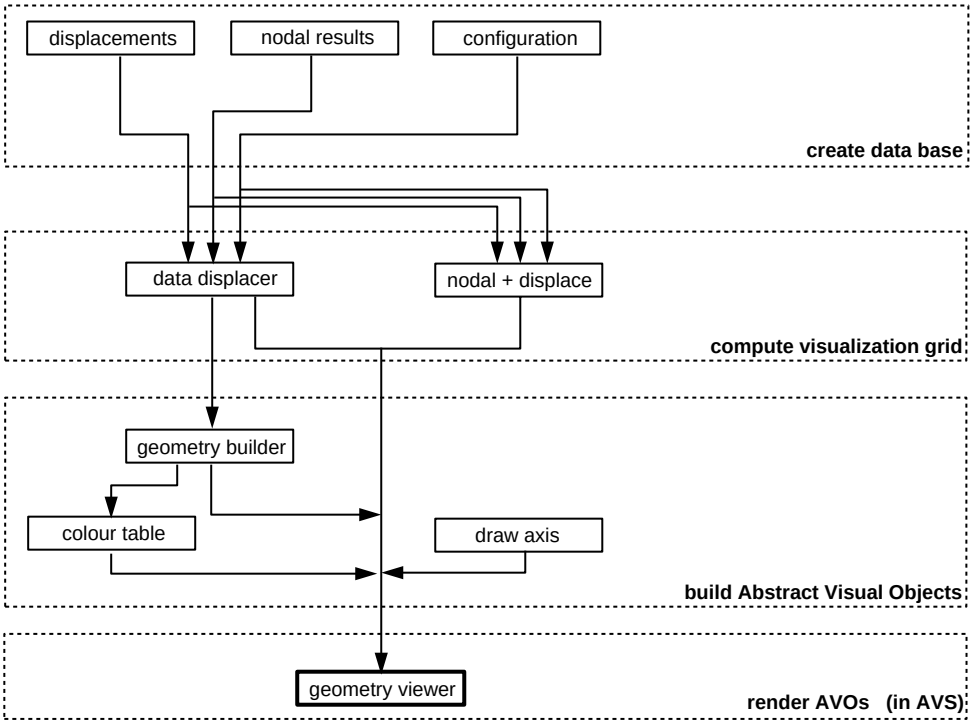


Abb. 7.15: Serielle Modulnetzwerke 4

Das Modul *data displacer* (Abb. 7.6) entspricht in seiner Funktionalität weitgehend dem bereits besprochenen *data transformer*, enthält jedoch einen zusätzlichen Dateneingang (*input port*) für Verschiebungsdaten, sowie den Widget *amplitude*, mit dem jeder aufgebrachte Verschiebungsvektor multipliziert wird. *Amplitude = 0* entspricht demnach der Struktur im unverschobenen Ausgangszustand. Die Ausgabedaten dieses Moduls werden wie gehabt mit dem Modul *geometry builder* zu Abstrakten Visuellen Objekten weiterverarbeitet.

Das Modul *nodal + displace* ist ein besonderes Werkzeug für die Strukturmechanik, da es neben Ergebnissen aus einer statischen FE–Analyse auch den Eigenvektor ei-

ner dynamischen Modellanalyse verarbeitet. So kann aus einer Eigenschwingung ein Animationszyklus generiert und dargestellt werden. Hierzu wird mit *nr. of steps* die Anzahl der zu generierenden Bilder festgelegt, die überdies auch jede Phase einzeln betrachtet werden können (*file/step number*). Ein entsprechendes Modul namens *element + displace* verarbeitet Elementergebnisse anstelle von Knotenergebnissen.

Visualisierung am Beispiel eines Windturbinenflügels

In dem in Abbildung 7.16 dargestellten Beispiel handelt es sich um einen eingespannten Flügel einer Windturbine, der unter dynamischer Belastung steht [37]. Dargestellt wurde die verschobene Struktur in ihrer maximalen Verformung in der Abschwungphase. Im Bild rechts zu sehen ist das verformte Oberflächengitter mit 1320 Finiten Elementen, wohingegen im Bild links Konturflächen sowie Isolinien der von Mieses–Vergleichsspannung gezeigt werden.

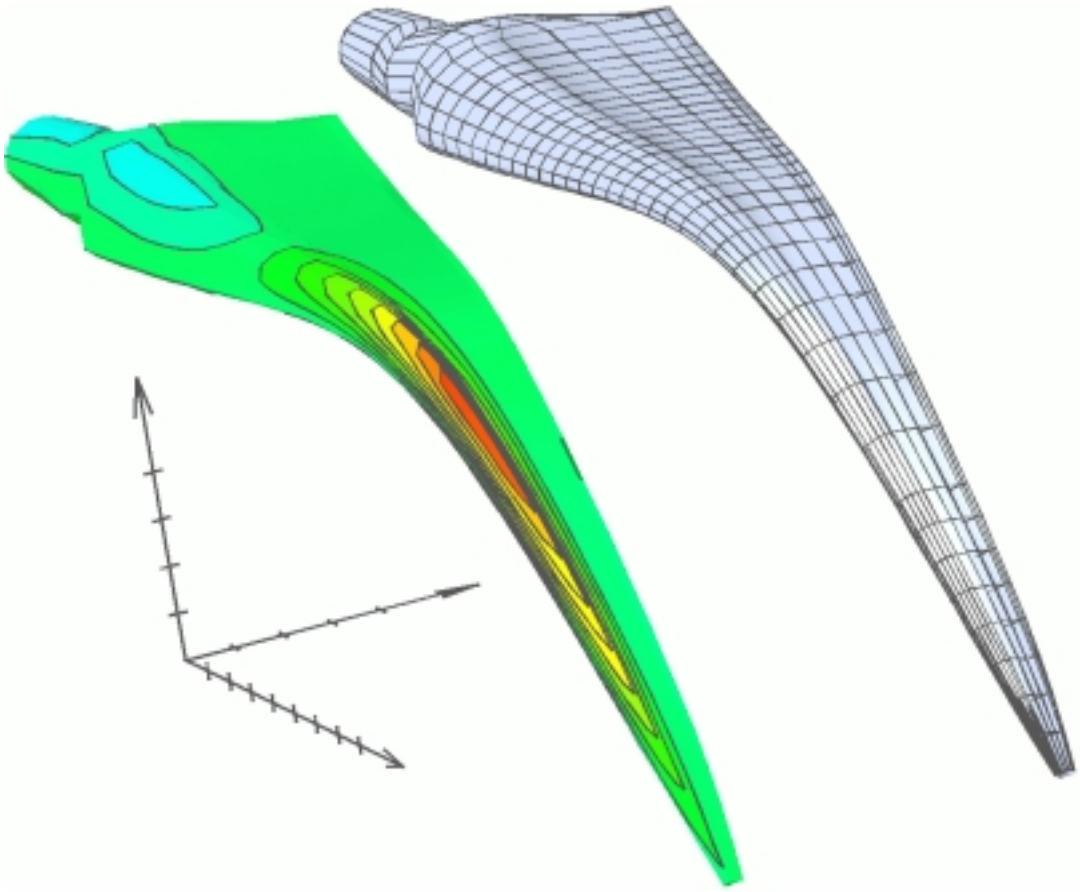


Abb. 7.16: Flügel einer Windturbine unter dynamischer Windlast

7.2 Parallele Anwendungen

In diesem Abschnitt werden zwei Parallelanwendungen diskutiert, die in den vorigen Kapiteln vorgestellt wurden.
Im ersten Fallbeispiel wird im sogenannten Standalone-Modus ein Parallelrechner eingesetzt, um das Postprozessing von Analyseergebnisdaten zu beschleunigen, die von einer seriellen Finite Element Rechnung stammen.
Das zweite Fallbeispiel behandelt eine integrierte Simulation. Hier wird interakiv in eine laufende Finite Element Analyse eingegriffen, indem der Benutzer zu gewissen Zeitpunkten die Möglichkeit wahrnimmt, einen Visualisierungsauftrag zu übermitteln, z.B. zum Ende eines Zeitschrittes des numerischen Lösungsverfahrens. Auf der Basis der gerade aktuell verfügbaren Analysedaten werden dann parallel die entsprechenden Visualiserungsgitterdaten erzeugt und zum seriellen Geometrie–Mapping und Rendering direkt vom Parallelrechner auf die Graphikworkstation übertragen.

7.2.1 Anwendungen im Standalone-Modus

Das in Abbildung 7.17 dargestellte Modulnetzwerk erlaubt eine interaktive Stromlinienberechnung sowohl seriell als auch parallel in Verbindung mit einem assoziierten Parallelrechner. Für den parallelen Fall werden zusätzliche Moduln benötigt, die die Steuerung und den Datenexport bzw. -Import übernehmen (siehe Abb. 7.17 Mitte).

Wie im seriellen Fall lesen die Moduln *nodal results* und *configuration* die Analysedaten ein, und das Modul *point generator* erzeugt die Startpunkte. Für eine rein serielle Stromlinienberechnung sorgt das Modul *flow server 3d*. Alternativ jedoch können die Gesamtdaten des Problems dem Modul *export FE-data* zur Verfügung gestellt werden, welches sie dann auf einen Parallelrechner überträgt.
Zuvor aber muß die Parallelanwendung mit Hilfe des Moduls *parallel control* interaktiv mit einer bestimmten Anzahl von Prozessoren initialisiert worden sein. Falls das Dateiensystem des Parallelrechners nicht über das lokale Netzwerk verfügbar sein sollte, muß das Parallelprogramm von Hand gestartet werden.
Außerdem ist es erforderlich, dem Parallelrechner den Auftrag bekannt zu machen. Während das Modul *parallel control* Informationen über die Herkunft der Daten übermittelt, legt das Modul *graphics parameters* die Visualisierungsparameter fest und transferiert sie zum Parallelrechner. Am Ende der parallelen Gittertransformation exportiert der Parallelrechner die Visualisierungsgitterdaten prozessorweise zurück an die Workstation, wo die Daten mit dem Modul *import str data* empfangen werden.

Die Erzeugung von AVOs im Modul *streamline builder* und das abschließende Rendering mit dem *geometry viewer* von AVS erfolgen wieder seriell.

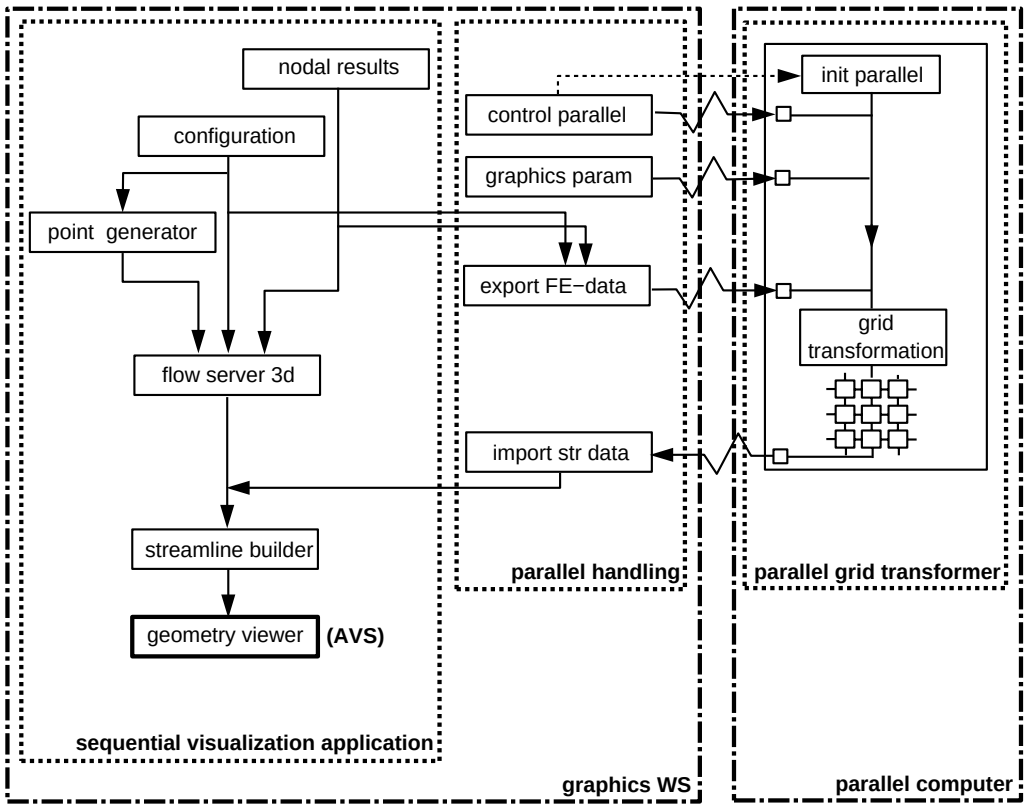


Abb. 7.17: Parallele Modulnetzwerke 1

Visualisierung am Beispiel einer Bénard-Konvektion

Als konkretes Rechenbeispiel wurde wieder das bereits beschriebene Bénard-Konvektionsproblem mit den Kenngrößen $Pr = 7, Ra = 19870$ herangezogen.
Es wurden 64 Stromlinien parallel auf 2 bis zu 32 Prozessoren berechnet (Abb. 7.19), indem entlang des Vektorfeldes integriert wurde, wobei das in Abschnitt 4.3.8.4 behandelte Shoot-Verfahren zur Anwendung kam. Die Maximalzahl der Iterationen wurde mit 2500 festgelegt. Das sequentielle Gesamtproblem wurde dem Masterprozessor übertragen, der die Daten den restlichen Prozessoren bekanntgab (*broadcast).* Die Startpunkte wurden so umverteilt, daß in einem Netzwerk mit 32 Prozessoren jeder Prozessor exakt 2 Stromlinien zu berechnen hatte.
Abbildung 7.18 zeigt die Ergebnisse, wie sie auf einer Parsytec MC3 sowie auf einer Intel Paragon erhalten wurden.
Speed-ups $S(n_p)$ a) und Effizienzen $E(n_p)$ b) für beide Parallelrechner sind einander gegenübergestellt und zusammen mit den Gesamtrechenzeiten $t(n_p)$ c) und der normalisierten Kommunikationszeit $t_c(n_p)/t(n_p)$ über der Prozessorzahl n_p aufgetragen.

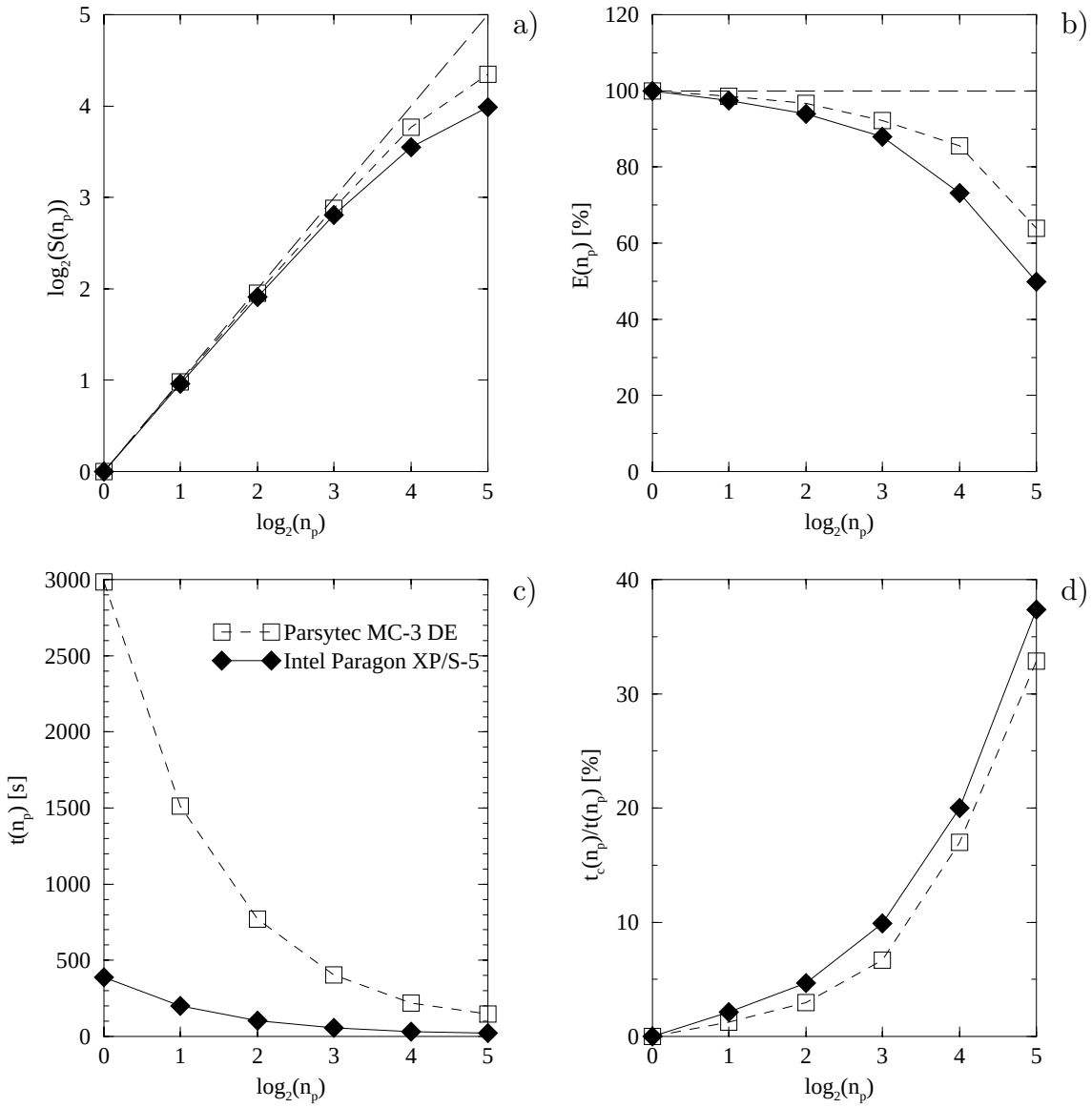


Abb. 7.18: Benchmarks für eine verteilte Stromlinienberechnung mit 2 bis zu 32 Parallelprozessoren.

Die Übertragungszeiten beider Parallelrechner liegen ziemlich gleich, weil die beteiligte Graphikworkstation, eine Stardent GS1000, vergleichsweise langsam ist. Deshalb werden die Übertragungszeiten des Transmitters nicht berücksichtigt, um die Ergebnisse nicht zu verfälschen. Für das Beispiel ist festzuhalten, daß der Einsatz eines Parallelrechners gerechtfertigt war, da, wie zu erwarten stand, sich mit steigender Prozessorzahl die absolute Rechenzeit stark verminderte. Beim Einsatz von bis zu 8 Prozessoren ist der Speedup sogar nahezu linear. Die Linearität ist ja der Idealfall für die Effizienz eines parallelen Algorithmus. Allerdings fällt der Einfluß der Kommunikationszeiten mit wachsender Prozessorenzahl immer mehr ins Gewicht.

Während die reinen Rechenzeiten sich mit jeder Verdoppelung der Anzahl eingesetzter Prozessoren beinahe halbieren lassen, nimmt der Kommunikationsaufwand jedoch überproportional zu.
Offensichtlich ist auch, daß das gerechnete Problem nicht beliebig skalierbar ist, denn 64 Stromlinienstartpunkte auf mehr als 64 Prozessoren aufzuteilen ist wenig sinnvoll.

Anmerkung

Einen Anhaltswert für eine sequentielle Berechnung des Problems mag bekommen, wer von der Absolutrechenzeit (2980 s) mit dem Parsytecrechner (ein eigesetzter Prozessor) die Rüstzeit abzieht (ca. 2.3 Prozent), die für den Transfer des Rechenauftrages und der Problemdaten sowie für den Rücktransfer der Berechnungsergebnisse bei paralleler Verarbeitung aufgewendet werden mußte.

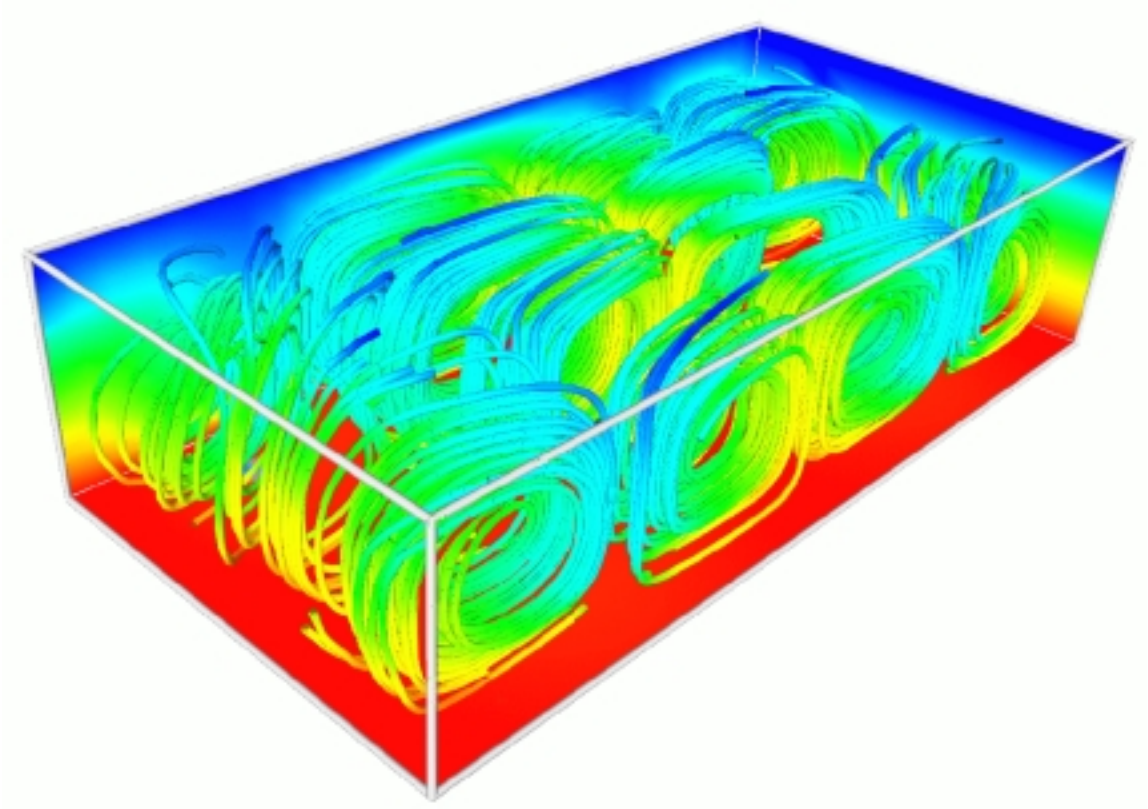


Abb. 7.19: Quasi stationärer Zustand einer Bénard-Konvektion

7.2.2 Anwendungen im integrierten Modus

Es wird der Fall betrachtet, bei dem die Visualisierung von Analyseergebnisdaten während des Ablaufs der parallelen Analyseberechnung erfolgen soll. Die erste Möglichkeit, Analyseergebnisdaten direkt an ein empfangsbereites Modul auf der Graphik–Workstation zu transferieren, soll erwähnt, aber nicht weiter untersucht werden. Konzentrieren wir uns auf die zweite Möglichkeit, bei der das Visualisierungsgitter parallel auf den Prozessoren der Analyse berechnet und die erhaltenen Gitterdaten dann für das Geometrie–Mapping und Rendering prozessorweise an die Graphik–Workstation exportiert wird. Beide Möglichkeiten sind in Abbildung 7.20 enthalten.

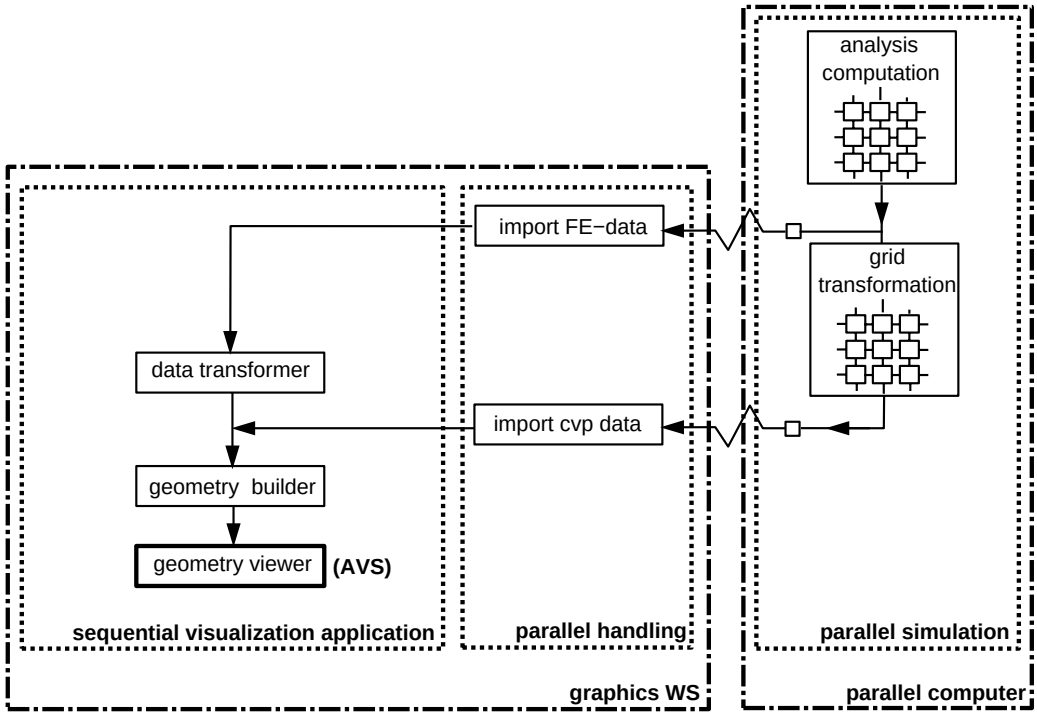


Abb. 7.20: Parallele Modulnetzwerke 2

Der Benutzer signalisiert interaktiv den Wunsch, während der Analyse Zwischenergebnisse zu visualisieren. Dabei wird eine Marke gesetzt, die der Masterprozessor zu einer bestimmten Zeit, wie z.B. am Ende eines Rechenzeitschritts, einliest und somit feststellt, daß weitere Parameter und Daten zur Initialisierung der Berechnungen des für die Visualiserung maßgeblichen Gitters eingelesen werden müssen. Auf dieser Basis können dann die erforderlichen Routinen des parallelen Gittertransformators ablaufen. Die Analyseberechnung wird solange ausgesetzt.
Über das Importmodul stellt der Benutzer einen Transmitter zur Verfügung. Schlägt eine Übertragung der Visualisierungsgitterdaten über das Netz fehl, nimmt die Ana-

lyse ihre Simulationsberechnung genauso wieder auf wie nach erfolgreicher Übertragung.

Integrierte Visualisierung am Beispiel einer Kreiszylinderumströmung

Das folgende Rechenbeispiel behandelt eine Quasi-Echtzeit-Simulation nach dem obigen Schema. In dem FE-Problem, das in Abbildung 7.22 illustriert ist, wird eine reibungsfreie Strömung von dissoziiertem Stickstoff um einen zweidimensionalen Kreiszylinder mit einem Radius von 2.54 cm berechnet. Der inkrementelle Verlauf der Lösung bildet das Experiment von Hornung [39] nach, das unter Freistrombedingungen gewonnen wurde ($Ma_\infty = 6.14, \rho_\infty = 5.349 * 10^{-3} kg/m^3, T_\infty = 1822K$). Als Parallelrechner standen wieder die Intel Paragon und die Parsytec MC3 zur Verfügung. Gerechnet wurde auf 2 bis zu 32 Prozessoren.
Von den Zwischenergebnissen, die der laufenden Rechnung entnommen wurden, sind drei Zustände in Abbildung 7.22 dargestellt. Gezeigt wird die Lösung nach 500, 1500 und 4000 Zeitschritten. Parallel berechnet wurden Konturflächen sowie Isolinien des Druckes. Im Vergleich nahm die Berechnung des Visualisierungsgitters weniger als 20 Prozent der Rechenzeit eines Lösungszeitschrittes in Anspruch. Die Gebietsaufteilung des verteilten FE–Gitternetzes mit 8256 Elementen ist in Abbildung 7.21 für 32 Prozessoren zu sehen.

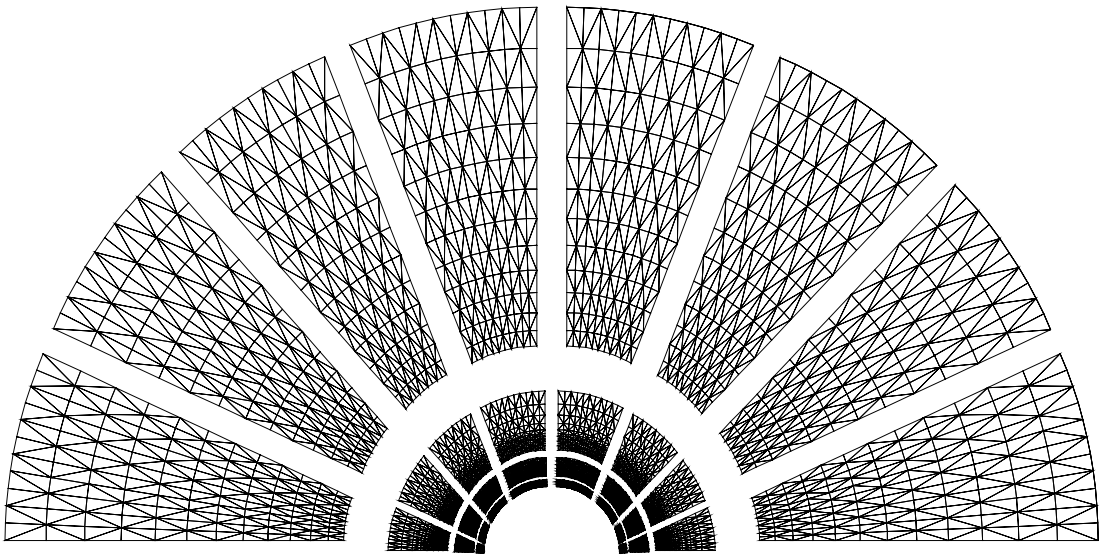


Abb. 7.21: Verteiltes Gitternetz für 32 Parallelprozessoren

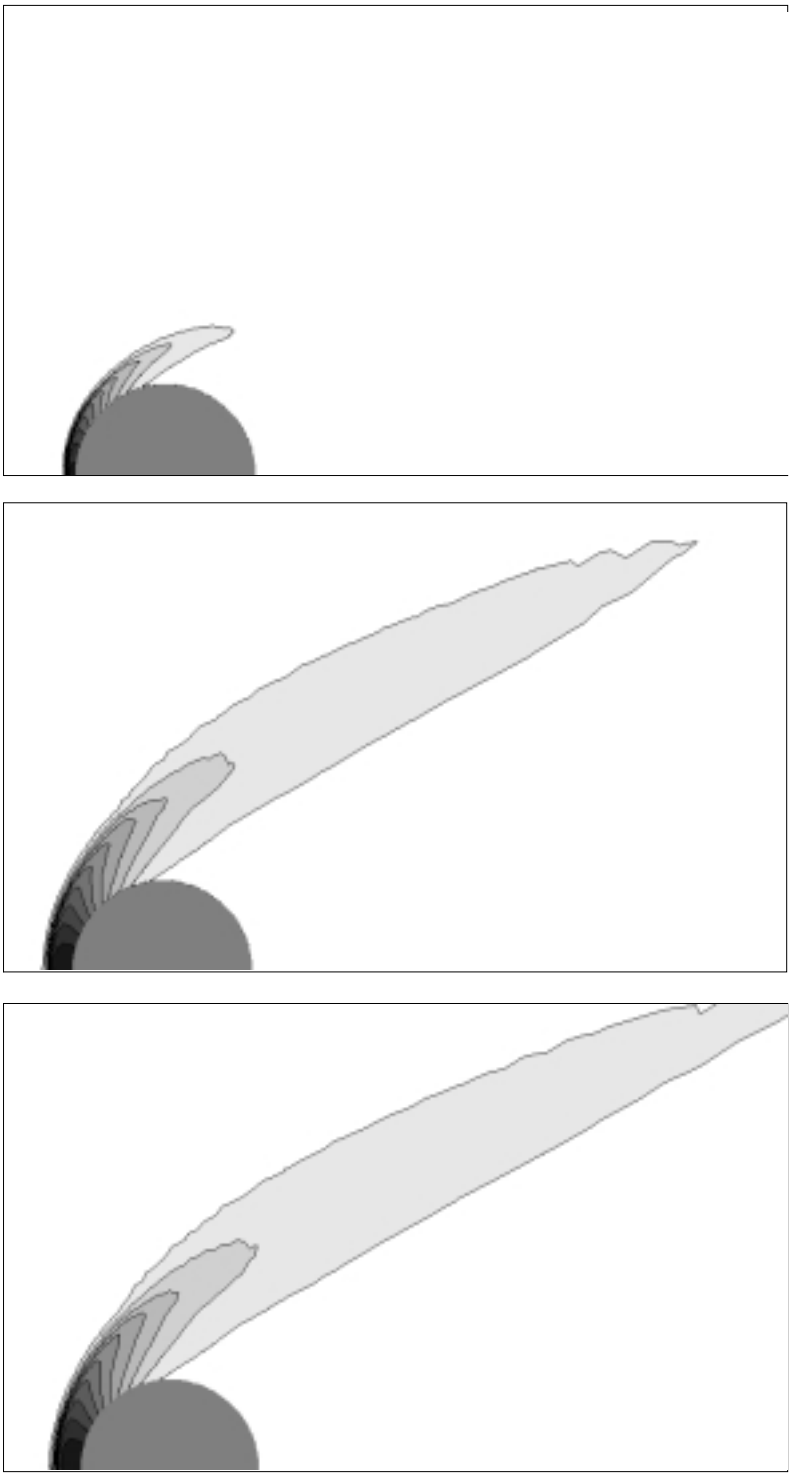


Abb. 7.22: Druckverteilung um einen Kreiszylinder in dissoziiertem Stickstoff nach 500, 1500 und 4000 Zeitschritten.

Einen Vergleich der Rechenzeiten für verschiedene Prozessorzahlen liefern die Abbildungen 7.23 und 7.24. Abbildung 7.23 bezieht sich dabei auf die Rechenzeit, die für einen numerischen Zeitschritt inklusive der Berechnung des Visualisierungsgitters benötigt wurde. In Abbildung 7.24 hingegen geht nur die Berechnung des Visualisierungsgitters ein. Benchmarks für eine Berechnung des Problems ohne Visualisierung sind [83, 15] zu entnehmen.

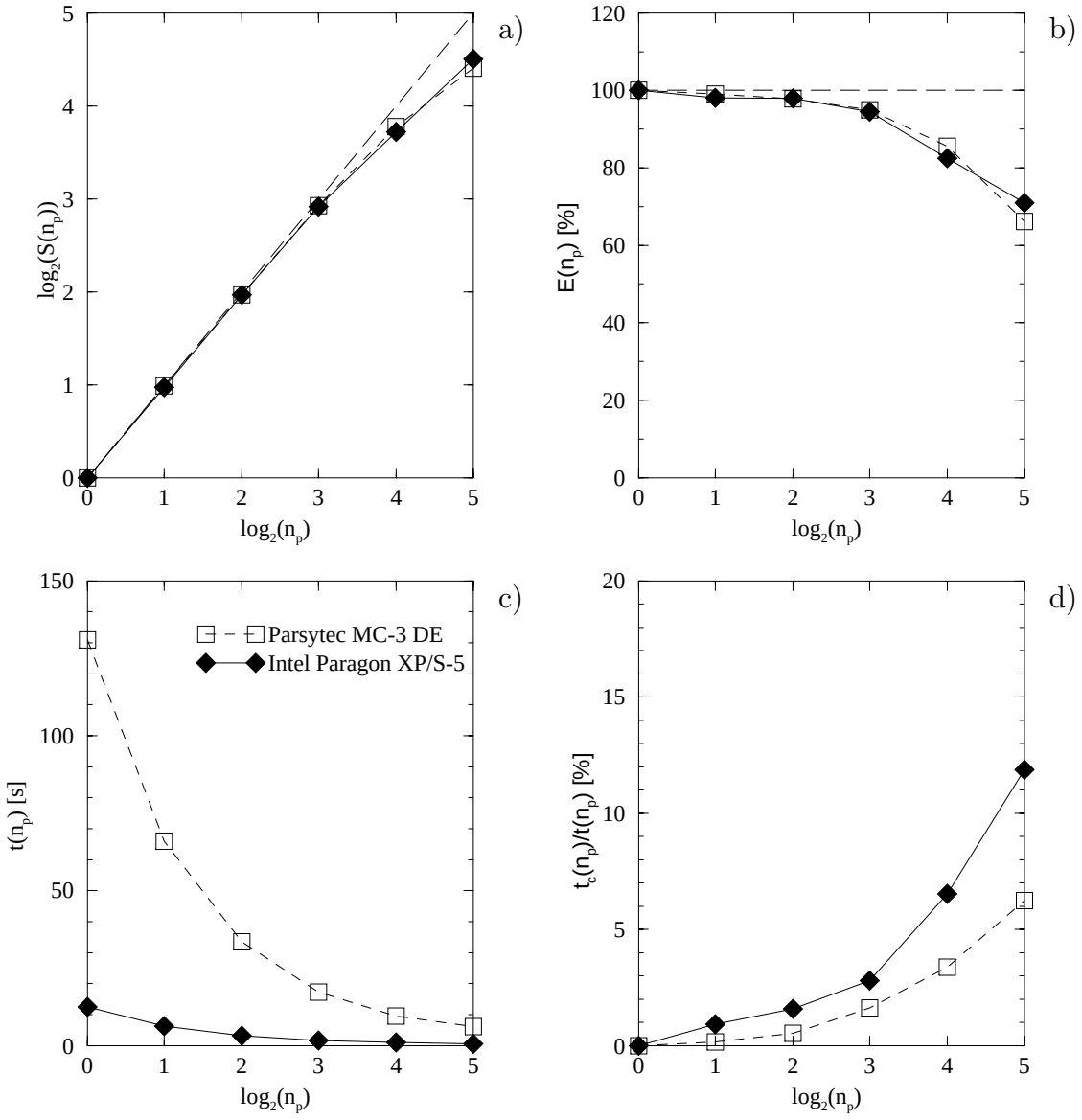


Abb. 7.23: Benchmarks für eine verteilte Konturflächenberechnung mit 2 bis zu 32 Parallelprozessoren (Analyse und Graphik).

Für beide Parallelrechner, Parsytec MC-3 und Intel Paragon, sind in den Diagrammen jeweils die Rechenzeit pro Zeitschritt $t(n_p)$, Speedup $S(n_p)$ und Effizienz $E(n_p)$

sowie die normaliserte Kommunikationszeit $t_c(n_p)/t(n_p)$ über der Prozessorzahl n_p aufgetragen.

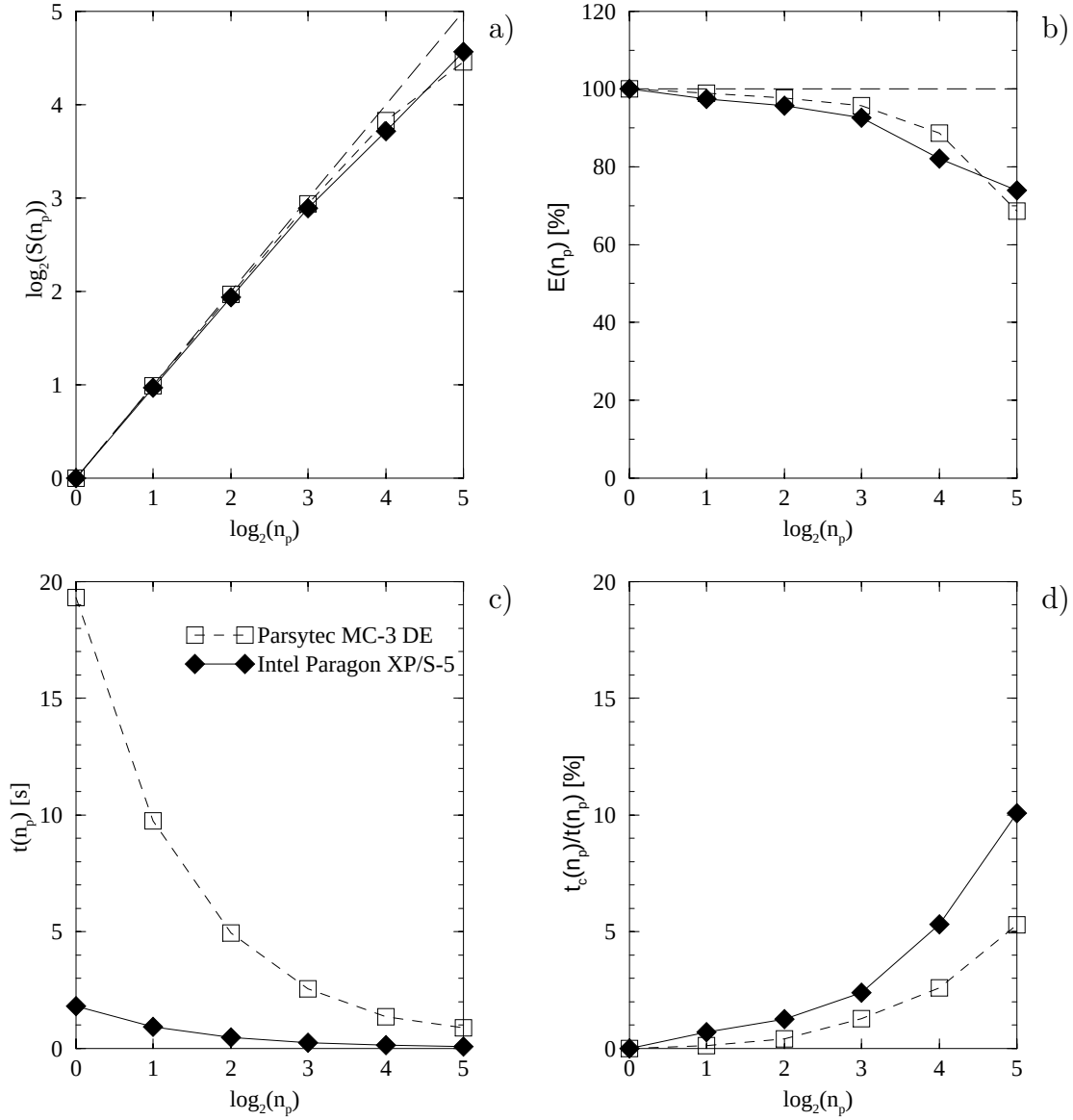


Abb. 7.24: Benchmarks für eine verteilte Konturflächenberechnung mit 2 bis zu 32 Parallelprozessoren (nur Visualisierung).

Das Verhältnis von Kommunikationszeit zu Gesamtrechenzeit ist bei der Berechnung des Visualisierungsgitters vom gewählten Algorithmus und den Wahlparametern abhängig. In diesem Beispiel ist ein Wahlparameter die Anzahl der zu berechnenden Isolinien und Konturflächen. Die normalisierte Kommunikationszeit verringert sich wesentlich mit steigendem Rechenaufwand. Ob wie hier 12 Konturflächen berechnet werden sollen oder 50, der Aufwand für die Kommunikation wächst im Verhältnis zum Rechenaufwand nur geringfügig an.

Kapitel 8

Zusammenfassung

Die vorliegende Arbeit befaßt sich mit dem Einsatz massiv paralleler Rechner in der Visualisierung. Es wurde ein Visualisierungsprogrammsystem konzipiert und entwickelt, das in erster Linie auf die numerische Strömungsmechanik zugeschnitten ist, jedoch auch Probleme der Strukturmechanik visualisieren kann.
Wesentliches Merkmal des Programmsystems ist seine Flexibilität hinsichtlich modularem Aufbau, Betriebsmodus und Rechnereinsatz. Der modulare Aufbau gestattet es dem Benutzer, durch Einbindung unterschiedlicher Moduln in ein Modulnetzwerk individuelle, interaktiv modifizierbare Anwendungen zu erstellen. Der interaktive Zugang und die Steuerung der Moduln wurde dabei unter die Verwaltung des Programmsystems AVS [82] gestellt.
Den Moduln übergeordnet ist der Betriebsmodus. Es werden vier Betriebsmodi unterschieden: serieller, paralleler, alleinstehender (*standalone*) und gekoppelter Betrieb. Somit kann das System sowohl rein seriell auf einer Graphikworkstation als auch in Verbindung mit einem Parallelrechner betrieben werden. Im standalone Modus wird es unabhängig von einer numerischen FE–Analyse für das Pre– und Postprozessing eingesetzt. Eine Auslagerung zeitaufwendiger Berechnungen der Gitterdaten, die für die Visualisierung repräsentativ sind (Gittertransformation), auf einen assoziierten Parallelrechner führt hier zu einer merklichen Beschleunigung des Postprozessing.
Andererseits kommt dem System eine integrierende Funktion zu, wenn es eng an eine serielle oder parallele numerische FE–Analyse gekoppelt und somit zu einem Hybridsystem wird. Auf diese Weise wird eine Visualisierung von Daten aus einer laufenden FE–Analyse möglich. Im gekoppelten Modus werden überdies Aufgaben wahrgenommen, die über die eigentliche Visualisierung von Ergebnisdaten hinaus-

gehen.
So ist die Datenreduktion im Vorfeld der Auswertung nur eine Konsequenz dieser engen Kopplung. Ferner werden Aufgaben der Steuerung und der Einflußnahme auf die numerische Simulation sowie die Regelung des direkten Datenaustauschs zwischen beteiligten Systemkomponenten übernommen. Das führt zu einer Beschleunigung des gesamten Simulationsprozesses. Zum einen ist das dem Einsatz von Parallelrechnern zuzuschreiben und der damit verbundenen Aufgabenteilung und Datenparallelität und zum anderen der engen Verknüpfung, indem Algorithmen der FE–Analyse und der Gittertransformator auf denselben Datenstrukturen operieren, und der Datenaustausch direkt ohne Zwischenspeicherung über das lokale Netzwerk geschieht. Zudem besteht so die Möglichkeit, durch Visualisierung während des Ablaufs der Simulation Fehlentwicklungen frühzeitig zu erkennen und entsprechende Maßnahmen einzuleiten, was den Gesamtaufwand erheblich reduziert.

Das modulare Visualisierungsprogrammsystem umfaßt im wesentlichen die voneinander unabhängigen Komponenten Gittertransformator, Geometrie–Mapping und Rendering, sowie Datenaustausch, die alle über eine interaktive Benutzeroberfläche gesteuert werden. Die eigentliche Visualisierung wurde in zwei Bereiche untergliedert, deren Aufgabengebiete sich eindeutig voneinander abgrenzen lassen: der Gittertransformator, der für die Datenreduktion und die Berechnung des Visualisierungsgitters zuständig ist, sowie den darstellenden Teil, den Geometrie–Mapper einerseits, der aus den Ergebnissen der Gittertransformation Abstrakte Visuelle Objekte generiert, und den Renderer andererseits, der diese Objekte letztendlich in Bilder faßt.

Ein Schwerpunkt der Arbeiten lag in der Entwicklung verschiedener Visualisierungstechniken und -algorithmen, die in der Systemkomponente Gittertransformator zusammengefaßt sind. Die Algorithmen transformieren aus den zur Verfügung stehenden Analyseergebnissen selektive Daten, die für die Visualisierung repräsentativ sind (Visualisierungsgitterdaten). Dies kann zum Beispiel die Integration eines Vektorfeldes sein, um die Koordinaten von Stromlinien zu erhalten.
Als Basis der Gittertransformation fungiert ein Datenschema, das nur die notwendigsten Datenfelder enthält, damit alle vier erwähnten Betriebsweisen unterstützt werden können. Kenntnis interner Ansatzfunktionen der verschiedenen Finiten Elemente wurden nicht vorausgesetzt. Die dadurch verursachten Fehler liegen außerhalb der visuellen Wahrnehmbarkeit.
Standard–Darstellungsweisen wie eine kontinuierliche Darstellung von Ergebnissen (Fringeplot), Isolinien, Konturflächen, Isoflächen, Schnitte, Vektoren und Stromlinien werden ebenso zur Verfügung gestellt wie Volumenausschnitte, die Verfolgung von Partikeln und verschiedener in der Strömung losgelassener Flächen. Außerdem können Bilderfolgen algorithmisch erstellt und im Geometrie–Mapper und Renderer

in Animationszyklen eingebunden werden. Als Beispiele hierfür seien nur die Wanderung einer Schnittebene durch einen kräftebeaufschlagten Körper mit Darstellung der Spannungen auf den Schnitten angeführt, die Verfolgung von Partikeln in einer Düsenströmung oder eine Sequenz von Flächen gleicher Temperatur in einer Konvektionszone.
Alle diese Funktionen weisen interne Parameter auf, von denen die wesentlichen interaktiv zugänglich sind. Dies betrifft die Einschränkung von Wertebereichen genauso wie die Wahl von Stückzahlen, verschiedener Generierungsmodi (z.B. linear oder logarithmisch) und der Wahl physikalischer Größen und diverser Skalierungsfaktoren. Ferner liefern die Algorithmen Visualisierungsgitterdaten in einem den Ausgangsgitterdaten ähnlichen Format, sodaß auf die Daten weitere Algorithmen angesetzt werden können. Dadurch lassen sich beispielsweise Vektoren in Schnitten darstellen oder Isoflächen in einem ausgewählten Teilvolumen.
Der Gittertransformator kann auf verschiedenen seriellen und parallelen Rechnern physikalisch oder modular getrennt von der Darstellung betrieben werden und kommt ohne eine Graphikbibliothek aus. Vorteil der Trennung von Gittertransformation und Geometrie–Mapping und Renderer ist nicht nur, daß sich das Paket aus seriellen und parallelen Routinen leicht in ein bestehendes serielles oder paralleles numerisches Analyseprogrammsystem integrieren läßt, sondern auch die Wahl des Renderers über festgelegte Schnittstellen offengehalten wird.

Der Geometrie–Mapper und Renderer übernimmt die Ergebnisdaten aus der Gittertransformation und faßt sie mit Hilfe einer standardisierten Graphikbibliothek in Abstrakte Visuelle Objekte (Primitivenbildung und Polygon–Rendering). Anschließend werden die AVOs am Bildschirm dargestellt.
Dazu wurden entsprechende Moduln entwickelt, die zum einen die Schnittstelle zum Gittertransformator überbrücken und zum anderen die Abstrakten Visuellen Objekte aufbauen. Schnittstellenmoduln erhalten ihre Daten entweder von einem sequentiellen Gitterberechnungsmodul im selben Modulnetzwerk, oder durch Einlesen abgespeicherter Dateien mit Visualisierungsgitterdaten oder aber direkt über das lokale Netz von einer fernen Workstation oder einem Parallelrechner.
In die Objektbildung gehen interaktive Wahlparameter mit ein, die die Farbmodellwahl, Objektauswahl, Schattierungstechniken, Normalengenerierung, die Erstellung von Farbwertetabellen, Darstellungsmodus und Namensgebung der Objekte und die Generierung einer Animationsfolge betreffen.
Die Darstellung am Bildschirm wird von einem AVS–Modul (*geometry viewer*) übernommen, dem einzigen Modul, das nicht neu entwickelt wurde.

Für die serielle Anwendung wurden beide Systemkomponenten in den kommerziellen Application Builder AVS integriert, der auch der Benutzeroberfläche als Basis dient,

die neben den beiden Hauptkomponenten des Visualisierungsgprogrammsystems in Grundzügen auch den Ablauf der numerischen Simulation steuert und überwacht.
In der parallelen Anwendung hingegen übernimmt die Benutzeroberfläche auch andere Aufgaben wie etwa die Verwaltung und Zuweisung der Parallelprozessoren und die Steuerung des Datenaustausches zwischen den Komponenten. Somit ist gewährleistet, daß alle diese Bereiche miteinander korrelieren und jederzeit separat angesprochen werden können.
Desweiteren stellt das Programmsystem die Werkzeuge zur Verfügung, fremde Komponenten einzubinden und zu steuern. Eine Steuerung wurde implementiert, soweit es die Kopplung von Analyse und Visualisierung im parallelen Betrieb betrifft.

Ein weiterer Schwerpunkt der Arbeit lag auf dem Datenaustausch zwischen beteiligten Komponenten des Programmsystems. Eine Transmittersoftware für eine Punkt–zu–Punkt–Kommunikation wurde entwickelt, die zur Laufzeit einen direkten Datenaustausch zwischen Prozessen einzelner Systemkomponenten ermöglicht. Dabei können zwei kommunizierende Prozesse auf demselben Rechner oder auf verschiedenen Rechnern arbeiten. Im Unterschied zu kommerzieller Software [63] müssen Verbindungen mit Rechnern nicht im voraus spezifiziert und aufgebaut werden, sondern können interaktiv jederzeit angefordert werden.
Transmitter sind vermittelnde Prozesse, die auf den Sockets der Interprozeßkommunikation IPC [54] basieren. Sockets agieren auf Transportebene gemäß dem OSI–Referenzmodell [13] und stehen den meisten UNIX–Rechnern zur Verfügung. Ein Transmitter ist ein unabhängiger Prozeß, der von einem der beiden Seiten, dem Initiator, initialisiert wird, die einen Datentransfer wünscht. Sobald beide Seiten mit dem Transmitter verbunden sind, ist ein Datenaustausch möglich. Mit Aufnahme der Kommunikation wird ein spezielles Datenkennfeld ausgetauscht, das Auskunft über Art der zu übermittelnden Information gibt. Eine Verbindung besteht nur solange, bis die im Datenkennfeld spezifizierten Aufgaben abgearbeitet sind und terminiert dann automatisch.
Transmitter werden zum Beispiel zur direkten Übertragung von Ergebnisdaten aus einer laufenden FE–Analyse an einen Gittertransformatator eingesetzt. Hierzu setzt der Benutzer zu einem Zeitpunkt interaktiv einen Transmitter ab, mit dem die Analyseseite dann Verbindung aufnehmen kann.

Literaturverzeichnis

[1] G. Amdahl. *Validity of the Single Processor Approach to Achieving Large Scale Computer Capabilities.* Proc. AFIPS Comp. Conf. 30., 1967.

[2] J.D. Anderson. *Modern Compressible Flow.* Mc. Graw-Hill, 1982.

[3] J. Argyris, I.St. Doltsinis, and H. Friz. *Hermes Space Shuttle: Exploration of reentry aerodynamics.* Comp. Meth. Appl. Mech. Eng.,73, 1989.

[4] J. Argyris, I.St. Doltsinis, H. Friz, and J. Urban. *Physical and computational aspects of chemically reacting hypersonic flows.* Comp. Meth. Appl. Mech. Eng.,111, 1984.

[5] J. Argyris, H. Friz, and F. Off. Domain decomposition and operator splitting for parallel finite element computations of viscous real gas flow. In *Flow Simulation with High-Performance Computers*, volume 38 of *DFG Priority Research Programme, Results 1989-1992 (E.H.Hirschel, ed.) NNFM.* Vieweg, Braunschweig, 1993.

[6] J. Argyris and H.-U. Schlageter. Visualization of analysis data with reference to cfd. In *Flow Simulation with High-Performance Computers*, volume 38 of *DFG Priority Research Programme, Results 1989-1992 (E.H.Hirschel, ed.) NNFM.* Vieweg, Braunschweig, 1993.

[7] J. Argyris and H.-U. Schlageter. Parallel interactive and integrated visualization. In *Flow Simulation with High-Performance Computers*, DFG Priority Research Programme, (E.H.Hirschel, ed.) NNFM. Vieweg, Braunschweig, 1996.

[8] P. Bastian. *Parallele adaptive Mehrgitterverfahren.* ICA, Universität Stuttgart, November 1994. Dissertation.

[9] G. Beddies und J. Szimmat. *Petrov-Galerkin Finite Element Approximation dreidimensionaler Konvektionsströmungen.* ICA-Bericht Nr. 42. ICA, Universität Stuttgart, 1993.

[10] D. Braess. *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie.* Springer-Verlag, Berlin, Heidelberg, New York, 1991.

[11] G.F. Carey. *Parallel Computing, Methods, Algorithms and Applications.* Wiley, 1989.

[12] Ronald Cok. *Parallel Programs for the Transputer.* Prentice Hall., Englewood Cliffs, New Jersey, 1991.

[13] J.D. Day and H. Zimmermann. *The OSI Reference Model*, volume 71 (1334-1340). Proceedings of the IEEE, Dez. 1983.

[14] INTEL Supercomputing Systems Division. *Paragon Network Qeueing System Manual.* INTEL Corporation, Jan. 1994.

[15] I.S. Doltsinis and J. Urban. An adaptive operator technique for hypersonic flow simulation on parallel computers. In *Flow Simulation with High-Performance Computers*, DFG Priority Research Programme, (E.H.Hirschel, ed.) NNFM. Vieweg, Braunschweig, 1996.

[16] I.St. Doltsinis and S. Nölting. *Studies on Parallel Processing for Coupled Field Problems*, volume 89,No.1-3. Comput. Meths. Appl. Engrg., 1991.

[17] I.St. Doltsinis and S. Nölting. *Generation and Decomposition of Finite Element Models for Parallel Computations*, volume 2, No.5/6. Computing Systems and Engineering, 1992.

[18] D.L. Eager, J. Zahorjan, and E.D. Lazowska. *Speedup Versus Efficiency in Parallel Systems.* IEEE TOC 38,3, March 1989.

[19] Patran Division PDA Engineering. *PATRAN Plus, A Division of PDA Engineering*, volume 1+2, Release 2.3. User manual, Costa Mesa, California, 1988.

[20] T. Fischer. *Graphische Oberfläche zur interaktiven Darstellung parallel erzeugter Daten.* ICA, Universität Stuttgart, 1996. Studienarbeit.

[21] M.J. Flynn. *Some Computer Organizations and their Effectiveness.* IEEE TOC 21, 1972.

[22] G. Fox and M. Johnson. *Solving Problems on Concurrent Processors.* Prentice Hall International Editors, 1988.

[23] E.A. Franke, S.D. Huffman, W.M. Carter, J.P. Baumgartner, and D.J. Wenzel. *AVTP-an architecture for visualization using remote parallel/distributed computing*, volume 2410, (230-237). Proceedings of the SPIE - The International Society for Optical Engineering, USA, Feb. 1995.

[24] R.B. Haber, D.A. McNabb *Visualization Idioms: A Conceptual Model for Scientific Visualization Systems*, University of Illinois at Urbana-Champaign, (74-93). Visualization in Scientific Computing, 9/1990 IEEE.

[25] Tom Gaskins. *PHIGS Programming Manual*. O'Reilly & Associates, Inc., Sebastopol, CA, first edition, 1992.

[26] A. Geist, A. Beguelin, J. Dongarra, W. Jinag, R. Manchek, and V. Sunderam. *PVM 3 User's Guide and Reference Manual*. Oak Ridge National Laboratory, Oak Ridge, Tennessee, May 1993.

[27] P. Pacheco. *A User's Guide to MPI*. University of San Francisco, 1995.

[28] Parsytec Computer GmbH. *Parix 1.2, Reference Manual*, März 1993.

[29] Ingolf Grieger. *Graphische Datenverarbeitung*. Springer Verlag, Berlin, 2. Auflage, 1992.

[30] J.L. Gustafson. *Reevaluating Amdahl's Law*. Comm. ACM.31,2, 1988.

[31] W. Hackbusch. *Theorie und Numerik elliptischer Differentialgleichungen*. Teubner-Verlag, Stuttgart, 1986.

[32] Hans-Ulrich Heiss. *Prozessorzuteilung in Parallelrechnern*. Unix und seine Werkzeuge. BI-Wissenschaftsverlag, Mannheim Bonn, 1. Auflage, 1994.

[33] R. Hinden, J. Haverty, and A. Sheltzer. *The DARPA Internet: Interconnecting Heterogenous Computer Networks with Gateways*, volume 16 (38-48). IEEE Computer Magazine, Sept. 1983.

[34] K. Hinrichs und J. Nievergelt. *Programmierung und Datenstrukturen*. Springer Verlag, Berlin.

[35] C. Hirsch. *Numerical Computation of Internal and External Flow, Fundamentals of Numerical Discretisation*, volume 1. J. Wiley & Sons, Chichester-New York, 1990.

[36] C. Hirsch, J. Torreele, D. Keymeulen, D. Vucinic, and J. Deceuper. *Distributed Visualization in CFD*, volume 8. Speedup Journal, 1993.

[37] A. Hopf. *Auslegung eines Rotorblattes für eine Windturbine mit Hilfe der Methode der Finiten Elemente*. ICA, Universität Stuttgart, 1989. Studienarbeit.

[38] F.R.A. Hopgood and D.A. Duce. *A Primer for PHIGS*. John Wiley & Sons, New York, first edition, 1991.

[39] H.G. Hornung. Non-equilibrium dissociating nitrogen flow over spheres and cylinders. *Journal of Fluid Mechanics*, Vol. 53:149–176, May 1972.

[40] T.L.J. Howard and W.T. Hewitt. *A practical introduction to PHIGS and PHIGS PLUS*. Addison-Wesley Publishing Company, London, first edition, 1991.

[41] W.M. Hsu. *Segmented Ray Casting for Data Parallel Volume Rendering*. Parallel Rendering Symposium Proceedings, 1993.

[42] B. Huurdeman, H. Friz. *Entwicklung eines Finite Element Programmsystems zur Simulation turbulenter reaktiver Strömungen, Teil 1: Numerische Verfahren, Turbulenzmodelle und einfache Verbrennungsmodelle.* ICA–Bericht, Universität Stuttgart, 1996.

[43] Advanced Visual Systems Inc. *AVS developer's guide.* Waltham, 1992,1994.

[44] A. Wierse, U. Lang, R. Rühle. *Architectures of Distributed Visualization Systems and their Enhancements.* Eurographics Workshop on Visualization in Scientific Computing, Abingdon, 1993.

[45] Advanced Visual Systems Inc. *AVS user's guide.* Waltham, 1992,1994.

[46] IBM Inc. *IBM AIX Visualization Data Explorer, user's guide.* IBM Publication SC38–0081, 1995.

[47] IBM Inc. *IBM AIX Visualization Data Explorer, user's reference guide.* IBM Publication SC38–0081, 1995.

[48] IBM Inc. *IBM AIX Visualization Data Explorer, programmer's reference guide.* IBM Publication SC38–0497-5, 1995.

[49] Khoral Research, Inc. *Overview over the Khoros 2.0 Software Development System.* Program Services Volume 3, 1995.

[50] Khoral Research, Inc. W. Young, D. Argiro und S. Kubica. *Cantata: Visual Programming Environment for the Khoros System.* März 1995.

[51] Silicon Graphics Computer Systems Inc. *IRIS Explorer.* Technical Report BP–TR–1E–01, 1991.

[52] Advanced Visual Systems Inc. *AVS & Express user's guide.* 1996.

[53] G. Oberbrunner. *MP & Express Preliminary Specification.* Advanced Visual Systems Inc., May 1996.

[54] W. Joy, R. Fabry, F. Leffler, M. McKusick, and M. Karels. *UNIX Programmer's Supplementary Documents*, volume 1. Berkeley Software Architecture Manual, 4.3BSD Edition, Computer Systems Research Group, University of California, Berkeley, May 1986.

[55] K.R. Kirchartz. *Dreidimensionale Konvektion in quaderförmigen Behältern.* Habilitationsschrift. Universität (TH) Karslruhe, 1988.

[56] R. Kober. *Parallelrechner-Architekturen.* Springer Verlag, Heidelberg, 1988.

[57] Linda Kosko. *PHIGS Reference Manual.* O'Reilly & Associates, Inc., Sebastopol, CA, first edition, 1992.

[58] J. Krystynak. *Highspeed network issues in a distributed visualization application.* Proceedings. Supercomputing '92, IEEE Comput. Soc. Press, Los Alamitos, CA, USA, Nov. 1992.

[59] U. Lang, R. Lang, and R. Rühle. *Integration of visualization and scientific calculation in a software system.* Proceedings IEEE Visualization, 1991.

[60] P.F. Linnington. *Fundamentals of the Layer Service Definitions and Protocol Specifications*, volume 71 (1341-1345). Proceedings of the IEEE, Dez. 1983.

[61] K.-L. Ma, J.S. Painter, C.D. Hanson, and M.F. Krogh. *A Data Distributed Parallel Algorithm for Ray-Traced Volume Rendering.* Parallel Rendering Symposium Proceedings, 1993.

[62] P. Mackerras. *A Fast Parallel Marching Cubes Implementation on the Fujitsu AP1000.* Technical Report TR-CS-92-10. Department of Computer Science, Australian National University, 1992.

[63] T.G. Mattson. *Programming environments for parallel and distributed computing: a comparison of p4, PVM, Linda, and TCGMSG*, volume 9,(138-161). International Journal of Supercomputer Applications and High Performance Computing, 1995.

[64] P. McLendon. *Graphics Library Programming Guide.* Silicon Graphics Computer Systems, Inc., Mountain View, California, 1991.

[65] D. Modiano, M. Giles, and E. Murman. *Visualization of three-dimensional CFD solutions.* Number 89-0138. AIAA, 1989.

[66] S. Nölting. *Parallelisierung komplexer Finite Element Systeme.* ICA, Universität Stuttgart, 1999. Dissertation.

[67] Adrian Nye. *Xlib Programming Manual.* O'Reilly & Associates, Inc., Sebastopol, CA, third edition, 1992.

[68] Adrian Nye. *Xlib Reference Manual.* O'Reilly & Associates, Inc., Sebastopol, CA, third edition, 1992.

[69] P.F. Preparata and M.I. Shamos. *Computational Geometry, An Introduction.* Springer Verlag, New York, 1985.

[70] Valerie Quercia and Tim O'Reilly. *X Window System User's Guide.* O'Reilly & Associates, Inc., Sebastopol, CA, fourth edition, 1993.

[71] J.D. Schade und E. Kunz. *Strömungslehre.* de Gruyter, Berlin, 1980.

[72] H.-U. Schlageter. An integrated visualization system with support of parallel computers. In *Workshop on Visualization.* D. Kröner and R. Rautman (eds.), 1995.

[73] Schlichting und Truckenbrodt. *Aerodynamik des Flugzeuges*. Springer Verlag, Berlin, Heidelberg, New York, 1969.

[74] Schönthaler. *Software-Entwicklungswerkzeuge: Methodische Grundlagen*. Teubner Verlag, Stuttgart.

[75] I. Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, London, third edition, 1989.

[76] W. Richard Stevens. *UNIX Network Programming*. Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[77] W. Richard Stevens. *Advanced Programming in the UNIX environment*. Addison-Wesley Publishing Company, 1994.

[78] Andrew Tannenbaum. *Computer Netzwerke*. Wolframs Verlag, 2. Auflage, 1992.

[79] J. Torreele, D. Keymeulen, C.S. van den Berghe, J. Graat, and C. Hirsch. *Description of SIMD/MIMD-CFVIEW*. CEC Deliverable PASHA-D3.6/D4.6, 1993.

[80] E. Truckenbrodt. *Fluidmechanik*. Springer Verlag, dritte Auflage, Band 2, 1992.

[81] R. Uhl. *Berechnung und Darstellung von Stromlinien*. ICA, Universität Stuttgart, 1989. Studienarbeit.

[82] C. Upson. *The Application Visualization System: A Computational Environment for scientific visualization*. IEEE GG&A 9 No 4, 30–42, 1989.

[83] J. Urban. *Zur physikalischen Modellierung und numerischen Simulation aerothermodynamisher Vorgänge in hypersonischen Strömungen.* ICA, Universität Stuttgart, 1998. Dissertation.

[84] H. Wüstenberg. *FEPS 3.3 - Finite Element Programming System*. ICA, Universität Stuttgart, 1986. User's Guide.

Hans-Ulrich Schlageter
Türnenstr. 10
D–78648 Trossingen

Lebenslauf

Persönliches		Geboren am 1. Februar 1960 in Lörrach, verheiratet, 2 Kinder
Schulbildung	1966 – 1970	Grundschule, Lörrach
	1970 – 1977	Hans–Thoma–Gymnasium, Lörrach
	1977 – 1980	Technisches Gymnasium, Lörrach
	Juni 1980	Abitur
Ausbildung	10/80 – 04/88	Universität Stuttgart Studium der Luft– und Raumfahrttechnik, Abschluß als Diplom–Ingenieur
	09/82 – 01/83	Dornier GmbH, München Hauptpraktikum
	10/86 – 07/87	Institut für Statik und Dynamik, Universität Stuttgart Studienarbeit
	11/87 – 03/88	Daimler-Benz AG, Stuttgart Diplomarbeit
Zivildienst	07/88 – 02/90	Übernachtungsheim für Obdachlose
Beruf	10/88 – 03/90	Institut für theoretische Chemie, Universität Stuttgart geprüfter Hilfswissenschaftler
	05/90 – 03/96	Institut für Computeranwendungen, Universität Stuttgart, Wissenschaftlicher Mitarbeiter Im Anschluß an diese Tätigkeit entstand die vorliegende Dissertation.
	05/96 – 09/99	Angestellter der Firma TWT-GmbH, Abteilung IT/Projekte & Services.
	seit 10/99	Angestellter der Firma Perbit Software-GmbH, Abteilung Anwendungsprogrammierung & Support.