Technische Universität München
Institut für Informatik

Diplomarbeit

Ein adaptives hierarchisches Verfahren zur Approximation und effizienten Visualisierung von Funktionen und seine Anwendung auf digitale 3–D Höhenmodelle

Thomas Gerstner

Aufgabensteller: Prof. Dr. Christoph Zenger
Betreuer: Dr. habil. Michael Griebel

Beginn: 15. November 1994
Ende: 15. Juni 1995

Ich versichere, daß ich diese Diplomarbeit selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Juni 1995 ________________________________

Zusammenfassung

Es wird eine hierarchische Interpolationstechnik vorgestellt, die die effiziente graphische Darstellung von glatten Funktionen ermöglicht. Das Verfahren arbeitet aus diesem Grund anstatt mit den in der Finite–Elemente–Methode gebräuchlichen regulären Gittern mit semiregulären Gittern. Es läßt sich vorteilhaft zur Kompression und Darstellung von Bilddaten und dreidimensionalen digitalen Höhenmodellen für Geländedaten verwenden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die graphische Darstellung (Visualisierung) wird zu einem immer wichtigeren Hilfsmittel zum Verständnis und zur Interpretation komplexer Daten, wie z.B. physikalischen Meßwerten oder den Ergebnissen aus numerischen Simulationen. Visualisierungsprogramme haben dabei die Aufgabe, dem Benutzer eine vielfältige Zahl von Möglichkeiten zur geeigneten Darstellung dieser Daten zu geben. Dabei soll den Datensatz idealerweise interaktiv dargestellt, analysiert und verändert werden können.

Mit der kontinuierlichen Steigerung der Rechenleistung und Speicherfähigkeit von Computern steigt auch die Komplexität der so darzustellenden Datenmengen. Supercomputer können z.B. in aufwendigen numerischen Simulationen komplexe physikalische Probleme sehr detailliert berechnen. Durch Parallelisierung und Vernetzung können heutzutage riesige Mengen von Daten erzeugt und gespeichert werden. Zur Visualisierung kann aber im Normalfall nur ein einzelner Rechner verwendet werden, der dabei auf große Probleme stößt, weil er diese Datenmengen nicht mehr verarbeiten und darstellen kann.

Zwei Beispiele für diese Problemstellung sind die Bildverarbeitung und die Generierung eines dreidimensionalen Höhenmodells für Geländedaten zur Generierung einer synthetischen Sicht. Vor allem die Speicherung der anfallenden Daten stellt hier ein Problem dar. Integraler Bestandteil in beiden Beispielen ist die graphische Darstellung dieser Daten, die in Echtzeit stattfinden muß.

Wir wollen in dieser Arbeit ein Konzept vorstellen, mit dessen Hilfe große Datenmengen, gespeichert und dargestellt werden können. Hierzu werden die Daten in geeigneter Weise durch hierarchische Methoden komprimiert, mit der Vorgabe, daß ihre graphische Darstellung effizient erfolgen kann. Dieses Verfahren führt zu sehr guten Ergebnissen, falls die Daten gewissen Glattheitsbedingungen genügen.

Die Kompression der Daten erfolgt durch eine hierarchische Interpolationstechnik auf semiregulären Gittern. Durch die Verwendung von semiregulären Gittern wird sichergestellt, daß sich der Interpolant auf einem hierarchischen Level aus Basisfunktionen mit nicht überlappenden Trägern zusammensetzt. Dadurch kann die graphische Darstellung des Interpolanten effizient erfolgen.

Aufgrund der inhärenten Adaptivitätseigenschaften von hierarchischen Basen kann die Speicherung genügend glatter Bereiche effizienter erfolgen als bei nicht–adaptiven Verfahren, wodurch sich ein Kompressionseffekt einstellt. In dem vorgestellten Verfahren müssen diese komprimierten Daten zur graphischen Darstellung nicht wieder vollständig dekomprimiert werden, wodurch der Speichergewinn durch die Kompression erhalten bleibt. Die hierarchische Sichtweise erlaubt auch eine leichte Aufteilung des Gesamtgebiets in Teilgebiete, woduch bei einer ausschnittsweisen Darstellung auch nur die benötigten Teilgebiete im Speicher gehalten werden müssen.

Das Verfahren bietet weiterhin die Möglichkeit der adaptiven Darstellung des Interpolanten, wodurch uninteressantere Bereiche weniger genau gezeichnet werden können als interessante. Die Auswahl dieser Bereiche kann sowohl fehlerabhängig als auch ortsabhängig erfolgen. So kann die graphische Darstellung weiter beschleunigt werden.

Wir werden zeigen, daß der Interpolationsfehler in der Maximums- und der L2–Norm mit der Ordnung $O(h^2)$ abfällt, falls für die Ausgangsfunktion u die Glattheitsbedingungen $\Delta u \leq c$ gelten. Diese Eigenschaften stimmen mit aus der Finite–Elemente–Methode bekannten Standard–Interpolationstechniken auf regulären Gittern überein.

Mit diesem Verfahren konnten gute Ergebnisse bei der Bildkompression erzielt werden. Vor allem sein Einsatz für Videokonferenzen, wo Bilder schnell komprimiert, dekomprimiert und dargestellt werden müssen ist vielversprechend.

Bei der Darstellung der Höhendaten eines Testgebiets in den Alpen und Voralpen konnte die Zahl der zu zeichnenden Polygone um mindestens 80% ohne Beeinträchtigung der Qualität reduziert werden, wodurch die graphische Darstellung erheblich beschleunigt wurde. In flachen Gebieten sind hierbei sogar noch weit höhere Einsparungen möglich.

Wir werden uns zunächst im *zweiten Kapitel* mit den Grundlagen der graphischen Datenverarbeitung befassen und wir werden die Aufgaben von Graphiksystemen betrachten. Weiterhin werden wir über die Arbeitsweise von Visualisierungsprogrammen reden, ihre Probleme aufzeigen und unseren Lösungsansatz vorstellen.

In *Kapitel 3* werden einige Standard–Techniken zur Interpolation und zur ihrer Fehlerabschätzung erläutert. Wir werden die Vorteile der hierarchischen Sichtweise aufzeigen und Verfahren zur Interpolation auf regulären und dünnen Gittern beschreiben. Außerdem werden die Probleme bei der graphischen Darstellung der zugehörigen Interpolanten aufgezeigt.

Kapitel 4 stellt daraufhin eine Interpolationstechnik auf semiregulären Gittern vor, bei dem die graphische Darstellung des Interpolanten effizient erfolgen kann. Wir werden feststellen, daß die Interpolationstechnik die Verwendung unterschiedlicher Basisfunktionen zuläßt, wobei sich aber nur ein linearer Interpolant fehlerfrei graphisch darstellen läßt. Bemerkenswert sind die Möglichkeiten der einfachen Erzeugung von bilinearen Basen und Näherungen für Greensche

Funktionen als Basen. Die verschiedenen Verfahren werden dann auf ihre Glattheitsvorraussetzungen und Fehlerschranken experimentell untersucht.

Implementatorische Fragen werden in *Kapitel 5* erörtert. Hierzu werden verschiedene Standard–Interpolations–, Durchlauf– und Speicherstrategien vorgestellt. Wir werden daraufhin diese Verfahren auf die Anwendbarkeit auf die Interpolation auf semiregulären Gitter untersuchen und dann geeignete Verfahren auswählen. Weiterhin werden wir auf die Algorithmen zur effizienten graphischen Darstellung des Interpolanten eingehen.

Kapitel 6 beschreibt die konkrete Anwendung des Verfahrens auf die Probleme der Bildverarbeitung und der Generierung eines dreidimensionalen Höhenmodells für Geländedaten zur Erzeugung einer synthetischen Sicht. Es werden die Problemstellungen Standardverfahren erläutert, die Anwendung unseres Verfahrens besprochen und eine Reihe von Simulationsergebnissen präsentiert und bewertet.

In *Kapitel 7* werden dann mögliche Erweiterungen und weitere Anwendungen des Verfahrens aufgezeigt.

Ich möchte allen danken, die mir bei der Erstellung dieser Arbeit geholfen haben. Allen voran bedanke ich mich bei Prof. Christoph Zenger für sein stetiges Interesse an der Arbeit und Dr. Michael Griebel für die intensive Betreuung. Weiterhin bedanke ich bei den übrigen Mitarbeitern des Lehrstuhls, besonders Hans Bungartz, Stefan Zimmer, Tilman Neunhoeffer und Thomas Grauschopf für ihre immer hilfreiche Unterstützung. Ferner möchte ich mich bei meinen Kollegen Anton Frank und Andreas Paul für viele Hilfen und Kommentare während der Arbeit bedanken.

Kapitel 2

Grundlagen

2.1 Computergraphik

2.1.1 Einleitung

Seit der Einführung von Raster–Graphik–Displays Anfang der 80er Jahre und mit der ständigen Weiterentwicklung von Hard- und Software in Graphikbereich wird leistungsfähige Computergraphik einem immer breiteren Publikum zugänglich gemacht. Sie wurde aufgrund der hohen Verarbeitungsfähigkeit von Bilddaten des Menschen zu einer der wichtigsten Art und Weise, mit einem Computer zu kommunizieren.

Mittlerweile ist fast jeder Rechner, vom Personal Computer bis hin zum Supercomputer, mit leistungsfähiger Graphik–Hardware ausgestattet. Graphische Benutzeroberflächen sollen dabei dem Benutzer den Umgang mit dem System erleichtern. Graphikbibliotheken und Graphiktools erlauben dem Programmierer und dem Anwender einen leichten Zugang zur Computergraphik. Der Gebrauch einer nahezu unbegrenzten Menge von Farben ist dabei ebenso möglich, wie die Verwendung von Techniken zur Erstellung dreidimensionaler Animationen.

Die Anwendungen der Computergraphik sind vielfältig. Sie reichen von der wissenschaftlichen Visualisierung, über CAD (Computer Aided Design), der Medizin, der Kartographie, der Meteorologie, elektronischem Publizieren und Werbung bis hin zur Computerkunst. Dies führt dazu, daß wir heute nicht nur im direkten Umgang mit dem Computer, sondern in vielen Bereichen des täglichen Lebens der Computergraphik begegnen.

Hochauflösende Graphikbildschirme und ausgeklügelte graphische Algorithmen erlauben die Darstellung von immer realistischeren Bildern bis hin zur Photoqualität oder in Echtzeit erzeugbaren synthetischen Filmen. Die hierfür nötigen Raytracing– und Rendering–Programme sind mittlerweile breit verfügbar und erfreuen sich auch im PC–Bereich immer größerer Beliebtheit. Ein Ende der Entwicklung von Hard– und Software im Bereich der Computergraphik ist noch lange nicht abzusehen, zumal die Anforderungen an die Graphik immer weiter steigen.

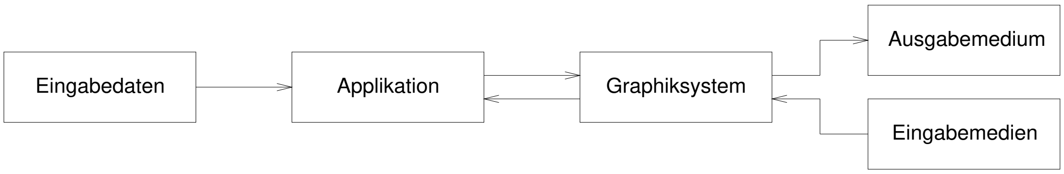

Abbildung 2.1: Schematisches Grundkonzept eines Graphiksystems

Mit Hilfe von leicht bedienbaren Graphikprogrammen kann der Benutzer Computergraphiken selbst leicht erzeugen. Doch die so neu gewonnenen Freiheiten in Farb– und Formgebung stellen ihn jedoch vor eine schier unübersehbare Auswahl an Möglichkeiten bei der Erstellung von Graphiken. Dies kann dazu führen, daß die Aussage eines Bildes durch eine unglückliche Wahl von Form oder Farbe ungewollt verändert wird. Richtlinien und Standards zum effektiven Gebrauch von Computergraphik sind notwendig, aber erst in der Entstehung.

2.1.2 Graphiksysteme

Zur Erstellung dreidimensionaler Computergraphiken sind viele graphische Algorithmen vonnöten, die die Modellierung einer Szene, ihre Beleuchtung oder ihre Darstellung realisieren, die speziell bei komplexen Szenen sehr zeitaufwendig sein können. Viele der graphischen Algorithmen, wie z.B. der Z–Buffer– Algorithmus zur Bestimmung verdeckter Flächen, können in leistungsfähigeren Systemen durch spezielle Graphik–Hardware beschleunigt werden. Unter einem Graphiksystem versteht man die logische Einheit aus Hardware und Software, die für graphische Darstellung in einem Computersystem zuständig ist. Um den Umgang mit dem Graphiksystem zu erleichtern, versucht man, seine Schnittstelle möglichst einheitlich und unabhängig von der verwendeten Hard– und Software zu gestalten.

Ein Graphiksystem bildet somit die Schnittstelle zwischen der Graphik–Applikation und dem Ausgabemedium, z.B. einem Bildschirm oder Drucker. Zusätzlich bietet es normalerweise die Rückkoppelung zur Applikation über Eingabemedien (z.B. Tastatur oder Maus) an, was es dem Benutzer erlaubt, in das Geschehen einzugreifen (Abb. 2.1). Dadurch erlaubt das Graphiksystem hohen Grad an Interaktivität im Umgang mit dem Rechner.

Das Graphiksystem leistet aber noch mehr, außer den reinen Zugang zu Ein- oder Ausgabegeräten anzubieten, es erleichtert dem Benutzer den Umgang mit der Computergraphik auf viele Weisen. Bei der Ausgabe auf Rasterdisplays erledigt es die nötige Scan–Konvertierung (die Umwandlung von Vektordaten in Rasterdaten) mit all den dabei auftretenden Problemen wie Clipping und Aliasing. Es bietet Möglichkeiten, die Daten zu transformieren, z.B. zu drehen, zu skalieren oder von 3–D nach 2–D zu projizieren. Weiterhin kann es die aufwendige Berechnung verdeckter Flächen oder von Lichtquellen und Texturen ausführen.

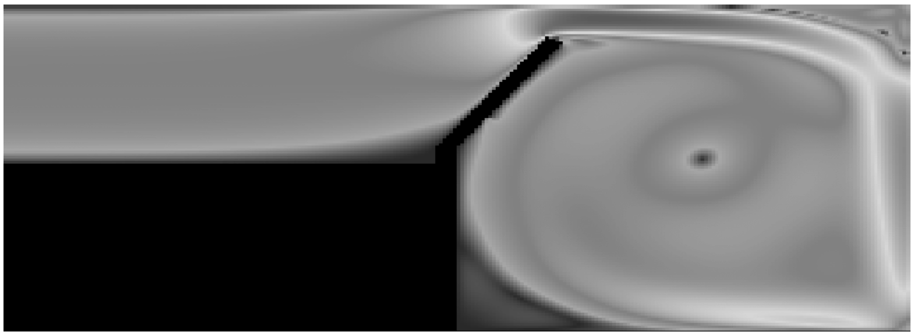

Abbildung 2.2: Visualisierung der Strömung in einer Mikropumpe

Nicht zuletzt stellt das Graphiksystem eine Reihe von Grundprimitiven, wie Linien, Kreise oder Polygone zur Verfügung, die dem Benutzer die Erstellung von Graphiken erleichtern soll. Diese Art der geometrischen Modellierung erlaubt eine schnelle graphische Darstellung und wird vor allem in Echtzeit–Applikationen verwendet. Für CAD–Anwendungen oder zur Erzeugung photorealistischer Bilder werden andere Methoden, wie z.B. CSG (Constructive Solid Geometry) zur geometrischen Modellierung verwendet.

Mit der Zeit haben sich Standards für Graphiksysteme herausgebildet, die in Graphikbibliotheken, wie GKS (Graphical Kernel System [3]), PHIGS (Programmer's Hierarchical Interactive Graphics System [2]) oder GL (Graphics Library [1]), zusammengefaßt werden und somit einem breiten Anwendungsgebiet auf unterschiedlichen Architekturen zur Verfügung stehen. Diese Standards werden allerdings von längst nicht allen Herstellern unterstützt und werden es wohl auch in naher Zukunft nicht sein.

2.2 Wissenschaftliche Visualisierung

2.2.1 Numerische Simulation

Eine wichtige Anwendung von Computergraphik ist die wissenschaftliche Visualisierung. Mit ihrer Hilfe können z.B. die meist sehr umfangreichen Daten aus numerischen Simulationen graphisch dargestellt werden. Im Unterschied zur Visualisierung, der reinen graphischen Darstellung von Daten, will die wissenschaftliche Visualisierung neben der graphischen Darstellung mit Hilfe von Computergraphik spezifische Eigenschaften des Datensatzes hervorheben und so dem Betrachter tiefere Informationen über ein Bild geben. In vielen Fällen kann dies besonders gut durch Bildfolgen geschehen.

Mit der Entwicklung von schnelleren Prozessoren und größeren Speichern gewinnt die numerische Simulation immer mehr an Bedeutung. Mit ihrer Hilfe können komplexe physikalische Probleme mit numerischen Verfahren auf Großrechnern und Rechenanlagen näherungsweise gelöst werden. Sie spart deswegen Ko-

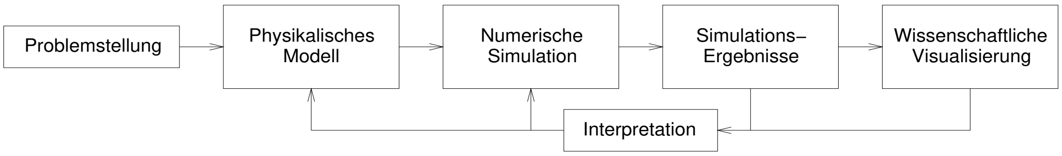

Abbildung 2.3: Numerische Simulation als iterativer Prozeß

sten gegenüber aufwendigen physikalischen Experimenten, schont die Umwelt und kann darüber hinaus Probleme untersuchen, die experimentell überhaupt nicht erforschbar wären, weil ihre Dimensionen zu groß oder zu klein sind.

Ein wichtiges Grundprinzip der numerischen Simulation ist das Prinzip der Diskretisierung. Hierbei werden die mathematischen Gleichungen, die ein physikalisches Phänomen beschreiben, nur an endlich vielen (diskreten) Punkten näherungsweise berechnet. Mit Hilfe dieser diskreten Ergebnisse versucht man dann Aussagen über das Verhalten des Gesamtsystems zu erhalten. Wir werden im nächsten Kapitel genauer auf die mathematischen Eigenschaften der Diskretisierung eingehen.

Zu den heute so simulierbaren Problemen zählen z.B. Strömungs– und Wärmeleitungs–Probleme, Schmelzprozesse und das Kristallwachstum, sowie viele Probleme aus der Statik und Mechanik. Probleme, die zur Zeit noch nicht simulierbar sind, es aber in naher Zukunft wohl sein werden, werden unter dem Begriff 'Grand Challenges' zusammengefaßt. Zu ihnen zählt z.B. die Klimaforschung, die Wettervorhersage und die Simulation von Umwelt– und Luftverschmutzung. Wegen diesen breiten, weit über die Informatik und Mathematik hinausgehenden Anwendungsgebieten erfordert die numerische Simulation eine interdisziplinäre Zusammenarbeit von Informatikern, Mathematikern, Naturwissenschaftlern und Ingenieuren.

Die wissenschaftliche Visualisierung entwickelte sich Ende der 80er Jahre, als die Wissenschaftler erkannt haben, daß sie die Unmengen von Daten aus ihren Simulationen nicht mehr ohne graphische Unterstützung interpretieren konnten. Um möglichst genaue Aussagen über das Verhalten des physikalischen Problems feststellen zu können, ist es notwendig, daß der Wissenschaftler interessante oder problematische Teilbereiche aus dem Datensatz auswählen und genauer betrachten kann (Abb. 2.3).

Die Hauptanwendung der wissenschaftlichen Visualisierung liegt in der Bewertung der Simulationsergebnisse und der Rückkopplung auf die Modellbildung und verwendeten Algorithmen. Es ist wichtig, daß die Interpretation der Daten sowohl aus der direkten (mathematischen) Bewertung der Simulationsergebnisse als auch aus ihrer Visualisierung erfolgen muß. In beiden Fällen können z.B. fehlerhafte Ergebnisse nicht erkannt werden, die im jeweils anderen Fall erkannt worden wären. Weitere Aufgaben der Visualisierung sind die Erkennung von Struktur, Form oder Extrema, die aus den reinen Daten nicht herauslesbar wären.

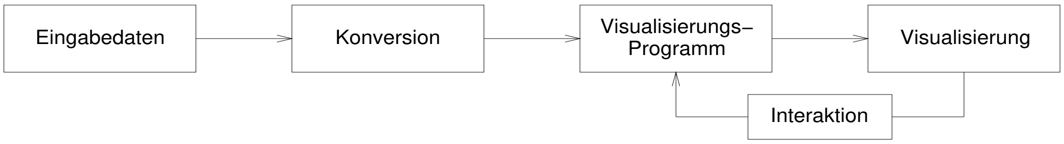

Abbildung 2.4: Arbeitsweise eines Visualisierungsprogramms

Eine weitere wichtige Anwendung der wissenschaftlichen Visualisierung ist der Wissenstransfer in Form von Publikationen und Präsentationen. Ein Bild kann wesentlich mehr Aussagekraft haben als z.B. mathematische Formeln oder Zahlenkolonnen, vor allem für ein fachfremdes Publikum. Die Visualisierung darf sich allerdings hier nicht verselbständigen und zum Selbstzweck werden, sondern soll weiterhin nur ein Hilfsmittel zum Verständnis von Problemen und deren Lösungsmöglichkeiten sein.

2.2.2 Visualisierungsprogramme

Ein Visualisierungsprogramm hat nun die Aufgabe, Daten aus numerischen Simulationen oder Messungen zu visualisiern. Der Benutzer soll dabei die Möglichkeit haben, den Datensatz interaktiv auf eine der schon angesprochenen Weisen zu transformieren (Abb. 2.4). Wichtig sind zusätzliche Auswahlfunktionen, wie Schnittbildungen oder die Darstellung von Isolinien oder –flächen. Weiterhin soll das Visualisierungsprogramm dem Benutzer bei der Farbgebung durch die Bereitstellung geeigneter Farbtabellen helfen. Zum Beispiel bei der Darstellung einer Wärmeverteilung sollte man normalerweise für die Extrema die Farben blau (kalt) und rot (warm) verwenden und die Zwischenwerte zwischen diesen Farben interpolieren, um dem Betrachter das Verständnis des Bildes zu erleichtern. Nicht zuletzt soll das Visualisierungsprogramm Möglichkeiten zur Annotation des erzeugten Bildes geben, ohne die eine Visualisierung ihre Aussage verlieren kann.

Visualisierungsprogramme fallen in zwei Klassen: Spezialisierte Visualisierungsprogramme sind auf eine Anwendung zugeschnitten und beherrschen meist nur die zur Visualisierung dieser Daten nötigen Algorithmen. Ein allgemeines Visualisierungsprogramm bietet eine Vielzahl von Visualisierungsalgorithmen an, die bei verschiedenen Anwendungen ihren Einsatz finden können.

Allgemeine Visualisierungsprogramme unterscheiden sich in ihrer Mächtigkeit. Angefangen von reinen Funktionsplottern reichen sie bis hin zu Application Buildern, welche sich durch graphische Programmierbarkeit auszeichnen und eine Vielzahl von Möglichkeiten zur Datenkonvertierung und Darstellung bieten. Beispiele für solche Programme sind Explorer, AVS und Khoros. Wir wollen uns im Rahmen dieser Diplomarbeit nur auf Probleme von allgemeinen Visualisierungsprogrammen eingehen, wobei aber manche Lösungsansätze auch bei spezialisierten Visualisierungsprogrammen Anwendung finden können.

2.2.3 Probleme bei Visualisierungsprogrammen

Die numerische Simulation stellt hohe Anforderungen an Speicherplatz und Rechenzeit, um die Wirklichkeit möglichst genau beschreiben zu können. Die zu berechnenden Probleme werden deswegen in kleinere und damit leichter berechenbare Teilprobleme aufgeteilt und später wieder zusammengesetzt (divide & impera–Verfahren). Eine solche Aufteilung des Problems erlaubt auch eine Parallelisierung der Simulation auf mehrere Prozessoren oder Rechner.

Die Visualisierung dieser Daten, die meist nur auf einem Rechner stattfindet, stößt dann aber auf große Probleme, weil sie die Menge an u.U. parallel berechneten Daten nur schwer bearbeiten kann. Eine parallele Visualisierung ist momentan aufgrund von Kommunikationsengpässen wegen der Echtzeitanforderungen der Visualisierung zur Zeit noch nicht realisierbar.

Eine weitere Methode, die numerische Mathematik verwendet, um komplexere Probleme berechnen zu können, ist die Verwendung von adaptiven, d.h. sich an das Problem anpassenden, Datenstrukturen. Visualisierungsprogramme müssen diese Daten erst in ihre eigenen nicht–adaptiven Datenstrukturen konvertieren, um mit ihnen arbeiten zu können. Dabei kann es wiederum passieren, daß sie die bei der Konversion in ihre Datenstrukturen entstehende Datenflut nicht verarbeiten können.

Mit der steigenden Komplexität der Daten werden auch die Echtzeitanforderungen an Visualisierungsprogramme zunehmend unerfüllbar, zumal die Daten ja möglichst wirklichkeitsnah und übersichtlich dargestellt werden sollen. Dies liegt vor allem daran, daß bei den meisten Computern Speichergröße und Rechenleistung schneller als die graphische Leistungsfähigkeit ansteigt. Eine Ausnahme bilden speziell für die graphische Darstellung entwickelte Graphik–Workstations und Supercomputer, die sich daher immer größerer Beliebtheit erfreuen.

2.2.4 Lösungsansätze

Durch eine geeignete Kompression kann die Menge der zu speichernden Daten zwar erheblich reduziert werden, doch zu ihrer graphischen Darstellung müssen die Daten wieder dekomprimiert werden. Dies führt dazu, daß so der Gewinn an Speicherplatz wieder verloren geht. Wir werden ein Verfahren vorstellen, bei dem die graphische Darstellung auf komprimierten Daten erfolgen kann.

Die Kompression dieser Daten erfolgt durch Interpolation. Datenwerte, die durch ihre umliegenden Werten mit vorgegebener Genauigkeit interpoliert werden können, müssen nicht abgespeichert werden. Bei der graphischen Darstellung müssen diese Datenwerte bei dem vorgestellten Verfahren nicht explizit dargestellt werden, was nicht nur den benötigten Speicherplatz nicht vergrößert, sondern auch den graphischen Aufwand verringert. Diese Interpolation erfolgt mit hierarchischen Methoden, was eine adaptive Kompression der Daten ermöglicht. Wir werden im nächsten Kapitel genauer auf diese Methoden eingehen.

Zusätzlich kann bei diesem Verfahren die graphische Darstellung der Daten beschleunigt werden, indem der Datensatz nur mit einer vom Anwender vorgegebenen Genauigkeit dargestellt wird. Dies ist zum Beispiel sinnvoll, wenn es das Ziel der Visualisierung ist, globale Strukturen zu erkennen, Feinstrukturen den Aufwand der graphischen Darstellung aber erheblich vergrößern und zum Gesamteindruck nur wenig Information beitragen. Diese Form der graphischen Adaptivität kann aufgrund von fehlerabhängigen oder ortsabhängigen Kriterien erfolgen.

Weiterhin erlaubt das Verfahren durch die hierarchische Sichtweise eine leichte Aufteilung des Datensatzes in Teilgebiete. Falls nur ein Teil eines großen Datensatzes betrachtet werden will, so müssen nur die entsprechenden Teilgebiete im Speicher gehalten werden.

Wir wollen uns im nächsten Kapitel mit den mathematischen Grundlagen für hierarchische Interpolationstechniken befassen und auf einige Standardverfahren eingehen. Wir werden ihre Probleme bei der graphischen Darstellung und deren Lösungsmöglichkeiten aufzeigen.

Kapitel 3

Hierarchische Teilraumzerlegungen

3.1 Mathematische Grundlagen

3.1.1 Diskretisierung

Die Diskretisierung ist der Vorgang, kontinuierliche Funktionen, in diskrete, also endliche, Daten umzuwandeln, die von einem Rechner weiterverarbeitet werden können. Dabei werden vom Rechner nur die diskreten Daten gespeichert; fehlende Funktionswerte können mit Hilfe von Approximationstechniken ermittelt werden.

Mathematisch gesehen wird bei der Diskretisierung eine Funktion u aus einem Funktionenraum V auf eine diskrete Funktion $\hat{u}$ aus einem einen endlich dimensionalen Raum $\hat{V}$ abgebildet. Beide Funktionen sind dabei auf dem gleichen Grundgebiet Ω definiert:

$$u \in V \longrightarrow \hat{u} \in \hat{V}. \tag{3.1}$$

Der Raum $\hat{V}$ wird von einer endlichen Menge von Basisfunktionen ψ_i aufgespannt, die Anzahl der Basisfunktionen entspricht dabei der Dimension des Raumes. Die Gesamtheit der Basisfunktionen heißt die Basis von des Raumes.

$$\hat{V} = span(\psi_1, \ldots \psi_n), \quad n = dim(\hat{V}). \tag{3.2}$$

Ein Teilraum $\hat{T}_i$ ist definiert als der Unterraum von $\hat{V}$, der von der Basisfunktion ψ_i aufgespannt wird:

$$\hat{T}_i = span(\psi_i). \tag{3.3}$$

$\hat{V}$ kann nun als direkte Summe über alle Teilräume dargestellt werden:

$$\hat{V} = \bigoplus_{i=1}^{n} \hat{T}_i. \tag{3.4}$$

Hat man nun einen zur Diskretisierung geeigneten Raum $\hat{V}$ gewählt, so kann man die diskrete Funktion $\hat{u}$ als Summe über die Basiskoeffizienten f_i und zugehörigen Basisfunktionen ψ_i darstellen:

$$\hat{u} = \sum_{i=1}^{n} f_i \cdot \psi_i. \tag{3.5}$$

Die Funktion u wird dann durch die Koeffizienten f_i näherungsweise repräsentiert. Der Fehler $\hat{e}$ der bei der Diskretisierung entsteht, ist definiert als die Differenz zwischen der Originalfunktion und der diskretisierten Funktion:

$$\hat{e} = |u - \hat{u}|\,. \tag{3.6}$$

Man versucht diesen Fehler nun bezüglich einer Norm abzuschätzen. Wir wollen hierzu die Maximums– und die L2–Norm verwenden:

$$\begin{aligned} \|\hat{e}\|_\infty &= \max_{x\in\Omega}(|\hat{e}(x)|), \\ \|\hat{e}\|_2 &= \int_{x\in\Omega} (\hat{e}(x))^2 \; d\Omega. \end{aligned}$$

3.1.2 Interpolation

Zunächst hat die Wahl der Basis keine geometrische Sicht, typisch für die numerische Mathematik ist allerdings eine Ortsinterpretation der Basisfunktionen. Die Koeffizienten f_i entsprechen dann dem Beitrag der Basisfunktion ψ_i an der (Stütz–)Stelle x_i:

$$f_i = f(x_i), \;\; \psi_i = \psi(x_i). \tag{3.7}$$

Die diskretisierte Funktion $\hat{u}$ kann somit dargestellt werden als:

$$\hat{u} = \sum_{i=1}^{n} f(x_i) \cdot \psi(x_i). \tag{3.8}$$

Zur Speicherung von $\hat{u}$ werden nur die Basiskoeffizienten f_i verwendet. Die Funktionswerte von $\hat{u}$ an zwischen den Stützstellen liegenden Punkten werden anhand der Basisfunktionen $\psi(x_i)$ bestimmt.

Bei der Interpolation entspricht die Funktion f der Ausgangsfunktion u (siehe Abb. 3.1 a). Die Funktion $\hat{u}$ stimmt demnach mit u an den Stützstellen x_i exakt überein:

$$f(x_i) = u(x_i). \tag{3.9}$$

Dazu werden die Basisfunktionen $\psi(x_i)$ so gewählt, daß sie an der Stelle $f(x_i)$ den Wert 1 und auf dem Rand des zu $\psi(x_i)$ gehörenden Teilraums den Wert 0 haben.

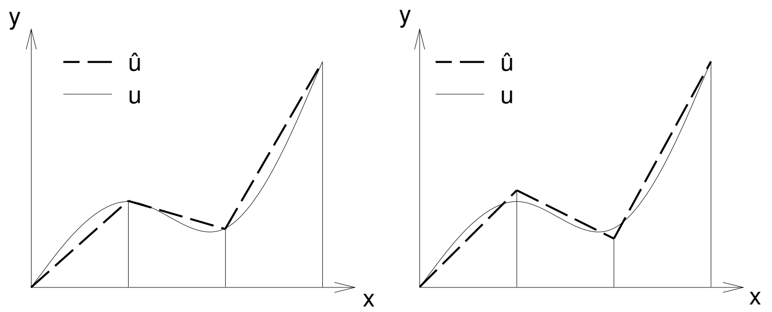

Abbildung 3.1: Lineare Interpolation (a) und Approximation (b)

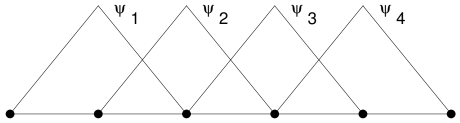

Abbildung 3.2: Lineare Knotenbasis

Je nach Wahl der Basisfunktionen $\psi(x_i)$ hat der Interpolant $\hat{u}$ unterschiedliche Eigenschaften. Zum Beispiel bei der Polynominterpolation wird u unter der Verwendung von n Stützstellen mit Hilfe der Lagrange'schen Interpolationspolynome durch ein Polynom $(n+1)$ten Grades interpoliert. Bei der Spline-Interpolation werden stückweise kubische Polynome verwendet, um u durch eine möglichst glatte Kurve anzunähern.

f kann allerdings auch evtl. günstiger gewählt werden, wie die numerische Approximationsrechnung zeigt. f wird dabei so gewählt, daß der Diskretisierungsfehler in einer vorgegebenen Norm minimal wird (Abb. 3.1 b). Dabei muß $\hat{u}$ an den Stellen x_i nicht notwendigerweise mit u übereinstimmen. Da aber die optimale Bestimmung von f sehr aufwendig ist, wollen wir uns hier auf die in der Regel leichter berechenbare Interpolation beschränken.

Im einfachsten Fall wählt man zur Interpolation im Eindimensionalen als Basisfunktionen lineare Knotenbasen $\psi_k(x_i)$ auf einem regulären Gitter G_h mit Maschenweite h (Abb. 3.2 a). Die Basisfunktionen sind dabei für alle x_i gleich gestaltet:

$$\psi_k(x_i) = \begin{cases} 1 + \frac{x_i}{h} & , \text{ falls } -h \leq x_i \leq 0 \\ 1 - \frac{x_i}{h} & , \text{ falls } 0 \leq x_i \leq h \\ 0 & sonst \end{cases} \tag{3.10}$$

Durch die Summenbildung aus Formel 3.8 entsteht ein stückweise linearer Interpolant. Für jeden Punkt $x \in \Omega$ mit $x_i \leq x \leq x_{i+1}$ gilt:

$$\frac{\hat{u}(x) - u(x_i)}{u(x_{i+1}) - u(x_i)} = \frac{x - x_i}{x_{i+1} - x_i} \tag{3.11}$$

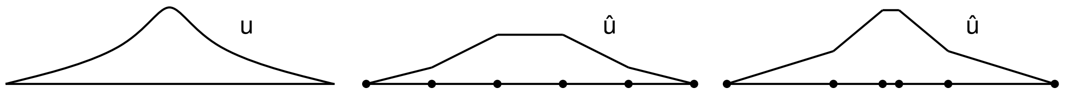

Abbildung 3.3: Interpolation von u auf regulärem und adaptivem Gitter

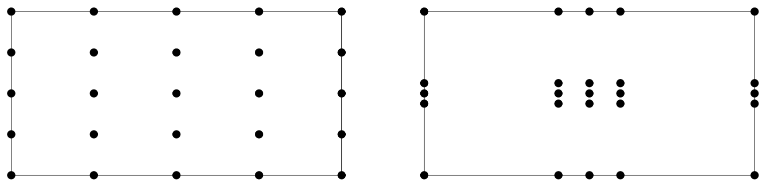

Abbildung 3.4: Zweidimensionales reguläres und adaptives volles Gitter

Das zugrundeliegende Gitter muß nun nicht gleichförmig gewählt werden, sondern kann an adaptiv gewählt werden, d.h. an interessanteren Stellen feiner auflösen, als an uninteressanten Stellen. Wie man in Abb. 3.3 sieht, kann eine geeignete Auswahl der Gitterpunkte die Approximation von u verbessern.

d-lineare Knotenbasen für höherdimensionale Funktionen $\hat{u}^{(d)}$, können leicht durch Produktbildung von eindimensionalen Basisfunktionen hergeleitet werden:

$$\psi_k^{(d)}(x_1, \ldots, x_d) = \prod_{j=1}^{d} \psi_k(x_j). \tag{3.12}$$

Ist n_j die Zahl der Gitterpunkte in Richtung j, so läßt sich $\hat{u}^{(d)}$ schreiben als:

$$\hat{u}^{(d)} = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_j} \psi_k^{(d)}(x_{i_1}, \ldots, x_{i_d}) \cdot u^{(d)}(x_{i_1}, \ldots, x_{i_d}). \tag{3.13}$$

Hierbei entstehen d-dimensionale volle bzw. adaptive Gitter. Wie man im Beispiel für den zweidimensionalen Fall (Abb. 3.4) sieht, muß man um ein Teilgebiet (im obigen Beispiel um den Mittelpunkt) genauer auflösen zu können, auch entlang der Horizontalen und Vertikalen genauer auflösen, was in einer erhöhten Zahl von Punkten resultiert, die eigentlich nicht benötigt werden.

Im folgenden Abschnitt werden wir sehen, daß die Verwendung hierarchischer Basen eine effizientere Darstellung von u in solchen Fällen ermöglicht.

3.1.3 Hierarchische Basen

Die Verwendung hierarchischer Basen zur Interpolation ist nicht neu. Yserentant berichtet z.B. in [26] über eine Arbeit von Faber aus dem Jahre 1909, in der diese Idee vorgestellt wird. Wir wollen hierzu eine Hierarchie auf den Teilräumen definieren, die Teilräume werden dabei über ihren Level l indiziert:

$$\hat{T}_i \to \hat{T}_{l,i}. \tag{3.14}$$

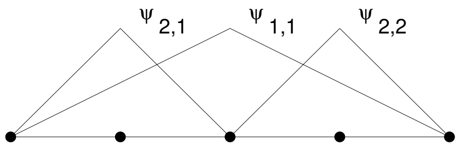

Abbildung 3.5: Lineare hierarchische Basis

$\hat{T}_{l,i}$ heißt dabei hierarchisch höher als $\hat{T}_{m,i}$ falls $l < m$. Sei nun l_{max} ein vorgegebener maximaler Level und die Zahl der Teilräume auf einem Level l gleich n_l. Der Funktionenraum $\hat{V}$ läßt sich jetzt schreiben als:

$$\hat{V} = \bigoplus_{l=1}^{l_{max}} \bigoplus_{i=1}^{n_l} \hat{T}_{l,i}. \tag{3.15}$$

Die Teilräume $\hat{T}_{l,i}$ werden dabei von den entsprechend indizierten Basisfunktionen $\psi_{l,i}$ aufgespannt:

$$\hat{T}_{l,i} = span(\psi_{l,i}). \tag{3.16}$$

Liegt $\cup_{l=0}^{\infty} \cup_{i=1}^{n_l} \hat{T}_{l,i}$ in V dicht, so kann u exakt durch die unendliche Summe über die Level dargestellt werden als:

$$u = \sum_{l=0}^{\infty} \sum_{i=1}^{n_l} v_{l,i} \cdot \psi_{l,i}. \tag{3.17}$$

Die Koeffizienten $v_{l,i}$ heißen dabei hierarchische Überschüsse. In 3.5 sieht man ein Beispiel für eindimensionale hierarchische Basen.

Die hierarchischen Überschüsse können bei bestimmten Basen mit Hilfe von Interpolationssternen $I_{(l,i),h}$ berechnet werden, wir werden später in einigen bestimmten Fällen diese Interpolationssterne explizit angeben:

$$v_{l,i} = I_{(l,i),h} \quad u_{l,i} \tag{3.18}$$

Zur Interpolation wird nun nur eine endliche Menge von Teilräumen und Basisfunktionen verwendet. Wir wollen diese Menge allgemein durch eine Maske $M_{l,i}$ definieren. $M_{l,i}$ enthält alle Kombinationen aus Leveln und Indizes, die zur Interpolation verwendet werden. Der Interpolant $\hat{u}$ läßt sich somit darstellen als:

$$\hat{u} = \sum_{l,i \in M_{l,i}} v_{l,i} \cdot \psi_{l,i}. \tag{3.19}$$

Der Fehler $\hat{e}_{M_{l,i}}$, der bei der Interpolation gemacht wird, ist somit:

$$\hat{e}_{M_{l,i}} = \sum_{l,i \notin M_{l,i}} |v_{l,i} \cdot \psi_{l,i}|. \tag{3.20}$$

Üblicherweise wird der Fehler e mit Hilfe der Dreiecksungleichung abgeschätzt:

$$\|\hat{e}_{M_{l,i}}\| = \| \sum_{l,i \notin M_{l,i}} v_{l,i} \cdot \psi_{l,i} \| \leq \sum_{l,i \notin M_{l,i}} \|v_{l,i} \cdot \psi_{l,i}\| \leq \sum_{l,i \notin M_{l,i}} |v_{l,i}| \cdot \|\psi_{l,i}\|. \tag{3.21}$$

Falls sich $\|\psi_{l,i}\|$ bestimmen läßt, erhält man eine Schranke für den Interpolationsfehler über eine Abschätzung der hierarchischen Überschüsse.

$$|v_{l,i}| \leq \int_{\Omega} \|u\| \cdot \|\psi_{l,i}\| d\Omega. \tag{3.22}$$

Der Aufwand N für die Generierung des Interpolanten $\hat{u}$ ist definiert als die Zahl der benötigten Teilräume:

$$N = dim(\hat{V}) = \left| \sum_{l,i \in M_{l,i}} T_{l,i} \right|. \tag{3.23}$$

In gewisser Weise ist es das Ziel der Interpolation den Aufwand bei möglichst kleinem Fehler möglichst gering zu halten:

$$\frac{N}{\|\hat{e}_{M_{l,i}}\|} = min. \tag{3.24}$$

3.1.4 Auswahl der Basisfunktionen

Idealerweise möchte man den Interpolationsfehler $\hat{e}_{M_{l,i}}$ begrenzt halten:

$$\|\hat{e}_{M_{l,i}}\| \leq \epsilon. \tag{3.25}$$

Der Fehler, der durch das Weglassen einer Basisfunktion entsteht sei $\hat{e}_{m,j}$:

$$\hat{e}_{m,j} = \left| \hat{e}_{M_{l,i}} - \hat{e}_{(M_{l,i} \setminus \{m,j\})} \right|. \tag{3.26}$$

Die Maske $M_{l,i}$ wird dann idealerweise so gewählt, daß für alle Elemente aus $M_{l,i}$ der Fehler um mindestens ϵ reduziert wird.

$$\forall\ m,j \in M_{l,i} : \|\hat{e}_{m,j}\| > \epsilon. \tag{3.27}$$

Diese wäre im Sinne der Interpolation eine optimale Auswahl von Basisfunktionen zur Interpolation. Im Normalfall ist diese Wahl jedoch zu aufwendig, bzw. nicht möglich. Zur Abschätzung von $\epsilon_{l,i}$ kann man jedoch den hierarchischen Überschuß $v_{l,i}$ verwenden. Dadurch läßt sich $M_{l,i}$ explizit angeben:

$$M_{l,i} = \{m, j : v_{m,j} > \epsilon\}. \tag{3.28}$$

Diese Abschätzung ist korrekt, falls gilt:

$$\|\epsilon_{l,i}\| \leq v_{m,j}. \tag{3.29}$$

Levelweises Vorgehen

Die einfachste Form, $M_{l,i}$ zu wählen, ist die Aufnahme von Basisfunktionen nur bis zu einem festen Level n:

$$M_{l,i} = \{m, j : m \leq n\}. \tag{3.30}$$

Damit läßt sich $\hat{u}$ schreiben als:

$$\hat{u} = \sum_{l=1}^{n} \sum_{i=1}^{n_l} v_{l,i} \cdot \psi_{l,i}. \tag{3.31}$$

Der Fehler $\hat{e}$ ist dann definiert als:

$$\hat{e} = \left| \sum_{n+1}^{\infty} \sum_{i=1}^{n_l} v_{l,i} \cdot \psi_{l,i} \right|. \tag{3.32}$$

Dieses Vorgehen führt nur zu optimalen Ergebnissen, falls der Fehler $\hat{\epsilon}$ für alle Basisfunktionen auf einem Level gleich ist und mit jedem Level abfällt.

Adaptives Vorgehen

Falls man einen maximalen Level l_{max} vorgibt, und falls für alle Punkte $\{m, j\} mit j > l_{max}$ gilt:

$$\hat{\epsilon}_{m,j} \leq \epsilon, \tag{3.33}$$

so kann man $M_{l,i}$ adaptiv wählen als:

$$M_{l,i} = \{m, j : \hat{\epsilon}_{m,j} > \epsilon \text{und} m < n\}. \tag{3.34}$$

Eine weitere Variante zur adaptiven Wahl von $M_{l,i}$ entsteht durch die Einführung einer Relation $s(l, i)$ zwischen den Teilräumen der Level l und $l + 1$:

$$s : \left\{ \begin{array}{ccl} \mathbb{N}^2 & \to & \mathcal{P}^{\mathbb{N}} \\ (l, i) & \to & \{j_1, \ldots, j_{s_{l,i}}\}, \ 1 \leq i, j_k, s_{l,i} \leq n_l \end{array} \right. \tag{3.35}$$

Man wählt dann $M_{l,i}$ so, daß wenn ein Punkt $\{m, j\}$ nicht zu $M_{l,i}$ gehört, dann auch nicht alle Punkte $s(l, i)$. Dadurch läßt sich $M_{l,i}$ ebenfalls explizit angeben und es gilt:

$$M_{l,i} = \{m, j : \exists \ s(m, j) : v_{m+1,s(m,j)} \geq \epsilon\} \tag{3.36}$$

Diese Beziehungen kann man mit Hilfe eines Baumes auftragen. Die Elemente der Menge $s(l, i)$ sind dabei die Söhne des Knotens $\{l, i\}$. Beide Methoden führen nur unter gewissen Glattheitsvorraussetzungen für die Ausgangsfunktion u zu optimalen Ergebnissen.

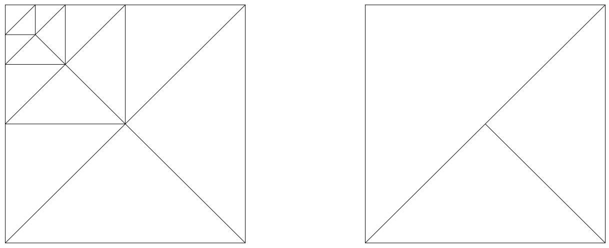

Abbildung 3.6: Zulässige und unzulässige Triangularisierung

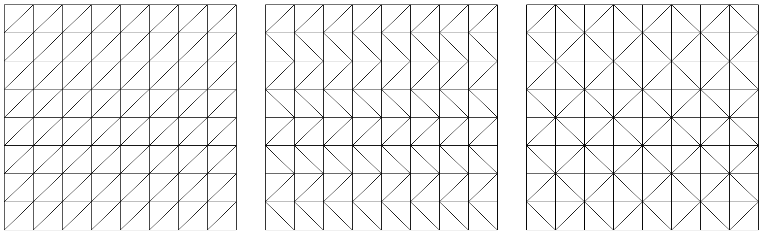

Abbildung 3.7: Verschiedene uniforme Triangularisierungen auf vollen Gittern

3.2 Triangularisierungen

Wir wollen uns im folgenden auf den zweidimensionalen Fall beschränken. Für die Darstellung zweidimensionaler Funktionen ist das einfachste zweidimensionale graphische Grundelement, das ein Graphiksystem zur Verfügung stellt, ein Polygon (siehe 2.1.2). Das Graphiksystem kann aber sinnvollerweise nur Polygone darstellen, deren Eckpunkte alle in einer Ebene (coplanar) liegen. Die einfachsten Polygone, für die diese Bedingung immer erfüllt ist, sind Dreiecke. Polygone, deren Eckpunkte nicht coplanar sind, können zwar in Dreiecke unterteilt werden, doch normalerweise kann das Graphiksystem diese Arbeit nicht erledigen. Wir betrachten deshalb die Zerlegung von Gebieten in Dreiecke.

Eine Triangularisierung T eines zweidimensionalen Gebiets $\Omega^{(2)}$ ist eine Menge von Dreicken, so daß Ω die disjunkte Vereinigung dieser Dreiecke ist. Je mehr Dreiecke verwendet werden, desto genauer ist normalerweise die Darstellung des Gebiets.

Eine Triangularisierung T heißt zulässig (siehe auch Abb. 3.6), wenn gilt:

1. Ist der Schnitt zweier Dreiecke genau ein Punkt, so ist dieser Punkt ein gemeinsamer Eckpunkt von beiden Dreiecken.
2. Besteht der Schnitt zweier Dreiecke aus mehr als einem Punkt, so ist dieser Schnitt eine gemeinsame Kante von beiden Dreiecken.

Eine Triangularisierung heißt regelmäßig, wenn dabei alle Dreiecke kongruent sind. Eine Triangularisierung heißt uniform, wenn dabei die Fläche aller Dreiecke gleich ist.

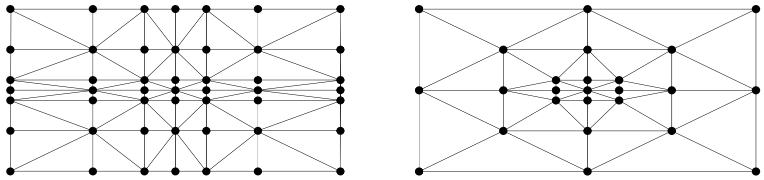

Abbildung 3.8: Triangularisierungen auf adaptiven Gittern

Abbildung 3.9: Reguläre und irreguläre Verfeinerung

Zur graphischen Darstellung von Interpolanten wollen wir nur Triangularisierungen betrachten, bei denen die Eckpunkte aller Dreiecke auf Gitterpunkten liegen. Vereinfachenderweise sprechen wir deswegen von Triangularisierungen auf Gittern.

Triangularisierungen auf vollen Gittern sind uniform. Je vier Gitterpunkte bilden ein Rechteck, jedes Rechteck kann in zwei Dreiecke aufgeteilt werden. Diese Aufteilung kann nach verschiedenen Schemata erfolgen, Abb. 3.7 zeigt drei der geläufigsten Aufteilungen.

Dieses Verfahren funktioniert auch für adaptive volle Gitter, wobei die entstehende Triangularisierung nicht mehr regelmäßig ist, wie man in Abb. 3.8 a sieht. Die erhöhte Zahl von Punkten entlang den Koordinatenachsen um das verfeinerte Gebiet schlägt sich auch in einer erhöhten Zahl von Dreiecken bei der Triangularisierung wieder. Idealerweise möchte man eine Triangularisierung wie in Abb. 3.8 b angedeutet, die im weniger interessanten Gebiet keine zusätzlichen Dreiecke benötigt. Diese Form der Adaptivität wird durch lokale Verfeinerungen erreicht.

Ist eine Triangularisierung T gegeben, so erhält man eine feinere Triangularisierung, indem Dreiecke weiter unterteilt werden. In der Finite–Elemente–Methode unterscheidet man zwischen regulären und irregulären Verfeinerungen (Abb. 3.9). Bei der regulären Verfeinerung ist es garantiert, daß die neu entstehenden Dreiecke ähnlich zum Ausgangsdreieck sind, während bei der irregulären Verfeinerung innere Winkel verändert werden können.

Problematisch bei der Verfeinerung ist das Einhalten der Zulässigkeit einer Triangularisierung. Unter Umständen kann z.B. eine lokale Verfeinerung Auswirkungen auf das ganze Gebiet haben. Normalerweise sind aufwendige Datenstrukturen und Algorithmen notwendig, um die Zulässigkeit der Triangularisierungen sicherzustellen.

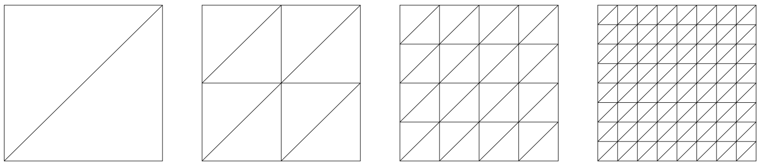

Abbildung 3.10: Sukzessive reguläre Verfeinerung

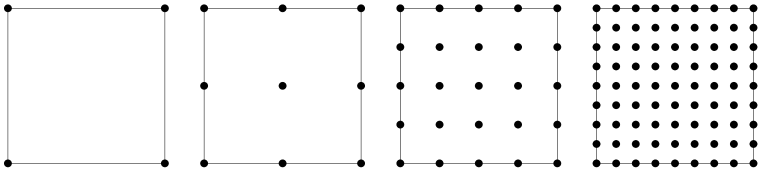

Abbildung 3.11: Reguläre volle Gitter mit Maschenweiten $h = 1, \frac{1}{2}, \frac{1}{4}$ und $\frac{1}{8}$

3.3 Gitterhierarchien

Ein Gitter G auf einem Gebiet Ω ist eine Menge von diskreten Punkten innerhalb oder auf dem Rand von Ω. Die Eckpunkte aller Dreiecke einer beliebigen Triangularisierung definieren ein Gitter.

3.3.1 Reguläre Gitter

Wir wählen nun als unser Ausgangsgebiet das Einheitsquadrat $\Omega^{(2)} = [0,1]^2$. Die Aufteilung von $\Omega^{(2)}$ in zwei Dreiecke entlang einer Diagonale stellt eine Triangularisierung dar. Verfeinert man nun diese Dreiecke sukzessive regulär, so führt das zu den Triangularisierungen aus Abb. 3.10. Die so entstehenden Gitter G_l sind voll, regulär und haben eine Maschenweite $h = 2^{-l}$ (Abb. 3.11).

$$G_l = \bigcup_{0 \le i,j \le 2^l+1} (i \cdot 2^{-l}, j \cdot 2^{-l}). \tag{3.37}$$

Das reguläre Gitter G_l läßt sich nun auf die folgende Weise in Teilgitter $\bar{G}_{mn}$, $0 \le m, n \le l$, unterteilen (Abb. 3.12):

$$\bar{G}_{mn} = \begin{cases} \bigcup\limits_{0 \le i \le 2^n+1, 0 \le j \le 2^m+1} ((2i+1) \cdot 2^{-m}, (2j+1) \cdot 2^{-n}) & \text{, falls } m, n > 0, \\ \bigcup\limits_{0 \le i \le 1, 0 \le j \le 2^m+1} (i, (2j+1) \cdot 2^{-n}) & \text{, falls } m = 0, n > 0, \\ \bigcup\limits_{0 \le i \le 2^m+1, 0 \le j \le 1} ((2i+1) \cdot 2^{-m}, j) & \text{, falls } m > 0, n = 0, \\ \bigcup\limits_{0 \le i \le 1, 0 \le j \le 1} (i, j) & \text{, falls } m = 0, n = 0. \end{cases} \tag{3.38}$$

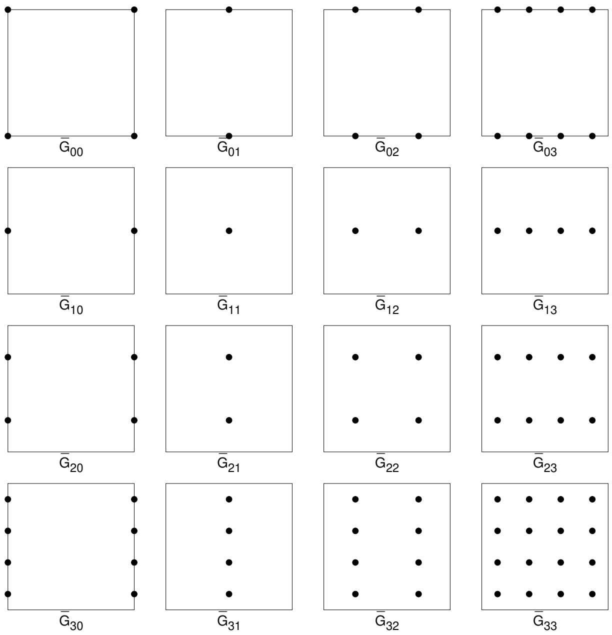

Abbildung 3.12: Die Teilgitter $\bar{G}_{ij}, 0 \leq i, j \leq 3$

Reguläre Gitter vom Level l finden sich in diesem Schema als Vereinigung aller Teilgitter mit Indizes $i, j \leq l$ wieder:

$$G_l = \bigcup_{i=1}^{l} \bigcup_{j=1}^{l} \bar{G}_{ij}. \tag{3.39}$$

Auf diesem Gitterschema läßt sich eine Ordnungsrelation '<' definieren. Hierzu bestimmen wir den Level eines Teilgitters $\bar{G}_{ij}$:

$$level(\bar{G}_{ij}) = \min\{l : \bar{G}_{ij} \in G_l\}. \tag{3.40}$$

Damit gilt dann:

$$\bar{G}_{ij} < \bar{G}_{kl} \iff level(\bar{G}_{ij}) < level(\bar{G}_{kl}) \tag{3.41}$$

In Abb. 3.13 a findet sich diese Ordnung schematisch wieder.

3.3.2 Dünne Gitter

Auf dem Teilgitterschema 3.12 kann man nun verschiedene alternative Ordnungen wählen. Reguläre Gitter entsprechen einer quadratischen Aufteilung dieses Schemas. Eine weitere wichtige Ordnung entsteht durch eine diagonale Aufteilung des Schemas, wie sie in Abb. 3.13 b gezeigt wird:

$$G_{ij} < G_{kl} \iff (i < k) \wedge (j < l) \wedge (i + j < k + l) \tag{3.42}$$

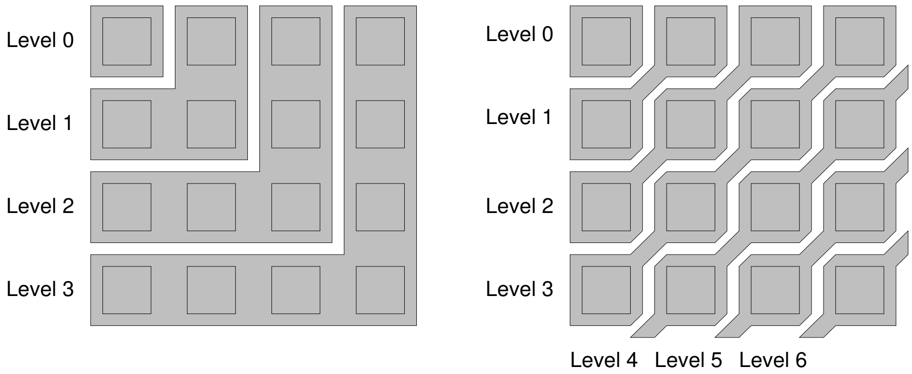

Abbildung 3.13: Level bei (a) regulären, (b) dünnen Gittern

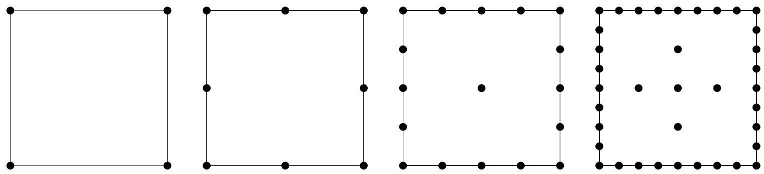

Abbildung 3.14: Dünne Gitter vom Level 0 bis 3

Diese Ordnung führt zu sogenannten dünnen Gittern $\tilde{G}_l$ (Abb. 3.14):

$$\tilde{G}_l = \bigcup_{i+j \leq l} \bar{G}_{ij} \tag{3.43}$$

Ein reguläres dünnes Gitter vom Level l beinhaltet offentsichtlich weniger Punkte als ein reguläres volles Gitter vom gleichen Level, ist also inhärent schon adaptiv. Werden Standard–Adaptivitätstechniken auf reguläre dünne Gitter angewandt, führt das dann zu adaptiven dünnen Gittern. Dünne Gitter werden vor allem zur effizienten Berechnung von elliptischen partiellen Differentialgleichungen verwendet. Wir wollen auf ihre Eigenschaften in 3.4.2 eingehen. Eine ausführliche Beschreibung von dünnen Gittern und ihren Eigenschaften findet sich in [28] und [8]:

3.4 Hierarchische Teilraumzerlegungen

Mit Hilfe des Teilgitterschemas 3.12 wollen wir eine Hierarchie auf den Teilräumen definieren. $\bar{T}_{mn}$ sei dabei der Unterraum von V^k, der an allen Gitterpunkten hierarchisch höherer Gitter als $\bar{G}_{mn}$ verschwindet. Diese Teilraumschemata unterscheiden sich je nach Art der verwendeten Gitter und Basisfunktionen. Wir werden sie zunächst für die zwei Fälle der eben vorgestellten dünnen und regulären Gittern betrachten.

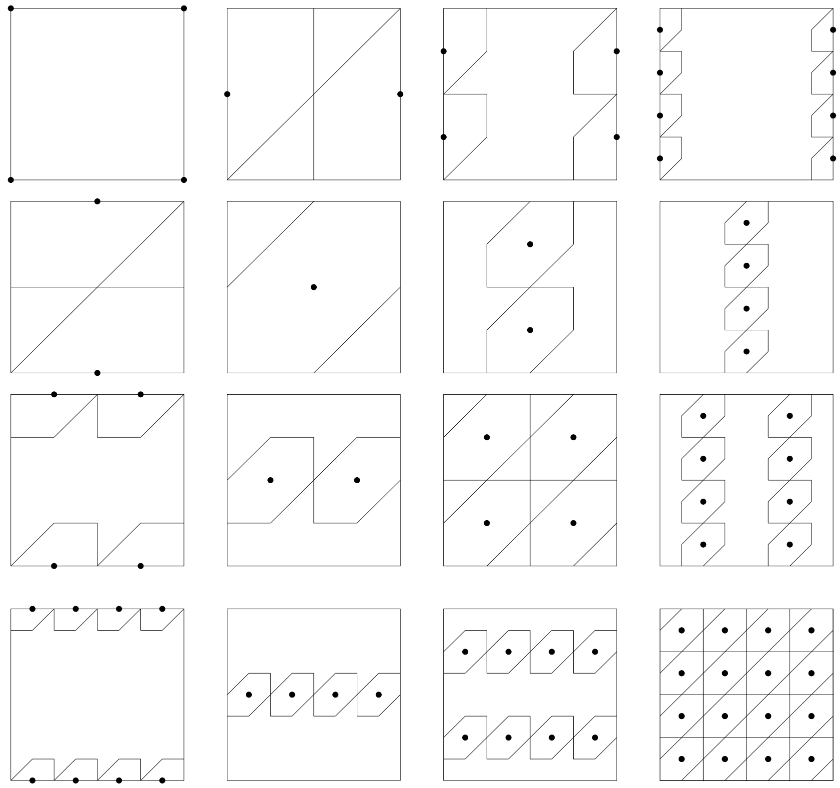

Abbildung 3.15: Teilraumschema für dreieckige Finite Elemente

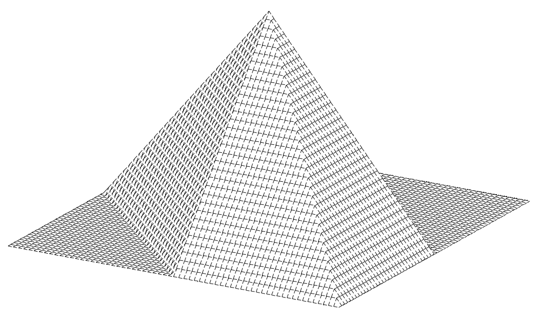

Abbildung 3.16: Basisfunktion für dreieckige Finite Elemente

3.4.1 Dreieckige Finite Elemente

Die Finite–Elemente–Methode arbeitet mit einer ganze Reihe von Elementen, wir werden uns in dieser Arbeit mit der Sicht auf die graphische Darstellung auf dreieckige Finite Elemente beschränken. Dabei arbeitet die Finite–Element–Technik mit Teilräumen auf regulären Gittern. Innerhalb eines Levels überlappen diese Teilräume, wie man in Abb. 3.15 sieht. Auf diesen Teilräumen definiert man nun Basisfunktionen für dreieckige Finite Elemente (siehe Abb. 3.16):

$$\psi(x,y) = \begin{cases} 1 - \frac{x}{h_x} & \text{, falls } 0 \leq x \leq 1 \wedge x \geq y \\ 1 - \frac{y}{h_y} & \text{, falls } 0 \leq y \leq 1 \wedge y \geq x \\ 1 - \frac{y}{h_y} + \frac{x}{h_x} & \text{, falls } -1 \leq x \leq 0 \wedge 0 \leq y \leq 1 \\ 1 + \frac{x}{h_x} & \text{, falls } -1 \leq x \leq 0 \wedge x \geq y \\ 1 + \frac{y}{h_y} & \text{, falls } -1 \leq y \leq 0 \wedge y \geq x \\ 1 + \frac{y}{h_y} - \frac{x}{h_x} & \text{, falls } 0 \leq x \leq 1 \wedge -1 \leq y \leq 0 \\ 0 & \text{, sonst} \end{cases} \tag{3.44}$$

Diese Basisfunktionen entstehen bei linearer Interpolation an den Rändern der Dreieckselemente. Hierzu werden lineare Knotenbasen an den Eckpunkten des Dreiecks aufgesetzt. Wir werden die zugehörigen Interpolationssterne in 4.2.1 angeben.

Gelten für u die Glattheitsbedingungen:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \leq \epsilon,$$

so gilt für die Ordnung des Interpolationsfehlers (siehe [6]):

$$\|\hat{e}\| = O(h^2).$$

Der Aufwand ist hierbei:

$$\|N\| = O(2^{2l}).$$

Dies bedeutet, daß man etwa die doppelte Menge von Teilräumen verwenden muß, um den Interpolationsfehler zu halbieren.

Diese Basen werden in vielen Finite-Elemente-Pogrammpaketen, wie zum Beispiel PLTMG [4], MGGHAT [19] oder KASKADE [11] zur Lösung partieller Differentialgleichungen verwendet. Diese Verfahren arbeiten auf adaptiven regulären Gittern. Ein wichtiger Gesichtspunkt dabei ist eine geeignete Wahl von Adaptivitätskriteren und Datenstrukturen zur Sicherstellung der Zulässigkeit der entstehenden Triangularisierungen.

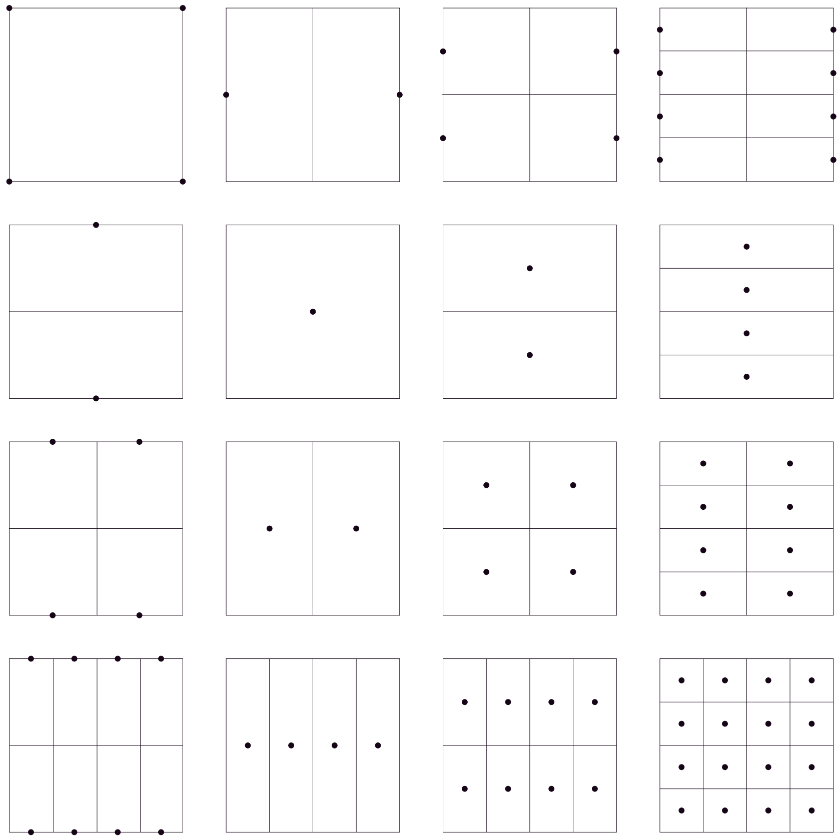

Abbildung 3.17: Teilraumschema bei dünnen Gittern

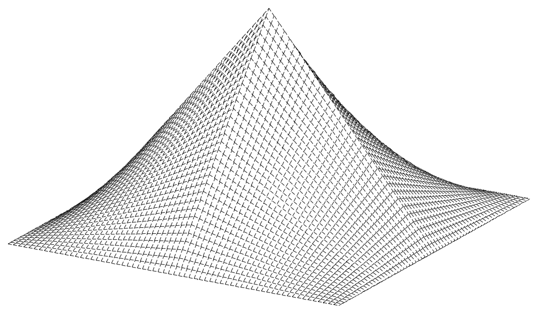

Abbildung 3.18: Bilineare Basisfunktion

3.4.2 Dünne Gitter

Nach Zenger [28] arbeitet das Teilraumschema für dünne Gitter (Abb. 3.17) ebenfalls mit überlappenden Trägern auf einem Level. Jeder Teilraum $\bar{T}_{mn}$ erstreckt sich dabei über das ganze Grundgebiet.

Als Basisfunktionen verwendet man bilineare hierarchische Basen. Diese entstehen analog zu bilinearen Knotenbasen durch Produktbildung von eindimensionalen hierarchischen Basen aus 3.5. Diese Basisfunktionen können durch folgenden Standard-9-Punkte Interpolationsstern erzeugt werden:

$$I_{(l,i),h} = \begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \end{bmatrix}_{(x,y),h} . \tag{3.45}$$

Wiederum können wir den Interpolationsfehler im regulären Fall abschätzen. Gelten für u die Glattheitsbedingungen:

$$\frac{\partial^4 u}{\partial x^2 \partial x^y} \le \epsilon, \tag{3.46}$$

so gilt für die Ordnung des Interpolationsfehlers (siehe [8]):

$$\|\hat{e}\| = O(h^2 \cdot ld(h^{-1})). \tag{3.47}$$

In Fall der dünnen Gitter ist allerdings der Aufwand nur:

$$\|N\| = O(l \cdot 2^l). \tag{3.48}$$

Dies bedeutet, daß man bei einem nur geringfügig schlechteren Interpolationsfehler deutlich weniger Basisfunktionen zur Erstellung des Interpolanten benötigt.

Aufgrund des Überlappens der Träger entspricht der Dünngitterinterpolant bei regulären dünnen Gittern einer stückweise bilinearen Funktion auf dem regulären vollem Gitter. Dies läßt sich schon einsehen, indem man nur die Überschneidungen der Teilräume $\bar{T}_{1,l}$ und $\bar{T}_{l,1}$ betrachtet. Der Dünngitterinterpolant auf adaptiven dünnen Gittern entspricht einem bilinearen Interpolant einem adaptiven vollen Gitter.

3.4.3 Nachteile der vorgestellten Verfahren

Wir wollen nun als $\hat{u}_g$ die Funktion definieren, die bei der graphischen Darstellung von $\hat{u}$ entsteht. Im Eindimensionalen läßt sich $\hat{u}$ bei linearer Interpolation durch einen Linienzug durch die Punkte (x_i, f_i) exakt darstellen. Analog dazu muß der Interpolant im Zweidimensionalen durch Dreiecke zusammensetzbar sein, wenn die graphische Darstellung des Interpolanten exakt mit ihm übereinstimmen will. Dies entspricht einer linearen Interpolation im zweidimensionalen (wie sie z.B. bei Verwendung von Dreieckigen Finiten Elementen geschieht).

Bei anderen Interpolationstechniken wird bei der graphischen Darstellung zusätzlich zum Interpolationsfehler $\hat{e}$ ein graphischer Fehler $\hat{e}_g$ gemacht:

$$\hat{e}_g = |\hat{u} - \hat{u}_g|. \tag{3.49}$$

Dies ist unproblematisch, falls die Ordnung des graphischen Fehlers kleiner als die des Interpolationsfehlers ist. Wir wollen aber im folgenden die Interpolationstechnik so wählen, daß der graphische Fehler $\hat{e}_g = 0$ ist.

Die Teilraumschemata von dünnen und regulären Gittern arbeiten mit Überlappungen der Teilräume innerhalb eines Levels. Bei der graphischen Darstellung des Interpolanten führt das dazu, daß die entsprechenden Dreiecke nicht unabhängig voneinander gezeichnet werden können, wodurch diese erheblich erschwert wird und nicht mehr effizient ausführbar ist. Wir werden im folgenden Kapitel ein Teilraumschema vorstellen, das ohne Überlappungen auf einem Level arbeitet.

Kapitel 4

Semireguläre Gitter

4.1 Teilraumzerlegung

Wir wollen wir uns im folgenden wieder auf den zweidimensionalen Fall beschränken und das Gebiet $\Omega^{(2)}$ rechteckig wählen. Man kann nun $\Omega^{(2)}$ sukzessiv irregulär verfeinern. Man beginnt dabei, wie bei der sukzessiven regulären Verfeinerung mit der Aufteilung des Gesamtgebiets in zwei Dreiecke entlang einer Diagonalen. Bei der Aufteilung von rechtwinkligen Dreiecken führt das ebenfalls zu ähnlichen Dreiecken, vorrausgesetzt man unterteilt immer den rechten Winkel. Führt man diese Verfeinung für die entstehenden Dreiecke rekursiv weiter durch, so entstehen die irregulären Triangularisierungen aus (Abb. 4.1).

Betrachtet man nun die zu diesen Triangularisierungen zugehörigen Gitter, so sieht man, daß bei jedem zweiten Schritt reguläre Gitter entstehen. Die in den Zwischenschritten entstehenden Gitter entsprechen um 45 Grad gedrehten regulären Gitter, wobei das Grundgebiet $\Omega^{(2)}$ festgehalten wird (Abb. 4.2). Wir wollen diese Gitter *semireguläre Gitter* nennen. Entsteht ein semireguläres Gitter durch die Verfeinerung eines regulären Gitters vom Level l, so wollen wir ihm den Level $l + \frac{1}{2}$ zuordnen. Dementsprechend wollen wir den Levelindex l in der Folge $0, \frac{1}{2}, 1, \frac{3}{2}, \ldots$ von Semileveln laufen lassen.

Vereinfachend und zur besseren Unterscheidung wollen wir bei der Interpolation unter Verwendung des Zwischenschritts über semireguläre Gitter von einer Interpolation auf semiregulären Gittern sprechen.

Mißt man die Gitterweite semiregulärer Gitter nicht diagonal, sondern wie bei regulären Gittern entlang den Koordinatenachsen, gilt für die Maschenweiten von semiregulären Gittern:

$$h_l = 2^{-\lfloor l \rfloor} \tag{4.1}$$

Diese Definition ist sinnvoll, da wir später die Interpolationssterne für semireguläre Gitter nicht diagonal, sondern – wie üblich – entlang den Koordinatenachsen aufsetzen wollen.

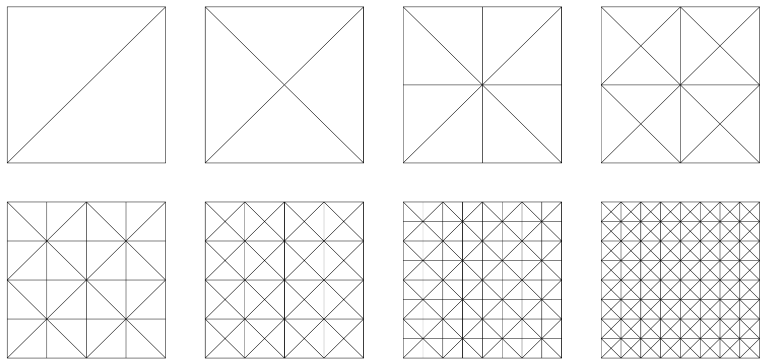

Abbildung 4.1: Sukzessive irreguläre Verfeinerung

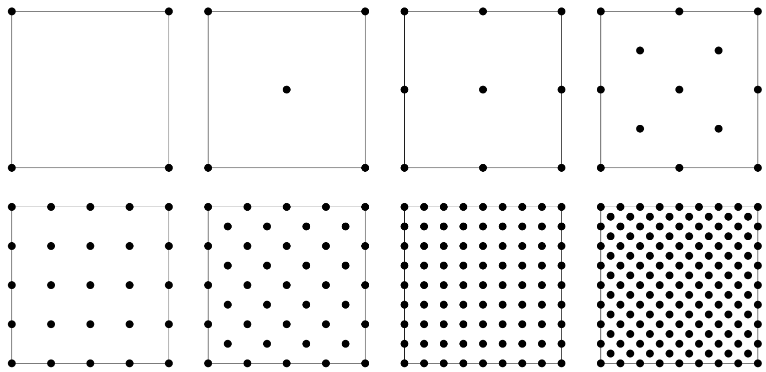

Abbildung 4.2: Gitter bei sukzessiver irregulärer Verfeinerung

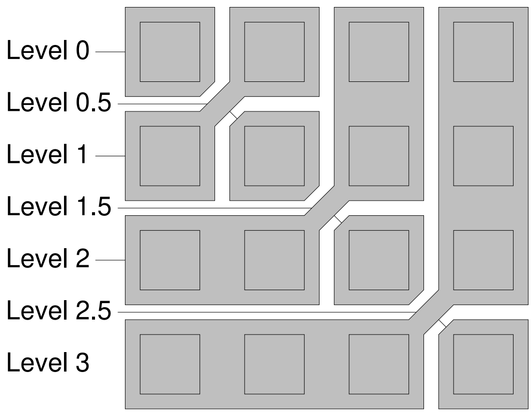

Abbildung 4.3: Level bei semiregulären Gittern

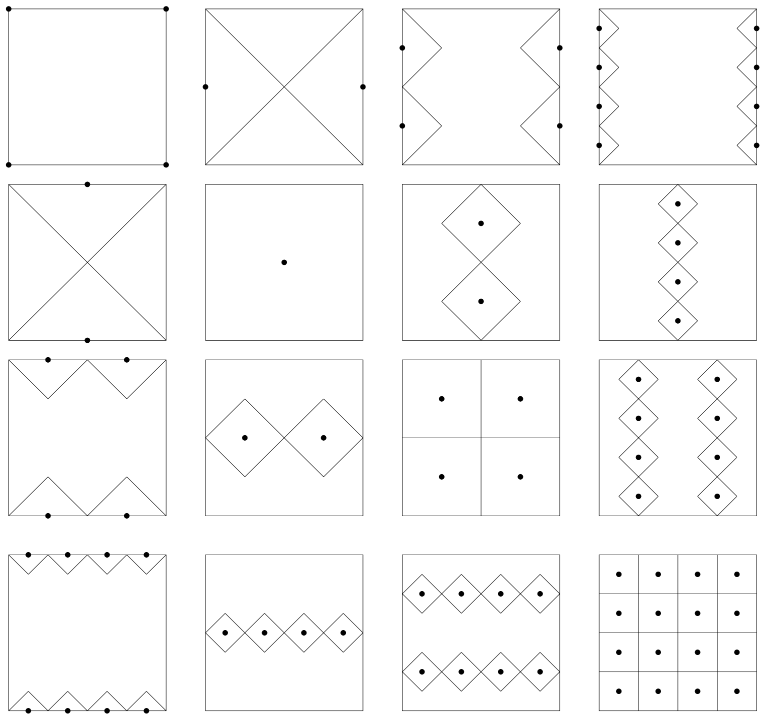

Abbildung 4.4: Teilraumschema bei semiregulären Gittern

Betrachten wir nun das Teilgitterschema 3.12, so finden wir semireguläre Gitter vom Level $l + \frac{1}{2}$ als reguläre Gittern G_l zusätzlich dem Diagonalelement $\bar{G}_{l+1,l+1}$ wieder:

$$G_{l+\frac{1}{2}} = (\bigcup_{i=1}^{l} \bigcup_{j=1}^{l} \bar{G}_{ij}) \cup \bar{G}_{l+1,l+1}. \tag{4.2}$$

Der Levelbegriff entspricht damit bei semiregulären Gittern einer quadratischen Ordnung auf dem Teilgitterschema, wobei aber die Diagonalelemente $\bar{G}_{l,l}$ hierarchisch höher stehen, als die Elemente in der gleichen Zeile und Spalte $\bar{G}_{i,l}$ und $\bar{G}_{i,l}, i < l$.

Wir können nun ein Teilraumschema für semireguläre Gitter (Abb. 4.4) angeben, das ohne Überlappungen auf einem Level arbeitet. Das Schema verwendet hierbei in den Außerdiagonalelementen analog zu den semiregulären Gittern um 45 Grad gedrehten Teilräume. In den Diagonalelementen entspricht das Schema dem Dünngitterteilraumschema. Man sieht, daß die Vereinigung aller Teilräume eines Levels genau dem Gesamtgebiet Ω entspricht.

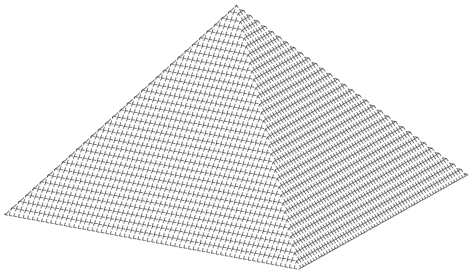

Abbildung 4.5: Lineare Basisfunktion für semireguläre Gitter

4.2 Interpolationstechniken

Wir werden nun verschiedene Basen auf diesem Teilgitterschema definieren. Für zeichnerische Zwecke sind aber nur die linearen und konstanten Basen interessant. Die anderen Basen werden jedoch aufgrund ihrer interessanten mathematischen Eigenschaften vorgestellt.

4.2.1 Lineare Interpolation

Wir wollen nun auf den semiregulären Gittern einen linearen Interpolanten definieren. Hierzu verwenden wir die folgende lineare Basisfunktion und ihr um 45 Grad gedrehtes und um $\sqrt{2}$ verkleinertes Pendant (Abb. 4.5):

$$\psi(x,y) = \begin{cases} 1 - \frac{x}{h_x} & \text{, falls } 0 \leq x \leq 1 \text{ und } y \leq x \leq -y, \\ 1 - \frac{y}{h_y} & \text{, falls } 0 \leq y \leq 1 \text{ und } x \leq y \leq -x, \\ \frac{x}{h_x} & \text{, falls } 0 \leq x \leq 1 \text{ und } -y \leq x \leq y, \\ \frac{y}{h_y} & \text{, falls } 0 \leq y \leq 1 \text{ und } -x \leq y \leq x, \\ 0 & \text{, sonst.} \end{cases} \tag{4.3}$$

Diese linearen Basisfunktionen erhält man mit Hilfe folgender Standard–Interpolationssterne:

$$I^{diag_1}_{(l,i),h} = \begin{bmatrix} -\frac{1}{2} & & \\ & 1 & \\ & & -\frac{1}{2} \end{bmatrix}_{(l,i),h}, \quad I^{diag_2}_{(l,i),h} = \begin{bmatrix} & & -\frac{1}{2} \\ & 1 & \\ -\frac{1}{2} & & \end{bmatrix}_{(l,i),h}, \tag{4.4}$$

$$I^{vert}_{(l,i),h} = \begin{bmatrix} -\frac{1}{2} \\ 1 \\ -\frac{1}{2} \end{bmatrix}_{(l,i),h}, \quad I^{horiz}_{(l,i),h} = \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}_{(l,i),h}. \tag{4.5}$$

Die Wahl des Interpolationssterns wird aufgrund lokaler Bedingungen gemacht. Hierzu wollen wir die einzelnen Gitterpunkte eines Teilgitters mit p in x–Richtung und q in y–Richtung durchnumerieren. Für den Teilraum $T_{i,j}$ gilt dann:

$$I_{(l,i),h} = \begin{cases} I^{vert}_{(l,i),h} & \text{, falls } i > j, \\ I^{horiz}_{(l,i),h} & \text{, falls } i < j, \\ I^{diag_1}_{(l,i),h} & \text{, falls } i = j \text{ und } (p+q) \text{ gerade}, \\ I^{diag_2}_{(l,i),h} & \text{, falls } i = j \text{ und } (p+q) \text{ ungerade}. \end{cases} \tag{4.6}$$

Man erhält das Interpolationsschema für dreieckige Finite Elemente, wenn man die Interpolationssterne für $i = j$ gleich wählt (siehe [6]). Für die Ordnung des Interpolationsfehlers gelten daher die Abschätzungen aus 3.4.1.

4.2.2 Interpolation mit Greenschen Funktionen

Die mathematisch wohl interessanteste Basisvariante auf dem Teilraumschema für semireguläre Gitter sind Basen, die eine Näherung von Greenschen Funktionen darstellen. Hierzu wählen wir die folgenden 5–Punkt–Interpolationssterne:

$$I^{kreuz_{diag}}_{(l,i),h} = \begin{bmatrix} -\frac{1}{4} & & -\frac{1}{4} \\ & 1 & \\ -\frac{1}{4} & & -\frac{1}{4} \end{bmatrix}_{(l,i),h}, \quad I^{kreuz_{coax}}_{(l,i),h} = \begin{bmatrix} & -\frac{1}{4} & \\ -\frac{1}{4} & 1 & -\frac{1}{4} \\ & -\frac{1}{4} & \end{bmatrix}_{(l,i),h} \tag{4.7}$$

Der Rand des Interpolationsgebiets muß hier speziell behandelt werden. Zunächst wollen wir linear am Rand interpolieren. Da Randpunkte nur in Nebendiagonalelementen und damit nur bei nicht ganzzahligen Leveln auftreten, bestimmen wir:

$$I_{(l,i),h} = \begin{cases} I^{vert}_{(l,i),h} & \text{, falls } x = 0 \text{ oder } x = 1, \\ I^{horiz}_{(l,i),h} & \text{, falls } y = 0 \text{ oder } y = 1, \end{cases}, \tag{4.8}$$

und im Inneren des Gebietes:

$$I_{(l,i),h} = \begin{cases} I^{kreuz_{coax}}_{(l,i),h} & \text{, falls } i \neq j, \\ I^{kreuz_{diag}}_{(l,i),h} & \text{, falls } i = j, \end{cases}. \tag{4.9}$$

Diese Wahl der Interpolationssterne führt zu Greenschen Basisfunktionen mit quadratischem Träger (Abb. 4.6). Wählt man am unteren Rand des Gebietes alternativ den folgenden Interpolationsstern, bzw. an den übrigen Rändern die entsprechend gedrehte Variante:

$$\begin{bmatrix} & -\frac{1}{2} & \\ -\frac{1}{4} & 1 & -\frac{1}{4} \end{bmatrix}_{(l,i),h}, \tag{4.10}$$

so führt das zu Greenschen Basisfunktion mit rundem Träger (Abb. 4.7).

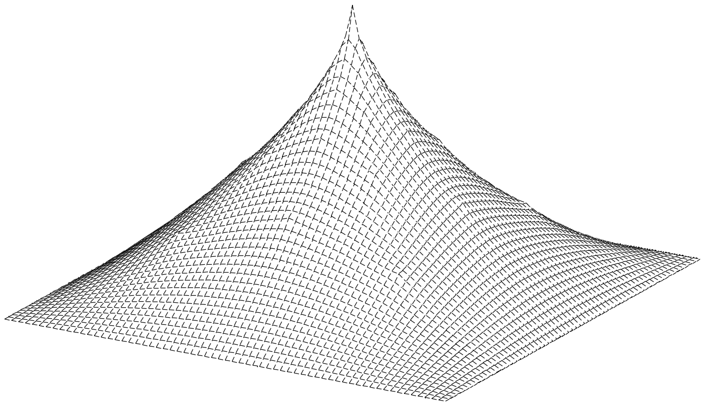

Abbildung 4.6: Greensche Basisfunktion mit quadratischem Träger

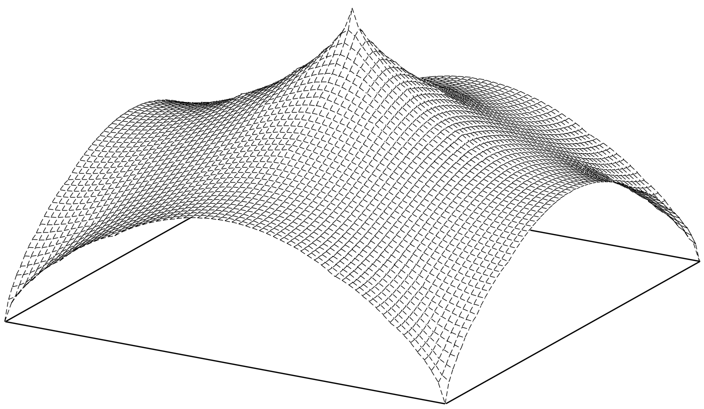

Abbildung 4.7: Greensche Basisfunktion mit rundem Träger

4.2.3 Bilineare Interpolation

Erstaunlicherweise erhalten wir bilineare Basen, falls wir die beiden bereits vorgestellten Interpolationstechniken mischen. Wählt man die Technik für lineare Basen auf den regulären Gittern und die Technik für Greensche Basen auf den semiregulären Gittern, so erhält man bilineare Basen auf den irregulären Gittern und lineare Basen auf den regulären Gittern. Auch die genau umgedrehte Mischung der Techniken ist möglich und führt auch zu den genau umgedrehten Ergebnissen.

$$I_{(l,i),h} = \begin{cases} I^{vert}_{(l,i),h} & \text{, falls } \; i < j \\ I^{horiz}_{(l,i),h} & \text{, falls } \; i > j \\ I^{kreuz_{diag}}_{(l,i),h} & \text{, falls } \; i = j \end{cases} \tag{4.11}$$

$$I_{(l,i),h} = \begin{cases} I^{kreuz_{coax}}_{(l,i),h} & \text{, falls } \; i \neq j \\ I^{diag_1}_{(l,i),h} & \text{, falls } \; i = j \text{ und } (p+q) \text{ ungerade} \\ I^{diag_2}_{(l,i),h} & \text{, falls } \; i = j \text{ und } (p+q) \text{ gerade} \end{cases} \tag{4.12}$$

4.2.4 Interpolation mit konstanten Basen

Speziell für zweidimensionale Darstellungen interessant sind konstante Basen, wie in Kapitel 7 gezeigt werden wird. Hierbei ist die Ortsabhängigkeit der Interpolationssterne noch größer als in den vorhergehenden Fällen.

$$I_{(l,i),h} = \begin{cases} I^{W}_{(l,i),h} = \begin{bmatrix} -1 & 1 \end{bmatrix}_{(l,i),h}, & \text{falls } \; i \neq j \text{ und } x \text{ gerade und } y \text{ gerade} \\ I^{N}_{(l,i),h} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}_{(l,i),h}, & \text{falls } \; i \neq j \text{ und } x \text{ gerade und } y \text{ ungerade} \\ I^{S}_{(l,i),h} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}_{(l,i),h}, & \text{falls } \; i \neq j \text{ und } x \text{ ungerade und } y \text{ gerade,} \\ I^{O}_{(l,i),h} = \begin{bmatrix} 1 & -1 \end{bmatrix}_{(l,i),h}, & \text{falls } \; i \neq j \text{ und } x \text{ ungerade und } y \text{ ungerade,} \\ I^{NW}_{(l,i),h} = \begin{bmatrix} -1 & \\ & 1 \end{bmatrix}_{(l,i),h}, & \text{falls } \; i = j \text{ und } x \text{ gerade und } y \text{ gerade,} \\ I^{NO}_{(l,i),h} = \begin{bmatrix} & -1 \\ 1 & \end{bmatrix}_{(l,i),h}, & \text{falls } \; i = j \text{ und } x \text{ gerade und } y \text{ ungerade,} \\ I^{SW}_{(l,i),h} = \begin{bmatrix} & 1 \\ -1 & \end{bmatrix}_{(l,i),h}, & \text{falls } \; i = j \text{ und } x \text{ ungerade und } y \text{ gerade,} \\ I^{SO}_{(l,i),h} = \begin{bmatrix} 1 & \\ & -1 \end{bmatrix}_{(l,i),h}, & \text{falls } \; i = j \text{ und } x \text{ ungerade und } y \text{ ungerade.} \end{cases} \tag{4.13}$$

4.3 Numerische Experimente

Die numerischen Experimente wurden an einer Silicon Graphics Indigo^2 Workstation mit 64 MByte Hauptspeicher berechnet. Die Testfunktion u wurde hierzu auf einem quadratischen Gitter mit $2^n + 1$ Punkten in x– und y–Richtung für $n = 10$ diskretisiert. Der Interpolationsfehler $\|\hat{e}\|$ wurde punktweise folgendermaßen berechnet:

$$\|\hat{e}\|_\infty = \max\{i,j : (u(i,j) - \hat{u}(i,j))\} \tag{4.14}$$

$$\|\hat{e}\|_2 = \frac{\sqrt{\sum_{i,j\in\Omega}(u(i,j) - \hat{u}(i,j))^2)}}{2^n + 1} \tag{4.15}$$

Als Adaptivitätskriterium wählen wir:

$$\epsilon = 2^{-l},\ 1 \leq l \leq 10 \tag{4.16}$$

Am Ende des Kapitels sehen wir die Ergebnisse tabellarisch für $l = 1 \ldots 10$ und die graphischen Darstellungen für den Interpolant, das zugehörige Gitter und den zugehörigen Fehler für $l = 1 \ldots 5$. In den Fällen, wo sich das Gitter hierbei in zwei aufeinanderfolgenden Schritten nicht ändert, wurden diese Schritte ausgelassen und dafür l bis 6 bzw. 7 gezählt. Das Gitter wird aus programmtechnischen Gründen um 90 Grad gedreht dargestellt.

Lineare Basen

Wir wählen nun die Testfunktion u_1 so, daß die Glattheitsbedingungen

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \leq c$$

erfüllt sind:

$$u_1 = 2 \cdot (x \cdot (1 - x)) + 2 \cdot (y \cdot (1 - y)). \tag{4.17}$$

Man sieht, daß die Zahl der benötigten Punkte asymptotisch gegen h^2 strebt (Tabelle 4.8). Der Fehler in der L2-Norm ist ebenfalls nahe bei h^2. Für die Interpolation mit den anderen vorgestellten Basisfunktionen gelten die gleichen Eigenschaften.

Im zweiten Beispiel wollen wir das Verhalten des Verfahrens bei Singularitäten untersuchen. Hierzu wählen wir:

$$u_2 = (x^2 + y^2)^{\frac{1}{4}} \cdot \cos\left(\frac{\arctan(\frac{x}{y})}{2}\right). \tag{4.18}$$

Man sieht, daß die Singularität aufgrund der diagonalen Verfeinerungsmöglichkeiten gut aufgelöst werden kann.

Bilineare Basen

Um zu testen, wie gut bilineare Basen auf semiregulären Gittern Funktionen mit für dünne Gitter optimalen Glattheitseigenschaften interpolieren können, wählen wir:

$$u_3 = 16 \cdot x \cdot (1 - x) \cdot y \cdot (1 - y). \tag{4.19}$$

In Tabelle 4.9 sieht man, daß in diesem Fall für die benötigten Punkte und die Fehlernormen ebenfalls $O(h^2)$ gilt.

Greensche Basen

Für Funktionen u, für die gilt:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{4.20}$$

(harmonische Funktionen) führt diese Interpolation zu erstaunlich guten Ergebnissen. Wir wählen u_4 als die Lösung der Laplace-Gleichung

$$\Delta u = 0 \tag{4.21}$$

bei Dirichlet–Randbedingungen:

$$u_4 = \sin(\pi \cdot x) \cdot \frac{\sinh(\pi \cdot (1 - x))}{\sinh(\pi)} \tag{4.22}$$

Wie man in Abbildung 4.9 sieht kann u_4 nahezu exakt bis auf den Rand interpoliert werden.

$\|e_{n,n}\|_\infty$	$\frac{\|e_{n-1,n-1}\|_\infty}{\|e_{n,n}\|_\infty}$	$\|e_{n,n}\|_2$	$\frac{\|e_{n-1,n-1}\|_2}{\|e_{n,n}\|_2}$	$\|p_{n,n}\|$	$\frac{\|p_{n-1,n-1}\|}{\|p_{n,n}\|}$
1.0000000	-	0.6913003	-	4	0.000000
0.5000000	2.0000000	0.3496336	1.9984047	5	1.250000
0.2500000	2.0000000	0.1746771	2.0015991	9	1.800000
0.1250000	2.0000000	0.0874083	1.9984047	13	1.444444
0.0625000	2.0000000	0.0436691	2.0016069	25	1.923077
0.0312500	2.0000000	0.0218520	1.9984047	41	1.640000
0.0156250	2.0000000	0.0109170	2.0016382	81	1.975610
0.0078125	2.0000000	0.0054629	1.9984046	145	1.790123
0.0039062	2.0000000	0.0027290	2.0017632	289	1.993104
0.0019531	2.0000000	0.0013656	1.9984042	545	1.885813
0.0009766	2.0000000	0.0006820	2.0022637	1089	1.998165

Tabelle 4.1: Punktweiser Fehler bei linearer Interpolation von u_1

$\|e_{n,n}\|_\infty$	$\frac{\|e_{n-1,n-1}\|_\infty}{\|e_{n,n}\|_\infty}$	$\|e_{n,n}\|_2$	$\frac{\|e_{n-1,n-1}\|_2}{\|e_{n,n}\|_2}$	$\|p_{n,n}\|$	$\frac{\|p_{n-1,n-1}\|}{\|p_{n,n}\|}$
0.3528629	-	0.0499528	-	4	0.000000
0.1760267	2.0045985	0.0411410	1.2141830	5	1.250000
0.1242500	1.4167138	0.0266806	1.5419835	8	1.600000
0.0617518	2.0120887	0.0228433	1.1679847	13	1.625000
0.0305035	2.0244152	0.0102440	2.2299119	24	1.846154
0.0148820	2.0496908	0.0044636	2.2950079	37	1.541667
0.0076262	1.9514387	0.0024542	1.8187955	59	1.594595
0.0038882	1.9613840	0.0012922	1.8991957	97	1.644068
0.0019440	2.0000378	0.0006364	2.0304448	179	1.845361
0.0009760	1.9919309	0.0003321	1.9161547	342	1.910614

Tabelle 4.2: Punktweiser Fehler bei linearer Interpolation von u_2

$\|e_{n,n}\|_\infty$	$\frac{\|e_{n-1,n-1}\|_\infty}{\|e_{n,n}\|_\infty}$	$\|e_{n,n}\|_2$	$\frac{\|e_{n-1,n-1}\|_2}{\|e_{n,n}\|_2}$	$\|p_{n,n}\|$	$\frac{\|p_{n-1,n-1}\|}{\|p_{n,n}\|}$
0.3480749	-	0.2196337	-	5	0.000000
0.1137963	3.0587545	0.0597261	3.6773490	13	2.600000
0.1137963	1.0000000	0.0597261	1.0000000	13	1.000000
0.0586960	1.9387418	0.0322769	1.8504284	33	2.538461
0.0305253	1.9228609	0.0152272	2.1196878	53	1.606061
0.0148468	2.0560201	0.0088745	1.7158351	121	2.283019
0.0078120	1.9005238	0.0045602	1.9460893	293	2.421488
0.0038576	2.0250744	0.0023410	1.9479874	469	1.600683
0.0019518	1.9764850	0.0011128	2.1037408	985	2.100213
0.0009761	1.9995325	0.0006130	1.8153609	1973	2.003046

Tabelle 4.3: Punktweiser Fehler bei bilinearer Interpolation von u_3

$\|e_{n,n}\|_\infty$	$\frac{\|e_{n-1,n-1}\|_\infty}{\|e_{n,n}\|_\infty}$	$\|e_{n,n}\|_2$	$\frac{\|e_{n-1,n-1}\|_2}{\|e_{n,n}\|_2}$	$\|p_{n,n}\|$	$\frac{\|p_{n-1,n-1}\|}{\|p_{n,n}\|}$
0.2105136	-	0.0744805	-	5	0.000000
0.2105136	1.0000000	0.0744805	1.0000000	5	1.000000
0.2105136	1.0000000	0.0744805	1.0000000	5	1.000000
0.0703768	2.9912368	0.0091945	8.1005101	8	1.600000
0.0298231	2.3598075	0.0060133	1.5290368	10	1.250000
0.0298231	1.0000000	0.0060133	1.0000000	10	1.000000
0.0107006	2.7870592	0.0012379	4.8577359	18	1.800000
0.0071787	1.4906010	0.0011312	1.0942946	20	1.111111
0.0038451	1.8669947	0.0008164	1.3856540	27	1.350000
0.0014017	2.7430832	0.0002432	3.3561404	41	1.518519

Tabelle 4.4: Punktweiser Fehler bei Interpolation von u_4 mit Greenschen Basen

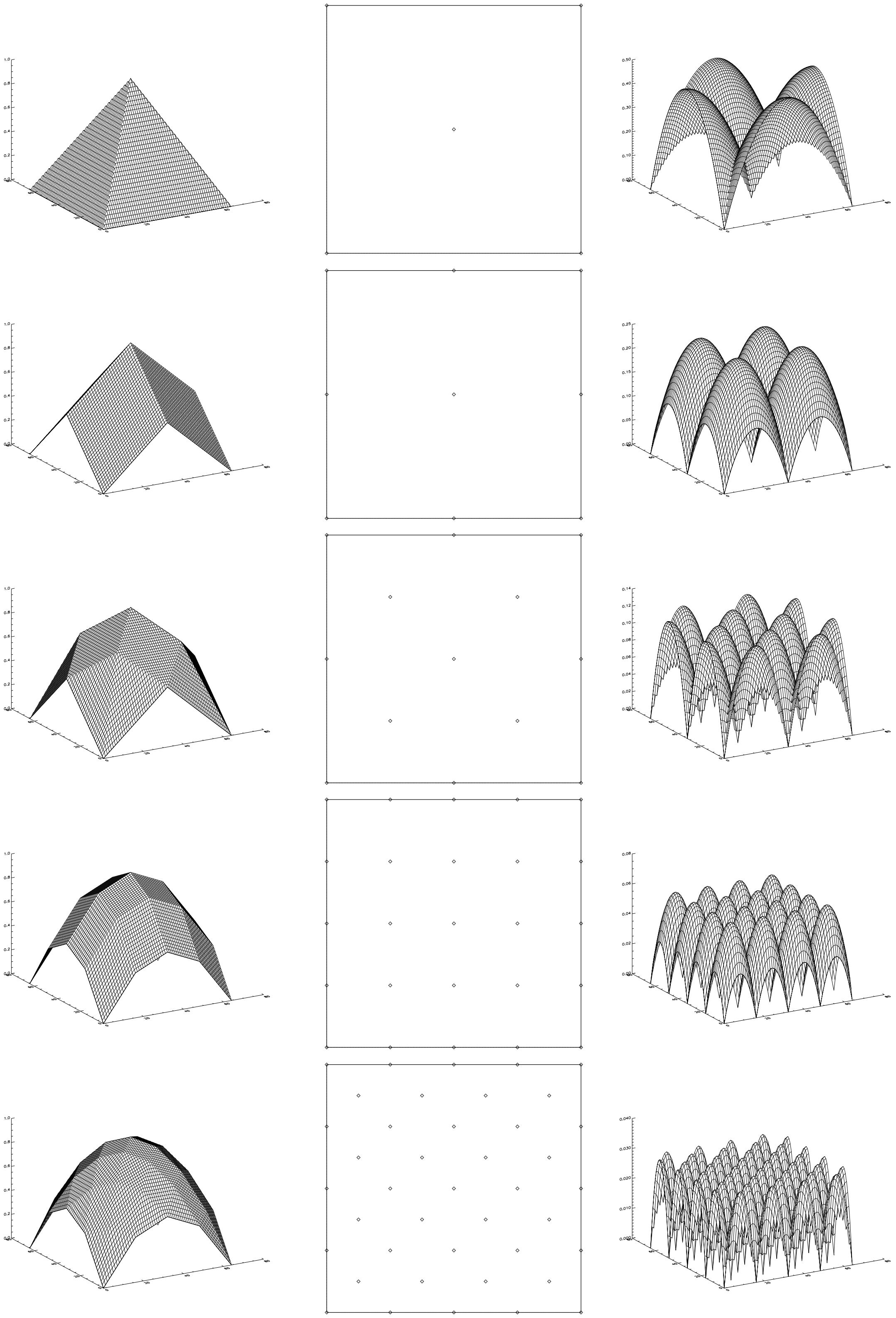

Abbildung 4.8: Interpolant, Gitter und Fehler bei linearer Interpolation von u_1

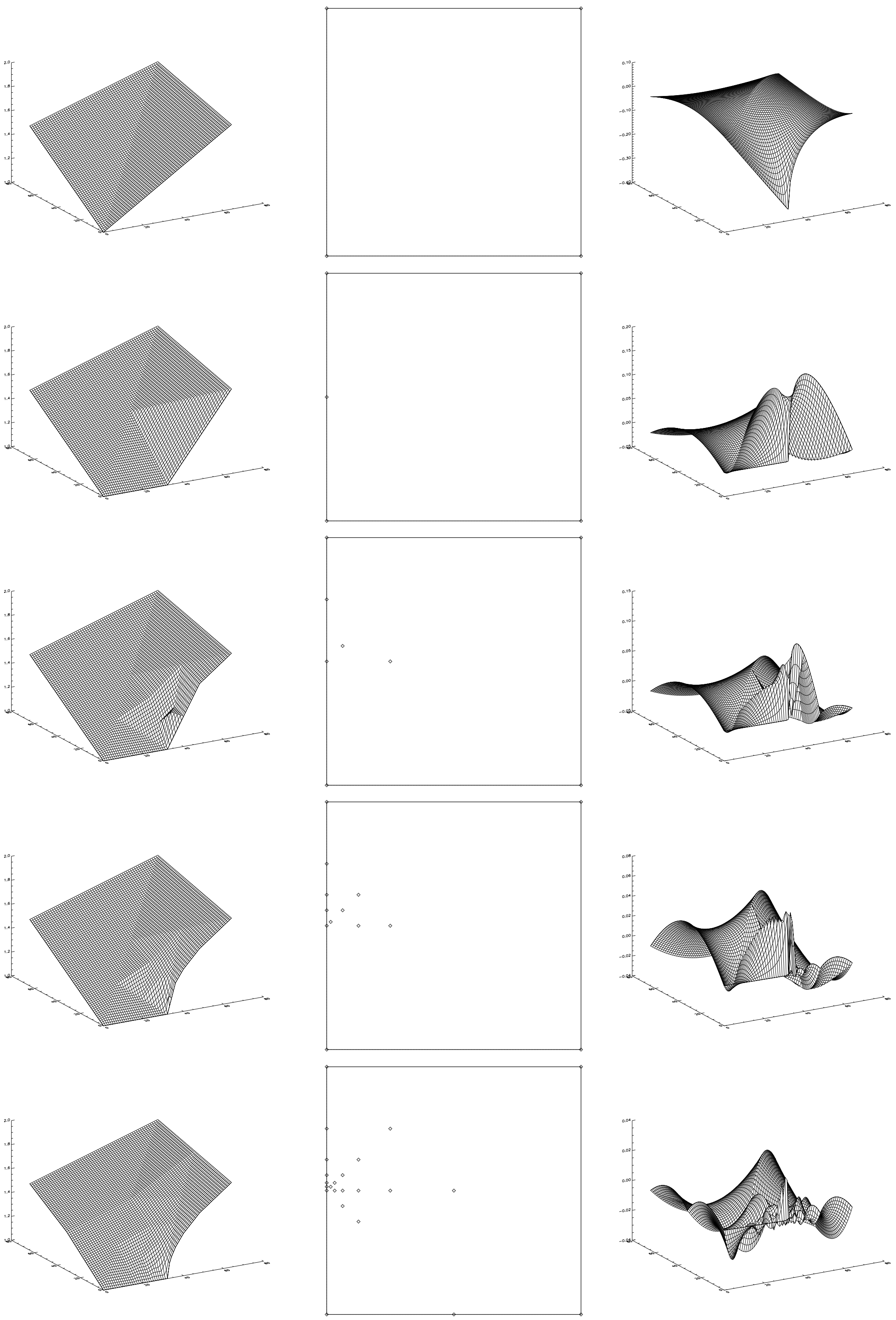

Abbildung 4.9: Interpolant, Gitter und Fehler bei linearer Interpolation von u_2

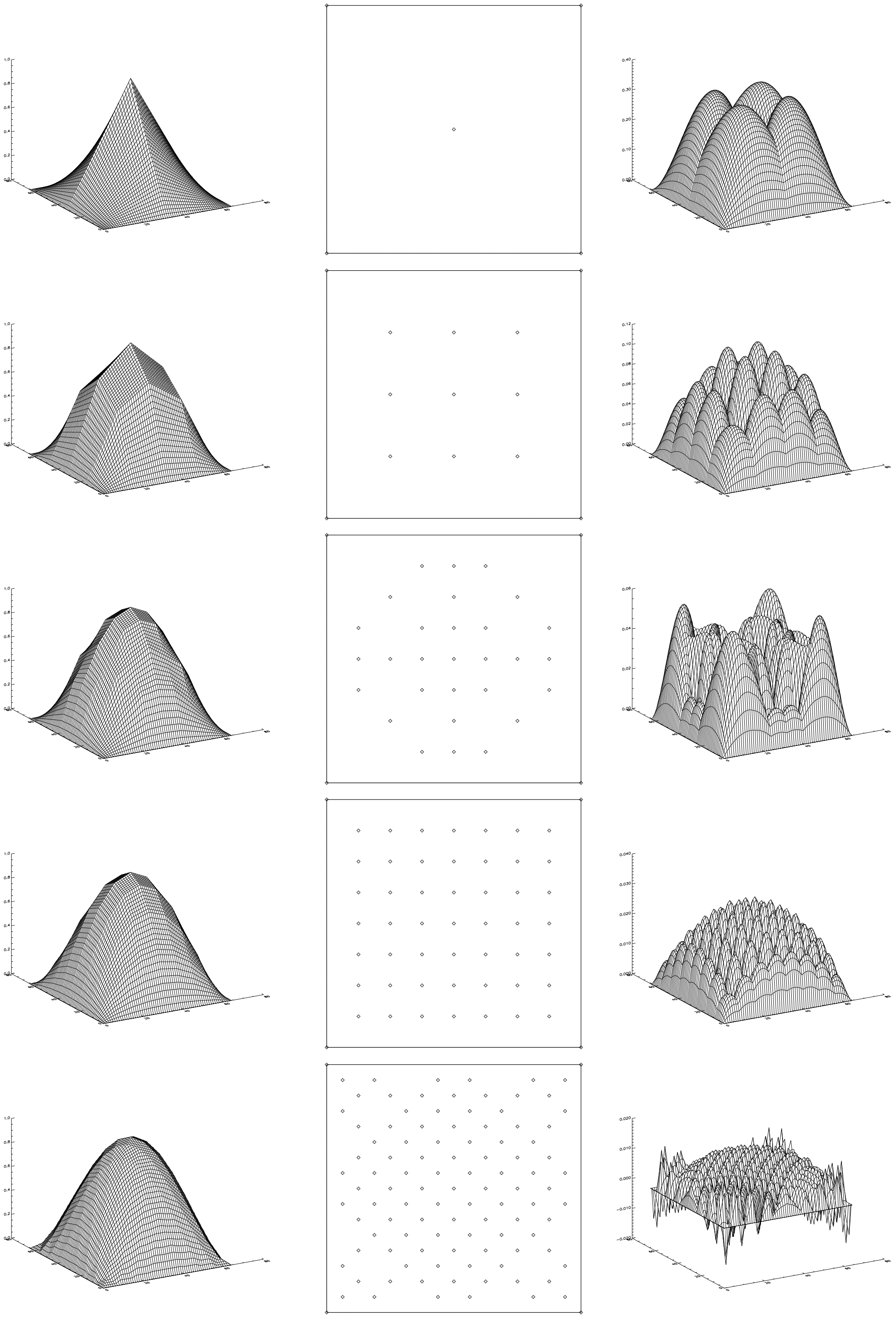

Abbildung 4.10: Interpolant, Gitter und Fehler bei bilinearer Interpolation von u_3

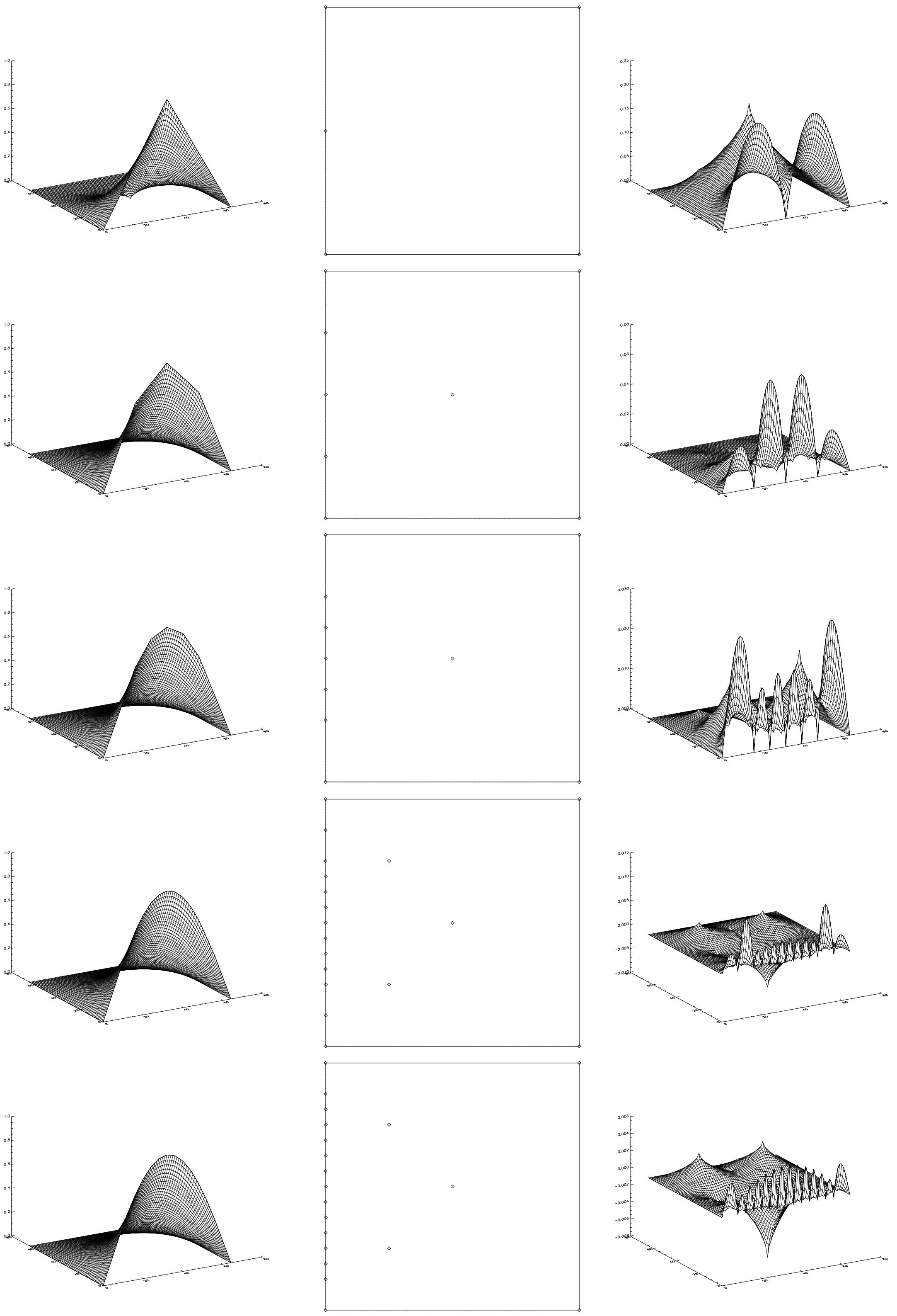

Abbildung 4.11: Interpolant, Gitter und Fehler bei Interpolation von u_4 mit Greenschen Basen

Kapitel 5

Algorithmen und Verfahren

5.1 Vorgehensweise

Wir werden in diesem Kapitel auf die effiziente Realisierung der im vorigen Kapitel beschriebenen Interpolationstechniken eingehen. Hierbei spielen algorithmische Aspekte sowohl bei der Diskretisierung der Originalfunktion u, als auch bei der Speicherung der diskreten Daten und ihrer graphischen Darstellung eine Rolle.

Wir wollen bei der Wahl von u zwei Fälle unterscheiden: Im ersten Fall geht man von einer mathematischen Funktion u aus, die als Funktionsvorschrift vorliegt. u wird dann an den benötigten Punkten ausgewertet, wobei allerdings die Auswertung bei komplizierten Vorschriften aufwendig sein kann. Im zweiten Fall ist u ein schon diskreter Datensatz, wobei normalerweise auch nur auf die diskreten Werte zugegriffen werden kann. Ist eine Interpolationsvorschrift zur Bestimmung der Werte zwischen den diskreten Daten gegeben, so läßt sich dieser diskrete Datensatz wieder wie eine kontinuierliche Funktion behandeln. In beiden Fällen wollen wir die Menge der zugegriffenen Daten als Originaldaten bezeichnen.

Wir wollen nun u nun sukzessive interpolieren. Das Ziel der Interpolation ist eine diskrete Darstellung $\hat{u}$ von u mit einem vorgegebenem maximalen Fehler ϵ. Hierzu wählen wir den hierarchischen Überschuß als Fehlerkriterium. Um $\hat{u}$ abzuspeichern, speichern wir nur Punkte $x_{l,i}$, deren hierarchischer Überschuß $v_{l,i} \geq \epsilon$ ist. Ist u als diskreter Datensatz gegeben, so wollen wir bei Speicherung von $\hat{u}$ einen Kompressionseffekt erzielen, d.h. die Menge der gespeicherten Daten soll kleiner sein als der Ausgangsdatensatz. Daraufhin soll die graphische Darstellung der gespeicherten Daten erfolgen. Da hierbei eine Interpolation zurück zu der Originalfunktion stattfindet, die in diesem Fall vom Graphiksystem erledigt wird, sprechen wir von Rückinterpolation.

5.2 Interpolationsstrategien

Wir wollen nun u Punkt für Punkt nach einer festen Durchlaufstrategie interpolieren. Hierzu müssen wir den hierarchischen Überschuß für jeden dieser Punkte berechnen. Diese Berechnung erfolgt mittels der im vorigen Kapitel angegebenen Interpolationssterne. Der Interpolationsstern kann nun sowohl auf den Originaldaten oder auf bereits interpolierten Daten, sofern alle benötigten Daten vorliegen, aufgesetzt werden. Theoretisch möglich sind auch Mischformen, bei denen interpolierte Daten verwendet werden, wenn sie vorliegen und Originaldaten, wo keine interpolierten Daten gegeben sind, benutzt werden. Für solche Mischformen ist allerdings die Angabe einer geeigneten Fehlerschranke für die Interpolation problematisch.

5.2.1 Interpolation auf den Originaldaten

Zunächst scheint die Wahl der Originaldaten zur Interpolation die sinnvollere Methode, da die Originaldaten, im Gegensatz zu den interpolierten Daten, nicht fehlerbehaftet sind. Die Rückinterpolation kann aber nur die gespeicherten Daten verwenden und muß demnach nur auf den bereits rückinterpolierten Daten arbeiten. Nach der Interpolation gilt zwar für jeden Punkt, daß sein maximaler Fehler bezüglich der Originaldaten ϵ ist, bei der Rückinterpolation wird aber der hierarchische Überschuß eines Punktes evtl. aus mit einem Fehler ϵ behafteten Daten interpoliert und bekommt damit einen Fehler, der größer als ϵ ist. Falls sich dieser Fehler weiter fortsetzt spricht man von Fehlerakkumulation.

Dieser Fehlerakkumulation kann entgegengewirkt werden, indem ϵ mit einem Kompensationsfaktor versehen wird, d.h. als Fehlerkriterium wird nicht ϵ sondern $\kappa \cdot \epsilon$ verwendet. κ ist dabei im allgemeinen Fall eine level- und ortsabhängige Große und ist eine obere Schranke für die Fehlerakkumulation. Eine genauere Untersuchung dieser Fehlerkompensation findet sich für die Dünngitterkompression in [13] und [20].

Wie schon angesprochen, kann der Zugriff auf die Originaldaten teuer sein, z.B. wenn u als Funktionsvorschrift vorliegt. Oft muß bei der Interpolation auf die gleichen Funktionswerte mehr als einmal zugegriffen werden, was die Effizienz des Verfahrens verschlechtert. Idealerweise will man einen Punkt nicht mehrfach auswerten, weswegen man die schon ausgewerteten Punkte geeignet speichern muß. Im Prinzip gelten dafür die gleichen Strategien, wie für die Speicherung von interpolierten Daten.

Falls der Zugriff auf die Originaldaten leicht ist, benötigt allerdings diese Methode keinen zusätzlichen Speicherplatz. Besonders bei der Kompression großer Datensätze kann diese Methode deswegen sinnvoll sein.

5.2.2 Interpolation auf bereits interpolierten Daten

Um den bei der Interpolation entstehenden Fehler durch ϵ zu beschränken, dürfen Punkte, deren hierarchischer Überschuß $v_{l,i} \geq \epsilon$ ist, nicht interpoliert werden, d.h. der Originalwert dieser Punkte wird übernommen. Für die übrigen Punkte muß der interpolierte Wert mit Hilfe der Interpolationssterne berechnet werden. Wir wollen diese beiden Klassen von Werten als bereits interpolierte Daten bezeichnen.

Falls bei der Interpolation auf bereits interpolierten Daten zugegriffen werden soll, müssen diese zunächst vorhanden sein, d.h. in einem der vorhergehenden Interpolationsschritte interpoliert worden sein. Nicht bei allen Interpolationstechniken und Durchlaufstrategien ist dies möglich. Wir werden aber im nächsten Paragraphen einige dieser Strategien angeben, bei denen diese Bedingung erfüllt ist.

Dafür tritt bei der Rückinterpolation der Daten kein zusätzlicher Fehler auf, sofern die Rückinterpolation auf die gleiche Art und Weise passiert, wie die Interpolation. Die Fehlerschranke ϵ gilt damit auch wirklich bei der Darstellung des Interpolanten.

Der Nachteil des Verfahrens ist allerdings, daß die bereits interpolierten Daten geeignet gespeichert werden müssen. In 5.4 werden wir auf verschiedene Speicherstrategien für die schon interpolierten Daten eingehen.

5.3 Durchlaufstrategien

Zunächst wollen wir uns nun der Frage zuwenden, in welcher Reihenfolge die Originaldaten interpoliert werden. Diese Reihenfolge nennt man den Durchlauf der Daten. Für die Bestimmung des hierarchischen Überschusses wird dabei für jeden Punkt neben der Position des Punktes auch die Größe und Art des Interpolationssterns benötigt. Wir wollen speziell drei Durchlaufstrategien betrachten, die für unsere hierarchische Interpolationstechniken interessant sein könnten.

5.3.1 Linearer Durchlauf

Beim linearen Durchlauf werden die Punkte in einer geometrischen Reihenfolge durchgelaufen (Abb. 5.1). Während sich die Position des Punktes direkt aus dem Durchlauf ermitteln läßt, ist die Bestimmung des Interpolationssterns aufwendig, da z.B. für jeden Punkt erst sein Level bestimmt werden muß.

Ein linearer Durchlauf kann z.B. zeilen- oder spaltenweise erfolgen. Bei der iterativen Lösung partieller Differentialgleichungen gibt es auch Abwandlungen, wie das Red-Black-Verfahren, bei denen im ersten Schritt nur jeder zweite Punkt gewählt wird und im zweiten Schritt die übrigen Punkte gewählt werden. Wir geben ein Beispiel für spaltenweisen Durchlauf an:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Abbildung 5.1: Linearer Durchlauf

```
for i=0 to hx
  for j=0 to hy
    interpoliere punkt(i,j)
  endfor
endfor
```

Beim linearen Durchlauf geht die hierarchische Sichtweise völlig verloren. Dadurch muß ein maximaler Level vorgegeben werden, bis zu dem interpoliert werden kann. Dies ist jedoch nicht weiter problematisch, falls u ein diskreter Datensatz ist, der ohnehin nur an diskreten Punkten ausgewertet werden kann.

Im allgemeinen ist eine Interpolation auf bereits interpolierten Daten nicht möglich, wenn wie im unseren Fall die Interpolationssterne levelabhängig definiert werden. Daher wenden wir uns nun hierarchischen Durchlaufstrategien zu.

5.3.2 Levelweiser Durchlauf

Beim levelweisen (oder Breadth–First) Durchlauf wird die hierarchische Struktur der Interpolation genutzt. Hier werden die Punkte in der Reihenfolge ihrer Level durchgelaufen. Innerhalb eines Levels können die Punkte dann beliebig, z.B. linear durchgelaufen werden. Wenn zu einem Level mehr als ein Teilgitter aus dem schon bekannten Teilgitterschema 3.12 gehören, erfolgt der Durchlauf anhand ihrer Anordnung im Schema.

Die Berechnung der Position der Punkte ist deswegen aufwendiger als beim linearen Durchlauf, dafür ist aber die Bestimmung des Interpolationssterns einfach, da zum einen der Level für jeden Punkt gegeben ist und zum anderen der Typ des Interpolationssterns nach Formel 4.6 aus der Position des Punktes innerhalb eines Teilgitters berechnet werden kann.

Prinzipiell gibt es zwei Möglichkeiten des levelweisen Durchlaufs, die sich darin unterscheiden, ob zuerst mit hierarchisch höheren (Top–Down) oder mit hierarchisch tieferen (Bottom–up) Leveln begonnen wird (Abb. 5.2).

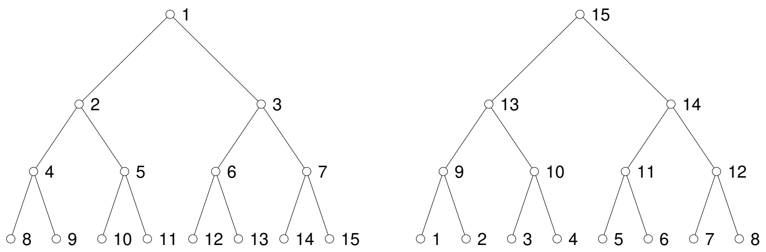

Abbildung 5.2: Top–Down und Bottom–Up Durchlauf

Top-down Durchlauf

Beim Top–Down Durchlauf wird mit dem hierarchisch höchsten Level begonnen und sukzessive der Levelindex *level* erhöht. Da diese Erhöhung natürlich nicht unendlich lang geschehen kann (dann wäre u exakt dargestellt, siehe 3.17), wird eine Abbruchbedingung benötigt. Dieses Abbruchbedingung kann durch ein Adaptivitätskriterium definiert werden oder auch ein fester Level sein, was z.B. sinnvoll ist, wenn u ein diskreter Datensatz ist. Im folgenden Beispiel haben wir ein levelweises Abbruchkriterium gewählt:

```
level=0
repeat
  for i,j in punkte(level)
    interpoliere punkt(i,j)
  endfor
  level=level+0.5
until <abbruchbedingung>
```

Beim Top–Down Durchlauf kann die Interpolation auf den schon interpolierten Daten arbeiten. Jeder Level muß dabei nicht notwendigerweise vollständig durchlaufen werden, wenn sichergestellt ist, daß alle benötigten Daten berechnet werden. Bei einem adaptiven Abbruchkriterium kann es z.B. sinnvoll sein, Punkte nicht durchzulaufen, deren hierarchischer Vater interpoliert wurde (siehe auch Kapitel 3.1.4). Falls u die Glattheitsbedingungen aus 3.4.1 erfüllt, ist es garantiert, daß dadurch kein zusätzlicher Fehler gemacht wird, der größer als ϵ ist, für den allgemeinen Fall kann dies allerdings nicht garantiert werden. Aus diesem Grund versucht man, mit einer begrenzten Vorrausschau (Lookahead) sicherzustellen, daß zumindest in den folgenden Leveln kein solcher Fehler gemacht wird.

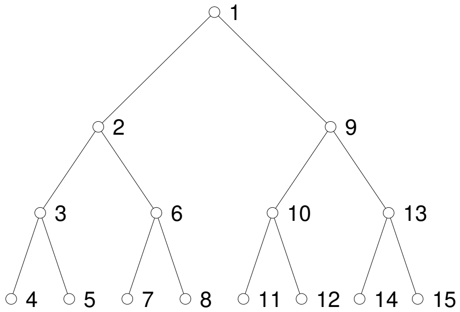

Abbildung 5.3: Depth-first Durchlauf

Bottom–Up Durchlauf

Beim Bottom–Up Durchlauf wird mit dem hierarchisch tiefsten Level begonnen. Natürlich muß dazu entweder ein maximaler Level bekannt sein oder angegeben werden. Bei einer diskreten Originalfunktion u ist dieser Level jedoch leicht angebbar. Im Durchlauf wird dann der Levelindex *level* rückwärts bis zum hierarchisch höchsten Level gezählt.

```
for level=maxlevel to 0 step -0.5
  for i,j in punkte(level)
    interpoliere punkt(i,j)
  endfor
endfor
```

Die Interpolation kann aber im allgemeinen nicht auf interpolierten Werten arbeiten. Die Bottom–Up Methode hat aber speichertechnische Vorteile, wie wir in 5.4 sehen werden.

5.3.3 Depth–First–Durchlauf

Der Depth–First–Durchlauf ist ebenfalls ein hierarchischer Durchlauf, nur daß der Baum nicht Level für Level betrachtet wird, sondern rekursiv durchgelaufen wird. Wir betrachten hierzu auch keine Teilgitter mehr, sondern benötigen nur die Baumstruktur aus 3.35.

Die Rekursion beginnt dabei beim hierarchisch höchsten Punkt und erfolgt dann kaskadenartig. Die Abbruchbedingung kann wiederum adaptiv oder levelabhängig sein. In Abb. 5.3 sehen wir den depth-first Durchlauf bis zu einem festen Level schematisch.

```
interpoliere_rekursiv(i,j)
  begin
    falls <abbruchbedingung>
      interpoliere punkt(i,j)
    sonst
      interpoliere_rekursiv(sohn_1(i,j))
      interpoliere_rekursiv(sohn_2(i,j))
      ...
      interpoliere_rekursiv(sohn_n(i,j))
    end
  end
```

Auch hier ist die Berechnung des Interpolationssterns einfach. Die Interpolation kann nur in Spezialfällen auf den interpolierten Werten arbeiten, z.B. wenn diese Werte im Funktionsaufruf übergeben werden können.

5.4 Speicherstrategien für bereits interpolierte Daten

Wir wollen uns nun um die geeignete Speicherung der interpolierten Daten während des Interpolationsvorganges kümmern. Es ist wichtig, daß der Zugriff auf diese Daten effizient erfolgen kann, da auf diese Daten während der Interpolation evtl. öfter zugegriffen werden muß.

5.4.1 Modifikation der Originaldaten

Falls die Originaldaten ein diskreter Datensatz sind, so kann ein interpolierter Wert im Speicher an der Stelle des Originalwertes abgelegt werden. Dieses Verfahren funktioniert prinzipiell mit beliebigen Datenstrukturen. Bei dieser Technik wird kein zusätzlicher Speicherplatz zur Speicherung der interpolierten Werte benötigt. Ein Nachteil hierbei ist natürlich, daß so Originaldaten verloren gehen.

5.4.2 Felder

Felder sind eine sehr einfache Datenstruktur mit schnellem Zugriff. Die interpolierten Daten werden einer ihrer x- und y-Position eindeutig zugeordneten Stelle im Feld abgelegt, der Zugriff erfolgt auf dieselbe Weise. Der Nachteil eines Feldes ist, daß es vorab statisch alloziert werden muß. Es muß dabei so groß alloziert werden, daß Punkte bis zu einem vorgegebenen maximalen Level darin abgelegt werden können. Müssen relativ zur Feldgröße nur sehr wenige Punkte abgespeichert werden, so bleiben allerdings weite Teile des Speichers unbenutzt.

5.4.3 Hash-Verfahren

Hash-Verfahren versuchen die bei Feldern entstehende Platzverschwendung zu minimieren. Hierzu wird vorab so viel Speicherplatz alloziert, wie man vermutet, daß benötigt wird, um alle Punkte abzuspeichern. Über eine Hashfunktion wird dann aus den Koordinaten des Punktes die Adresse des Eintrags im Speicher ermittelt. Diese Zuordnung ist allerdings in der Regel nicht eindeutig. Falls durch diese Hashfunktion zwei Punkte auf die gleiche Speicheradresse abgebildet werden, muß mit Hilfe von Kollisionsauflösungstechniken eine neue Adresse gefunden werden.

5.4.4 Bäume

Bäume sind im Gegensatz zu Feldern und Hashverfahren eine dynamische Datenstruktur. Sie werden nicht vorab alloziert, sondern während der Interpolation aufgebaut. Bei einem Baum muß allerdings zusätzlicher Speicherplatz für die Verzeigerungsstruktur aufgewendet werden. Die interpolierten Daten werden während der Interpolation in den Blättern gespeichert. Wenn allerdings später auf hierarchisch höher liegende Daten zurückgegriffen werden muß, so muß der Baum nach oben zurückverfolgt werden, bis das gesuchte Blatt gefunden wird. Zusätzliche Probleme kann die Behandlung des Gebietsrandes machen.

5.4.5 Parameterübergabe

Falls die Interpolation rekursiv erfolgt, können die interpolierten Werte auch als Aufrufparameter der Interpolationsfunktion implizit gespeichert werden. Da aber die Zahl der Aufrufparameter beschränkt ist, muß die Zahl der zu speichernden Werte konstant sein und alle interpolierenden Werte zu jedem Zeitpunkt zugänglich sein.

5.4.6 Hybridverfahren

Weiterhin sind noch Hybridverfahren zu erwähnen, die aus zwei oder mehreren der obigen Speichertechniken zusammengesetzt sind und die versuchen, die Vorteile dieser Techniken zu vereinen. So kann z.B. die Speicherung von Punkten auf höheren Leveln auf einem Feld und auf niedrigeren Level mit Hilfe von Bäumen erfolgen. Weiterhin kann die Effizienz des Verfahrens erhöht werden, indem die Information evtl. doppelt gespeichert wird, wie z.B. bei einer Implementierung der Dünngitterinterpolation, wo sowohl Bäume als auch Parameterübergabe zur Speicherung verwendet werden (siehe [8]).

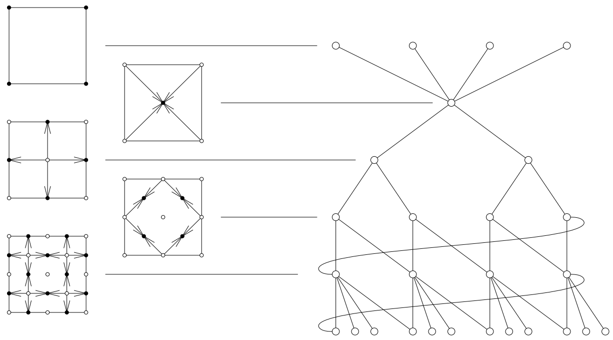

Abbildung 5.4: Vater–Sohn Relation für die Level 1-4

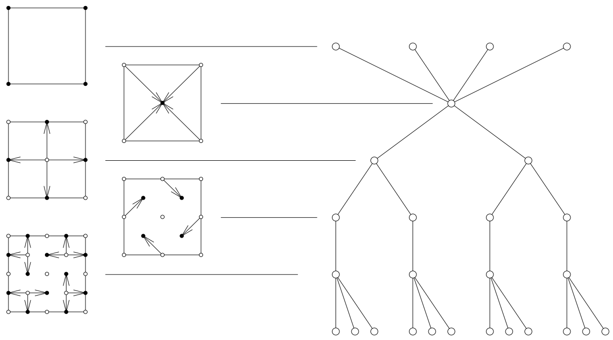

Abbildung 5.5: Beseitigung von doppelten Vätern

5.5 Anwendung

Wir wollen nun untersuchen, welche der in den vorigen Paragraphen besprochenen Strategien zur Interpolation, Speicherung und Durchlauf für die Interpolation mit semiregulären Gittern überhaupt anwendbar sind. Welche Strategien dann im konkreten Fall zur Anwendung kommen, wollen wir dann im nächsten Kapitel besprechen.

Zunächst wollen wir die Verwandschaftsfunktion $s(l,i)$ für semireguläre Gitter definieren. Ensteht ein Punkt $\{l,i\}$ durch die Verfeinerung eines Dreiecks T, so sei der Vater des Punktes der hierarchisch niedrigste Eckpunkt von T. Da ein Punkt durch die Verfeinerung von zwei Dreiecken doppelt entstehen kann, kann ein Punkt ein oder zwei Väter haben.

Diese Verwandtschaftsbeziehungen sieht man schematisch mit Hilfe des Graphen (Abb. 5.4). Man sieht, daß ein Depth–First–Durchlauf hier sehr ineffizient ist. Die Verwandtschaftsbeziehungen kann man allerdings auch eindeutig machen, wie in Abb. 5.5 gezeigt, wodurch aber die Vorfahrenrelation im Baumdurchlauf nicht mehr gegeben ist.

Bei linearer Interpolation kann eine Speicherung der interpolierten Werte über Parameterübergabe, wie im folgenden Programmstück gezeigt, geschehen. Dies entspricht auch einer Interpolation auf bereits Interpolierten Daten. Für bilineare Interpolation und Interpolation mit Greenschen Basen ist dies nicht möglich. Bei diesen Interpolationstechniken muß dann eines der anderen vorgestellten Verfahren verwendet werden, um die interpolierten Werte zu speichern. Der Durchlauf kann aber levelweise erfolgen.

```
interpoliere(Coord p1, Coord p2, Coord p3, Data f1, Data f2, Data f3)
  begin
    f := u((p1+p3)/2);
    h := f - 0.5*(f1+f3);
    falls h>epsilon
      begin
        interpoliere(p2, (p1+p3)/2, p1, f2, f, f1);
        interpoliere(p3, (p1+p3)/2, p2, f3, f, f2);
      end
  end
```

5.6 Speicherstrategien für komprimierte Daten

Wir wollen nun alle nicht mit einem Fehler ϵ interpolierbaren Daten zur späteren graphischen Darstellung speichern. Die fehlenden Punkte sollen dann vom Graphiksystem bei der graphischen Darstellung interpoliert werden.

Hierzu kann entweder die gleiche Datenstruktur wie für die interpolierten Daten verwendet werden oder eine Datenstrukturtransformation vorgenommen werden. Zum Beispiel können die auf einem Feld gespeicherten Daten nach Kenntnis der benötigten Punkte evtl. effizienter mittels einer Hashtabelle gespeichert werden. Zu beachten ist dabei, daß bei der Speicherung der bereits interpolierten Daten eine Unterscheidung zwischen den interpolierten und den Originalwerten gemacht wird.

Bei der Speicherung können alternativ die Funktionswerte oder deren hierarchische Überschüsse gespeichert werden. In beiden Fällen kann der Interpolant aus den gespeicherten Werten berechnet werden. Für die graphische Darstellung des Interpolanten werden allerdings die Funktionswerte benötigt, weswegen die Speicherung der hierarchischen Überschüsse ineffizient erscheint. Wir werden im nächsten Kapitel jedoch sehen, daß ihre Speicherung bei der adaptiven Zeichnung doch sinnvoll sein kann, wenn die auch die graphische Darstellung adaptiv erfolgen soll.

100					
90	80				
80	70	60			
70	60	50	40		
60	50	40	30	20	
50	40	30	20	10	0

Abbildung 5.6: Gouraud–Shading eines Dreiecks mit den Werten 100, 50, und 0 an den Ecken

5.7 Graphische Darstellung

Die graphische Darstellung erfolgt nachdem die gespeicherten Daten eingelesen wurden. Sie kann sehr effizient erfolgen, wenn die eingelesenen Daten werden depth-first durchgelaufen werden und die Dreiecke, die den Punkten an den Blättern entsprechen gezeichnet werden. Hierzu muß allerdings garantiert sein, daß für jeden Punkt auch sein hierarchischer Vater abgespeichert wurde.

```
zeichne(Coord p1, Coord p2, Coord p3, Level l)
  begin
    falls u((p1+p3)/2) nicht vorhanden
      zeichne_dreieck(p1, p2, p3)
    sonst
      zeichne(p2, (p1+p3)/2, p1, l+1) /* linkes Dreieck  */
      zeichne(p3, (p1+p3)/2, p2, l+1) /* rechtes Dreieck */
  end
```

Bei zweidimensionaler graphischer Darstellung können bei konstanter Interpolation einfarbige Dreiecke verwendet werden. Für lineare Interpolation stellt uns das Graphiksystem mit Gouraud–Shading ein Hilfsmittel zur Zeichnung zur Verfügung. Hierbei wird das Innere des Dreiecks linear zwischen den Eckpunkten interpoliert (Abb. 5.6). Für andere Interpolationstechniken, oder falls Gouraud Shading nicht vorhanden ist, kann diese Berechnung auch von der Applikation gemacht werden. Bei dreidimensionaler graphischer Darstellung kann der Interpolant bei linearer Interpolation durch Dreiecke ohne zusätzlichen Fehler gezeichnet werden. Durch die Zeichnung von entsprechend großen Dreiecken interpoliert die graphische Darstellung fehlende Werte.

Kapitel 6

Konkrete Anwendungsbeispiele

Wir wollen nun unser Verfahren beispielhaft auf zwei konkrete Probleme anwenden. Die Bildkompression und die Generierung eines dreidimensionalen Höhenmodells für Geländedaten stehen zwar nicht im direkten Bezug zur wissenschaftlichen Visualisierung, haben aber eine ähnliche Problematik.

In beiden Beispielen ist die Visualisierung integraler Bestandteil des Verfahrens und Echtzeitanforderungen sind gegeben. Die Datenmengen können bei herkömmlicher Handhabung leicht den Rahmen des Bewältigbaren sprengen, sie haben aber im allgemeinen keine optimalen Glattheitsvorraussetzungen.

6.1 Bildkompression

6.1.1 Problemstellung

Ein digitales Bild ist eine Menge diskreter Grau- oder Farbwerte auf einem in der Regel rechteckigen Gebiet. Je mehr Bildpunkte (Pixel) dabei zur Auflösung des Bildes verwendet werden, und je größer der Wertebereich (Farbtiefe) pro Bildpunkt gewählt wird, desto genauer wird das Bild dargestellt.

Sobald ein Bild in digitaler Form vorliegt, kann es auf unzählige Arten bearbeitet und manipuliert werden. Als mögliche Bereiche sind hier die Bildaufwertung, die Muster- und die Objekterkennung zu nennen. Die digitale Bildverarbeitung ist ein mittlerweile unerläßliches Instrument in vielen Anwendungsbereichen, wie der Astronomie, der Meteorologie, der Medizin und der Fertigungstechnik.

Speziell bei hohen Auflösungen steigt die Anzahl der benötigten Bildpunkte stark an, wodurch die Übertragung und Speicherung von Bildern aufwendig wird. Ist n und m die Menge der Bildpunkte in x- und y-Richtung, und W die zur Darstellung verwendeten Bits pro Bildpunkt, dann ist der benötigte Speicherplatz S des Bildes :

$$S = n \cdot m \cdot W \ \text{ Bit.} \tag{6.1}$$

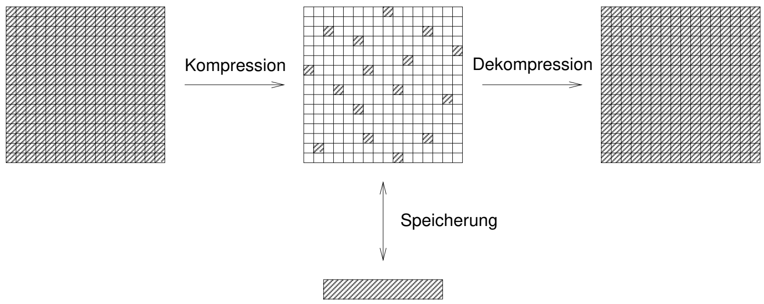

Abbildung 6.1: Bildkompression

Ein 768×576 Farbbild (PAL-Auflösung, wie sie zur Codierung von Fernsehbildern verwendet wird) mit 8 Bit Farbtiefe benötigt demnach ca. 3.5 MByte zur Speicherung. Bei diesen großen Datenmengen bemüht man sich notwendigerweise um die eine Reduktion der benötigten Daten. Der Kompressionsfaktor k ist hierbei der Quotient aus dem benötigten Speicherplatz des Originalbildes und des komprimierten Bildes:

$$k = \frac{S_{orig}}{S_{komp}} \tag{6.2}$$

Noch problematischer als bei Einzelbildern ist die Übertragung von Bildfolgen, wie sie z.B. bei Videokonferenzen benötigt werden. Damit für den Betrachter der Eindruck einer kontinuierlichen Bewegung entsteht, müssen dabei mindestens als 24 Bilder pro Sekunde dargestellt werden. Für solche Bildfolgen in obiger Auflösung wäre die dazu benötigte Übertragungsrate ca. 81 MByte pro Sekunde. Die Übertragung solch riesiger Datenmengen stellt selbst mit Hilfe der derzeit schnellsten Rechnernetze ein großes Problem dar.

Die momentanen Lösungsansätze hierzu sind eine Verkleinerung der Auflösung und die Verwendung von effizienten Kompressionsalgorithmen. Wir wollen im folgenden nun einige dieser Ansätze vorstellen.

6.1.2 Standardverfahren

Die einfachsten Vorgehensweisen zur Bildkompression sind eine Verringerung der Auflösung oder eine Verkleinerung des Wertebereichs (Quantisierung). Dies hat jedoch eine sichtbare Rasterung des Bildes oder eine Verfälschung von Farben besonders bei glatten Farbübergängen zu Folge. Heutzutage existieren eine Menge von Verfahren zur Bildkompression unterschiedlichster Qualität und Zielsetzung. Generell kann man diese Verfahren in zwei Gruppen einteilen:

- Verfahren ohne Informationsverlust
- Verfahren mit Informationsverlust

Hierbei ist anzumerken, daß ein Informationsverlust nicht notwendigerweise eine sichtbare Qualitätsminderung des Bildes zu Folge hat. Die Verfahren ohne Informationsverlust sind hierbei weitestgehend Standardverfahren zur Datenreduktion aus der Signalbearbeitung, wie die Huffman-Codierung, der Fano-Code oder Run-Length-Encoding.

Verfahren mit Informationsverlust orientieren sich in der Bildkompression an den psychooptischen Eigenschaften des Auges, wie z.B. der unterschiedlichen Unterscheidungsmöglichkeiten von Farbtönen im Farbspektrum. Zu diesen Verfahren gehören z.B. JPEG (für Bilder) oder MPEG (für Bildfolgen).

Entscheidend bei all diesen Verfahren ist die Effizienz. Besonders bei Bildfolgen muß vor allem die Dekompression in Echtzeit stattfinden können, damit für den Betrachter der Eindruck eines kontinuierliche Ablaufs von Bildern entsteht. Mit der heutigen Technologie sind solche Echtzeitanforderungen bei den angesprochenen Datenmengen nur mit Hilfe spezieller Bildkompressions-Hardware möglich.

6.1.3 Lösungsansätze

Wir wollen nun ein Verfahren zur Datenreduktion von Bildern angeben. Dazu betrachten wir ein Bild als zweidimensionale, diskrete Funktion auf einem regulärem vollen Gitter. Als Adaptivitätskriterium wählen wir ϵ als den maximalen Fehler, den das rückinterpolierte Bild haben darf. Alle Punkte, für die der hierarchische Überschuß $< \epsilon$ ist, können eliminiert werden.

Für den Fall $\epsilon = 0$ ist das Verfahren somit verlustfrei, d.h. alle Punkte werden entweder übernommen oder können exakt interpoliert werden. Falls $\epsilon > 0$ gewählt wird, wird das Verfahren verlustbehaftet. Der mathematische Fehler entspricht jedoch im allgemeinen Fall nicht dem optischen Fehler. Wir nehmen jedoch in unserem Verfahren den mathematischen Fehler als Näherung für den optischen Fehler.

Um möglichst hohe Kompressionsraten zu erhalten, versucht man den benötigten Speicherplatz der so interpolierten Bilddaten zu minimieren. Wir werden zeigen, daß gute Kompressionsraten erreicht werden können, indem interpolierbare Punkte geeignet markiert und dann die markierten Bilddaten mit Hilfe von Standard-Kompressionsverfahren effizient abgespeichert werden.

Wir werden bei der Kompression folgendermaßen vorgehen:

1. Originalbild einlesen

2. Originalbild interpolieren, hierbei interpolierbare Punkte markieren

3. Daten komprimieren

4. Komprimierte Daten abspeichern

Die Dekompression erfolgt dann auf folgende Weise:

1. Bilddatenspeicher allozieren
2. Gespeicherte Daten einlesen
3. Eingelesene Daten dekomprimieren und in Bilddatenspeicher übertragen, markierte Punkte rückinterpolieren
4. Bilddatenspeicher darstellen

Das Verfahren arbeitet umso effizienter, je mehr Schritte gleichzeitig ausgeführt werden können. Deswegen ist es günstig, die Daten in genau der Reihenfolge abzulegen, in der sie interpoliert werden. Im besten Fall können die Daten gleich im Anschluß noch komprimiert werden.

Speziell bei großen Bildern ist es wichtig, daß die Kompression auf dem Original-Datensatz arbeiten kann und keine Kopie von ihm erstellt werden muß. Falls die reduzierten Daten nicht während des Algorithmus ausgegeben werden, ist nur ein bottom-up Vorgehen möglich. Das begründet sich darauf, daß Punkte die markiert worden sind, nicht mehr zur Interpolation verwendet werden können.

Zur Kompression können Standardverfahren verwendet werden, die das häufige Vorkommen gleicher Werte ausnutzen. Bei der Lauflängencodierung (run length encoding, RLE) werden Werte die mehrmals hintereinander auftreten, nicht einzeln aufgeführt, sondern stattdessen wird der Wert selbst nur einmal und dann die Anzahl des Auftretens dieses Wertes abgespeichert. Bei der Huffman-Codierung wird jedem Wert eine Bitfolge zugeordnet. Diese Bitfolge ist um so kürzer, je häufiger der Wert in den Daten vorkommt. Diese Bitfolgen werden dann zusammen mit einer Zuordnungstabelle abgespeichert. Die Lempel-Ziv-Codierung arbeitet ähnlich wie die Huffman-Codierung, mit dem Unterschied, daß Wertefolgen anstatt von Einzelwerten betrachtet und codiert werden.

6.1.4 Beispielrechnungen

Wir haben als Testbild eine digitalisierte Grauwertversion des berühmten Bildes „Mona Lisa“ von Leonardo da Vinci gewählt (Abb. 6.2). Das Original ist im Louvre, Paris zu sehen, sofern es nicht von einer Menschenmenge verdeckt wird.

Wir wollen nun zunächst untersuchen, wie gut unser Interpolationsverfahren Bilder darstellen kann. Hierzu betrachten wir zunächst nur die Zahl der Punkte, die bei verschiedenen Basisfunktionen benötigt werden, um ein Bild bis auf einen Fehler ϵ zu interpolieren. Wir sehen, daß bei der konstante Interpolation am meisten und unter Verwendung von Greenschen Basen am wenigsten Punkte benötigt werden (Abb. 6.3). Der Unterschied zwischen den einzelnen Basisfunktionen ist allerdings nicht sehr groß.

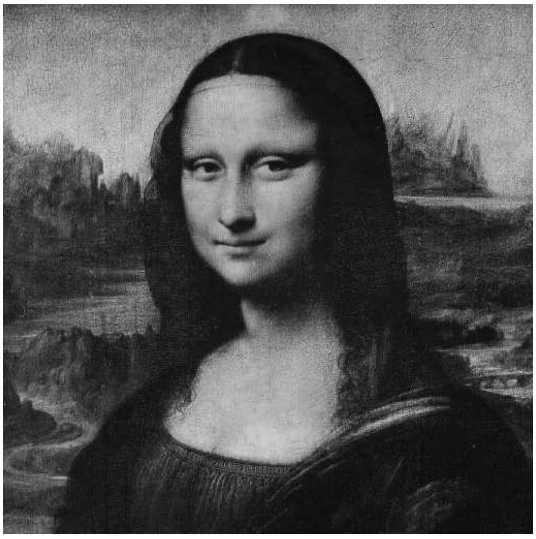

Abbildung 6.2: Bild `Mona`, 513x513 Bildpunkte, 255 (8 Bit) Graustufen

Im Abb. 6.4 sieht man, daß die Zahl der benötigten Punkte durch eine Glättung des Bildes mit Hilfe eines Gauß–Filters stark verringert werden kann. Darunter sehen wir, daß die besten Ergebnisse bei der Kompression erzielt werden, wenn das durch RLE komprimierte reduzierte Bild nochmals mit einem der anderen Verfahren komprimiert wird (Abb. 6.5).

Wir sehen nun auf den nächsten Seiten das Bild `Mona` nach der Rückinterpolation, bzw. die zur Interpolation verwendeten Punkte, bei verschiedenen ϵ. Bei jedem Bild ist die Zahl der benötigten Punkte angegeben. Der zulässige Fehler ist dabei von oben nach unten und von links nach rechts: 128, 64, 32, 16, 8 und 4 Grauwertstufen. Die ersten beiden Bildtafeln verwenden konstante Basen zur Interpolation, die anderen beiden verwenden lineare Basen.

Nun wollen wir die Qualität der dekomprimierten Bilder bei verschiedenen ϵ betrachten. Bei großem ϵ sieht man sehr schön die jeweils verwendeten Basisfunktionen. Bei $\epsilon = 16$ ist nahezu kein Unterschied mehr zum Originalbild zu erkennen. Bei der Interpolation mit konstanten Basen entsteht eine Aufteilung des Bildes in quadratische Farbflächen, wodurch Kanten verstärkt werden. Bei der Verwendung von linearen Basen entsteht bei größen ϵ eine Glättung des Bildes. Die Bilder bei bilinearen und Greenschen Basen unterscheiden sich nur durch einen noch stärkeren Glättungseffekt von diesem und wurden deswegen nicht gedruckt.

Insgesamt sehen wir, daß bei der Kompression des Bildes `Mona` gute Ergebnisse erzielt werden können, obwohl es nur sehr wenig glatt ist. Schon unter Verwendung nur sehr weniger Punkte kann man das Bild erkennen, allerdings werden relativ viele Punkte benötigt, um das Bild ohne optischen Verlust darzustellen.

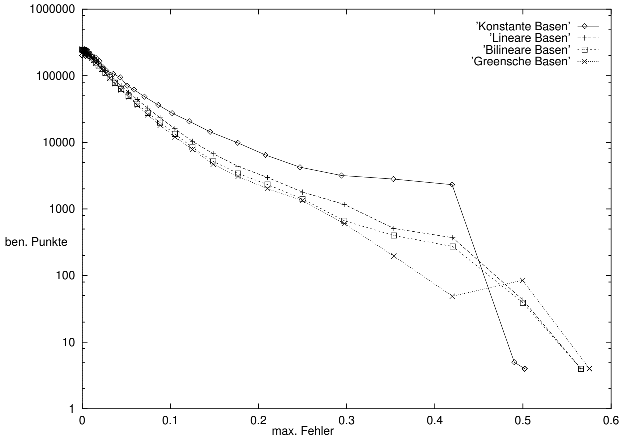

Abbildung 6.3: Interpolation von `Mona` bei verschiedenen Basisfunktionen

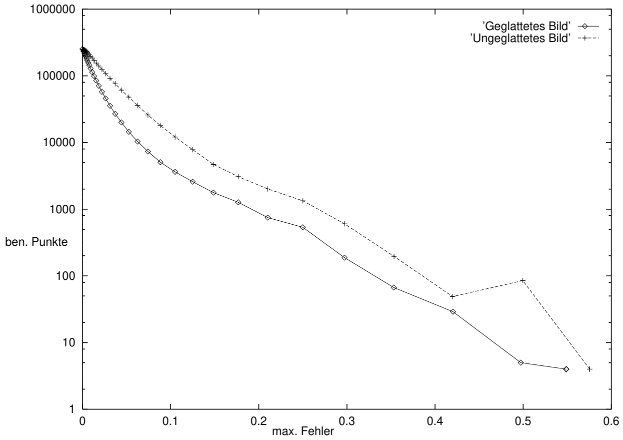

Abbildung 6.4: Benötigte Punkte bei `Mona` ungeglättet und geglättet

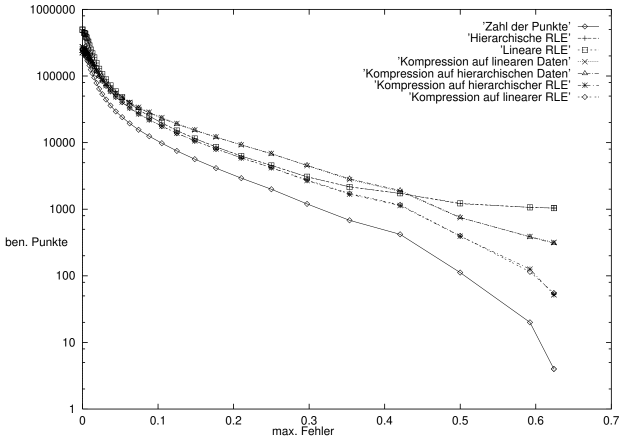

Abbildung 6.5: Benötigter Speicherplatz für `Mona` nach Kompression

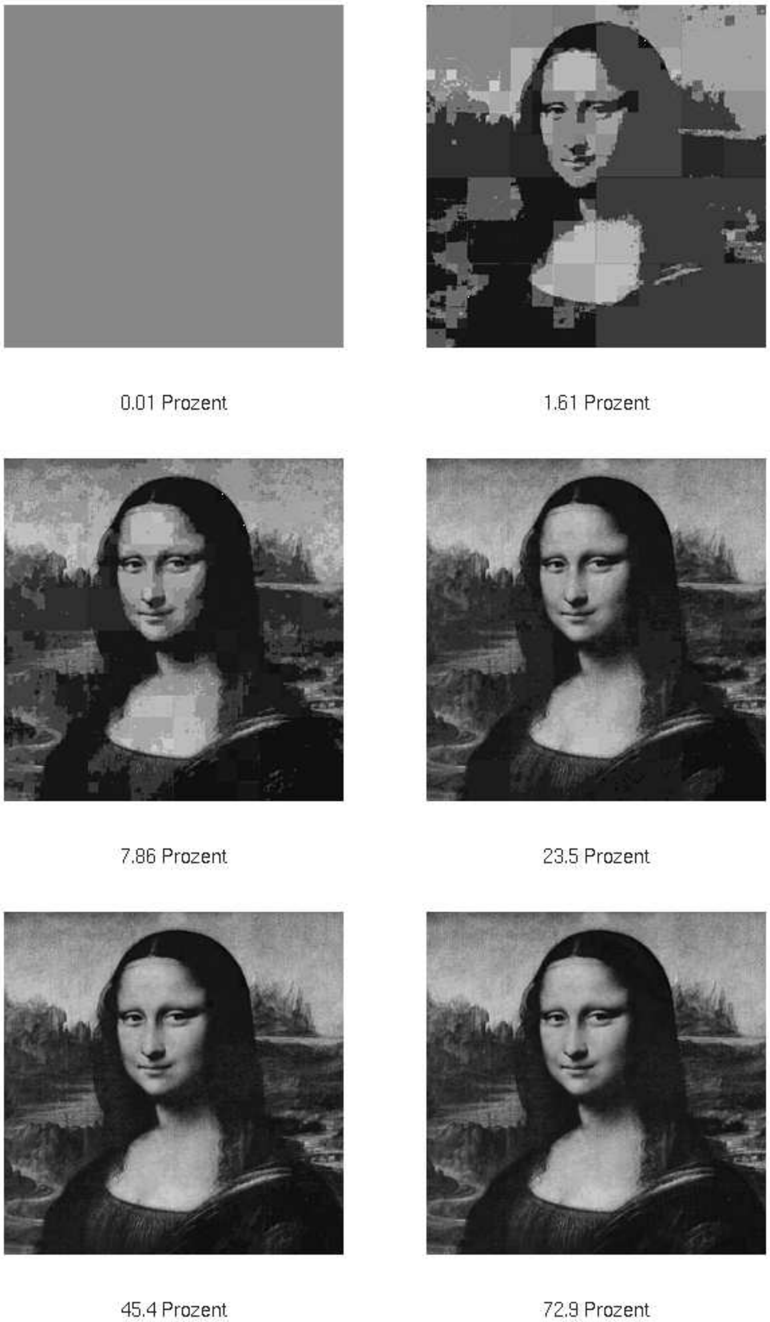

Abbildung 6.6: `Mona`: rückinterpoliertes Bild mit konstanten Basen

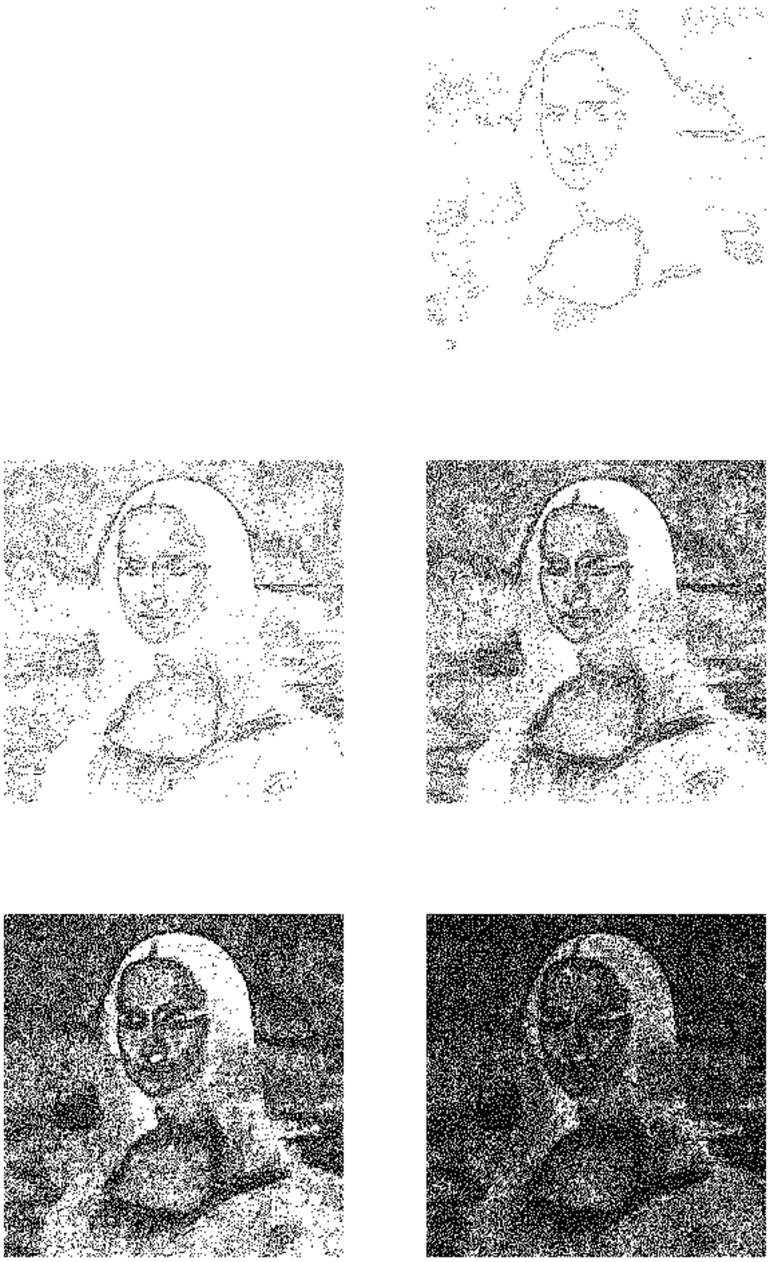

Abbildung 6.7: `Mona`: benötigte Punkte bei konstanten Basen

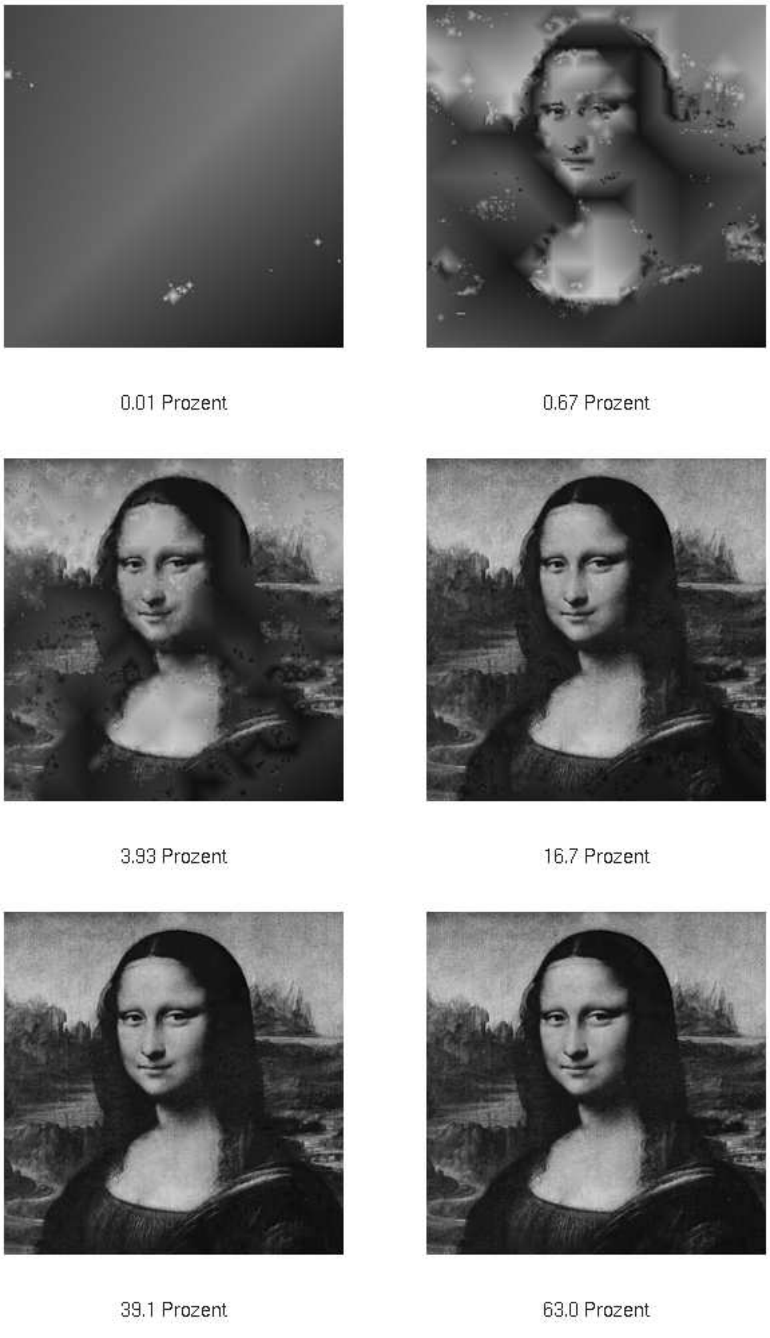

Abbildung 6.8: `Mona`: rückinterpoliertes Bild mit linearen Basen

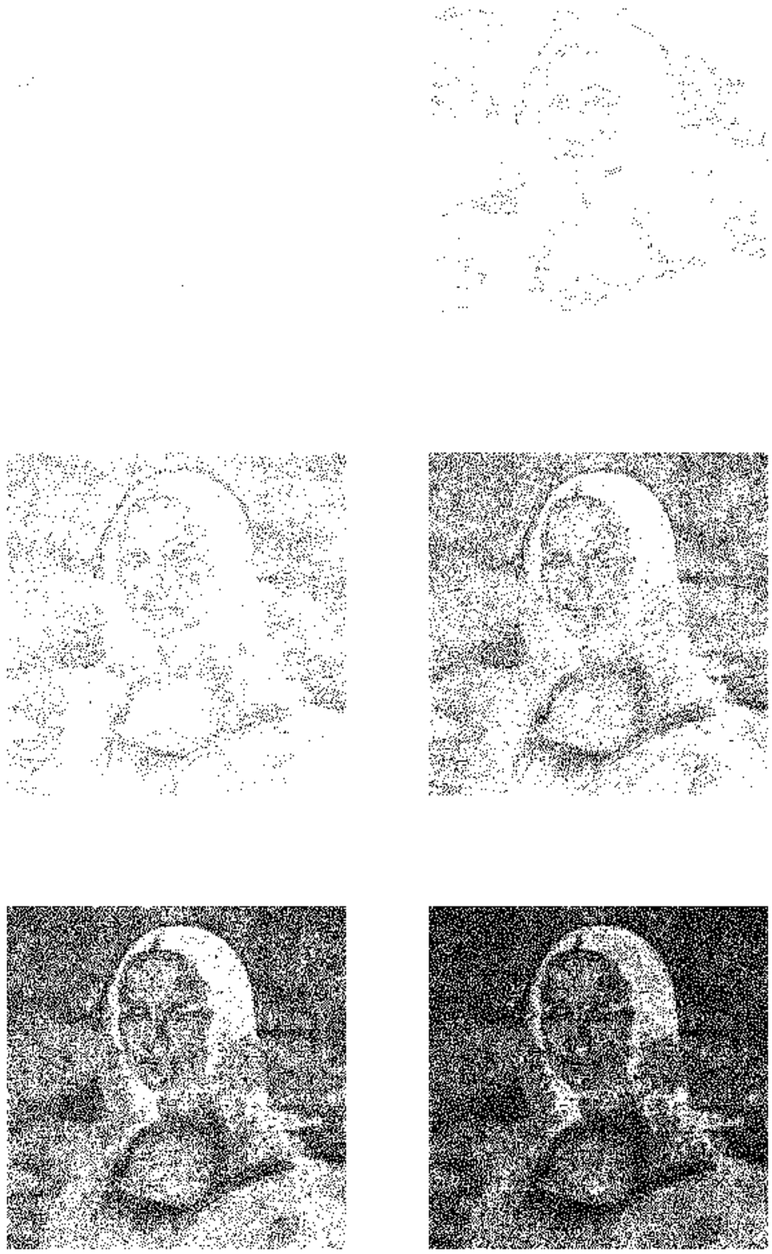

Abbildung 6.9: `Mona`: benötigte Punkte bei linearen Basen

6.1.5 Probleme und Verbesserungen

Verbessern ließe sich das Verfahren durch geeignetere, sich an psychooptischen Erkenntnissen orientierende, Adaptivitätskriterien. Der mathematische Fehler eignet sich in unserem Fall sich nur bedingt als Abschätzung für den optischen Fehler.

Bildfolgen können auf zwei Arten komprimiert werden. Zum einen können die Einzelbilder separat behandelt werden, oder die gesamte Bildfolge kann als ein einzelnes, dreidimensionales Gebiet von Bildpunkten interpretiert werden. Im letzteren Fall muß jedoch die gesamte Bildfolge vor der Kompression vorliegen.

Bei Videokonferenzen, einer typischen Anwendung, bei der Kompression von Bildfolgen notwendig ist, ist dies jedoch nicht der Fall. Hierbei werden zwei oder mehrere Teilnehmer einer Videokonferenz von Kameras aufgenommen, und die Kamerabilder an die übrigen Teilnehmer versandt. Falls die elektronische Verbindung zwischen den Teilnehmern nicht schnell genug ist, um die aufgenommenen Bilder in Echtzeit weiterzugeben, müssen diese komprimiert verteilt werden. Hierbei nimmt man auch einen Verlust an Bildqualität in Kauf.

Mit Hilfe unseres Verfahrens kann diese Kompression effizient, z.B. mit nachfolgender RLE, durchgeführt werden, da schon mit der Verwendung relativ weniger Punkten ein erkennbares Bild entsteht. Die Menge der zu übertragenden Daten, und damit auch die Qualität der Bilder kann mit Hilfe des ϵ der momentanen Leitungsgeschwindigkeit angepaßt werden.

In diesem Beispiel ist noch nicht die Möglichkeit der adaptiven Zeichnung, die unser Verfahren bietet, zum Zuge gekommen. In einem Bild werden normalerweise alle Bereiche als gleich wichtig angesehen. In unserem nächsten Beispiel können wir diese Eigenschaft nun ausnutzen.

6.2 Computergenerierte synthetische Sicht

6.2.1 Problemstellung

Schlechte Sichtverhältnisse, wie sie z.B. durch Nebel, Dunkelheit, Wolken, Regen oder Schnee verursacht werden, beeinträchtigen einen Piloten bei der Steuerung seines Flugzeugs. Besonders bei Start und Landung, wo der Pilot auf die Außensicht angewiesen ist, ist je nach Grad der Sichtverhältnisse die Flugsicherheit nicht mehr oder nur noch teilweise gewährleistet.

Eine computergenerierte synthetische Sicht kann zusammen mit einer geeigneten Anzeigetechnik hier die visuelle Information für den Piloten entscheidend verbessern. Zusätzlich können so dem Piloten Hilfen zur Flugführung angezeigt werden. Weiterhin findet ein mit einer solchen künstlichen Sicht ausgerüsteter Flugsimulator Anwendung im Flugtraining und in der numerischen Simulation der Flugeigenschaften von Flugzeugen.

Der Computer hat hierbei die Aufgabe, Landschaftsdaten, die aus den reinen Höheninformationen, Terraindaten und Objektdaten bestehen, und Flugführungsinformationen in Echtzeit darzustellen. Zudem muß der Computer in ständiger Interaktion mit dem Navigations- und Ortungssystem des Flugzeugs stehen.

Die Anforderungen an das Computersystem sind hierbei enorm. Nicht nur muß die Graphikeinheit eine möglichst exakte Simulation der Umwelt darstellen, auch die Bewältigung der für Darstellung nötigen Daten ist wegen den Genauigkeitsanforderungen problematisch. Nimmt man für Bayern ein rechteckiges Gebiet von 300 km mal 500 km an, legt darüber ein reguläres volles Gitter mit 50 m Gitterweite und codiert man die Höhendaten mit 16 Bit, so entspricht das einer Menge von ca. 114 MByte Höheninformationen. In der Nähe von Flughäfen muß dann zusätzlich noch genauer aufgelöst werden, weil hier die Darstellung noch genauer erfolgen muß.

Der Computer muß aus diesem Datensatz den zur Darstellung relevanten Ausschnitt auswählen und ihn darstellen. Mit herkömmlichen Methoden muß dabei die Graphikeinheit nicht selten mehrere hunderttausend Polygone pro Sekunde zeichnen können. Solch eine Speicher– und Graphikleistung ist zwar mittlerweile möglich, für den industriellen Einsatz aber aus Kostengründen nicht erschwinglich. Ziel ist deswegen eine effizientere Form der Darstellung und Speicherung der Höhendaten.

6.2.2 Standardverfahren

Üblicherweise werden die Höhendaten mit Hilfe eines regulären vollen Gitters gespeichert. Das Gitter wird über das darzustellende Gebiet gelegt und der jeweilige Höhenwert an jedem Gitterpunkt gespeichert. Das Gitter muß hierbei sehr fein gewählt werden, damit für den Piloten ein möglichst realistisches Bild der Umgebung, vor allem bei Tiefflügen, entstehen kann. Die höhere Auflösung in der

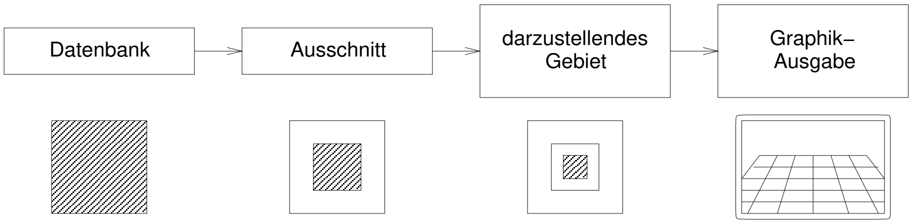

Abbildung 6.10: Datenfluß bei einem Flugsimulator

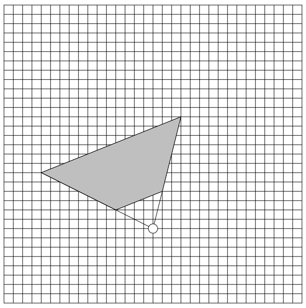

Abbildung 6.11: Der sichtbare Ausschnitt beim Flug über ein Gelände

Umgebung von Flughäfen wird mit Hilfe weiterer Gitter, die kleiner sind, dafür aber eine feinere Maschenweite haben, realisiert.

Neben den Höhendaten werden Geländedaten dargestellt, die angeben, in welchem Terrain (z.B. Wald, Felder, Berge) sich das dargestellte Gebiet befindet. Mit Hilfe dieser Daten werden die Höhendaten später bei der Darstellung eingefärbt oder texturiert, um den Betrachter ein realistischeres Bild der Umgebung zu geben.

Als dritten Datentyp werden dann noch Objektdaten benötigt. Objekte sind hierbei Häuser, Fernsehtürme, Hochspannungsmasten oder andere Gebäude, die die Sicherheit des Flugzeuges gefährden oder dem Piloten weitere wichtige Informationen geben können. Diese Objekte werden dann auf den eingefärbten oder texturierten Höhendaten plaziert. Aus Effizienzgründen, sowohl von der Speicherung als auch von der Darstellung her, werden diese Objekte standardisiert, d.h. verschiedene Häuser werden durch ein und dasselbe Objekt `Haus` dargestellt.

Von all diesen Daten machen die Höhendaten bei weitem den größten Teil aus. Wie oben schon angedeutet, passen die Daten schon bei groben Auflösungen auf relativ kleinen Gebieten nicht mehr in den Hauptspeicher eines normalen Rechners. Daher müssen diese Daten in einer externen Datenbank organisiert werden (Abb. 6.10).

Von den gesamten Höhendaten wird nur der Teil, den der Pilot gerade sehen kann, gezeichnet. Der darzustellende Teil des Gitters ist dabei ein Trapez, dessen Eckpunkte abhängig von der Position und der Orientierung des Flugzeugs sind (siehe Abb. 6.11).

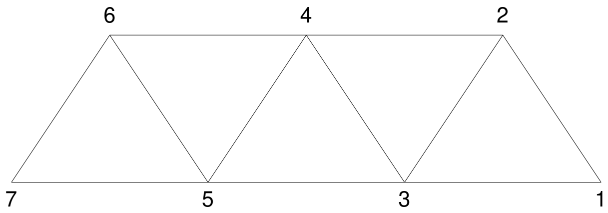

Abbildung 6.12: Einfaches `Mesh` aus Dreiecken

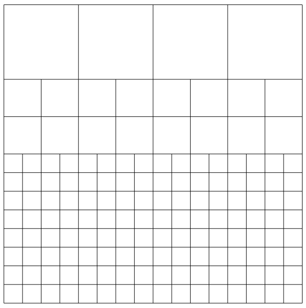

Abbildung 6.13: Blockbildung bei der Zeichnung von vollen Gittern

Bei der graphischen Darstellung dieses Ausschnitts muß dann jedes Rechteck, das ganz oder teilweise innerhalb dieses Trapezes liegt, gezeichnet werden. Das Rechteck wird hierzu nach einem vorgegebenen System (siehe Abb. 3.7) in zwei Dreiecke aufgeteilt, und diese dann gezeichnet.

Zur graphischen Darstellung müssen die 3-D Daten je nach Blickwinkel des Betrachters transformiert und dann in 2-D Bildschirmkoordinaten abgebildet werden. Nachdem die bei der Triangularisierung entstandenden Dreiecke viele gemeinsame Eckpunkte haben, werden diese Transformationen für einen hohen Prozentsatz der Punkte mehrfach erledigt, falls jedes Dreieck separat gezeichnet wird. Deswegen bieten Graphiksysteme die Datenstruktur `Mesh` (Abb. 6.12) an, bei der eine Kette von Dreiecken dem Graphiksystem übergeben werden kann. Das Graphiksystem kann nun mehrfache Eckpunkte erkennen und muß dadurch diese Transformationen nicht mehr mehrfach ausführen, was die graphische Darstellung erheblich beschleunigt.

Ein weiterer Lösungsansatz, die Graphik zu beschleunigen, entsteht durch die Beobachtung, daß die gezeichneten Dreiecke mit wachsender Entfernung vom Betrachter immer mehr, aber immer kleiner werden und damit für den Betrachter vom Informationsgehalt bedeutungsloser werden. Deswegen zeichnet man dort immer weniger, dafür aber größere Dreiecke.

Hierzu wird das darzustellende Gebiet in Blöcke unterteilt. In dem Block, der dem Betrachter am nächsten ist, wird jeder Gitterpunkt zur Triangulation verwendet, im nächsten nur jeder zweite, im übernächsten nur jeder vierte, usw. (siehe Abb. 6.13).

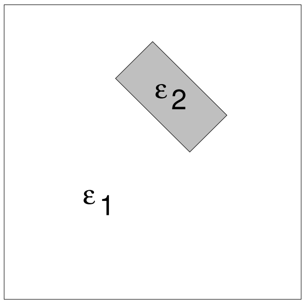

Abbildung 6.14: Unterschiedliche Wahl des ϵ_{min} , $\epsilon_2 < \epsilon_1$

Dabei besteht jedoch die Gefahr, daß die Landschaft verfälscht wird, wenn Punkte weggelassen werden. So kann die Spitze eines Berges verschwinden oder ein Tal sich verbreitern. Aus diesen Gründen kann mit der Blockbildung erst in einem unkritischem Abstand begonnen werden, was den Effizienzgewinn schmälert.

Auch die Komplexität der Daten bleibt ein Problem. Die Organisation der Datenbank ist aufwendig, der Zugriff langsam und kann nur durch geeignete Pufferung verbessert werden. Zusätzlichen Aufwand bedeutet die Spezialbehandlung von Flughäfen und ihrer Umgebung.

6.2.3 Lösungsansätze

Wir werden uns nun auf die effiziente Speicherung und Darstellung der Höhendaten beschränken, da sie den größten und somit problematischen Teil der anfallenden Daten ausmachen. Für die übrigen Daten können allerdings ähnliche Überlegungen angestellt werden.

Wir werden die schon gemachte Beobachtung ausnutzen, daß der Informationsgehalt der Dreiecke mit wachsender Entfernung vom Betrachter sinkt. Die Zahl der zu zeichnenden Dreiecke kann verringert werden, indem der Fehler, der bei der Zeichnung zugelassen wird, ebenso mit der Entfernung vergrößert wird. Somit müssen im Mittel mit wachsender Entfernung weniger, dafür aber größere Dreiecke gezeichnet werden.

Zunächst wollen wir uns der Frage zuwenden, welche Punkte generell abgespeichert werden müssen. Auch wenn bei der graphischen Darstellung bei den meisten Punkten ein größerer Fehler zugelassen wird, so muß dennoch auf das gesamte Gebiet bis auf einen Fehler ϵ_{min} zurückgegriffen werden können. Die Wahl dieses ϵ_{min} bestimmt damit die Größe des gesamten Speicherbedarfs an Höhendaten und sollte daher sehr sorgfältig getroffen werden.

Zum einen hängt ϵ_{min} vom Fehler der Originaldaten ab, die z.B. von einem Vermessungssatelliten stammen. Je größer dieser Fehler, desto größer kann auch ϵ_{min} gewählt werden. Weiterhin muß ϵ_{min} nicht über das gesamte Gebiet nicht gleich gewählt sein, sondern kann in Abhängigkeit von der Wichtigkeit des Gebiets gewählt werden, z.B. kleiner in der Nähe von Flughäfen (Abb. 6.14).

Die nun so auf einen Fehler ϵ_{min} reduzierten Höhendaten müssen nun geeignet gespeichert werden, sodaß der Zugriff möglichst effizient ist. Nachdem die Anzahl der abzuspeichernden Punkte und somit der benötigte Speicherplatz bekannt ist, bieten sich Hash–Verfahren an. Hierzu müssen zusammen mit jedem Punkt neben den Höhenwert des Punktes die Koordinaten des Punktes mit abgespeichert werden.

Durch den erhöhten Speicherbedarf lohnen sich Hash-Verfahren erst bei genügend hohem Kompressionsfaktor. Falls x–, y–Koordinate und Höhenwert je zwei Byte zur Speicherung benötigen, so ist die Speicherung mit Hash–Verfahren erst ab einem Kompressionsfaktor von 3 effizienter als mit einem Feld.

Falls die Menge der so komprimierten Daten nicht in den Hauptspeicher des Rechners paßt, muß weiterhin auf eine — allerdings kleinere als beim herkömmlichen Verfahren — externe Datenbank zugegriffen werden.

Eine spezielle Problematik ist die Wahl, ob die hierarchischen Überschüsse oder die Höheninformationen zur Speicherung verwendet werden Für die graphischen Darstellung werden die Absolutwerte benötigt, während als Adaptivitätskriterium die hierarchischen Überschüsse verwendet werden. Nachdem es aber das Ziel des Verfahrens ist, daß wesentlich weniger Punkte zur Zeichnung verwendet als reduziert werden, ist die Speicherung der Überschüsse sinnvoller.

Haben wir nun somit die komprimierten Daten vorliegen, können wir uns nun ihrer Darstellung zuwenden. Zunächst müssen wir das darzustellende Gebiet bestimmen, dann Punkte aus diesem Gebiet reduzieren und dann die Dreiecke auf den übrigen Punkten zeichnen.

Zur Reduktion der Punkte benötigen wir nun eine Vorschrift, die besagt, wie groß das ϵ für einen Punkt mit Abstand d vom Betrachter gewählt wird. Diese Adaptivitätsfunktion a

$$a : d \rightarrow \epsilon \tag{6.3}$$

kann verschieden gewählt werden. In Abb. 6.15 sehen wir eine Auswahl von Funktionstypen, die hierfür sinnvoll erscheinen.

Falls nun ein Punkt aufgrund der Adaptivitätsfunktion eliminiert werden kann, darf er nicht aus dem Speicher gelöscht werden, sondern muß markiert werden. Hierzu verwenden wir ein Bitfeld über alle Punkte, das angibt, ob ein Punkt zur Zeichnung verwendet wird oder nicht. Somit ergibt sich folgendes Vorgehen:

1. Zeichnendes Gebiet (Trapez) berechnen

2. Bitfeld auf 0 initialisieren

3. Bitfeld nach Adaptivitätsfunktion f aufbauen

4. Dreiecke aus Bitfeld zeichnen

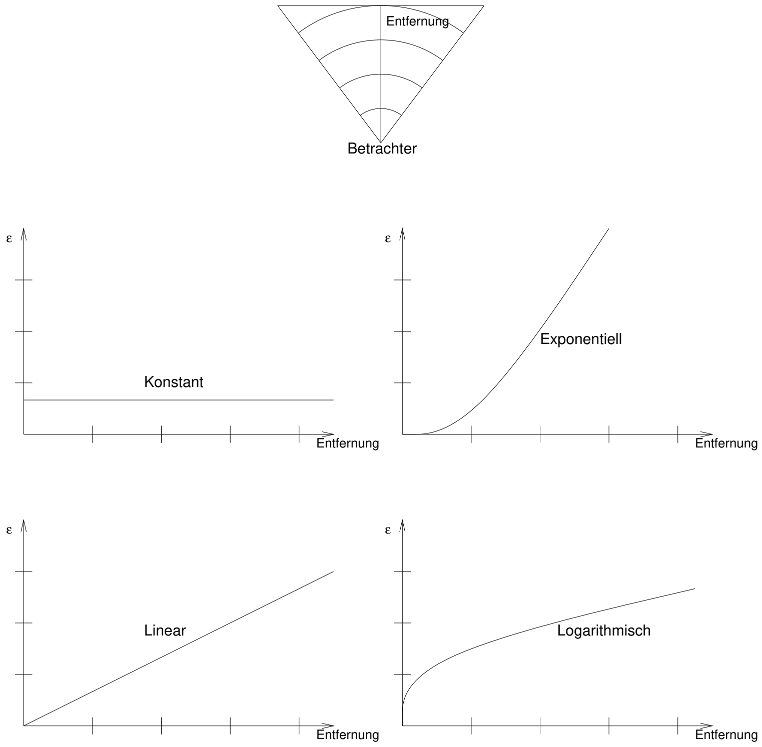

Abbildung 6.15: Verschiedene Adaptivitätsfunktionen

Zum Aufbau des Bitfeldes verwenden wir den bottom-up Algorithmus aus Kap. 5.2, der allerdings nicht auf dem Gesamtgebiet, sondern nur auf dem zu zeichnenden Gebiet arbeitet.

Um nun die Höhendaten zeichnen zu können, benötigen wir nun eine Zuordnung von den markierten Punkten zu den zeichnenden Dreiecken. Diese Zuordnung ist einfach, falls sichergestellt ist, daß wenn ein Punkt markiert ist auch alle seine Vorfahren markiert sind. Dabei kann diese Markierung rekursiv erfolgen. Sie ist effizient, weil man davon ausgehen kann, daß man, wenn man während des Durchlaufs auf einen schon markierten Punkt stößt, abbrechen kann, weil dann die Vorfahren jenes Punktes schon markiert sind. Aus diesem Grund führt auch der Bottom–Up Algorithmus führt zu keiner Fehlerakkumulation (siehe 5.2.1).

Die Zeichnung der Dreiecke erfolgt nun ebenfalls rekursiv. Wählt man beim rekursiven Baumdurchlauf abwechselnd das linke und das rechte Dreieck, so kann man sogar sukzessive ein `Mesh` aufbauen (Abb. 6.16). Da sichergestellt ist, daß für jedem Punkt auch seine Vorfahren vorhanden sind, ist es garantiert, daß die Dreiecke der Triangularisierung in einem `Mesh` aufzählbar sind.

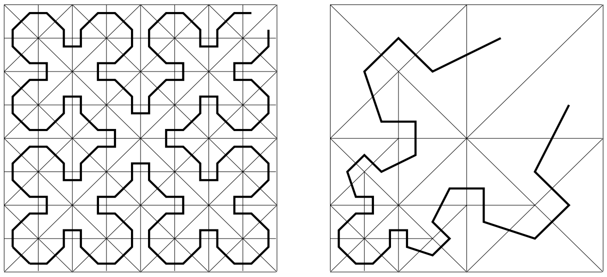

Abbildung 6.16: Reguläre und adaptive Mesh-Erzeugung

```
zeichne(Coord p1, Coord p2, Coord p3, Level l)
  falls (p1+p3)/2 nicht markiert oder l=maximaler Level
    zeichne_dreieck(p1, p2, p3)
  sonst
    falls Level gerade
      zeichne(p2, (p1+p3)/2, p1, l+1) /* linkes Dreieck  */
      zeichne(p3, (p1+p3)/2, p2, l+1) /* rechtes Dreieck */
    sonst
      zeichne(p3, (p1+p3)/2, p2, l+1) /* rechtes Dreieck */
      zeichne(p2, (p1+p3)/2, p1, l+1) /* linkes Dreieck  */
```

6.2.4 Beispielrechnungen

Getestet wurde das Verfahren mit Hilfe von Höhendaten, die freundlicherweise von Prof. Sachs (Lehrstuhl für Flugregelung und Flugmechanik, TU München) bereitgestellt wurden. Der benutzte Rechner war eine Indigo[2] der Firma Silicon Graphics.

Das Testgebiet ist etwa 46x30 km groß und befindet sich im Alpen- und Voralpenland in der Umgebung von Oberstdorf (Abb. 6.17). Das Gebiet wurde deshalb ausgewählt um die Qualität des Verfahrens auch für Höhendaten extrem wenig glatte Daten zu untersuchen. Weitere Informationen zu den Höhendaten und zur Wahl der Adaptivitätsfunktionen finden sich im Anhang B.2.

Für dieses Testbeispiel ist $\epsilon_{min} = 0$ gewählt worden. In der Tabelle 6.1 sieht man aber, daß die benötigten Punkte bei kleinem ϵ schon stark abfallen. In den Tabellen 6.2-6.5 wurden die gezeichneten Polygone für ein spezielles Gebiet aufgetragen. Das sichtbare Gebiet liegt in direkter Nähe von Oberstdorf. Man blickt dabei in einer Höhe von etwa 500 m über dem Boden vom Ortsmittelpunkt nach Süden in die Alpen.

Man sieht, daß die Zahl der gezeichneten Bilder pro Sekunde mit sinkender Zahl von Polygonen steigt, aber nicht im genau proportionalem Verhältnis. Dies liegt daran, daß für jedes Bild Aufgaben, wie z.B. das Löschen des Bildschirm-

Abbildung 6.17: Karte des Testgebiets

$\|e_{n,n}\|_\infty$	$\|e_{n,n}\|_2$	$\frac{\|e_{n-1,n-1}\|_2}{\|e_{n,n}\|_2}$	$\|p_{n,n}\|$	$\frac{\|p_{n-1,n-1}\|}{\|p_{n,n}\|}$
499.9765625	102.172634	0.0000000	452	0.000000
249.9960937	52.196493	1.9574617	1287	2.847345
124.9843750	25.454158	2.0486071	3786	2.941725
62.4531250	12.877254	1.9766759	9851	2.601954
31.2343750	6.492942	1.9832695	24967	2.534463
15.6093750	3.241723	2.0048849	54405	2.179076
7.7812500	1.526945	2.1230123	99312	1.825420
3.8750000	0.645206	2.3642895	151004	1.520501
1.9375000	0.240884	2.6784932	195343	1.293628
0.9375000	0.075962	3.1742059	226768	1.160871
0.0000000	0.000000	0.0000000	248687	1.096658

Tabelle 6.1: Fehler und benötigte Punkte bei linearer Interpolation

Fehler (ϵ)	Polygone	frames/sec
0	126239	0.34
5	13760	2.01
10	4253	4.03
15	2333	5.19
20	1291	6.06

Tabelle 6.2: Zu zeichnende Polygone bei konstanter Adaptivitätsfunktion

Fehler (ϵ)	Polygone	frames/sec
0	126239	0.34
5	24665	1.29
10	9693	2.50
15	5233	3.63
20	3293	4.54

Tabelle 6.3: Zu zeichnende Polygone bei linearer Adaptivitätsfunktion

Fehler (ϵ)	Polygone	frames/sec
0	126239	0.34
5	15588	1.45
10	5130	2.42
15	2685	2.90
20	1582	3.15

Tabelle 6.4: Zu zeichnende Polygone bei logarithmischer Adaptivitätsfunktion

Fehler (ϵ)	Polygone	frames/sec
0	126239	0.34
5	48842	0.69
10	37740	0.84
15	32905	0.94
20	29843	1.01

Tabelle 6.5: Zu zeichnende Polygone bei exponentieller Adaptivitätsfunktion

puffers, oder das Neuinitialisieren des Graphiksystems zusätzlich Zeit benötigen. Die graphische Darstellung bei logarithmischen und exponentiellen Adaptivitätsfunktionen wird zusätzlich durch die ihre aufwendige Berechnung verlangsamt.

Wir wollen nun die graphischen Darstellungen des Testgebiets für die verschiedenen Adaptivitätsfunktionen untersuchen (Abb. 6.18-6.22). Man sieht, daß man bei der Darstellung des vollen Gitters vor allem in der Weite die einzelnen Dreiecke nicht mehr erkennen kann. Demnach ist der Informationsgehalt dieser Dreiecke klein. In der Nähe hingegen würde man sich eine bessere Auflösung wünschen, was aber an der Vorgabe der Auflösung des vollen Gitters scheitert.

Wählen wir nun eine konstante Adaptivitätsfunktion, so verschlechtert sich die Auflösung gleichmäßig über das gesamte Gebiet. Dies hat zwar zur Folge, daß weiter entfernte Gebiete in einer gewünschten Auflösung gezeichnet werden, jedoch führt dies dazu, daß sich die Auflösung in der Nähe auch verringert.

Zu besseren Ergebnissen führt hier eine lineare Adaptivitätsfunktion. Selbst bei einer Steigerung von ϵ werden nur die Berge in der Weite gröber, während die feine Auflösung in der direkten Nähe erhalten bleibt.

Eine logarithmische Wahl der Adaptivitätsfunktion führt zu ähnlichen, nur leicht abgeschwächten Effekten, wie bei konstanten Adaptivitätsfunktionen. Die Auflösung in der Nähe wird zwar verbessert, aber bleibt nicht so gut erhalten wie bei linearen Adaptivitätsfunktionen. Die Zahl der gezeichneten Dreiecke ist allerdings deutlich niedriger.

Die Verwendung einer exponentiellen Adaptivitätsfunktion führt, wie zu erwarten war, zu den optisch besten Ergebnissen, da auch am meisten Polygone verwendet wurden. Die Kompressionsraten sind in diesem Fall allerdings auch geringer.

Ein bei der Darstellung einer computergenerierten synthetischen Sicht oft verwendetes Verfahren ist Depth–Cueing. Hierbei werden Dreiecke, in Abhängigkeit von der Entfernung zum Betrachter schwächer gezeichnet. Das hat den Effekt, daß weiter weg liegende Gebiete undeutlicher und dunkler zu sehen sind, als näher liegende. Dieses Verfahren läßt sich auch in unserem Fall anwenden, wodurch der Fehler durch die Zeichnung von gröberen Dreiecken kaschiert wird. Wir haben jedoch bei dem Druck der Bilder in dieser Arbeit auf diese Technik verzichtet, um den wirklichen Fehler zu zeigen.

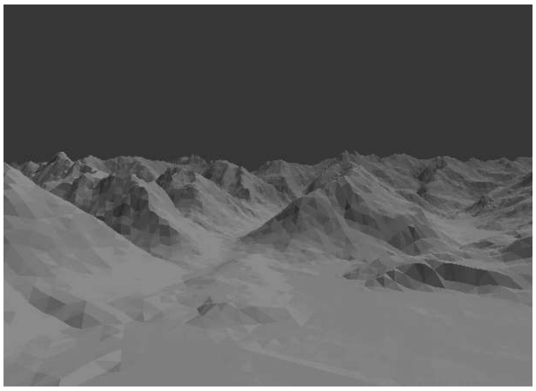

Abbildung 6.18: Volles Gitter

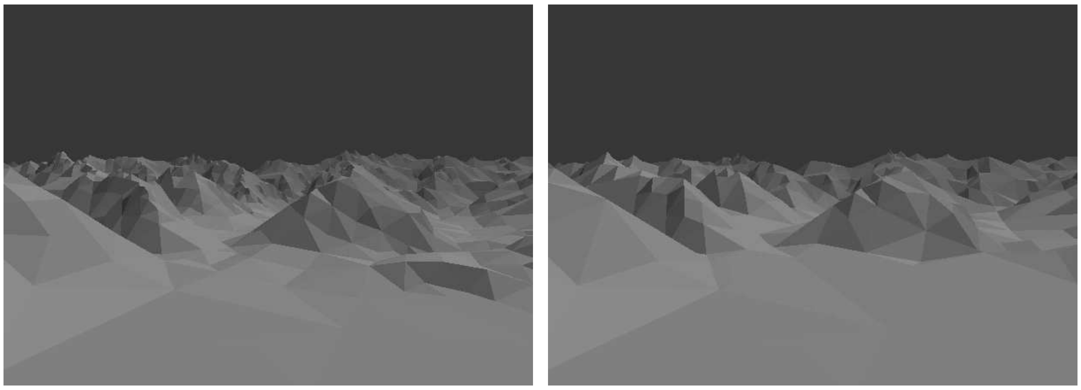

Abbildung 6.19: Konstante Adaptivitätsfunktion mit $\epsilon = 5, 10$

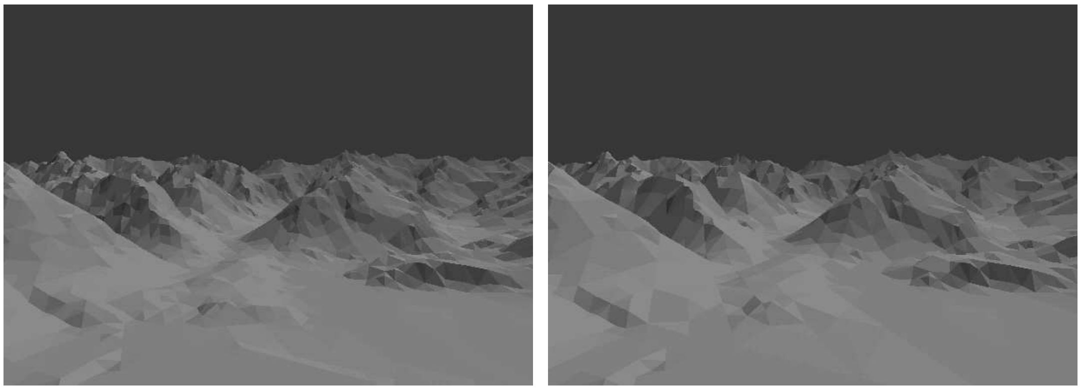

Abbildung 6.20: Lineare Adaptivitätsfunktion mit $\epsilon = 5, 10$

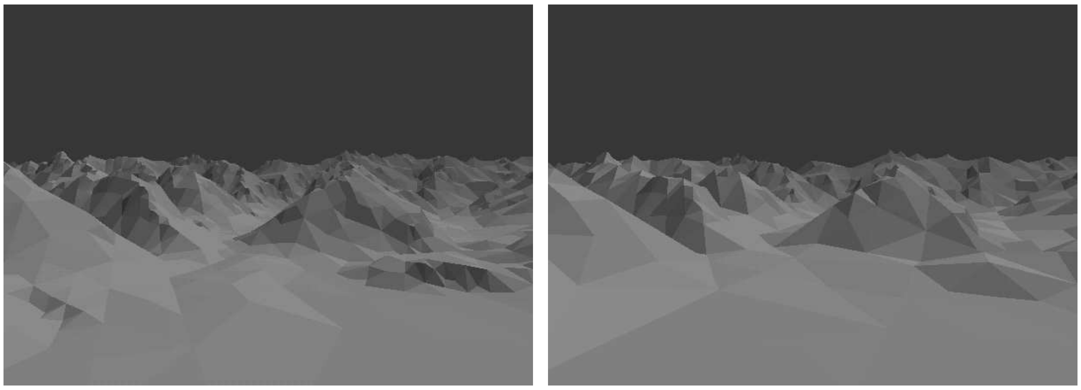

Abbildung 6.21: Logarithmische Adaptivitätsfunktion mit $\epsilon = 5, 10$

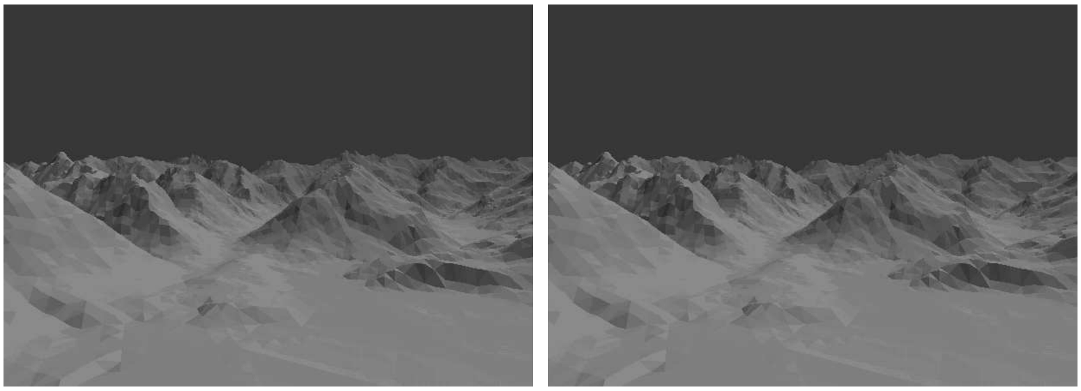

Abbildung 6.22: Exponentielle Adaptivitätsfunktion mit $\epsilon = 5, 10$

6.2.5 Probleme und Verbesserungen

Nach der Darstellung der reinen Höhendaten müssen nun die Objekt- und Geländedaten gezeichnet werden. Durch die fehlerbehaftete Zeichnung kann es vorkommen, daß ein Haus plötzlich halb im Boden versinkt oder in der Luft hängt, falls die Höhe des Geländes sich ändert. Die genaue Position der Objekte muß also bei jedem Schritt errechnet werden.

Da Dreiecke unterschiedlicher Größe gezeichnet werden, kann es vorkommen, bei der Zeichnung der Geländedaten ihr nicht mehr mit den Dreiecksrändern auf dem vollen Gitter übereinstimmt. Dadurch wird sich das Gelände zusammen mit den gezeichneten Dreiecken ändern.

Weiterhin problematisch ist die Tatsache, daß die Zeichnung beim hierarchisch höchsten Level beginnt. Wenn das Gesamtgebiet sehr viel größer ist als das zu zeichnende Gebiet entsteht so ein unnötiger Rechen- und Zeichenaufwand. Eine Lösung dieses Problems wäre ein Hybridverfahren, bei dem die obersten Level mit Hilfe eines vollen Gitters gespeichert werden.

Kapitel 7

Ausblick

Das vorgestellte Verfahren eignet sich gut zur Kompression glatter Daten, Höhendaten erfüllen die benötigten Glattheitsvorraussetzungen zum größten Teil, Bilddaten nur relativ wenig.

Trotzdem wäre es interessant, die Echtzeitfähigkeit des Verfahrens z.B. für Videokonferenzen genauer zu untersuchen. Speziell die Wahl geeigneter Adaptivitätskriterien ist hierbei interessant, damit sowohl Flächeninformation, als auch Strukturinformation geeignet übertragen werden. Weitere Verbesserungen lassen sich sicher durch eine adpative Wahl der Basisfunktionen erreichen. Hierzu werden jedoch zunächst geeignete Auswahlkriterien benötigt.

Das Verfahren zur Generierung eines synthetischen Höhenmodells für Geländedaten müßte nun in einen bestehenden Flugsimulator eingebaut werden. Hierbei ist zu untersuchen, ob Probleme bei der der Darstellung von Terrain- und Objektdaten auftreten oder bei der Interaktion mit dem Hauptprogramm auftreten. Diese Daten könnten weiterhin nach einem ähnlichen Schema komprimiert und dargestellt werden, was aber erst eine genauere Untersuchung erfordert. Zu untersuchen ist weiterhin eine korrekte Wahl der Adaptivitätsfunktion z.B. nach psychooptischen Erkenntnissen.

Weiterhin ist die Anwendung des Verfahrens auf allgemeinere Gebietsgrößen zu untersuchen. Durch die Verwendung von gedrehten Elementen können dabei krummlinig berandete Ränder mit Hilfe von Gebietstransformationen evtl. besser angenähert werden, als bei regulären oder dünnen Gittern.

Wir haben unser Verfahren so gewählt daß der Fehler bei der graphischen Darstellung $\hat{e}_g = 0$ ist (siehe 3.49). Läßt man einen graphischen Fehler z.B. durch die Verwendung anderer Basisfunktionen zu, so kann dies das Verfahren verbessern, wenn der graphische Fehler etwa in der gleichen Größenordnung wie der Interpolationsfehler ist.

Interessant ist die Verallgemeinerung des Verfahrens auf höherdimensionale Fälle. Das Teilraumschema und die Interpolationssterne lassen sich hierzu leicht erweitern. Die graphische Darstellung bei linearer Interpolation ist dabei aber wahrscheinlich nicht mehr so einfach und effizient möglich. Die Interpolationseigenschaften der anderen Basisfunktionen sind aber sicherlich untersuchenswert.

Speziell die Eigenschaften der Greenschen Basisfunktionen bedürfen einer genaueren Untersuchung. Wir konnten hier nur experimentell ihre guten Eigenschaften bei der Interpolation von harmonischen Funktionen zeigen. Die Verwendung von semiregulären Gittern und den entsprechenden Prolongationsoperatoren könnte zu vielversprechenden Ergebnissen bei der iterativen Lösung partieller Differentialgleichungen mit Mehrgitterverfahren führen.

Weiterhin wäre die logische Weiterfürung der Hierarchisierung, wie sie in Kapitel 4 vorgestellt wurde, untersuchenswert. Dabei kann soweit gegangen werden, daß nicht mehr mehrere Teilgitter zu einem Level zusammengefaßt werden, sondern jedem Teilgitter ein eigener Level zugeteilt wird. So lassen sich eventuell weitere Interpolationen höherer Ordnung leicht erzeugen.

Die Kombination des Verfahrens als Simulations- und Visualisierungsverfahren wäre in der numerischen Simulation sicherlich vorteilhaft, falls die Visualisierungsergebnisse schnell berechnet werden können. Die Simulation kann so interaktiv über die Visualisierung vom Benutzer gesteuert werden.

Anhang A

Programmbeschreibungen

A.1 Interpolation: `grid`

Das Programm `grid` interpoliert Funktionen, Bild- oder Höhendaten auf einem quadratischen Feld der vorgegebenen Größe $2^S + 1, S \in \mathbb{N}$. Die Größe der Bild- bzw. Höhendaten müssen hierzu mit diesem Feld übereinstimmen.

Das Abbruchkriterium ϵ wird nach der folgen Formel bestimmt:

$$\epsilon = 2^{\frac{-iteration}{E}} \cdot a$$

Das Programm versteht folgende Aufrufoptionen:

`-S` <d> : Größe des Feldes ($2^S + 1$)
`-E` <f> : E in obiger Formel
`-a` <f> : a in obiger Formel
`-e` <d> : Anzahl der Iterationen
`-t` <d> : Top-down (1) oder Bottom-up (0)
`-f` <d> : Keep-father (1)
`-x` <d> : Interpolation für ganze Level:
Green (0), Linear (1), Konstant (2)
`-y` <d> : Interpolation für halbe Level:
Green (0), Linear (1), Konstant (2)
`-p` <d> : Ausgabe der Daten in Dateien (1)
`-i` <s> : Datei mit Bilddaten
`-h` <s> : Datei mit Höhendaten
`-c` <d> : Flag für Farbbilder (1)
`-k` <f> : Kompensationsfaktor [0-1]
`-r` <d> : Flag für RLE codierung (1)
`-n` <s> : Präfix für Filenamen

Die Argumente für die Aufrufparameter sind vom folgenden Datentyp:

<d> : Ganze Zahl
<f> : Gleitpunktzahl
<s> : Zeichenkette

A.2 Flugsimulator: `sview`

Das Programm `sview` simuliert den Flug über eine Landschaft auf reduzierten Daten. Die Art der Reduktion und die Auswahl des zugelassenen Fehlers geschehen dabei interaktiv.

Beim Start des Programmes werden Höhen- und optional Texturinformationen eingelesen. Mit Hilfe der Texturinformationen, die im RGB Format sein muß und deren Größe mit den Höhendaten übereinstimmen muß, kann die Landschaft eingefärbt werden und eine Umgebungskarte dargestellt werden.

Das Programm reduziert dabei die Daten vor jedem Bild anhand des angegebenen Adaptivitätskriteriums und stellt dann die reduzierten Daten da.

Das Programm versteht folgende Aufrufoptionen:

-S <d> : Größe des Feldes (2^S)
-f <s> : Dateiname der Höhendaten
-t <s> : Dateiname der Texturdaten

Die Argumente für die Aufrufparameter sind vom folgenden Datentyp:

<d> : Ganze Zahl
<s> : Zeichenkette

Die Steuerung erfolgt dabei über Tastatur. Folgende Tasten haben dabei eine spezielle Belegung:

$\uparrow$ Steuerung nach oben
$\downarrow$ Steuerung nach unten
$\leftarrow$ Steuerung nach links
$\rightarrow$ Steuerung nach rechts
$>$ Beschleunigung
$<$ Bremsen
SPACE Umschalten zwischen vollen und reduzierten Informationen

Im unteren Teil des Bildschirms befinden sich einige Regler und Knöpfe, mit den folgenden Bedeutungen:

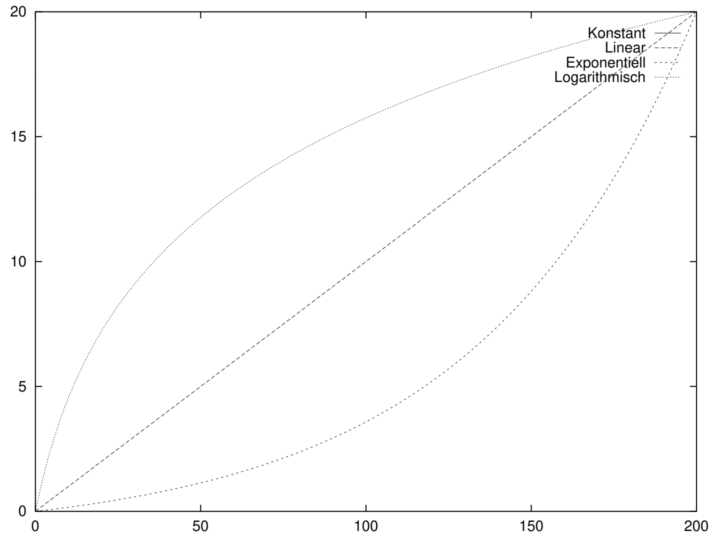

Abbildung A.1: Adaptivitätsfunktionen für $D = 200$ und $\epsilon = 20$

`Progression`	Einstellung der Steilheit der Adaptivitätsfunktion
`Far Clip`	Einstellung der maximalen Sichtweite
`Fog`	Einschalten von Depth Cueuing
`Map`	Einschalten der Karte (falls Texturdaten vorhanden)
`Texture`	Texturierung der Karte (falls Textur vorhanden)
`Gouraud`	Gouraud Schattierung (nur sinnvoll mit Texturdaten)
`Con`	Konstante Progression
`Lin`	Lineare Progression
`Log`	Logarithmische Progression
`Exp`	Exponentielle Progression

Außerdem werden die für das letzte gezeichnete Bild benötigten Polygone und die Anzahl der Bilder, die pro Sekunde gezeichnet werden, ausgegeben.

Folgende Funktionen wurden als Adaptivitätsfunktionen verwendet. Die Entfernung zum Betrachter sei d, der am weitesten entfernte Punkt sei D, ϵ sei ein Faktor für den zulässigen Fehler (Abb. A.1):

$$f_{konst} = \epsilon$$

$$f_{lin} = \frac{\epsilon \cdot d}{D}$$

$$f_{exp} = e^{\frac{\log(\epsilon+1)\cdot d}{D}}$$

$$f_{log} = \frac{\epsilon}{\log(\epsilon + 1)} \cdot \log(\frac{\epsilon \cdot d}{D} + 1)$$

Anhang B

Dateiformate

B.1 RGB–Bildformat

Das RGB Format von Silicon Graphics dient zur Speicherung von Bildern. Bilder werden bei der Speicherung durch RLE komprimiert. Jeder Farbkanal (Rot, Grün, Blau) wird dabei durch ein Byte repräsentiert. Für Grauwertbilder sind diese Werte alle gleich.

Die Ein- und Ausgabe von RGB Bildern wird durch die Standardfunktionen `longimagedata` und `writelongimagedata` realisiert. Im Speicher benötigt jeder Bildpunkt 4 Byte, da zusätzlich ein bei der Darstellung evtl. benötigter Helligkeitswert (alpha value) abgelegt wird.

B.2 Höhendaten

Die Höhendaten lagen uns als mit zwei Byte (short) codierte Höhenwerte auf einem 1200×1200 Gitter vor. Das zugrundeliegende Gitter hat eine Maschenweite von 90×60 Meter. Von diesen Werten wurde ein Ausschnitt von 513×513 Punkten in der Umgebung von Oberstdorf ausgewählt. Abbildung 6.17 zeigt eine Karte des Testgebiets.

B.3 Funktionsdaten

Die interpolierten Funktionsdaten werden von dem Programm `grid` im ASCII Format mit jeweils einem Wert pro Zeile als Gleitpunktzahlen linearisiert ausgegeben. Die einzelnen Spalten werden mit einer Leerzeile getrennt. Diese Form der Datenausgabe kann von vielen Funktionsplottern, wie z.B. `gnuplot` oder `IDL` eingelesen und dargestellt werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] *Graphics Library Programming Guide.* Silicon Graphics, Inc. März 1991. Document Number: 007-1210-040.

[2] ANSI, (American National Standards Institute): *American National Standard for Information Processing Systems - Programmer's Hierarchical Interactive Graphics System (PHIGS).* New York: ANSI: 1988.

[3] ANSI, (American National Standards Institute): *American National Standard for Information Processing Systems - Computer Graphics - Graphical Kernel System (GKS).* New York: ANSI: 1985.

[4] Bank, R.E.: *PLTMG Users' Guide, Edition 4.0.* 1985.

[5] Bank, R.E., T. Dupont und H. Yserentant: „The Hierarchical Basis Multigrid Method". *Konrad Zuse Zentrum Preprint, Berlin* **SC-87-2** (April 1987).

[6] Braess, D.: *Finite Elemente.* Heidelberg: Springer Verlag: 1992.

[7] Bronstein, I.N., und K.A. Semendjajew: *Taschenbuch der Mathematik.* 24. Aufl. Thun und Frankfurt/Main: Harri Deutsch: 1989.

[8] Bungartz, H.: *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung.* Dissertation. TU München, Institut für Informatik. Juni 1992.

[9] Cavaretta, A. S., W. Dahmen und C. A. Micchelli: *Stationary Subdivision.* Providence, Rhode Island: American Mathematical Society: 1991.

[10] Deuflhard, P., P. Leinen und H. Yserentant: *Concepts of an Adaptive Hierarchical Finite Element Code.* ZIB: 1988.

[11] Erdmann, B., J. Lang und R. Roiltsch: *KASKADE Manual, Version 2.0.* 1993.

[12] Foley, J., A. van Dam, S. Feiner und J. Hughes: *Computer Graphics - Principles and Practice.* 2. Aufl. Reading, Massachusettes: Addison Wesley: 1992.

[13] Frank, A.: *Hierarchische Polynombasen zum Einsatz in der Datenkompression mit Anwendung auf Audiodaten.* Diplomarbeit. TU München, Institut für Informatik. 1995.

[14] Griebel, M.: *Multilevel algorithms considered as iterative methods on indefinite systems.* SFB Bericht 342/29/91 A. Institut für Informatik, TU München. 1991.

[15] Griebel, M.: *Zur Lösung von Finite–Differenzen– und Finite–Element–Gleichungen mittels der Hierarchischen Transformations-Mehrgitter-Methode.* SFB Bericht 342/4/90 A. Institut für Informatik, TU München. 1990.

[16] Griebel, M., M. Schneider und Chr. Zenger: „A combination technique for the solution of sparse grid problems". In Groen, Pieter de, und Robert Beauwens (Hrsg.): *Iterative Methods in Linear Algebra: Proceedings of the IMACS International Symposium, Brüssel, 2.-4. 4. 1991.* Elsevier, Amsterdam: 1992: 263–281.

[17] Jackson, R., Lindsay MacDonald und Ken Freeman: *Computer Generated Color.* Chichester, United Kingdom: John Wiley & Sons: 1994.

[18] Keller, P., und M. Keller: *Visual Cues - Practical Data Visualization.* Los Alamitos, California: IEEE Computer Society Press: 1993.

[19] Mitchell, W. F.: *MGGHAT: Elliptic PDE Software with Adaptive Refinement, Multigrid and High Order Finite Elements.* 1993.

[20] Paul, A.: *Kompression von Bildfolgen mit hierarchischen Basen.* Diplomarbeit. TU München, Institut für Informatik. 1995.

[21] Ries, M., U. Trottenberg und G. Winter: „A Note on MGR methods ". *Sonderforschungsbereich 72, Universität Bonn* (1981).

[22] Rivara, M.C.: „Algorithms for Refining Triangular Grids Suitable for Adaptive and Multigrid Techniques". *International Journal for Numerical Methods in Engineering* **20** (1984) 745–756.

[23] Rüde, U.: *Mathematical and computational techniques for multilevel adaptive methods.* Frontiers in Applied Mathematics **13**. Philadelphia: SIAM: 1993.

[24] Stoer, J.: *Numerische Mathematik I.* 5. Aufl. Berlin Heidelberg: Springer Verlag: 1989.

[25] Stoer, J., und R. Bulirsch: *Numerische Mathematik II.* 3. Aufl. Berlin Heidelberg: Springer Verlag: 1990.

[26] Yserentant, H.: „Hiearchical Bases“. *in ICIAM 91: Proceedings of the Second International Conference on Industrial and Applied Mathematics* (1992).

[27] Yserentant, H.: „On the multi–level splitting of finite element spaces“. *Numerische Mathematik* **49** (1986) 379–412.

[28] Zenger, C.: „Sparse grids“. In Hackbusch, W. (Hrsg.): *Parallel Algorithms for Partial Differential Equations: Proceedings of the 6th GAMM-Seminar, Kiel, Januar 1990, Notes on Numerical Fluid Mechanics 31.* Vieweg, Braunschweig: 1991.

[29] Zenger, C.: *Sparse Grids.* SFB Bericht 342/18/90 A. Institut für Informatik, TU München. 1990.