Multiskalenmethoden zur Kompression und interaktiven Verarbeitung großer Datenmengen

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch–Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich–Wilhelms–Universität Bonn

vorgelegt von

Thomas Gerstner

aus

München

Bonn 2001

Angefertigt mit Genehmigung der Mathematisch–Naturwissenschaftlichen Fakultät der Rheinischen Friedrich–Wilhelms–Universität Bonn

1. Referent: Prof. Dr. Michael Griebel

2. Referent: Prof. Dr. Martin Rumpf

Tag der Promotion:

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Mengen an verfügbaren Daten sind in den letzten Jahren dramatisch angewachsen. Dies betrifft zum einen typische Meßdaten und zum anderen Ergebnisse von aufwendigen numerischen Simulationen. Zum Beispiel funken Satelliten heutzutage Terabytes an Höhen- oder Wetterinformationen pro Tag auf die Erde herunter. Medizinische Aufnahmesysteme, wie Computertomographie (CT) oder Magnetresonanzinferometrie (MRI) erfassen hochaufgelöste, dreidimensionale Bilder des menschlichen Körpers. Weiterhin erlauben mathematische Modelle eine Wetter- oder Klimavorhersage oder genaue Simulationen physikalischer Vorgänge. Ergebnis großangelegter Berechnungen, wie sie bei solchen Simulationen notwendig werden, können oft Gigabytes an Daten pro Zeitschritt der Simulation sein, welche dann geeignet ausgewertet werden müssen (siehe Abb. 1.1).

Während solche Simulationen oder Meßkampagnen Stunden oder gar Tage andauern können, erfordern auf der anderen Seite viele Anwendungen eine interaktive Verarbeitung der Daten. Beispiele dafür sind geographische Informationssysteme (GIS), (Flug-)Navigationssysteme, medizinische Bildverarbeitung, wissenschaftliche Visualisierung, oder auch Internet–basierte Kollaboration und Virtual–Reality Simulationssysteme. Dies betrifft nicht nur die reine Datenverwaltung und den Datenzugriff sondern auch die graphische Darstellung der Daten und Berechnungen mit ihnen.

Allerdings ist die Bandbeite heutiger Computersysteme in der Regel um Größenordnungen niedriger als die zur interaktiven Verarbeitung der Datenmengen benötigte. Dies betrifft sowohl den verfügbaren Speicherplatz als auch auch die Rechengeschwindigkeit und den Speicherdurchsatz, sowie die Graphikleistung und die Graphikauflösung. Dies ist ein prinzipielles Problem der Datenverarbeitung, welches in Zukunft eher noch zunimmt, da die verfügbaren Datenmengen und Ansprüche im Zuge der globalen Vernetzung immer schneller anwachsen.

Glücklicherweise ist es oft gar nicht notwendig, die Gesamtmenge an Daten auf einmal zu verarbeiten. Selbst wenn zum Beispiel eine graphische Darstellung von einer Milliarde Datenpunkten möglich wäre, wären solche Datenmengen vom Mensch gar nicht erfassbar. Viele Berechnungen erfordern auch nicht eine sofortige Bearbeitung aller Daten sondern müssen zu einem gegebenen Zeit-

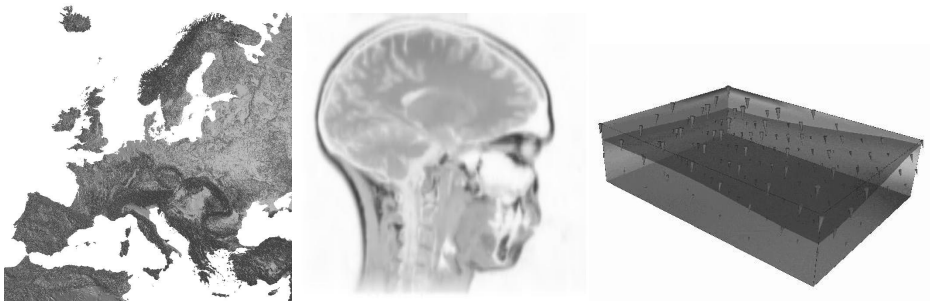


Abbildung 1.1: Typische Quellen für sehr große Datenmengen sind digitale Höhenmodelle, medizinische Bilder oder Ergebnisse numerischer Simulationen.

punkt lediglich auf einen begrenzten Teil der Daten zugreifen können. Vielmehr sind Interaktion und schnelle Antwortzeiten bei solchen Anwendungen von Bedeutung. Typische Anforderungen an eine geeignete Darstellung der Daten im Computer sind dabei:

- Effizienter beliebiger Datenzugriff
- Kompakte Darstellung der Daten
- Schnelle Ausführbarkeit von Operationen
- Skalierbarkeit der Algorithmen

Oft sind diese Anforderungen diametral zueinander. Zum Beispiel erlauben leistungsfähige Datenkompressionsverfahren in der Regel keinen effizienten Datenzugriff mehr. Zwar können solche Verfahren die Daten effizient speichern, um aber auf die Daten zugreifen zu können, müssen die Daten wieder zumindest partiell, oft sogar komplett entkomprimiert werden. Dadurch geht der Speichervorteil oft wieder verloren. Weiterhin benötigen viele Operationen auf den Daten zu ihrer effizienten Ausführung zusätzliche Informationen, die wiederum wertvollen Speicher verbrauchen. Zum Beispiel erfordern viele Algorithmen einen Zugriff auf die (geometrischen) Nachbarn eines Datenpunkts. Diese Information ist jedoch in einer unstrukturierten Punktmenge per se nicht enthalten und muß vorab berechnet werden, um zu vermeiden, daß bei einem solchen Zugriff die Gesamtmenge aller Datenpunkte durchsucht werden müssen. Unter Skalierbarkeit versteht man daher in diesem Kontext eine weitgehende Unabhängigkeit der Komplexität der Zugriffsalgorithmen und Operationen von der Gesamtdatenmenge. Datenmodelle mit guter Skalierbarkeit sind seit einigen Jahren ein zentrales Forschungsthema in vielen Anwendungsbereichen.

Multiskalenmethoden bieten einen guten Lösungsansatz für die oben aufgezeigten Probleme indem sie die Darstellung der Daten auf verschiedenen Detailstufen ermöglichen. Zur Visualisierung der Daten erlauben sie zum Beispiel eine Grobdarstellung der Gesamtdaten und bieten die Möglichkeit, sich lateral durch den Datensatz bewegen oder in interessante Gebiete hineinzoomen zu können (*pan & zoom interface*). In verteilten Anwendungen ermöglichen sie auch progressive Verfeinerung, d.h. ausgehend von einer groben Darstellung

der Daten werden Zusatzinformationen so lange berechnet und übertragen, bis der Benutzer den Vorgang abbricht oder alle Daten übertragen wurden. Dabei muß zu jedem Zeitpunkt die Darstellung der Daten konsistent sein. Darüber hinaus können viele Berechnungen mit hierarchischen Verfahren, zum Beispiel hierarchischen Suchalgorithmen, sehr effizient ausgeführt werden. Mehrgitterverfahren gehören zu den effizientesten Lösungsverfahren überhaupt, wenn die Berechnungsvorschrift durch partielle Differentialgleichungen vorgegeben ist.

Ziel dieser Arbeit ist es, die Einsatzmöglichkeiten von Multiskalenmethoden zur Verarbeitung sehr großer Datenmengen anhand verschiedener Beispielanwendungen aufzuzeigen. Dabei wird besonderes Augenmerk auf eine effiziente Darstellung der Daten im Speicher und die Interaktivität der entwickelten Algorithmen gelegt. Der hier gewählte Ansatz besteht aus der Kombination der folgenden drei Grundelemente:

- Hierarchische Datenorganisation
- Adaptive Verfeinerung
- Datenkompression

Eine hierarchische Datenorganisation erlaubt einen effizenten Datenzugriff und schnelle Ausführung von Suchoperationen. Adaptive Verfeinerung ermöglicht die effiziente Repräsentation der Daten und Fokussierung auf die interessanten Datenbereiche. Nicht zuletzt ermöglicht eine geeignete Datenkompression die effiziente Speicherung der Daten auf einem Speichermedium.

Bei der konkreten Konstruktion und Auswahl der einzelnen Verfahren ist offensichtlich darauf zu achten, daß die einzelnen Komponenten optimal zusammenarbeiten. Zum Beispiel muß die adaptive Verfeinerung auf die Datenhierarchie abgestimmt sein und das Kompressionsschema muß Berechnungen auf den komprimierten Daten erlauben. Ansonsten würden die Vorteile der einzelnen Verfahrenskomponenten wieder verloren gehen. Wie wir sehen werden, erlauben nur genau aufeinander abgestimmte Verfahrenskomponenten gleichzeitig eine optimale Laufzeit- wie auch eine optimale Speicherkomplexität. Der in dieser Arbeit gewählte Ansatz verwendet folgende Komponenten:

- Hierarchische Triangulierungen
- Saturierte Fehlerindikatoren
- Raumfüllende Kurven

Konkret basieren die hierarchischen Dreiecks- und Tetraedernetze zur Datenorganisation auf rekursiver Bisektion. Kantenbasierte saturierte Fehlerindikatoren steuern dann die adaptive Verfeinerung und raumfüllende Sierpinski–Kurven bilden die Grundlage für die Datenkompression. Wir werden sehen, daß diese Kombination einen sehr effizienten Umgang mit sehr großen Datenmengen für verschiedenste Anwendungen ermöglicht.

Insbesondere werden alle vorgestellten Algorithmen rekursiver Natur sein. Rekursion bietet einen natürlichen Zugang zu Multiskalenverfahren und erlaubt

eine gleichermassen elegante wie schnelle Implementierung. Auch wenn die betrachteten Verfahren prinzipiell dimensionsunabhängig sind, wird der Schwerpunkt in den Beispielen auf den in der Praxis wichtigsten Dimensionen zwei und drei liegen.

Der Beitrag dieser Arbeit liegt nun vor allem in der Entwicklung und Erweiterung von Verfahren zur interaktiven graphischen Darstellung (Visualisierung) großer Datenmengen. Insbesondere betrifft dies:

- die Entwicklung adaptiv hierarchischer Visualisierungsverfahren zur
  - Darstellung von Oberflächen und Isolinien [55, 57, 59, 60, 61],
  - Schnittbildung, Isoflächen–Extraktion oder Volumendarstellung [58, 59, 62, 63],
- die Berechnung von Fehlerindikatoren und Fehlerschranken zur Steuerung adaptiver Verfeinerung, insbesondere
  - Saturierungstechniken zur Vermeidung hängender Knoten [58, 59],
  - Saturierungstechniken zur Berechnung von minmax–Schranken zur Isolinien- und Isoflächen–Extraktion und für *multilevel backface culling* [58, 59],
  - Verfahren zur Topologie–Erhaltung und kontrollierten Topologie–Vereinfachung von Isolinien und Isoflächen [60, 62],
  - Erweiterungen des Fehlerindikator–Konzepts zur Fokussierung auf bestimmte Teilbereiche in den Daten [60],
- die Entwicklung von Kompressionsverfahren basierend auf
  - Wavelets mit Hilfe des *lifting* Schemas [57, 64],
  - raumfüllenden Kurven und relativen Sprungzeigern für hierarchische Triangulierungen [61],
- die Beschleunigung der Visualisierungsalgorithmen durch
  - Parallelisierung am Beispiel der Isoflächen–Extraktion [58],
  - hierarchische Sortierung bei der Darstellung mehrerer transparenter Isoflächen [63],

Die Anwendungsgebiete der Verfahren liegen dabei in den Bereichen GIS [57, 60, 61], Meteorologie [64, 134], Bildkompression [123], medizinische Bildverarbeitung [58, 59, 62, 63] und Chemie [59, 62, 63].

Diese Arbeit gliedert sich in sieben Kapitel, deren Aufbau sich wie folgt darstellt:

Kapitel 2 behandelt verschiedene Arten von hierarchischen Zerlegungen. Dabei werden hierarchische Zerlegungen zunächst in einem allgemeinen Kontext klassifiziert. Im folgenden werden hierarchische Zerlegungen basierend auf Simplexbisektion für strukturierte und unstrukturierte Daten erläutert.

Auf solchen Hierarchien aufbauende Interpolationsverfahren werden in *Kapitel 3* untersucht. Dabei wird zunächst allgemein Interpolation auf hierarchischen

Zerlegungen mit hierarchischen Finite Elemente Basen und Wavelets betrachtet. Dann werden verschiedene konkrete Konstruktionen für die Simplexbisektion angegeben.

Kapitel 4 untersucht dann die Berechnung von Fehlerindikatoren und adaptive Verfeinerung. Hierbei werden verschiedene Typen von Fehlerindikatoren konstruiert und qualitativ sowie quantitativ miteinander verglichen. Adaptivtät wird dann durch geeignete Saturierungstechniken und Schwellwertbildung der Fehlerindikatorwerte erreicht.

Speicherung und Kompression der Daten ist Gegenstand von *Kapitel 5*. Hierbei werden geeignete hierarchische Numerierungen der Simplizes hierarchischer Triangulierungen basierend auf raumüllenden Kurven eingeführt. Mit Hilfe dieser Numerierungen werden dann effiziente (Baum–)Kodierungs- und Kompressionsverfahren entwickelt.

Kapitel 6 zeigt verschiedene Anwendungsbeispiele aus verschiedenen Bereichen. Hierzu zählen die Konstruktion hierarchischer digitaler Höhenmodelle, adaptive Schnittbildung und Isoflächen–Extraktion, sowie Volumendarstellungen. Geeignete Erweiterungen des Fehlerindikator–Konzepts erlauben weiterhin die wichtige Topologieerhaltung der extrahierten Isolinien und Isoflächen.

Kapitel 7 schließt mit einem Ausblick auf Erweiterungen und weitere Anwendungsmöglichkeiten der vorgestellten Methodik.

Ich möchte an dieser Stelle all denjenigen danken, die mich bei dieser Arbeit unterstützt haben. Dies betrifft allen voran meinen Betreuer Prof. Dr. Michael Griebel für viele Anregungen und Diskussionen. Weiterhin bedanke ich mich bei den Co–Authoren meiner Artikel, die in diese Arbeit eingeflossen sind, Marc Hannappel, Renato Pajarola, Martin Rumpf und Ulrich Weikard, für die gute Zusammenarbeit. Nicht zuletzt möchte ich mich bei meinen Kollegen in der Abteilung für Wissenschaftliches Rechnen und Numerische Simulation an der Universität Bonn, insbesondere meinen Zimmerkollegen Thomas Schiekofer und Jochen Garcke, für ihre immer bereitwillige Unterstützung bei der Beantwortung meiner vielen Fragen bedanken.

Kapitel 2

Hierarchische Zerlegungen

Ziel dieser Arbeit ist die effiziente Verarbeitung großer Datenmengen. Wir wollen hier vor allem räumliche Daten $(\mathbf{x}_i, y_i)$ betrachten, d.h. jedem Datum sei neben seinem Wert $y_i \in \mathbb{R}$ auch ein Punkt $\mathbf{x}_i \in \mathbb{R}^d$ im Raum zugeordnet. Zum Beispiel hat bei einem digitalen Bild jeder Bildpunkt neben einem Farbwert auch eine Position im Bild. Diese Koordinaten leben alle in einem *Grundgebiet* $\Omega \subset \mathbb{R}^d$. Abhängig von der Anwendung kann das Grundgebiet sehr einfach gestaltet sein, z.B. ein Quadrat oder ein Würfel, aber auch sehr komplex, wie der Umriß eines Motorblocks oder eines menschlichen Kopfes.

Vorraussetzung für jeden Multiskalen-Algorithmus ist eine hierarchische Zerlegung des Grundgebietes. Auch wenn die Daten selbst nur an diskreten Punkten gegeben sind, ist eine kontinuierliche Sichtweise, d.h. die Betrachtung von Gebieten anstelle von einzelnen Punkten zum Verständnis von Multiskalen–Algorithmen sehr hilfreich. Kontinuierliche Modelle begegnen uns ohnehin sehr oft. Bei der Vergrößerung von digitalen Bildern werden kleine Quadrate oder Rechtecke anstatt einzelner separater Farbpixel gezeichnet. Beim Flug über eine virtuelle Landschaft erscheint das Gelände auch als kontinuierliche Fläche, selbst wenn die Höheninformationen nur an wenigen diskreten Meßpunkten vorgegeben sind. Wie wir aber sehen werden, impliziert eine hierarchische Zerlegung eines Gebietes in der Regel auch automatisch eine Hierarchie auf den Datenpunkten.

In diesem Kapitel werden wir im ersten Abschnitt zunächst hierarchische Zerlegungen in einem allgemeinen Kontext betrachten und klassifizieren. Im zweiten Abschnitt werden die in den folgenden Kapiteln verwendeten Konstruktionen basierend auf der Bisektion von Simplizes konkret für äquidistante Gitter und quadratische bzw. würfelförmige Grundgebiete definiert und auf ihre Eigenschaften untersucht. Der dritte Abschnitt dieses Kapitels wird dann diese Konstruktionen auf weniger strukturierte Grundgebiete, Gebietszerlegungen und Gitterstrukturen verallgemeinern.

2.1 Hierarchische Zerlegungen

Zerlegungen sind ein zentraler Bestandteil bei der Arbeit mit dem Computer in vielen Bereichen, zum Beispiel der rechnergestützten Geometrie, CAD, Datenbanken oder der numerischen Simulation. Hierarchische Zerlegungen sind spezielle Zerlegungen, die sie dadurch auszeichnen, daß sie verfeinerbar sind. Auf diese Weise können ausgehend von einer groben Zerlegung immer feinere (und genauere) Zerlegungen konstruiert werden. Als Urvater hierarchischer Zerlegungen gilt Archimedes, der vor über 2000 Jahren durch eine hierarchische Zerlegung des Kreises[1] die Zahl π erstaunlich genau berechnet hat.

In diesem Abschnitt wollen wir auf grundlegende Eigenschaften hierarchischer und nicht hierarchischer Zerlegungen eingehen sowie die prinzipiellen Möglichkeiten zur Konstruktion solcher Zerlegungen anhand einiger Beispiele erläutern.

2.1.1 Zerlegungen

Zerlegungen werden in vielen Fachgebieten eingesetzt. Nachdem viele Begriffe in den verschiedenen Bereichen unterschiedlich und zum Teil auch widersprüchlich verwendet werden, wollen wir im folgenden einige grundlegende Eigenschaften von Zerlegungen kurz definieren.

Eine *Zerlegung* $T = \{S_1, \ldots, S_m\}$ eines Grundgebiets Ω ist eine Unterteilung von Ω in kleinere *Teilgebiete* S_i, zum Beispiel Dreiecke oder Vierecke. Hierbei wollen wir davon ausgehen, daß sich die Teilgebiete nicht schneiden dürfen und daß die Vereinigung der Teilgebiete das gesamte Grundgebiet bilden. Eine Zerlegung kann verschiedene mehr oder weniger restriktive Eigenschaften haben:

- *uniform*: alle Teilgebiete sind gleich
- *semi–uniform*: alle Teilgebiete kommen aus einigen wenigen Kongruenzklassen
- *regulär*: alle Teilgebiete sind zueinander ähnlich
- *semi–regulär*: alle Teilgebiete kommen aus einigen wenigen Ähnlichkeitsklassen
- *homogen*: alle Teilgebiete sind vom gleichen Typ (z.B. nur Dreiecke oder Vierecke)

Insbesondere bei Triangulierungen[2] d.h. Zerlegungen in Dreiecke oder Tetraeder findet man oft noch eine weitere Eigenschaft:

[1] Er verwendete hierfür eine Folge ein- und umschreibender Polygone mit 3, 6, 12, 24, 48 und 96 Kanten.

[2] wir wollen den Begriff Triangulierung sowohl für den zwei- als auch für den dreidimensionalen Fall verwenden, da die Bezeichnungen Tetraedisierung oder Tetraederzerlegung ungebräuchlich sind und der vielleicht korrekte Begriff Simplizialkomplex in diesem Kontext normalerweise nicht verwendet wird.

- *zulässig*: der Schnitt zweier Teilgebiete ist entweder leer oder eine gesamte gemeinsame Seitenfläche, eine gesamte gemeinsame Seitenlinie oder ein gemeinsamer Eckpunkt

Viele dieser Eigenschaften sind sowohl aus theoretischer als auch aus praktischer Sicht wünschenswert. Bei uniformen oder regulären Zerlegungen spart man sich oft aufwendige Fallunterscheidungen und Mehrfachberechnungen. Auch aus Sichtweise der Datenkompression lassen sich reguläre Zerlegungen effizienter kodieren als inhomogene (siehe Abschnitt 5.1). Wie wir sehen werden sind insbesondere in höheren Dimensionen reguläre Zerlegungen schwer zu realisieren, daher wird dort oft auf semi–reguläre Zerlegungen zurückgegriffen. Zulässige Triangulierungen haben Vorteile bei der Konstruktion stetiger Interpolanten (siehe Kapitel 3).

Beispiele für Zerlegungen finden sich in Abb. 2.1–2.3. Das Bienenwabenmuster, das Rechtecksgitter und das rechtwicklige Dreiecksgitter sind sind uniforme Zerlegungen. Dreiecke und Quadrate lassen sich auch zu einer semi–uniformen Zerlegung kombinieren, oft werden dabei die Dreiecke zur Approximation des Gebietsrandes verwendet. Viele weitere Beispiele für uniforme und semi–uniforme Zerlegungen findet man in natürlicher Weise auch bei Kristallgittern (z.B. sog. Laves–Netze [119]). Die adaptiven Quadtree- und Dreiecksgitter sind wichtige Beispiele für nicht uniforme, aber reguläre Zerlegungen. Durch Kombination von Dreiecken und Vierecken lassen sich so auch semi–reguläre Zerlegungen realisieren. Die Delaunay–Triangulierung ist eine immer zulässige Zerlegung aber in der Regel nicht regulär sondern nur homogen. Weitere Beispiele für homogene Zerlegungen sind das relozierte Quadratgitter und allgemeine Triangulierungen. Nicht homogen sind allerdings Tesselierungen bestehend aus beliebigen Polygonen. Die letzten Beispiele sind im übrigen alle zulässig im Sinne obiger Definition.

2.1.2 Konstruktion von Zerlegungen

Seit vielen Jahrhunderten beschäftigt sich die Wissenschaft mit der Konstruktion von Zerlegungen. Ein schönes Beispiel ist die Vermessung des Königreichs Hannover, die Carl Friedrich Gauss Anfang des 19ten Jahrhunderts mit Hilfe einer Triangulierung der größten Städte durchgeführt hat[3]. Die Entwicklung neuerer Verfahren zur Konstruktion von Zerlegungen wird vor allem in den Bereichen FEM und CAD vorangetrieben.

Bei der Konstruktion von Zerlegungen gibt es vor allem zwei prinzipielle Fragestellungen:

- Konstruktion einer Zerlegung anhand einer vorgegebenen Punktmenge
- Konstruktion einer Zerlegung eines vorgegebenen Grundgebietes

Im ersten Fall bilden die (Daten-)punkte oft die Eckpunkte der Teilgebiete, manchmal auch deren Zentren. Wichtige Beispiele hierfür sind die Delaunay

[3]diese Triangulierung ist auf jedem deutschen Zehn–Mark–Schein abgebildet

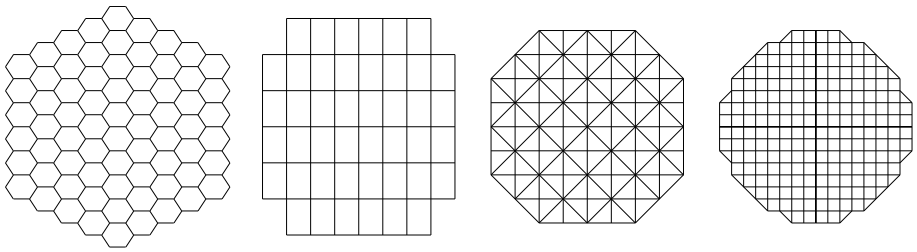


Abbildung 2.1: Uniforme Zerlegungen: Bienenwabenmuster, Rechtecksgitter, rechtwinkliges Dreiecksgitter und kombiniertes Quadrat/Dreiecksgitter.

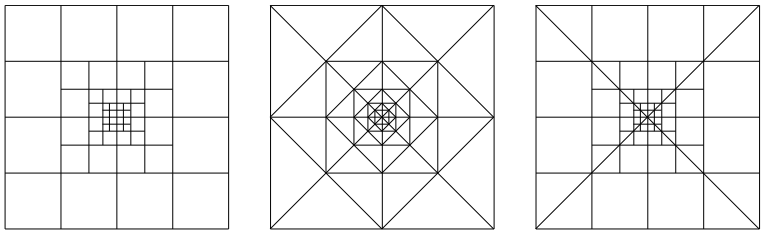


Abbildung 2.2: Reguläre Zerlegungen: Adaptives Quadtree- und adaptives Dreiecksgitter sowie kombiniertes Quadrat/Dreiecksgitter.

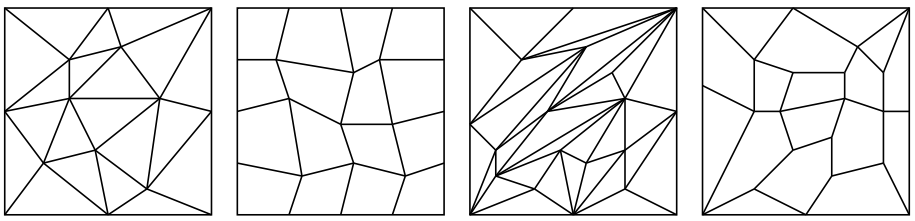


Abbildung 2.3: Irreguläre Zerlegungen: Delaunay Triangulierung, reloziertes Quadratgitter, zulässige allgemeine Triangulierung und zulässige allgemeine Tesselierung.

Triangulierung und das Voronoi Diagramm. Das Voronoi Diagramm ordnet jedem Datenpunkt das Gebiet zu, das aus denjenigen Punkten besteht, die dem Datenpunkt näher liegen als allen anderen Datenpunkten. Die entstehenden Teilgebiete sind demnach Polygone in 2D bzw. Polyeder in 3D. Die Delaunay Triangulierung entsteht durch Verbindung aller im Voronoi Diagramm benachbarten Datenpunkte. Die Delaunay Triangulierung stellt eine in gewisser Weise optimale Triangulierung einer gegebener Punktmenge dar, nachdem sie den minimalen Winkel aller Dreiecke bzw. Tetraeder der Triangulierung maximiert.

Das Voronoi Diagramm ist ein Beispiel für eine *gebietszentrierte Diskretisierung*, die Delaunay Triangulierung ein Beispiel für eine *knotenzentrierte Diskretisierung*. Je nach Anwendung erweist sich die ein odere andere Art der Diskretisierung als vorteilhalfter.

Im zweiten Fall werden oft ausgehend von einer Zerlegung des Gebietsrandes Punkte im Inneren des Grundgebiets geeignet eingefügt und anhand dieser Punkte das Gebiet wiederum wie im ersten Fall zerlegt. Auf diese Art und Weise wird die Gittergenerierung zur numerischen Simulation physikalischer Prozesse, wie zum Beispiel der Strömung um den Tragflügel eines Flugzeugs, durchgeführt. Die einfachste Art, solche Punkte einzufügen ist durch Überdeckung des Grundgebiets mit einem regelmäßigen Gitter. Eine weitere Möglichkeit ist durch (unter Umständen blockweise) Transformation einer leicht konstruierbaren Zerlegung eines einfachen Gebiets auf das komplexere Grundgebiet. Solche Zerlegungen werden oft noch in einem Nachbearbeitungsschritt bezüglich bestimmter Gütekriterien, wie Lage oder Aspekt der einzelnen Teilgebiete, optimiert. Weitere Informationen zu diesem Themenbereich finden sich in den Büchern [94, 135].

2.1.3 Hierarchische Zerlegungen

In vielen Anwendungen ist es notwendig, mit einem Datensatz nicht nur im Original sondern auch auf verschiedenen Detailstufen arbeiten zu können. Zum Beispiel bei der graphischen Darstellung einer komplexen Szene können so weit vom Betrachter entfernte komplexe Objekte, die auf nur ein paar Pixel des Bildschirms projiziert werden, in niedrigerer Detailstufe gezeichnet werden um die Interaktion zu verbessern. Vorraussetzung hierfür sind geeignete hierarchische Zerlegungen. Hierarchische Zerlegungen werden auch zur Konstruktion geometrischer Mehrgitterverfahren und zur Gitterverfeinerung von Diskretisierungen bei der Lösung partieller Differentialgleichungen benötigt.

Eine *hierarchische Zerlegung* ist eine Menge von Zerlegungen wachsender Komplexität wobei ausgehend von einer groben Zerlegung des Grundgebiets feinere Zerlegungen durch sukzessive Unterteilung realisiert werden. Hierbei ordnet man einer Zerlegung T^l einen *Level* l zu, wobei wir der gröbsten Zerlegung der Level 0 zuordnen wollen und eine Zerlegung von Level $l+1$ durch Unterteilung der Zerlegung von Level l entsteht. Hierbei besteht eine *Unterteilung* aus ein oder mehreren voneinander unabhängigen Unterteilungsschritten. Ein *Unterteilungsschritt* wiederum ersetzt ein oder mehrere (in der Regel nebeneinander liegende) Teilgebiete durch eine größere Menge an kleineren Teilgebieten.

Eine hierarchische Zerlegung heißt uniform, regulär oder homogen, sofern die Zerlegungen aller Level diese Eigenschaft haben. Weitere wichtige Eigenschaften

von hierarchischen Zerlegungen sind:

- *vollständig*: es werden jeweils alle Teilgebiete eines Levels unterteilt
- *regelmäßig*: es wird immer der gleiche Unterteilungsschritt verwendet
- *geschachtelt*: ein Unterteilungsschritt ersetzt immer nur ein einzelnes Teilgebiet
- *gebiets–vereinfachend*: gröbere Zerlegungen überdecken nicht das ursprüngliche Grundgebiet sondern Approximationen davon
- *dimensionsunabhängig*: der Unterteilungsschritt kann praktisch in beliebigen Dimensionen angewendet werden

Unvollständige Zerlegungen werden oft auch *adaptive*, d.h. angepaßte, Zerlegungen genannt, weil je nach Problemstellung nur bestimmte Gebiete unterteilt werden. Wir wollen hier im folgenden vor allem vollständige hierarchische Zerlegungen betrachten, Adaptivität wird dann Gegenstand eines eigenen Kapitels (Kapitel 4) sein. Wie wir in Kürze sehen werden, können insbesondere regelmäßige, geschachtelte Zerlegungen durch einfache rekursive Algorithmen realisiert werden. Die Verwendung einheitlicher Unterteilungsschritte erleichtert weiterhin die Konstruktion zulässiger Zerlegungen. Gebiets–vereinfachende Zerlegungen sind oft notwendig um hierarchische Zerlegungen auf komplexeren Grundgebieten konstruieren zu können, da ansonsten grobe Zerlegungen am Rand des Gebiets möglicherweise zu fein aufgelöst sein müßten. Dimensionsunabhängige Zerlegungen sind besonders wichtig für Anwendungen[4] in höheren Dimensionen (d.h. größer als drei).

Beispiele für hierarchische Zerlegungen finden sich in Abb. 2.4–2.6. In diesen Bildern entspricht der Level der Zerlegung der Liniendicke, wobei die Dicke mit dem Level abnimmt. Rechtecks- oder Quadergitter lassen sich leicht durch Koordinatenbisektion hierarchisch zerlegen. Weitere uniforme hierarchische Zerlegungen basieren auf Bisektion oder Quadrisektion von Dreiecken. Es wird angenommen daß die vier angegebenen Zerlegungen die einzigen uniformen hierarchischen Zerlegungen in zwei Dimensionen sind [119]. Das Bienenwabenmuster ist zwar eine Zerlegung, aber keine hierarchische, da Summe von Sechsecken kein größeres Sechseck ergibt. Allerdings läßt sich mit Hilfe von gleichseitigen Dreiecken und Sechsecken eine semi–uniforme gebiets–vereinfachende hierarchische Zerlegung realisieren. Durch semi–uniforme Zerlegungen bestehend aus Quadraten und Dreiecken lassen sich auch Gebietsränder gut hierarchisch approximieren. Hyperquader beliebiger Dimension lassen sich durch Koordinatenbisektion leicht hierarchisch zerlegen. Eine weitere dimensionsunabhängige hierarchische Zerlegung basiert auf der Bisektion von Simplizes, im gezeigten Beispiel Tetraeder.

Aus numerischer Sicht weitere wichtige Eigenschaften von hierarchischen Zerlegungen sind:

- *quasi–uniform*: das Verhältnis der Größen der einzelnen Teilgebiete ist nach oben beschränkt

[4] wie Data Mining, die Numerische Lösung der Schrödingergleichung, die Untersuchung von Warteschlangensystemen oder die Simulation von Polymerketten

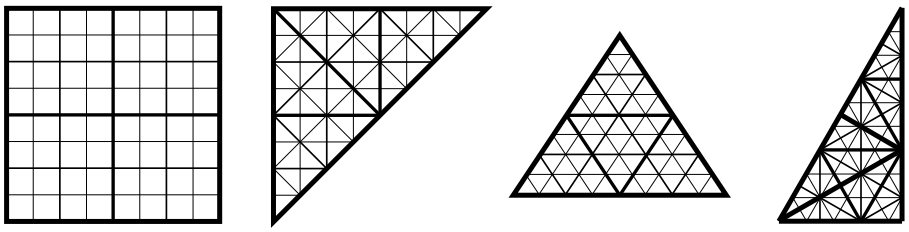


Abbildung 2.4: Uniforme hierarchische Zerlegungen: Quadrate, gleichschenklig rechtwinklige Dreiecke, gleichseitige Dreiecke und rechtwinklige Dreiecke mit 30 Grad Winkel.

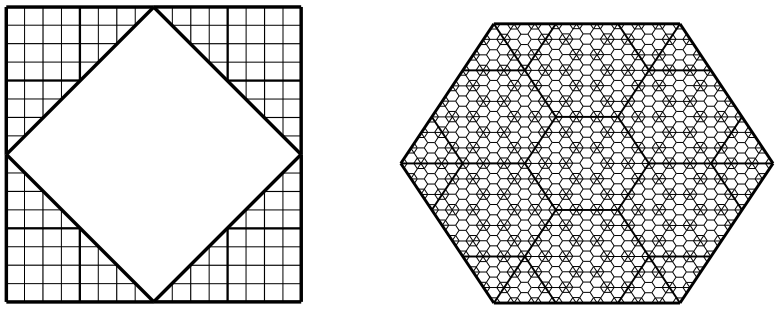


Abbildung 2.5: Semi–uniforme hierarchische Zerlegungen: Vierecke und Dreiecke, sowie Sechsecke und Dreiecke.

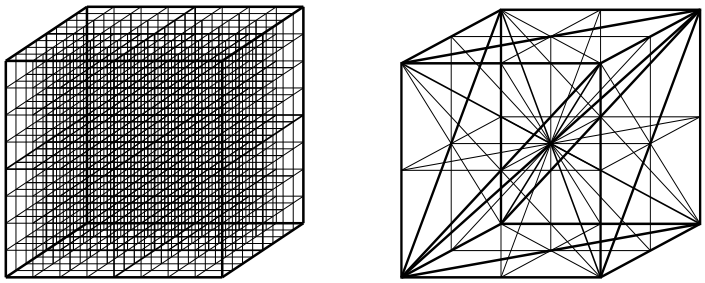


Abbildung 2.6: Uniforme und semi–uniforme hierarchische Zerlegungen in 3D: Oktree und Tetraederbisektion.

- *form–regulär*: die Größen aller Innenwinkel sind nach unten beschränkt

Diese beiden Eigenschaften kommen aus Fehlerabschätzungen in der Finite Elemente–Theorie [14]. Hierbei wird angenommen, daß nicht ein gegebener Datensatz hierarchisch dargestellt wird, sondern eine unbekannte Funktion (die Lösung einer partiellen Differentialgleichung) auf dem Grundgebiet durch lokale Verfeinerung immer genauer approximiert wird. Anzumerken ist, daß semi–reguläre hierarchische Zerlegungen automatisch auch form–regulär sind. Quasi–uniformität ist i.d.R. auch dann erfüllt, wenn die Zahl der Level l nicht zu groß wird.

2.1.4 Konstruktion hierarchischer Zerlegungen

Verfahren zur Konstruktion hierarchischer Zerlegungen sind ein Forschungsgebiet, welches vor allem durch die Erfindung von Mehrgitterverfahren [72] in den siebziger Jahren angestoßen wurde. Weitere wichtige Anwendungsbereiche sind räumliche Datenbanken und die rechnergestützte geometrische Modellierung [33]. Hierarchische Zerlegungen zur Visualisierung großer Datenmengen sind hingegen ein relativ neues Forschungsgebiet.

Prinzipiell gibt es zur Konstruktion hierarchischer Zerlegungen zwei Ansätze:

- *top–down* Methoden und
- *bottom–up* Methoden

Top–down Methoden starten von einer groben Zerlegung des Grundgebiets und unterteilen dann (i.d.R. nebeneinander liegende) Teilgebiete sukzessive. Bottom–up Techniken starten von einer fein aufgelösten Zerlegung des Grundgebiets und fassen Teilgebiete sukzessive zusammen.

In Pseudocode läßt sich die top–down Konstruktion wie folgt schreiben:

T sei grobe Zerlegung des Grundgebiets
während (Zerlegung T nicht fein genug)
    wähle $\{S_1, S_2, \ldots, S_n\} \in T$
    wähle $\{S'_1, S'_2, \ldots, S'_m\}$ sodaß $\cup_{i=1}^{m} S'_i = \cup_{j=1}^{n} S_j$ und $m > n$
    setze $T \rightarrow (T \setminus \{S_1, S_2, \ldots, S_n\}) \cup \{S'_1, S'_2, \ldots, S'_m\}$

Die bottom–up Konstruktion sieht entsprechend so aus:

T sei feine Zerlegung des Grundgebiets
während (Zerlegung T nicht grob genug)
    wähle $\{S_1, S_2, \ldots, S_m\} \in T$
    wähle $\{S'_1, S'_2, \ldots, S'_n\}$ sodaß $\cup_{i=1}^{n} S'_i = \cup_{j=1}^{m} S_j$ und $n < m$
    setze $T \rightarrow (T \setminus \{S_1, S_2, \ldots, S_m\}) \cup \{S'_1, S'_2, \ldots, S'_n\}$

Offensichtlich sind die top–down und bottom–up Konstruktionen zueinander invers, d.h. zu jeder top–down Konstruktion gibt es eine entsprechende bottom–up Konstruktion und umgekehrt. Die inverse Operation zu einem Unterteilungsschritt nennt man Vereinfachungsschritt.

Gebiets–vereinfachende hierarchische Zerlegungen werden dadurch realisiert, daß die Bedingung $\cup_{i=1}^{n} S'_i = \cup_{j=1}^{m} S_j$ aufgeweicht wird und nur im Inneren des Grundgebiets gelten muß, d.h. $(\cup_{i=1}^{n} S'_i) \cap (\cup_{j=1}^{m} S_j)$ liegt am Rand des Grundgebiets. Falls diese Bereiche vollständig außerhalb des Grundgebiets liegen, spricht man von *äußerer Überdeckung*, wenn sie vollständig innerhalb liegen, von *innerer Überdeckung* des Grundgebiets.

Im Prinzip könnte man einer hierarchischen Zerlegung nach jedem Unterteilungsschritt einen neuen Level zuordnen. Oft ist es allerdings notwendig, daß nicht alle Teilgebiete verfeinert werden, sondern nur bestimmte. In diesem Fall spricht man von *adaptiver Verfeinerung*. Üblicherweise ordnet man daher einer hierarchischen Zerlegung erst nach der maximalen Menge von unabhängigen Unterteilungsschritten einen neuen Level zu. Zwei Unterteilungsschritte heißen dabei voneinander *unabhängig*, falls kein Teilgebiet des einen Unterteilungsschritts durch Unterteilung des anderen Unterteilungsschrittes entstanden ist.

Die verschiedenen Ansätze zur Konstruktion hierarchischer Zerlegungen unterscheiden sich stark, je nachdem ob geschachtelte oder nicht geschachtelte Zerlegungen benötigt werden. Wir wollen im folgenden zunächst den geschachtelten und dann den nicht–geschachtelten Fall betrachten.

Geschachtelte Zerlegungen

Geschachtelte Zerlegungen bilden einen in vielfacher Weise besonderen Fall. Zum Beispiel ist hier die Unabhängkeit der Unterteilungsschritte leicht entscheidbar und solche Zerlegungen lassen sich top–down durch sehr einfache kaskadenartige rekursive Algorithmen dieser Art realisieren:

zerlege(Gebiet S, Level l)
    falls ($l < l_{max}$)
        wähle { S'_1, S'_2, ... S'_m } sodaß $\cup_{i=1}^{m} S'_i = S$
        zerlege(S'_1, l+1);
        zerlege(S'_2, l+1);
        ...
        zerlege(S'_m, l+1);

Außerdem müssen die Zerlegungen nicht explizit a priori konstruiert werden, sondern die Teilgebiete können auch erst bei Bedarf gebildet werden. Im obigen Beispiel sind solche hierarchische Zerlegungen vollständig und regelmäßig. Adaptive Zerlegungen werden dadurch realisieret, daß nicht alle Gebiete verfeinert werden, d.h. die Rekursion wird je nach Gebiet S schon vor l_{max} abgebrochen. Nicht regelmäßige Zerlegungen können mit Hilfe mehrerer **zerlege**–Funktionen definiert werden.

Beispiele für Unterteilungs- oder Vereinfachungsschritte sind in Abb. 2.7 und 2.8 dargestellt. Bei weitem am genausten untersucht sind reguläre Unterteilungen einfacher Polygone. Zum Beispiel kann ein Dreieck in zwei oder vier ähnliche Dreiecke [4] unterteilt werden. Ein Tetraeder kann zwar nicht ausschliesslich in zueinander ähnliche Tetraeder unterteilt werden, aber man kann ihn in z.B. in acht Tetraeder aus zwei Ähnlichkeitsklassen unterteilen [12]. Sehr viel leichter gestalten sich Unterteilungen von Vierecken bzw. Quadern, allerdings sind die durch Verfeinerung entstehenden Vierecke bzw. Quader außer in achsenparallelen Fällen nicht mehr ähnlich zueinander. Analoge Unterteilungen gibt es auch für Pyramiden und Prismen, die oft bei der Verfeinerung von unstrukturierten Tetraedergittern auftreten können.

Nicht geschachtelte Zerlegungen

Nicht geschachtelte Zerlegungen werden vor allem zur Konstruktion von Zerlegungen auf unstrukturierten Punktmengen, wie z.B. Meßpunkten, benötigt. Im diesem Bereich gibt es drei Stoßrichtungen: Template–basierte Verfeinerung, hierarchische Delaunay Triangulierungen und zwischen den Leveln völlig unkorrellierte Konstruktionen. All diese Ansätze werden vor allem bei irregulären Triangulierungen eingesetzt.

Beim Template–basierten Ansatz kann zum Beispiel in einem bottom–up Vereinfachungsschritt in einer gegebenen Triangulierung ein Punkt, eine Kante oder ein Dreieck entfernt werden oder eine Kante durch einen Punkt, ein Dreieck durch einen Punkt oder ein Dreieck durch eine Kante ersetzt werden (Abb. 2.9, siehe z.B. [73, 79, 121]. Dabei werden neben dem entfernten Element auch alle von ihm ausgehenden Kanten entfernt. Das so entstehende Loch wird dann mit Hilfe von weniger Dreiecken wieder zerlegt. Diese Zerlegung kann dabei unabhängig oder abhängig (optimiert) von der Topologie des Loches konstruiert werden.

Die wichtigsten Templates sind dabei die Stern- und die Loop–Triangulierung. Bei der Stern–Triangulierung wird ein Punkt im Inneren des Loches mit allen Ecken des Loches verbunden (siehe die unteren drei Fälle in Abb. 2.9). Im Gegensatz dazu verwendet die Loop–Triangulierung keine zusätzlichen Punkte im Inneren und verbindet die Eckpunkte des Loches alternierend miteinander (siehe die oberen drei Fälle in Abb. 2.9).

Bei optimierten Verfahren wird kein festes Muster zur Zerlegung des Loches verwendet, sondern es wird versucht z.B. die entstehenden Winkel zu maximieren. Dies resultiert in qualitativ bessere Zerlegungen allerdings auf Kosten eines erhöhten Kodierungs- und Speicherbedarfs. Für einen Vergleich der Verfahren siehe [22, 76, 112].

In drei Dimensionen wurden die entsprechenden Konzepte auf Tetraeder in [129, 137] verallgemeinert. Im Prinzip können in ähnlicher Weise auch irreguläre Viereckgitter oder allgemeine Polyedergitter vereinfacht werden. Hierbei gibt es allerdings weitaus weniger Möglichkeiten an Auswahl- und Zerlegungsschritten. Zum Beispiel kann ein Viereckgitter durch Entfernung eines Punktes nicht notwendigerweise wieder in Vierecke zerlegt werden.

Eine weitere wichtige Klasse von nicht geschachtelten hierarchischen Zerlegun-

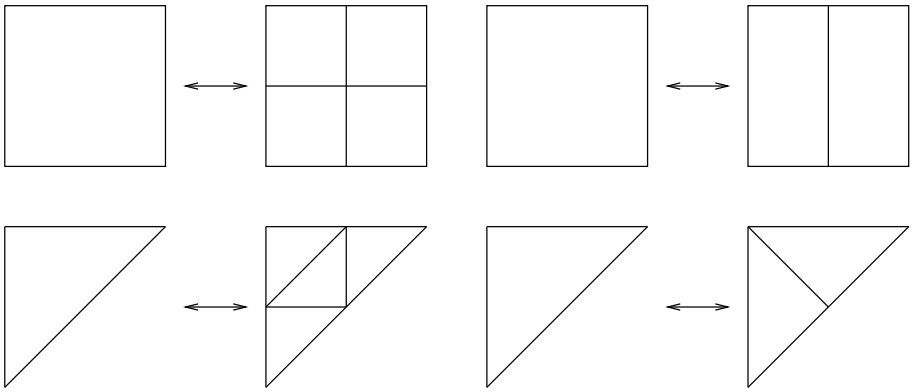


Abbildung 2.7: Geschachtelte Unterteilung/Vereinfachung in 2D: einfache und simultane Koordinatenbisektion von Vierecken, rote Dreiecksverfeinerung und Dreiecksbisektion.

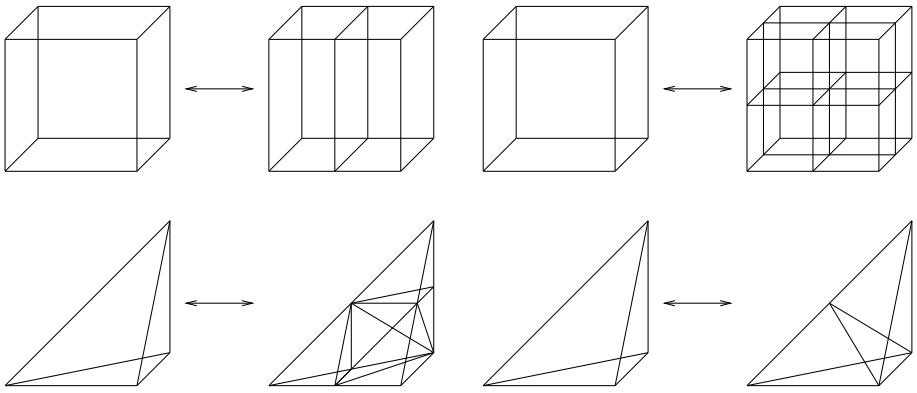


Abbildung 2.8: Geschachtelte Unterteilung/Vereinfachung in 3D: einfache und simultane Koordinatenbisektion von Quadern, rote Tetraederverfeinerung und Tetraederbisektion.

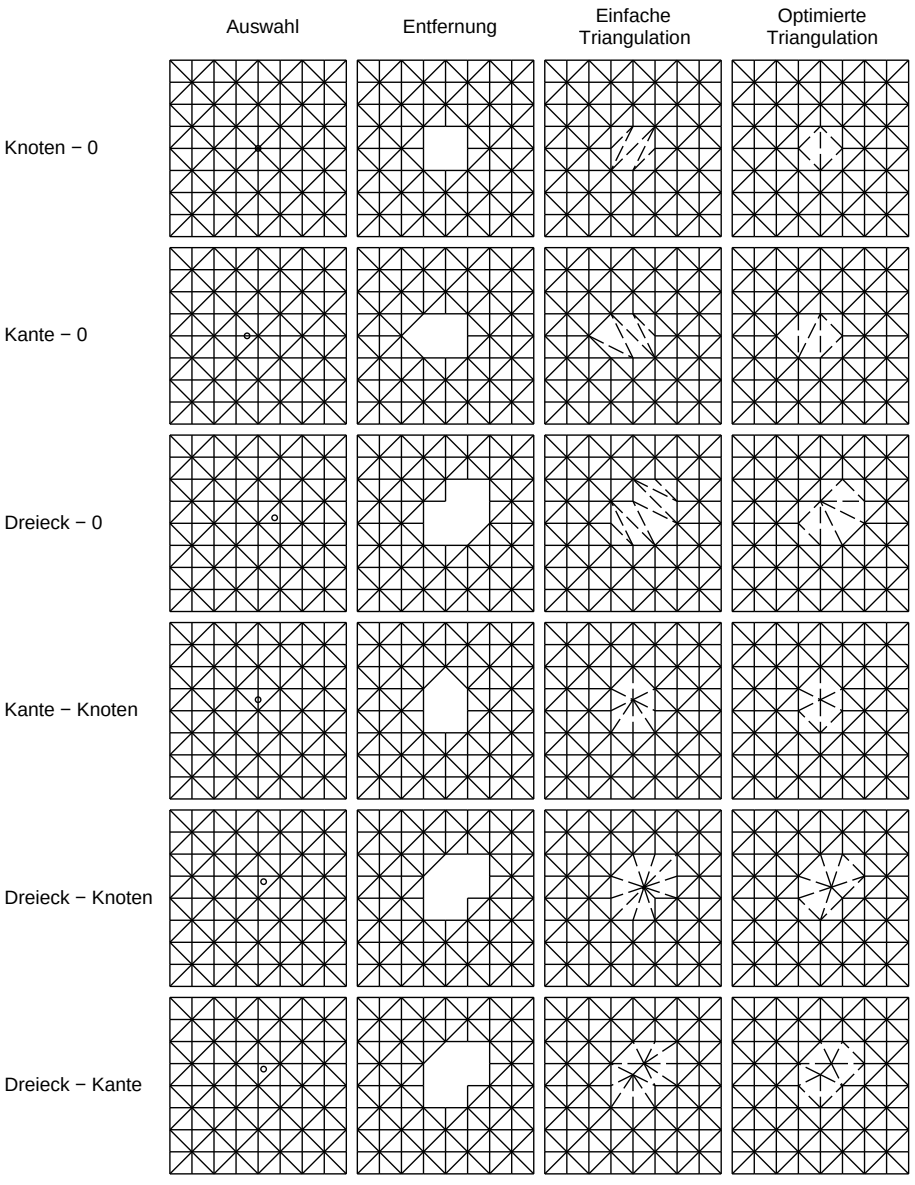


Abbildung 2.9: Verschiedene Möglichkeiten zur nicht geschachtelten Vereinfachung (bzw. Verfeinerung) eines Dreiecksgitters.

gen sind hierarchische Delaunay–Triangulierungen [20, 21, 29, 116, 126]. Bei diesem Ansatz werden ausgehend von einer Delaunay–Triangulierung sukzessive Punkte so eingefügt, daß wiederum Delaunay–Triangulierungen entstehen. Dafür müssen jeweils lokale Retriangulierungen vorgenommen werden. Abhängigkeiten zwischen den verschiedenen Leveln können durch einen azyklischen gerichteten Graphen dargestellt werden. Delaunay–artige Triangulierungen sind besonders dann wichtig, wenn aufgrund der Anwendung der kleinste Winkel aller Dreiecke maximiert werden soll.

Bei der dritten Klasse der unkorrelierten hierarchischen Zerlegungen werden ausgehend von einer fein aufgelösten Triangulierung sukzessive Teilmengen der Punkte ausgewählt und unabhängig von den Ausgangstriangulierungen neu trianguliert [78, 138]. Auf diese Weise sind Korrelationen zwischen den Dreiecken auf unterschiedlichen Leveln nur implizit und nicht wie in den vorherigen Klassen explizit definiert.

Natürlich ist es auch möglich, Mischformen aus geschachtelten und nicht geschachtelten oder strukturierten und unstrukturierten Zerlegungen zu verwenden (sog. hybride Modelle). Auf diese Weise können z.B. in einem Geländemodell irreguläre Zerlegungen zur Approximation von Flüssen oder Kammlinien und reguläre Zerlegungen für die restliche Landschaft verwendet werden [11]. Alternativ können auch lokal angepaßte Unterteilungsschritte, z.B. um wichtige Geländefeatures zu erhalten [120], verwendet werden.

2.2 Uniforme Simplexbisektion

Wie im letzten Abschnitt zu sehen war, gibt es eine große Vielfalt hierarchischer Zerlegungen. Die Simplexbisektion ist eines dieser Verfahren mit sehr vielen angenehmen Eigenschaften. Zum einen ist das Verfahren prinzipiell in beliebigen Dimensionen anwendbar. Zum zweiten wächst die Zahl der Teilgebiete nur langsam, denn sie verdoppelt sich nur mit jeder Unterteilung. Damit ist es z.B. leichter möglich, effiziente adaptive Zerlegungen zu konstruieren. Im Gegensatz dazu vervierfacht sich die Zahl der Gebiete bei Quadtrees und verachtfacht sich bei Octrees. Zum dritten ist der Simplex das einfachste geometrische Grundprimitiv und gut geeignet um komplexe Gebietsränder und Funktionen zu approximieren. Weitere angenehme Eigenschaften des Simplexbisektionsverfahren bei der Kompression, Speicherung und graphischen Darstellung von Daten werden in späteren Kapiteln aufgezeigt.

Der Einfachheit halber wollen wir im folgenden als Grundgebiet den Einheits–Hyperwürfel $[0,1]^d$ wählen und regelmäßige dyadische Gitter betrachten. Verallgemeinerung werden dann in Abschnitt 2.3 betrachtet. Bevor wir den dimensionsunabhängigen Zerlegungsalgorithmus angeben, werden wir zunächst die in der Praxis wichtigsten Dimensionen 1–3 genauer beleuchten.

2.2.1 Intervallbisektion

Wir wollen mit dem eindimensionalen Fall beginnen und hierbei die im Rest der Arbeit verwendete Level–Index Notation einführen. Grundgebiet Ω sei das

Intervall $S_0^0 := [0,1]$. Einem Intervall S_i^l ist dabei ein Verfeinerungslevel l zugeordnet, wobei innerhalb eines jeden Levels die Intervalle von $i = 0, 1, ...$ durchnumeriert sind.

Ein Bisektionsschritt erzeugt nun rekursiv aus einem Intervall S_i^l zwei neue Intervalle S_{2i}^{l+1} und S_{2i+1}^{l+1} der halben Größe,

$$S_i^l \longrightarrow \{S_{2i}^{l+1}, S_{2i+1}^{l+1}\}$$

mit

$$S_i^l = (a,b), S_{2i}^{l+1} := (a,(a+b)/2) \text{ und } S_{2i+1}^{l+1} := ((a+b)/2,b).$$

Hierbei soll es zunächst keine Rolle spielen, ob die Intervalle offen oder geschlossen sind, der Einfachheit halber wollen wir alle Intervalle offen schreiben. Natürlich gilt $S_i^l = (i \cdot 2^{-l}, (i+1) \cdot 2^{-l})$, wir wollen aber in Hinblick auf komplexere (irreguläre) Zerlegungen hier und in Zukunft die Zerlegungen hierarchisch (in diesem Fall mit a und b) parametrisieren. Dies spiegelt auch die rekursive Implementierung in einem Computerprogramm wieder:

```
zerlege(Koord a, b; Level l, Nummer n)
        falls (l < l_max)
                Koord new=(a+b)/2;
                zerlege(a, new, l+1, 2*n);
                zerlege(new, b, l+1, 2*n+1);
```

Die Menge aller Intervalle S_i^l auf einem Level l bilden dabei Zerlegungen

$$T^l := \{S_0^l, S_1^l, \ldots\}$$

des Einheitsintervalls, d.h. für alle Level l gilt:

$$\Omega = \bigcup_{S_i^l \in T^l} S_i^l.$$

Für die Intervallbisektion gilt damit $|S_i^l| = 2^{-l}$ und $|T^l| = 2^l$.

Auf diese Weise wird auf den Intervallen eine Binärbaum–Struktur induziert, wobei jedem Intervall ein Knoten im Baum entspricht. Die Söhne des Knotens S_i^l sind dabei S_{2i}^{l+1} und S_{2i+1}^{l+1}, während der Vater von S_i^l das Intervall $S_{\lfloor i/2 \rfloor}^{l-1}$ ist.

Alternativ kann man dem gröbsten Intervall anstatt der Nummer 0 auch die Nummer 1 geben. Ohne den Algorithmus zu ändern erhält dadurch jedes Intervall unabhängig von seinem Level eine eigene Nummer i. Zum Beispiel bekommen so die beiden Intervalle von Level 1 die Nummern 2 und 3, die von Level 2 die Nummern 4 bis 7 und so weiter. Solch eine Numerierung wird praktisch bei der Speicherung adaptiver Zerlegungen (Abschnitt 5.2.4) oder bei der Parallelisierung (Abschnitt 6.4.5) sein.

Prinzipiell ist es auch möglich, statt des Mittelpunktes eines jeden Intervalls $(a+b)/2$ einen anderen Punkt zur Unterteilung zu verwenden, z.B. $(2a+b)/3$ oder $(3a+b)/4$. Im Sinne der Definition wären dies immer noch uniforme Zerlegungen. Die symmetrische Intervallbisektion ist aber die mit großem Abstand wichtigste Unterteilungsvorschrift.

2.2.2 Dreiecksbisektion

Nun wollen wir das Einheitsquadrat $\Omega = [0,1]^2$ betrachten. Die vielleicht gebräuchlichste Art, die Intervallbisektion auf zwei Dimensionen zu verallgemeinern ist durch simultane Koordinatenbisektion. Dabei wird ein Quadrat rekursiv in vier Quadrate halber Seitenlänge unterteilt. Die auf diese Weise entstehende hierarchische Zerlegung wird auch Quadtree Zerlegung [118] genannt, nachdem in der entsprechenden Baumdarstellung jeder Knoten vier statt zwei Söhne hat.

Alternativ kann man eine hierarchische Zerlegung mit Hilfe von Dreiecken realisieren (siehe auch [102, 114, 143]. Hierbei wird das Grundgebiet zunächst entlang einer der beiden Diagonalen in zwei gleischenklig rechtwinklige Dreiecke S_0^0 und S_1^0 zerlegt. Jedes Dreieck wird nun rekursiv durch Halbierung des rechten Winkels in zwei ähnliche Dreiecke der Größe $1/\sqrt{2}$ unterteilt. Wir wollen ein Dreieck S_i^l durch die Angabe der Eckpunkte (a,b,c) definieren und dabei festlegen, daß der rechte Winkel am Punkt b liegt. Die Ausgangsdreiecke sind damit z.B.

$$S_0^0 := ((0,0)^T,(0,1)^T,(1,1)^T) \text{ und } S_1^0 := ((1,1)^T,(1,0)^T,(0,0)^T).$$

So kann die Unterteilungsregel durch

$$S_i^l = (a,b,c), S_{2i}^{l+1} := (b,(a+b)/2,a) \text{ und } S_{2i+1}^{l+1} := (c,(a+b)/2,b).$$

definiert werden. In Pseudocode sieht diese Konstruktion etwa so aus:

```
zerlege(Koord a, b, c; Level l, Nummer n)
        falls (l < l_max)
                Koord new=(a+c)/2;
                zerlege(b, new, a, l+1, 2*n);
                zerlege(c, new, b, l+1, 2*n+1);
```

Die Menge aller 2^{l+1} Dreiecke auf einem Level l bilden dabei Triangulierungen $T^l := \{S_0^l, S_1^l, \ldots\}$ des Einheitsquadrates. Die beiden Katheten eines Dreiecks von Level l haben die Länge $2^{-l/2}$, die Hypotenuse hat die Länge $2^{-(l-1)/2}$.

Der durch Unterteilung eines Dreiecks entstehende neue Punkt der Triangulierung wird *Verfeinerungsknoten* genannt und liegt in der Mitte der Hypotenuse des Dreiecks. Wenn alle Dreiecke eines Levels verfeinert werden, sind die auf diese Weise entstehenden Triangulierungen zulässig weil jeder Verfeinerungsknoten im Inneren des Grundgebiets für je zwei benachbarte Dreiecken identisch ist. Verfeinerungsknoten am Rand des Gebietes besitzen nur ein zugehöriges Dreieck.

Die Menge der Verfeinerungsknoten eines Levels bilden äquidistante Gitter, entweder rechtwinklige (für ungerade Level) oder um 45 Grad gedrehte (für gerade Level). Letztere werden auch Quincunx[5] Gitter genannt weshalb auch die ge-

[5]nach dem Augenmuster der 5 auf einem sechsseitigen Würfel. Berühmt ist auch ein Nagelbrett gleichen Namens und Musters das von Francis Galton (1822-1911) zur Demonstration der Überlagerung von Normalverteilungen gebaut wurde

samte hierarchische Zerlegung manchmal auch Quincunx Hierarchie genannt wird.

Anstatt des Mittelpunkts der Verfeinerungskante kann auch ein anderer Punkt auf der Hypotenuse als Verfeinerungsknoten bestimmt werden. Damit wäre die Verfeinerung zwar immer noch geschachtelt aber nicht mehr regulär. Weitere irreguläre Zerlegungen werden genauer in Abschnitt 2.3 behandelt.

2.2.3 Tetraederbisektion

Betrachten wir nun den Einheitswürfel $\Omega = [0,1]^3$. Analog zur Dreiecksbisektion wollen wir zunächst das Grundgebiet in Tetraeder zerlegen. Hierfür gibt es verschiedene Möglichkeiten. Zum Beispiel kann man den Einheitswürfel durch Abschneiden von vier Ecken in fünf Tetraeder zerlegen. Eine weitere Möglichkeit ist, den Einheitswürfel entlang einer der vier Würfeldiagonalen in sechs kongruente Tetraeder zerlegen. Die Ausgangstetraeder wären somit:

$$\begin{aligned}
S_0^0 &:= ((0,0,0)^T,(0,0,1)^T,(0,1,1)^T,(1,1,1)^T),\\
S_1^0 &:= ((0,0,0)^T,(0,0,1)^T,(1,0,1)^T,(1,1,1)^T),\\
S_2^0 &:= ((0,0,0)^T,(0,1,0)^T,(0,1,1)^T,(1,1,1)^T),\\
S_3^0 &:= ((0,0,0)^T,(0,1,0)^T,(1,1,0)^T,(1,1,1)^T),\\
S_4^0 &:= ((0,0,0)^T,(1,0,0)^T,(1,0,1)^T,(1,1,1)^T),\\
S_5^0 &:= ((0,0,0)^T,(1,0,0)^T,(1,1,0)^T,(1,1,1)^T).
\end{aligned}$$

Diese Unterteilung ist natürlicher im Hinblick auf reguläre Gitterstrukturen und die Dimensionsunabhängigkeit des Verfahrens. Jeder Tetraeder wird nun entlang der Verbindungsebene zwischen dem Mittelpunkt der längsten Kante und den beiden gegenüberliegenden Ecken in zwei Teile unterteilt (siehe auch [2, 3, 95, 99, 115]). Die rekursive Definition der hierarchischen Zerlegung gestaltet sich allerdings geringfügig schwieriger als in den vorhergehenden Fällen und erfordert eine Fallunterscheidung:

```
zerlege(Koord a, b, c, d; Level l, Nummer n)
        falls (l < l_max)
                Koord new=(a+b)/2;
                falls (l mod 3 = 0)
                        zerlege(a, c, d, new, l+1, 2*n);
                        zerlege(c, d, b, new, l+1, 2*n+1);
                sonst
                        zerlege(a, c, d, new, l+1, 2*n);
                        zerlege(d, c, b, new, l+1, 2*n+1);
```

Auf diese Weise werden zunächst der Mittelpunkt des Würfels, dann die Mittelpunkte der sechs Seiten des Würfels und schließlich die Mittelpunkte der zwölf

Kanten des Würfels eingefügt. Die Verfeinerungsknoten liegen dabei immer auf der längsten Kante eines Tetraeders. Im Inneren des Würfels wird jeder Verfeinerungsknoten je nach Level von vier, sechs oder acht Tetraedern geteilt. Dadurch entstehen wiederum äquidistante Gitter, die entweder parallel zu den Koordinatenachsen oder um 45 Grad in den verschiedenen Richtungen gedreht sind.

Durch die Verfeinerung treten auch drei verschiedene Arten von Tetraedern auf. Innerhalb eines Level sind die Tetraeder kongruent und alle drei Level wiederholen sich die Tetraedertypen zyklisch. Die angegebene Tetraederverfeinerung durch Bisektion ist damit semi–regulär.

2.2.4 Allgemeine Simplexbisektion

Nun wollen wir die in den vorigen Abschnitten betrachteten hierarchischen Zerlegungen auf beliebig dimensionale Hyperwürfel $[0,1]^d$ verallgemeinern. Die Konstruktion geht auf Maubach [99] zurück. Eine weitere Möglichkeit zur Konstruktion hierarchischer Simplizialgitter basierend auf Bisektion ist von Traxler [136] beschrieben.

Im folgenden bezeichnen fettgedruckte Zeichen Vektoren oder Multiindizes. Insbesondere seien $\mathbf{0}$ der Nullvektor $(0,\ldots,0)^T$, $\mathbf{1}$ der Einsvektor $(1,\ldots,1)^T$ und $\mathbf{e}_i$ der i–te Einheitsvektor. Ferner sei definiert $|\mathbf{i}| := i_1 + \ldots + i_d$.

Ein d–Simplex $S = (\mathbf{x}_0,\ldots,\mathbf{x}_d)$ ist definiert als die konvexe Hülle der $d+1$ linear unabhängigen Punkte $\mathbf{x}_0,\ldots,\mathbf{x}_d$. Insbesondere ist ein 0–Simplex ist ein Punkt, ein 1–Simplex ein Intervall, ein 2–Simplex ein Dreieck und ein 3–Simplex ein Tetraeder. Hierbei sollen die Punkte keiner Ordnung unterstehen, d.h. die Simplizes $(\ldots,\mathbf{x}_i,\ldots,\mathbf{x}_j,\ldots)$ und $(\ldots,\mathbf{x}_j,\ldots,\mathbf{x}_i,\ldots)$ seien identisch.

Zunächst wollen wir den Hyperwürfel Ω in Simplizes zerlegen. Sei $\mathbf{i} = (i_1,\ldots,i_d)$ eine Permutation von $(1,\ldots,d)$. Ein Kuhn–Simplex $S^0_{n(\mathbf{i})} = (\mathbf{x}_0,\ldots,\mathbf{x}_d)$ ist rekursiv definiert durch $\mathbf{x}_0 = \mathbf{0}$ und $\mathbf{x}_j = \mathbf{x}_{j-1} + \mathbf{e}_{i_j}$. Hierbei sei $n(\mathbf{i})$ irgendeine bijektive Abbildung von $\{\mathbf{i}\}$ auf $\{0,\ldots,d!-1\}$. Die Vereinigung der genau $d!$ verschiedenen Kuhn–Simplizes bildet die Kuhn–Triangulierung $T^0 := \{S_0,\ldots,S_{d!-1}\}$ von Ω.

Sei $e = (\mathbf{x}_i,\mathbf{x}_j)$ die längste Kante von $S^l_i = (\mathbf{x}_0,\ldots,\mathbf{x}_d)$ und $\mathbf{x}_n = (\mathbf{x}_i+\mathbf{x}_j)/2$ deren Mittelpunkt. Eine Bisektionsunterteilung von S^l_i erzeugt dann zwei neue Simplizes

$$S^{l+1}_{2i} := (\mathbf{x}_0\ldots\mathbf{x}_{i-1}\mathbf{x}_n\mathbf{x}_{i+1}\ldots\mathbf{x}_d) \text{ und } S^{l+1}_{2i+1} := (\mathbf{x}_0\ldots\mathbf{x}_{j-1}\mathbf{x}_n\mathbf{x}_{j+1}\ldots\mathbf{x}_d).$$

Vollständige hierarchische Bisektionszerlegungen der Kuhn–Triangulierung T^0 sind immer zulässig nachdem die längsten Kanten zweier benachbarter Simplizes immer eindeutig sind und geteilt werden. Es gilt ausserdem $|T^l| = d!\cdot 2^l$.

Durch die Verfeinerung treten d verschiedene Typen von Simplizes auf, die sich alle d Level zyklisch wiederholen. Der zweidimensionale Fall bildet allerdings hier die einzige Ausnahme, da nur ein Typ Dreieck auftritt (siehe Abschnitt 2.2.2).

In analoger Weise zu den vorher besprochenen niederdimensionalen Fällen kann

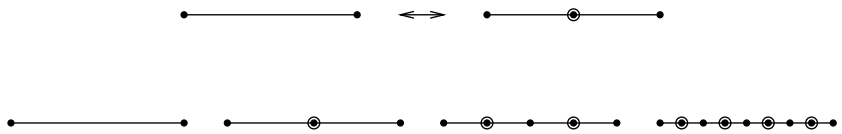


Abbildung 2.10: Intervallbisektion: Verfeinerungsregel und hierarchische Zerlegung eines Intervalles.

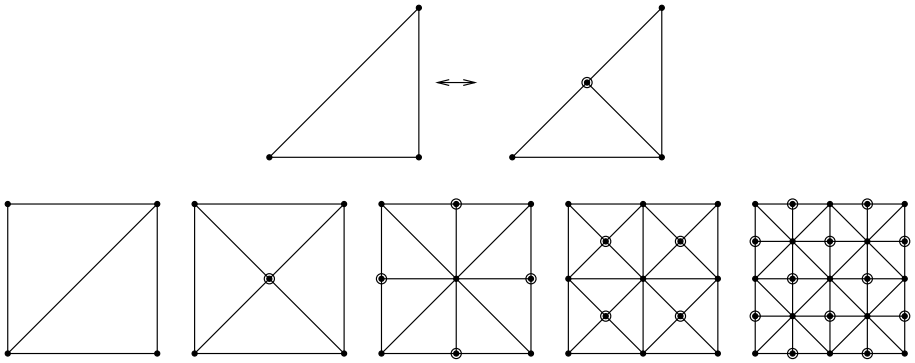


Abbildung 2.11: Dreiecksbisektion: Verfeinerungsregel und hierarchische Zerlegung eines Quadrates.

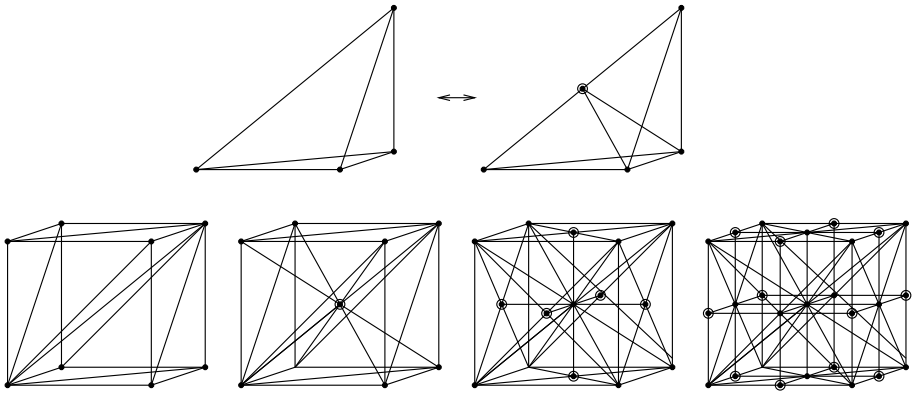


Abbildung 2.12: Tetraederbisektion: Verfeinerungsregel und hierarchische Zerlegung eines Würfels.

eine eindeutige Indizierung der Simplizes dadurch erreicht werden, daß die Kuhn–Simplizes statt von 0 bis $d! - 1$ von $d!$ bis $2 \cdot d! - 1$ numeriert werden.

2.3 Verallgemeinerungen

Im diesem Abschnitt wollen wir das uniforme Bisektionsverfahren auf komplexere Grundgebiete, Grundtriangulierungen und Gitterstrukturen verallgemeinern. Hierbei wollen wir die schönen Eigenschaften, wie Einfachheit der rekursiven Implementierung oder langsames Anwachsen der Teilgebiete mit dem Level, des uniformen Verfahrens so weit wie möglich beibehalten. Dabei wollen wir vor allem vollständige Zerlegungen betrachten, nachdem adaptive Zerlegungen eingehender in Kapitel 4 behandelt werden. Weiterhin werden wir versuchen, die Konstruktionen dimensionsunabhängig anzugeben.

2.3.1 Komplexere Grundgebiete

Als erstes wollen wir davon abkommen, daß das Grundgebiet der Einheitshyperwürfel ist. Dadurch wird es oft notwendig, gebietsvereinfachende Verfahren zu verwenden.

Transformation

Zunächst wollen wir jedoch einfache gebietserhaltende Verfahren betrachten. In manchen Fällen ist es möglich, das Grundgebiet durch eine (oft affine) Transformation auf ein einfacheres Gebiet abzubilden. Eine hierarchische Zerlegung des einfachen Gebiets wird dann durch die entsprechende inverse Transformation auf das komplexere Grundgebiet abgebildet. Speziell bei periodisierenden Transformationen, wie z.B. bei der Abbildung auf eine Kugeloberfläche, muß allerdings darauf geachtet werden, daß die Verfeinerungsknoten an den entsprechenden Seiten übereinstimmen.

Blockweise Transformation

Falls die Konstruktion einer hierarchischen Zerlegung durch eine einzelne Transformation nicht gelingt oder nicht zufriedenstellend ist, wird oft das Grundgebiet in einfacher zu transformierende Blöcke zerlegt und diese Blöcke dann separat voneinander transformiert. Hierbei ist natürlich (insbesondere in drei oder höheren Dimensionen) darauf zu achten, daß die Zerlegungen an den Blockgrenzen übereinstimmen. Bei Bisektionstriangulierungen und würfelförmigen Blöcken wird dies z.B. dadurch erreicht, daß die Diagonalen der Würfel (d.h. die ersten Verfeinerungskanten) alternierend gewählt werden.

Überdeckungen

Eine gebietsvereinfachende Zerlegung kann durch eine äußere Überdeckung des Grundgebiets mit einem einfachen Gebiet (z.B. einem Quadrat oder Würfel)

geschehen. Während der Verfeinerung werden dann Teilgebiete, die vollständig außerhalb des Grundgebiets liegen, eliminiert. Somit wird neben dem Grundgebiet selbst auch der Rand des Grundgebiets sukzessive immer genauer (polygonal) approximiert.

Eine entsprechende innere Überdeckung läßt sich im Prinzip in analoger Weise realisieren, indem nach einer Verfeinerung zusätzliche Teilgebiete zwischen dem Gebietsrand und der inneren Approximation eingesetzt werden. Solche Verfahren sind jedoch nicht mehr rekursiv implementierbar.

In beiden Fällen kann eine bessere Randapproximation dadurch realisiert werden, daß Verfeinerungsknoten auf dem Rand der Zerlegung nicht auf die Mitte der Verfeinerungskante sondern auf den Grundgebietsrand gelegt werden.

2.3.2 Komplexere Grundtriangulierungen

In manchen Fällen ist neben einem Grundgebiet eine grobe Grundtriangulierung vorgegeben, welche hierarchisch verfeinert werden soll. Diese Fragestellung ist oft bei der Diskretisierung partieller Differentialgleichungen gegeben, wobei die Grundtriangulierungen z.B. von einem Gittergenerator stammen. Das prinzipielle Problem bei der Konstruktion hierarchischer Bisektionstriangulierungen ist hier, daß die Verfeinerungskanten benachbarter Simplizes übereinstimmen, da ansonsten die entstehenden Triangulierungen nicht mehr zulässig sind.

Das Verfahren, jeweils die längste Kante (*longest edge bisection*) als Verfeinerungskante zu wählen [114] schlägt in der Regel fehl, da die längsten Kanten zweier benachbarter Dreiecke nicht notwendigerweise übereinstimmen. Als Ausweg können die Verfeinerungskanten vorgegeben werden [2, 95] oder strukturell z.B. basierend auf der Numerierung der Eckpunkte der Simplizes [99] ausgewählt werden.

In zwei Dimensionen läßt sich z.B. jede Triangulierung bestehend aus k Dreiecken in $\lfloor k/2 \rfloor$ Vierecke plus evtl. ein Randdreieck zerlegen. Die ersten Verfeinerungskanten sind dann die Diagonalen dieser Vierecke, bzw. die Randkante des Randdreiecks. Danach werden die Dreiecke nach der *split newest vertex* Strategie verfeinert, d.h. die Verfeinerungskante liegt immer gegenüber dem zuletzt erzeugten Knoten eines Dreiecks. Auf diese Weise entstehen immer zulässige Triangulierungen. Im uniformen Fall sind übrigens die *split newest vertex* und die *longest edge bisection* Strategie identisch.

In höheren Dimensionen gestalten sich entsprechende Konstruktionen allerdings schwieriger und es werden vor allem verschiedene Absättigungsstrategien, wie in Kapitel 4 besprochen, verwendet.

2.3.3 Komplexere Gitterstrukturen

Im Gegensatz zur vorhergehenden Problematik kann auch eine feine Triangulierung vorgegeben sein, welche hierarchisch vergröbert werden soll. Dies ist oft bei triangulierten Meßdaten, wie sie z.B. von einem Laserscanner kommen, der Fall. Das prizipielle Problem hierbei ist, daß die Vereinigung zweier Simplizes i.d.R. keinen größeren Simplex ergibt.

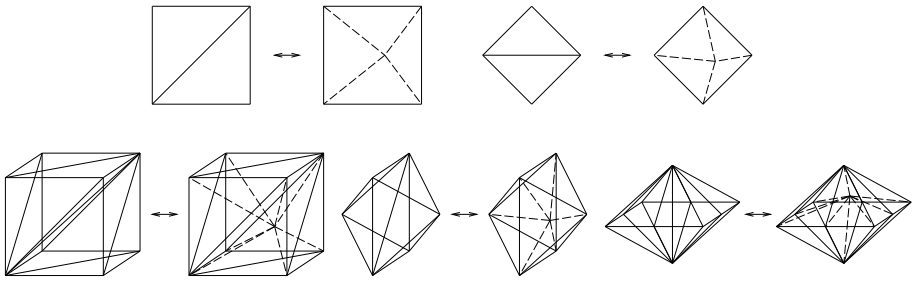


Abbildung 2.13: Nicht geschachtelte Bisektion in 2D und in 3D.

Daher muß hier auf eine nicht geschachtelte Variante des Verfahrens zurückgegriffen werden [142]. In ähnlicher Weise zur Randapproximation aus Abschnitt 2.3.1 müssen hierbei Verfeinerungsknoten nicht auf der Verfeinerungskante zu liegen kommen, sondern können auch neben ihr liegen.

Die top–down Konstruktion entfernt eine gemeinsame Kante von n Simplizes und trianguliert das so entstehende Loch durch $2n$ neue radiale Kanten bezüglich eines ausgewählten Verfeinerungsknotens (siehe Abb. 2.13). Die jeweils nächste Verfeinerungskante kann nach der *split newest vertex* Strategie gewählt werden. Die Wahl des Verfeinerungsknotens kann nach verschiedenen Kriterien erfolgen. Oft wird dafür ein Fehlermaß benötigt, wir werden daher auf diese Problematik erst wieder in Kapitel 4 eingehen, nachdem entsprechende hierarchische Funktionenräume definiert worden sind.

Für eine entsprechende bottom–up Konstruktion wollen wir zunächst den zweidimensionalen Fall betrachten. Dann entfernt ein Vergröberungsschritt einen Eckpunkt der Triangulierung, der genau vier Dreiecken gemeinsam ist, und ersetzt die vier Dreiecke durch zwei größere. In der Ausgangstriangulierung müssen aber nicht notwendigerweise nur solche vierfachen Eckpunkte vorkommen. Dieses Problem kann durch eine Retriangulierung der Punktmenge gelöst werden. Dabei werden Kanten der Triangulierung solange vertauscht bis nur noch vierfache und achtfache Eckpunkte alternierend auftauchen. Solche Triangulierungen werden aus diesem Grund auch $4-8$ Triangulierungen genannt [142]. Die Retriangulierung muß nach jeder Vergröberung bis hin zur gröbsten Triangulierung wiederholt werden.

Im dreidimensionalen Fall werden alternierend Punkte, die 12, 8, bzw. 16 Tetraedern gemeinsam sind, entfernt (Abb. 2.13). Auch hier müssen entsprechende Retriangulierungen vorgenommen werden.

2.4 Vergleich der Verfahren

In diesem Kapitel haben wir gesehen, welche prinzipiellen Möglichkeiten es gibt, hierarchische Zerlegungen zu konstruieren und wir haben einige konkrete Konstruktionen basierend auf Simplexbisektion kennengelernt. Die wichtigsten Vorteile der Simplexbisektion waren dabei die Dimensionsunabhängigkeit, das langsame Anwachsen der Teilgebiete mit dem Level und die Einfachheit der

Teilgebiete, wodurch sich auch Gebietsränder gut approximieren lassen.

Für die veschiedenen Verfahren haben wir jeweils eine geschachtelte und eine nicht geschachtelte Variante vorgestellt. Dabei erweist sich die nicht geschachtelte Version als vorteilhaft, wenn die Ausgangsdaten selbst unstrukturiert sind, z.B. aus einer triangulierten Punktemenge bestehen. Bei Daten, die in einem Gitter angeordnet sind, ist es in jedoch der Regel besser, eine reguläre und geschachtelte Version zu verwenden, da diese sehr viel zu konstruieren ist und effizientere Algorithmen erlaubt. In den späteren Beispielen werden wir i.d.R. die reguläre geschachtelte Variante verwenden.

Kapitel 3

Hierarchische Funktionenräume

Die im letzten Kapitel vorgenommenen Konstruktionen hierarchischer Zerlegungen geschahen rein aufgrund geometrischer Überlegungen. Auf diese Weise wurde ein gegebenes Grundgebiet Ω zwar abhängig von den Positionen der Datenpunkte $\mathbf{x}_i$ aber unabhängig von den Datenwerten y_i hierarchisch zerlegt. Dadurch wäre es zum Beispiel möglich, bei gegebenen Höhendaten einer Insel den Grundriß der Insel hierarchisch zu zerlegen, aber ihr Höhenprofil wäre noch nicht hierarchisch darstellbar. Dies soll in diesem Kapitel durch die Definition geeigneter hierarchischer Interpolationstechniken geschehen.

Dabei werden wir ähnlich wie im letzten Kapitel vorgehen. Im ersten Abschnitt werden wir zunächst Interpolation auf nicht hierarchischen Zerlegungen betrachten. Der zweite Abschnitt behandelt dann die einfachsten hierarchischen Interpolationsarten basierend auf hierarchischen Finiten Elementen sowie deren Verallgemeinerungen durch Wavelets. Im dritten Teil werden dann einige konkrete Konstruktionen basierend auf Simplexbisektion angegeben.

3.1 Zerlegungsbasierte Interpolation

Interpolation ist die Technik bei einem gegebenen Datensatz bestehend aus Datenpunkten $\mathbf{x}_i$ und Datenwerten $y_i, 1 \leq i \leq N$, auch Werte zwischen den Datenpunkten zu ermitteln. Dabei liegt die Idee zugrunde, daß durch die Daten eine kontinuierliche Funktion $f : \mathbb{R}^d \to \mathbb{R}$ aufgespannt wird, die an jedem beliebigen Punkt ausgewertet werden kann. Die Interpolationsbedingung besagt dann, daß

$$f(\mathbf{x}_i) = y_i \text{ für alle } 1 \leq i \leq N.$$

Falls diese Bedingungen nicht exakt, sondern nur näherungsweise erfüllt werden, spricht man auch von Approximation. Klassische Approximationstechniken versuchen einen gegebenen Datensatz durch eine möglichst einfache Funktion so zu nähern, daß der Fehler (z.B. im Sinne der kleinsten Quadrate) möglichst klein wird. Wir wollen in den nächsten Abschnitten allerdings Approximationen

durch Vereinfachung des Datensatzes, d.h. durch Weglassen von Datenpunkten bei der Konstruktion der Interpolanten, realisieren.

Die verschiedenen Interpolationstechniken unterscheiden sich vor allem in der Wahl von f. Dabei liegt immer ein gewisses Modell für das Verhalten von f zugrunde. Mögliche Kriterien an f sind:

- Stetigkeit
- Glattheit (Stetigkeit der Ableitungen)
- Globale oder stückweise Definition

Globale Stetigkeit ist nicht immer von Wichtigkeit, zum Beispiel bei digitalen Bilden geht man meist davon aus, daß f stückweise konstant ist. Bei digitalen Geländemodellen nimmt man hingegen oft an, daß f zumindest stetig, manchmal auch glatter ist. Bei der Glattheit von f wird in der Regel unterschieden, ob Glattheit zwischen den Datenpunkten oder auch an den Datenpunkten benötigt wird. Oft ist Glattheit an den Datenpunkten weniger wichtig was auch die Konstruktion von Interpolanten erheblich vereinfacht. Stückweise Interpolation ist in der Regel einfacher zu realisieren und bietet zufriedenstellendere Ergebnisse als globale Konstruktionen.

So gut wie alle Interpolationstechniken verwenden zur Konstruktion von f eine *Basis*

$$\phi = \{\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots\}$$

und somit eine Darstellung von f als Linearkombination

$$f(\mathbf{x}) = \sum_{i=1}^{N} c_i \cdot \phi_i(\mathbf{x}).$$

mit Koeffizienten c_i. Die Koeffizienten können duch Lösung des Gleichungssystems

$$\mathbf{\Phi c} = \mathbf{y}$$

mit $\mathbf{\Phi} := (\phi_j(\mathbf{x}_i))_{ij}$, $\mathbf{c} := (c_i)_i$, und $\mathbf{y} := (y_i)_i$ berechnet werden.

Auf diese Weise schlagen sich die Modellannahmen für f genau in der Wahl einer geeigneten Basis nieder. Dabei heißt eine Basis *lokal*, falls die Träger der Basisfunktionen $supp(\phi_i)$ kompakt sind, d.h. wenn die ϕ_i außerhalb eines bestimmten Bereichs gleich 0 sind. Ansonsten nennt man die Basis *global*. Lokale Basen implizieren bei geeigneter Sortierung eine Bandstruktur auf $\mathbf{\Phi}$. Vollbesetzte Matrizen $\mathbf{\Phi}$ würden die Berechnung der Koeffizienten $\mathbf{c}$ für große Datenmengen unmöglich machen.

Durch die Wahl einer Basis ϕ wird ein *Funktionenraum* V, bestehend aus allen Funktionen, die aus Linearkombination der Basisfunktionen erzeugt werden können, definiert. Bei gegebenen Daten $\mathbf{y}$ ist f dann ein Element aus diesem Funktionenraum.

Bei der *zerlegungsbasierten Interpolation* nimmt man zur Konstruktion von f neben den Daten $(\mathbf{x}_i, y_i)$ auch noch eine Zerlegung T (wie im letzten Kapitel besprochen) zu Hilfe. Dies ist vor allem in Dimensionen größer gleich zwei

einsichtig, nachdem durch die Zerlegung Nachbarschaften zwischen den Datenpunkten definiert werden, was die Konstruktion einer (in der Regel lokalen) Basis wesentlich erleichtert.

3.1.1 Lagrange Interpolation

Ein klassisches Beispiel für eine globale Konstruktion ist die Lagrange Interpolation. Dabei liegt die Idee zugrunde, daß durch die N Datenpunkte ein Polynom von maximalem (totalen) Grad N aufgespannt wird. Für diese Konstruktion wird zunächst keine Zerlegung T benötigt, wir werden aber später Lagrange Interpolanten auf den einzelnen Teilgebieten einer Zerlegung verwenden.

Im Eindimensionalen ist die resultierende Basis durch die Lagrange'schen Fundamentalpolynome

$$\phi_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}.$$

gegeben. Damit gilt

$$\phi_i(x_j) = \delta_{ij}$$

wobei δ_{ij} das Kronecker–Symbol ist, d.h.

$$\delta_{ij} = \begin{cases} 1 & \text{falls } i = j \\ 0 & \text{sonst} \end{cases} .$$

Somit ist $\mathbf{\Phi} = \mathbf{I}$ die Einheitsmatrix und $\mathbf{c} = \mathbf{y}$. Im übrigen wird jede Basis mit dieser Eigenschaft *Stützpunkt-* oder *Lagrange Basis* genannt. Wie wir aber in Kürze sehen werden bilden die Lagrange'schen Fundamentalpolynome nicht die einzige Basis dieser Art.

Im Mehrdimensionalen gestaltet sich die Konstruktion allerdings wesentlich schwieriger. Zum einem ist ein Lagrange'sches Basispolynom $\phi_i(\mathbf{x})$ von totalem Grad N, also

$$\phi_i(\mathbf{x}) = \sum_{i_1=0}^{k_1} \sum_{i_2=0}^{k_2} \dots \sum_{i_d=0}^{k_d} a_{\mathbf{i}} p_{\mathbf{i}}(\mathbf{x}),$$

mit

$$p_{\mathbf{i}}(\mathbf{x}) = x_1^{i_1} \cdot x_2^{i_2} \cdot \ldots \cdot x_d^{i_d},$$

wobei $k_1 + k_2 + \ldots + k_d = N$, nicht eindeutig definiert, da N auf verschiedene Weisen durch eine solche Summe von Polynomgraden k_i dargestellt werden kann. Dies führt zu Problemen, da bei gegebenem $\mathbf{k}$ für ein oder mehrere Punkte $\mathbf{x}_j$ das entsprechende Fundamentalpolynom nicht notwendigerweise existiert, bzw. dazu äquivalent das Gleichungssystem

$$\mathbf{Pa} = \mathbf{e}_j,$$

mit $\mathbf{P} := (p_{\mathbf{i}}(\mathbf{x}_j))_{j\mathbf{i}}$, $\mathbf{a} := (a_{\mathbf{i}})_{\mathbf{i}}$ und $\mathbf{e}_j$ dem j–tem Einheitsvektor, nicht lösbar ist. Dieses Problem kann durch eine Suche nach einem passenden Grad $\mathbf{k}$ gelöst werden, jedoch wird dies in größeren Dimensionen schon für relativ kleine N unpraktikabel.

In jedem Fall ist die Matrix $\mathbf{P}$ vollbesetzt, was im Mehrdimensionalen zu einem unvertretbaren Aufwand für große N führt, da Neville–artige Schemen oder dividierte Differenzen aus dem Eindimensionalen nicht mehr anwendbar sind. Außerdem neigen globale Interpolationspolynome zu Überschwingern und Oszillationen, was sich negativ auf die Approximationsgüte auswirkt.

3.1.2 Finite Elemente

Um die Nachteile der globalen Lagrange Konstruktion auszugleichen, werden oft stückweise lokal definierte Basen verwendet. Um Lokalität geeignet definieren zu können, wird dabei in höheren Dimensionen eine Zerlegung T benötigt. Finite Elemente sind ein klassisches Beispiele für solche stückweise definierte Basen. Dabei wird die gesuchte Funktion f mit Hilfe einer Lagrange Basis ϕ aufgebaut. Die so entstehenden Basisfunktionen haben dann lokalen, d.h. finiten, Träger.

Bei äquidistanten Gittern, wie sie oft zur Diskretisierung partieller Differentialgleichungen verwendet werden, kann man die Basisfunktionen ϕ alle gleich wählen und erhält damit den besonders einfachen Fall:

$$\phi_i(\mathbf{x}) = \phi(\mathbf{x} - \mathbf{x}_i).$$

Wir wollen uns allerdings im folgenden wenn möglich nicht nur auf den äquidistanten Fall beschränken.

Stückweise konstante Interpolation

Die einfachste Möglichkeit ist die Verwendung stückweise konstanter Basisfunktionen, d.h.

$$\phi_i(\mathbf{x}) := \begin{cases} 1 & \text{falls } \mathbf{x} \in S_i \\ 0 & \text{sonst} \end{cases} .$$

Dies funktioniert so allerdings nur, wenn gemeinsame Ränder zweier oder mehrerer Teilgebiete S_i eindeutig einem der Teilgebiete zugeordnet sind, d.h. die S_i wirklich eine disjunkte Zerlegung des Grundgebiets bilden. Ansonsten können entweder manche Interpolationsbedingungen nicht erfüllt werden (z.B. bei knotenzentrierten Diskretisierungen) oder die Darstellung wäre keine Zerlegung der Eins, d.h. für einen Datensatz, bei dem alle Datenwerte $y_i = 1$ sind, würde nicht $f \equiv 1$ gelten.

Meist wird diese Konstruktion daher bei zellenzentrierten Diskretisierungen verwendet. Auf diese Weise erhalten alle Punkte, die in der Umgebung eines Datenpunkts liegen, den gleichen Funktionswert wie der entsprechende Datenwert. Bei knotenzentrierten Diskretisierungen würden alle Punkte innerhalb eines Teilgebiets den gleichen Funktionswert wie der Datenwert an einem der Eckpunkte des Teilgebiets erhalten.

Diese Form der Interpolation wird i.d.R. bei digitalen Bildern angewandt, wobei die Teilgebiete Pixel (2D) bzw. Voxel (3D) sind. Die entstehenden Interpolanten sind nicht stetig an den Pixel bzw. Voxelrändern.

Stückweise lineare Interpolation

Die bei Triangulierungen mit Abstand am häufigsten verwendete Konstruktion basiert auf stückweise linearer Interpolation und ist knotenzentriert. Hierbei sind die Basisfunktionen stückweise linear auf jedem Simplex, wobei ausgenutzt wird, daß die Summe von linearen Funktionen wieder linear ist.

Seien $\mathbf{x}_{i_0}, \ldots, \mathbf{x}_{i_d}$ die $d+1$ Eckpunkte des Simplex S_i aus der Triangulierung T. Auf dem Simplex S_i gibt es für Datenwerte $y_{i_0}, \ldots y_{i_d}$ an seinen Eckpunkten eine eindeutige lineare Funktion $g_{S_i,\mathbf{y}}(\mathbf{x}) = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_d x_d$, für die $g_{S_i,\mathbf{y}}(\mathbf{x}_{i_k}) = y_{i_k}$ gilt. Die Polynomkoeffizienten a_j sind durch das so entstehende Gleichungssystem bestimmt. Dann ist

$$\phi_j(\mathbf{x}) := g_{S_i,\mathbf{e}_j}(\mathbf{x}) \text{ für } \mathbf{x} \in S_i,\ i = 1, 2, \ldots$$

die stückweise lineare Finite Element–Basisfunktion zum Punkt $\mathbf{x}_j$ und es gilt für $\mathbf{x} \in S_i$:

$$f(\mathbf{x}) = \sum_{k=1}^{N} y_k \cdot \phi_k(\mathbf{x}) = \sum_{j=0}^{d} y_{i_j} \cdot g_{S_i,\mathbf{e}_{i_j}}(\mathbf{x}) = g_{S_i,\mathbf{y}}(\mathbf{x}).$$

In vollkommen analoger Weise werden stückweise d–lineare Interpolanten auf Zerlegungen bestehend aus allgemeinen (nichtdegenerierten) Hyperquadern konstruiert. Dabei ist $g_{S_i,\mathbf{y}}(\mathbf{x}) = \sum_{\mathbf{k}} a_{\mathbf{k}} \prod_{j=1}^{d} x_j^{k_j}$, $k_j \in \{0,1\}, j = 1, \ldots, d$, eine durch die 2^d Eckpunkte des Hyperquaders S_i eindeutig bestimmte d–lineare Funktion.

Finite Elemente höherer Ordnung

Die Konstruktion stückweiser Interpolanten höherer Ordnung funktioniert im Prinzip ähnlich wie im linearen Fall. Dabei ist g z.B. quadratisch oder kubisch. Bei der Konstruktion von g können allerdings ein paar Probleme auftauchen. Sei $g_{S_i,\mathbf{y}}(\mathbf{x}) = \sum_{\mathbf{k} \subset \mathbb{N}_0^d} a_{\mathbf{k}} \prod_{j=1}^{d} x_j^{k_j}$ ein allgemeines multivariates Polynom. Dann müssen zur Konstruktion von g auf S_i genau $|\mathbf{k}|$ Interpolationsbedingungen fixiert werden.

Dafür werden in der Regel neben den Eckpunkten von S_i noch weitere Punkte außerhalb von S_i zu Hilfe gezogen. Dabei treten zwei Schwierigkeiten auf. Zum einen müssen die Punkte so gewählt werden, daß das entstehende Gleichungssystem zur Bestimmung der Koeffizienten $a_{\mathbf{k}}$ regulär ist. Zum anderen muß darauf geachtet werden, daß der Interpolant an den Rändern der Teilgebiete stetig ist. Dies ist vor allem in Dimensionen größer als 1 ein Problem. Die Basisfunktion ϕ_i ist zwar immer an den Eckpunkten $\mathbf{x}_j \neq \mathbf{x}_i$ von S_i gleich 0, kann aber auf dem restlichen Rand von S_i ungleich 0 sein. Stetigkeit muß daher durch weitere Interpolationsbedingungen sichergestellt werden.

Im Prinzip können die Interpolationspunkte aller Teilgebiete S_i unterschiedlich gewählt werden. Falls dies nicht der Fall ist, d.h. wenn die Interpolationspunkte für mehrere (nebeneinander liegende) Teilgebiete $S_{i_1}, S_{i_2}, \ldots$ identisch sind, spricht man von *innerer Interpolation* auf der Vereinigung der Teilgebiete $\cup_j S_{i_j}$ (Beispiele hierfür sind sog. Clough–Tocker Elemente [14]).

3.1.3 Splines und B–Splines

Die im letzten Abschnitt konstruierten Funktionen sind zwar stetig, aber an den Teilgebietsränden nicht glatt. Glattere Interpolanten können durch kompatible Wahl der Funktionswerte und höherer Ableitungen der Basisfunktionen an den Teilgebietsrändern konstruiert werden. Dies führt zu verallgemeinerten Spline–Konstruktionen. Dabei muß man allerdings davon abkommen, daß $\mathbf{\Phi}$ die Einheitsmatrix ist, und die Berechnung der Koeffizienten $\mathbf{c}$ erfordert die Lösung eines (meist bandstrukturierten) linearen Gleichungssystems.

Bei B–Spline Flächen bilden die Datenpunkte sogenannte Kontrollpunkte, durch die die Ableitungen der Fläche implizit bestimmt werden. Auf diese Weise wird eine Approximationsfunktion definiert, die nicht notwendigerweise durch diese Punkte führen muß, aber in deren konvexer Hülle liegt. B–Splines werden in vielen Bereichen von CAD eingesetzt, sind aber nicht interpolatorisch.

3.2 Hierarchische Interpolation

In diesem Abschnitt wollen wir nun die Interpolationstechniken des letzten Abschnitts auf hierarchische Zerlegungen übertragen. Dabei liegt die Idee zugrunde, daß durch die hierarchische Zerlegung eine Menge immer genauer werdender Approximationen definiert werden kann.

Sei $\{T^l\}$ eine hierarchische Zerlegung. Basierend auf dieser Zerlegung ist es möglich, $l_{max}+1$ verschiedene Interpolanten f^l unterschiedlicher Komplexität mit Hilfe einer separaten Basis für jeden Level $\phi^l = \{\phi^l_1, \phi^l_2, \ldots\}$ zu konstruieren:

$$f^l(\mathbf{x}) = \sum_{i:\mathbf{x}_i \in T^l} c^l_i \cdot \phi^l_i(\mathbf{x}).$$

Durch die verschiedenen Basen werden auch $l_{max}+1$ Funktionenräume

$$V^l = span\{\phi^l_i\}$$

aufgespannt.

Diese Repräsentation von f nennt sich pyramidale Darstellung und erlaubt prinzipiell schon die Darstellung von f auf unterschiedlichen Detailstufen. Sie hat allerdings mehrere Nachteile. Zum einen erfordert die sie die Lösung mehrerer Gleichungssysteme (natürlich unterschiedlicher Größe) zur Berechnung der Koeffizienten c^l_i, nämlich eines für jeden Level. Zum zweiten ist die Darstellung nicht sonderlich kompakt, da zur Speicherung einer hierarchischen Darstellung von f die Koeffizienten aller Level und nicht nur des feinsten Levels benötigt werden. Zum dritten gestaltet sich adaptive Verfeinerung, d.h. eine Zusammensetzung von f aus Teilen f^l unterschiedlicher Level, schwierig.

Diese Probleme können in sowohl mathematisch als auch implementatorisch eleganter Weise durch Wavelets gelöst werden. Im folgenden werden wir zunächst die Grundlagen der Wavelet–Theorie beschreiben. Daraufhin werden wir verschiedenen Typen von Wavelets charakterisieren. In den danach folgendenen

Abschnitten werden dann konkrete Wavelet–Konstruktionen genauer betrachtet.

3.2.1 Wavelets

In vielen Textbüchern werden die Wavelet–Analyse und Wavelets über die kontinuierliche Fourier–Analyse hergeleitet. Diskretisierung führt dann zur schnellen Wavelet–Transformation. Diese Vorgehensweise hat jedoch zwei Nachteile. Zum einen ist die Verallgemeinerung der eindimensionalen Fourier–Analyse auf allgemeine mehrdimensionale Zerlegungen nur sehr schwer möglich. Zum anderen sind die betrachteten Eingabedaten immer endlich und die Weg über die kontinuierliche Wavelet–Analyse ist damit eigentlich nicht nötig. Wir werden daher die Wavelet–Analyse gleich für endliche Funktionenräume einführen (siehe [132]) wodurch sich alle Filter und Transformationen als (endliche) Matrizen schreiben lassen.

Bei der Wavelet–Analyse liegt die Idee zugrunde, nicht $l_{max}+1$ verschiedene Basen, wie in der pyramidalen Darstellung, zu konstruieren, sondern nur eine einzige für alle Level. Der wichtigste Fall ist hier wenn die sowohl die Zerlegung als auch die Funktionenräume geschachtelt sind, d.h.

$$V^0 \subset V^1 \subset \ldots \subset V^{l_{max}},$$

da ansonsten die Darstellung nicht effizienter als die pyramidale wäre. Dann kann f durch eine Teleskopsumme über die Level dargestellt werden:

$$f(\mathbf{x}) = \sum_{i:\mathbf{x}_i \in T^0} c_i^0 \cdot \phi_i^0(\mathbf{x}) + \sum_{l=0}^{l_{max}-1} \sum_{i:\mathbf{x}_i \in T^{l+1} \setminus T^l} d_i^l \cdot \psi_i^l(\mathbf{x}).$$

Das heißt beim Übergang von Level l zu Level $l+1$ werden nur die Datenpunkte $\mathbf{x}_i$ betrachtet, die durch Verfeinerung neu hinzukommen und an diesen Punkten aufsetzende Basisfunktionen hinzugefügt. Die so definierten Basisfunktionen ψ_i^l heißen *Wavelets* und die d_i^l *Wavelet–Koeffizienten*. Die Basisfunktionen ϕ_i^l der pyramidalen Darstellung heißen im diesem Kontext auch *Skalierungsfunktionen*. Für die Wavelets ψ_i^l muß offenbar gelten:

$$span\{\psi_i^l\} = V^{l+1} \setminus V^l = W^l,$$

sie spannen also gerade den Differenzraum zwischen V^l und V^{l+1} auf. Wir wollen nun die Basisfunktionen in Zeilenvektoren zusammenfassen, d.h.

$$\mathbf{\Phi}^l := (\phi_{i_1}^l \ldots \phi_{i_{n^l}}^l)$$

und

$$\mathbf{\Psi}^l := (\psi_{i_1}^l \ldots \psi_{i_{n^{l+1}-n^l}}^l),$$

wobei $n^l := dim(V^l)$ die Zahl der Gitterpunkte der Zerlegung auf dem Level l ist und damit $n^{l+1} - n^l$ die Zahl der Gitterpunkte, die durch Verfeinerung der Zerlegung des Levels l entstanden sind.

Nachdem die Räume V^l geschachtelt sind, muß es eine $n^l \times n^{l-1}$ Matrix $\mathbf{P}^l$ geben, mit der

$$\mathbf{\Phi}^{l-1} = \mathbf{\Phi}^l \mathbf{P}^l$$

gilt, und nachdem W^{l-1} nach Definition ein Teilraum von V^l ist, eine $n^l \times (n^l - n^{l-1})$ Matrix $\mathbf{Q}^l$ für die gilt

$$\mathbf{\Psi}^{l-1} = \mathbf{\Phi}^l \mathbf{Q}^l.$$

Diese Gleichungen werden auch *Zwei–Skalen Relationen* der Skalierungsfunktionen und der Wavelets genannt und die Matrizen $\mathbf{P}^l$ und $\mathbf{Q}^l$ werden *Synthesematrizen* genannt. Im shift–invarianten Fall haben $\mathbf{P}^l$ und $\mathbf{Q}^l$ Bandstruktur und nur wenige verschiedene Einträge, die sich versetzt wiederholen.

Mit Hilfe dieser Matrizen kann unter Kenntnis der Koeffizientenvektoren $\mathbf{c}^{l-1}$ und $\mathbf{d}^{l-1}$ der Koeffizientenvektor $\mathbf{c}^l$ durch

$$\mathbf{c}^l = \mathbf{P}^l \mathbf{c}^{l-1} + \mathbf{Q}^l \mathbf{d}^{l-1}$$

berechnet werden. Dieser Vorgang heißt *Synthese* oder Rekonstruktion. Der inverse Schritt, auch *Analyse* oder Zerlegung genannt, ist durch die Gleichungen

$$\mathbf{c}^{l-1} = \mathbf{A}^l \mathbf{c}^l$$

und

$$\mathbf{d}^{l-1} = \mathbf{B}^l \mathbf{c}^l$$

gegeben, wobei $\mathbf{A}^l$ eine $n^{l-1} \times n^l$ Matrix und $\mathbf{B}^l$ eine $(n^l - n^{l-1}) \times n^l$ Matrix definiert durch

$$\begin{pmatrix} \mathbf{A}^l \\ \mathbf{B}^l \end{pmatrix} = \begin{pmatrix} \mathbf{P}^l & \mathbf{Q}^l \end{pmatrix}^{-1}$$

sind. Die Matrizen $\mathbf{A}^l$ und $\mathbf{B}^l$ heißen Analysematrizen.

Der rekursive Prozeß, welcher beginnend mit $l = l_{max}$ und endend mit $l = 1$ jeweils $\mathbf{c}^{l-1}$ und $\mathbf{d}^{l-1}$ aus $\mathbf{c}^l$ berechnet, wird *Filter–Bank Algorithmus* oder *inverse Wavelet Transformation* genannt. Der umgekehrte Prozess heißt *Wavelet Transformation*. In diesem Sinne ist in den Koeffizienten

$$\mathbf{c}^l \text{ und } \mathbf{c}^0, \mathbf{d}^0, \ldots, \mathbf{d}^{l_{max}-1}$$

die gleiche Information enthalten. Die reguläre Matrix T definiert durch

$$\mathbf{c}^{l_{max}} = \mathbf{T} \begin{pmatrix} \mathbf{c}^0 \\ \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{l_{max}-1} \end{pmatrix}$$

heißt *Wavelet–Transformationsmatrix*. Sie ist definiert aus Produkten der Matrizen $\mathbf{P}^l$ und $\mathbf{Q}^l$ mit Sortierungsmatrizen. Die inverse Wavelet–Transformationsmatrix $\mathbf{T}^{-1}$ besteht in analoger Weise aus Produkten der Matrizen $\mathbf{A}^l$ und $\mathbf{B}^l$ mit Sortierungsmatrizen. Die Wavelet–Transformationsmatrizen werden jedoch in der Regel nicht explizit aufgestellt. Bei geeigneter Wahl der Skalierungsfunktionen und Wavelets kann mit Hilfe der Matrizen $\mathbf{A}^l$, $\mathbf{B}^l$, $\mathbf{P}^l$, und $\mathbf{Q}^l$ die Wavelet- und inverse Wavelet–Transformationen in $O(n^{l_{max}})$ Operationen durchgeführt werden.

3.2.2 Typen von Wavelets

Schon aus der Definition deutlich, daß die Wavelets nicht eindeutig definiert sind, da Linearkombinationen der $\psi_{i_j}^l$ ebenfalls Basen von $V^{l+1} \setminus V^l$ bilden. An die Waveletbasis ψ kann man daher wieder unterschiedliche Anforderungen stellen, z.B.:

- Lokalität
- Grobgitter–Interpolation
- Glattheit
- Orthogonalität
- verschwindende Momente

Die Eigenschaft, daß die Basisfunktionen ψ_i^l minimalen lokalen Träger haben ist oft von entscheidender Bedeutung für eine effiziente Durchführung der Wavelet- und inversen Wavelet–Transformation. Dann haben zumindest die Matrizen $\mathbf{P}^l$ und $\mathbf{Q}^l$ Bandstruktur (für $\mathbf{A}^l$ und $\mathbf{B}^l$ ist dies nicht immer der Fall, aber wir werden Fälle kennenlernen, bei denen dies auch für die Analyse–Matrizen gilt). In der angegebenen Wavelet–Analyse werden zwar immer die Feingitterpunkte aber nicht notwendigerweise die Grobgitterpunkte exakt interpoliert, was aber für viele Algorithmen, wie wir später sehen werden, eine wichtige Eigenschaft ist. Wavelets, die neben den Feingitterpunkten auch die Grobgitterpunkte interpolieren, werden *Interpolets* genannt. Selbst wenn die Skalierungsfunktionen glatt sind, muß dies nicht notwendigerweise auch für die Wavelets gelten. Glattheit ist ebenfalls in vielen Anwendungen (z.B. PDEs oder der Visualisierung) wichtig. Orthogonalität und die Existenz verschwindender Momente ist bei der Datenkompression sowie bei der Diskretisierung von PDEs mit Wavelet Basen von zentraler Bedeutung.

Orthogonale Wavelets

Eine orthogonale Waveletbasis ist dadurch charakterisiert, daß alle Skalierungsfunktionen zueinander, alle Wavelets zueinander und die Wavelets und Skalierungsfunktion untereinander orthogonal sind, d.h.

$$(\phi_i^l, \phi_j^l) = \delta_{ij}, (\psi_i^l, \psi_j^l) = \delta_{ij} \text{ und } (\phi_i^l, \psi_j^l) = 0.$$

für alle i, j und l. Damit gilt insbesondere $\mathbf{A}^l = (\mathbf{P}^l)^T$ und $\mathbf{B}^l = (\mathbf{Q}^l)^T$. Dadurch sind nicht nur $\mathbf{A}^l$ und $\mathbf{B}^l$ leicht berechenbar, sondern sie haben für lokale Wavelets garantiert auch Bandstruktur. Beispiele für orthogonale Wavelets sind die Haar oder Daubechies Wavelets [27].

Semiorthogonale Wavelets

Orthogonalität ist jedoch eine starke Einschränkung. Zum Beispiel existieren keine Wavelets, die gleichzeitig glatt, symmetrisch und kompakten Träger haben. Daher wird oft auf semiorthogonale Wavelets (auch *Prewavelets* genannt)

zurückgegriffen. Hier müssen die Wavelets nur noch orthogonal zu allen gröberen Skalierungsfunktionen oder

$$(\phi_i^{l-1}, \psi_j^l) = 0.$$

für alle i, j und l. Beispiele für semiorthogonale Wavelets sind die B–Spline Wavelets nach Chui und Wang [18]. Semi–orthogonale Wavelets für irreguläre hierarchische Triangulierungen wurden in [40, 48, 86, 98, 131] entwickelt.

Biorthogonale Wavelets

Im Gegensatz zu orthogonalen Wavelets gibt es jedoch bei semiorthogonalen Wavelets keine Garantie, daß die Analysematrizen $\mathbf{A}^l$ und $\mathbf{B}^l$ Bandstruktur besitzen. Weiterhin eignen sich sowohl orthogonale als auch semiorthogonale Wavelets nicht für adaptive Verfeinerung, da i.d.R. bei der Verfeinerung nicht nur ein einzelner Punkt eingefügt werden kann sondern immer nur eine Menge von Punkten. Dies hat letztendlich zur Entwicklung biorthogonaler Wavelets [24] geführt.

Die eindeutig definierte duale Basis $\tilde{\mathbf{\Phi}}^l$ zu einer gegebenen Basis $\mathbf{\Phi}^l$ ist definiert durch

$$(\tilde{\mathbf{\Phi}}^l, \mathbf{\Phi}^l) = \mathbf{I},$$

wobei $(\cdot, \cdot)$ das komponentenweise Skalarprodukt auf Matrizen bezeichnet. In analoger Weise existiert zur Wavelet–Basis $\mathbf{\Psi}^l$ eine duale Wavelet–Basis $\tilde{\mathbf{\Psi}}^l$. Biorthogonale Wavelets sind dann charakterisiert durch

$$(\phi_i^l, \tilde{\psi}_j^l) = 0 \text{ und } (\psi_i^l, \tilde{\phi}_j^l) = 0$$

für alle i, j und l, d.h. die Skalierungsfunktionen sind orthogonal zu den dualen Wavelets und die Wavelets sind orthogonal zu den dualen Skalierungsfunktionen. Auf diese Weise können auch bandstrukturierte Analysematrizen, z.B. durch lifting (siehe Abschnitt 3.2.6) erreicht werden. Biorthogonale Wavelets für irreguläre hierarchische Triangulierungen wurden z.B. in [34, 128] mit Hilfe von B–Splines entwickelt.

3.2.3 Hierarchische Finite Elemente

Hierarchische Finite Elemente gehen wohl auf Faber [46] zurück, der Anfang des Jahrhunderts zur Definition von Stetigkeit eine eindimensionale stückweise lineare hierarchische Basis konstruiert hat. Diese Ansätze wurden aber erst in den achtziger Jahren im Rahmen von adaptiver Verfeinerung bei Finite Elemente Methoden wiederentdeckt [4, 102, 149].

Bei hierarchischen Finiten Elementen liegt die Idee zugrunde, als Basis für $V^{l+1} \setminus V^l$ einfach $\{\phi_{i_1}^{l+1}, \phi_{i_2}^{l+1}, \ldots\}$ mit $\mathbf{x}_{i_j} \in T^{l+1} \setminus T^l$ zu verwenden, d.h.

$$\psi_i^l(\mathbf{x}) := \phi_i^{l+1}(\mathbf{x})$$

für $l = 0, \ldots, l_{max} - 1$. Daher muß nichts getan werden, um die Wavelet–Basisfunktionen zu berechnen, daher wird diese Basis auch manchmal auch *lazy wavelet* Basis genannt.

Falls die Skalierungsfunktionen $\{\phi_i^l\}$ eine Lagrange–Basis bilden, ist diese Basis damit auch interpolatorisch auf gröberen Leveln, d.h. jede Approximation interpoliert alle Punkte der gröberen Zerlegung. Falls die Skalierungsfunktionen lokal sind, ist die hierarchische Finite Element Basis zugleich die Wavelet–Basis mit dem lokalst möglichen Träger.

Weiterhin gilt

$$d_i^l = y_i^l - \left(\sum_{j:\mathbf{x}_j \in T^0} c_j^k \cdot \phi_j^0(\mathbf{x}_i) + \sum_{k=0}^{l-1} \sum_{j:\mathbf{x}_j \in T^{k+1} \setminus T^k} d_j^k \cdot \psi_j^l(\mathbf{x}_i) \right),$$

d.h. die Wavelet–Koeffizienten haben eine geometrische Bedeutung: der Koeffizient d_i^l entspricht gerade der Differenz zwischen dem Funktionswert am Punkt $\mathbf{x}_i$ und dem Interpolanten vom Level $l-1$ an diesem Punkt. Nachdem bei einer lokalen Basis nur wenige Basisfunktionen dort ungleich Null sind, lassen sich die Wavelet–Koeffizienten sehr schnell und leicht als Linearkombination der Funktionswerte benachbarter Punkte von $\mathbf{x}_i$ berechnen. Die inverse Wavelet–Transformation entsteht dann genau durch Umkehrung der Vorzeichen in dieser Linearkombination. Für reguläre hierarchische Zerlegungen lassen sich die Gewichte der Linearkombination vorab berechnen.

3.2.4 Hierarchische Lagrange Interpolation

Die Verallgemeinerung finiter Elemente höherer Ordnung auf hierarchische Zerlegungen führt zwangsweise dazu, daß lokale Gleichungssysteme zur Berechnung der Polynom- und damit der Wavelet–Koeffizienten gelöst werden müssen. Um dies zu umgehen, wurden in [15] zur Definition einer eindimensionalen hierarchischen Lagrange–Basis zur Berechnung der Polynomkoeffizienten Funktionswerte benachbarter Punkte gröberer Gitterlevel verwendet. Die Verallgemeinerung auf den mehrdimensionalen Fall erfolgt dann durch Tensorprodukt–Bildung. Auf diese Weise kann auf dem Level l eine an den Gebietsrändern C^0–stetige stückweise Polynombasis vom Grad l konstruiert werden. Die so entstehende Basis ist interpolatorisch und biorthogonal. Allerdings gestaltet sich eine Verallgemeinerung der Konstruktion auf beliebige Zerlegungen schwierig.

3.2.5 Hierarchische B-Splines

In [36, 50] wurde die B–Spline Konstruktion auf hierarchische Zerlegungen ebenfalls für den Fall rechteckiger, äquidistanter Gitter verallgemeinert. Auf diese Weise können B–Splines durch Hinzufügen von Punkten lokal verfeinert werden. An die eingefügten Punkte und Teilgebiete müssen allerdings starke Anforderungen gestellt werden, damit alle bei der Konstruktion benötigten Freiheitsgrade abgedeckt werden. Diese Konstruktion kann daher ebenfalls nicht auf allgemeine Zerlegungen angewandt werden. Auf diese Weise entsteht eine nicht interpolatorische parametrische hierarchische Basis, welche vor allem zum Datenapproximation und zur Datenmanipulation auf rechteckigen Gittern verwendet wird. Flexiblere B–Spline Wavelets auf adaptiven Quadtrees wurden in [68] entwickelt.

3.2.6 Lifting

Lifting [26, 133] bezeichnet den Vorgang, eine bestehende biorthogonale Basis (wie z.B. aus den vorhergehenden drei Beispielen) so zu modifizieren, daß wieder eine biorthogonale Basis allerdings mit besseren Eigenschaften, wie mehr verschwindende Momente oder Orthogonalität, entsteht. Hierbei werden von den Wavelets Anteile gröberer Skalierungsfunktionen abgezogen oder hinzugenommen. Formal besteht die Konstruktion aus

$$\mathbf{Q}^l \longrightarrow \mathbf{Q}^l - \mathbf{P}^l\mathbf{S}^l,$$

$$\mathbf{A}^l \longrightarrow \mathbf{A}^l + \mathbf{S}^l\mathbf{B}^l.$$

für eine beliebige Matrix $\mathbf{S}^l$. Die Matrizen $\mathbf{P}^l$ und $\mathbf{B}^l$ bleiben unberührt. Durch geeignete Wahl von $\mathbf{S}^l$ können die so entstehenden Wavelets mehr verschwindende Momente oder bessere Orthogonalitätseigenschaften bekommen. In der Regel wird $\mathbf{S}^l$ so dünn besetzt wie möglich gewählt um den Rechenaufwand gering zu halten.

Die Wavelet–Transformation kann am Platz, d.h. ohne zusätzlichen Speicheraufwand vorgenommen werden. Dabei werden nach jedem Analyse–Schritt mit Hilfe der gerade berechneten Wavelet–Koeffizienten $\mathbf{d}^{l-1}$ die Koeffizienten der Skalierungsfunktionen $\mathbf{c}^{l-1}$ entsprechend der Matrix $\mathbf{S}^l$ modifiziert.

3.3 Hierarchische Interpolation bei der Simplexbisektion

Wir wollen nun abschliessend zu diesem Kapitel in diesem Abschnitt einige konkrete Konstruktionen hierarchischer Basen und Wavelets basierend auf Bisektionszerlegungen definieren. Wir werden die entsprechenden Zwei–Skalen Relationen sowie die Einträge der Analyse- und Synthese–Matrizen angeben. Der Abschnitt ended mit einem kleinen Programm, welches zeigen soll, wie leicht ein solches Verfahren umgesetzt werden können.

3.3.1 Hierarchische Finite Elemente

Die grundlegenste Konstruktion hierarchischer Finiter Elemente basiert auf stückweise linearer Interpolation. In einer Dimension entspricht das der Konstruktion von Faber [46]. Sei nun die hierarchische Zerlegung die Intervallbisektion aus Abschnitt 2.2.1. Die Skalierungsfunktionen entsprechen dann linearen Hutfunktionen, wobei die Skalierungsfunktion ϕ_i^l am Punkt $\mathbf{x}_i^l$ gleich Eins ist, an den benachbarten Punkten gleich Null und linear abfallend auf den Intervallen S_i^l und S_{i-1}^l. Die Wavelet–Basisfunktionen bestehen dann gerade aus den ungeraden Skalierungsfunktionen (siehe Abb. 3.1).

Die Wavelet–Koeffizienten können dann einfach durch

$$d_i^l = -\frac{1}{2}f(\mathbf{x}_{i-1}^l) + f(\mathbf{x}_i^l) - \frac{1}{2}f(\mathbf{x}_{i+1}^l)$$

berechnet werden.

In zwei Dimensionen geht die Konstruktion auf Rivara [114] zurück. Betrachten wir nun die Dreiecksbisektion aus Abschnitt 2.2.2. Als Skalierungsfunktionen werden nun zwei Typen von Basisfunktionen verwendet. Beide haben die Form einer Pyramide mit quadratischer Grundfläche, wobei der eine Typ achsenparallel und der andere um 45 Grad gedreht ist (Abb. 3.2). Der erste Typ tritt hierbei in geraden Leveln, der zweite in ungeraden Leveln auf. Die Waveletbasen bestehen wiederum aus jeder zweiten Skalierungsfunktion, und liegen damit auf einem achsenparallelen Gitter in den ungeraden Leveln und auf einem um 45 Grad gedrehten (Quincunx-) Gitter in den geraden Leveln. In Abb. 3.5 ist die Zwei–Skalen Relation für den ersten Typ von Skalierungsfunktionen abgebildet. Die entsprechende Relation für den zweiten Typ entsteht durch Rotation aller Basisfunktionen um 45 Grad und Skalierung mit $1/\sqrt{2}$.

Die Wavelet–Koeffizienten können wiederum durch

$$d_i^l = -\frac{1}{2} f(\mathbf{x}_{a(i)}^l) + f(\mathbf{x}_i^l) - \frac{1}{2} f(\mathbf{x}_{b(i)}^l)$$

ermittelt werden, wobei hier $a(i)$ und $b(i)$ die Indizes der Endpunkte der Verfeinerungskante, auf der $\mathbf{x}_i^l$ liegt, sind.

In drei Dimensionen treten nun drei verschiedene Typen von Skalierungsfunktionen und Wavelets auf, die allerdings nicht mehr zueinander ähnlich sind. Basierend auf der Tetraederbisektion aus Abschnitt 2.2.3 hat der Träger des ersten Typs die Form eines Würfels, der des zweiten Typs die Form eines Oktaeders und der des dritten Typs die Form eines Diamanten. Die Basisfunktionen sind stückweise linear auf den der Verfeinerungskante benachbarten Tetraedern, wie in Abb. 3.3 gezeigt. Dargestellt sind hier transparente Isoflächen der jeweiligen Skalierungsfunktionen für Isowerte von 1.0, 0.66 und 0.33. Die verschiedenen Typen treten je nach Lage der Verfeinerungskante um 90 Grad in den entsprechenden Richtungen gedreht auf. Die entsprechenden Zwei–Skalen Relationen sind in Abb. 3.6 abgebildet.

Die Wavelet–Koeffizienten werden im dreidimensionalen Fall (wie übrigens dann auch in beliebigen Dimensionen) genau wie im zweidimensionalen Fall durch lineare Interpolation der Funktionswerte an den Endpunkten der Verfeinerungskante berechnet. Damit ist die Wavelet–Transformation und die inverse Wavelet–Transformation von ihrem Rechenaufwand unabhängig von der Dimension für die Simplexbisektion. Dies ist nicht nur für hochdimensionale Anwendungen von großer Bedeutung, sondern auch schon sehr vorteilhaft in zwei und drei Dimensionen.

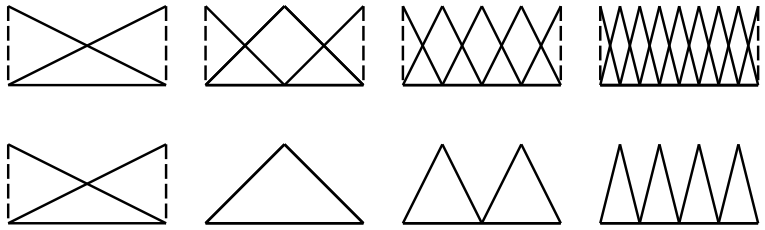


Abbildung 3.1: Skalierungsfunktionenen (obere Reihe) und Waveletbasisfunktionen (untere Rehe) bei Intervallbisektion und linearer Interpolation.

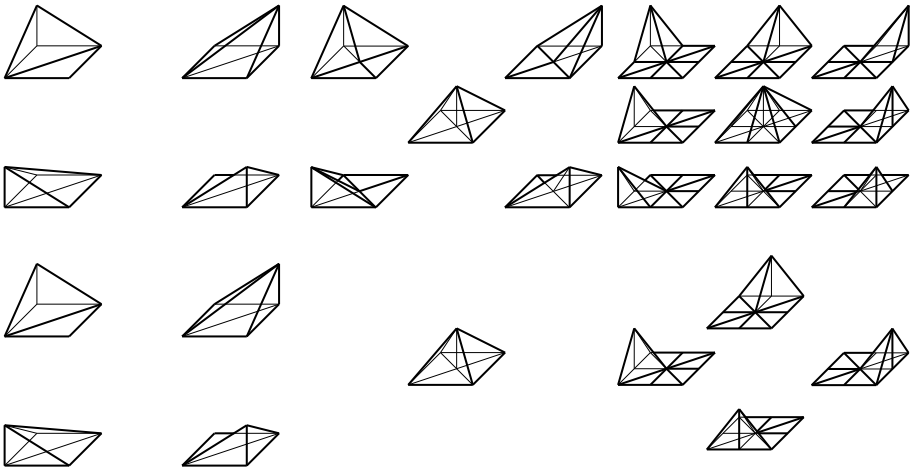


Abbildung 3.2: Skalierungsfunktionenen (obere Reihe) und Waveletbasisfunktionen (untere Rehe) bei Dreiecksbisektion und linearer Interpolation.

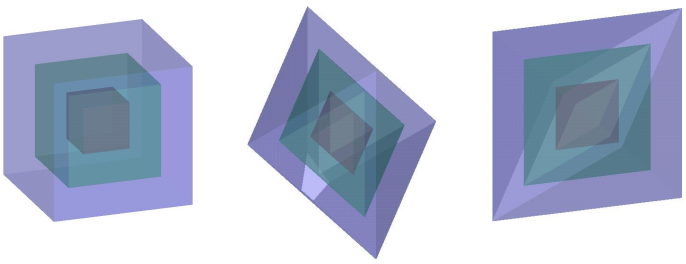


Abbildung 3.3: Die Typen von Skalierungsfunktionen und Waveletbasisfunktionen bei Tetraederbisektion und linearer Interpolation.

.

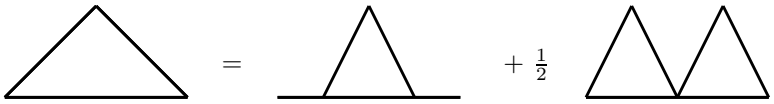


Abbildung 3.4: Zwei–Skalen Relation für die Skalierungsfunktionen im eindimensionalen Fall.

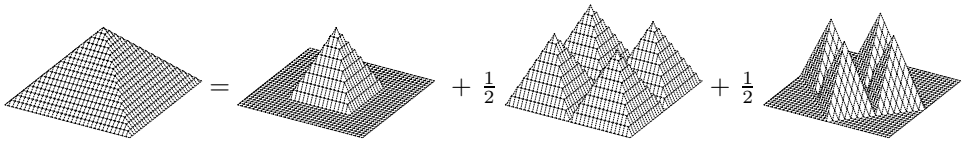


Abbildung 3.5: Zwei–Skalen Relation für die Skalierungsfunktionen im zweidimensionalen Fall.

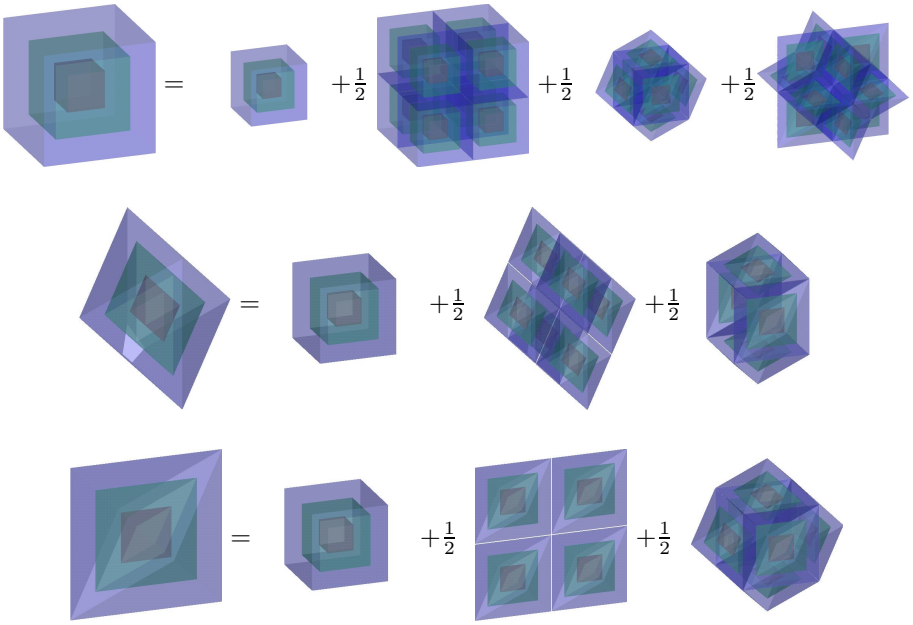


Abbildung 3.6: Zwei–Skalen Relationen für die Skalierungsfunktionen im dreidimensionalen Fall.

.

3.3.2 Lifting

Die stückweise linearen hierarchischen Finite Elemente Basen aus dem letzten Abschnitt haben die Eigenschaft, auch auf den Grobgitterpunkten interpolatorisch zu sein. Dies ist einerseits vorteilhaft (z.B. bei der Visualisierung), andererseits aber auch störend. Nachdem die Punkte auf der gröbsten Zerlegung immer exakt interpoliert werden findet auf diese Weise keine richtige Glättung des Datensatzes statt, wenn die Punkte der gröberen Gitter rein aufgrund eines strukturellen Kriteriums ausgewählt werden.

Die einfachste Wavelet–Konstruktion auf Triangulierungen geht analog zur Haar Konstruktion im Eindimensionalen und basiert auf stückweise konstanter Interpolation [75]. Doch selbst für digitale Bilder ist diese Form der Interpolation wegen der notwendigerweise auftretenden Blockbildung nicht wirklich akzeptabel.

Durch lifting kann die hierarchische Finite Elemente Basis jedoch leicht modifiziert werden. Wir wollen hier nun neben dem Glättungseffekt erreichen, daß die Basisfunktionen verschwindende Momente bekommen. Dies ist in vielen Anwendungen z.B. bei der Datenanalyse wichtig, da mit dem nullten verschwindenden Moment das Integral der einzelnen Wavelet–Basisfunktionen gerade gleich Null ist. Somit bleibt der Mittelwert der Daten bei jeder Approximation erhalten. Zum zweiten wollen wir das Wavelet so symmetrisch wie möglich konstruieren. Dies hat gleichzeitig den Nebeneffekt, daß dann neben dem nullten auch das erste Moment verschwindet. Dies ist allerdings nur im Inneren des Gebiets möglich.

Wir wollen nun den zweidimensionalen Fall betrachten. Durch lifting wird ein bestehendes Wavelet durch Kombination mit Skalierungsfunktionenen gröberer Level modifiziert. Wir wollen dafür nun die vier Skalierungsfunktionen derjenigen Gitterpunkte, die dem Verfeinerungsknoten benachbart sind, hinzuziehen. Hier ist wichtig, daß die Skalierungsfunktionen bezüglich des richtigen Levels gewählt werden. Die Skalierungsfunktionen an den Endpunkten der Verfeinerungskante müssen vom Level $l-2$ gewählt werden während die anderen beiden Skalierungsfunktionen vom Level $l-1$ gewählt werden müssen. Wenn die Lifting–Koeffizienten für alle vier Skalierungsfunktionen gleich $1/12$ gesetzt werden, hat das so entstehende Wavelet Integralwert Null, wie man leicht nachprüfen kann.

Am Rand des Gebiets fehlen allerdings solche Nachbarn und manche benachbarten Skalierungsfunktionen sind am Rand abgeschnitten. An diesen Stellen müssen die Koeffizienten abgeändert werden. Dies ist am leichtesten durch Spiegelung der fehlenden Teile erreicht. Die sich so ergebenden Typen von Wavelets sind in Abb. 3.7 abgebildet. Das gesamte Wavelet–Analyse Programm inklusive Randbehandlung ist in C–Code in Abb. 3.8 dargestellt. Die Wavelet–Synthese wird durch Abarbeitung der doppelten for–Schleifen in umgekehrter Reihenfolge und durch Ersetzung aller $+=$ durch $-=$ und umgekehrt erreicht.

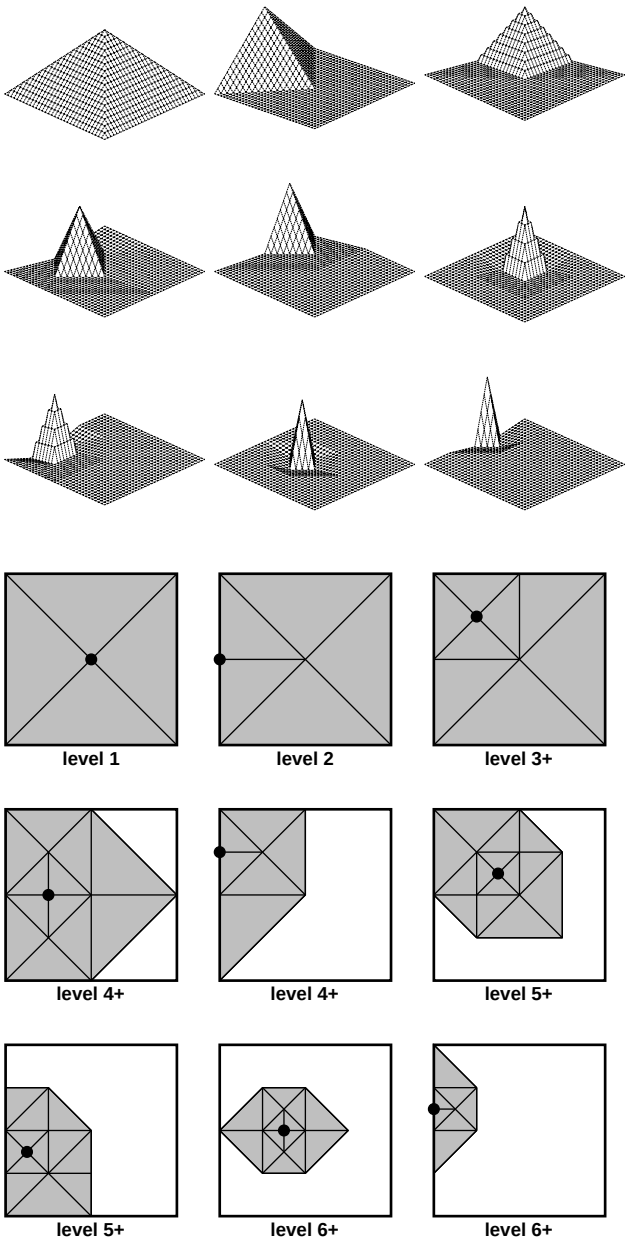


Abbildung 3.7: Die verschiedenen lifting Wavelet–Basisfunktionen in 2D.

```
wavelet_analysis_2D() {
    int i, j, h, l;
    for (l=LEVEL; l>=1; l--) {
        h=1<<(LEVEL-l);

        for (i=h; i<=SIZE-h; i+=2*h)
            for (j=0; j<=SIZE; j+=2*h)
                F[i][j]-=0.5*(F[i-h][j]+F[i+h][j]);

        for (i=0; i<=SIZE; i+=2*h)
            for (j=h; j<=SIZE-h; j+=2*h)
                F[i][j]-=0.5*(F[i][j-h]+F[i][j+h]);

        for (i=h; i<=SIZE-h; i+=2*h)
            for (j=0; j<=SIZE; j+=2*h) {
                double p=1.0/12.0, m=1.0/12.0;
                if (i==h)      m*=2.0;
                if (i==SIZE-h) p*=2.0;
                if (j!=0)      F[i][j-h]+=F[i][j]/12.0;
                if (j!=SIZE)   F[i][j+h]+=F[i][j]/12.0;
                F[i-h][j]+=F[i][j]*m; F[i+h][j]+=F[i][j]*p;
            }

        for (i=0; i<=SIZE; i+=2*h)
            for (j=h; j<=SIZE-h; j+=2*h) {
                double p=1.0/12.0, m=1.0/12.0;
                if (j==h)      m*=2.0;
                if (j==SIZE-h) p*=2.0;
                if (i!=0)      F[i-h][j]+=F[i][j]/12.0;
                if (i!=SIZE)   F[i+h][j]+=F[i][j]/12.0;
                F[i][j-h]+=F[i][j]*m; F[i][j+h]+=F[i][j]*p;
            }

        for (i=h; i<=SIZE-h; i+=2*h)
            for (j=h; j<=SIZE-h; j+=2*h)
                if (((i+j)/(2*h))&1) F[i][j]-=0.5*(F[i-h][j-h]+F[i+h][j+h]);
                else                 F[i][j]-=0.5*(F[i+h][j-h]+F[i-h][j+h]);

        for (i=h; i<=SIZE-h; i+=2*h)
            for (j=h; j<=SIZE-h; j+=2*h) {
                double pp=1.0/12.0, pm=1.0/12.0, mp=1.0/12.0, mm=1.0/12.0;
                if (i==h)      { mm*=2.0; mp*=2.0; }
                if (i==SIZE-h) { pm*=2.0; pp*=2.0; }
                if (j==h)      { mm*=2.0; pm*=2.0; }
                if (j==SIZE-h) { mp*=2.0; pp*=2.0; }
                F[i-h][j-h]+=F[i][j]*mm; F[i+h][j-h]+=F[i][j]*pm;
                F[i-h][j+h]+=F[i][j]*mp; F[i+h][j+h]+=F[i][j]*pp;
            }
    }
}
```

Abbildung 3.8: Eine 2D Wavelet–Analyse basierend auf Bisektion.

3.4 Vergleich der Verfahren

In diesem Kapitel haben wir verschiedene hierarchische Interpolationstechniken basierend auf hierarchischen Finiten Elementen und Wavelets kennengelernt. Hierbei wurden vor allem Verfahren untersucht, die auf allgemeine hierarchische Zerlegungen anwendbar sind und daher nicht auf globaler Tensorprodukt–Bildung basieren. Zum einen wurden die Verfahren bezüglich ihrer theoretischen Eigenschaften miteinander verglichen. Zum anderen wurden einige konkrete Konstruktionen basierend auf stückweise linearer Interpolation für hierarchische Bisektionszerlegungen angegeben.

Der prinzipielle Unterschied zwischen hierarchischen Finiten Elementen (*lazy wavelets*) und den meisten anderen Wavelet–Konstruktionen war dabei die Interpolationseigenschaft an den Grobgitterpunkten. In vielen Anwendungen, z.B. solche die adaptive Verfeinerung erfordern, ist diese Eigenschaft jedoch für die Effizienz der Algorithmen von entscheidender Bedeutung. Wir werden daher im folgenden meist die hierarchische Finite Elemente Basis verwenden. Der Preis für die Interpolationseigenschaft ist allerdings ein fehlender Glättungseffekt an den Grobgitterpunkten, da alle Punkte einer gegebenen Zerlegung immer exakt interpoliert werden.

Kapitel 4

Adaptivität

In diesem Kapitel wollen wir nun die Konstruktion adaptiver, d.h. nicht uniformer Zerlegungen betrachten. Die Grundidee dabei ist, daß durch selektive Verfeinerung eine effizientere Darstellung eines gegebenen Datensatzes als durch uniforme Verfeinerung erreicht wird. Auf diese Weise können z.B. flache Teilbereiche eines Geländemodells durch größere Dreiecke approximiert werden während bergige Bereiche durch kleinere Dreiecke genauer dargestellt werden. Besonders in interaktiven Anwendungen ist eine optimale Darstellung der Daten wegen der begrenzten Ressourcen von zentraler Bedeutung.

Die Grundlage für adaptive Verfeinerung werden sogenannte *Fehlerindikatoren* $\eta(S)$ sein, welche angeben, welcher Fehler gemacht wird, wenn das Teilgebiet S nicht verfeinert wird. Auf diese Weise entsteht dann eine adaptive hierarchische Zerlegung durch Selektion aller Teilgebiete S, für die

$$\eta(S) < \varepsilon$$

gilt, d.h. es werden alle Teilgebiete selektiert, deren Fehler kleiner als ein von Benutzer vorgegebener Schwellwert ε ist. Man kann zeigen, daß diese Form der Approxmation im Rahmen der *best–n–term* Approximation unter bestimmten Bedingungen nur um einen konstanten Faktor schlechter als die bestmögliche ist [25].

Als weiteres Verfeinerungskriterium wird der geometrische Abstand zu einem bestimmten Referenzpunkt $\mathbf{x}_r$, d.h.

$$dist(S, \mathbf{x}_r)$$

dienen. Der Referenzpunkt kann z.B. der Standpunkt des Beobachters [92], sein Blickpunkt und Blickrichtung [96] oder das Zentrum einer Lupe sein, mit der ein Datensatz betrachtet wird [13, 19, 106].

In beiden Fällen ist es notwendig, sogenannte hängende Knoten zu verhinden, die auftreten, wenn zwei benachbarte Teilgebiete nicht konform verfeinert werden. Hierfür gibt es je nach Gitterhierarchie und Interpolation verschiedene Lösungen:

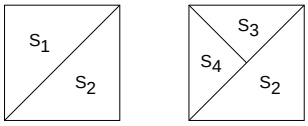


Abbildung 4.1: Zulässige und unzulässige Triangulierung.

- konforme Abschlüsse von Nachbargebieten (z.B. bei roter Dreiecks bzw. Tetraederverfeinerung durch grüne und blaue Verfeinerung [4, 19, 70]),
- rekursives Splitting von Nachbarn (z.B. oft bei Bisektionsverfeinerungen [2, 37, 85, 92, 107]),
- Projektion der hängenden Knoten (z.B. durch konforme Interpolation [106, 105]),
- Nebenbedingungen (bei Delaunay Triangulierungen oder irregulären Dreiecksgittern [28, 80]),
- einfaches Auffüllen entstehender Löcher (nur zur Visualisierungszwecken)

Wir wollen in diesem Kapitel als erstes die Saturierungseigenschaft als weitere Möglichkeit zur Verhinderung hängender Knoten behandeln. Im folgenden werden wir allgemein auf verschiedene Arten zur Definition von Fehlerindikatoren eingehen. Der darauffolgende Abschnitt betrachtet dann hierarchische Fehlerindikatoren und deren Berechnung. Verschiedene Saturationstechniken zur Erfüllung der Saturierungseigenschaft werden daraufhin erläutert und es wird auf deren besondere Eigenschaften hingewiesen. Das Kapitel schließt mit einem qualitativen und quantitativen Vergleich der besprochenen Fehlerindikatoren.

4.1 Saturierungseigenschaft

Falls durch die Schwellwertbildung ein Teilgebiet verfeinert wird, aber ein bezüglich der Verfeinerungskante benachbartes Teilgebiet nicht, dann entsteht an dem Verfeinerungsknoten ein sogenannter hängender Knoten (Abb. 4.1) wodurch die Zerlegung unzulässig wird. Dies führt zu Löchern bei der Darstellung der Daten oder der entsprechenden Isoflächen. Daher muß sichergestellt werden, daß wenn ein Teilgebiet verfeinert wird, auch alle Teilgebiete, die die Verfeinerungskante gemeinsam haben, verfeinert werden.

Dies kann durch das Aufsetzen der Fehlerindikatoren an den Verfeinerungsknoten, d.h.

$$\eta(S) = \eta(\mathbf{x}_{ref}(S))$$

erreicht werden. Auf diese Weise erhalten alle Teilgebiete, die den Verfeinerungsknoten gemeinsam haben, den gleichen Fehlerindikatorwert. Nun kann es aber trotzdem passieren, daß obwohl $\eta(S) < \varepsilon$ gilt, für einen Nachfolger S' von S, dessen Verfeinerungsknoten auf dem Rand von S liegt, $\eta(S') > \varepsilon$ ist. Auf diese Weise könnte ein adjazentes Teilgebiet verfeinert werden müssen und wiederum würde ein hängender Knoten entstehen. Falls aber der Fehlerindikator

die *Saturierungseigenschaft*

$$\eta(S) \geq \eta(S_i)$$

für alle Teilgebiete $S_i \in \mathcal{C}(S)$, die durch Verfeinerung von S entstehen, erfüllt, dann können keine hängenden Knoten für alle Werte von ε auftreten [106, 151].

Wie ein Fehlerindikator, der nicht die Saturierungseigenschaft erfüllt, angepaßt werden kann wird in Abschnitt 4.4 besprochen nachdem wir Fehlermetriken und entsprechende Fehlerindikatoren kennengelernt haben. Der große Vorteil dieses Verfahrens zur Verhinderung hängender Knoten ist, daß während des Baumdurchlaufes an einem Teilgebiet unabhängig von dessen Nachbarn entschieden werden kann, ob es verfeinert werden soll oder nicht. Auf diese Weise sind auch keine aufwendigen (und Speicherplatz verbrauchenden) Datenstrukturen, wie bei anderen Verfahren, notwendig, um effizient auf Nachbarn zugreifen zu können.

4.2 Fehlermetriken

Grundlage für die Konstruktion eines Fehlerindikators ist ein geeignetes Fehlermaß. In diesem Abschnitt wollen wir nun verschiedene oft verwendete Fehlermetriken betrachten. Für einen Überblick an Techniken zur Fehlerschätzung siehe [140].

Wir wollen zunächst annehmen, daß es eine Norm $\|\cdot\|$ gibt, die den geometrischen Approximationsfehler mißt. Es gibt eine Menge von Normen, die üblicherweise zur Berechnung von Fehlern verwendet werden. Populäre Beispiele sind Integralnormen, wie die Lebesgue Normen L_p. Für $p = \infty$ entspricht diese Norm dem maximalen vertikalen Abstand zwischen der Approximation und dem Original. Für $p = 1$ dem Integral der absoluten Differenz. Andere geometrische Normen können Ableitungen (wie z.B. Sobolev Normen), diskrete Krümmungsmaße oder Haussdorff–Maße enthalten.

Wir wollen nun die Approximation von f bezüglich einer gröberen Zerlegung mit f_ε bezeichnen. Demnach ist der Approximationsfehler durch $f_\varepsilon(\mathbf{x}) - f(\mathbf{x})$ gegeben. Die globale Norm des Approximationsfehlers kann durch

$$\|f_\varepsilon(\mathbf{x}) - f(\mathbf{x})\| = \left\| \sum_{l,i:\bar{\eta}(\mathbf{x}_{ref})<\varepsilon} c_{li} \cdot \psi_{li}(\mathbf{x}) \right\| \leq \sum_{l,i:\bar{\eta}(\mathbf{x}_{ref})<\varepsilon} |c_{li}| \cdot \|\psi_{li}(\mathbf{x})\|.$$

berechnet, bzw. abgeschätzt werden. Der Fehlerindikator η reflektiert nun die lokale Anwendung dieser Norm, d.h. die Restriktion der Norm auf den Träger der Basisfunktion.

Sei $\eta^*(\mathbf{x})$ ein Maß auf Punkten $\mathbf{x}$ welches den Effekt der lokalen Approximation bereits auf dem Teilgebiet S (mit $\mathbf{x} = \mathbf{x}_{ref}(S)$) anstatt bis zum lokal feinstem Gitterlevel durchzulaufen, mißt. Weiterhin sei $supp(\mathbf{x})$ der Träger der stückweise linearen Basisfunktion die zum Punkt $\mathbf{x}$ gehört. Wir wollen nun annehmen, daß durch $\eta^*(\mathbf{x})$ der Abstand zwischen $f^{l_{max}}$ und f^l (mit $S \in \mathcal{T}^l$) lokal auf $supp(\mathbf{x})$ bezuglich einer Metrik $d_{supp(\mathbf{x})}$ gemessen wird, d.h.

$$\eta^*(\mathbf{x}) = d_{supp(\mathbf{x})}(f^{l_{max}}, f^l).$$

Wir wollen nun einige weitverbreitete Metriken betrachten:

- Man kann eine lokale Norm der Differenzfunktion verwenden, wie z.B.

$$\eta^*(\mathbf{x}) := \|f^{l_{max}} - f^l\|_{p,supp(\mathbf{x})},$$

  wobei $\|\cdot\|_{p,supp(\mathbf{x})}$ die übliche L_p–Norm für $p \in [1,\infty]$ eingeschränkt auf das Gebiet $supp(\mathbf{x})$ ist. Durch die Hölder–Ungleichung werden die Fehlerindikatoren offensichtlich schärfer für steigende Werte von p.

- Anstatt der Funktionswerte ist es auch möglich Ableitungen zu betrachten, z.B.

$$\eta^*(\mathbf{x}) := \|\nabla f^{l_{max}} - \nabla f^l\|_{p,supp(\mathbf{x})}.$$

  Im allgemeinen wird das resultierende Fehlermaß um eine Ordnung h schärfer als das basierend auf Funktionswerten. Oft wird die Norm des Gradienten als Fehlerindikator verwendet. Dies ist allerdings z.B. zur Visualisierung fragwürdig, nachdem die graphische Darstellung durch Dreiecke ohnehin linear exakt ist. Daher verbessert eine Verfeinerung im Bereichen gleichmäßig großer Gradienten die graphische Darstellung nicht.

- Drittens mag man daran interessiert sein, die geometrische Glattheit der Approximation unabhängig von den wahren Funktionswerten zu betrachten. Ein mögliches Maß hierfür ist die diskrete Krümmung. Für Flächen wird hierfür die absolute Krümmung $\kappa = \sqrt{\kappa_1^2 + \kappa_2^2}$ verwendet, wobei die κ_i die Hauptachsenkrümmungen sind. Wie im Fall von Minimalflächen mit verschwindender mittlerer Krümmung oder Zylindern mit verschwindender Gaußscher Krümmung klar wird, macht ein Fehlermaß basierend auf mittlerer oder Gaußscher Krümmung wenig Sinn.

- Eine vierte Möglichkeit für ein geeignetes Maß hat einen engen Bezug zu geometrischen Körpern [87]. Im einfachsten Fall einer skalaren Funktion f wäre ein geeigneter Ansatz die Graphen von $f^{l_{max}}$ und f^l auf $supp(\mathbf{x})$ zu vergleichen. Falls $\mathrm{dist}(\cdot,\cdot)$ eine geeignete geometrische Metrik auf Graphen ist führt dies zu $\eta^*(\mathbf{x}) := \mathrm{dist}(graph(f^{l_{max}}), graph(f^l))$. Für flache Oberflächen unterscheidet sich dieser Fehlerindikator allerdings nur wenig von denjenigen, die auf der Differenzfunktion basieren.

4.3 Hierarchische Fehlerindikatoren

Üblicherweise ist ein Fehlermaß welches Funktionen auf einem groben Gitter mit Funktionen auf dem feinsten Gitter vergleicht sogar in einem Vorverarbeitungsschritt zu aufwendig auszuwerten. Eine oft verwendete Vereinfachung vergleicht Daten auf dem momentanen Gitterlevel nur mit dem nächstfeineren Level. Dieser *one level look ahead* Fehlerindikator soll mit $\eta(\mathbf{x})$ bezeichnet werden. Allerdings kann es sein daß die Saturierungsbedingung als minimale Vorraussetzung um die Stetigkeit der Darstellung zu garantieren für $\eta(\mathbf{x})$ fehlschlägt.

4.3.1 Hierarchische Offset Fehlerindikatoren

In Analogie zur Norm der Differenzfunktion können wir die hierarchische Offset Funktion f_δ definiert auf einem Teilgebiet als

$$f_\delta|_S = f^l|_S - f^{l-1}|_S$$

Die Werte von f_δ auf $\mathcal{T}^l \setminus \mathcal{T}^{l-1}$ beziehen sich bei Bisektionsverfeinerung auf die Werte der Originaldaten durch die rekursive Formel

$$f^l(\mathbf{x}_{ref}(S)) = \frac{f^l(\mathbf{x}_1) + f^l(\mathbf{x}_2)}{2} + f_\delta(\mathbf{x}_{ref}(S))$$

wobei $\mathbf{x}_1$ und $\mathbf{x}_2$ die Endpunkte der Verfeinerungskante sind. Für glatte Daten, wie $f \in C^2$, ist $|f_\delta(\mathbf{x}_{ref}(S))| = O(d(S)^2)$, mit $d(S)$ dem Durchmesser des Simplex, was die Saturierungseigenschaft asymptotisch auf Gittern $\mathcal{T}^l$ für l genügend groß erfüllt. Daher definieren wir den hierarchischen L_∞ Fehlerindikator durch

$$\eta_\infty(x) := |f_\delta(\mathbf{x})|$$

Anstatt der L_∞–Norm können wir in analoger Weise verschiedene weitere Integralnormen augewandt auf die Differenzfunktion verwenden. Mittels Schwerpunktintegration erhält man z.B. für einen d–dimensionalen Simplex S

$$\eta_p(x) = \frac{1}{d+1} \left(\sum_{S\,,\, x \in \mathcal{T}(S)} |S| \right)^{\frac{1}{p}} |f_\delta(x)|$$

für $1 \leq p < \infty$, wobei $|S|$ das Volumen des Simplex ist. Kleinere p führen zu einem früheren Stoppen beim Durchlauf der hierarchischen Zerlegung auf Simplizes mit geringer Größe.

4.3.2 Gradientenbasierte Fehlerindikatoren

Anstatt den Fehler bezüglich der Funktionswerte zu messen, können wir den Fehler des Funktionsgradienten betrachten:

$$\eta_{1,p}(x) := \begin{cases} \left(\sum_{S \in \mathcal{T}} |T| \right)^{\frac{1}{p}} \|\nabla f_\delta|_S\| & \text{for } \; 1 \leq p < \infty, \\ \max\limits_{S \in \mathcal{T}} \|\nabla f_\delta|_S\| & \text{for } \; p = \infty. \end{cases}$$

Die Auswertung dieser Fehlerindikatoren benötigt allerdings einigen Aufwand. Indem Simplizes durch deren respektive Verfeinerungskanten ersetzt werden, erhält man ohne die Skalierung zu verändern zumindest für $p = \infty$

$$\eta_{1,e} := \frac{2\,|f_\delta(\mathbf{x}_{ref}(S))|}{d(e_{ref}(S))},$$

wobei $d(e_{ref})$ die Länge der Verfeinerungskante e_{ref} ist.

4.3.3 Krümmungsbasierte Fehlerindikatoren

Anstatt der Datenfunktion selbst kann man auch deren Isoflächen betrachten. Der kontinuierliche Krümmungsvektor einer Iso–Hyperfläche kann durch

$$-|\nabla f|\mathrm{div}\left(\frac{\nabla f}{|\nabla f|}\right).$$

ausgewertet werden. Nun betrachten wir ein diskretes Gegenstück. Wir ersetzen die Divergenz durch eine diskrete Differenz und skalieren sie mit der inversen Maschenweite. Dadurch erhält man eine skalierte diskrete Krümmungsgröße die lokal die Qualität der Approximation aus der Sicht der visuellen Erscheinung mißt [106].

Auf jedem Simplex steht der Datengradient ∇f^l immer senkrecht zu einer Isolinie oder Isofläche. Daher mißt auf jeder Fläche G die Normalenkomponente des Sprungs des normalisierten Gradientensprungs, bezeichnet durch $[\frac{\nabla f^l}{|\nabla f^l|}]_G$, lokal den Knick in der Funktion und motiviert die Definition

$$\eta_N(\mathbf{x}_{ref}(S)) := \left[\frac{\nabla f^l}{|\nabla f^l|}\right]_{F_{ref}(S)},$$

wobei F_{ref} die Verfeinerungsebene am Punkt $\mathbf{x}_{ref}$ ist. Hierbei ist der Sprungoperator $[\cdot]_F$ definiert als die Differenz des Arguments auf beiden Seiten der Fläche. Wir können ebenfalls die Vereinfachung des vorigen Indikators anwenden und bezeichnen den resultierenden Indikator mit $\eta_{N,e}$.

4.4 Saturierung

Wie eben schon festgestellt wurde, erfüllen die hierarchischen Fehlerindikatoren im allgemeinen nicht die Saturierungsbedingung. Man kann dieses Problem durch einen modifizierten Fehlerindikator $\bar{\eta}$, der als *minimal saturierter* Fehlerindikator größer oder gleich η definiert ist, beheben. Dieser Vorgang wird *Saturierung* des Fehlerindikators genannt. Die Definition ist konstruktiv in dem Sinne, daß in einem bottom–up, breadth–first Durchlauf der Gitterhierarchie die originalen Fehlerindikatorwerte erhöht werden. In Pseudocode sieht dieser Mechanismus wie folgt aus:

für $(l = l_{\max} - 1\,;\, l \geq 0\,;\, l--)$
  für alle $S \in \mathcal{T}^l$ und $\mathbf{x} = \mathbf{x}_{ref}(S)$
    $\bar{\eta}(\mathbf{x}) = \max\{\max_{S_i \in \mathcal{C}(S)} \bar{\eta}(\mathbf{x}_{ref}(S_i)), \eta(\mathbf{x})\};$

Hierbei sollte betont werden, daß ein depth–first Durchlauf der Hierarchie nicht ausreichen würde.

Alternativ kann eine Saturation eines Fehlerindikators erreicht werden, indem man rekursiv definiert

$$\eta^+(\mathbf{x}) = \eta(\mathbf{x}) + \max_{S_i \in \mathcal{C}(S)} \eta^+(\mathbf{x}_{ref}(S_i)) \qquad \text{für } \mathbf{x} = \mathbf{x}_{ref}(S).$$

Auf dem feinsten Gitterlevel, wo $\mathcal{C}(S) = \emptyset$, sei $\eta^+(x) = \eta(x)$. Dieser Fehlerindikator wird wegen des wesentlich stärkeren Anwachsens der Indikatorwerte auch *recursive blowup* Fehlerindikator genannt. Die verschiedenen Fehlerindikatoren stehen offensichtlich in folgender Beziehung zueinander:

$$\eta \leq \bar{\eta} \leq \eta^+$$

Mit Hilfe der Dreiecksungleichung erhält man zusätzlich auch $\eta^*_\infty \leq \eta^+_\infty$ und $\eta^*_{1,\infty} \leq \eta^+_{1,\infty}$. Der Fehlerindikator η^+, kann, obwohl er der größte und daher schwächste vom Original η^* abgeleitete ist, weitere wünschenswerte Eigenschaften haben. Zum Beispiel ist eine leichte Berechnung von min/max–Werten zur Isoflächenextraktion oder von Kriterien für *multilevel backface culling* (siehe Abschnitt 6.4) möglich.

4.5 Vergleich der Fehlerindikatoren

Bisher wurden qualitative Aspekte der Fehlermessung und Eigenschaften für verschiedene Awendungen diskutiert. Im folgenden wollen wir uns einer quantitativen Diskussion zuwenden. Dafür sollen einige Testprobleme in 2D sowie in 3D untersucht werden.

In 2D betrachten wir zwei Beispiele aus unterschiedlichen Klassen von Datensätzen. Zum einen wurde ein typischer Meßdatensatz, der eine geographische Karte darstellt, ausgewählt. Der Datensatz hat eine Auflösung von 257^2 Punkten welche durch ein hierarchisches Dreiecksgitter trianguliert wurden (Abb. A.1(b)). Er beinhaltet sowohl Regionen mit großer Rauhigkeit als auch Gebiete, die fast planar sind. Auf der anderen Seite wollen wir die Techniken auf einen typischen numerischen Datensatz anwenden, welcher bereits auf einem hierarchischen Dreiecksgitter berechnet wurde. Dieser Datensatz ist charakterisiert durch glatte Gebiete, die sich mit schmalen Übergangszonen wo die Datenfunktion sehr steil ist, abwechseln. Es ist ein Zeitschritt einer Cahn–Hilliard Simulation, ebenfalls auf einem regulären 257^2 Gitter definiert (Abb. A.1(c)). Er stellt die Dichte einer Metalllegierung nach schneller Kühlung dar, was zu Phasentrennung führt. Alles in allem ist der numerische Datensatz wesentlich glatter als die geographische Karte.

Im 3D Fall betrachten wir Isoflächen und farbschattierte Schnitte des 129^3 Buckyball Datensatzes[1] (Abb. A.1(f) und A.1(d)). Er stellt die Elektronendichte um ein C60 Kohlenstoff–Molekül dar. Die Berechnung der Isoflächen und Schnitte erfolgt dabei mit den Techniken aus Kapitel 6.3 und 6.4.

Nachdem die Fehlerindkatoren verschiedene Wertebereiche haben, wird eine Kompatibilität dadurch gesichert, indem die maximalen Indikatorwerte auf 1 normalisiert werden. Die Kosten der Darstellung werden hauptsächlich von der Anzahl der besuchten Gitterzellen im rekursiven Baumdurchlauf bestimmt. Ein alternatives Maß für die Kosten wäre die Zahl der ausgegebenen Primitive. Beide Maße unterscheiden sich üblicherweise nur um eine Konstante und daher ist es an sich gleich, welches man wählt. Die folgenden Ergebnisse basieren auf der Zahl der besuchten Gitterzellen.

[1] Copyright von AVS

Ein kritische Maß für die Qualität der Ergebnisse wäre der z.B. visuelle Eindruck der gezeichneten Bilder. Dies ist allerdings sehr schwer, wenn nicht unmöglich, zu quantifizieren. Um eine vergleichbare Vorstellung der Qualität zu bekommen wählen wir den Kehrwert der gobalen Norm der Differenz zwischen der adaptiv extrahierten Funktion und dem Original. In diesem Kontext ist die Effizienz E eines Fehlerindikators der Quotient aus Qualität und Kosten und damit

$$E_\eta(f, \varepsilon) = \frac{1}{k \cdot \|f_\eta - f\|}$$

wobei k die Zahl der besuchten Gitterzellen für die Approximation f_η ist.

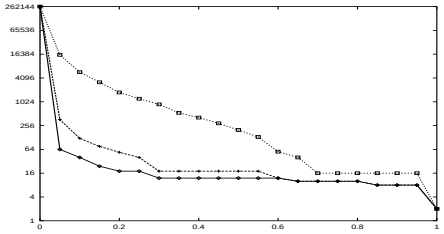
Abb. 4.2 und 4.3 vergleichen die Ergebnisse für die verschiedenen Klassen von Fehlerindikatoren. Die Skalierung auf den y–Axen ist jeweils logarithmisch. Für den geographischen Datensatz sind die verschiedenen Charakteristiken der hierarchischen Offset Fehlerindikatoren im Vergleich zu gradienten–basierten Indikatoren klar sichtbar. Die gradienten–basierten Indikatoren zeigen hier vor allem eine wesentlich bessere Skalierbarkeit. Der glattere numerische Datensatz zeigt ein ähnliches, jedoch weniger ausgeprägtes Verhalten.

Die Graphen für $\bar{\eta}_N$ und $\bar{\eta}_{N,e}$ sind vor allem für den geographischen Datensatz fast identisch. Daher scheint die Vereinfachung von $\bar{\eta}_{N,e}$ zulässig und macht ihn für praktische Zwecke besser geeignet als $\bar{\eta}_N$, nachdem $\bar{\eta}_{N,e}$ leichter zu berechnen ist. Weiterhin verhalten sich die Indikatoren $\bar{\eta}_1$ und $\bar{\eta}_2$ im Vergleich zu $\bar{\eta}_\infty$ sehr ähnlich.

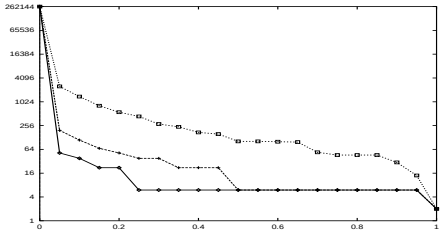
Die Effizienz dieser Indikatoren ist in Abb. 4.4 abgebildet. Es wird klar, daß $\bar{\eta}_\infty$ weniger effizient als $\bar{\eta}_1$ und $\bar{\eta}_2$ ist. Im Falle des geographischen Datensatzes und für nicht zu hohe Schwellwerte im Falle des numerischen Datensatzes unterscheidet sich die Qualität der drei Indikatoren nur wenig. Daher ist der Hauptgrund für die geringe Effizienz von $\bar{\eta}_\infty$, daß für große Schwellwerte eine zu große Zahl von Gitterzellen besucht wird. Im 3D–Fall sind die Ergbnisse ähnlich.

Schließlich vergleichen wir die verschiedenen Methoden zur Sicherung der Saturierungseigenschaft für den η_∞ Indikator. In unseren Experimenten sind die Unterschiede zwischen dem geographischen und numerischen Datensatz klar in der Charakteristik von $\bar{\eta}_\infty$ sichtbar. Allerings verschwinden diese Unterschiede für η_∞^+. Dies ist ebenfalls wahr für andere Indikatoren, wie z.B. für η_2^+ im Vergleich zu $\bar{\eta}_2$. Daher ist die Anwendung eines Indikators vom η^+–Typ nur dann sinnvoll, falls die Vorteile bezüglich der min/max–Schranken oder *multilevel backface culling* ausgenutzt werden.

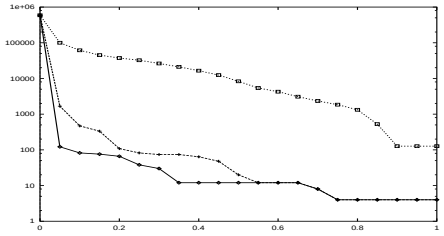
Wir wollen ebenfalls zeigen, daß für sinnvolle Schwellwerte der visuelle Unterschied zwischen den Originalen und den Approximationen sehr gering ist. (Abb. A.1(a) und A.1(b)). Für die geographische Karte besteht das adaptive Bild aus 13666 Dreiecken wohingegen das Original eine etwa zehnmal höhere Zahl benötigt (131072 Dreiecke). Weiterhin zeigen wir extrahierte Isoflächen des Buckyball Datensatzes mit 128709 und 590018 Dreiecken (Abb. A.1(e) und A.1(f)), wo auch die guten Approximationseigenschaften sichtbar werden. Für all diese Bilder wurde der $\bar{\eta}_\infty$ Fehlerindikator verwendet.



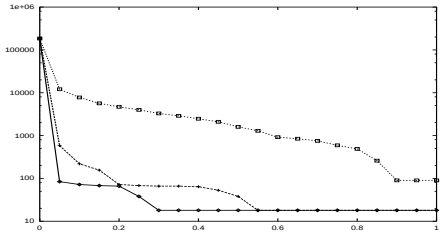
(a) geographische Daten



(b) numerische Daten

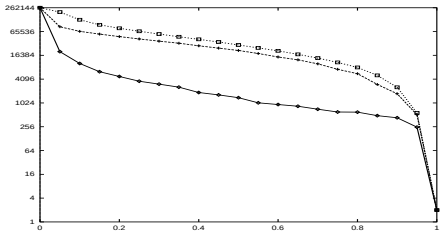


(c) Isofläche

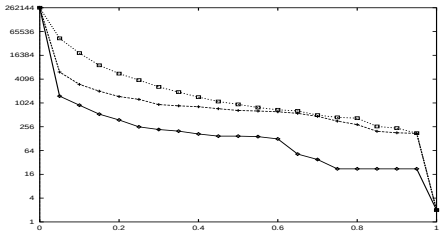


(d) Schnitt

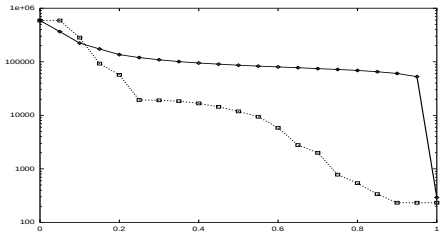
Abbildung 4.2: Vergleich der Fehlerindikatoren $\bar{\eta}_1$ (- - -), $\bar{\eta}_2$ (—) und $\bar{\eta}_\infty$ ($\cdots$) basierend auf den verschiedenen lokalen L_1, L_2 und L_∞ Normen des hierarchischen Offsets bezüglich der Zahl der besuchten Gitterzellen für verschiedene Schwellwerte.



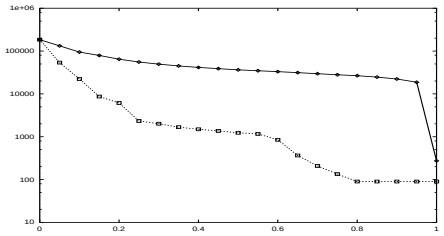
(a) geographische Daten



(b) numerische Daten

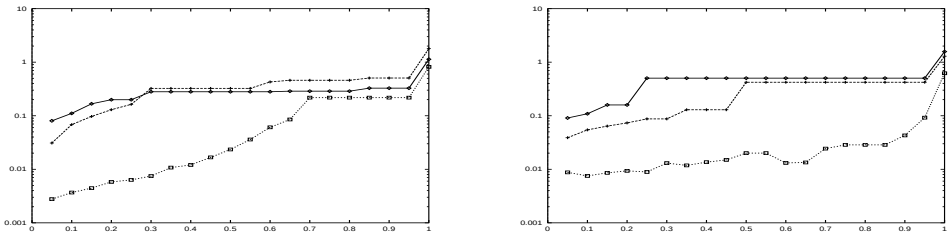


(c) Isofläche



(d) Schnitt

Abbildung 4.3: Vergleich der besuchten Gitterzellen für die Fehlerindikatoren $\bar{\eta}_{1,\infty}$(—), $\bar{\eta}_N(\cdots)$, und $\bar{\eta}_{N,e}$(- - -, nur 2D).



(a) geographische Daten

(b) numerische Daten

Abbildung 4.4: Effizienz als Funktion des Schwellwertes für $\bar{\eta}_\infty$ ($\cdots$), $\bar{\eta}_1$ (—) und $\bar{\eta}_2$ (- - -).

Kapitel 5

Speicherung und Kompression

In den bisherigen Kapiteln haben wir gesehen, wie ein gegebenes Grundgebiet hierarchisch zerlegt werden kann, wie gegebene Datenwerte hierarchisch interpoliert werden können und wie mit Hilfe von adaptiver Verfeinerung effiziente Approximationen des Datensatzes konstruiert werden können. Eine Datenstruktur, die zusammen mit den zugehörigen Verfahren solche hierarchisch adaptive Approximationen erlaubt, nennt man *Multiskalenmodell*.

Bisher war der Fokus mehr auf die verschiedenen Algorithmen und weniger auf die zugrundeliegenden Datenstrukturen gelegt. In diesem Kapitel wollen wir uns nun mit solchen Datenstrukturen zur effizienten Speicherung und Kompression von Multiskalenmodellen beschäftigen. Kompression und effiziente Speicherung ist besonders bei sehr großen Datenmengen, wie wir sie betrachten, von zentraler Bedeutung, da ansonsten die Vorteile der schnellen Approximationsverfahren durch den langsamen Datentransfer zwischen Haupt- und Hintergrundspeicher wieder verloren gehen würden. Die Herausforderung bei der Konstruktion von Kompressionsverfahren liegt dabei darin, daß alle Verfahren auf den komprimierten Daten arbeiten können, d.h. daß keine separate (z.B. partielle) Dekompression notwendig ist.

In diesem Kapitel werden wir Multiskalenmodelle zunächst bezüglich ihres prinzipiellen Speicherbedarfs klassifizieren. Daraufhin werden wir konkret Verfahren zur Speicherung adaptiv hierarchischer Modelle basierend auf Bisektionszerlegungen betrachten. Wir werden zeigen, daß die Kodierung der entstehenden Baumstrukturen sehr effektiv mit Bitcodes und mit Hilfe von raumfüllenden Kurven geschehen kann. Der dritte Teil beschäftigt sich dann mit der Kompression großer Datensätze mit Hilfe dieser Bitcodes und relativen Sprungzeigern. Das Kapitel schließt mit einem Anwendungsbeispiel.

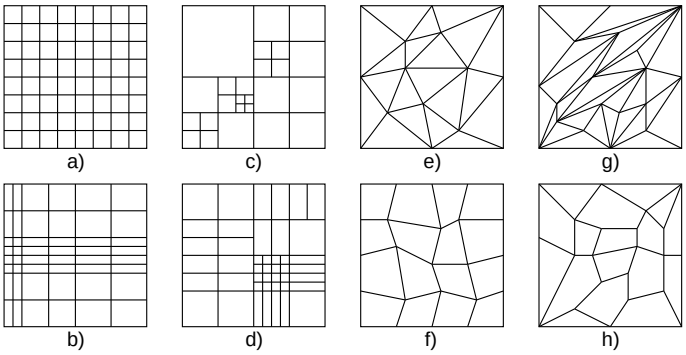


Abbildung 5.1: Die verschiedenen Klassen von räumlichen Zerlegungen.

5.1 Speicherkomplexitäten

Wir wir in den Kapiteln 2 und 3 gesehen haben, gibt es viele Möglichkeiten ein Multiskalenmodell für einen Datensatz zu definieren. Wir wollen nun hier die verschiedenen Modelle bezüglich ihrer Speicherkomplexität klassifizieren.

Ein Multiskalenmodell sei definiert als eine Menge Punkte $(\mathbf{x}_i, y_i), 1 \leq i \leq N$, sowie Interpolationsvorschriften um Werte zwischen den Punkten zu ermitteln. Eine Interpolationsvorschrift beinhaltet üblicherweise wie (z.B. ein polynomialer Grad) und von wo (z.B. eine lokale Gittertopologie) Daten interpoliert werden. Die Datenwerte y_i sind immer fest vorgegeben und müssen für jeden Datensatz in irgendeiner Form gespeichert werden. In diesem Kontext können Multiskalenmodelle grob danach klassifiziert werden wieviel zusätzlicher Speicher für die Koordinaten der Punkte $\mathbf{x}_i$ und für die Interpolationsvorschriften benötigt wird. Falls der zusätzlicher Speicherbedarf von der Ordnung $O(N)$ ist, wird die entsprechende Zusatzinformation explizit genannt. Falls die Ordnung niedriger (z.B. $O(1), O(\log N) oder O(\sqrt{N})$) ist, wird sie implizit genannt, nachdem der zusätzliche Speicherbedarf keinen praktischen Einfluß auf die Gesamtgröße hat. Dies führt zu den folgenden vier Klassen von Multiskalenmodellen mit wachsender Allgemeinheit, Speicherbedarf und Approximationsgüte.

- Koordinaten und Interpolation implizit: reguläre Gitter (Abb. 5.1a), gradierte Gitter (Abb. 5.1b),
- Koordinaten implizit, Interpolation explizit: Quadtrees (Abb. 5.1c), adaptive Tensorprodukt Gitter (Abb. 5.1d),
- Koordinaten explizit, Interpolation implizit: TINs, Delaunay Triangulierungen (Abb. 5.1e), relozierte reguläre Gitter (Abb. 5.1f),
- Koordinaten und Interpolation explizit: datenabhängige Triangulierungen (Abb. 5.1g), allgemeine Tesselierungen (Abb. 5.1h).

Üblicherweise ändern globale Transformationen und Blockstrukturierung die Klasse nicht. Sicherlich erlauben explizitere Modelle größere Flexibilität als weniger explizite (siehe [39, 49, 54]). Allerdings wachsen typischerweise auch

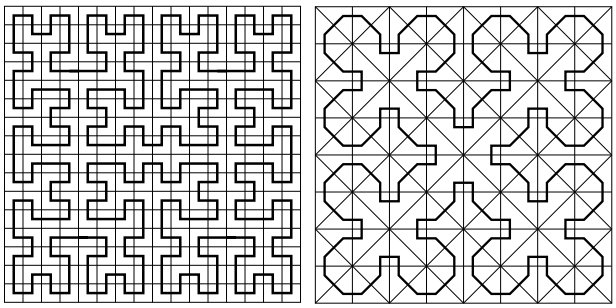


Abbildung 5.2: Approximationen der geschlossenen Hilbert und Sierpinski Kurve.

die Komplexität der zugehörigen Algorithmen und Datenstrukturen. Es ist nicht klar, welches Modell bezüglich des Verhältnisses von Gesamtkosten zu Flexibilität am vorteilhaftesten ist, da dies stark vom Typ der Daten und der Anwendung abhängig ist.

5.2 Speicherung hierarchischer Triangulierungen

Je nach Klasse des Modells kann schon allein die Speicherung mehr oder weniger aufwändig sein. Während für Modelle der ersten Klasse Arrays oder Block–Arrays ausreichend sind, müssen für die anderen Klassen komplexere Speicher- und Zugriffsalgorithmen wie Hashing [66] oder Zeiger-/Element–basierte Konstruktionen [4, 102] verwendet werden. Bei solchen Speicherverfahren ist jedoch oft die Größe des zusätzlichen Speicherbedarfs ein vielfaches der Ausgangsdaten, was diese Verfahren für große Datenmengen nicht mehr anwendbar macht.

Im folgenden werden wir jedoch sehen, daß sich Modelle der zweiten Klasse sehr effizient mit wenigen Bits pro Datenpunkt speichern lassen, wenn auf den Teilgebieten eine raumfüllende Kurve definiert werden kann. Dies wollen wir am Beispiel von Bisektionstriangulierungen zeigen. Hierbei wollen wir zunächst die raumfüllende Sierpinski–Kurve über eine geeignete hierarchische Numerierung der Dreiecke definieren und dann weitere schöne Eigenschaften dieser Numerierung aufzeigen.

5.2.1 Raumfüllende Kurven

Seit ihrer Erfindung Ende des letzten Jahrhunderts haben sich raumfüllende Kurven als ein flexibles Hilfsmittel sowohl in der Analysis als auch in vielen Anwendungen herausgestellt. Neben der Indizierung in räumlichen Datenbanken und der Datenkompression [119] sind dies auch dynamische Lastbalanzierung in parallelen numerischen Algorithmen [66] oder approximative Lösungen des *travelling salesman* Problems [9, 42].

Viele Probleme benötigen die Konstruktion beliebig dimensionaler raumfüllender Kurven. Während sich die Hilbert Kurve [77] (Abb. 5.2 links) auf beliebige

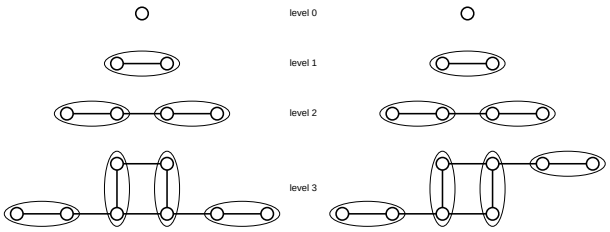


Abbildung 5.3: Adjazenzgraph der Bisektions–Triangulierungen in zwei (links) und höheren (rechts) Dimensionen.

Dimensionen relativ leicht generalisieren läßt, scheinen nützliche Verallgemeinerungen der Sierpinski Kurve [127] (Abb. 5.2 rechts) nur schwer realisierbar. Dies ist an sich recht schade, da die Sierpinski Kurve in zwei Dimensionen viele Vorteile gegenüber der Hilbert Kurve und anderen raumfüllenden Kurven hat [42].

Wir werden nun raumfüllende Kurven über geeignete Aufzählungen hierarchischer Zerlegungen definieren. Dabei werden wir nur Simplizes als Teilgebiete betrachten. Die zweidimensionale Sierpinski Kurve wird dann aus einer ganz bestimmten Aufzählung der Dreiecke in einem hierarchischen Dreiecksgitter basierend auf rekursiver Bisektion entstehen.

Eine Triangulierung wird *sortiert* genannt, wenn die Simplizes der Triangulierung hintereinander aufzählbar sind. Hier wollen wir verlangen, daß zwei aufeinander folgende d–Simplizes S_i und S_{i+1} einen gemeinsamen $d-1$ Teilsimplex haben. Wir sagen $S_i \prec S_j$, falls S_i ein Vorgänger von S_j ist (also $i < j$). Eine hierarchische Triangulierung wird *hierarchisch sortiert* genannt, falls alle Triangulierungen $T_k, k \geq 0$ sortiert sind, und falls für je zwei S_i und $S_j \in T_{k+1}$ mit $S_i \prec S_j$ für deren Väter $P(S_i) \preceq P(S_j)$ gilt.

Durch Wahl eines Punkts im Inneren eines jeden Simplex und durch Verbindung der Punkte aller aufeinanderfolgender Simplizes in einer sortierten Triangulierung wird eine sich selbst nicht schneidende Kurve $C(S)$ konstruiert. Das Limit $k \to \infty$ der Kurven $C(S_k)$ in einer sortierten hierarchischen Triangulierung wird *raumfüllende Kurve* genannt und alle Kurven $C(S_k)$ mit endlichem k sind Approximationen dieser raumfüllenden Kurve. Diese Approximationen sind auch raumfüllend in dem Sinne, daß die zugrundeliegende Triangulierung das Grundgebiet überdeckt. Eine raumfüllende Kurve wird *geschlossen* genannt, falls der erste und der letzte Simplex im Gitter einen gemeinsamen $d-1$ Teilsimplex haben. Die raumfüllenden Kurven in Abb. 5.2 sind beide geschlossen.

Eine wichtige Erkenntnis ist nun, daß die Triangulierungshierarchie basierend auf Simplexbisektion nicht hierarchisch sortierbar für $d \geq 3$ ist. Um dies zu zeigen, betrachten wir den Adjazenzgraph der Simplizes in solchen Triangulierungen. Der Graph hat als Knoten die Simplizes und Kanten zwischen denjenigen Simplizes die einen gemeinsamen $d-1$ Teilsimplex haben. Abb. 5.3 zeigt die Adjazenzgraphen für die ersten vier Verfeinerungslevel bei Bisektionstriangulierungen, wobei Geschwister jeweils durch eine Ellipse markiert sind. In drei und mehr Dimensionen existiert keine Tour durch den Graphen nach drei Verfeinerungsschritten, was die Konstruktion einer raumfüllenden Kurve ver-

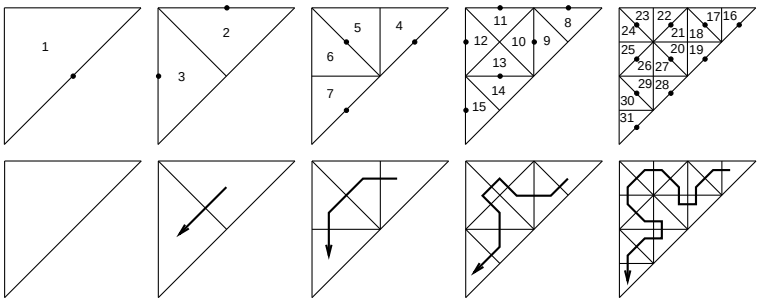


Abbildung 5.4: Numierierung der Dreiecke und entsprechende Sierpinski Kurve.

hindert. Die einzig mögliche Tour im zweidimensionalen Fall entspricht gerade der Sierpinski Kurve.

Interessanterweise wurde bisher keine strikte Verallgemeinerung der Sierpinski Kurve auf höhere Dimensionen gefunden. Die bisher einzige weitere Konstruktion beliebig dimensionaler hierarchischer Simplizialzerlegungen geht auf Freudenthal [52] zurück. Diese Konstruktion startet mit der Zerlegung eines Simplex durch Bisektion der d ausgehenden Kanten eines jeden Knotens. Im zweiten Schritt wird der so bleibende Rumpf (in 3D ein Oktaeder) durch ebene Bisektion unterteilt bis nur noch Simplizes übrig sind. In zwei und drei Dimensionen wird diese Verfeinerungsart auch rote oder reguläre Verfeinerung genannt und wird von vielen Autoren zur Gittergenerierung bei der Diskretisierung partieller Differentialgleichungen verwendet [3, 12]. Diese Konstruktion führt jedoch ebenfalls nicht zu raumfüllenden Kurven, nachdem die $d+1$ Simplizes, die im ersten Schritt abgeschnitten werden, nur einen gemeinsamen $d-1$–Teilsimplex mit dem Inneren gemeinsam haben und daher nur Endpunkte der raumfüllenden Kurve sein können. Nachdem nicht mehr als zwei Endpunkte existieren können, schlägt die Konstruktion simplizialer raumfüllender Kurven schon für $d > 1$ fehl (und damit auch schon im zweidimensionalen Fall).

5.2.2 Numerierung

Im folgenden Abschnitt wollen wir nun die Numerierung der Dreiecke, welche zur Sierpinski Kurve führt, konkret angeben (siehe [75]). Hierbei wollen wir uns auf ein einzelnes Ausgangsdreieck mit Knoten $\mathbf{x}_1 = (0,0), \mathbf{x}_2 = (0,1)$ und $\mathbf{x}_3 = (1,1)$ beschränken. Randknoten können später leicht identifiziert werden und machen keine zusätzlichen Probleme, wenn solche Dreiecke zusammengefügt werden müssen.

Während der Verfeinerungsprozedur wird jedes Dreieck S in ein linkes und ein rechtes Teildreieck relativ zu seinem Verfeinerungsknoten $\mathbf{x}_{ref}(S)$ zerteilt. Nun wollen wir die Dreiecke rekursiv numerieren. Für ein Vaterdreieck mit Nummer n wird sein linkes Teildreieck die Nummer $2n$ und sein rechtes Teildreieck die Nummer $2n+1$ erhalten falls der Level des Vaterdreiecks ungerade ist und umgekehrt falls er gerade ist (Abb. 5.4). Der rekursive Durchlauf des Baums von Dreiecken kann startend mit $l = 0$ und $n = 1$ in Pseudo–Code folgendermaßen skizziert werden:

```
recursive_descent(Koord x1, x2, x3; Level l, Nummer n) {
    Koord xref=(x1+x3)/2;
    falls (est[xref] < eps) extract_triangle(x1, x2, x3);
    falls sonst (level & 1) {
        recursive_descent(x2, xref, x1, l+1, 2*n);
        recursive_descent(x3, xref, x2, l+1, 2*n+1);
    } sonst {
        recursive_descent(x3, xref, x2, l+1, 2*n);
        recursive_descent(x2, xref, x1, l+1, 2*n+1);
    }
}
```

Diese Numerierung entspricht genau dem Pfad der Sierpinski raumfüllenden Kurve [117, 127] (vgl. ebenfalls Abb. 5.4). Dem Pfad der Kurve folgend werden alle Dreiecke auf einem gegebenem Level durchlaufen, wobei je zwei aufeianderfolgende Dreiecke eine gemeinsame Seite haben. Diese Eigenschaft bleibt ebenfalls für adaptive Triangulierungen über verschiedene Level erhalten (siehe Abb. A.2) und kann z.B. die Visualisierung von Dreiecksgittern beschleunigen, indem gemeinsame Knoten nicht mehrfach von der Graphik verarbeitet werden müssen (siehe Kapitel 6.1). Eine Verallgemeinerung der Konstruktion insbesondere für Gebiete mit Löchern findet sich in [141]

Im folgenden zeigen wir zunächst ein paar Grundeigenschaften von numerierten Binärbäumen, die zu solchen adaptiven Triangulierungen gehören, und erläutern, wie die Baumstruktur basierend auf diesen Eigenschaften effizient kodiert werden kann. Dafür werden wir die *Bitcodes* der Dreiecksnummern benötigen. Der Bitcode ist einfach die Binärdarstellung dieser Nummern. Zum Beispiel ist der Bitcode der Nummer 2 die 10, der Bitcode der 7 die 111 und der Bitcode der 26 die 11010.

5.2.3 Randdreiecke

Zunächst wollen wir diejenigen Dreiecke, deren Verfeinerungsknoten auf dem Rand des Ausgangsdreiecks liegen, identifizieren. Die Identifikation dieser Dreiecke ist wichtig um die doppelte Speicherung von Knoten, die auf den Rändern der Ausgangstriangulierung liegen, zu verhindern. Randknoten fallen in zwei Klassen. Auf geraden Leveln liegen sie auf der Hypotenuse des Ausgangsdreiecks, auf ungeraden Leveln auf den beiden Katheten (Abb. 5.5, links). Dieses Zwei–Level Pattern ist charakteristisch für Bisektionstriangulierungen. Im folgenden wird während einer Induktion immer ein Level übersprungen und entweder durch die geraden oder durch die ungeraden Level gewandert.

Zur Identifikation der Randdreiecke müssen wir also zwei Fälle betrachten. Sei x der Bitcode eines Dreiecks. Im ersten Fall, nämlich falls das Vaterdreieck auf dem Rand liegt, liegen zwei der vier Kinddreiecke zwei Level feiner ebenfalls auf dem Rand. Nachdem die vier Kinddreicke einen sog. Fächer um den Verfeinerungsknoten bilden, sind die Bitcodes dieser Dreiecke durch $x00$ und $x11$ gegeben. Im anderen Fall, falls das Vaterdreieck nicht auf dem Rand liegt, liegt auch kein Kinddreieck auf dem Rand (Abb. 5.5, rechts). Der Induktionsbeweis startet mit dem Dreiecken auf Level 0 und 1 für den geraden und den ungeraden

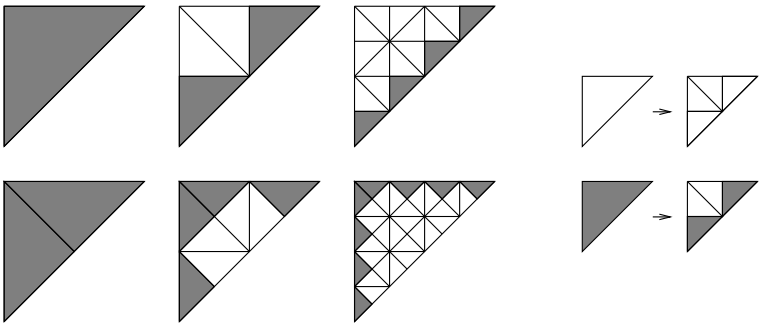


Abbildung 5.5: Identifikationsregel für Randdreiecke.

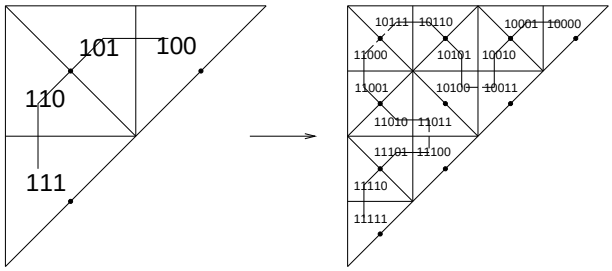


Abbildung 5.6: Identifikationsregel für gemeinsame Verfeinerungsknoten.

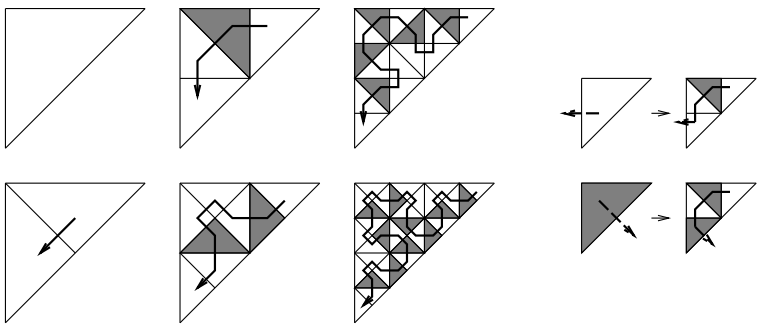


Abbildung 5.7: Identifikationsregel für Auf–Dreiecke.

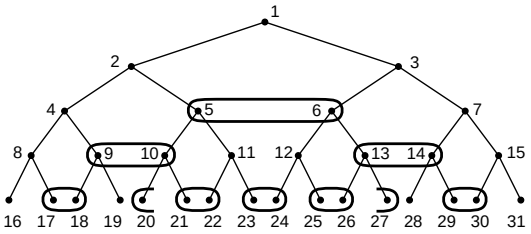


Abbildung 5.8: Binärbaum von Dreiecken mit gemeinsamen Verfeinerungsknoten markiert.

Fall.

5.2.4 Gemeinsame Verfeinerungsknoten

Wir wollen uns daran erinnern, daß jeder Verfeinerungsknoten außer am Rand des Ausgangsdreiecks von zwei Dreiecken geteilt wird und ihm daher zwei Nummern zugeordnet sind. Ausgehend von einer Nummer n des einen Dreiecks kann die jeweils andere Nummer durch den Bitcode von n leicht berechnet werden. Die entsprechende Transformation kann durch ein Hinzufügen des Symbols \$ an das Ende des Bitcodes und durch iterative Anwendung der folgenden angeordneten Menge von Ersetzungsregeln definiert werden:

$$\{\ 00\$ \to \$11,\ 11\$ \to \$00,\ 01\$ \to 10,\ 10\$ \to 01,\ \$ \to .\ \}.$$

Auf diese Weise werden beginnend mit den zwei am wenigsten signifikanten Bits alle Bits in Paaren invertiert bis eine 01 oder 10 auftaucht. Zum Beispiel ist das gegenüberliegende Dreieck der Nummer 13 (1101) die 14 (1110) und das gegenüberliegende Dreieck der Nummer 27 (11011) die 20 (10100) (Abb. 5.8). Kein entsprechender Nachbar wird gefunden, d.h. das Dreieck liegt am Rand, falls die letzte Regel angewandt wird oder falls die zweite oder vierte Regel auf die zwei am weitesten links stehenden Bits des Codes angewandt werden müssen.

Diese Eigenschaft kann wiederum durch Induktion bewiesen werden. In Abb. 5.6 (links) sehen wir, daß in einem Fächer von vier Dreiecken jeweils das zweite und dritte Dreieck einen gemeinsamen Verfeinerungsknoten haben. Für die anderen zwei Dreiecke liegt ihr Verfeinerungsknoten auf dem Rand des Dreiecks. Die in diesem Fall notwendige Transformation addiert 1 zu dem $x01$ Dreieck oder subtrahiert 1 von dem $x10$ Dreieck. Equivalent dazu werden einfach die letzten beiden Bits invertiert.

Nach zwei Verfeinerungsschritten hat sich der Fächer in vier Fächer aufgespalten (Abb. 5.6, rechts). Hier sind zusätzlich zu den Verfeinerungsknoten innerhalb eines jeden der vier Fächer ein Verfeinerungsknoten zwischen Fächer zwei und drei zweien Dreiecken gemeinsam. Das erste Dreieck des zweiten Fächers ($x0111$) liegt neben dem letzten Dreieck des dritten Fächers ($x1000$). Wiederum muß die Transformation nur alle Bits startend von rechts invertieren. Die Inversion stoppt bei einer 01 oder 10.

Diese Transformation kann auch zur Hash–Speicherung (siehe [66, 67]) von adaptiven Triangulierungen verwendet werden. Die Hash–Funktion hat nur sicherzustellen daß identische Verfeinerungsknoten auf die gleiche Adresse abgebildet werden, z.B. durch die Verwendung der kleineren der beiden Nummern. Wenn man eine gleichmäßige Verteilung der Nummern annimmt, ist die mittlere Komplexität dieser Transformation, also die mittlere Zahl der Bits die invertiert werden müssen, unabhängig von der Tiefe des Baumes, sondern limitiert durch eine Konstante. Die Zahl der Bits, die zu invertieren sind ist 2 mit Wahrscheinlichkeit 1/2, 4 mit Wahrscheinlichkeit 1/4, 6 mit Wahrscheinlichkeit 1/8, usw., was in einer mittleren Zahl von $\sum_{l=1}^{l_{max}} 2l \cdot 2^{-l} < 4$ resultiert. Hashing erlaubt effizienten beliebigen Zugriff auf die Daten, benötigt allerdings auch die Speicherung von Hash–Schlüsseln, die für große Triangulierungen mehr Platz als die Daten selbst brauchen können. Für viele Anwendungen wird jedoch kein wirklich beliebiger Zugriff benötigt und daher können effizientere Schemen verwendet werden, wie wir in Abschnitt 5.2.7 sehen werden.

5.2.5 Nachbarn

In manchen Fällen, wie z.B. zur Berechnung von Krümmungen ist es nötig, auf alle Nachbarn eines Dreiecks zugreifen zu können. Hierfür können einerseits rekursive Algorithmen (siehe [45, 100]) verwendet werden. Diese haben jedoch den Nachteil, daß neben vielen Funktionsaufrufen der maximale Level der Triangulierung in die Berechnungskomplexität mit eingeht.

Auf der anderen Seite können alle Nachbarn eines Dreiecks durch einfache Bitoperationen gefunden werden. Zwei der drei Nachbarn eines Dreiecks mit Nummer n sind ohnehin bekannt. Sie liegen auf der raumfüllenden Kurve und haben somit die Nummern $n-1$ bzw. $n+1$. Das dritte fehlende Dreieck liegt nun entweder gegenüber der Hypotenuse oder gegenüber einer der Katheten. Im ersten Fall kann dessen Nummer kann durch die Transformation von Abschnitt 5.2.4 ermittelt werden. Im zweiten Fall kann es als Nachbar eines Randdreicks gesehen werden und durch analoge Bitinversion gefunden werden (siehe Abschnitt 5.2.3).

Im Falle adaptiver Triangulierungen müssen die jeweiligen Nummern, je nachdem ob der Level des Nachbardreiecks größer oder kleiner ist, noch durch zwei dividiert oder mit zwei multipliziert und evtl. eins addiert werden.

5.2.6 Auf- und Ab–Dreiecke

Ein Teilbaum des Binärbaums von Dreiecken ist eine Teilmenge G der Menge von gültigen Dreiecksnummern $\{1, 2, \ldots, 2^{l_{max}}\}$. Eine adaptive Triangulierung kann daher durch die Menge $F \subset G$ aller Blätter des Teilbaumes definiert werden. Die Menge F wird auch die *Front* des Teilbaumes genannt (Abb. 5.9). Falls die adaptive Triangulierung keine hängenden Knoten enthält, hat der zugrundeliegende Binärbaum folgende Eigenschaften:

1. Vollständigkeit:
$$\forall\ n \in G, n > 1 \text{ auch } \lfloor n/2 \rfloor \in G$$

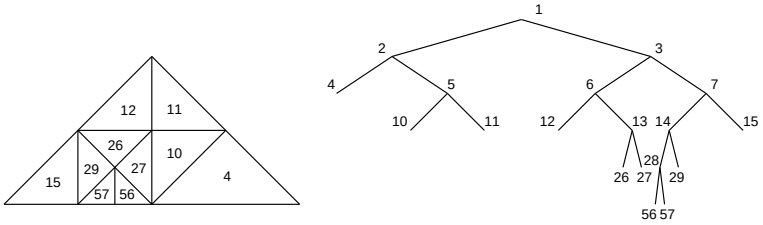


Abbildung 5.9: Adaptive Triangulierung und entsprechender Binärbaum.

2. Zwillinge:

   $\forall$ gerade $n \in G$ auch $n+1 \in G$ und $\forall$ ungerade $n \in G$ auch $n-1 \in G$

3. Balanzierung:

$$\forall\, n \in F \text{ entweder } n+1 \in F \text{ oder } \begin{cases} (n+1)/2 \in F, & n \in (0|1)^*01(11)^* \\ (n+1)\cdot 2 \in F, & n \notin (0|1)^*01(11)^* \end{cases}$$

Während die ersten beiden Eigenschaften recht intuitiv sind, benötigt die dritte Eigenschaft mehr Aufmerksamkeit. Sie besagt, daß man die Front des Baumes von links nach rechts durchwandern kann indem man entweder auf dem gleichen Level bleibt oder einen Level auf- oder absteigt. Für ein gegebenes Dreieck sind allerdings nur zwei der drei Fälle möglich. Die Klasse von Dreiecken, bei denen man entweder auf dem gleichen Level bleiben kann oder einen Level hochgehen kann (*Auf–Dreiecke*) kann ebenfalls durch einen Blick auf den Bitcode ihrer Nummer identifiziert werden: Der Bitcode von Auf–Dreiecken endet mit 01 oder enthält 01 gefolgt von einer geraden Anzahl von Einsen. Die Klasse von Dreiecken, bei denen man entweder auf dem gleichen Level bleiben kann oder einen Level hinuntergehen kann (*Ab–Dreiecke*) sind durch das entsprechede Komplement definiert.

Wir wollen nun auch diese Eigenschaft beweisen. In Abb. 5.7 (links) sind alle Auf–Dreiecke grauschattiert. Das so entstehende Muster kann nicht überraschenderweise wiederum rekusiv definiert werden. Im ersten Fall ist das zweite Dreieck in einem Fächer immer ein Auf–Dreieck. Im zweiten Fall tauchen Auf–Dreiecke auch als viertes Dreieck in einem Fächer auf, aber nur wenn das Vaterdreieck ebenfalls ein Auf–Dreieck ist. Abb. 5.7 (rechts) faßt diese zwei Fälle durch Definition der entsprechenden Erzeugungsregeln zusammen. Diese Erzeugungsregeln können nun wieder in Bitcodes umgewandelt werden. Auf diese Weise lassen sich Auf–Dreiecke dadurch identifizieren, daß ihr Bitcode mit 01 (Fall 1) endet oder mit 11 endet, wobei vorher 01 steht (Fall 2). Zusammen entspricht dies der Sprache $(0|1)^*01(11)^*$.

Nur etwa 1/3 aller Dreiecke sind Auf–Dreiecke. Es ist möglich, Auf- oder Ab–Dreiecke während des Baumdurchlaufs durch den Automat von Abb. 5.10 zu erkennen. Der Automat wird auf den am weitesten links liegenden Zustand initialisiert und ändert seinen Zustand entlang einer der beiden Übergangskanten mit jedem Verfeinerungsschritt (0 für das erste und 1 für das zweite Kind). Falls der Automat in dem doppelt umkreisten Zustand angelangt, ist das momentane Dreieck ein Auf–Dreieck, sonst ist es ein Ab–Dreieck.

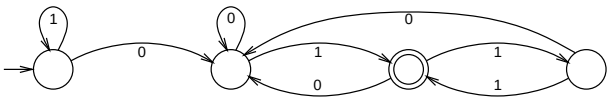


Abbildung 5.10: Automat zur Erkennung von Auf–Dreiecken.

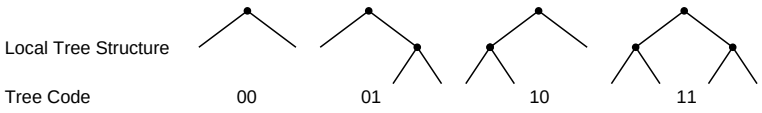


Abbildung 5.11: Lookup–Tabelle für dem Baumcode.

5.2.7 Bit–Codierung

Mit Hilfe der Eigenschaften 1 und 3 ist es nun möglich, eine adaptiv hierarchische Triangulierung mit Hilfe von einem Bit pro Dreieck in der Front des Baumes zu kodieren. Wir wollen ein Bleiben auf dem gleichen Level durch eine 0 und einen Levelwechsel durch eine 1 kodieren. Um korrekt zu starten, ist es weiterhin nötig den Level des ersten Dreiecks in der raumfüllenden Kurve zu speichern. Auf diese Weise können die Nummern der Dreiecke in der Front F sukzessive durch die Nummer des momentanen Dreiecks und des nächsten Bits im Code berechnet werden. Zum Beispiel ist der *Front–Code* für die Triangulierung in Abb. 5.9

$$2\ 100101011.$$

Nachdem die Zahl der Dreiecke in einer adaptiven Triangulierung etwa zweimal die Zahl der Gitterpunkte ist, werden etwa zwei Bits pro Gitterpunkt benötigt, um den Baum zu speichern.

Alternativ kann man die Eigenschaften 1 und 2 verwenden, um die lokale Baumtopologie mit Hilfe der Tabelle von Abb. 5.11 zu kodieren. Für einen Preorder–Durchlauf des Binärbaums ist der *Baum–Code* für die Triangulierung in Abb. 5.9 gegeben durch:

$$11\ 01\ 00\ 11\ 01\ 00\ 10\ 10\ 00.$$

Auf diese Weise werden etwa etwa 4 Bits pro Gitterpunkt benötigt, um die Triangulierung zu speichern.

In beiden Fällen wird viel weniger Speicherplatz zur Speicherung einer gegebenen Triangulierung im Vergleich zu expliziteren Schemen, z.B. Delaunay Triangulierungen, benötigt. Diese Multiskalenmodelle benötigen die Speicherung von Koordinanten, die üblicherweise mehrere Bytes pro Gitterpunkt verbrauchen. Der Front–Code kann zur Speicherung auf Hintergrundmedien oder für Anwendungen, wie Bildkompression, wo immer der gesamte Datensatz verarbeitet wird, verwendet werden. In unseren Anwendungen müssen wir jedoch den Baum–Code verwenden, nachdem in einem adaptiven Baumdurchlauf nur ein sehr kleiner Teil des gesamten Datensatzes zu einem bestimmten Zeitpunkt verarbeitet wird.

5.3 Kompression hierarchischer Triangulierungen

Grundlage eines jeden Kompressionsverfahrens ist die Trennung relevanter und redundanter Daten. Dies geschieht i.d.R. durch eine geeignete Transformation der Daten. Ein Beispiel hierfür ist die Wavelet–Transformation. Diejenigen Wavelet–Koeffizienten, deren Betrag klein ist, werden als unwichtig oder nicht relevant betrachtet und bei der Kompression nicht abgespeichert. Bei der Speicherung gibt es im Prinzip zwei Ansätze. Entweder werden die Wavelet–Koeffizienten einfach zu Null gesetzt, im Hinblick darauf, daß ein darauffolgendes Datenkodierungs–Verfahren (wie z.B. die Huffman–Kodierung) diese Nullen als redundant erkennt und entsprechen effizient kodiert. Alternativ dazu kann ein Strukturkodierungs–Verfahren versuchen, nur diejenigen Teile der Hierarchie, die nicht durch Schwellwertbildung wegfallen, zu erfassen [69, 124]. Der erste Ansatz hat Vorteile, wenn nur wenig Daten irrelevant sind, der zweite, wenn nur wenig Daten relevant sind (für einen Vergleich siehe [123]). Wir wollen hier den zweiten Ansatz wählen, da zum einen bei interaktiven Anwendungen hohe Kompressionsraten benötigt werden und wir zum anderen auf den komprimierten Daten arbeiten müssen.

Zur Kompression wollen wir nun eine adaptive Grundtriangulierung konstruieren. Dies wird dadurch erreicht, daß ein Schwellwert ε_0 gewählt wird und alle Dreiecke S deren Fehlerindikator $\bar{\eta}(\mathbf{x}_{ref}(S)) < \varepsilon_0$ ist, selektiert werden. Die übrigen Dreiecke, bzw. die Datenwerte an ihren Verfeinerungsknoten werden nicht gespeichert, weil die Datenwerte genau oder nahezu (bis auf ε_0) von ihren hierarchischen Nachbarn interpoliert werden können. Falls ε_0 auf 0 gesetzt wird, ist die Kompression verlustfrei, d.h. das Original kann ohne Verlust rekonstuiert werden. Falls ε_0 so gewählt wird, daß der Approximationsfehler kleiner als der Datenfehler der Eingabe ist, spricht man üblicherweise auch von verlustfreier Kompression. Größere ε_0 resultieren allerdings in eine verlustbehaftete Kompression.

In diesem Abschnitt werden wir sehen, wie ein Multiskalenmodell, das über einer solchen Triangulierung definiert ist, effizient komprimiert werden kann und wie die komprimierten Daten verarbeitet werden können. Dies erlaubt die Verarbeitung sehr großer Datenmengen ohne die Daten teilweise von einer Festplatte oder anderen Hintergrundmedien zu dekomprimieren. Die Konzepte sind ähnlich zu zeigerlosen (z.B. sogenannten linearen) Quadtrees [53, 90, 83], allerdings benötigt die Relation Dreiecke – Knoten für Triangulierungen (siehe Abschnitt 2.2.2) spezielle Aufmerksamkeit.

5.3.1 Lineare Speicherung

Wir wollen uns zunächst mit der Speicherung und Verarbeitung der Datenwerte der Grundtriangulierung beschäftigen. Wie wir in Abschnitt 5.2.7 gesehen haben, ist es möglich, die Daten mit Hilfe von zwei Feldern, einem bestehend aus den Datenwerten in der Reihenfolge ihres Auftretens im Baumdurchlauf und einem bestehend aus dem Code des zugrundeliegenden Baums, zu speichern. Nachdem allerdings alle inneren Knoten während des Baumdurchlaufs zweimal

auftauchen, muß man vermeiden, daß Datenwerte doppelt gespeichert werden. Dies kann auf mehrere Arten geschehen.

Offensichtlich müssen die Daten nur bei ihrem ersten Auftreten gespeichert werden. Eine Möglichkeit hierfür wäre es, einfach bereits verarbeitete Daten zu markieren, was allerdings zusätzlichen Speicher benötigt. Eine weitere Lösung wäre die Verwendung der Ersetzungsregeln von Abschnitt 5.2.4 und Daten nur dann zu speichern, wenn die Nummer des momentanen Dreiecks kleiner als die des gegenüberligenden Dreiecks ist. Alternativ können die entsprechenden Knoten bei ihrem zweiten Auftauchen durch die Sprache $(0|1)^*10(00|11)^*$ definiert und durch den Automaten, welche die Sprache parst, erkannt werden.

Während man die so abgelegten Daten durchläuft, ist es nun möglich, Daten, die früher während des Durchlaufs gespeichert wurden, zu erkennen, allerdings ist ihre Position im Speicher nicht bekannt. Eine Möglichkeit, dieses Problem zu lösen ist Daten, die später noch benötigt werden, auf einen Stapel zu legen und sie nach ihrer zweiten Verwendung wieder zu entfernen. Es zeigt sich, daß dies durchaus möglich ist, allerdings werden mehrere Stapel, nämlich einer für jeden Level benötigt. Der Binärbaum von Dreiecken kann nun durchlaufen werden indem die folgenden Aktionen für ein gegebenes Dreieck mit Nummer n auf Level l ausgeführt werden:

$$n \in \begin{cases} (0|1)^*01(00|11)^* & : \text{ lese Datum und lege es auf Stapel } l, \\ (0|1)^*10(00|11)^* & : \text{ entferne Datum von Stapel } l, \\ \text{sonst} & : \text{ lese Datum (Knoten liegt auf Rand).} \end{cases}$$

Natürlich können die drei verschiedenen Fälle auch durch die Transformation von Abschnitt 5.2.4 identifiziert werden.

Eine obere Schranke für die Größe eines Stapels von Level l ist $2^{\lfloor l/2 \rfloor}$, welche für uniforme Triangulierungen angenommen wird. Daher ist der gesamte zusätzliche Speicherbedarf maximal von der Ordnung $O(\sqrt{N})$. Die Speichermenge, die für eine gegebene Triangulierung wirklich für die einzelnen Stapel benötigt wird, kann sehr viel kleiner sein, bis hin zu $O(1)$ für hochgradig adaptive Triangulierungen.

Ein einfaches adaptives pyramidales Multiskalenmodell könnte nun aus mehreren Modellen für verschiedene Schwellwerte $\varepsilon_0, 2\varepsilon_0, 4\varepsilon_0, \ldots$, bis hin zu einem maximalen Schwellwert bestehen. Das Multiskalenmodell kann dann mit Hilfe der Front–Codes der Triangulierungen für die verschiedenen Schwellwerte und der Datenwerte nur für ε_0 gespeichert werden. Für unsere Anwendungen werden wir allerdings dynamische adaptive Verfeinerung brauchen. Die hierfür benötigte zusätzliche Information wird nun im folgenden berechnet.

5.3.2 Sprungzeiger

Im vorigen Abschnitt haben wir gesehen, wie eine adaptive Triangulierung kodiert und komprimiert werden kann, und wie die gesamte Triangulierung durchlaufen werden kann. Es ist allerdings nicht möglich, sie teilweise zu durchlaufen, d.h. einen Teilbaum des adaptiven Binärbaums zu extrahieren. Nachdem die Daten in einem großen Feld angelegt sind, müßte man die Zahl von Daten, die übersprungen werden, wenn der Durchlauf lokal stoppt, wissen.

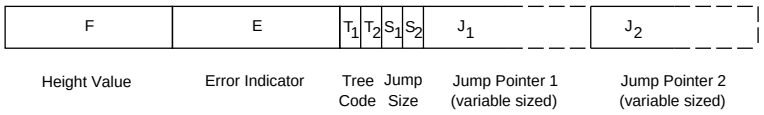


Abbildung 5.12: Datentyp zur Speicherung adaptiv hierarchischer Triangulierungen.

Eine Idee, um dieses Problem effizient zu lösen, ist die zusätzliche Speicherung relativer Sprungzeiger. Für jeden Knoten im Baum ist dies exakt die Zahl der Knoten im Teilbaum darunter. Hierbei wird die teuere Speicherung von absoluten Zeigern, die normalerweise sehr viel mehr Speicher als die Daten selbst benötigen, vermieden. Manche der relativen Zeiger können recht groß werden, z.B. der hierarchisch höchste Zeiger erhält die Zahl $N-1$. An den meisten Knoten sind diese Zahlen aber klein, z.B. auf dem lokal feinsten Level der adaptiven Triangulierung sind sie 0 und auf dem zweitfeinsten maximal 2.

Daher reservieren wir Speicher variabler Größe für die Zeiger und verwenden einen Bitcode für die reservierte Größe. Hier wollen wir zwei Bits verwenden, nämlich 00 für 1 Byte, 01 für 2 Bytes, 10 für 4 Byte und 11 für 8 Bytes. Für einen Baumcode von 00 ist der Sprungzeiger auch 0 und muß daher nicht gespeichert werden. Nachdem wir variable Größen verwendet haben, sind die Sprungzeiger nun nicht die Zahl der Dreiecke im Teilbaum darunter, sonder die Zahl der Bytes, die jeweils belegt wurde. Sie können in einem bottom–up Durchlauf des Baumes berechnet werden. Auf diese Weise ist der mittlere Speicherbedarf für einen Sprungzeiger weniger als ein Byte unabhängig von der Tiefe des Baumes.

Nachdem jeder Knoten zwei Teilbäume hat (wir erinnern uns an die Dualität Dreiecke zu Knoten) führt dies zu dem Datentyp von Abb. 5.12. Die Datenwerte T_1, S_1 and J_1 entsprechen dem ersten Vorkommen des Knotens in der raumfüllenden Kurve, währen die Daten mit Index 2 zum zweiten gehören. Für Randknoten sind die letzeren Einträge leer. Der Baum–Code und die Größe des Sprungzeigers benötigen jeweils 2 Bits für ihre Speicherung. Daher benötigen T_1, T_2, S_1 und S_2 zusammen genau ein Byte. Auf diese Weise werden nur ungefähr 3 Byte pro Knoten benötigt, um die gesamte Baumstruktur zu speichern.

Dieses Schema erlaubt einen sehr schnellen und effizienten adaptiven Baumdurchlauf, nachdem nur einige wenige Bitoperationen notwendig sind, um die gesuchten Datenwerte zu finden.

5.4 Anwendungsbeispiel

Wir wollen nun die Leistung des Kompressionsalgorithmus anhand des *gtopo30*[1] Datensatzes zeigen. Dieser Datensatz besteht aus fein aufgelösten weltweiten Höhendaten (siehe Abb. A.3). Der Datensatz wurde auf zwei 8193×8193 Gitter interpoliert, was in über 134 Millionen Höhenwerten resultiert. Jeder Höhenwert ist in Metern über Meerespiegel angegeben und wird mit 2 Bytes als eine (vorzeichenbehaftete) Ganzzahl (es gibt ja einige Bereiche auf der Erde, die meh-

[1] Courtesy US Geological Survey

ε_0	Punkte	Knoten im Baum	Größe (auf Platte)	Größe (im Speicher)	rel. Fehler
0	134.250.498	268.435.454	302.055.446	809.697.258	0.0%
2	40.726.357	81.433.161	91.631.895	248.751.126	0.004%
4	30.971.462	61.924.693	69.683.546	190.170.314	0.020%
8	23.299.051	46.580.830	52.420.742	144.005.944	0.064%
16	15.578.706	31.141.001	35.050.073	97.012.493	0.220%
32	8.886.835	17.757.981	19.993.453	55.750.190	0.659%
64	4.468.305	8.921.476	10.051.830	28.175.287	1.513%
128	2.150.101	4.285.449	4.835.918	13.605.848	2.500%
256	1.042.534	2.072.666	2.344.186	6.605.564	3.677%
512	404.078	801.818	908.419	2.561.195	5.153%
1024	156.618	309.955	352.015	992.912	7.023%
2048	61.212	120.714	137.548	388.057	9.500%
4096	23.808	46.984	53.489	148.810	12.968%
8192	9.353	18.209	21.017	59.120	18.315%
16384	3.579	6.887	8.055	22.561	26.649%
32768	1.387	2.613	3.136	8.670	40.761%
65535	502	921	1.154	3.113	60.395%

Tabelle 5.1: Kompressionsergebnisse für den *gtopo30* Datensatz.

rere hundert Meter unter dem Meerespiegel liegen) gespeichert. Höhenwerte im Meer sind als *nodata* Werte markiert. Alle Berechnungen wurden auf einer SGI Onyx^2 R10000 (195 MHz) ausgeführt.

Wir verwenden ebenfalls 2 Bytes für den Fehlerindikator $\bar{\eta}$, wodurch es möglich ist bis zu 65535 verschiedene Approximationen mit unterschiedlichen Auflösungen zu extrahieren. Die Indikatorwerte wurden so skaliert, daß die geringst mögliche Auflösung etwa 500 Dreiecke enthält.

Die Kompressionsergebnisse sind in Tabelle 5.1 aufgeführt. Für verschiedene Schwellwerte ε_0 vergleichen wir die Zahl an gespeicherten Höhenwerten, die Zahl der Dreiecke in der entsprechenden Triangulierung, die Zahl der inneren Knoten im Binärbaum, den relativen Approximationsfehler $\|f - f_{\varepsilon_0}\|_1 / \|f\|_1$, und den Speicherbedarf mit Hilfe der Datentypen der vorigen Abschnitte. Für die verschiedenen Stapel wird zusätzlicher temporärer Speicher benötigt, aber die Menge ist vernachlässigbar.

Man sieht, daß sich die Zahl der Punkte in etwa verdoppelt, wenn sich ε_0 verdoppelt. Dies entspricht der asymptotischen Approximationstheorie für Fehlerindikatoren vom L_p–Typ für glatte Funktionen. Die Zahl der inneren Knoten im Baum entspricht dem algorithmischen Zusatzaufwand, der durch den Baumdurchlauf entsteht. Wie erwartet ist diese Zahl maximal doppelt so groß wie die Zahl der Punkte der Triangulierung unabhängig vom Schwellwert.

Der Speicherbedarf ist in den Spalten vier und fünf aufgelistet. Die Speicherung auf Platte wird mit dem Front–Code des Abschnitts 5.2.7 erreicht. Hierbei werden neben den zwei Bytes für die Höhenwerte etwa zwei Bits pro Datum für die Speicherung der Baumstruktur benötigt. Dieser Bitstrom kann in einem Nachbearbeitungsschrit weiter durch ein verlustfreies Kodierungsverfahren, wie

das Lempel–Ziv oder Huffman–Verfahren komprimiert werden. Weitere Experimente bei der Kompression solcher Daten (siehe Abb. A.2) zeigen so einen weiteren Gewinn um bis zu einem Faktor von zwei [123]. Hierbei stellt sich auch heraus, daß eine Ersetzung der Höhenwerte durch die entsprechenden Wavelet–Koeffizienten keinen weiteren Kompressionsgewinn ergibt. Der interne Speicherbedarf mit Hilfe des Datentyps aus Abschnitt 5.3.2 ist in Bytes weniger als 7 mal die Zahl der Punkte. Das heißt, daß weniger als 3 Bytes pro Punkt im Mittel für die komplette Baumstruktur verbraucht werden. Für einen Schwellwert von 0 wäre der Speicherbedarf größer als für ein vollbesetztes zweidimensionales Array (etwa 1/4 Gbyte). Doch schon für einen Schwellwert von 2 ist der gesamte Speicherbedarf weniger als für das Original.

Als Fehlerindikator wird der hierarchischer Offset Fehlerindikator basierend auf der L_1–Norm verwendet. Der relative Approximationsfehler wird demnach in Korrespondenz zum Typ des Fehlerindikators ebenfalls in der L_1–Norm gemessen. Der Approximationsfehler hängt nicht linear von ε_0 ab. Er steigt schneller als ε_0 für kleine Werte und langsamer für große. Für einen relativen Fehler von 5% werden nur etwa 0.3% der Zahl der Punkte benötigt.

Zusammenfassend läßt sich sagen, daß sich das Kompressionsverfahren basierend auf raumfüllenden Kurven, Bitcodes und relativen Sprungzeigen durch gute Kompressionsraten und Skalierbarkeit auszeichnet. Weiterhin erlaubt das Verfahren ein direktes Arbeiten auf den komprimierten Daten für verschiedenste Anwendungsprobleme, wie im nächsten Kapitel gezeigt wird.

Kapitel 6

Anwendungen

Es gibt eine große Fülle von Anwendungsbereichen in denen eine interaktive Verarbeitung großer Datenmengen gefordert ist. Digitale Höheninformationen bilden zum Beispiel die Grundlage von geographischen Informationssystemen (GIS) oder von (Flug-)Navigationssystemen. Dreidimensionale digitale Bilddaten müssen in der medizinischen oder chemischen Bildverarbeitung interaktiv graphisch dargestellt werden. Der größte Wachstumsbereich liegt sicherlich bei Internet–basierten Informationssystemen, wo Unmengen an Daten nicht nur schnell graphisch dargestellt sondern auch effizient über ein langsames und unsicheres Netzwerk transferiert werden müssen.

In diesem Kapitel werden wir nun sehen, wie die Multiskalenverfahren aus den vorhergehenden Kapiteln in solchen Anwendungen eingesetzt werden können. Hierbei liegt ein besonderes Augenmerk auf der schnellen interaktiven Visualisierung der Daten. Der Basisalgorithmus, der allen Visualisierungstechniken gemeinsam ist, hat dabei folgende rekursive Struktur:

```
Betrachte(Teilgebiet) {
  falls (VonInteresse(Teilgebiet)) {
    falls (Fehler(Gebiet) < Schwellwert) Visualisiere(Teilgebiet)
    sonst {
      Betrachte(Unterteilung 1)
      ...
      Betrachte(Unterteilung n)
    }
  }
}
```

Bei gegebener Verfeinerungsstrategie (in den Beispielen wird das die uns mittlerweile wohl bekannte Simplexbisektion sein) unterscheiden sich die verschiedenen Verfahren nur durch die zwei Funktionen `VonInteresse` und `Visualisiere`.

Die in diesem Kapitel betrachteten Beispielprobleme sind die graphische Darstellung digitaler Höhenmodelle, die Erzeugung fraktaler Oberflächen, sowie die Visualisierung dreidimensionaler Bilddaten durch Schnittbildung, Isoflächen–Extraktion oder Volumendarstellung.

6.1 Darstellung digitaler Höhenmodelle

Die Menge an verfügbaren digitalen Höhendaten, wie sie zum Beispiel durch Satellitenmessungen gewonnen werden, hat in den letzten Jahren stark zugenommen. Aus dieser Tatsache sind viele große Anwendungsbereiche, wie die digitale Kartographie oder geographische Interfomationssysteme [84] erst entstanden. Die schieren Menge der verfügbaren Daten erfordert eine sehr effiziente Datenrepräsentation und -kompression [8, 17].

Unsere Hauptproblemstellung ist hier die interaktive Visualisierung von digitalen Höhenmodellen mittels farbschattierter Dreiecke. Allerdings kann die zu visualisierende Datenmenge die Menge an interaktiv darstellbaren Daten leicht um mehrere Größenordnungen übersteigen. Dieses Problem kann durch Multiskalenmodelle gelöst werden, indem man zwei Beobachtungen macht. Zum einen müssen Gebiete, die nicht sichtbar sind, z.B. außerhalb des Blickfensters oder kleiner als ein Bildschirmpixel sind, nicht dargestellt und vor allem auch nicht verarbeitet werden. Zum anderen können weniger interessante (z.B. flache) Gebiete durch größere Dreiecke approximiert werden. Auf diese Weise kann die Gesamtzahl von Dreiecken, die gezeichnet werden muß, drastisch reduziert werden, was eine interaktive Visualisierung an sich erst möglich macht (siehe [7, 45, 81, 92, 107, 109, 113]).

6.1.1 Sphärische Transformation

Terrestrische Höheninformationen leben an sich alle im gleichen Grundgebiet, der Erde. Speziell im Hinblick auf geographische Informationssysteme, in denen solche Datensätze vereinigt, geschnitten, extrahiert oder vergröbert werden müssen, wollen wir ein einheitliches globales Koordinatensystem wählen. Der Einfachheit wollen wir dafür als Grundgebiet die Kugeloberfläche wählen.

Um eine hierarchische Triangulierung für globale topographische Daten zu erzeugen, ist es notwendig, eine Kugel mit Dreiecken zu pflastern. Hierfür verwenden wir die wohlbekannte Transformation von geographischen Koordinaten $(\phi, \psi) \in \Omega = [0, 2\pi] \times [-\pi/2, \pi/2]$ (mit Längengrad ϕ und Breitengrad ψ) in sphärische Koordinaten $(u, v, w) \in \mathbb{R}^3$ durch die Formel

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = (1 + f(\phi, \psi)) \cdot \begin{pmatrix} \cos\phi \cdot \cos\psi \\ -\sin\phi \cdot \cos\psi \\ \sin\psi \end{pmatrix},$$

wobei $f(\phi, \psi)$ der normalisierte Höhenwert an der gegebenen Koordinate ist. Das Rechteck Ω wird in zwei Quadrate mit Seitenlänge π unterteilt. Auf diese Weise besteht die gröbste Triangulierung der Kugel aus vier Dreiecken, wobei jedes Dreieck ein 90 Grad Segment auf der Kugel abdeckt. Aufgrund der Periodizität müssen für die Berechnung des Fehlerindikators Werte bei $\phi = 0$ und $\phi = 2\pi$ natürlich konsistent gehalten werden.

Wir haben geographische Koordinaten trotz des Auftretens von Singularitäten an den Polen aus mehreren Gründen gewählt. Zum einen erlauben sie eine intuitive Handhabung der Datensätze nachdem der Benutzer mit dem Koordinatensystem vertraut ist. Zum zweiten ist die Koordinatentransformation

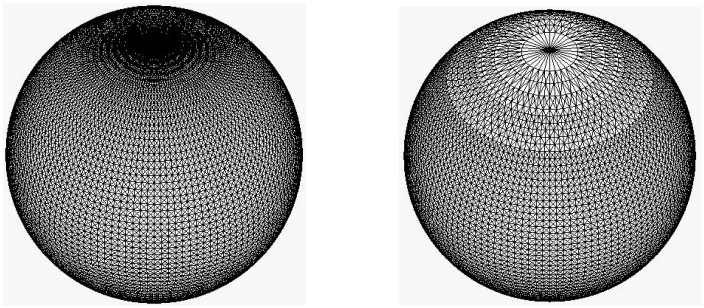


Abbildung 6.1: Kompensation der Polsingularitäten durch den Fehlerindikator.

sehr schnell, besonders wenn die auftretenden Sinus- und Kosinus–Werte vorab berechnet werden. Zum dritten können die Singularitäten an den Polstellen kompensiert werden, indem die Fehlerindikator–Werte (vor der Saturation) mit $\cos\psi$ multipliziert werden. Dieser Skalierungsfaktor stellt das Verhältnis der Flächen des planaren Dreiecks zu dem entsprechenden sphärischen Dreieck in Abhängigkeit von ψ dar. Letztere werden sehr schnell kleiner in Richtung der Pole. Abb. 6.1 (rechts) zeigt diesen Kompensationseffekt (siehe auch [65]).

Natürlich ist es auch möglich hierarchische Triangulierungen für andere Approximationen der Kugel, z.B. basierend auf den fünf Platonischen Körpern [38, 47, 122] oder der Fußball–Tesselierung zu konstruieren. Die einzigen Bedingungen sind eine gültige Anfangstriangulierung und die Übereinstimmung von Verfeinerungskanten bei benachbarten Dreiecken.

6.1.2 Referenz–Schwellwerte

Das Hauptelement des Visualisierungsalgorithmus ist ein top–down Durchlauf durch den Binärbaum von Dreiecken. Der Durchlauf wird lokal gestoppt, falls eine der folgenden Bedingungen zutrifft: falls das momentane Dreieck S vollständig außerhalb des Blickfensters liegt, wird nichts gezeichnet. Ansonsten, falls der Fehlerindikatorwert des momentanen Dreiecks $\bar{\eta}(\mathbf{x}_{ref}(S))$ unterhalb eines weiteren Schwellwertes $\varepsilon_1 \geq \varepsilon_0$ liegt, wird das Dreieck gezeichnet. Ersterer Check kann durch einfaches Koordinaten–Clipping ausgeführt werden.

Der Zeichen–Schwellwert ε_1 könnte vom Benutzer gewählt werden. Dies würde allerdings ein kontinuierliches Anpassen des Schwellwertes erfordern, um die momentane Bildwiederholrate und -qualität halten zu können, besonders während man in oder aus den Datensatz zoomt. Wir wollen daher den Zeichen–Schwellwert dynamisch variierbar halten. Sei der Zoom–Faktor z definiert als das Verhältnis aus der momentanen Skala zur globalen Skala. Wie man leicht sieht, ist dann die korrekte Art und Weise, den Zeichen–Schwellwert zum Zoom–Faktor zu beziehen durch die Formel

$$\varepsilon_1 = \varepsilon_r \cdot z^{-5/4}$$

gegeben. Hierbei ist ε_r ein Referenz–Schwellwert für die globale Skala. Durch eine Änderung von ε_r anstatt ε_1 kann der Benutzer die allgemeine Bildqualität

kontrollieren und die Menge von gezeichneten Dreiecken kann unabhängig von dem momentanen Zoom–Faktor z gehalten werden.

Abb. A.3 zeigt einen Teil des Gesamtdatensatz, der in einer interaktiven Anwendung visualisiert wird. Die Dreiecke sind mit Hilfe einer einfachen geographischen Farbtabelle farbschattiert. In den Bildern sind alle Höhenwerte durch einen Faktor 10 überhöht. Auf diese Weise erreicht man eine Zeichenrate von 500000 Dreiecke pro Sekunde (auf einer Onyx2) unabhängig von der momentanen Position und dem Zoom–Faktor. Bilder moderater Qualität können in einer hohen Bildrate gezeichnet werden, was interaktive Überflüge erlaubt, während Bilder hoher Qualität für eine genauere Ansicht maximal ein paar Sekunden benötigen.

Eine weitere Beschleunigung der Visualisierung kann mit Hilfe sogenannter *T–strips* erreicht werden, da auf diese Weise Punkte, die mehreren Dreiecken gemeinsam sind nicht mehrfach (bzw. weniger oft) von der Graphik verarbeitet werden müssen. Solche T–strips werden automatisch erzeugt, wenn der Durchlauf durch den Dreiecksbaum entlang der raumfüllenden Kurve (Abschnitt 5.2.2) erfolgt (siehe auch [107]).

6.1.3 Hinzufügen von Datensätzen

Durch die Baumstruktur des Multiskalenmodells ist es leicht möglich, zu einem gegebenen globalen Datensatz (wie dem *gtopo30*) lokale Datensätze auf kleineren Skalen hinzuzufügen. Dies wird einfach durch die Konstruktion eines Multiskalenmodells für den neuen Datensatz und durch Ersetzung des entsprechenden Teilbaums im globalen Baum erledigt. Um eine zulässige Triangulierung zu bewahren müssen allerdings möglicherweise Punkte in der Nähe des Randes des neuen Datensatz von den bestehenden Punkten interpoliert und eingefügt werden. Weiterhin müssen möglicherweise die Fehlerindikator–Werte modifiziert werden, damit weiterhin die Saturierungsbedingung erfüllt ist.

Um dieses Vorgehen zu zeigen haben wir zwei ineinander geschachtelte Datensätze zu unserem globalem Modell hinzugefügt. Der erste überdeckt den Niederrhein und seine Umgebung[1] während der zweite die Stadt Bonn[2] darstellt. Höhenwerte sind in diesen Datensätzen in Dezimetern bzw. Zentimetern angegeben. Beide Datensätze wurden jeweils auf ein 8193×8193 Gitter interpoliert, welches das nächst größere dyadische Quadrat in geographischen Koordinaten überdeckt.

Abb. A.4 zeigt, daß die Datensätze, die auf diese Weise in das Modell eingefügt wurden, glatt in das globale Modell hineinpassen. Alle drei Datensätze wurden mit einem relativ kleinen $\varepsilon_0 = 2$ komprimiert, was in einen gesamten Speicherbedarf von weniger als 0.5 GByte resultiert. Für die Bilder wurde der globale Referenz–Schwellwert ε_r auf 1000 gesetzt und nicht weiter verändert. Die Zahl der auf diese Weise gezeichneten Dreiecke sind 83491, 93967, 49323 und 97353, und unterscheiden sich daher maximal um einen Faktor 2. Es wurden allerdings verschedene Farbtabellen für die Bilder verwendet um den visuellen Eindruck zu verbessern.

[1] Copyright von SFB 350, Universität Bonn
[2] Copyright von DLR, Köln–Porz

6.1.4 Fokussierung

Wir wollen nun noch einmal etwas genauer auf die Fehlermessung eingehen. Betrachten wir einmal einen Teil des *gtopo30* Datensatzes, welcher die Nordsee und ihre Umgebung darstellt (Abb. A.5). Die Approximationen basieren hier auf der L_1–Norm. Es ist klar sichtbar, daß in rauhen, bergigen Regionen wie in Norwegen und Schottland mehr Dreiecke verwendet werden müssen um den globalen Fehler beschränkt zu halten, während in relativ flachen Gegeden wie in Norddeutschland und Holland größere Dreiecke verwendet werden können. Allerdings fallen in diesen Gebieten die Approximationen der Küstenlinien nicht sonderlich zufriedenstellend aus.

Manche Wichtigkeitsmaße können nicht einfach durch eine geometrische Norm, wie im vorigen Beispiel definiert werden. Beispielsweise will man möglicherweise auf ein paar Gebiete mit größerem Interesse fokussieren. Solche Wichtigkeitsmaße werden üblicherweise durch Nebenbedingungen modelliert. Es ist sehr einfach, Nebenbedingungen in unser Modell zu integrieren während die Zulässigkeit der Triangulierung erhalten bleibt. Dies wird zum Beispiel einfach dadurch erreicht, daß Fehlerindikatorwerte in den ausgewählten Bereichen vergrößert und saturiert werden.

Wir wollen das Nordsee Beispiel noch einmal betrachten. Nur mit der geometrischen Norm werden Küstenlinien nicht zufriedenstellend aufgelöst, weil eine grobe Approximation flacher Küsten nicht zu einem großen Fehler führt. Abhängig von der Anwendung können Küstenlinien aber wichtig sein. Nachdem Ozeangebiete im Datensatz markiert sind, können Nebenbedingungen einfach dadurch realisiert werden, daß die entsprechenden Fehlerindikatorwerte am Rand der Zerlegung mit einem konstanten Faktor multipliziert werden, was zu visuell erheblich besseren Approximationen führt (Abb. A.6).

6.1.5 Topologie–Erhaltung

Vor allem bei hydrologischen Modellierungen und Simulationen können Änderungen in der Topologie des Höhenmodells überraschende und ungewollte Effekte haben. Kleine Änderungen in den Höhenwerten können zu großen Änderungen in der Große und Struktur z.B. von Fluß–Einzugsgebieten haben. Auch bei der Visualisierung der Höheninformationen sieht man den Einfluß der Topologie des Höhenmodells, zum Beispiel wenn bei einer zu groben Approximation eine Insel mit dem benachbarten Festland verschmilzt oder ein vorher zusammenhängendes Land plötzlich vom Meer geteilt wird. Wir wollen nun zeigen, wie in unserem Multiskalen–Algorithmus die Topologie der Höhendaten erhalten werden kann.

Topologie kann durch die Menge von kritischen Punkten definiert werden. Ein kritischer Punkt ist definiert als ein Punkt im Raum wo eine Isolinie ihre Zusammenhangskomponenten ändert. Nachdem unser Multiskalenmodell auf stückweise linearer Interpolation basiert, können kritische Punkte nur an Eckpunkten der Triangulierung auftreten. Bei Multiskalenmodellen ist allerdings Vorsicht geboten. Ein Punkt, der auf der feinsten Auflösung nicht kritisch ist, kann bezüglich einer gröberen Auflösung kritisch werden. Daher müssen kritische

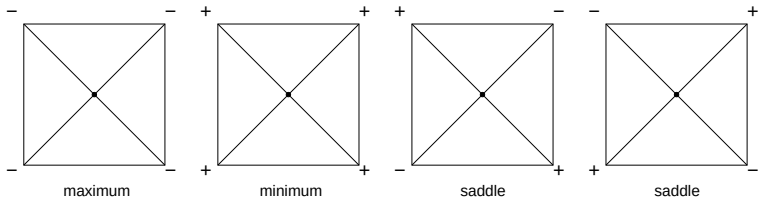


Abbildung 6.2: Die vier topologischen Fälle, bei denen kritische Punkte auftreten.

Punkte hierarchisch definiert werden.

Immer wenn ein Paar von Dreiecken verfeinert wird, ist der gemeinsame Verfeinerungsknoten, der in das Höhenmodell eingefügt wird, ein Kandidat für einen kritischen Punkt auf dem momentanen Level. Die vier möglichen Fälle, wo der Verfeinerungsknoten wirklich kritisch ist, sind in Abb. 6.2 abgebildet. Ein '–' zeigt an, daß der Höhenwert an diesem Knoten kleiner ist als der Höhenwert an dem Verfeinerungsknoten, ein '+', daß er größer ist. Dadurch, daß Fehlerindikator–Werte an kritischen Punkten auf ∞ gesetzt werden, hat jedes approximative DHM die gleiche topographische Struktur wie das Original für alle Werte von ε.

Als ein Beispiel betrachten wir ein digitales Höhenmodell in Westdeutschland in der Nähe des Laacher Sees[3]. In Abb. A.7 zeigen wir Isolinien und die zugehörigen adaptiven Triangulierungen für die L_∞–Norm ohne Topologie–Erhaltung. Man sieht, daß z.B. für eine zu grobe Triangulierung der See im Vordergrund eine Öffnung an seiner oberen Seite bekommt. Mit Topologie–Erhaltung (Abb. A.8) behalten alle Isolinien ihre Struktur und solch ungewollte Effekte sind eliminiert.

6.2 Fraktale Interpolation

Anstatt für einen gegebenen Datensatz eine Multiskalenmodell aufzubauen, können mit dem gleichen Algorithmus mit Hilfe fraktaler Interpolation künstliche Höhenmodelle erzeugt werden. Hierbei werden die Höhenwerte an den Verfeinerungsknoten von den Nachbarpunkten interpoliert und mittels eines Zufallsgenerators perturbiert (siehe [32]). Dieses Verfahren wird daher auch *midpoint displacement* Algorithmus genannt. Das Verfahren entspricht gerade der Brownschen Brücke zur hierarchischen Diskretisierung stochastischer Prozesse [56]. Dem Wiener Prozess (oder der Brownschen Bewegung) entsprechen hier normalverteilte Zufallsvariablen, deren Varianz

$$N(0, 2^{\alpha l})$$

ist, wobei l der momentane Gitterlevel und α der sogenannte fraktale Exponent sind. Höhere Werte für α liefern rauhere Landschaften während kleinere Werte glattere Gebiete erzeugen. Abb. A.9 zeigt verschiedene fraktale Zufallswelten für $\alpha = 0.5$ basierend auf verschiedenen Startwerten des (Pseudo-) Zufallszahlengenerators.

[3]Copyright von LVerA Rheinland–Pfalz

Umgekehrt können für einen gegebenen Datensatz die zugehörigen fraktalen Exponenten in Abhängigkeit von Ort und Skala durch inverses Fitting der Wavelet–Koeffizienten an obiges Bildungsgesetz berechnet werden. Dies erlaubt dann die fraktale Klassifikation realer Landschaften.

6.3 Schnittbildung

Wir wollen nun die zweidimensionale Welt der digitalen Höhenmodelle verlassen und uns dreidimensionalen Volumendaten zuwenden. Solche Daten begegnen uns vor allem bei medizinischen Aufnahmesystemen, wie der Computer–Tomographie, oder als Ergebnis numerischer Simulationen, wie zum Beispiel Strömungen in komplexen Geometrien. In den letzten Jahren ist die Auflösung und Verfügbarkeit von volumetrischen Datensätzen stark angewachsen.

Bei der Visualisierung von Volumendaten hat man neben der Datenmenge vor allem mit der Abbildung der dreidimensionalen Datenfunktion auf ein zweidimensionales Medium, einen Computerbildschirm oder ein Blatt Paper zu kämpfen. Bei einer solchen Abbildung gehen notwendigerweise Informationen verloren. Vielleicht die einfachste, aber dennoch oft verwendete Abbildung ist die Schnittbildung mit einer vorgegebenen Schnittebene. Indem diese Schnittebene interaktiv bewegt wird, erhält der Betrachter einen guten Eindruck von der Dreidimensionalität der Daten. In den folgenden Beispielen verwenden wir das Multiskalenmodell basierend auf Tetraederbisektion und linearer Interpolation.

In den zweidimensionalen Beispielen der Abschnitte 6.1 und 6.2 bestand die `VonInteresse`–Funktion einfach aus der Ausschnittbildung durch ein Clipping–Fenster. Diejenigen Teilgebiete, die außerhalb dieses Fensters lagen, wurden nicht betrachtet. Auf diese Weise wurde der Visualisierungsalgorithmus praktisch unabhängig von der Gesamtmenge der Daten, was eine sehr wichtige Eigenschaft bei der interaktiven Visualisierung ist.

Zur Berechnung von Schnitten durch einen dreidimensionalen Datensatz ist das Ziel der `VonInteresse`–Funktion einfach der, alle diejenigen Tetraeder zu finden, die eine vorgegebene Schnittebene schneiden. Wenn die gewünschte Auflösungsstufe erreicht ist, wird dann das Dreieck, das durch den Schnitt des momentanen Tetraeders mit der Schnittebene entsteht, gezeichnet. Abb. A.10 zeigt verschiedene solche Schnitte durch einen medizinischen Datensatz. Wie man sieht, entstehen durch eine adaptive Tetraederverfeinerung auch adaptive Triangulierungen auf den Schnittebenen. Nachdem die adaptiven Tetraedergitter zulässig sind, bilden auch Schnitte durch das Tetraedergitter zulässige Triangulierungen. Wiederum ist die Komplexität der Visualisierung unabhängig von der Gesamtdatenmenge, da nur diejenigen Tetraeder, die die Schnittebene schneiden, besucht werden.

Wir wollen hier anmerken, daß die Berechnung adaptiver Schnitte nicht auf Ebenen beschränkt ist, sondern Schnitte mit beliebigen Schnittfunktionen auf diese Weise leicht möglich sind. Weiterhin ist auch die gleichzeitige Darstellung mehrerer Schnitte kein Problem.

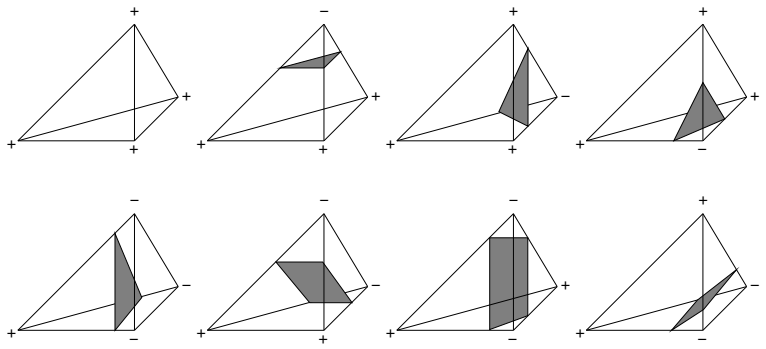


Abbildung 6.3: Lookup–Tabelle mit 8 der 16 Fälle zur Isoflächen–Extraktion.

6.4 Isoflächen–Extraktion

Isoflächen–Extraktion ist ein weiteressehr verbreitetes und nützliches Tool zur Visualisierung von Volumendaten. Eine Isofläche einer dreidimensionalen Funktion ist dabei definiert als diejenigen Bereiche des Grundgebiets wo die Funktion einen konstanten Wert, den sogenannten Isowert, annimmt. Der Bedarf an Multiskalenmethoden wird deutlich, wenn man versucht, Isoflächen von großen Datensätzen interaktiv zu extrahieren. Ohne Detailstufen kann die Zahl der Dreiecke aus der eine Isofläche zusammengesetzt ist leicht die Zahl der Dreiecke, die mit interaktiven Bildraten dargestellt werden können, übersteigen. Weiterhin kann die Extraktionsphase einfach zu langsam sein, um den Isowert interaktiv ändern zu können.

Multiskalenmethoden adressieren beide Probleme. In Prinzip ermöglichen sie dem Benutzer, die Genauigkeit der extrahierten Isofläche zu kontrollieren, indem die Zahl der Dreiecke reduziert wird (bzw. umgekehrt). Auf diese Weise kann der Benutzer große Volumendatensätze erkunden, indem er vereinfachte Approximationen der Isofläche für die Betrachtung der Gesamtstruktur verwendet und detailliertere Auflösungen zur Inspektion einer bestimmten Gegend oder wenn er in den Datensatz zoomt. Der Benutzer kann auch den Isowert interaktiv auf einer gröberen Detailstufe adjustieren.

Standard–Verfahren zur Isoflächen–Extraktion sind die Marching Cubes [97] oder Marching Tetrahedra [110] Verfahren, welche allerdings den gesamten Datensatz durchsuchen. Bislang existieren es zur Beschleunigung der Isoflächen–Extraktion drei Ansätze: geometrische [147], Saatzellen- [82], und Spannraum- [96] Methoden. Allerdings eignen sich nur geometrischen Methoden für Multiskalen–Algorithmen.

6.4.1 Extraktionsalgorithmus

Unsere Multiskalen–Ansatz basiert auf rekursiver Bisektion von Tetraedern. [58, 151]. Im Gegensatz zu Oktrees [125, 146], wo eine trilinere Interpolation verwendet wird, stimmen die Datenmodelle der triangulierten Isofläche und des volumetrischen Datensatzes überein (sie verwenden beide stückweise linea-

re Interpolation). Daher werden eine Menge Probleme, die durch verschiedene Datenmodelle entstehen, elimiert. Dies schließt auch Probleme bezüglich der Topologie der Isoflächen ein, wie wir sehen werden. Alternativ zur Bisektion kann auch rote (oder reguläre) Tetraederverfeinerung [70] verwendet werden, allerdings gestaltet sich hier adaptive Verfeinerung wesentlich schwieriger und kann nur durch Templates mit grüner (irregulärer) Verfeinerung erreicht werden.

Der Multiskalen Algorithmus basiert wiederum auf einem depth–first Durchlauf des Binärbaums von Tetraedern. An jedem Tetraeder wird auf das Fehlerkriterium überprüft. Falls das Kriterium wahr ist, wird der Durchlauf abgebrochen und die lokale Isofläche mit Hilfe eine lookup–Tabelle 6.3 extrahiert.

Weiterhin wird der Baumdurchlauf durch die `VonInteresse` Funktion lokal gestoppt, falls der Tetraeder kein Kandidat für einen Schnitt mit der Isofläche ist. In unserem Fall wird überprüft, ob der Isowert im Datenintervall aus allen Datenwerten innerhalb des Tetraeders enthalten ist. Diese Information kann explizit in einem bottom–up Durchlauf des Baumes (siehe [147]) gewonnen werden. Auf diese Weise ist wiederum die Komplexität des adaptiven Extraktions–Algorithmus von der Ordnung der Ausgabe (also der Zahl der gezeichneten Dreiecke), unabhängig von der Größe der Eingabe (ausser in ganz pathetischen Fällen, wie wenn der ganze Datensatz aus schachbrettartigem Rauschen besteht).

Mit Hilfe der η^*–Fehlerindikatoren (siehe Abschnitt 4.4) ist es ebenfalls möglich eine Schranke $\beta_0(S)$ für die Differenz der Originalfunktion und seiner linearen Approximation auf einem Tetraeder S auf Level l zu berechnen. Dies kann in der Implementierung der `VonInteresse`–Funktion verwendet werden. Man erhält

$$\min_{x \in S} f^l(x) - \beta_0(S) \leq f \leq \max_{\mathbf{x} \in S} f^l(\mathbf{x}) + \beta_0(S)\,.$$

und definiert

$$\beta_0(S) = \begin{cases} \frac{1}{2}\eta^+_\infty(\mathbf{x}_{ref}(S)), & \text{für hierarchische Offset Indikatoren} \\ d(S)\eta^+_{1,\infty}(\mathbf{x}_{ref}(S)), & \text{für gradientenbasierte Indikatoren} \end{cases}$$

In beiden Fällen kann die Speicherung von min/max–Werten vermieden werden. Beispiele für auf diese Weise extrahierte Isoflächen sieht man in Abb. A.1(e) und A.1(f).

6.4.2 Multilevel Backface Culling

Als weitere Beschleunigung des Verfahrens kann in dem Isoflächenextraktions–Algorithmus schon auf groben Tetraederen überprüft werden, ob alle Dreiecke, die von dem Algorithmus extrahiert werden, vom Betrachter weggeneigt sind, und demnach nicht gezeichnet werden müssen. Sei $n = \frac{\nabla f^l}{\|\nabla f^l\|}$ die Normale eines Dreiecks in der finalen Isoflächen–Triangulation auf dem Tetraeder $S \in \mathcal{T}^l$ und v der Blickvektor vom dem Objekt zum Auge (wir beschränken uns hier auf Parallelprojektion). Falls $n \cdot v \geq 0$ ist, ist das Dreieck zum Betrachter geneigt, ansonsten muß es nicht gezeichnet werden. Man erhält eine signifikante Beschleunigung des Isoflächen–Extraktions Algorithms, falls auf einem sehr

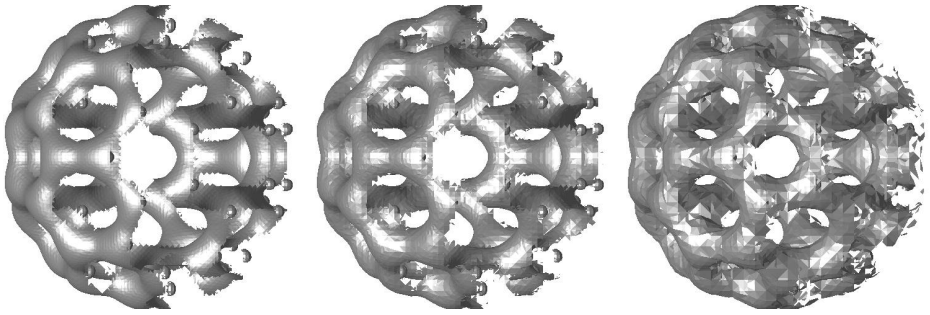


Abbildung 6.4: Hier wird der Effekt von *multilevel backface culling* mit Hilfe des Fehlerindikators $\eta^+_{1,\infty}$ gezeigt. Die eigentliche Blickrichtung ist von links. Die Schwellwerte sind 0.0, 0.1, und 0.4.

viel gröberen Gitter diejenigen Tetraeder erkannt werden, die nur Isoflächen–Dreiecke enthalten, welche vom Betrachter weg zeigen. In diesem Fall kann man den Baumdurchlauf auf diesem Level stoppen. Falls $\beta_N(S)$ eine Schranke für die Variation von n in $S \in \mathcal{T}^l$ ist, erhält man den Test

$$n \cdot v + \beta_N(S) \leq 0.$$

Eine mögliche Wahl für $\beta_N(S)$ ist $\eta^+_N(\mathbf{x}_{ref}(S))$. Läßt man die Normalisierung weg und betrachtet stattdessen eine Schranke $\beta_1(S)$, welche den möglichen Offset in $\|\nabla f\|$ mißt, erhält man das alternative Stoppkriterium

$$\nabla f^l \cdot v + \beta_1(S) \leq 0, \text{ mit } \beta_1(\mathbf{x}) = \eta^+_{1,\infty}(\mathbf{x}_{ref}(S)).$$

Wie man leicht sieht, spart man im Mittel, während man das Objekt beliebig dreht, bis zur Hälfte der Berechnungszeit für die Isofläche, da nur fast die Hälfte der Tetraeder durchlaufen werden muß (Abb. 6.4).

6.4.3 Topologie–Erhaltung

Die Form der extrahierten Isoflächen hängt stark von der Wahl des Fehlerindikators ab. Es gibt eine große Vielfalt von geometrischen Fehlerindikatoren, die verwendet werden können, um Verfeinerung von Tetraedergittern für Visualisierungszwecke zu steuern [59]. Allerdings erlauben diese im allgemeinen nicht, mögliche Änderungen in der topologischen Struktur der Isofläche zu kontrollieren, auch wenn konservative Fehlermaße (z.B. basierend auf der L_∞–Norm) verwendet werden.

Im Prinzip wäre es möglich, zunächst die Isofläche auf der feinsten Zerlegung topologisch korrekt zu extrahieren [23, 104, 130, 139, 150] und dann ein Topologie–erhaltendes Vereinfachungsverfahren [16, 40, 73, 79, 87] anzuwenden. Diese Verfahren arbeiten jedoch in einem bottom–up Ansatz auf dem extrahierten Dreiecksgitter in der höchsten Auflösung und würden daher nicht unsere Echtzeit–Beschränkungen treffen.

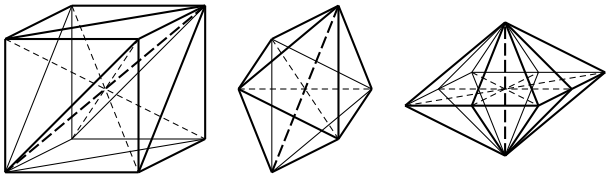


Abbildung 6.5: Die drei Typen von umgebenden Polyedern, die während der Verfeinerung auftauchen. Verfeinerungskanten sind fett gestrichelt, neue Kanten, die durch Verfeinerung entstehen, sind dünn gestrichelt.

Kritische Punkte

Kritische Punkte können im Dreidimensionalen analog wie im Zweidimensionalen als diejenigen Punkte im Raum wo eine Isofläche ihren Genus ändern würde, definiert werden. Nachdem unser Datenmodell stückweise linear ist, können wiederum kritische Punkte nur an Knoten des Tetraedergitters auftauchen. Dies wäre nicht so im Fall von Oktaedergittern und trilinieare Interpolation [139].

Die Grundidee, Topologie-Erhaltung in unseren Multiskalen–Algorithmus zu integrieren ist folgende Annahme: falls ein adaptives Tetraedergitter alle kritischen Punkte enthält, dann ist jede approximative Isofläche vom gleichen Genus wie die entsprechende Isofläche auf dem feinsten Gitter

Kritische Punkte müssen allerdings wiederum hierarchisch, basierend auf der Bisektionshierarchie der Tetraeder, definiert werden, weil ja nicht kritische Punkte auf der feinsten Auflösung kritisch bezüglich einer gröberen Auflösung werden können (siehe [5]). Daher ist ein Verfeinerungsknoten ein hierarchisch kritischer Punkt, falls der Genus einer Isofläche, beschränkt auf alle Tetraeder, die die Verfeinerungskante gemeinsam haben, sich ändert, wenn die Tetraeder verfeinert werden. Falls ein adaptives Tetraedergitter alle hierarchisch kritischen Punkte enthält, ist eine Topologie–Erhaltung erreicht.

Lookup–Tabellen

Obwohl hierarchisch kritische Punkte in einem Vorverarbeitungsschritt gefunden werden konnen, sollte ihre Identifikation so effizient wie möglich erfolgen. Zwei Isoflächen für Isowerte $f(\mathbf{x}_{\text{ref}}) - \tau$ und $f(\mathbf{x}_{\text{ref}}) + \tau$ (τ klein) zu extrahieren und ihre Zusammenhangskomponenten zu vergleichen, wäre zu komplex und zeitaufwendig. Nachdem allerdings die Topologie der Isofläche nur von der relativen Ordnung der Datenwerte an den Eckpunkten der Tetraeders, die die Verfeinerungskante gemeinsam haben, abhängt, können kritische Punkte wiederum effizient mit Hilfe von Lookup–Tabellen identifiziert werden. Dies ist hier jedoch ein wenig aufwändiger als im zweidimensionalen Fall.

Der umgebende Polyeder einer Verfeinerungskante ist definiert als der Rand aller adjazenten Tetraeder, die die Verfeinerungskante gemeinsam haben. In unserem Fall von regulären Gittern und rekursiver Bisektion tauchen drei Typen von Polyedern auf: ein triangulierter Würfel, ein Oktaeder, und ein Diamant (siehe Abb. 6.5). Der Würfel tritt in Leveln $(l \bmod 3) = 0$ aud, der Oktaeder in Leveln

(l mod 3) = 1, und der Diamant in Leveln (l mod 3) = 2.

Die m Knoten eines umgebenden Polyeders werden mit einem +–Zeichen markiert, falls der Datenwert an dem Knoten größer ist als der Wert an dem Verfeinerungsknoten und mit einem −–Zeichen, falls er kleiner ist. Die Lookup–Tabelle für jeden Typ von Polyeder besteht aus den 2^m möglichen Fällen und enthält ein Bit, welches anzeigt, ob der Verfeinerungsknoten kritisch ist, oder nicht.

Vereinfachte Lookup–Tabellen können durch die Identifizierung aller Symmetrieklassen für die +/− Bitmuster der verschiedenen Typen von Polyedern und durch Überprüfung, ob der Verfeinerungsknoten kritisch ist oder nicht, für jede dieser Klassen erzeugt werden.

Abb. 6.6 zeigt die resultierenden 25 Klassen für den Würfel. Nicht gezeigt ist Fall 1, wo alle Knoten das gleiche Vorzeichen haben, was zu einem lokalen Minimum oder Maximum führen würde, welches immer kritisch ist. Die Bilder zeigen Isoflächen nach Verfeinerung für einen Isowert von $f(\mathbf{x}_{\text{ref}})$. Kritische Punkte tauchen auf, falls zwei oder mehr Komponenten der Isofläche eine (nicht mannigfaltige) Punktverbindung an dem Verfeinerungsknoten in der Mitte der Würfels haben. Dies ist wahr in den Fällen 4b, 5a, 5b, 7b, 8b, 12b und 13.

Die Hauptklassen sind identisch zu der Marching Cubes Lookup–Tabelle [97], hier müssen aber ein paar Nebenklassen betrachtet werden. Diese Nebenklassen tauchen auf, weil die Endpunkte der Verfeinerungskante (von vorne–unten–links nach hinten–oben–rechts) unterschiedliche Konnektivität haben als die anderen Knoten des Polyeders.

Automatische Erzeugung der Lookup–Tabellen

Manuelle Konstruktion solcher Lookup–Tabellen ist natürlich potentiell fehleranfällig, nachdem Nebenklassen leicht vergessen oder falsch klassifiziert werden können. Es gibt aber einen effizienten und exakten Weg, Lookup–Tabellen zur Identifikation von kritischen Punkten automatisch zu erstellen. Er besteht aus den folgenden vier Schritten:

1. konstruiere den Kantengraph des Polyeders,
2. lösche alle Kanten zwischen einem + und einem − Knoten aus dem Graph,
3. zähle die überbleibenden zusammenhängenden Komponenten des Graphs,
4. falls die Zahl der Komponenten 1 oder größer als 2 ist, dann ist der Verfeinerungsknoten kritisch.

Als ein Beispiel wollen wir Fall 13 aus Abb. 6.6 betrachten. Die Knoten des Würfels sind von vorne nach hinten, unten nach oben, links nach rechts numeriert. In diesem Fall sind die Vorzeichen der Knoten durch $--+++--$ gegeben. Man beachte, daß der Würfel trianguliert ist, und daher jede Würfelseite eine Diagonalkante, wie in Abb. 6.5 gezeigt, besitzt.

1. Kantengraph für den Würfel:

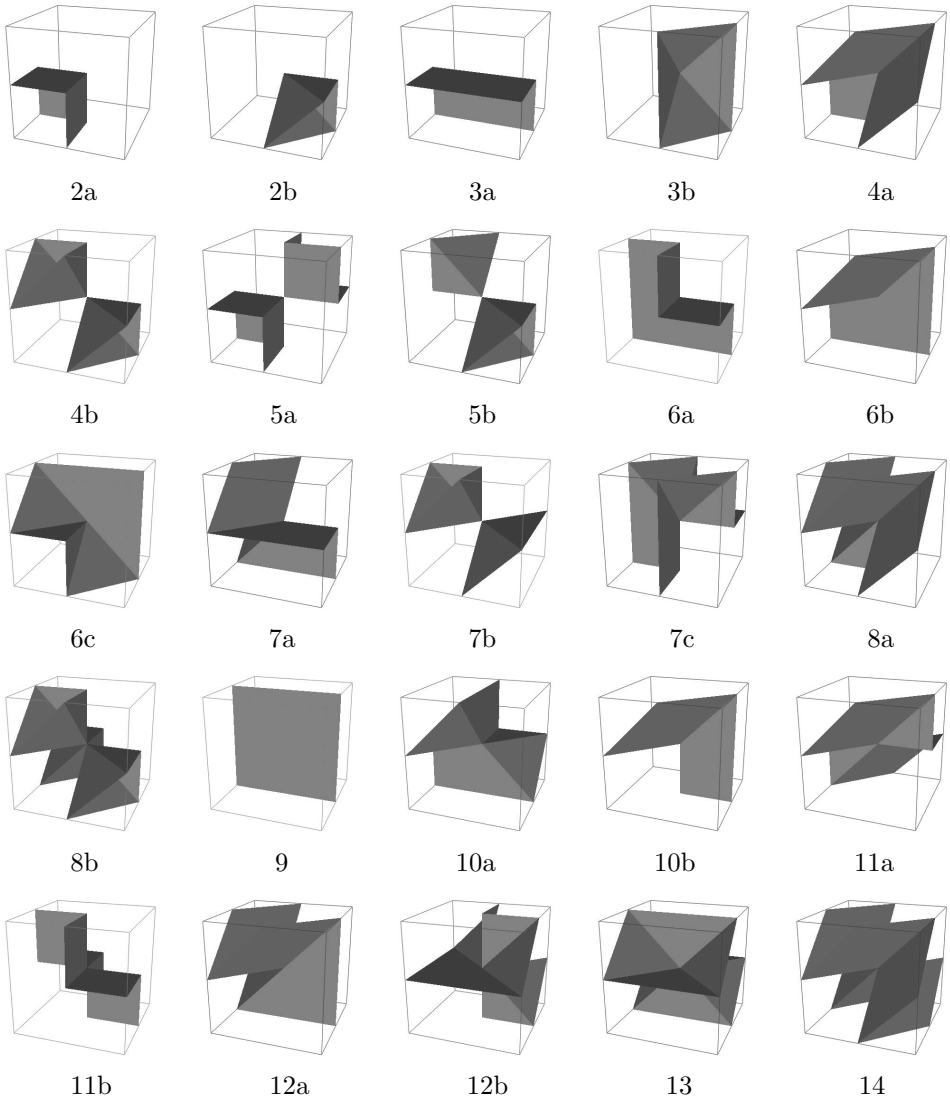


Abbildung 6.6: Topologische Klassen bei der Isoflächenextraktion für den Würfel.

	−	−	+	+	+	+	−	−
−	×							
−	×	×						
+	×		×					
+	×	×	×	×				
+	×				×			
+	×	×			×	×		
−	×		×		×		×	
−		×	×	×	×	×	×	×

2. lösche Kanten:

	−	−	+	+	+	+	−	−
−	×							
−	×	×						
+			×					
+			×	×				
+					×			
+					×	×		
−	×						×	
−		×					×	×

3. Zähle Komponenten (transitive Hülle):

	−	−	+	+	+	+	−	−
−	×							
−	×	×						
+			×					
+			×	×				
+					×			
+					×	×		
−	×	×					×	
−	×	×					×	×

4. 3 Komponenten → kritisch

Die Gesamtlaufzeit dieses Algorithmus ist von der Ordnung $O(2^m \cdot m^2)$ für jeden Typ von Polyeder. Natürlich muß diese Berechnung nur einmal im Fall von regulärer Gitterverfeinerung gemacht werden. Dementsprechend tauchen für reguläre Tetraederbisektion kritische Punkte in 68/256 Fällen für den Würfel, in 8/64 Fällen für den Oktaeder und in 400/1024 Fällen für den Diamanten auf.

Diese automatische Konstruktion von Lookup–Tabellen ist nicht beschränkt auf reguläre Gitter, sondern kann auch für alle irregulären Tetraedergitter, die durch Kantenbisektion generiert werden (siehe Abschnitt 2.3.3), verwendet werden.

Kritische Intervalle

Ein naheliegender Weg, Topologie–Erhaltung in den Multiskalen–Isoflächen Extraktionsalgorithmus zu integrieren, wäre es, Fehlerindikatorwerte an hierarchisch kritischen Punkte auf unendlich zu setzen. Allerdings ist für eine gegebene Isofläche nur ein Teil aller kritischen Punkte wirklich relevant und viele Tetraeder würden unnötigerweise verfeinert werden.

Daher wird jedem kritischen Punkt ein kritisches Intervall zugewiesen. Das kritische Intervall ist definiert als der Bereich von Isowerten, für die die Isofläche ihre Topologie ändert, wenn die Tetraeder an dem kritischen Punkt verfeinert werden. Mit dieser Information ist es dem Extraktionsalgorithmus möglich, nur an denjenigen kritischen Punkten, bei denen das kritische Intervall den momentanen Isowert enthält, zu verfeinern.

Die kritischen Intervalle sind relativ leicht zu ermitteln. Sie sind definiert als

$$I(\mathbf{x}_{\text{ref}}) := [f(\mathbf{x}_{\text{ref}}), \min\{f(\mathbf{x}_1), f(\mathbf{x}_2)\}]$$

für Verfeinerungskanten, deren Endpunkte $\mathbf{x}_1$ und $\mathbf{x}_2$ positive Vorzeichen haben und als

$$I(\mathbf{x}_{\text{ref}}) := [\max\{f(\mathbf{x}_1), f(\mathbf{x}_2)\}, f(\mathbf{x}_{\text{ref}})]$$

für negative Vorzeichen. Verfeinerungskanten deren Endpunkte unterschiedliche Vorzeichen haben führen nie zu kritischen Punkten.

Um zu verhindern, daß kritische Punkte während des Baumdurchlaufs vergessen werden, müssen kritische Punkte ähnlich wie die Fehlerindikatorwerte saturiert werden. Jedem Tetraeder wird dabei ein kritisches Intervall zugeordnet, das als das kritische Intervall seines Verfeinerungsknotens definiert ist. In einem levelweisen bottom–up Durchlauf des Tetraederbaumes werden kritische Intervalle vereinigt, indem die oberen und unteren Intervallgrenzen des momentanen Tetraeders und seiner zwei Kinder genommen werden.

Beispiele

In unserem Isoflächen–Extraktionsalgorithmus ist die Extraktionsgeschwindigkeit pro Dreieck fast so hoch wie die Zeichengeschwindigkeit pro Dreieck auf modernen Graphik–Workstations. Dies gilt auch die die Topologie erhaltende Version, nachdem nur ein einziger zusätzlicher Check (nämlich ob der Isowert in dem momentanen kritischen Intervall enthalten ist) für jeden Tetraeder während des Baumdurchlaufs notwendig ist

Wir wollen die Leistung des Algorithmus anhand zweier Beispiele zeigen. Das erste Beispiel ist die Nullmenge der algebraischen Funktion

$$f(x, y, z) = \sqrt{x^2 + y^2} - (\frac{1}{2}x + \frac{1}{2}y - z + 0.01)^2$$

welche auf $[-1, 1]^3$ definiert ist und auf einem uniformen 65^3 Gitter gesampelt wurde. Diese Funktion hat eine starke Diagonalsingularität in der Nähe des Ursprungs, wie in Abb. A.11 zu sehen ist. In Tabelle 6.1 vergleichen wir die Zahl der Dreiecke für verschiedene geometrische Fehlerschranken ε (L_∞ Norm). Bei der Topologie erhaltenden Version wird die Originaltopologie exakt bei nur geringfügig höherer Dreieckszahl erhalten. Ohne Topologie–Erhaltung geht die feinen Strukturen der Funktion für große Fehlerschranken verloren.

Das zweite Beispiel ist der uns schon bekannte Buckyball Datensatz. Abb. A.12 zeigt Isoflächen für verschiedene geometrische Fehlerschranken ε (L_∞–norm) und für einen Isowert, wo der Datensatz eine komplexe topologische Struktur aufweist. Die genauen Werte von ε und die entsprechenden Dreieckszahlen

ε	ohne Topologie-Erhaltung	mit Topologie-Erhaltung
0.0	59290	59290
1/64	25456	25464
1/16	7124	7528
1/4	1936	3503
1	374	3095

Tabelle 6.1: Zahl der Dreiecke für die algebraische Funktion für variierende geometrische Fehlerschranken ε.

ε	ohne Topologie-Erhaltung	mit Topologie-Erhaltung
0.0	395798	395798
0.01	153539	163818
0.05	29594	110101
0.1	12552	109981

Tabelle 6.2: Zahl der Dreiecke für den Buckyball Datensatz für variierende geometrische Fehlerschranken ε.

sind in Tabelle 6.2 aufgelistet. Ohne Topologie–Erhaltung hält die Isofläche nicht die korrekte Konnektivität über verschiedene Detailstufen (drittes Bild von links), oder fällt komplett auseinander (viertes Bild). Mit Topologie–Erhaltung wird die Originaltopologie exakt bewahrt. Im letzteren Fall ist allerdings die geometrische Vereinfachung limitiert. Ab einer gewissen Detailstufe führt eine Erhöhung des Schwellwerts ε nicht mehr zu niedrigeren Dreieckszahlen.

6.4.4 Kontrollierte Topologie–Verfeinfachung

Wir wir im vorigen Beispiel gesehen haben, führt Topologie–Erhaltung üblicherweise zu einem Limit für die minimale Zahl an Dreiecken in der Isoflächen–Triangulierung. Abhängig von der Anwendung kann es aber wünschenswert sein, die resultierenden Dreiecksgitter weiter zu vereinfachen, was kontrollierte Änderungen im Genus der Isoflächenkomponenten erfordert.

Während der ursprungliche — nicht Topologie erhaltende — Multiskalen Algorithmus Topologie in natürlicher Weise vereinfacht, so tut er dies doch in einer unkontrollierten Art und Weise. Wiederum wäre es im Prinzip möglich kontrollierte Topologie vereinfachende Simplifizierungs–Algorithmen für Dreiecksgitter, wie z.B. Filterung und Resampling [74], α–Hüllen [41], oder Oktree–basierte Verfahren [1], in einem Nachverarbeitungsschrit der Isoflächen–Extraktion anzuwenden. Allerdings arbeiten diese Algorithmen wiederum in einem bottom–up Ansatz auf dem extrahierten Dreiecksgitter in der höchsten Auflösung und eignen sich deshalb nicht für interaktive Anwendungen.

Der Topologie erhaltende Algorithmus, der in den vorigen Abschnitten besprochen wurde, kann jedoch leicht in einen kontrolliert Topologie vereinfachenden Algorithmus transfomiert werden. Wir wollen dafür eine Gewichtsfunktion $w(\mathbf{x}_{\text{ref}})$ auf allen hierarchisch kritischen Punkten definieren, welche die Wich-

tigkeit dieses Punktes anzeigen soll. Der Topologie vereinfachende Algorithmus kann dann diejenigen kritischen Punkte weglassen und muss nicht an ihnen verfeinern, die nicht wichtig sind, also $w(\mathbf{x}_{\text{ref}}) < \delta$, wobei δ ein weiterer benutzerdefinierter Schwellwert ist.

Auf diese Weise hat der Benutzer durch die Spezifikation einer geeigneten Gewichtsfunktion w auf den kritischen Punkten zusammen mit einem Schwellwert δ die vollständige Kontrolle über die Topologiex–Verfeinfachung. Die eigentlichen Werte der Gewichte w an den hierarchisch kritischen Punkten können zusammen mit dem Fehlerindikator η und den kritischen Intervallen I vorab berechnet werden.

Eine passende Wahl für die Gewichtsfunktion w wird stark von dem Typ der Anwendung und der Struktur des Datensatzes abhängen. Wir wollen nun ein paar Möglichkeiten für die Gewichtsfunktion betrachten:

- Eine offensichtliche Wahl für w wäre die Größe des kritischen Intervalls. Dies bedeutet keinen zusätzlichen Berechnungsaufwand im betrachteten Rahmen. Indem die Intervallgröße mit dem maximalen Gradienten der Daten auf allen adjazenten Tetraedern multipliziert wird, erhält man einen Indikator für die Menge an topologischen Änderungen in der Isoflächenstruktur, die durch die Verfeinerung der Tetraeder, die den kritischen Punkt umgeben, erzeugt wird.
- Eine andere Möglichkeit wäre die Größe der entsprechenden lokalen Isofläche, also die Zahl der Tetraeder, die die Familie von Isoflächen innerhalb des kritischen Intervalls schneiden würde. Dies ergibt einen Kontrollmechanismus über die lokale geometrische Komplexität.
- Man könnte auch den Abstand zwischen Isoflächenkomponenten, die durch den kritischen Punkt separiert werden, was natürlich nur Sinn im Fall von Sattelpunkten macht, verwenden. Dies erlaubt Kontrolle über die geometrische Trennung zwischen Oberflächenkomponenten, also getrennte Komponenten können vereinigt und vereinfacht werden, falls der Abstand zwischen ihnen klein genug ist.

Ein ausführlicher Vergleich von Vereinfachungs–Techniken für verschiedene Anwendungen würde sicherlich den Rahmen dieser Arbeit sprengen. Wir wollen daher hier nur ein einfaches aber charakteristisches Beispiel betrachten. Hierfür verwenden wir einen typischen bio–medizinischen Datensatz, einen CT Scan eines Hummers[4]. In Abb. A.13 sehen wir daß die Original–Isofläche (1089004 triangles) stark verrauscht ist. Der Topologie vereinfachende Algorithmus kann die Zahl der Dreiecke auf 609893 reduzieren, aber das Rauschen in den Daten wird natürlich bleiben. Wir wollen daher die Gewichtsfunktion abhängig von der Größe des kritischen Intervalls definieren, d.h.

$$w(\mathbf{x}_{\text{ref}}) := |I(\mathbf{x}_{\text{ref}})|.$$

Die Topologievereinfachung kann so diese Artefakte gut reduzieren, während die Gesamtstruktur des Tieres erhalten bleibt. Die vereinfachten Isoflächen bieten visuell bessere Darstellungen des Datensatzes mit nur 367271 Dreiecken für $\delta = 0.2$ und 262357 Dreiecken für $\delta = 0.7$.

[4]Copyright von AVS

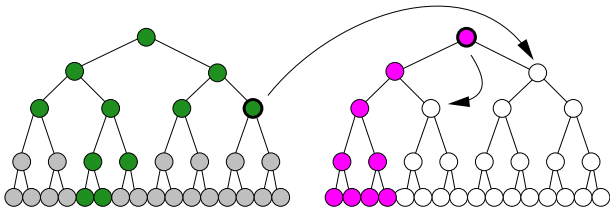


Abbildung 6.7: Lastbalanzierung bei der Isoflächen–Extraktion: der linke Prozess ist mit seinem Teilbaum gerade fertig geworden und übernimmt einen Teilbaum des rechten Prozesses.

6.4.5 Parallelisierung

Für sehr große Datenmengen kann selbst unser Multiskalenverfahren für interaktive Anwendungen zu langsam sein. Eine weitere Beschleunigung des Verfahrens wäre jedoch durch Parallelisierung möglich. Im folgenden wollen wir nun erläutern, wie der Isoflächen Extraktions–Algorithmus effizient parallel auf einer Workstation mit mehreren Prozessoren ausgeführt werden kann (siehe [6, 10, 51]).

Die erste Idee hierfür wäre, ein statisches Gebietszerlegungs–Verfahren bezüglich des Tetraedergitters zu verwenden. Allerdings kann dieser Ansatz nicht das komplizierte Verhalten des adaptiven und hierarchischen Isoflächen–Extraktionsalgorithmus handhaben. Große und kleine Bereiche der Isofläche würden in Gebieten der gleichen Größe zu liegen kommen. Daher würde die Zahl der von dem Algorithmus besuchten Tetraeder in jedem Gebiet – welche proportional zur benötigten Rechenzeit ist – stark variieren und der Vorteil durch die Parallelisierung würde zum Teil wieder verloren gehen. Das Problem ist daher, wie man verhindern kann, daß Prozessoren untätig werden.

Hier wollen wir eine effiziente Lösung dieses Problems liefern, welche unabhängig von der konkreten Form der Isofläche und leicht implementierbar ist. In Hinblick auf die interaktive Visualisierung beschränken wir uns hier auf den Fall einer Workstation mit gemeinsamen Speicher, wo Prozesse (Threads) vergleichbare Zugriffszeiten auf beliebige Speicherbereiche im kompletten Datensatz haben. Der Austausch von Gebietsdaten kann kritisch in verteilten Umgebungen sein, wenngleich auch unvermeidbar im Fall extrem großer Datenmengen. In diesem Fall müssen Lastbalanzierungs–Entscheidungen sorgfältiger getroffen werden.

Zu Beginn der Parallelisierung werden mehrere Prozesse gestartet, von denen jeder einen Teilbaum der gesamten Tetraeder–Hierarchie erhält, der unabhängig von den Teilbäumen der anderen Prozesse behandelt werden kann. Wir identifizieren den obersten Knoten des momentanen Teilbaums eines Prozesses durch eine Markierung. Genauer gesagt, nehmen wir an, daß der rechte Teilbaum des Prozesses bisher noch nicht bearbeitet wurde. Falls ein Prozess beginnt, seinen rechten Teilbaum zu bearbeiten, wird die entsprechende Markierung um eins nach unten in seinen rechten Teilbaum verschoben.

Wenn nun ein Prozeß seinen gesamten Teilbaum zu Ende bearbeitet hat, muß eine neue Aufgabe für ihn gefunden werden. Dafür wird der Eintrittsknoten eines rechten Teilbaums eines anderen Prozesses als des momentan freien Prozessors

ε	gezeichnete Dreiecke	besuchte Tetraeder	Bild/sec (1 Proz)	Bild/sec (4 Proz)
0.02	81184	201757	3.45	8.33
0.01	128709	307384	2.27	5.26
0.005	211219	487107	1.43	3.44
0.0025	315440	727419	0.98	2.43
0.00125	439230	984029	0.74	1.78
0.0	590018	1259669	0.58	1.36

Tabelle 6.3: Zahl der erzeugten Dreiecke, besuchte Tetraeder, Bilder pro Sekunde für den skalaren und parallelen Fall für verschiedene Schwellwerte und dem hierarchischen Fehlerindikator η_H^+.

ausgewählt. Wir wählen hierfür immer den Prozess, dessen Markierung sich auf dem gröbsten Gitterlevel befindet. Diese Markierung wird zunächst nach links unten und dann rekursiv nach rechts unten weiter verschoben bis ein Knoten erreicht wird, dessen rechter Teilbaum bisher nicht besucht wurde (Abb. 6.7). Diese Strategie stellt sicher, daß ein untätiger Prozeß immer mit dem größten unbesuchten Teilbaum versorgt wird.

Wir wollen schließlich die eigentliche Implementierung im Falle von 6 Tetraedern auf dem gröbsten Level der Gitterhierarchie, z.B. für einen Würfel, der in Tetraeder unterteilt wurde, betrachten. Wir indizieren diese Tetraeder mit den Zahlen von 6 bis 11. Die übrigen Tetraeder werden rekursiv numeriert, wobei das linke und rechte Kind die Nummern $2i$ bzw. $2i+1$ für einen Tetraeder mit Index i erhalten. Wir müssen nur ein Feld mit solchen Indizes, jeweils einen für jeden Prozeß speichern. Falls ein Prozeß seinen linken Teilbaum beendet hat, setzt er seine Markierungsnummer auf den rechten Teilbaum bis der gesamte Baum verarbeitet wurde. Ein arbeitsloser Prozeß sucht nach der kleinsten Markierungsnummer $i_{\min}$ aus dem Feld der Indizes, setzt seine Nummer auf $2i_{\min}+1$ und modifiziert den anderen Index.

Falls die Workstation mit m Prozessoren ausgestattet ist, können wir m Prozesse starten. Einer von ihnen wird dafür reserviert, Graphikprimitive aus einem Puffer an die Graphik–Hardware weiterzuleiten. Die übrigen $m-1$ Prozesse können mit der eigentlichen Isoflächen–Extraktion beschäftigt werden. Hierbei sammeln sie die zu zeichnenden Dreiecke in Puffern, die sie an den ersten Prozess weiterleiten. Auf einer 4 Prozessor SGI R10000 workstation mit Infinite Reality Graphik kann auf diese Weise eine Beschleunigung von 2.4 und eine Rate von 800.000 Dreiecke pro Sekunde erreicht werden.

Als Beispieldatensatz wollen wir einen MRI Scan eines Frauenkopfes[5] betrachten. Abb. A.14 zeigt verschiedene extrahierte Isoflächen, wobei die Farbe eines Bereichs dem Prozessor entspricht, der ihn extrahiert hat. Detaillierte Ergebnisse bezüglich der Geschwindigkeit und der Zahl der extrahierten Dreiecke sind in Tabelle 6.3 aufgelistet. In diesem Beispiel wurde der hierarchische Offset Fehlerindikator basierend auf der L_∞ Norm verwendet.

[5]Copyright von MeVIS Bremen

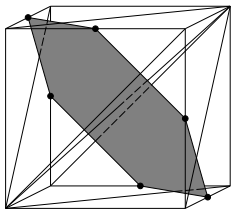


Abbildung 6.8: Sortierpunkte und Normalenebene des Ausgangs–Tetraedergitters.

6.5 Volumendarstellung

Zur volumetrischen Darstellung großer Datensätze werden neben direkten Volumendarstellungen, wie *ray casting*, *splatting* oder 3D *texture mapping* indirekte Volumendarstellungen, wie die Extraktion mehrerer transparenter Isoflächen eingesetzt. In beiden Settings werden oft Hardwarebeschleunigung und Multiskalenmethoden benötigt, um den Echtzeit–Anforderungen gerecht zu werden. Hierfür gibt es verschiedene Ansätze bei 3D texture mapping [88, 144], raycasting [44, 103, 145], splatting [71, 89, 93] oder bei der *shear–warp* Transformation [148]. Für einen detaillierten Vergleich dieser Methoden siehe [101].

6.5.1 Transparente Isoflächen

Die Extraktion mehrerer transparenter Isoflächen basiert auf der Fähigkeit momentaner Graphikprozessoren große Mengen an transparenten Dreiecken sehr schnell mittels *alpha blending* zeichnen zu können. Die Darstellung transparenter Isoflächen wird oft als Ersatz zu direkten Volumendarstellungs–Methoden eingesetzt, vor allem wenn Transferfunktionen (d.h. Abbildungen von den Datenwerten auf die entsprechenden Transparenzen) mit wenigen stark ausgeprägten Spitzen verwendet werden.

Das Zeichnen transparenter Isoflächen durch *alpha blending* benötigt allerdings eine *back–to–front* Darstellung der Dreiecke. Dies erfordert eine blickrichtungsabhängige Sortierung der Isoflächendreiecke. Im Prinzip könnten die Isoflächengitter erst extrahiert und dann sortiert werden, aber dann dominiert die Sortierzeit die gesamte Darstellungszeit, sobald der Benutzer die Blickrichtung ändert. Dies ist besonders problematisch, nachdem die Rotation der Isofläche die vorherrschende Interaktion des Benutzers während der Visualisierung ist und demnach die Sortierung für jedes Bild gemacht werden muß.

Multiskalenmethoden erlauben jedoch eine back–to–front Sortierung während der Extraktionsphase wodurch obige Probleme eliminiert werden. Wir werden nun zeigen, wie diese Sortierung mit praktisch keinen Zusatzkosten für Bisektionsgitter durchgeführt werden kann. Dabei treten drei verschiedene Sortierprobleme auf: Sortierung der Ausgangstetraeder, rekursive Sortierung der Kindtetraeder während des Baumdurchlaufs und Sortierung der Isoflächenkomponenten innerhalb eines Tetraeders während der Extraktion.

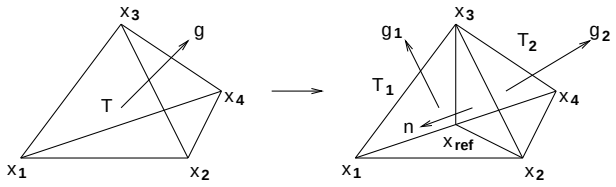


Abbildung 6.9: Gradienten g und Normale der Trennebene n während der Bisektion von S in S_1 und S_2.

Ausgangstetraeder

Wir wollen nun annehmen, daß die Anfangstriangulierung der Kuhn–Triangulierung entspricht. Die Tetraeder müssen nun beginnend mit dem am weitesten entfernten Tetraeder und endend mit dem am naheliegendsten Tetraeder sortiert werden. Sei nun v der Blickvektor (vom Auge zum Objekt) und n die Normale der Trennebene zweier Tetraeder S_i und S_j. Wir wollen nun annehmen, daß S_i in Richtung der Normale n liegt und S_j sich in der Gegenrichtung befindet. Dann liegt S_i hinter S_j falls $v \cdot n > 0$ und vor S_j falls $v \cdot n < 0$. Falls $v \cdot n = 0$ könnten diese beiden Tetraeder parallel bearbeitet werden.

Für die sechs Ausgangstetraeder sind zehn Normalen von Trennebenen zwischen je zwei Tetraedern möglich. Diese Normalen können leicht mittels des Vektors zweier bestimmter Punkte definiert werden, wobei jeder Punkt auf dem Rand eines der Tetraeder liegt. Diese sechs Punkte sind die Mittelpunkte derjenigen Kanten deren Endpunkte nicht Endpunkte der Diagonale des Würfels sind. Auf diese Weise wird eine Normalenebene aufgespannt (siehe Abb. 6.8). Mit dieser Information kann ein beliebiger Sortieralgorithmus wie *quicksort* direkt angewandt werden um die Tetraeder zu sortieren.

Rekursive Sortierung während der Verfeinerung

Während des Baumdurchlaufs wird ein Tetraeder in zwei Kindtetraeder aufgespalten. Die Trennebene zwischen diesen Tetraedern wird durch die Punkte $\mathbf{x}_2$, $\mathbf{x}_3$ und $\mathbf{x}_{ref}$ aufgespannt (Abb. 6.9). Die Normale n der Trennebene zeige nun in Richtung von S_1. Dann muß ähnlich wie im vorigen Abschnitt S_1 vor S_2 besucht werden, falls $v \cdot n > 0$ und in umgekehrter Reihenfolge sonst.

Die Normale n könnte durch $\overline{\mathbf{x}_2\mathbf{x}_{ref}} \times \overline{\mathbf{x}_3\mathbf{x}_{ref}}$ während des Baumdurchlaufs berechnet werden, aber es ist effizienter diese Information für jeden Typ von Referenztetraeder vorab zu berechnen.

In unserem Fall gibt es 72 Referenztetraeder. Von den drei Grundtypen an Tetraedern, die sich alle drei Verfeinerungslevel abwechseln existieren 24 Instanzen durch respektive Rotationen und Spiegelungen. Zum Beispiel teilen sich die sechs Tetraeder vom ersten Typ die gleiche Diagonale des Würfels, wobei es vier mögliche Diagonalen gibt.

Die Nummern der Referenztetraeder können im Falle regulärer Gitter hierarchisch berechnet werden. Hier müssen also basierend auf der Nummer N des

N	0	1	2	3	4	5	6	7	8	9	10	12
N_1	24	26	28	30	32	34	33	31	38	40	41	42
N_2	25	27	29	31	33	35	36	37	39	26	24	43
N	12	13	14	15	16	17	18	19	20	21	22	23
N_1	35	42	41	46	47	25	43	34	32	45	44	36
N_2	44	45	30	28	39	37	47	46	40	38	29	27
N	24	25	26	27	28	29	30	31	32	33	34	35
N_1	48	50	52	54	56	58	60	62	64	66	68	70
N_2	49	51	53	55	57	59	61	63	65	67	69	71
N	36	37	38	39	40	41	42	43	44	45	46	47
N_1	51	59	61	55	57	65	69	71	67	63	49	53
N_2	70	54	56	58	60	68	64	50	62	66	52	48
N	48	49	50	51	52	53	54	55	56	57	58	59
N_1	10	15	17	6	1	18	1	8	15	20	22	17
N_2	16	0	11	0	19	9	7	23	21	2	16	2
N	60	61	62	63	64	65	66	67	68	69	70	71
N_1	3	8	3	13	20	10	6	22	5	13	5	18
N_2	9	14	12	7	11	4	21	4	14	19	23	12

Tabelle 6.4: Abbildungstabelle für die Nummern der Referenztetraeder.

Vatertetraeders die Nummern N_1 und N_2 der Kindtetraeder ermittelt werden. Es zeigt sich, daß diese Abbildung sehr erratisch ist und keine einfache geschlossene Formel angegeben werden kann. Daher geben wir die gesamte Abbildung in Tabelle 6.4 an.

Isoflächendreiecke innerhalb eines Tetraeders

Nachdem wir nun alle Tetraeder sortiert haben, bleibt nur noch übrig die Isoflächendreiecke innerhalb eines jeden Tetraeders zu sortieren. Dieser Fall tritt nur selten für fein aufgelöste Datensätze und große Unterschiede zwischen den Isowerten auf. Aber unser Multiskalen–Algorithmus versucht schon auf groben Tetraedern die Isoflächen zu extrahieren und dabei schneiden oft mehrere Isoflächen einen gegebenen Tetraeder was diese Sortierung notwendig macht.

Sei nun $\{i_1, \ldots, i_m\}$ die sortierte Menge von Isowerten startend mit dem kleinten i_1 und endend mit dem größten i_m. Wir wollen annehmen, daß für die sortierte Teilmenge $\{i_j, \ldots, i_k\}$ aller Isowerte die entsprechenden Isoflächen den momentanten Tetraeder schneiden. Dann müssen die Isoflächen entweder startend mit dem kleinsten Isowert i_j und endend mit dem größten Isowert i_k oder in umgekehrter Reihenfolge gezeichnet werden. Nachdem lineare Interpolation in jedem Tetraeder verwendet wird, sind alle Isoflächen parallel zueinander. Die Reihenfolge der Isoflächen ist daher durch das Skalarprodukt aus der Normale der Isofläche mit dem Bilckvektor bestimmt. Die Normale der Isofläche ist allerding gerade der Gradient g der linearen Funktion, die durch die Datenwerte an den Eckpunkten des Tetraeders aufgespannt wird. Wir haben daher den Sortiertest: falls $v \cdot g > 0$ muß der größte Isowert zuerst gezeichnet werden, falls $v \cdot g < 0$ der kleinste. Für $v \cdot g = 0$ sind die Isoflächendreiecke parallel zum

Blickvektor und demnach unsichtbar. Wie diese Gradienten schnell berechnet werden können, wird im nächsten Abschnitt gezeigt.

Hierarchische Gradientenberechnung

Wie wir im vorigen Abschnitt gesehen haben, erfordert die Transparenzsortierung die Datengradienten. Diese Gradienten können im Prinzip vorab berechnet werden, aber sie benötigen sehr viel Speicher. Nachdem die Zahl der Tetraeder im Gitter etwa sechs mal die Zahl der Knoten ist, wäre der Speicherbedarf etwa $6 \cdot 3n^3$ Gleitpunktzahlen zusätzlich zu den n^3 Datenwerten. Daher ist es empfehlsam, diese Gradienten während des Baumdurchlaufs zu berechnen, aber in unbedachter Implementierung würde die Gradientenberechnung die Gesamt–Extraktionszeit dominieren. Wir zeigen nun, wie diese Gradienten sehr effizient hierarchisch berechnet werden können.

Wir wollen uns daran erinnern, daß der Gradient g auf dem Tetraeder S für eine lineare Funktion $f(x, y, z) = ax + by + cz + d$ durch

$$g = \nabla f = \begin{pmatrix} a \\ b \\ c \end{pmatrix}.$$

gegeben ist. Die Koeffizienten a, b, c können für Datenwerte $f_1, \ldots, f_4$ an den entsprechenden Knoten $(x_1, y_1, z_1), \ldots, (x_4, y_4, z_4)$ durch die Lösung des Gleichungssystems.

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}.$$

berechnet werden. Basierend auf der Lösung dieses Systems wollen wir nun die Gradienten g_1, g_2 der zwei Kindtetraeder berechnen. Wir betrachten zunächst das erste Kind S_1 (siehe Abb. 6.9). Die Koeffizienten a_1, b_1, c_1 von g_1 sind durch das folgende System

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_{ref} & y_{ref} & z_{ref} & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_{ref} \end{pmatrix},$$

gegeben, wobei f_{ref} der Datenwert am Verfeinerungsknoten ist. Wir wollen nun das erste System mittels Ersetzung der vierten Zeile durch die Summe der ersten und vierten Zeile dividiert durch zwei ersetzen:

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ \frac{x_1+x_4}{2} & \frac{y_1+y_4}{2} & \frac{z_1+z_4}{2} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \frac{f_1+f_4}{2} \end{pmatrix}.$$

Nachdem $\mathbf{x}_{ref} = (\mathbf{x}_1 + \mathbf{x}_4)/2$ ist, sehen wir daß das System für das Kind sich vom System für den Vater nur in der vierten Komponente der rechten Seite

unterscheidet. Sei nun die Inverse obiger Matrix gegeben durch (w_{ij}). Dann haben wir für a

$$a = w_{11}f_1 + w_{12}f_2 + w_{13}f_3 + w_{14}(f_1 + f_4)/2$$

und für a_1

$$a_1 = w_{11}f_1 + w_{12}f_2 + w_{13}f_3 + w_{14}f_{ref}.$$

Differenzenbildung liefert

$$a_1 = a + w_{14}(f_{ref} - (f_1 + f_4)/2).$$

Indem dieser Schritt für b, c und d wiederholt wird, kommen wir zu

$$\begin{pmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + (f_{ref} - (f_1 + f_4)/2) \begin{pmatrix} w_{14} \\ w_{24} \\ w_{34} \\ w_{44} \end{pmatrix}.$$

In komplett analoger Weise kann der Gradient g_2 von S_2 aus g durch Ersetzung von w_{i4} mit w_{i1} ermittelt werden. Nun müssen nur die entsprechenden w_{ij} für alle 72 Referenztetraeder berechnet und abgespeichert werden. Die Nummern der Referenztetraeder können während des Baumdurchlaufs mit Hilfe des Numerierungsschemas des vorhergehenden Abschnitts berechnet werden.

Beispiele

Abbildung A.16 zeigt drei Isoflächen des Buckyball Datensatzes für äquidistante Isowerte von 0.05, 0.15 and 0.25. Die Farben der Isoflächen sind blau, rot und grün (from außen nach innen) mit Transparenzen von 0.3, 0.5, und 0.7. Die Fehlerschranken ε der sechs Bilder sind 0.0, 0.01, 0.02, 0.04, 0.08, 0.16 und 0.32. Die Zahl der Dreiecke sind 1775762, 979730, 438042, 277698, 171465, and 92309.

Das zweite Beispiel in Abb. A.15 zeigt die Elektronendichte um die Kappe einer Nanoröhre[6]. Zusätzlich zu den Isoflächen sind die verschiedenen Atome des Moleküls dargestellt (Wasserstoff in blau, Bor in rot, und Nitrat in grün). Die Isoflächen sind 0.05 (gelb), 0.2 (grün) und 0.35 (blau). DIe ε–Werte sind 0.0, 0.01, 0.02, 0.04, 0.08 und 0.12 resultierend in 230452, 175054, 143126, 96327, 55933 und 42630 Dreiecke.

Die Atome werden als kleine Texturen gezeichnet und während des Baumdurchlaufs an die entsprechende Stelle gesetzt. In diesem Beispiel ist es nicht nötig die Texturen über Tetraedergrenzen hinweg aufzusplitten nachdem die Texturen klein genug ist und in die Zentren der Tetraeder gesetzt wurden. In anderen Fällen mag es jedoch notwendig sein solche Texturen zwischen verschiedenen Tetraedern aufzusplitten.

Die kombinierte Zeichnungs- und Extraktionszeit ist etwa 280000 Dreiecke pro Sekunde auf einer SGI Onyx^2. Im Vergleich zum gleichen Algorithmus für nicht transparente Isoflächen zeigt sich somit, daß die Zeit zum Sortieren moderat ist und die Gesamtleistung nicht signifikant verschlechtert.

[6]Copyright von von A. Caglar, Universität Bonn

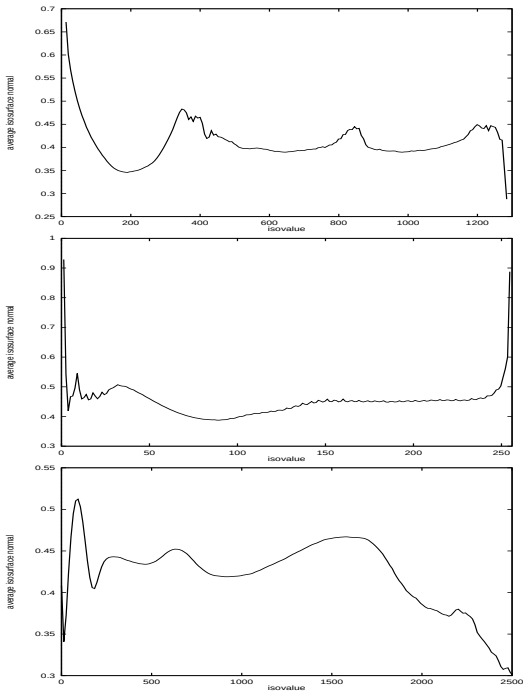


Abbildung 6.10: Die Größe der mittleren Isoflächennormalen $\bar{g}$ für den Zahn- (oben links), Schaf- (oben rechts) und Knie–Datensatz (unten).

Wahl der Isowerte

In vielen Anwendungen gibt es eine gute Korrespondenz zwischen Datenwerten und brauchbaren Isowerten. Zum Beispiel haben bei medizinischen Aufnahmen Knochen, Gewebe und Blutgefäße bestimmte bekannte Reflektivitäten, die vom Aufnahmesystem zurückgeliefert werden. In vielen anderen Anwendungen kann es sein, daß diese Korrespondenz unbekannt ist oder sich zwischen den Datensätzen ändert. Sicherlich ist Ausprobieren oft der beste Weg brauchbare Isowerte zu finden. In manchen Fällen kann es aber auch hilfreich sein, dem Benutzer erste Schätzungen guter Isoflächen anzubieten und ihn entscheiden zu lassen, welche nützlich sind und welche nicht.

Im Vergleich zum Design von Transferfunktionen bei direkten Volumendarstellungen ist die Zahl der Freiheitsgrade bei transparenter Isoflächen–Extraktion ziemlich klein. Es müssen nur ein paar wenige Isowerte, Farben und Transparenzen ausgewählt werden. Wohl der naheliegendste Ansatz brauchbare Isowerte zu finden, ist es, das Gradientenfeld des Datensatzes zu betrachten. Wir wollen nun das (diskrete) Mittel der Normalen einer Isofläche $\bar{g}(i)$

$$\bar{g}(i) = \frac{\sum_{S_j : s(i) \cap S_j \neq \emptyset} area(s(i) \cap S_j) \cdot |g(S_j)|}{\sum_{S_j : s(i) \cap S_j \neq \emptyset} area(s(i) \cap S_j)}$$

definieren, wobei i der Isowert, $s(i)$ die triangulierte Isofläche und $g(S_j)$ der

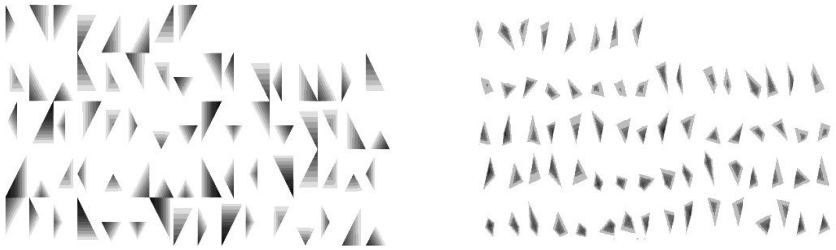


Abbildung 6.11: Texturen für Volumendarstellungen mit Tetraeder–Splatting.

Datengradient auf S_j (was der Normale der Isofläche entspricht) ist. Die lokalen Minima und Maxima von $\bar{g}(i)$ charakterisieren dann mögliche Isowerte. Die Maxima trennen homogene Gebiete und die Minima stellen die Zentren dieser homogenen Gebiete dar.

Wir wollen nun ein paar konkrete Graphen $\bar{g}(i)$ für drei Datensätze[7] [111]) betrachten. Diese Datensätze, CT bzw. MRI scans eines Zahns, das Herz eines Schafs und eines Knies zählen als Benchmarks im Design von Transferfunktionen. Die berechneten $\bar{g}(i)$ sind in Abb. 6.10 dargestellt. Für den Zahn–Datensatz gibt es drei klar sichtbare Minima. Das linke Minimum entspricht der zylindrischen äußeren Schale des Mediums, in das der Zahn gesetzt wurde bevor er gescannt wurde. Die anderen zwei Minima geben die Oberfläche des Zahns und den Zahnschmelz (Abb. A.17 oben). Der Schaf–Datensatz zeigt nur ein sichtbares Minimum, welches der Oberfläche des Herzens und den inneren Blutgefäßen entspricht (Abb. A.17 mitte). Der Knie–Datensatz zeigt drei klare Minima trotz des hohen Grades an Rauschen in dem Datensatz, die in etwa der Haut, dem Muskelgewebe und dem Knochen entsprechen (Abb. A.17 unten). Obwohl wir es hier nicht getan haben, sollten solch verrauschte Daten vor der Isoflächen–Extraktion eigentlich geglättet werden.

6.5.2 Splatting

Das Numerierungsschema des vorigen Abschnittes kann auch dazu verwendet werden, Splats der einelnen Referenztetraeder für direkte Volumendarstellungen anzufertigen. Abb. 6.11 zeigt solche Texturen für die 72 Referenztetraeder für den Fall stückweise konstanter Interpolation. Eine Volumendarstellung entsteht dann durch gewichtete back–to–front Akkumulation dieser Splats. Ergebnisse von Volumendarstellungen für verschiedene Datensätze finden sich in Abb. A.18. Für weitere Erläuterungen dieser Technik siehe z.B. [89] oder [93].

[7] Copyright von Bill Lorensen, General Electric

Kapitel 7

Ausblick und Schlußbemerkungen

In dieser Arbeit wurden verschiedene Multiskalenmethoden zur interaktiven Verarbeitung großer Datenmengen vorgestellt. Ausgangspunkt hierfür war eine hierarchische Zerlegung des Grundgebiets. Basierend auf einer solchen Zerlegung konnte dann ein gegebener Datensatz mittels eines hierarchischen Interpolationsverferfahren auf verschiedenen Skalenstufen dargestellt werden. Effiziente Approximationen konnten dann dynamisch durch adaptive Verfeinerung, die von saturierten Fehlerindikatoren gesteuert wurde, erzeugt werden. Die Speicherung und Kompression solcher Multiskalenmodelle erfolgte im Fall von Bisektionstriangulierungen mit Hilfe von Bitcodes, raumfüllenden Kurven und relativen Sprungzeigern. Verschiedenste Anwendungen von der interaktiven graphischen Darstellung digitaler Höhenmodelle über Isoflächen–Extraktion bis hin zu Volumendarstellungen haben dann die vielfältigen Einsatzmöglichkeiten der entwickelten Techniken aufzeigt und die Vorteile gegenüber bestehenden Verfahren unterstrichen.

Wichtig hierbei war daß die Verfahren von ihrer Komplexität unabhängig von der Gesamtdatenmenge sind und ihr Aufwand proportional zur Größe der Ausgabemenge (z.B. der gezeichneten Dreiecke) ist. Weiterhin ist bei der Entwicklung geeigneter Datenstrukturen von zentraler Bedeutung daß die Verfahren auf den komprimierten Daten arbeiten können. Auf diese Weise skalieren die Verfahren hervorragend sowohl bezüglich ihrer Laufzeit- als auch der Speicherkomplexität. Dies macht eine interaktive Arbeit mit sehr großen Datenmengen erst möglich.

Neben der Datenkompression und Visualisierung sind weitere Anwendungsgebiete von Multiskalenmethoden:

- Skalenanalyse: Wavelet–Methoden erlauben im Gegensatz zur Fourier–Analyse bekannterweise die Lokalisierung sowohl in der Frequenz als auch im Raum und eignen sich deshalb hervorragend für eine lokalisierte Frequenzanalyse von Daten.

- Strukturerkennung: Weiterhin eignen sich Wavelets auch besonders gut zur Kantenerkennung in digitalen Bildern [75].
- Parameterableitung: Bei digitalen Höhenmodellen sind z.B. Neigung und Krümmung des Reliefs sogenannte Primärattribute, welche in viele Berechnungen eingehen. Neigung und Krümmung sind jedoch klar abhängig von der betrachteten Skala und können erst durch Multiskalenmodelle geeignet skalenabhängig definiert werden.
- Sichtbarkeit: Die gegenseitige Sichtbarkeit zweier Punkte auf einem digitalen Höhenmodell ist Grundproblem vieler Anwendungen und kann effizient durch hierarchische Techniken ermittelt werden [30].
- Abstandsberechnung: Ein dazu verwandtes Problem ist die Berechnung des Abstandes zweier Punkte auf einer Triangulierung, was ebenfalls hierarchisch in optimaler Laufzeit geschehen kann [91].
- Glättung: Anisotrope Diffusionsverfahren basierend auf adaptiv hierarchischen Zerlegungen werden z.B. zur Rauschunterdrückung und Segmentierung von digitalen Bildern eigesetzt [59].
- Progressive Verfeinerung: Wie schon erwähnt werden Multiskalenmodelle vor allem in verteilten Anwendungen, wie z.B. Videokonferenzen, eingesetzt [35, 43, 108].
- Lösung partieller Differentialgleichungen: Natürlich und nicht zuletzt sind adaptiv hierarchische Finite Elemente Verfahren in Kombination mit Mehrgitterverfahren optimal zur Diskretisierung und Lösung partieller Differentialgleichungen [4, 66, 67, 102, 149].

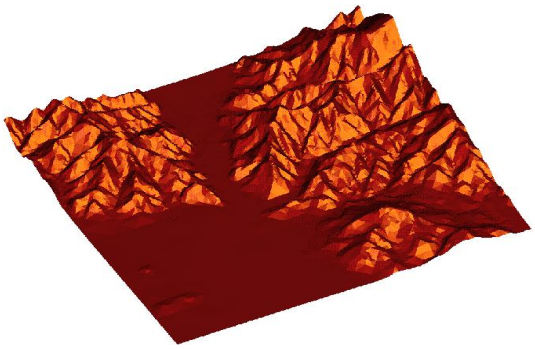
Mit entsprechenden Modifikationen können die entwickelten Datenstrukturen und Verfahren sicherlich auch in diesen Bereichen eingesetzt werden, was ein möglicher Gegenstand zukünftiger Forschung sein könnte.

Anhang A

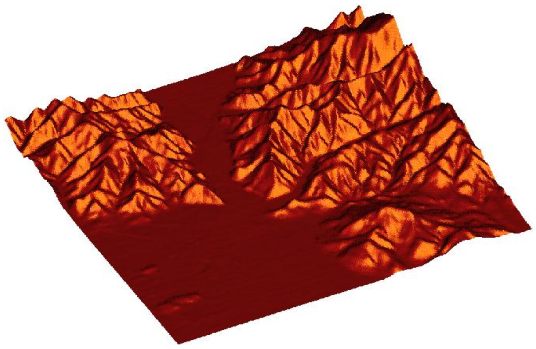
Farbbilder

Aus praktischen Gründen sind alle Farbbilder in diesem Anhang zusammengefaßt. Um ein Auffinden der zugehörigen Textstellen zu erleichten, sind die einzelnen Bilder zusammen mit kurzen Beschreibungen und den dazu gehörenden Kapiteln in folgender Tabelle aufgelistet:

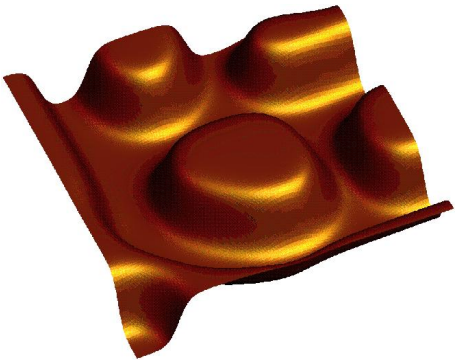
Nummer	Beschreibung	Kapitel
A.1	Testdatensätze zum Vergleich von Fehlerindikatoren	4.5
A.2	Raumfüllende Kurven zur Bildkompression	5.4
A.3	Global adaptive digitale Höhenmodelle	6.1.2
A.4	Zoom durch verschiedene digitale Höhenmodelle	6.1.3
A.5,A.6	Digitale Höhenmodelle mit Fokussierung	6.1.4
A.7,A.8	Digitale Höhenmodelle mit Topologie–Erhaltung	6.1.5
A.9	Fraktale Interpolation	6.2
A.10	Adaptive Schnittbildung	6.3
A.11,A.12	Isoflächen–Extraktion mit Topologie–Erhaltung	6.4.3
A.13	Isoflächen–Extraktion mit Topologie–Vereinfachung	6.4.4
A.14	Parallelisierung der Isoflächen–Extraktion	6.4.5
A.15,A.16	Extraktion transparenter Isoflächen	6.5.1
A.17	Testdatensätze zur Auswahl von Isowerten	6.5.1
A.18	Volumendarstellung durch Splatting	6.5.2



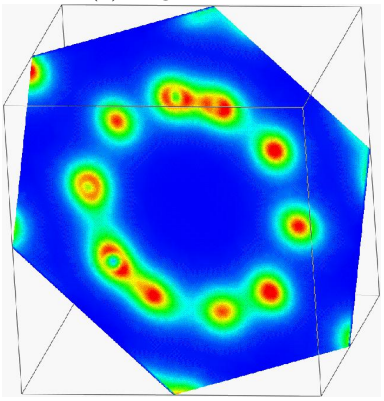
(a) Approximation der geographischen Karte



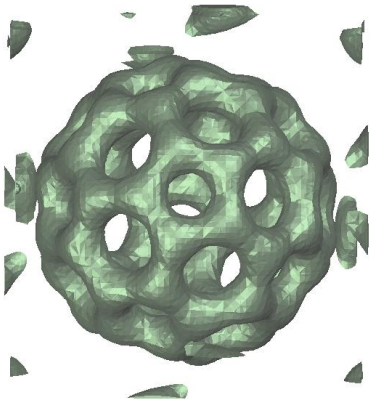
(b) Originaldaten



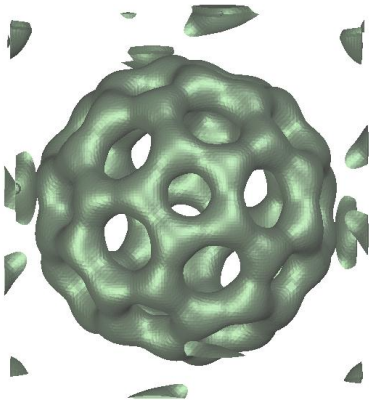
(c) Cahn–Hilliard Simulation



(d) Schnitt durch den Buckyball



(e) Approximative Isofläche



(f) Originaldaten

Abbildung A.1: Effizienzvergleich der Fehlerschätzer: Verwendete Datensätze.

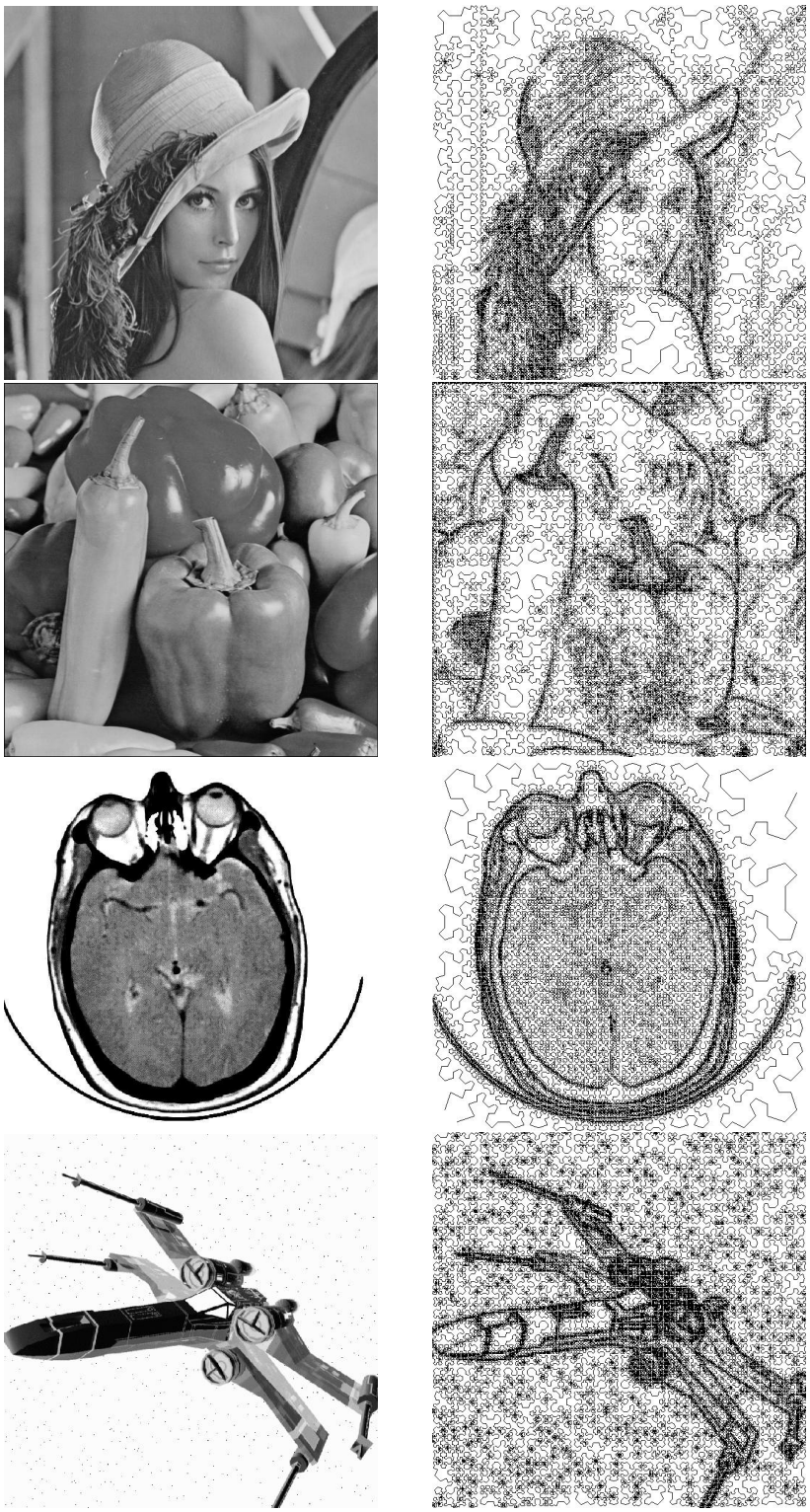


Abbildung A.2: Raumfüllende Kurven zur Bildkompression: Verwendete Bilddaten sowie und zugehörige Sierpinski–Kurven.

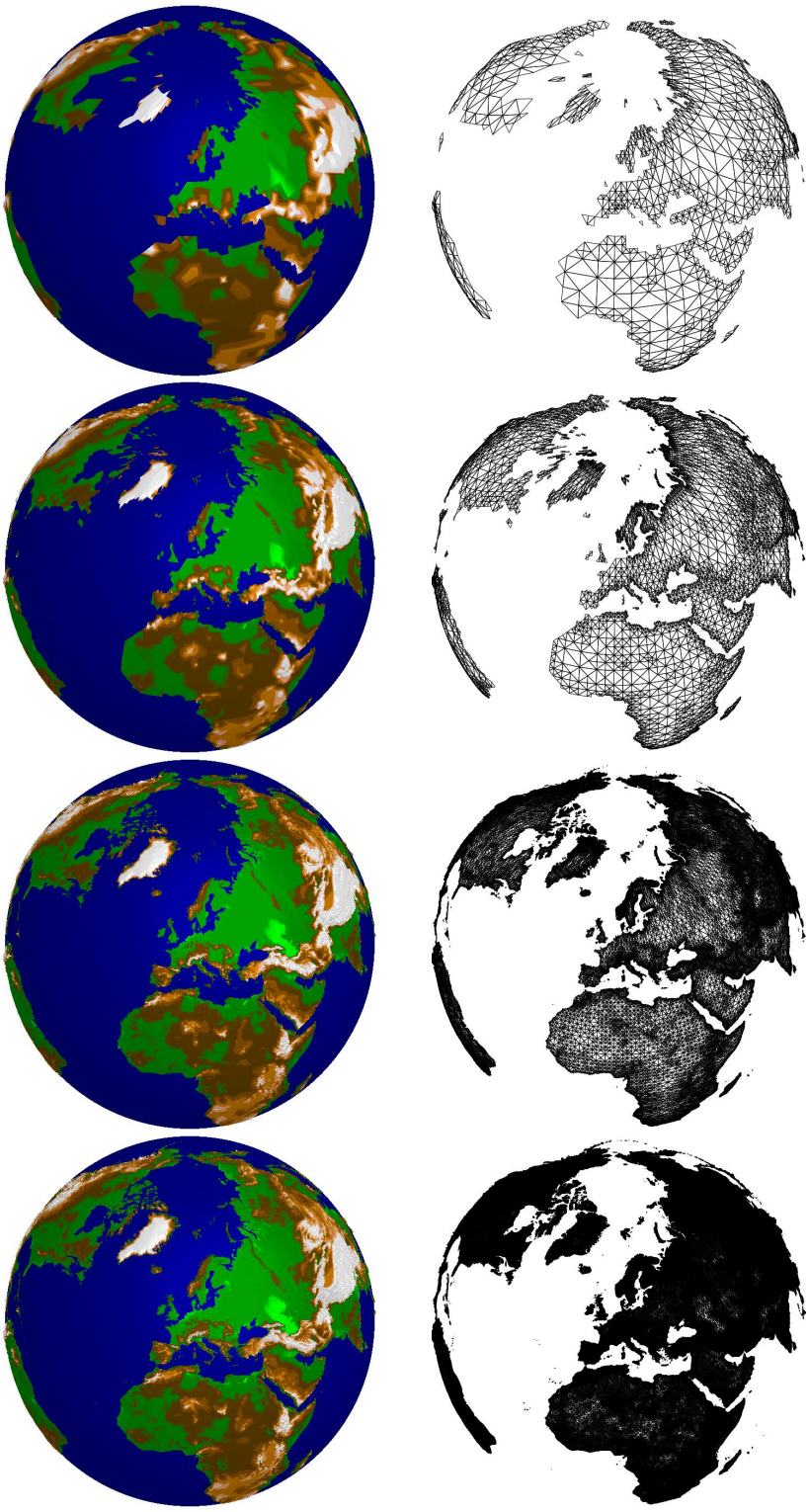


Abbildung A.3: Approximative Höhenmodelle und zugehörige adaptive Triangulierungen für Fehlerschranken ε_0 von 16384, 4096, 1024 und 256.

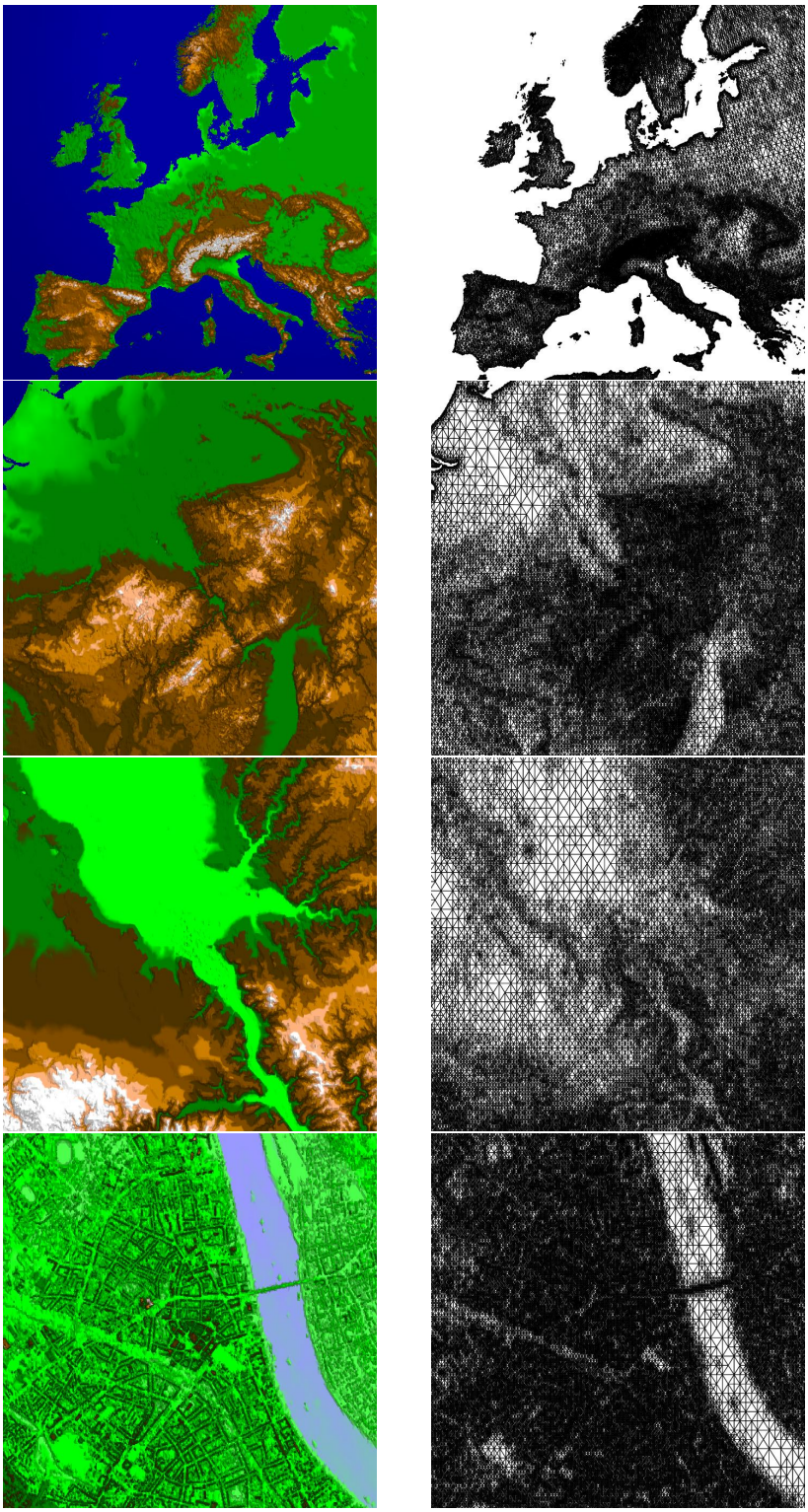


Abbildung A.4: Approximative Höhenmodelle und zugehörige adaptive Triangulierungen für zoom Faktoren von 4, 32, 512 und 4096.

Abbildung A.5: Adaptive Triangulierungen einer geographischen Karte ohne Fokus auf Küstenlinien.

Abbildung A.6: Adaptive Triangulierungen einer geographischen Karte mit Fokus auf Küstenlinien.

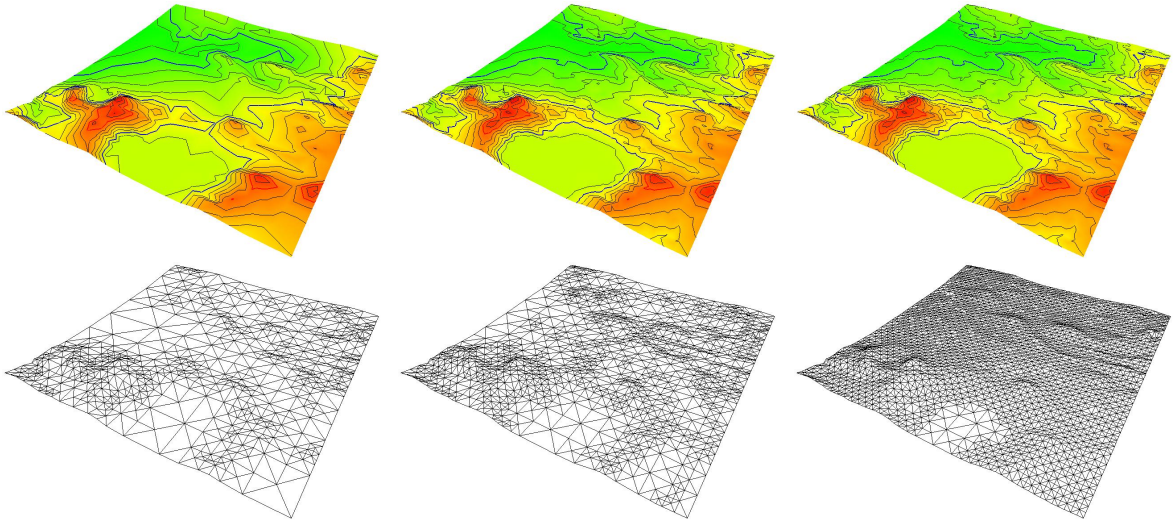


Abbildung A.7: Approximative Höhenmodelle mit Isolinien und zugehörige adaptive Triangulierungen ohne Topologie–Erhaltung.

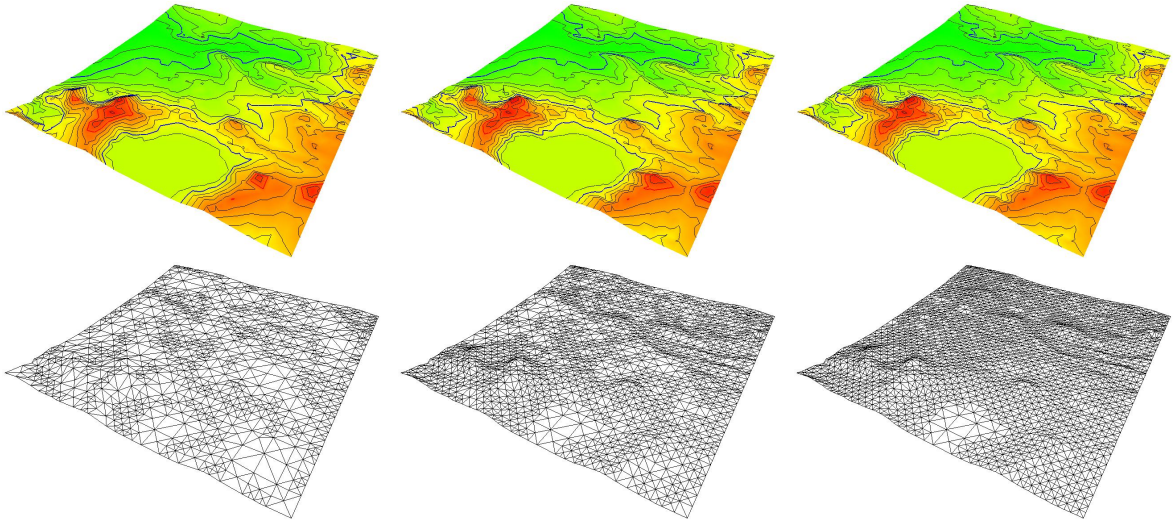


Abbildung A.8: Approximative Höhenmodelle mit Isolinien und zugehörige adaptive Triangulierungen mit Topologie–Erhaltung.

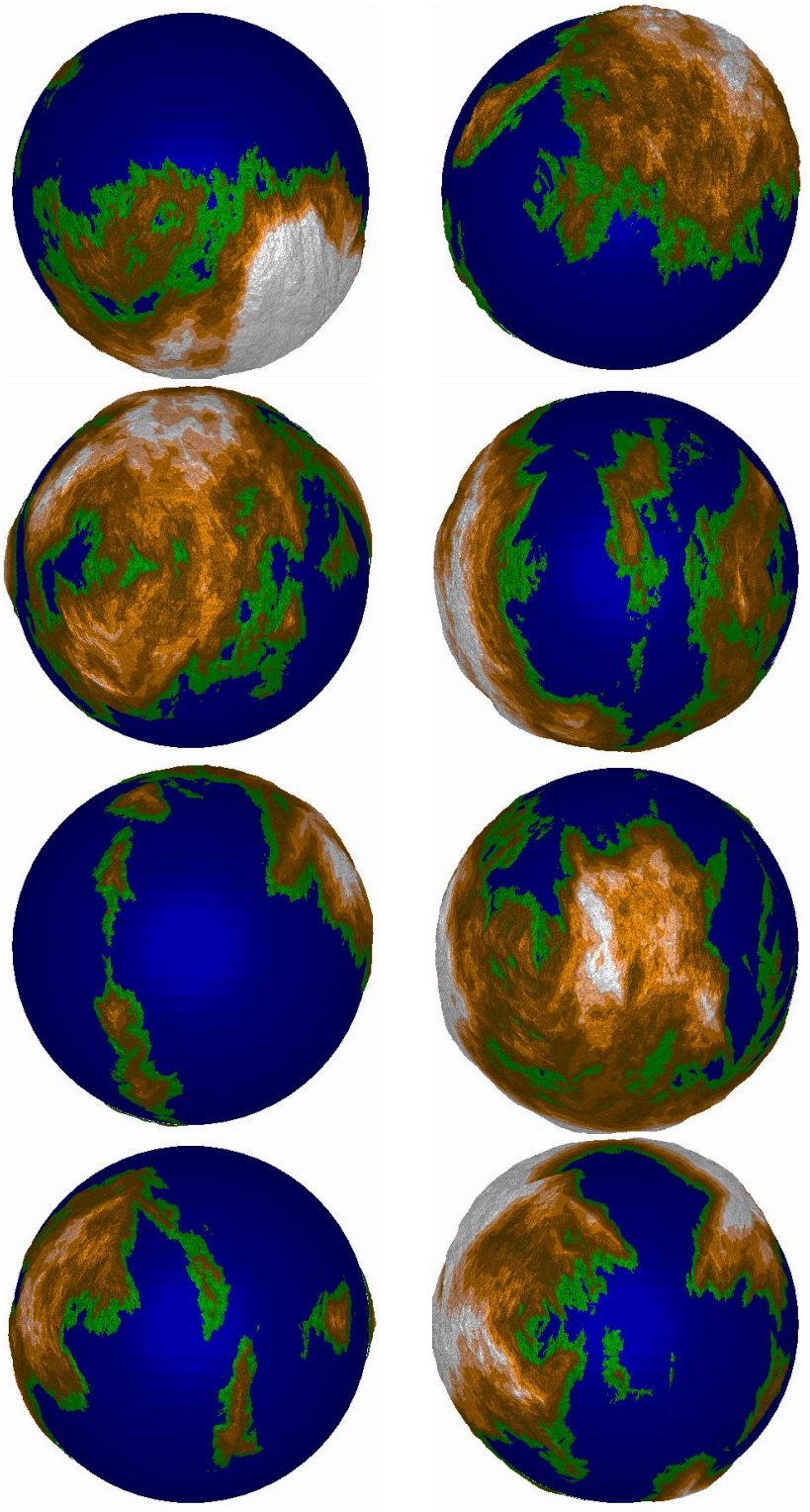


Abbildung A.9: Fraktale Welten erzeugt durch Dreiecksbisektion für verschiedene Startwerte des Zufallszahlengenerators.

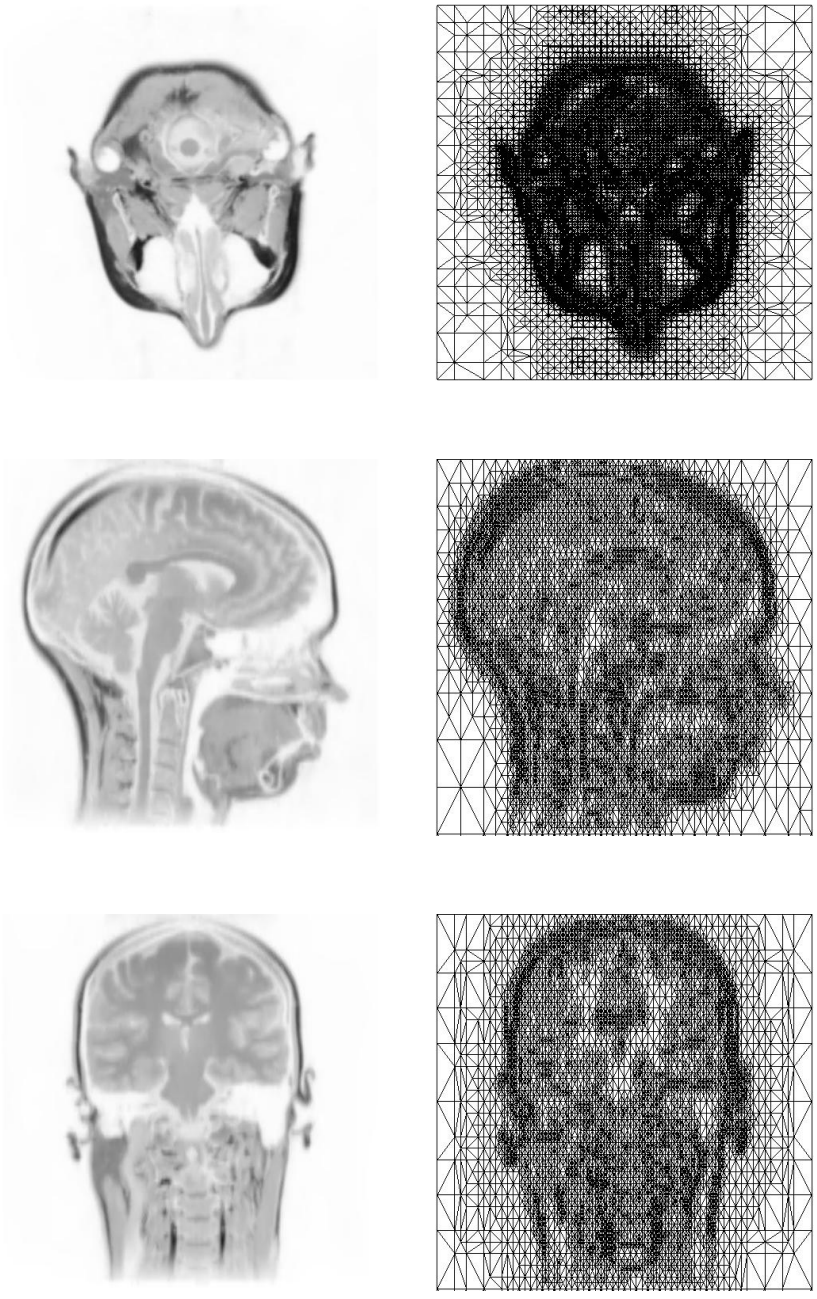


Abbildung A.10: Verschiedene adaptive Schnitte und zugehörige Triangulierungen durch einen medizinischen Bilddatensatz.

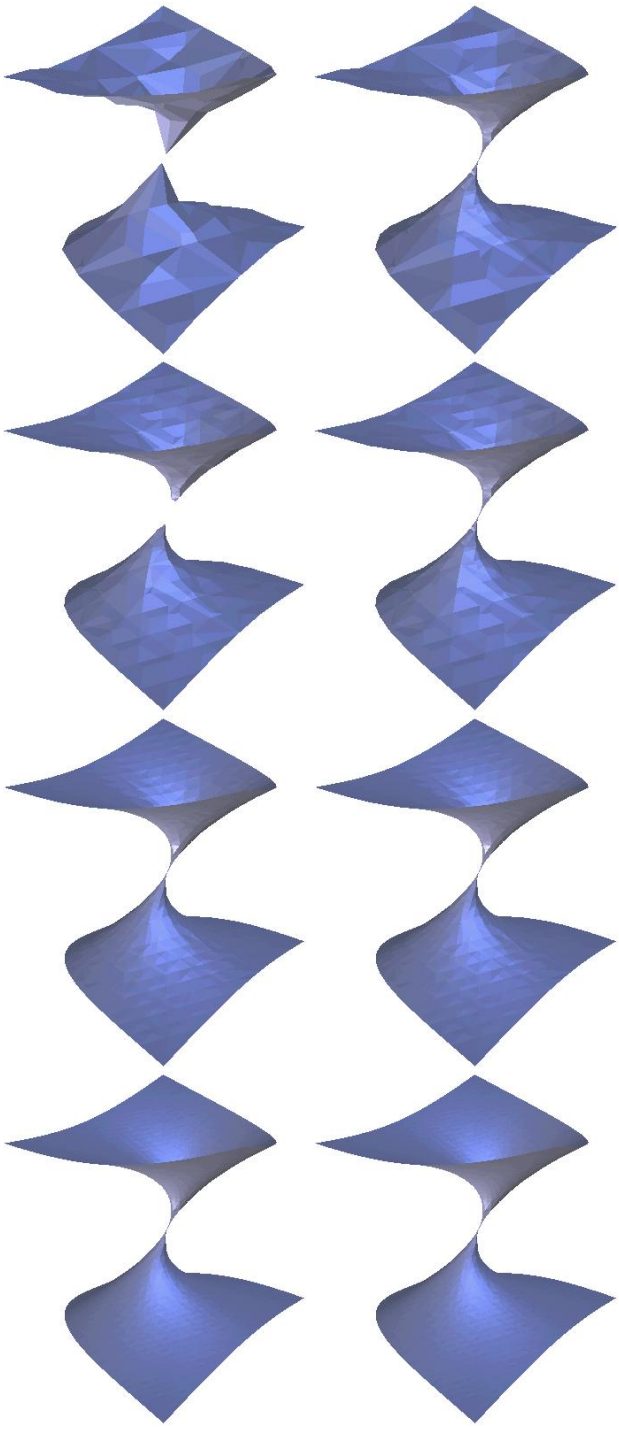


Abbildung A.11: Isoflächen einer algebraischen Funktion ohne (linke Reihe) und mit (rechte Reihe) Topologie–Erhaltung für verschiedene Fehlerschranken.

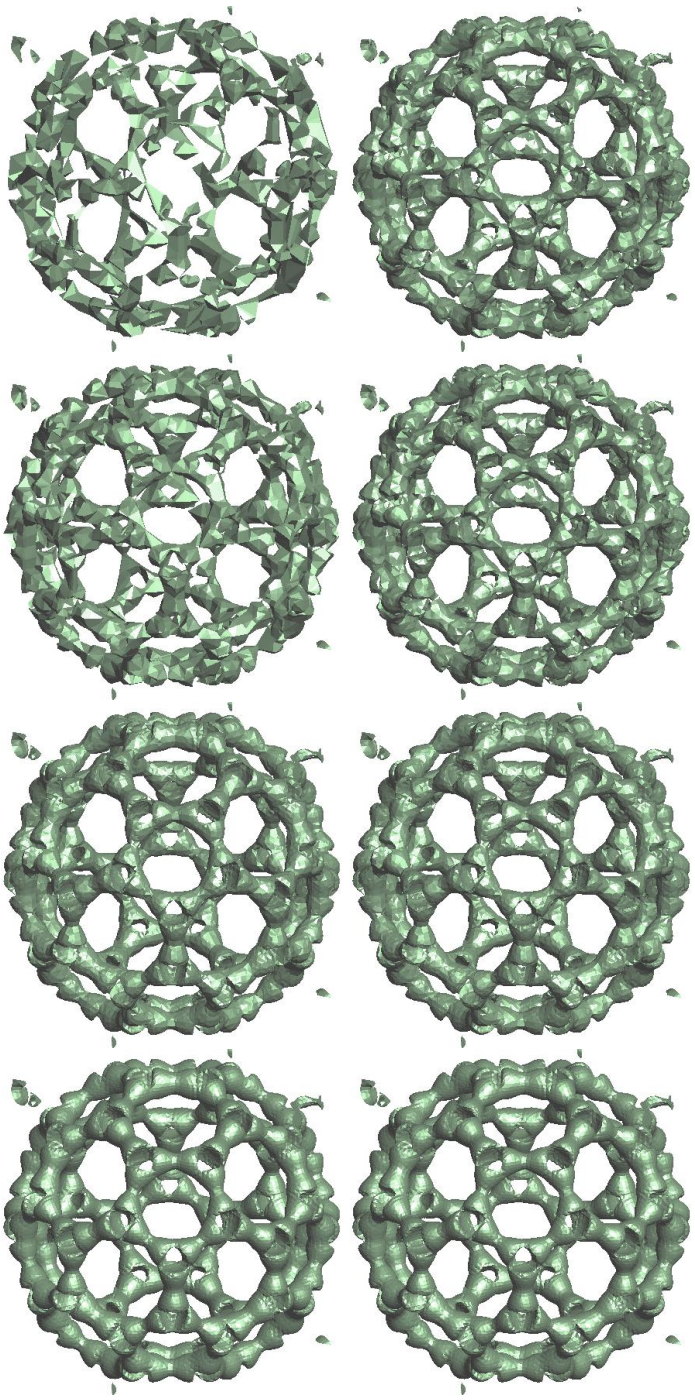


Abbildung A.12: Isoflächen des Buckyballs–Datensatzes ohne (linke Reihe) und mit (rechte Reihe) Topologie–Erhaltung für verschiedene Fehlerschranken.

Abbildung A.13: Isoflächen des Hummer–Datensatzes: Original (oben links), mit Topologie–Erhaltung (oben rechts) und kontrollierter Topologie–Vereinfachung für verschiedene Schranken δ (unten links und rechts).

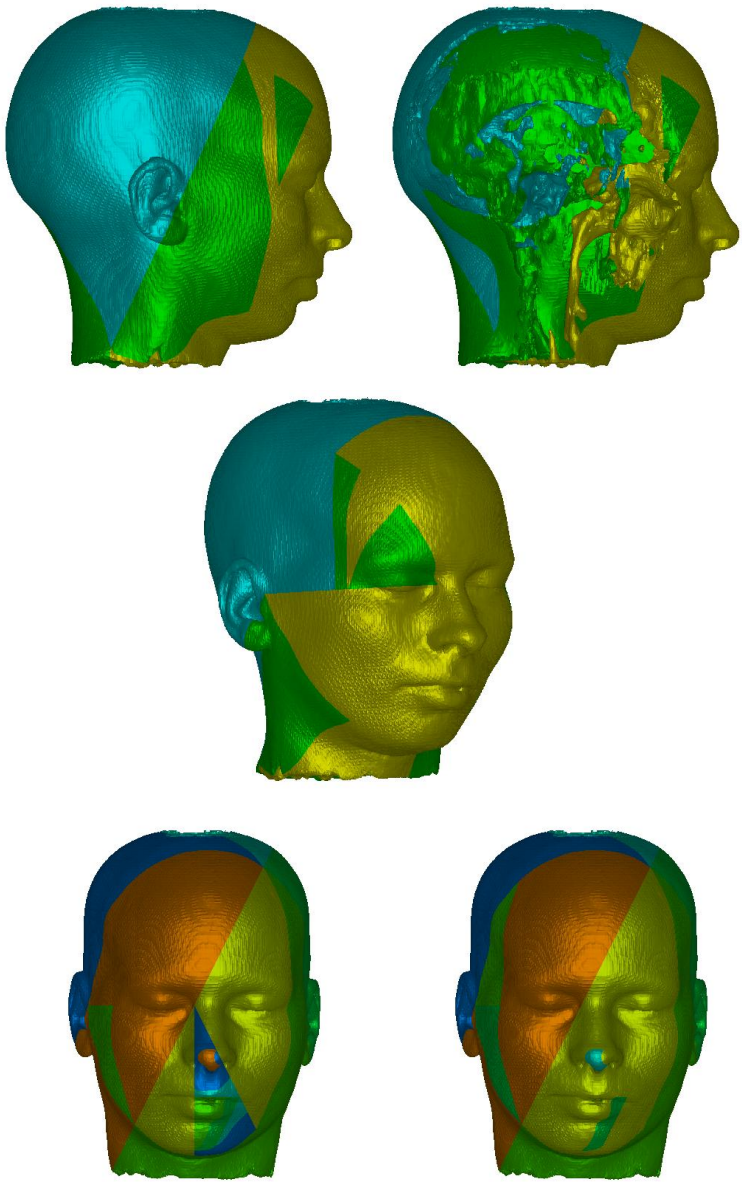


Abbildung A.14: Verschiedene parallel extrahierte Isoflächen eines menschlichen Kopfes. Farben zeigen die Bereiche an, die von den jeweiligen Prozessoren extrahiert wurden.

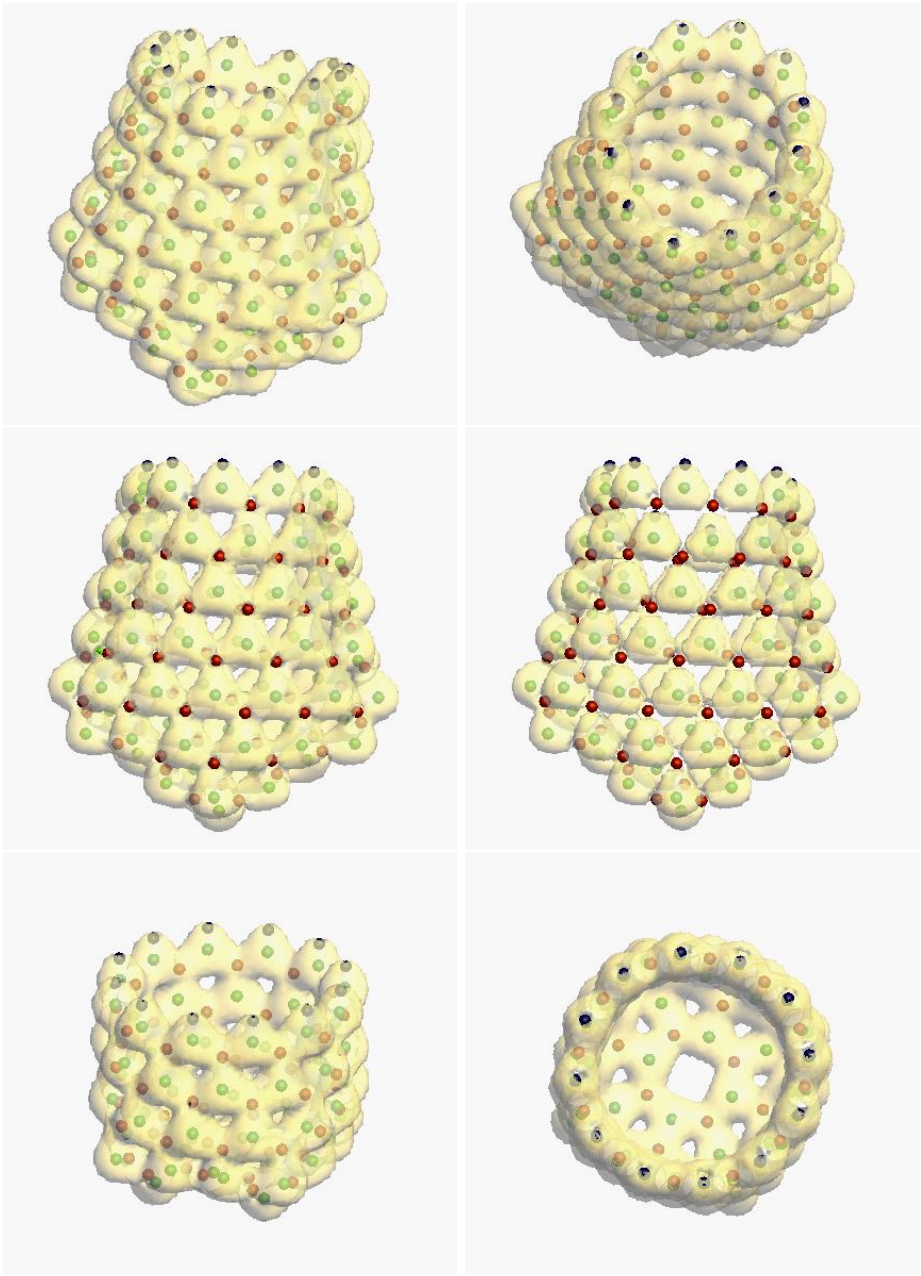


Abbildung A.15: Transparente Isoflächen mit zugehörigen Atompositionen zweier Nanokappen.

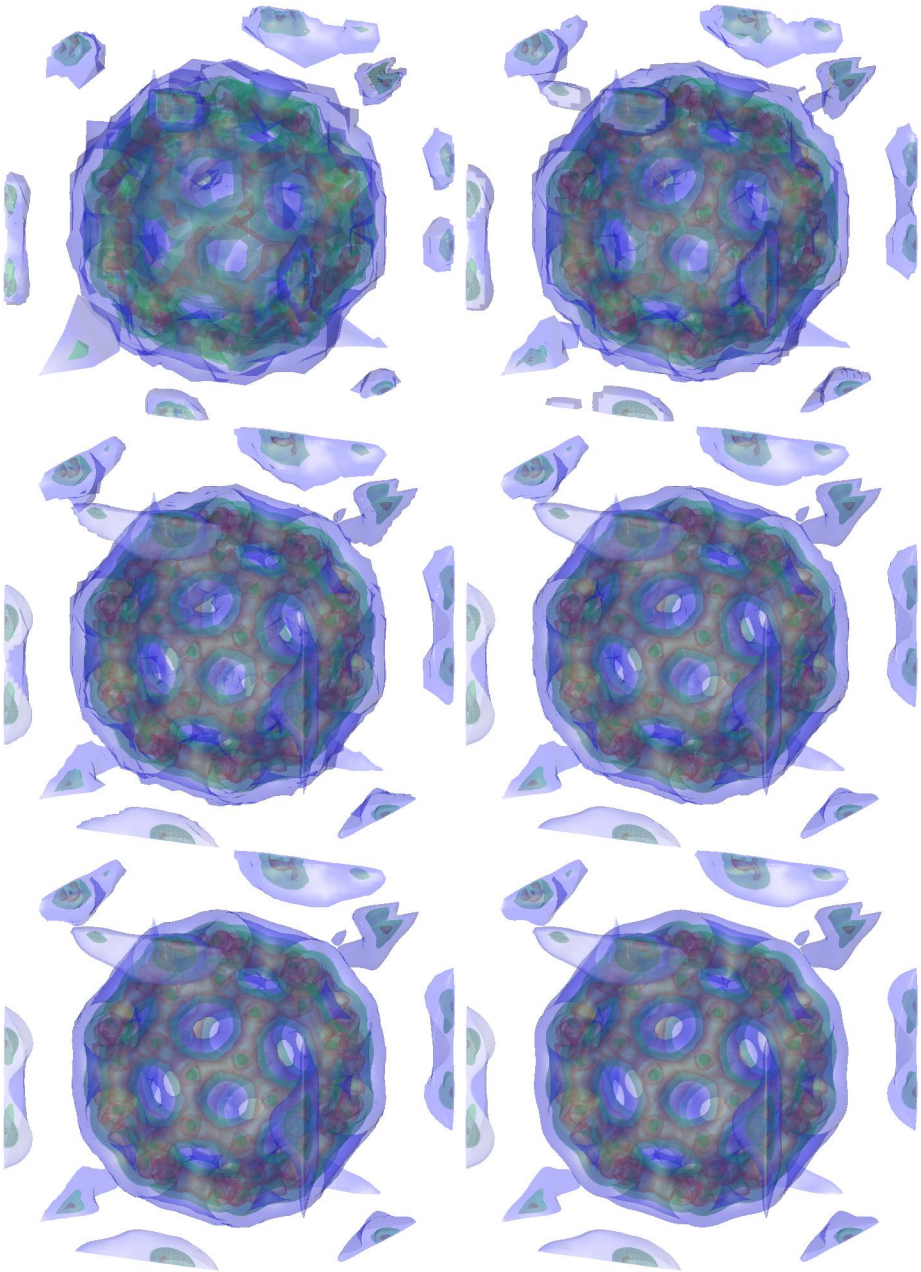


Abbildung A.16: Mehrere transparente Isoflächen des Buckyball Datensatzes.

Abbildung A.17: Mehrere transparente Isoflächen der Zahn-, Schaf- und Knie–Datensätze nach automatischer Isowert–Auswahl.

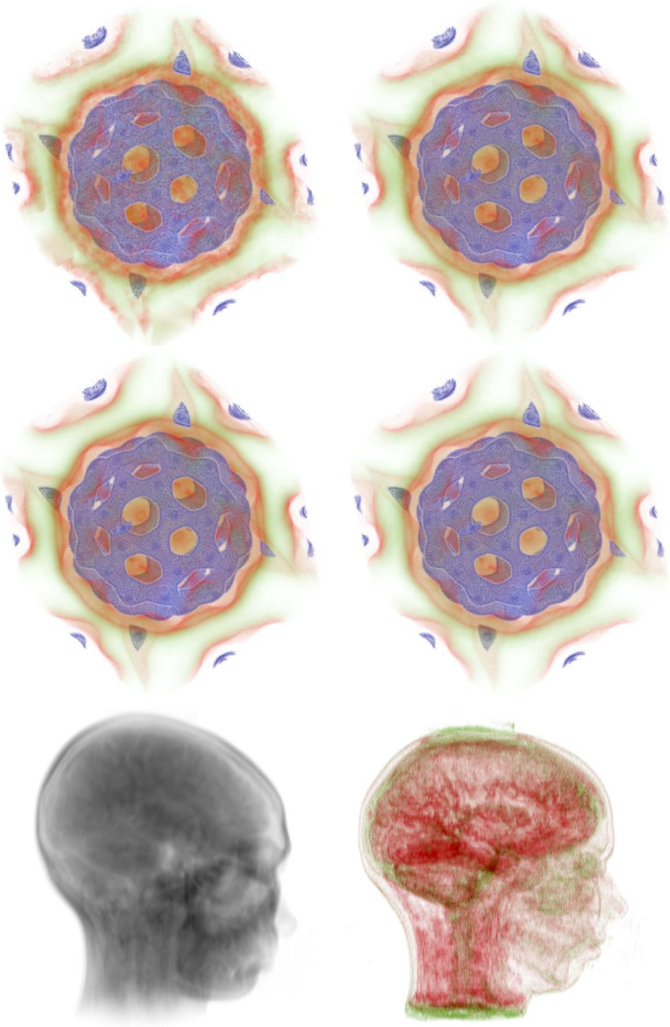


Abbildung A.18: Volumendarstellungen des Buckyballs für verschiedene Fehlertoleranzen, sowie eines Kopfes mit verschiedenen Transferfunktionen.

Literaturverzeichnis

[1] C. Andujar, D. Ayala, P. Brunet, R. Joan-Arinyo, J. Sole: Automatic generation of multiresolution boundary representations: *Computer Graphics Forum* 15(3):87–96, 1996.

[2] D. Arnold, A. Mukherjee, L. Pouly: Locally adapted tetrahedral meshes using bisection, *SIAM J. Sci. Comp.* 22(2):431–448, 2000.

[3] E. Bänsch: Local mesh refinement in 2 and 3 dimensions, *IMPACT Comp. Sci. Eng.* 3:181–191, 1991.

[4] R. Bank, T. Dupont, H. Yserentant: The hierarchical basis multigrid method, *Numer. Math* 52:427–458, 1988.

[5] C. Bajaj, D. Schikore: Topology preserving data simplification with error bounds. *Computers & Graphics* 22(1):3–12, 1998.

[6] C. Bajaj, V. Pascucci, D. Thompson, X.Y. Zhang: Parallel accelerated isocontouring for out–of–core visualization, in *Proc. IEEE Parallel Visualization and Graphics Symposium*, pp. 97–104, IEEE Computer Society Press, 1999.

[7] L. Balmelli, J. Kovacevic, M. Vetterli: Quadtrees for embedded surface visualization: constraints and efficient data structures, in *Proc. IEEE Int. Conf. Image Processing (ICIP)*, 1999.

[8] A. Barrera, A. Hinojosa: A Hierarchical Method for Representing Terrain Relief, in *Proc. Pecora 9 Symposium*, Sioux Falls, pp. 87–92, 1984.

[9] J. Bartholdi, L. Platzman: Heuristics based on spacefilling curves for combinatorial problems in Euclidean space, *Management Sci.*, 34:291–305, 1988.

[10] D. Bartz, W. Straßer, R. Grosso, T. Ertl: Parallel construction and isosurface extraction of recursive tree structures, in *Proc. WSCG '98*, University of West Bohemia, Plzen, Czech Republic, 1998.

[11] K. Baumann, J. Döllner, K. Hinrichs, O. Kersting: A hybrid, hierarchical data structure for real–time terrain visualization, in *Proc. Computer Graphics International '99*, pp. 85-92, 1999.

[12] J. Bey: Tetrahedral grid refinement, *Computing* 55:355-378, 1995.

[13] E. Bier, M. Stone, K. Pier, W. Buxton, T. DeRose: Toolglass and magic lenses: the see–through interface, *Computer Graphics (SIGGRAPH '93 Proc.)*, pp. 73–80, 1993.

[14] D. Braess: Finite Elemente, Springer, 1992.

[15] H. Bungartz: Concepts for higher order finite elements on sparse grids, in *Houston Journal of Mathematics*, A.V. Ilin, L.R. Scott (eds.), pp. 159-170, Houston, 1996.

[16] A. Certain, J. Popović, T. DeRose, T. Duchamp, D. Salesin, W. Stuetzle: Interactive multiresolution surface viewing, *Computer Graphics (SIGGRAPH '96 Proc.)*, pp. 91–98, 1996.

[17] Z. Chen, W. Tobler: Quadtree representations of digital terrain, in *Proc. Auto–Carto*, vol. 1, pp. 475–484, 1986.

[18] C. Chui, J. Wang: On Compactly Supported Spline Wavelets and a Duality Principle, *Trans. Amer. Math. Soc.* 330:903–915, 1992.

[19] P. Cignoni, C. Montani, R. Scopigno: MagicSphere: an insight tool for 3D data visualization. *Computer Graphics Forum* 13(3):317–328, 1994.

[20] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, R. Scopigno: Multiresolution representation and visualization of volume data, *IEEE Trans. Vis. Comp. Graph.* 3(4):352–369, 1997.

[21] P. Cignoni, E. Puppo, R. Scopigno: Representation and visualization of terrain surfaces at variable resolution, *The Visual Computer* 13(5):199–227, 1997.

[22] P. Cignoni, C. Montani, R. Scopigno: A comparison of mesh simplification algorithms, *Computers & Graphics* 22(1):37–54, 1998.

[23] P. Cignoni, F. Ganovelli, C. Montani, R. Scopigno: Reconstruction of topologically correct and adaptive trilinear isosurfaces, *Computers & Graphics* 24(3):399–418, 2000.

[24] A. Cohen, I. Daubechies, J. Feauveau: Bi–orthogonal bases of compactly supported wavelets, *Comm. Pure Appl. Math.* 45:485–560, 1992.

[25] A. Cohen, W. Dahmen, I. Daubechies, R. DeVore: Tree approximation and encoding, report no. 174, Institut für Geometrie und Praktische Mathematik, RWTH Aachen, 1999.

[26] J. Carnicer, W. Dahmen, J. Peña: Local decomposition of refinable spaces, *Appl. Comp. Harm. Anal.* 3:127–153, 1996.

[27] I. Daubechies: Ten Lectures on Wavelets, SIAM, 1992.

[28] M. De Berg, K. Dobrindt: On the levels of detail in terrain, *Graph. Mod. Im. Proc.* 60(1):1-12, 1998.

[29] L. De Floriani: A pyramidal data structure for triangle–based surface description, *IEEE Comp. Graph. Appl.* 9(2):67–78, 1989.

[30] L. De Floriani, P. Magillo: Visibility computations on hierarchical triangulated terrain models, *Geoinformatica* 1(3):219-250, 1997.

[31] L. De Floriani, E. Puppo, P. Magillo: Geometric Structures and Algorithms for Geographic Information Systems, Chapter 7 in *Handbook of Computational Geometry*, J.R. Sack, J. Urrutia (eds.), Elsevier Science, 1999.

[32] A. Dixon, G. Kirby, D. Wills: Artificial planets with fractal feature specification, *The Visual Computer* 15:147–158, 1999.

[33] D. Douglas, T. Peucker: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, *Canadian Cartographer* 10:112–122.

[34] A. Dreger, M. Gross, J. Schlegel: Multiresolution triangulat B–spline surfaces, in *Proc. Computer Graphics International '98*, pp. 166-177, 1998.

[35] M. Drews, L. Rothenbücher, C. Seibert: Progressive Übertragung von hierarchischen Videodaten, Praktikumsbericht, Institut für Informatik, Universität Bonn, 2000.

[36] M. Duchaineau: Dyadic splines, PHD thesis, dept. of computer science, Univ. California Davis, 1996.

[37] M. Duchaineau, M. Wolinsky, D. Sigeti, M. Miller, C. Aldrich, M. Mineev–Weinstein: ROAMing Terrain: Real–time Optimally Adapting Meshes, in *Proc. IEEE Visualization '97*, IEEE Computer Society Press, 1997.

[38] G. Dutton: A Hierarchical Coordinate System for Geoprocessing and Cartography, Lecture Notes in Earth Sciences 79, Springer, 1999.

[39] N. Dyn, D. Levin, S. Rippa: Data Dependent Triangulations for Piecewise Linear Interpolation, *IMA J. Num. Anal.* 10:137–154, 1990.

[40] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, W. Stuetzle: Multiresolution analysis of arbitrary meshes, *Computer Graphics (SIGGRAPH '95 Proc.)*, pp .173–182, 1995.

[41] J. El-Sana, A. Varshney: Controlled simplification of genus for polygonal models, in *Proc. IEEE Visualization '97*, pp. 403–412, IEEE Computer Society Press, 1997.

[42] M. Ellerbrake: Eine schnelle näherungsweise Lösung des Euklidischen Travelling Salesman Problems mittels raumfüllender Kurven, Diplomarbeit, Institut für Angewandte Mathematik, Universität Bonn, 2000.

[43] K. Engel, R. Grosso, T. Ertl: Progressive iso–surfaces on the web, in *Proc. IEEE Visualization '98*, Late Breaking Hot Topics, IEEE Computer Society Press, 1998.

[44] T. Ertl, R. Westermann, R. Grosso: Multiresolution and hierarchical methods for the visualization of volume data, *Fut. Gen. Comp. Sys.* 15(1):31–42, 1999.

[45] W. Evans, D. Kirkpatrick, G. Townsend: Right Triangular Irregular Networks, Technical Report 97–09, University of Arizona, 1997.

[46] G. Faber: Über stetige Funktionen, *Mathematische Annalen* 66:81–94, 1909.

[47] G. Fekete: Rendering and managing spherical data with sphere quadtrees, in *Proc. IEEE Visualization 90*, pp. 176–186, IEEE Computer Society Press, 1990.

[48] M. Floater, E. Quak: Piecewise linear prewavelets on arbitrary triangulations, *Numer. Math.* 82:221–252, 1999.

[49] R. Fowler, J. Little: Automatic extraction of irregular network digital terrain models, *Computer Graphics* 13(3):199–207, 1979.

[50] D. Forsey, R. Bartels: Hierarchical B–spline refinement, *Computer Graphics* 22(4):205–212, 1988.

[51] L. Freitag, R. Loy: Adaptive multiresolution visualization of large data sets using parallel octrees, in *Proc. SC99*, ACM SIGARCH and IEEE Computer Society, 1999.

[52] H. Freudenthal: Simplizialzerlegungen von beschränkter flachheit. *Annals of Mathematics* 43(2):580–582, 1942.

[53] I. Gargantini: An Effective Way to Represent Quadtrees, *Comm. ACM* 25:905–910, 1982.

[54] M. Garland, P. Heckbert: Fast polygonal approximation of terrains and height fields, Report CMU-CS-95-181, Computer Science Department, Carnegie Mellon Univ., 1995.

[55] T. Gerstner: Ein adaptives hierarchisches Verfahren zur Approximation und effizienten Visualisierung von Funktionen und seine Anwendung auf digitale 3–D Höhenmodelle, Diplomarbeit, Institut für Informatik, TU München, 1995.

[56] T. Gerstner, M. Griebel: Numerical integration using sparse grids, *Numerical Algorithms* 18:209–232, 1998.

[57] T. Gerstner: Adaptive hierarchical methods for landscape representation and analysis, in *Process Modelling and Landform Evolution*, S. Hergarten, H.J. Neugebauer (eds.), pp. 75–92, Lecture Notes in Earth Sciences 78, Springer, 1999.

[58] T. Gerstner, M. Rumpf: Multiresolutional parallel isosurface extraction based on tetrahedral bisection, in *Volume Graphics*, M. Chen, A.E. Kaufman, R. Yagel (eds.), pp. 267–278, Springer, 2000.

[59] T. Gerstner, M. Rumpf, U. Weikard: Error indicators for multilevel visualization and computing on nested grids, *Computers & Graphics* 24(3):363–373, 2000.

[60] T. Gerstner, M. Hannappel: Error measurement in multiresolution digital elevation models, in *Accuracy 2000*, G. Heuvelink, M. Lemmens (eds.), pp. 245-252, Delft University Press, 2000.

[61] T. Gerstner: Multiresolution visualization and compression of global topographic data, *GeoInformatica*, in revision, also in *Proc. Spatial Data Handling 2000*, P. Forer, A.G.O. Yeh, J. He (eds.), pp. 14–27, IGU/GISc, 2000.

[62] T. Gerstner, R. Pajarola: Topology preserving and controlled topology simplifying multiresolution isosurface extraction, in *Proc. IEEE Visualization 2000*, IEEE Computer Society Press, pp. 259–266, 2000.

[63] T. Gerstner: Fast multiresolution extraction of multiple transparent isosurfaces, in *Proc. VisSym 01*, eingereicht, auch als SFB 256 report 40, 2001.

[64] T. Gerstner, H. Helfrich, A. Kunoth: Wavelets in the geosciences, SFB 350 preprint, 2001.

[65] J. Göttelmann: Compression of atmospheric data by spherical wavelets, *Contr. Atmos. Phys.* 72(1):95–104, 1999.

[66] M. Griebel, G. Zumbusch: Parallel multigrid in an adaptive PDE solver based on hashing and space–filling curves, *Parallel Computing* 25:827–843, 1999.

[67] M. Griebel, G. Zumbusch: Parallel adaptive subspace correction schemes with applications to elasticity, *Comp. Meth. Appl. Mech. Eng.* 184:303–332, 2000.

[68] M. Gross, O. Staadt, R. Gatti: Efficient triangular surface approximations using wavelets and quadtree data structures, *IEEE Trans. Vis. Comp. Graph.* 2(2), 1996.

[69] M. Gross, L. Lippert, O. Staadt: Compression methods for visualization, *Fut. Gen. Comp. Sys.* 15(1):11-29, 1999.

[70] R. Grosso, C. Lürig, T. Ertl: The multilevel finite element method for adaptive mesh optimization and visualization of volume data. in *Proc. IEEE Visualization '97*, pp. 387–394, IEEE Computer Society Press, 1997.

[71] B. Guo: A multiscale model for structure–based volume rendering, *IEEE Trans. Vis. Comp. Graph.* 1(4):291–301, 1995.

[72] W. Hackbusch: Multi–Grid Methods and Applications, Springer, 1985.

[73] B. Hamann: A data reduction scheme for triangulated surfaces, *Comp. Aid. Geom. Des.* 11:197–214, 1994.

[74] T. He, L. Hong, A. Varshney, S. Wang: Controlled topology simplification, *IEEE Trans. Vis. Comp. Graph.* 2(2):171–184, 1996.

[75] D. Hebert, H. Kim: Image encoding with triangulation wavelets, in *Wavelet Applications in Signal and Image Processing III*, A.F. Laine, M.A. Unser, M.V. Wickerhauser (eds.), pp. 381–392, 1995.

[76] P. Heckbert, M. Garland: Survey of surface approximation algorithms, in *SIGGRAPH '97 Course Notes*, 1997.

[77] D. Hilbert: Über die stetige Abbildung einer Linie auf ein Flächenstück, *Mathematische Annalen* 38:459-460, 1891.

[78] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stützle: Mesh optimization, *Computer Graphics (SIGGRAPH '93 Proc.)*, pp. 19–26, 1993.

[79] H. Hoppe: Progressive meshes, *Computer Graphics (SIGGRAPH '96 Proc.)*, pp. 99–108, 1996.

[80] H. Hoppe: View–dependent refinement of progressive meshes, *Computer Graphics (SIGGRAPH '97 Proc.)*, pp. 13–20, 1997.

[81] H. Hoppe: Smooth view-dependent level-of-detail control and its application to terrain rendering, in *Proc. IEEE Visualization '98*, pp. 35-42, 1998.

[82] T. Itoh, Y. Yamaguchi: Isosurface generation using extrema graphs, in *Proc. IEEE Visualization '94*, pp. 77–83, IEEE Computer Society Press, 1994.

[83] E. Kawaguchi, T. Endo: On a method of binary picture representation and its application to data compression, *IEEE Trans. Patt. Anal. Mach. Intel.* 5:373–384, 1983.

[84] D. Koller, P. Lindstrom, W. Ribarsky, L. Hodges, N. Faust, G. Turner: Virtual GIS: a real–time 3D geographic information system, in *Proc. IEEE Visualization '95*, pp. 94–100, IEEE Computer Society Press, 1995.

[85] I. Kossaczky: A recursive approach to local mesh refinment in two and three dimensions, *J. Comp. Appl. Math* 55:275-288, 1994.

[86] U. Kotyczka, P. Oswald: Piecewise linear prewavelets of small support, in *Approximation Theory VIII*, vol. 2, C. Chui, L. Schumaker (eds.), pp. 235–242, World Scientific, 1995.

[87] R. Klein, G. Liebich, W. Straßer: Mesh reduction with error control, in *Proc. IEEE Visualization '96*, pp. 311–318, IEEE Computer Society Press, 1996.

[88] E. LaMar, B. Hamann, K. Joy: Multiresolution techniques for interactive texture–based volume visualization, in *Proc. IEEE Visualization '99*, pp. 355–362, IEEE Computer Society Press, 1999.

[89] D. Laur, P. Hanrahan: Hierarchical splatting: A progressive refinement algorithm for volume rendering, *Computer Graphics (SIGGRAPH '91 Proc.)*, pp. 285–288, 1991.

[90] M. Lee, H. Samet: Navigating through triangle meshes implemented as linear quadtrees, Report CAR–TR–887, Computer Science Department, University of Maryland, 1998.

[91] E. Liang, S. Lin: A hierarchical approach to distance calculation using the spread function, *Int. J. GIS.* 12(6):515–535, 1998.

[92] P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, G. Turner: Real–time, continuous level of detail rendering of height fields, *Computer Graphics (SIGGRAPH '96 Proc.)*, 1996.

[93] L. Lippert, M. Gross: Fast wavelet based volume rendering by accumulation of transparent texture maps. *Computer Graphics Forum* 14(3):431–444, 1995.

[94] V. Liseikin: Grid generation methods, Springer, 1999.

[95] A. Liu, B. Joe: Quality local refinement of tetrahedral meshes based on bisection, *SIAM J. Sci. Comp.* 16(6):1269–1291, 1996.

[96] Y. Livnat, H. Shen, C.R. Johnson: A near optimal isosurface extraction algorithm using the span space, *IEEE Trans. Vis. Comp. Graph.* 2(1):73–83, 1996.

[97] W. Lorensen, H. Cline: Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics* 21(4):163–169, 1987.

[98] T. Lounsbery, T. DeRose, J. Warren: Multiresolution analysis for surfaces of arbitrary topological type, *ACM Trans. Graph.* 16:34–73, 1997.

[99] J. Maubach: Local bisection refinement for n-simplicial grids generated by reflection, *SIAM J. Sci. Comp.* 16:210–227, 1995.

[100] J. Maubach: The efficient location of neighbors for locally refined n-simplicial grids, in *Proc. 5th International Meshing Roundtable*, pp.137-156, Sandia National Labratories, 1996.

[101] M. Meissner, J. Huang, D. Bartz, K. Mueller, R. Crawfis: A practical evaluation of popular volume rendering algorithms, in *Proc. Volume Visualization 2000*, pp. 81–91, ACM Press, 2000.

[102] W. Mitchell: Adaptive refinement for arbitrary finite element spaces with hierarchical bases, *J. Comp. Appl. Math.* 36:65–78, 1991.

[103] S. Muraki, Approximation and rendering of volume data using wavelet transforms, *Comp. Graph. Appl.* 13(4):50–56, 1993.

[104] B. Natarajan: On generating topologically consistent isosurfaces from uniform samples, *The Visual Computer* 11(1):52–62, 1994.

[105] G. Nielson, D. Holliday, T. Roxborough: Cracking the cracking problem with Coons patches, in *Proc. IEEE VIsualization '99*, pp. 285–290, IEEE Computer Society Press, 1999.

[106] M. Ohlberger, M. Rumpf: Adaptive projection methods in multiresolutional scientific visualization, *IEEE Trans. Vis. Comp. Graph.* 4(4):74–94, 1998.

[107] R. Pajarola: Large scale terrain visualization using the restricted quadtree triangulation, in *Proc. IEEE Visualization '98*, pp. 19–24, IEEE Computer Society Press, 1998.

[108] V. Pascucci, C. Bajaj, Time critical isosurface refinement and smoothing, in *Proc. Volume Visualization 2000*, pp. 33–42, ACM Press, 2000.

[109] A. Paul, K. Dobler: Adaptive realtime terrain triangulation, in *Proc. WSCG '97*, University of West Bohemia, Plzen, Czech Republic, 1997.

[110] B. Payne, A. Toga: Surface mapping brain function on 3D models. *IEEE Comp. Graph. Appl.* 10(5):33–41, 1990.

[111] H. Pfister (org.): The transfer function bake–off. in *Panel session at IEEE Visualization '00*, 2000.

[112] E. Puppo, R. Scopigno: Simplification, LOD and multiresolution principles and applications. in *Eurographics 97 Tutorial Notes*, Eurographics Association, 1997.

[113] S. Röttger, W. Heidrich, P. Slussallek, H.-P. Seidel: Real–time generation of continuous levels of detail for height fields, in *Proc. 6th Int. Conf. in Central Europe on Computer Graphics and Visualization*, pp. 315–322, 1998.

[114] M. Rivara: Algorithms for refining triangular grids suitable for adaptive and multigrid techniques, *Int. J. Num. Meth. Eng.* 20:745–756, 1984.

[115] M. Rivara, C. Levin: A 3D refinement algorithm suitable for adaptive and multi–grid techniques, *J. Comp. Appl. Math.* 8:281–290, 1992.

[116] J. Ruppert: A delaunay refinement algorithm for quality 2–dimensional mesh generation, *J. Algorithms* 18(3):548–585, 1995.

[117] H. Sagan: Space–filling Curves, Springer, 1994.

[118] H. Samet: Data structures for quadtree approximation and compression, *Comm. ACM* 28:973–993, 1985.

[119] H. Samet: The Design and Analysis of Spatial Data Structures, Addison–Wesley, 1990.

[120] L. Scarlatos, T. Pavlidis: Hierarchical triangulation using cartographic coherence, *Comput. Vis. Graph. Image Proc.* 54(2):147–161, 1992.

[121] W. Schroeder, J. Zarge, W. Lorensen: Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proc.)*, pp. 65–70, 1992.

[122] P. Schröder, W. Swelderns: Spherical wavelets: efficiently representing functions on the sphere, *Computer Graphics (SIGGRAPH '95 Proc.)*, pp. 161–172, 1993.

[123] F. Schumacher: Adaptiv hierarchische Datenkompression mit Fehlerkontrolle basierend auf raumfüllenden Kurven, Diplomarbeit, Institut für Angewandte Mathematik, Universität Bonn, 2000.

[124] J. Shapiro: Embedded image coding using zerotrees of wavelet coefficients, *IEEE Trans. Sig. Proc.* 41(12):3445-3462, 1993.

[125] R. Shekhar, E. Fayyad, R. Yagel, J. Cornhill: Octree-based decimation of marching cubes surfaces, in *Proc. IEEE Visualization '96*, pp. 335–244, IEEE Computer Society Press, 1996.

[126] J. Shewchuk: Tetrahedral mesh generation by Delaunay refinement, in *Proc. Fourteenth Annual Symposium on Computational Geometry* pp. 86-95, ACM, 1998.

[127] W. Sierpiński: Sur une nouvelle courbe continue qui remplit toute une aire plane, *Bull. Acad. Sci. de Cracovie* 462–478, 1912.

[128] O. Staadt, M. Gross, R. Weber: Multiresolution compression and reconstruction, in *Proc. IEEE Visualization '97*, pp. 337–364, IEEE Computer Society Press, 1997.

[129] O. Staadt, M. Gross: Progressive tetrahedralizations, in *Proc. IEEE Visualization '98*, pp. 397–402, IEEE Computer Society Press, 1998.

[130] B. Stander, J. Hart: Guaranteeing the topology of an implicit surface polygonization, *Computer Graphics (SIGGRAPH '97 Proc.)*, pp. 279-286, 1997.

[131] R. Stevenson: Piecewise linear (pre-)wavelets on non–uniform meshes, in *Multigrid Methods V*, W. Hackbusch, G. Wittum (eds.), Springer, 1998.

[132] E. Stollnitz, T. Derose, D. Salesin: Wavelets for Computer Graphics: Theory and Applications, Morgan Kaufman, 1996.

[133] W. Sweldens: The lifting scheme: A custom-design construction of biorthogonal wavelets, *Appl. Comput. Harmon. Anal.* 3:186–200, 1996.

[134] S. Theis, A. Hense: Filterung von LM–Vorhersagefeldern mittels einer Wavelettransformation, Manuscript, Meteorologisches Institut, Universität Bonn, 2000.

[135] J. Thomas: Handbook of grid generation, CRC Press, 1999.

[136] C. Traxler: An algorithm for adaptive mesh refinement in n dimensions, *Computing* 59:115–137, 1997.

[137] I. Trotts, B. Hamann, K. Joy, D. Wiley: Simplification of tetrahedral meshes, in *Proc. IEEE Visualization '98*, pp. 287–296, IEEE Computer Society Press, 1998.

[138] G. Turk: Re–tiling polygonal surfaces, *Computer Graphics (SIGGRAPH '92 Proc.)*, pp. 55–64, 1992.

[139] A. Van Gelder, J. Wilhelms: Topological considerations in isosurface generation, *ACM Trans. Graph.* 13(4):337–375, 1994.

[140] R. Verführt: A Review of a posteriori error estimation and adaptive meshrefinement techniques, Teubner Verlag, 1996.

[141] L. Velho, L. De Figueiredo, J. Gomes: Hierarchical generalized triangle strips, *The Visual Computer* 15:21–35, 1999.

[142] L. Velho, J. Gomes: Variable resolution 4–k meshes: concepts and applications, *Computer Graphics Forum* 19(4):195–212, 2000.

[143] B. Von Herzen, A. Barr: Accurate triangulations of deformed, intersecting surfaces, *Computer Graphics (SIGGRAPH '87 Proc.)*, pp. 103–110, 1987.

[144] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, T. Ertl: Level–of–detail volume rendering via 3D textures, In *Proc. Volume Visualization 2000*, pp. 7–13, ACM Press, 2000.

[145] R. Westermann: A multiresolution framework for volume rendering, In *Proc. Volume Visualization 94*, pp. 51–57, ACM Press, 1994.

[146] R. Westermann, L. Kobbelt, T. Ertl: Real–time exploration of regular volume data by adaptive reconstruction of isosurfaces, *The Visual Computer*, 15:100–111, 1999.

[147] J. Wilhelms, A. Van Gelder: Octrees for faster isosurface generation. *ACM Trans. Graph.* 11(3):201–227, 1992.

[148] Y. Yang, F. Lin, H. Seah. Fast multi–resolution volume rendering, in *Volume Graphics*, M. Chen, A. Kaufman, R. Yagel (eds.), pp. 185–197, Springer, 2000.

[149] H. Yserentant: On the multi–level splitting of finite element spaces, *Numer. Math.* 49:379–412, 1986.

[150] Y. Zhou, W. Chen, Z. Tang: An elaborate ambiguity detection method for constructing isosurfaces within tetrahedral meshes, *Computers & Graphics* 19(3):355–364, 1995.

[151] Y. Zhou, B. Chen, A. Kaufman: Multiresolution tetrahedral framework for visualizing volume data. in *Proc. IEEE Visualization '97*, pp. 135–142, IEEE Computer Society Press, 1997.