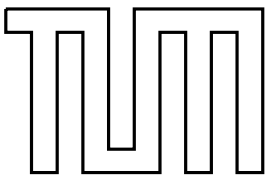
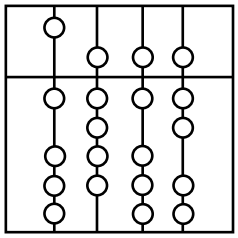


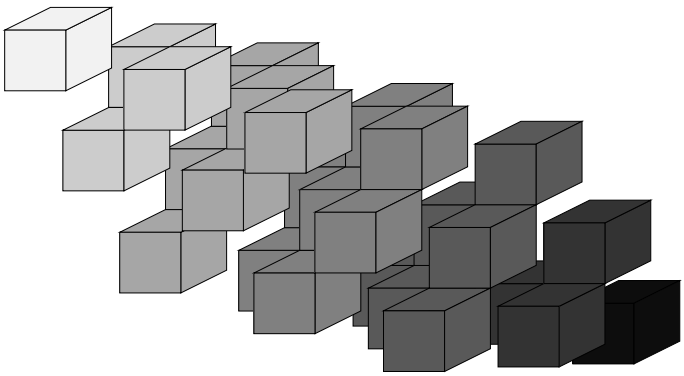
Technische Universität München
Institut für Informatik



Diplomarbeit

Kompression von Bildfolgen mit hierarchischen Basen

Andreas Paul



Technische Universität München
Institut für Informatik

Diplomarbeit

Kompression von Bildfolgen mit hierarchischen Basen

Andreas Paul

Aufgabensteller: Prof. Dr. Christoph Zenger
Betreuer: Dr. Hans-Joachim Bungartz

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

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

München, den 15. Juni 1995

_____

Inhaltsverzeichnis

Kapitel 1

Einleitung

Mit der fortschreitenden Entwicklung von Kommunikationstechniken, die von einer immer breiter werdenden Öffentlichkeit genutzt werden, hat das Gebiet der Datenkompression in den letzten Jahren stark an Bedeutung gewonnen. Es sollen immer mehr Daten immer schneller übertragen werden, was durch herkömmliche Übertragungstechniken kaum noch realisiert werden kann. Um dennoch die ständig wachsenden Anforderungen erfüllen zu können, werden bereits in vielen Bereichen die zu übertragenden Daten komprimiert.

Innerhalb von Computernetzwerken werden schon seit langem größere Datenmengen fast nie unkomprimiert übertragen. Neue Standards für Tonaufzeichnung im Bereich der Unterhaltungselektronik, Minidisk und Digitale Compact Cassette, arbeiten mit hochentwickelten Kompressionsverfahren, um möglichst viel Audiodaten auf kleinem Raum zu speichern. Unter dem immer populärer werdenden Begriff *Multimedia* verbreiten sich Anwendungen, die digital gespeicherte Videosequenzen abspielen. Dies wäre ohne Datenkompression nicht möglich. Im *World Wide Web*, dem Prototyp des von den USA propagierten *Information–Superhighway* erfreuen sich Videosequenzen aller Art höchster Beliebtheit. Bei den derzeit zur Verfügung stehenden Übertragungsgeschwindigkeiten innerhalb des *Internets* ist ein Austausch selbst kurzer Bildsequenzen von geringer Qualität in unkomprimierter Form völlig undenkbar.

Es zeigt sich, daß vor allem Bild- und Toninformationen in ihrer digitalen Darstellung großen Platzbedarf haben. Aber gerade die Übertragung und Speicherung genau dieser Art von Informationen ist von allgemeinem Interesse, da sich die menschliche Wahrnehmung in der Erfassung der Umwelt vor allem auf optische und akustische Eindrücke stützt. Die Kompressionsraten herkömmlicher Verfahren, wie Lempel–Ziv–, Huffman– oder Lauflängen–Codierung, sind zu gering, um Bild- und Toninformationen soweit zu komprimieren, daß eine sinnvolle Handhabung dieser Daten möglich wird. Dies wird schnell klar, wenn man bedenkt, daß allein die Bildinformationen eines 90 minütigen Videofilms 166.9 GB Speicherplatz benötigen, auf einer CD–ROM aber nur ca. 600 MB Daten Platz haben. Selbst eine Kompression auf nur 1% der Ausgangsdaten, also 1.6 GB,

würde nicht genügen, um diese Datenflut auf einer CD–ROM unterzubringen. Aus diesem Grund werden derzeit von internationalen Expertengruppen Verfahren entwickelt, die sich die besonderen Eigenschaften von Audio- und Videodaten zu Nutzen machen, um so Kompressionsraten erzielen zu können, die den aktuellen Anforderungen genügen.

Gleichzeitig ist man im Bereich der Numerik aufgrund der Entwicklung neuer Techniken in den letzten Jahren in der Lage, immer größere Problemestellungen in Angriff zu nehmen. Hierzu war es nötig, Datenstrukturen und Algorithmen zu entwerfen, die das effiziente Rechnen mit großen Datenmengen und häufig auch gleichzeitig deren speichereffiziente Darstellung erlauben. Ein interessantes und an Bedeutung gewinnendes Verfahren ist die von Zenger in [21] vorgestellte *Dünngittertechnik*. Sie stützt sich auf die *Theorie der hierarchischen Basen*, deren Entwicklung in die Anfänge dieses Jahrhunderts zurückreicht. Diese nutzt eine zur Interpolation und Integration von Funktionen entwickelte, Technik für die Lösung von partiellen Differentialgleichungen. Durch die Verwendung der Düngittermethode kann dabei die Lösung der Differentialgleichungen deutlich effizienter realisiert werden. Ein grundlegender Aspekt bei der Verwendung hierarchischer Basen und dünner Gitter ist eine Hierarchisierung der in den Daten enthaltenen Informationen und die adaptiven Verfeinerungsstrategien beim Erfassen dieser Informationen.

Aus Untersuchungen ist bekannt, daß sich bei der menschlichen Wahrnehmung ähnliche Gewichtungen von Informationen zeigen, wie sie durch die in der Dünngittertechnik verwendete Hierarchisierung entstehen. So weiß man beispielsweise, daß das Auge grobe Strukturen besser bzw. schneller wahrnimmt als Details. Letztere sind für das schnelle Erkennen eines Bildes von geringerer Bedeutung. Werden mit der Dünngittermethode mathematische Funktionen interpoliert, so wird zunächst in wenigen Schritten die grobe Struktur der Funktion approximiert. Im weiteren Verlauf wird genau an den Stellen verfeinert, d.h. Arbeit und Speicherplatz investiert, die auch dem Auge interessant erscheinen würden, also an Kanten, Singularitäten oder allgemein Stellen, die sich von ihrer Umgebung stark unterscheiden.

Die Vorgehensweise der Numerik orientiert sich also am menschlichen Wahrnehmungsapparat. Es ist demnach vielversprechend, die aus der Numerik stammenden Techniken zur Darstellung von Informationen, die zur menschlichen Wahrnehmung bestimmt sind, also im wesentlichen Bild- und Toninformationen, zu verwenden. Hierbei sind gute Kompressionsraten bei geringem Informationsverlust zu erwarten.

Eine erste Arbeit auf diesem Gebiet stammt von Katrin Hiller [11], die sich mit der Kompression von Grauwertbildern durch die Dünngittermethode befaßte. Das Ergebnis dieser Untersuchungen war so vielversprechend, daß weitere Forschungen auf dem Gebiet der Datenkompression mit der Dünngittermethode mehr als gerechtfertigt erschienen. Aus diesem Grund wurde gleichzeitig in Diplomarbeiten von A. Frank hierarchische Basen zur Kompression von Audiodaten, von Th.

Gerstner die Kompression und vor allem die effiziente Darstellung von Höhendaten unter Verwendung hierarchischer Konzepte untersucht. Der nächste konsequente Schritt, der der Bildkompression folgen soll und in dieser Arbeit getan wird, geht von der Grauwertbildkompression zur Erweiterung der Algorithmen zur Kompression von Bildfolgen, also Videodaten. Es zeigt sich jedoch bald, daß eine bloße Erweiterung nicht genügt, da im Zusammenhang mit der Verarbeitung von Videodaten Probleme auftreten, die in [11] nicht berücksichtigt werden konnten und mußten. Es wird also eine völlige Neuentwicklung unter dem speziellen Gesichtspunkt der Kompression von Videodaten nötig.

Im 2. Kapitel wird zunächst auf die Grundlagen der Video- und Fernsehtechnik eingegangen, um die Eigenschaften der Ausgangsdaten für die in dieser Arbeit zu entwickelnden Kompressionsverfahren besser einschätzen zu können. Auch soll ein Grundverständniss für die bei der Verarbeitung von Videoinformationen auftretenden Probleme geschaffen werden. Anschließend werden herkömmliche Standardkompressionstechniken gezeigt. Den Abschluß dieses Kapitels bilden die Beschreibungen bestehender, etablierter Verfahren zur Kompression von Bildern und Bildfolgen. Das 3. Kapitel bietet eine Einführung in die für diese Arbeit relevanten Aspekte der Dünngittertheorie. Im folgenden 4. Kapitel werden nun die zur Kompression von Daten nötigen Algorithmen und Techniken entwickelt, die auf einer gänzlich anderen Datenstruktur arbeiten als in [11]. Die Probleme und deren Lösungen, die durch die geänderte Datenstruktur und die Zielsetzung dieser Arbeit entstehen, werden ebenso aufgezeigt wie neue, erst durch die geänderte Datenstruktur mögliche Strategien und Überlegungen. Die Entwicklung der Algorithmen erfolgt anhand von Grauwertbildern, da diese einen leicht handzuhabenden Datensatz darstellen und im weiteren Verlauf verwandte Daten, nämlich Bildfolgen, komprimiert werden sollen. Bevor die entwickelten Algorithmen für die eigentliche Zielsetzung der Arbeit erweitert werden, geht das 5. Kapitel auf Alternativen zu den im 3. Kapitel vorgestellten Basisfunktionen ein. Es wird ein Konstruktionsprinzip für Polynombasen beliebigen Grades sowie eine allgemeine Methode zur Ermittlung des Interpolationsoperators bei vorgegebener Basis entwickelt. Die so erzeugten Basisfunktionen werden bezüglich ihrer Tauglichkeit zur Kompression von Bilddaten untersucht. Im 6. Kapitel werden nun die zuvor ausführlich getesteten Algorithmen zur Kompression von Grauwertbildern auf Bildfolgen erweitert. Hierzu werden die bis zu diesem Zeitpunkt gemachten Einschränkungen Schritt für Schritt fallen gelassen, um die Bearbeitung beliebiger Bildfolgen zu ermöglichen. Darüber hinaus wird auf Aspekte eingegangen, die bei der Echtzeit–Kompression und –Dekompression auftreten. Das 7. Kapitel faßt die gewonnenen Erkenntnisse und Ergebnisse zusammen und zeigt Ansatzpunkte für weitergehende Arbeiten auf diesem Gebiet.

Wie bereits erwähnt hat sich parallel zu dieser Anton Frank in seiner Diplomarbeit mit der Kompression von Audiodaten beschäftigt. Beide Arbeiten entstanden im Rahmen eines gemeinsamen Projektes, das sich allgemein mit der Kompression von Daten durch die Dünngittermethode beschäftigt. Da auch bei

der Verarbeitung von Audiodaten ähnliche Probleme und Datenstrukturen auftreten wie bei der Verarbeitung von Bildfolgen, wurden die grundlegenden Algorithmen gemeinsam entwickelt. Aus diesen Gründen wurden Teile der Arbeiten gemeinsam verfaßt. Dies betrifft die Kapitel 3,4 und 5, den Abschnitt 2.2 sowie Teile des Abschnitts 2.3 und des Anhangs.

An dieser Stelle möchte ich ganz besonders Dr. Hans Bungartz danken, der durch seine intensive Betreuung zum Entstehen dieser Diplomarbeit wesentlich beigetragen hat. Er stand jederzeit für Fragen und Diskussionen zur Verfügung und investierte weit mehr Zeit, als es erwartet werden durfte. Meinem Aufgabensteller Prof.Dr. Christoph Zenger bin ich für sein Interesse und seine sehr hilfreichen Hinweise zu Dank verpflichtet. Weiter danke ich den Mitarbeitern des Lehrstuhls sowie Veronika Thurner für zahlreiche aufschlußreiche Diskussionen, Thomas Gerstner der oft mit fruchtbarem Erfahrungsaustausch zur Seite stand und Ulrike Deisz die bei der Erstellung der Ausarbeitung half. Bei meiner Mutter und meinem Onkel bedanke ich mich für die großzügige Unterstützung während meines Studiums. Nicht zuletzt gilt mein Dank Anton Frank, der durch die angenehme und effektive Zusammenarbeit eine wertvolle Hilfe bei der Erstellung dieser Diplomarbeit darstellte.

Kapitel 2

Grundlagen der Videotechnik, Codierung und Kompression

2.1 Fernseh- und Videotechnik

Das Ausgangsmaterial für Bildfolgen, deren Kompression das Ziel dieser Arbeit ist, besteht meist aus Videosequenzen oder Bildern, die direkt von einer Kamera stammen. Daher ist es nötig, sich mit der zugrundeliegenden Technik auseinanderzusetzen, um verschiedene Eigenheiten, die bei der Verarbeitung von Bildfolgen auftreten, besser einschätzen zu können. Auch kann man so wertvolle Hinweise für eine effizientere Kompressionsstrategien gewinnen.

2.1.1 Entwicklung

Bei der Entwicklung der Fernseh- und Videotechnik war das maßgebliche Ziel, eine natürliche Szene mit einer Kamera aufzuzeichnen und die Bildinformationen auf elektronischem Wege, z.B. mittels elektromagnetischer Wellen, zu übertragen und anschließend wieder anzuzeigen.

Die Aufzeichnung und Wiedergabe von Bildern mit elektronischen Mitteln unterscheidet sich grundlegend von der herkömmlichen Methode, der Aufzeichnung durch einen mit lichtempfindlichen Chemikalien beschichteten Film. Bei der Aufzeichnung eines Bildes mit einer elektronischen Röhrenkamera und der Wiedergabe durch eine Braunsche Röhre wird die Bildfläche durch einen Elektronenstrahl zeilenweise abgetastet. Die Anzahl der der hierbei nötigen Zeilen ergibt sich aus der Dichte der lichtempfindlichen Nervenzellen auf der Netzhaut des Auges. Wird das Bild in mehr als 400 Zeilen zerlegt, so sind die einzelnen Zeilen nicht mehr unterscheidbar. Die gebräuchlichsten Video–Normen *PAL* und *NTSC* arbeiten beide mit mehr als 400 Zeilen. In der PAL–Norm wird ein Bild in 576, in der NTSC–Norm in 480 Zeilen aufgelöst. Das Verhältnis von Höhe zu Breite des Bildes (4:3) führt bei PAL zu einer Auflösung von 768, bei NTSC zu 640 Bildpunkten in horizontaler Richtung.

Für die photographische Aufnahme von Filmen hatte man sich bei Einführung des Tonfilms international auf eine Bildfrequenz von 24 Bildern pro Sekunde geeinigt. Bei der Entwicklung der Fernsehnormen bestand die Forderung nach Synchronität mit der Wechselstromfrequenz der Stromversorgung, d.h. die Netzfrequenz sollte ein ganzzahliges Vielfaches der Bildfrequenz sein. Der Hintergrund dieser Forderung ist der Umstand, daß die Wechselstromfrequenz zu Beginn der Entwicklung noch zum Takten der elektronischen Bauteile verwendet wurde. Absicht war es, Interferenzen zwischen Bild- und Taktfrequenz von vornherein auszuschließen. So führten die 60 Hz Wechselstromfrequenz in Nordamerika durch Halbierung der Frequenz zu einer Bildfrequenz von 30 Bildern pro Sekunde im Falle der NTSC–Norm, die 50 Hz Netzfrequenz in Europa zu 25 Bildern pro Sekunde in der PAL–Norm. Bei einer Bildfrequenz von 25 Bildern pro Sekunde bei Aufnahme und Wiedergabe wird eine kontinuierliche Bewegung vom Menschen als flüssig empfunden, obwohl sie in diskrete Einzelbilder zerlegt worden ist.

Die Verwendung einer Braunschen Röhre zur Wiedergabe der Bilder führte zu einer weiteren Unterteilung der Bilder. Läuft der Elektronenstrahl die Bildfläche 25 bzw. 30 mal pro Sekunde ab, so nimmt das Auge ein störendes Flimmern des Bildes wahr. Durch die Elektronen wird immer nur ein Bildpunkt zum Leuchten angeregt, der dann eine gewisse Zeit nachleuchtet. Ist die Zeit, die bis zum nächsten Durchlauf des Elektronenstrahls vergeht, zu lang, so verliert der Bildpunkt zuviel an Leuchtkraft, was schließlich bei sequentiellem Abarbeiten aller Bildzeilen zu dem erwähnten Flimmern führt. Um dies zu vermeiden müßte der Strahl die ganze Bildfläche mindestens 40 mal pro Sekunde ablaufen. Eine Möglichkeit, diese Frequenz zu erreichen, wäre, dasselbe Bild zweimal nacheinander abzulaufen. Dazu ist es jedoch nötig, das Bild entweder zweimal nacheinander komplett zu übertragen, was eine erhebliche Verschwendung an Bandbreite darstellt, oder das Bild zwischenzuspeichern, was zur Zeit der Entwicklung dieser Technik noch nicht möglich war. Die Alternative ist, das Bild nicht Zeile für Zeile aufzubauen, sondern z.B. zunächst die geradzahligen und dann die ungeradzahligen Zeilen abzulaufen (Abb. 2.1).

Die Bildfläche wird also zweimal überstrichen um ein Bild aufzubauen. Bei einem Durchgang werden die geraden Zeilen zum Leuchten angeregt, beim nächsten Durchgang die ungeraden. Der Nachleuchteffekt eines Bildpunktes klingt zwar immer noch gleich schnell ab, aber dafür werden die oberen und unteren Nachbarn eines Bildpunktes zum Leuchten angeregt, während seine eigene Helligkeit noch am Abklingen ist. So behält das Bild eine gewisse Gesamthelligkeit, und das Auge nimmt kein Flimmern mehr war. Die Technik, ein Bild in *Halbbilder* zu zerlegen, also gerade und ungerade Zeilen getrennt voneinander zu behandeln, wird *Interlacing* genannt. Die *Dominanz* einer Fernsehnorm legt fest, ob zuerst das Halbbild, das sich aus den geradzahligen Zeilen zusammensetzt, oder das aus den ungeraden Zeilen zusammengesetzte Halbbild übertragen und angezeigt wird. In der PAL–Norm ist die Dominanz gerade (even), im Fall von NTSC ungerade (odd).

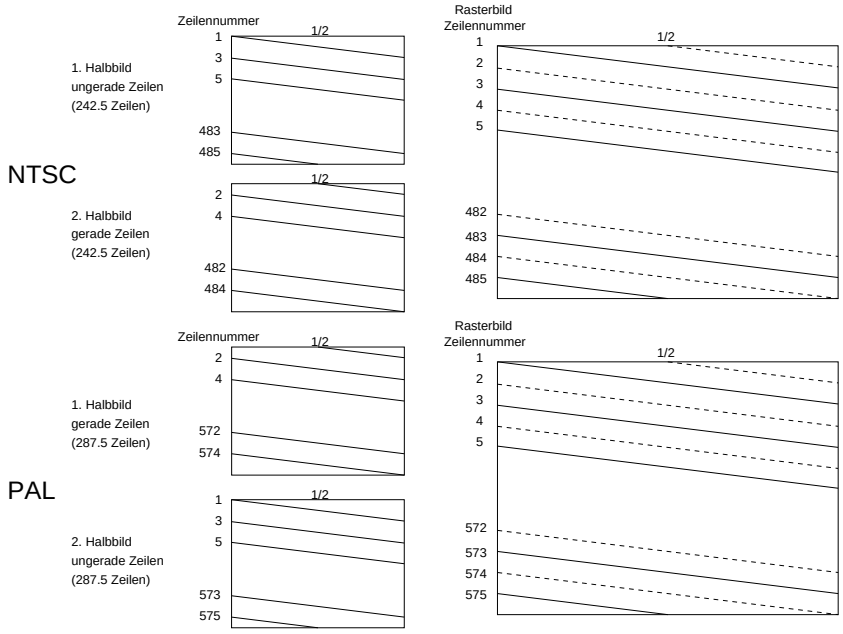


Abbildung 2.1: Zerlegung eines Bildes in Halbbilder

Man kommt durch diese Vorgehensweise also zu einer Halbbildfrequenz von 50 bzw. 60 Hz, die verhindert, daß das Bild flimmert, und zu einer Vollbildfrequenz von 25 bzw. 30 Hz, die den Eindruck kontinuierlicher Bewegungen erhält. Weitere Informationen zur Entwicklung der Fernseh- und Videotechnik findet man in [18].

	Pixel	Bilder/sec	Halbbilder/sec	Dominanz	MB/sec
PAL	768×576	25	50	gerade (even)	~ 32
NTSC	640×480	30	60	ungerade (odd)	~ 26

Inzwischen ist bekannt, daß eine Anzeigefrequenz von 50 Bildern pro Sekunde zwar nicht bewußt als Flimmern wahrgenommen wird, sich diese aber dennoch auf Dauer schädlich auswirken kann. Kopf- und Augenschmerzen sind mögliche Folgen. Das erklärt, warum moderne Computermonitore inzwischen grundsätzlich mit höheren Bildfrequenzen arbeiten, 70 Hz und mehr sind übliche Werte. Auch gibt es inzwischen Fernseher, die eine Bildfrequenz von 100 Hz haben. Hierzu werden die einzelnen Halbbilder, die digital zwischengespeichert werden, zweimal hintereinander angezeigt. Selbst bei Leuchtstofflampen, die ebenfalls mit der Netzfrequenz flackern, gibt es inzwischen Bestrebungen, die Frequenz zu verdoppeln, um nachteilige Langzeitauswirkungen zu vermeiden.

2.1.2 Farbmodelle

Nach der Zerlegung kontinuierlicher Bewegung in Einzelbilder und der Aufteilung der einzelnen Bilder in diskrete Bildpunkte muß für jedes dieser *Pixel* seine Helligkeit (*Luminanz*) bzw. seine Farbe (*Chrominanz*) bestimmt werden. Zur Bestimmung der Luminanz wird beispielsweise bei einer Schwarz–Weiß–Röhrenkamera der Ladestrom eines Elektronenstrahls, der die Bildfläche überstreicht, gemessen. Dieser ist proportional zur Helligkeit des jeweiligen Bildpunktes.

Die Bestimmung der Chrominanz gestaltet sich etwas schwieriger. Zunächst muß ein Modell zur Farbdarstellung gewählt werden. Vereinfacht kommt der menschlichen Wahrnehmung wohl das RGB–Farbmodell, das 1807 von Thomas Young entwickelt wurde, am nächsten. Die Netzhaut des Auges besteht aus verschiedenen Sinneszellen, Zapfen und Stäbchen. Letztere ermöglichen das Erkennen von Hell–Dunkel–Unterschieden und sind empfindlicher, d.h. sie sprechen schon bei geringeren Lichtstärken an, als die für die Farbunterscheidung verantwortlichen Zapfen. Es gibt verschiedene Zapfentypen, die auf unterschiedliche Wellenlängenbereiche des Lichtes ansprechen und dabei jeweils unterschiedlich stark reagieren (Abb. 2.2). Aufgrund der Verteilung der Frequenzbereiche kann man grob vereinfacht von Zellen sprechen, die jeweils auf die Farben Rot, Grün und Blau ansprechen. Dies wird dem Auge zwar nicht ganz gerecht, aber ein Großteil der sichtbaren Farben kann in diese drei Grundfarben zerlegt bzw. durch ihre Mischung erzeugt werden. Der Verwendung anderer Farbmodelle in der Anzeigetechnik von Videobildern, die unter Umständen alle Farben des sichtbaren Spektrums darstellen könnten, steht die technische Realisierbarkeit im Wege. Farbfernseher und Monitore erzeugen ihre Farben nach dem Prinzip des RGB–Farbmodells.

Für die Bestimmung der Rot-, Grün- und Blauanteile eines Bildes kann nun beispielsweise eine herkömmliche Schwarz–Weiß–Kamera nacheinander mit entsprechenden Farbfiltern versehen werden. Die dann gemessenen Helligkeitsverteilungen entsprechen dem Anteil der jeweiligen Farbe. Anstatt die Filter nacheinander aufzusetzen, kann das Licht natürlich auch durch halbdurchlässige Spiegel aufgeteilt werden und parallel drei Kameras mit verschiedenen Farbfiltern zugeführt werden. Auf diese Weise kann, auf Schwarz–Weiß–Aufnahmetechniken basierend, eine Farbkamera realisiert werden.

Der Grund, warum das RGB–Farbmodell, das für Aufnahme und Wiedergabe genutzt wird, sich normalerweise nicht in den Signalen, die zur Übertragung von Videobildern verwendet werden, widerspiegelt, ist historisch bedingt. Zunächst wurde das Schwarz–Weiß–Fernsehen entwickelt und eingeführt. Im Videosignal wurde nur die Luminanz übertragen. Bei der Entwicklung des Farbfernsehens bestand die Forderung der Abwärtskompatibilität, d.h. auch mit den alten Schwarz–Weiß–Empfängern mußte es möglich sein, Farbsendungen zu empfangen und korrekt darzustellen. Eine technische Nachrüstung der alten Geräte durfte dabei nicht nötig werden. Es war also nicht möglich, anstelle der Helligkeit die Rot-,

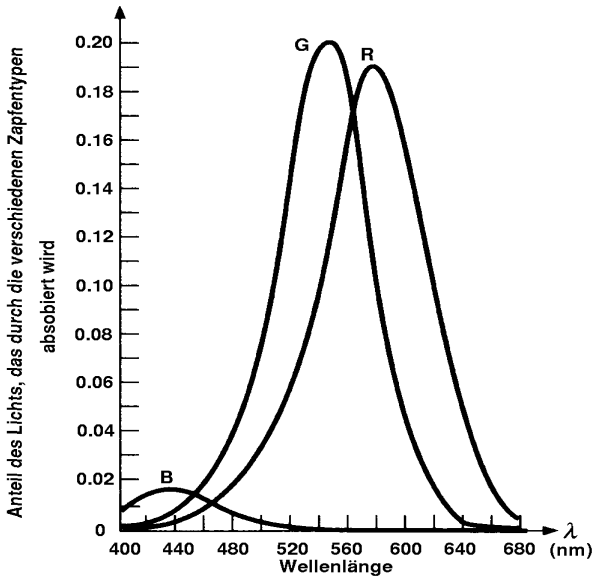


Abbildung 2.2: Absorption der Spektralfarben

Grün- und Blauanteile zu übertragen. Die Lösung dieser Problematik besteht in der Farbzerlegung nach dem YUV–Modell. Aus den Rot-, Grün- und Blauanteilen (R,G,B) einer Farbe errechnen sich die YUV–Komponenten folgendermaßen:

$$\begin{aligned} Y &= 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B, \\ U &= B - Y, \\ V &= R - Y. \end{aligned} \tag{2.1}$$

Der Y–Kanal entspricht der Luminanz eines Bildpunktes, also dem Signal, das die alten Schwarz–Weiß–Geräte zur Bilderzeugung benötigten. U und V sind die Differenzen der Luminanz mit dem Blau- bzw. Rotanteil der darzustellenden Farbe. Aufgrund der geringeren Empfindlichkeit des Auges gegenüber Farben benötigen der U– und V–Kanal eine deutlich geringere Bandbreite zur Übertragung als der Y–Kanal. So konnten diese Informationen in das alte von Schwarz–Weiß–Geräten verwendete Videosignal eingebettet werden, ohne den Schwarz–Weiß–Empfang zu stören (Abb. 2.3). Farbfernsehgeräte sind imstande, aus dem Signal die U– und V–Komponenten auszufiltern und mit ihnen die ursprünglichen RGB–Darstellung zu rekonstruieren.

Für die Wahl der Faktoren für den Rot-, Grün- und Blauanteil bei der Bestimmung des Y–Wertes gibt es verschiedene Standards bzw. Empfehlungen. Der Unterschied liegt meist im Wertebereich des Y–, U– und V–Kanals. Das im Bereich der digitalen Bildverarbeitung häufig verwendeten Farbmodell YCbCr, das eine Variante der YUV–Darstellung ist, zeichnet sich durch einen Wertebereich aus, der sich gut zur digitalen Speicherung eignet.

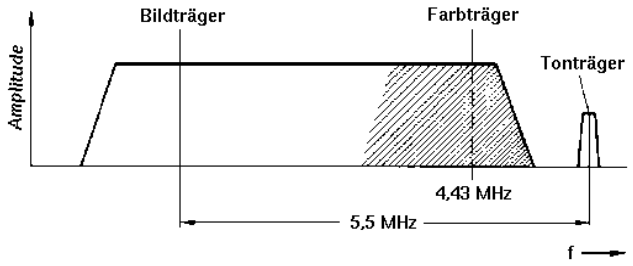


Abbildung 2.3: Frequenzband eines Fernsehkanals

Abgesehen vom YUV– und RGB–Modell gibt es noch viele weitere Farbdarstellungen, die im Bereich der Video- und Fernsehtechnik allerdings kaum eine Rolle spielen. Zu nennen sind CMY bzw. CMYK, mit denen in der Farbdrucktechnik gearbeitet wird. Im Gegensatz zum additiven RGB–Modell werden in diesem subtraktiven System die Farben durch ihre Cyan-, Magenta- und Gelbanteile dargestellt. Bei CMYK kommt noch eine Schwarzkomponente hinzu. Farben werden hierbei durch Subtraktion, d.h. Ausfiltern von Farbanteilen, von der Grundfarbe Weiß erzeugt. Im RGB–Farbmodell erfolgt die Mischung additiv, d.h. durch Überlagerung der einzelnen Farbkomponenten vor dunklem Hintergrund. Der Unterschied beruht auf der Tatsache, daß Papier, auf das meist gedruckt wird, fast immer weiß ist, während der Hintergrund von Bildröhren in der Regel dunkel bzw. schwarz ist.

Ein völlig anderes Farbverständnis liegt Farbmodellen, wie beispielsweise HLS oder HSV zugrunde. Diese verfolgen einen eher intuitiven Ansatz. Die erste Komponente H (Hue) legt den Farbton fest. Die beiden anderen Werte L und S (Lightness und Saturation) bzw. S und V (Saturation und Value) legen Helligkeit und Sättigung der Farbe fest. Mit diesen Werten ist es möglich, Farben auf dieselbe Weise zu definieren, nach der Maler Farbtöne mischen. Es ist in der Regel leichter in einem dieser Farbmodelle eine bestimmte Farbvorstellung zu definieren, als in den Modellen die Farben durch Mischung von drei Grundfarben erzeugen. Eine detailierte Beschreibung der gebräuchlichsten Farbmodelle und Darstellungsformen entnehme man [9] S. 574ff. Informationen über die Erzeugung und Darstellung von Farben mittels Computern findet man in [12].

2.1.3 Videosignale und Aufzeichnung

Ein Videosignal, das alle zum Aufbau eines Farbbildes benötigten Informationen enthält, sieht man in Abbildung 2.4. Sie zeigt den Verlauf des Videosignals für eine Bildzeile. Man erkennt die unterschiedlichen Signalverläufe für Luminanz und Chrominanz.

Es besteht die Möglichkeit, Luminanz und Chrominanz zu überlagern und so ein Composite–Video–Signal zu erhalten (Abb. 2.5). Diese Signalform findet sich

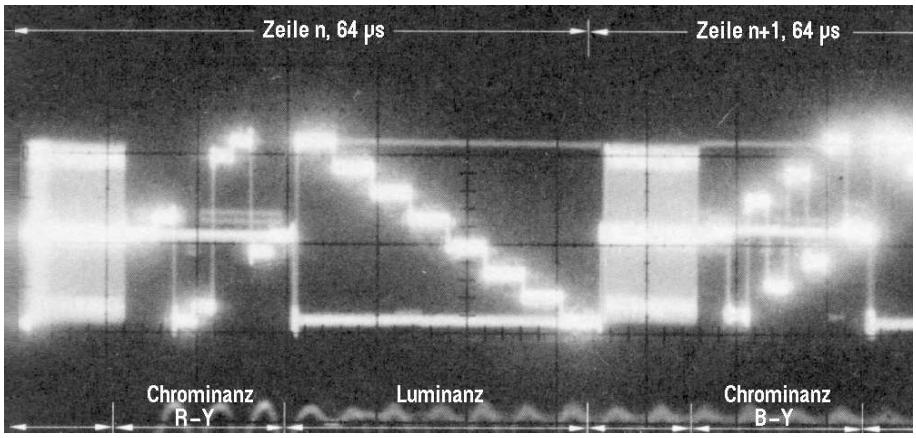


Abbildung 2.4: Verlauf eines Videosignals für eine Zeile

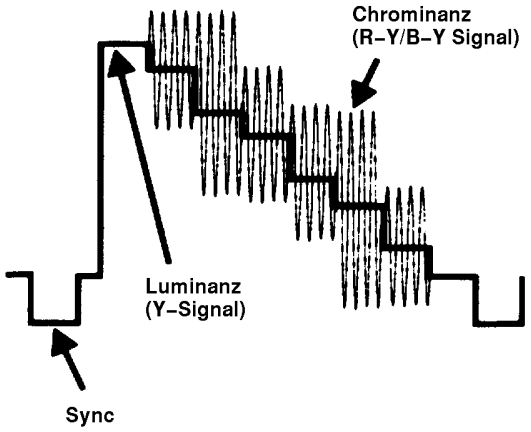


Abbildung 2.5: Composite–Video–Signal

häufig an den Videoausgängen von Fernsehern und VHS–Videorecordern wieder. Durch die Überlagerung der beiden unterschiedlichen Informationen wird das Signal anfälliger für Störungen. Der Verlust äußert sich, wenn überhaupt, in schlechteren Kontrasten. Der im Heimbereich auch vorkommende S–VHS–Standard arbeitet mit der getrennten Übertragung von Luminanz und Chrominanz und zeichnet sich demnach durch bessere Hell–Dunkel–Kontraste aus. Gegenüber einer getrennten Übertragung der RGB–Signale wird allerdings auch bei S–VHS ein deutlicher, wenn auch nicht dramatischer Qualitätsverlust sichtbar. Farbmonitore von Computern arbeiten praktisch immer mit der getrennten Übertragung der RGB–Signale.

Zur Aufzeichnung von Videosignalen auf Magnetbänder gibt es zahlreiche Standards. Im Heimbereich haben sich VHS (Abb. 2.7) und die Weiterentwicklung S–VHS durchgesetzt. Diese beiden Formate arbeiten mit geringeren Bildauflösungen als beim ursprünglichen Videobild, um den technischen Aufwand

und damit die Kosten möglichst gering zu halten. VHS zeichnet nur etwa 250 Zeilen eines Bildes auf, während S–VHS bis zu 400 Zeilen aufzeichnet.

Im professionellen Bereich gibt es verschiedene gebräuchliche Systeme, wie z.B. Betacam (Abb. 2.6) oder Chromatrack–M. Diese Formate zeichnen die verschiedenen Komponenten des Videosignals getrennt voneinander auf. Die Bildauflösung entspricht der des ursprünglichen Videobildes. So ist es im Gegensatz zu den VHS–Formaten möglich, nahezu verlustfreie Kopien von Bändern zu machen, was besonders beim Schneiden von Videofilmen eine große Rolle spielt. Schnitt ist dabei im Gegensatz zum photographischen Film oder Magnetbandaufzeichnungen von Audiosignalen nicht durch Zerschneiden und Wiederaneinanderkleben des Bandes möglich. Aufgrund der Menge der zu speichernden Informationen und zur Vermeidung zu hoher Bandgeschwindigkeiten wird bei der Aufnahme von Videosignalen auf Magnetbänder ein Schrägspuraufzeichnungsverfahren verwendet, das einen tatsächlichen, physikalischen Schnitt unmöglich macht.

2.1.4 Datenquellen und Digitalisierung

Im weiteren sollen die verschiedenen möglichen Ursprünge für Bildfolgen, die als Folge von Einzelbildern im Speicher des Rechners vorliegen, betrachtet werden.

Die naheliegenste Quelle für Einzelbildfolgen, die z.B. bei Videokonferenzen Verwendung findet, ist eine Videokamera. Diese nimmt 25 oder 30 Bilder pro Sekunde auf, je nachdem, welcher Videonorm sie genügt. Die Einzelbilder, die in der Kamera zunächst als analoges Signal vorliegen, müssen zur weiteren rechnergestützten Verarbeitung digitalisiert werden. Der Vorgang der Digitalisierung läßt sich in die Diskretisierung des Signals und eine Quantisierung der einzelnen Signalpegel zerlegen. Abbildung 2.8 zeigt die Diskretisierung und Quantisierung eines einfachen analogen Signals.

Eine geeignete Diskretisierung von Videosignalen ist bereits durch die Zerlegung der Videobilder in einzelne Bildpunkte vorgegeben. Daher liefern Videodigitizer gewöhnlich Bilddaten in derselben Auflösung, die Videobilder haben, also 768×576 oder 640×480 Pixel. Optional besteht häufig die Möglichkeit, die Daten auch in geringeren Auflösungen zu digitalisieren. Im Rahmen der Quantisierung wird der Helligkeit bzw. der Intensität der Farbanteile ein digitaler Wert zugeordnet. Die Anzahl der Bits, mit denen diese Werte dargestellt werden, bestimmt die Anzahl der unterscheidbaren Helligkeits- bzw. Farbabstufungen.

Für eine naturgetreue Wiedergabe eines Farbbildes werden für die Darstellung der Rot-, Grün- und Blauanteile der Farben häufig 8 Bit, also 24 Bit pro Bildpunkt verwendet. Ein Bild in PAL–Auflösung in der sogenannten 24–Bit–True–Color–Darstellung hat einen Speicherplatzaufwand von ca. 1.27 MByte. True-Color bedeutet, daß die Farbe für jedes Pixel einzeln gespeichert wird, statt mit einer Farbtabelle zu arbeiten und für jedes Pixel nur einen Index in diese Farbtabelle zu speichern.

An die Stelle einer Kamera kann auch ein Videoabspielgerät treten. Abgesehen

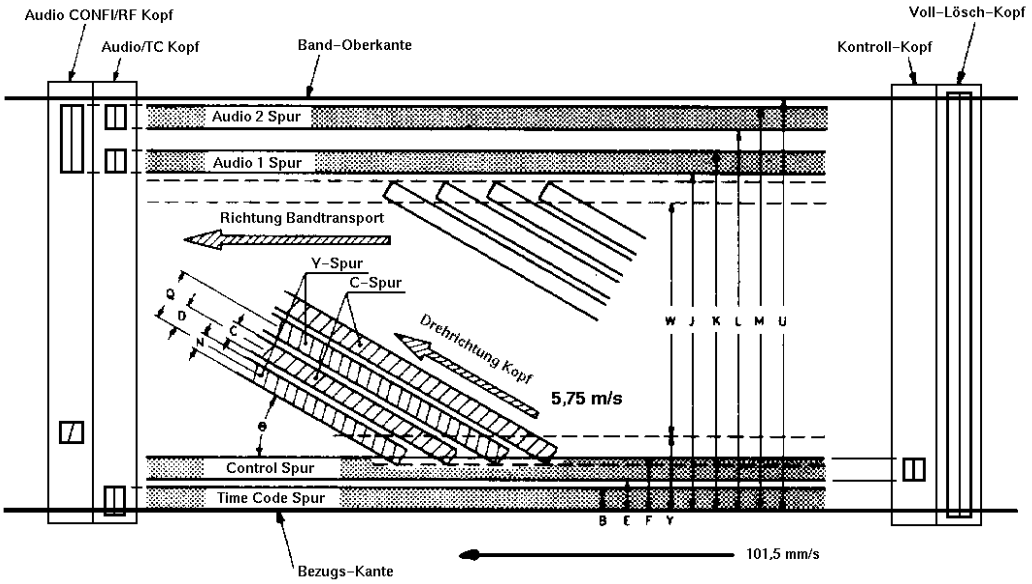


Abbildung 2.6: Aufzeichnung auf einem Betacam–Band

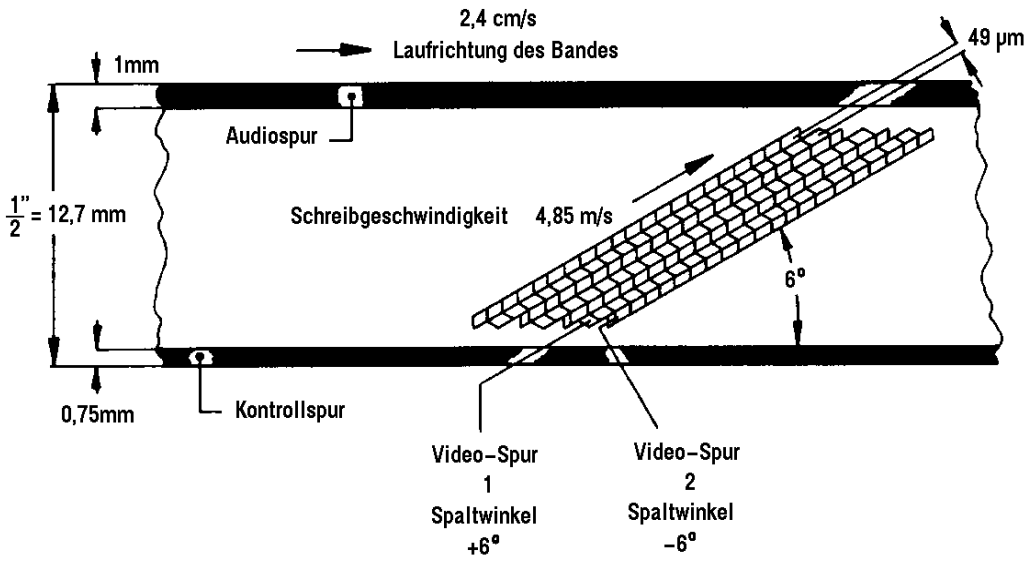


Abbildung 2.7: Aufzeichnung auf einem VHS–Band

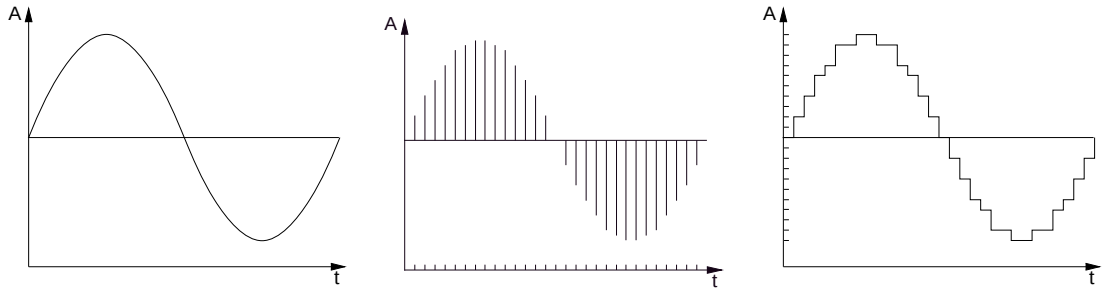


Abbildung 2.8: Diskretisierung und Quantisierung eines analogen Signals

von eventuellen Qualitätsverlusten durch die vorangegangene analoge Aufzeichnung der Videosignale stellt dies jedoch keine entscheidend andere Konfiguration dar. Der Videodigitalisierer liefert nach wie vor Bilder in voller PAL– bzw. NTSC–Auflösung.

Im Gegensatz zu dieser analogen, auf einem Meßvorgang beruhenden und damit zwangsläufig verrauschten Datenquelle stehen rein digitale, d.h. synthetische Quellen. Unter einer rein synthetischen Quelle versteht man einen Algorithmus bzw. ein Programm, das ohne Verwendung irgendwelcher zuvor digitalisierter Informationen, wie z.B. Texturen, Bilder berechnet. Auf diese Weise erzeugte Bilder zeichnen sich normalerweise durch exakte Kontraste und glatte Farbverläufe aus. Mischformen zwischen synthetischen und analogen Datenquellen treten auf, wenn beispielsweise in einem errechneten Bild digitalisierte Daten verwendet werden. Diese dienen dann meist dazu, das berechnete Bild realistischer erscheinen zu lassen. Als synthetische Erzeuger von Bildfolgen sind u.a. Raytrace–Programme zu nennen, mit deren Hilfe Bild für Bild flüssige Animationen erzeugt werden können.

Die einzelnen Bilder einer Folge sind also entweder durch einen Meßvorgang, gefolgt von einer Digitalisierung, entstanden, oder sie sind synthetisch, d.h. durch Berechnungen erzeugt worden. Letztere eignen sich in der Regel besser zur Kompression, es gibt jedoch auch Verfahren, die sich bestimmte Eigenschaften natürlicher Bilder zunutze machen.

2.2 Codierung und Kompression

Digitalisierte Analogsignale können zur Weiterverarbeitung in verschiedenen Formen, abhängig vom Verwendungszweck, dargestellt werden. Eine solche Darstellungsart bezeichnet man als *Codierung*. Ist die Codierung datenreduzierend, so spricht man auch von *Kompression*. Verschiedene gebräuchliche Arten und Vorgehensweisen zur Datencodierung und -kompression sollen im folgenden vorgestellt werden. Dabei wird die Übertragung bzw. Speicherung der Information als übliche Anwendung von Codier- und Kompressionsverfahren genutzt. Zur Messung

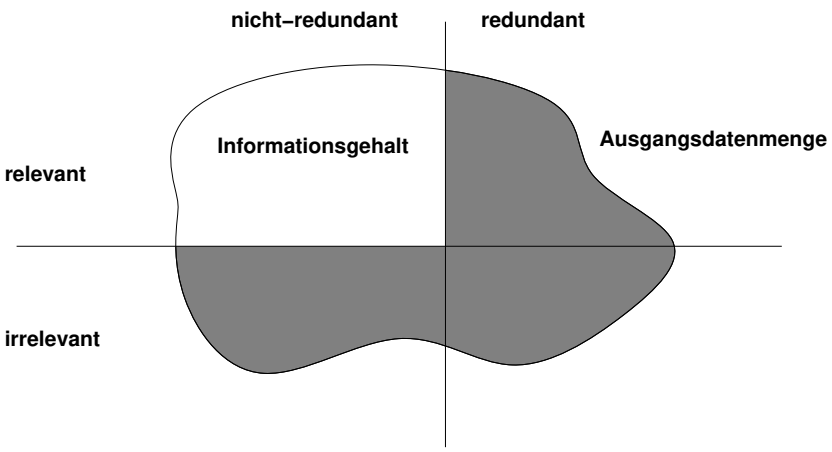


Abbildung 2.9: Informationsgehalt einer Datenmenge

der Effektivität eines Kompressionsverfahrens dient die *Kompressionsrate*. Darunter versteht man das Verhältnis des Platzbedarfes der Ausgangsdaten zum Platzbedarf der Daten in ihrer codierten Form.

2.2.1 Relevanz und Redundanz

Information läßt sich nach zwei unterschiedlichen Gesichtspunkten klassifizieren. Information bezeichnet man als *redundant*, wenn sie aus vorhandener Information vollständig rekonstruiert werden kann. Dies ist unabhängig vom Verwendungszweck der Information. Im Gegensatz dazu spielt die angestrebte Anwendung bei Betrachtung der Relevanz eine entscheidende Rolle. Informationen sind *relevant*, wenn auf sie aufgrund dieser Anwendung nicht verzichtet werden kann. *Irrelevante* Daten hingegen haben dafür keine Bedeutung. Abbildung 2.9 zeigt eine schematisierte Darstellung dieser Klassifizierung.

Häufig ist das Ziel von Codierung die Kompression, d.h. eine Reduktion der Daten auf ein Mindestmaß. Reduziert man die Daten um die redundanten Anteile, so verliert man keine Information. Man spricht daher von einem *verlustfreien* Verfahren (lossless compression). Entfernt man auch noch den irrelevanten Anteil, so können die Ausgangsdaten nicht mehr vollkommen rekonstruiert werden und man erhält eine *verlustbehaftete* Codierung der Daten (lossy compression).

2.2.2 Quellencodierung

Die *Quellencodierung*, die auch als Pulse–Code–Modulation (PCM) bezeichnet wird, ist im wesentlichen nichts anderes als der Vorgang der Digitalisierung. Das bedeutet, daß einem analogen Signal eine Folge diskreter Werte zugeordnet wird. Abbildung 2.8 zeigt die wesentlichen Schritte einer Digitalisierung. Für den Fall, daß bestimmte Eigenschaften des analogen Signals bekannt sind, z.B. die Häufig-

keit bestimmter Signalpegel, läßt sich diese Information nutzen, um das Signal effizienter zu codieren. Die Art der zu codierenden Information wird in der Regel nicht berücksichtigt. Quellencodierung geht in der einen oder anderen Art den folgenden Codierungsformen, die alle auf Folgen diskreter Werte arbeiten, voran.

2.2.3 Lauflängencodierung

Die *Lauflängencodierung* (run length encoding, RLE) stellt die einfachste Form der Codierung dar, die direkte Redundanz, d.h. Wiederholungen eines Wertes innerhalb der Information effizient codiert. Folgt mehrmals derselbe Wert hintereinander, so wird nicht jeder einzelne Wert für sich, sondern der Wert gefolgt von der Anzahl der Wiederholungen des Wertes codiert. Diese Anzahl wird auch als *Kettenlänge* bezeichnet. Um im Datenstrom Werte von Kettenlängen zu unterscheiden, kann beispielsweise auf spezielle Werte zurückgegriffen werden, die den Kettenlängen vorangestellt sind. Kann der gewählte spezielle Wert, der zur Markierung von Kettenlängen dient, selbst ein gültiger Wert des Datenstroms sein, so kann dieser durch die Wiederholung des Wertes codiert werden. Zur Lösung dieser Problematik existieren zahlreiche Ansätze, die jedoch oft an eine bestimmte Anwendung oder vorab bekannte Eigenschaften des Datenstroms gebunden sind. Dies gilt auch für den Fall, daß eine Kettenlänge durch die vorgesehene Anzahl an Bits nicht mehr darstellbar ist.

Abhängig von der Art der Daten läßt sich durch RLE–Codierung eine recht gute Kompressionsrate erreichen. Es folgt ein einfaches Beispiel einer Lauflängencodierung (Kompressionsrate 21:13):

42222222223210111112	→	4209321001052
Ausgangsdatenstrom		RLE–Codierung

Der zur Markierung der Kettenlängen verwendete Wert sei die Null, die selbst zum gültigen Wertebereich der Ausgangsdaten gehört.

Diese Form der komprimierenden Codierung findet häufig Verwendung bei der Codierung von Bildern, die mit Farbtabellen oder nur wenigen Farben arbeiten. Große einfarbige Flächen lassen sich RLE–codiert mit Hilfe weniger Werte darstellen. Enthält ein Bild mehr als 16 Farben bzw. Graustufen, so zeigt die Erfahrung, daß sich Lauflängenkomprimierung nicht mehr lohnt.

2.2.4 Huffman–Codierung

Eine *Huffman–Codierung* berücksichtigt die statistische Wahrscheinlichkeit, mit der bestimmte Werte in einem Datenstrom vorkommen. Je wahrscheinlicher ein Wert ist, desto kürzer ist das zugeordnete Codewort. Unter Codeworten versteht man dabei Bitfolgen, die den zugehörigen Wert eindeutig identifizieren. Diese Worte entnimmt der Algorithmus einer Codeworttabelle. Der Huffman–Algorithmus, der zur Erzeugung eines Huffmancodes dient, stellt dabei sicher,

daß eine im Mittel optimale Codierung gefunden wird. Eine genauere Beschreibung dieses Algorithmus findet sich z.B. bei Sedgewick [16, S. 375 ff.] oder Riedel [15, S. 26f.]. Wird die Codeworttabelle den sich ändernden Häufigkeiten des Datenstroms ständig angepaßt, so spricht man von einem adaptiven Huffman–Code.

Diese Art der Kompression findet sich unter anderem in den UNIX–Tools `pack` und `compact`, mit denen sich beliebige Dateien verlustfrei komprimieren lassen.

2.2.5 Lempel–Ziv–Codierung

Das von A. Lempel und J. Ziv 1977 und 1978 entwickelte Kompressionsverfahren beruht auf der Ersetzung von häufig auftretenden Wertefolgen in einem Datenstrom durch einen Verweis auf einen Eintrag in einem „Wörterbuch“ (dictionary), einer Codeworttabelle, die diese Wertefolgen enthält. Im Wörterbuch werden während der Codierung in geeigneter Weise „Wörter“, d.h. Folgen von Werten eingetragen, die im Datenstrom vorkommen. Bei nochmaligem Auftreten eines bereits im Wörterbuch vorhandenen Wortes wird nur noch ein Verweis auf das Wörterbuch codiert. Es gibt verschiedene Varianten des Algorithmus, die sich hauptsächlich in der Organisation und Handhabung des Wörterbuches unterscheiden.

Die meisten der gängigen Kompressionsprogramme stützen sich auf diesen Algorithmus. Dazu gehören `compress`, `gzip`, `arj`, `lha`, `zip` und `zoo`.

2.2.6 Transformationscodierung

Bei einer Transformationscodierung wird einer Anzahl von Werten eine Folge von Koeffizienten zugeordnet. Bei entsprechender Transformationsvorschrift, die zum Zwecke der Decodierung eindeutig umkehrbar sein muß, kann die Koeffizientenfolge besser zur Kompression geeignet sein als die Ausgangswerte. Das oft auftretende starke Abfallen der Koeffizienten kann zu einer höheren Kompressionsrate führen.

Bekannte Vertreter der Familie der Transformationscodierungen sind Walsh–Hadamard–, Fourier–, Sinus–, Cosinus– und Wavelet–Transformationen.

2.2.7 Vektorquantisierung

Vektorquantisierung zählt zu den verlustbehafteten Verfahren. Ein einzelner Wert steht dabei für eine Reihe von im Wertebereich benachbarten Werten.

$$\begin{array}{lcl} \{10,11,12\} & \mapsto & 1 \\ \{13,14,15\} & \mapsto & 2 \\ \{16,17,18\} & \mapsto & 3 \end{array} \qquad \Longrightarrow \qquad 12,13,17,10,12 \;\rightarrow\; 1,\,2,\,3,\,1,\,1$$

Die Kompression liegt in der Verkleinerung des gültigen Wertebereiches, so daß weniger Bits zur Unterscheidung der einzelnen Werte benötigt werden.

Vektorquantisierung findet z.B. bei der Konvertierung von Bildern mit 24 Bit Farbtiefe, d.h. 3 Byte pro Bildpunkt, in Bilder mit nur insgesamt 256 verschiedenen Farben Anwendung. Aus den 2^{24} möglichen Farbwerten des Ausgangsbildes werden 256 ausgewählt, die den tatsächlich im Bild vorkommenden Farbwerten möglichst nahekommen. Diese 256 Farben werden in einer Farbtabelle gespeichert. Die im Bild vorkommenden Farben werden durch den Index in der Farbtabelle ersetzt, der auf einen Farbwert verweist, der dem in den Ausgangsdaten vorkommenden Farbwert möglichst nahekommt. Das so codierte Bild bedarf nur noch eines Bytes pro Bildpunkt, statt der drei Bytes in den Ausgangsdaten.

2.2.8 Weitere Verfahren

Differenzencodierung

Unterscheiden sich aufeinanderfolgende Werte nur geringfügig, so bietet sich eine Differenzencodierung an. Hierbei wird nach dem ersten Wert jeweils nur die Differenz aufeinanderfolgender Werte codiert. Diese Differenzen können dann meist durch eine geringere Anzahl von Bits dargestellt werden.

Eingesetzt wird dieses Verfahren beispielsweise bei der ADPCM (adaptive differential pulse code modulation).

Vorhersagende Codierung

Kann ein Wert aus bereits übertragenen Werten interpoliert werden, so ist die explizite Codierung nicht nötig. Dies ist eine weitere Form der Redundanzreduktion. Man spricht von vorhersagender Codierung (predictive coding). Die Interpolation kann beliebig kompliziert sein. Einfach geschieht dies durch eine lineare Interpolation der letzten beiden Werte. Dies bezeichnet man als linear predictive coding (LPC).

Fraktale Codierung

Eine neuere Form der Kompression stellt die Kompression mit Hilfe von Techniken, die aus der fraktalen Geometrie stammen, dar. Fraktale Kompression ist verlustbehaftet. Sie findet praktisch nur bei der Codierung von Bildern Anwendung. Die Algorithmen zur Codierung sind extrem zeitaufwendig, erreichen dafür aber sehr gute Kompressionsraten. Ein besonderer Vorteil dieser Kompressionsmethode ist die Möglichkeit, durch die Decodierung ein Bild mit höherer Auflösung als zuvor zu gewinnen. Die geringere Auflösung des Ausgangsbildes wird im rekonstruierten Bild aber nicht sichtbar, wie das bei herkömlichen Verfahren der Fall wäre. Der praktischen Anwendung des Verfahrens steht nach wie vor der hohe Zeitaufwand der Codierung im Wege.

Codierungsverfahren	Verlust	Einsatz (in Beispielen)
PCM	ja	Digitalisierung
Huffman	nein	Text, u.a.
RLE	nein	Bilder
Lempel–Ziv	nein	Text, u.a.
Transformation	nein	Signale
Vektorquantisierung	ja	Bilder, Signale
Differenzen	nein	Bildfolgen
LPC	nein	Bildfolgen, Signale
Fraktal	ja	Bilder

Abbildung 2.10: Verschiedene Codierungsverfahren und ihre Einsatzgebiete

Darüber hinaus gibt es noch eine Fülle weiterer Codierungsverfahren, die aber entweder auf den oben genannten basieren oder einen stark eingeschränkten Anwendungsbereich haben.

2.2.9 Orthogonalität von Codierungsverfahren

Oft lassen sich verschiedene Codierungsverfahren kombinieren, um bessere Kompressionsraten zu erzielen. Zwei verschiedene Codierungsverfahren, die nacheinander angewandt werden, ergänzen sich dabei optimal, wenn sie orthogonal zueinander sind, also sich gegenseitig in ihrer Kompressionseffektivität nicht beeinflussen. Reduziert z.B. ein erstes Verfahren einen Datenstrom um 10% und ein zweites Verfahren denselben Datenstrom um 20% und sind die beiden Verfahren orthogonal zueinander, so erzielt das zweite Verfahren nach wie vor eine Reduktion um 20%, auch wenn es auf den durch das erste Verfahren bereits um 10% reduzierten Datenstrom angewandt wird. Die Gesamtreduktion liegt also bei 28%. Selbst wenn sich die Verfahren gegenseitig beinflussen, so ist es immer noch möglich, bessere Kompressionsraten zu erreichen, wenn man mehrere Verfahren kombiniert.

Eine teilweise Orthogonalität wird oft bei der Kombination von Transformationscodierung mit nachfolgender alphabetorientierter (z.B. Huffman–) Codierung ausgenutzt ([7, S. 316]).

2.3 Kompression von speziellen Daten

Wie bereits erwähnt, besteht die Möglichkeit, effizientere Kompressionsverfahren zu entwickeln, wenn bestimmte Eigenschaften der zu komprimierenden Daten und der beabsichtigte Verwendungszweck vorher bekannt sind.

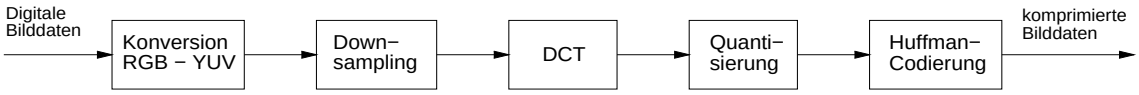


Abbildung 2.11: Blockdiagramm eines JPEG–Encoders

2.3.1 Bilddaten

Zur Kompression von Digitalbildern derzeit am weitesten entwickelt ist ein standardisiertes Verfahren, welches von der Joint Photographic Experts Group (JPEG), einer von der International Standardisation Organisation (ISO) eingesetzten Expertengruppe, entworfen worden ist. Das JPEG–Verfahren benutzt verschiedene der oben angesprochenen Verfahren, um ein optimales Ergebnis zu erzielen. Das Verfahren ist verlustbehaftet, wurde aber daraufhin ausgelegt, daß die Fehler für das menschliche Auge nicht wahrnehmbar sind.

Das Verfahren arbeitet auf Farb- sowie auf Grauwertbildern. Der Kompressionsvorgang gliedert sich in folgende Schritte: Zunächst werden die digital vorliegenden Bilddaten ins YUV–Modell konvertiert. Danach erfolgt ein Downsampling, d.h. eine Halbierung oder Viertelung der Auflösung im U– und V–Kanal. Dieser Verlust kann bei natürlichen Bildern vernachlässigt werden. Dadurch wird bereits eine Reduktion um 33% bzw. 50% erreicht. Die erhaltenen Daten werden nun in Blöcke der Größe 8×8 Pixel aufgeteilt. Jeder dieser Blöcke wird mittels einer diskreten Cosinus–Transformation (DCT), die in diesem Fall auf Ganzzahlarithmetik basiert, transformiert. Eine anschließende Quantisierung der resultierenden Koeffizienten bewirkt eine weitere Reduktion der Daten, jedoch auch den entscheidenden Informationsverlust. Die verbliebenen Daten werden zuletzt noch Huffman–codiert und mit zur Umkehrung des Vorgangs benötigter Information versehen. Abbildung 2.11 zeigt den schematisierten Ablauf der Kompression.

Weitere Informationen finden sich z.B. bei Pennebaker [14].

2.3.2 Videodaten

Unter Videodaten versteht man im Bereich der digitalen Datenkompression in der Regel eine Folge von Einzelbildern. Manche Verfahren ermöglichen eine gleichzeitige Kompression sowie das synchrone Abspielen von Bild- und Tondaten. Dies wird in der Regel durch Einbettung der Audiodaten in die Videodaten realisiert. Im weiteren wird auf die Problematik der Audiokompression, des Einbettens der Audiodaten in die Videodaten sowie des synchronen Abspielens der beiden Signale nicht weiter eingegangen.

Einfache Verfahren

Die einfachste Möglichkeit, Videodaten in einer Datei zu speichern, ist, die unkomprimierten Einzelbilder hintereinander abzulegen. Zum Abspielen müssen

dann lediglich die Bilder nacheinander geladen und angezeigt werden. Die Probleme, die sich daraus ergeben, sind, daß Videodaten sehr viel Platz benötigen und daß unter Umständen die Bilder nicht schnell genug geladen werden können, um beim Anzeigen die vorgesehene Bildrate zu erzielen. Aus diesem Grund verwenden auch die einfachen Dateiformate für Videodaten Kompressionsstrategien.

Da es sehr naheliegend ist, wird oft Differenzencodierung verwendet. Bei zeitlich aufeinanderfolgenden Bildern ist der Unterschied zwischen zwei Einzelbildern selten sehr groß. Bildet man also die Differenz der beiden Bilder, so eignet sich das Ergebnis in den meisten Fällen deutlich besser zur Codierung mit einem weiteren komprimierenden Verfahren als das Ausgangsmaterial. Da im Optimalfall ein Großteil der Bilddifferenz Null ist, kann meist schon durch anschließende RLE–Codierung eine gute Kompression erreicht werden.

Viele Dateiformate arbeiten nicht mit True–Color–Darstellung, sondern mit Farbpaletten–Bildern, die das Ergebnis einer Vektorquantisierung der True–Color–Bilder sind. Dies kann den Speicherbedarf beispielsweise um $\frac{1}{3}$ von 24 auf 8 Bit pro Bildpunkt reduzieren. Die Farbpalette kann sich zwar im Verlauf einer Animation ändern und muß dann erneut gespeichert werden, insgesamt wird aber immer noch deutlich weniger Platz benötigt als bei True–Color–Bildern. Der Unterschied des Bildes mit Farbpalette zur True–Color–Darstellung ist häufig nur gering. Der wesentliche Unterschied wird erst bei Bildern mit sehr glatten Farbverläufen und vielen unterschiedlichen Farbtönen deutlich, da hier bei Farbübergängen Abstufungen erkennbar werden.

Die meisten der gängigen Video–Dateiformate stützen sich in der einen oder anderen Form auf Vektorquantisierung, Differenzen– und RLE–Codierung.

Video–MPEG

Zu den komplizierteren Kompressionsverfahren zählt der sogenannte MPEG–Standard. MPEG steht für 'Motion Pictures Expert Group' und ist der Name der durch die ISO eingesetzten Arbeitsgruppe ISO/IEC JTC1/SC29/WG11. Die Entwicklung gliedert sich in mehrere Phasen:

- *MPEG*–1: Im ersten Abschnitt war das Ziel die Codierung bewegter Bilder und der zugehörigen Audioinformationen für digitale Speichermedien bei Bitraten von etwa 1.5 Mbit pro Sekunde. Hierfür wurde Oktober 1992 ein internationaler Standard (IS–11172) festgelegt.

- *MPEG*–2: In dieser Phase wurde die Codierung auf eine allgemeinere Ebene erweitert, als dies bei MPEG–1 der Fall war. So werden jetzt verschiedene Bildauflösungen (PAL, NTSC, HDTV etc.) unterstützt, sowie Halbbilder bei der Codierung berücksichtigt. Darüber hinaus wurde das Multiplexen von Datenströmen und die Syntax eines Kontrolldatenstroms definiert. Im November 1994 wurde hier ein internationaler Standard (IS–13818–1) festgelegt.

- *MPEG–4*: In der letzten Phase soll die Codierung von Audio- und Videodaten für sehr geringe Bitraten ausgelegt werden. Mit der Verabschiedung eines internationalen Standards wird hier im November 1998 gerechnet.

Die 3. Phase der Entwicklung, die sich mit dem Kontrolldatenstrom und dem Multiplexen von Datenströmen auseinander setzen sollte, wurde in MPEG–2 mit aufgenommen, was erklärt warum MPEG–3 nicht mehr als separater Entwicklungsabschnitt existiert.

Die Vorgehensweise von MPEG–1 ist etwa wie folgt: Ausgegangen wird von farbigen Einzelbildern mit einer Auflösung von 352×240 Bildpunkten bei 30 Bildern pro Sekunde in Nordamerika bzw. 352×288 Pixel bei 25 Bildern pro Sekunde in Europa. Die Bilder werden ins YUV–Farbmodell konvertiert und dann die Auflösung der U– und V–Kanäle weiter auf 176×120 bzw. 176×144 Bildpunkte verringert. Das Grundprinzip der Kompression beruht auf der Vorhersage von Bewegung in einer Bildsequenz. Diese geschieht jeweils für 16×16 Blöcke im Y–Kanal. Im Referenzbild, dem Bild, bezüglich dem die Bewegung festgestellt werden soll, wird ein Block gesucht, der dem im aktuellen Bild möglichst gut entspricht, d.h. die Differenz der beiden Blöcke muß unterhalb einer Schwelle bleiben. Aus der unterschiedlichen Position der Blöcke in den beiden Bildern ergibt sich ein Bewegungsvektor für diesen Block. Die verbleibende, möglichst geringe Differenz der beiden Blöcke wird mit Hilfe einer auf 8×8 Blöcken arbeitenden diskreten Cosinus–Transformation (DCT) codiert. Auch hier wird wie bei JPEG Quantisierung genutzt, um die Anzahl der Koeffizienten, die ungleich Null sind, zu reduzieren. Findet sich kein ähnlicher Block, so werden die ursprünglichen Daten des Blockes durch die DCT Komprimiert. Wenn möglich werden Koeffizienten und Bewegungsvektoren durch Differenzencodierung bzgl. vorangegangener Bilder dargestellt. Abschließend werden Bewegungsvektoren, DCT–Koeffizienten, Quantisierungsparameter und was sonst noch nötig ist Huffman Codiert.

Es gibt drei verschiedene Typen von komprimierten Einzelbildern in einem MPEG–1–Datenstrom, I–, P– und B–Frames. I– oder Intra–Frames werden völlig unabhängig von folgenden oder vorangegangenen Bildern komprimiert. D.h. sie werden etwa dem JPEG–Standard folgend komprimiert und dienen als Startpunkte für Kompression und Dekompression. Um einen schnellen beliebigen Zugriff auf die Einzelbilder einer Bildsequenz zu ermöglichen sollte etwa alle 0.4 Sekunden ein Bild als I–Frame Codiert werden. Für P–Frames (predicted) wird im folgenden Bild der Sequenz nach ähnlichen Blöcken für die Bestimmung von Bewegungsvektoren gesucht. Zur Codierung von B–Frames (bidirectional) werden für diese Suche das folgende und vorangegangene Bild herangezogen. Abgesehen von der Suche nach Bewegungsvektoren wird hier zusätzlich noch versucht Blöcke zwischen dem letzten und nächsten Bild zu interpolieren. Läßt sich ein Block weder durch Verschiebung noch durch Interpolation rekonstruieren, so werden auch hier die Ausgangsdaten durch DCT Codiert. Eine typische Abfolge von I–, P– und B–Frames ist:

I B B P B B P B B P B B I B B P B B P

Werden B–Frames verwendet, so gestalten sich Anwendungen mit Echtzeitkompression, -übertragung und -dekompression komplizierter, da Bilder nicht mehr in ihrer zeitlichen Reihenfolge übertragen werden können. B–Frames benötigen zu ihrer Dekompression Informationen aus zeitlich in der Zukunft liegenden Bildern. Es wird also ein Zwischenspeichern bzw. Puffern der einzelnen Bilder nötig.

Als die im Mittel zu erwartende Kompressionsrate bei Verwendung des MPEG–Verfahrens wird 1:48 angegeben. Pro Pixel werden also ca. 0.5 Bits benötigt.

Kapitel 3

Hierarchische Basen und dünne Gitter

Informationen, wie z.B. Signale, liegen meist in kontinuierlicher Form vor. Wie bereits in 2.1.4 erläutert, werden diese Signale zum Zweck der elektronischen Datenverarbeitung diskretisiert. Die Interpolation wird zur Umwandlung eines diskreten Datensatzes in eine kontinuierliche Funktion benutzt. Darüber hinaus ist durch Interpolation eine Entscheidung über die Redundanz von Werten des Datensatzes möglich. Kann ein Wert aus anderen Werten des Datensatzes hinreichend approximiert werden, so ist er überflüssig, also redundant. Wird bei der Approximation ein Fehler zugelassen, so werden Werte innerhalb der Fehlerschranke als irrelevant erachtet. Es ist klar, daß auf diese Weise für geeignete Datensätze eine erhebliche Anzahl von Punkten eliminiert werden kann, sofern man die Interpolationsstrategie richtig wählt.

3.1 Theorie der hierarchischen Basen

Gegeben sei ein Satz von Stützstellen x_i mit den Stützwerten u_i $(1 \leq i \leq N)$. Die allgemeine Interpolationsbedingung für einen Interpolanten u^I lautet nun:

$$u^I(x_i) = u_i, \tag{3.1}$$

beziehungsweise bei der Verwendung einer Basis $\{\psi_1, \ldots, \psi_N\}$ von Interpolationsfunktionen:

$$u^I(x_i) := \sum_{j=1}^{N} c_j \cdot \psi_j(x_i) = u_i. \tag{3.2}$$

Im Eindimensionalen ist die stückweise lineare *Stützpunktbasis* (Abb. 3.1) eine einfache Form einer Basis zur Interpolation eines auf einem äquidistanten Gitter der *Maschenweite* h angeordneten Datensatzes. Deren einzelne Basisfunktionen

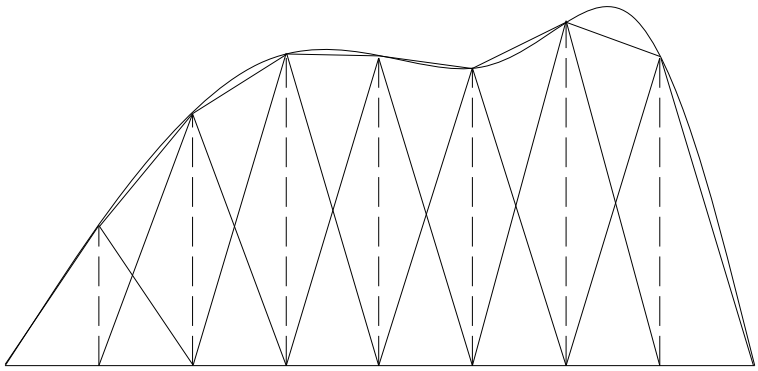


Abbildung 3.1: Funktion mit stückweise linearer Stützpunktbasis

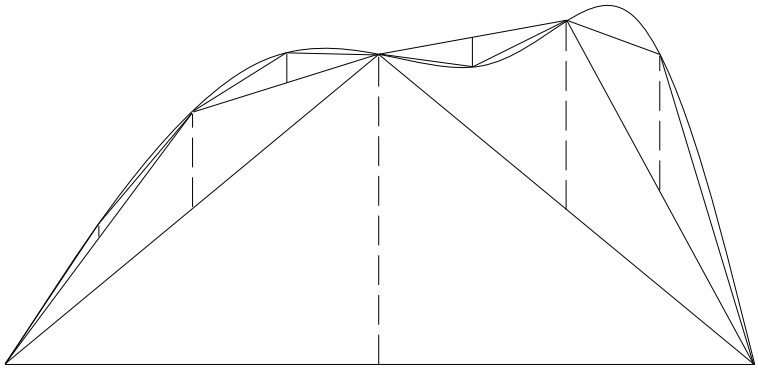


Abbildung 3.2: Funktion mit stückweise linearer hierarchischer Basis

erfüllen folgende Bedingungen:

$$\psi_i(x_j) = \delta_{ij} \tag{3.3}$$
$$\mathrm{supp}(\psi_i) = [x_i - h, x_i + h]. \tag{3.4}$$

Eine andere Variante ist die *hierarchische Basis* (Abb. 3.2), wie sie z.B. von Yserentant in [20] oder [19] dargestellt wurde. Hierbei wird nicht jeder Stützpunkt gleich behandelt, sondern es wird eine Hierarchie auf ihnen definiert, indem man das Gitter sukzessive verfeinert. Bei jeder Verfeinerung wird die Maschenweite des Gitters halbiert. Bei n Verfeinerungschritten, genannt *Stufen*, führt dies im eindimensionalen Fall zu einem Gitter mit einer Punktanzahl von $N = 2^n + 1$ ($n \in \mathbb{N}$) Punkten.

Die Bedingungen für eine hierarchische Basis lauten:

$$\varphi_i(x_i) = 1 \tag{3.5}$$
$$\varphi_i(x_i \pm h_k) = 0 \tag{3.6}$$
$$\mathrm{supp}(\varphi_i) = [x_i - h_k, x_i + h_k]. \tag{3.7}$$

Dabei bezeichnet k den k–ten der n Verfeinerungsschritte, die Stützstelle x_i taucht erstmals in dem Gitter der Stufe k auf. Das Intervall $\mathrm{supp}(\varphi_i)$, auf dem die Basisfunktion definiert ist, bezeichnet man als Träger der Basisfunktion φ_i. Die Basisfunktion und damit ihr Träger hängen von ihrer Stufe k ab. Das gröbste Gitter nimmt die höchste Stufe in der so erzeugten Hierarchie ein. Das Gitter der Stufe k bezeichnet man als *hierarchisch höherliegend* als das Gitter der Stufe $k+1$.

Einfache Funktionen, die die obigen Bedingungen erfüllen, sind stückweise lineare Funktionen, die der Übersichtlichkeit halber auf dem Rand des Gitters gleich Null sein sollen. Diese Funktionen bilden einen Raum, der durch obige Verfeinerungsstrategie in *Teilräume* von stückweise linearen Funktionen zerlegt wird, die auf Gitterpunkten höherer Hierarchiestufe verschwinden. Diese Teilräume werden im Eindimensionalen mit $T_j (1 \leq j \leq n)$ bezeichnet.

Die Koeffizienten der hierarchischen Basisfunktionen des Interpolanten bezeichnet man als *hierarchische Überschüsse* v_i. Diese zeichnen sich besonders dadurch aus, daß sie bei einer gewissen Glattheit der Daten von Stufe zu Stufe abfallen. Bei der sukzessiven Verfeinerung können die Überschüsse tieferliegender Stufen aus dem Funktionswert und den Überschüssen höherer Stufen berechnet werden. Ein weiterer Vorteil der auf den Daten definierten Hierarchie ist die dynamische Erweiterbarkeit und eine damit verbundene einfache Adaptionsfähigkeit. Als Adaptionskriterium kann die Größe des hierarchischen Überschusses dienen. Sie kann auch genutzt werden, um die Relevanz eines Punktes in einem Datensatz zu bewerten. Ist der Überschuß gleich Null, so ist dieser Stützwert redundant. Darüber hinaus ist es durch Festlegung einer Fehlerschranke ε möglich, einen gewissen Überschuß als Fehler zuzulassen und so Stützwerte als irrelevant zu klassifizieren. Wie aus der Interpolationstheorie bekannt, ist die Ordnung des maximalen Fehlers im Falle der stückweise linearen Interpolation $O(h^2)$.

Der hierarchische Überschuß berechnet sich im eindimensionalen, linearen Fall als Differenz zwischen dem tatsächlichen Wert an einem Gitterpunkt und dem aus den Randpunkten des Trägers interpolierten Wert an dieser Stelle, also:

$$v_i = u_i - \frac{u_a + u_b}{2} \quad \text{mit } a = x_i - h_k, b = x_i + h_k \tag{3.8}$$

oder in vereinfachter Operatorschreibweise:

$$v_i = \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \cdot u. \tag{3.9}$$

Dabei ist zu beachten, daß die Maschenweite h_k bei der Anwendung des Operators berücksichtigt wird.

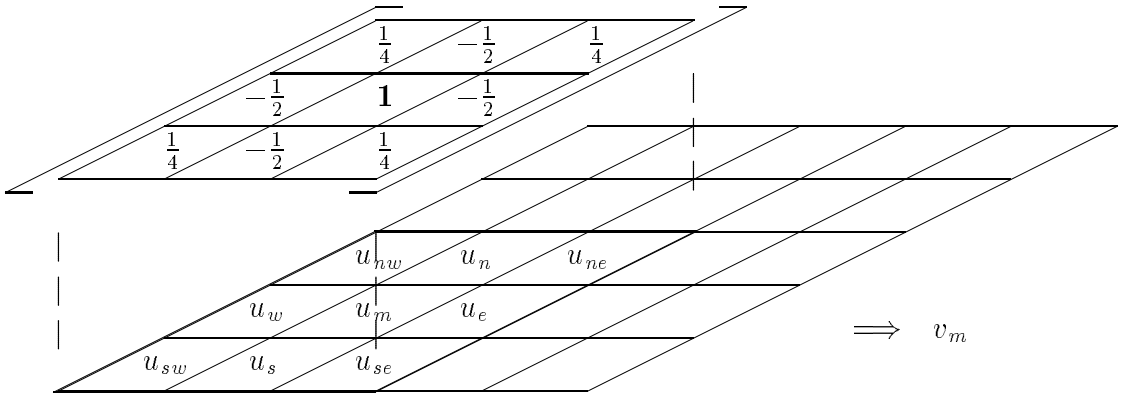


Abbildung 3.3: Aufsetzen des Operators auf das Gitter

Für die bilineare Interpolation im zweidimensionalen Fall berechnet sich der Überschuß durch Bildung des Tensorprodukts aus dem eindimensionalen Operator mit sich selbst:

$$v_i = \left(\begin{bmatrix} -\frac{1}{2} \\ 1 \\ -\frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \right) \cdot u = \begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \end{bmatrix} \cdot u \tag{3.10}$$

Dabei ist zu beachten, daß jetzt in x– und y–Richtung unterschiedliche Maschenweiten h_{k_x} und h_{k_y} auftreten können. Bezeichnet man die Position der zur Überschußberechnung verwendeten Gitterpunkte mit Himmelsrichtungen (n, s, w, e, nw, ne, sw, se), dann schreibt sich die Berechnung auch so:

$$v_i = u_i - \frac{1}{2}(u_{i_n} + u_{i_s} + u_{i_w} + u_{i_e}) + \frac{1}{4}(u_{i_{nw}} + u_{i_{ne}} + u_{i_{sw}} + u_{i_{se}}) \tag{3.11}$$

Die Anwendung des Operators zeigt Abbildung 3.3. Der Begriff des Teilraums wird ebenfalls in die zweite Dimension erweitert. So kommt man von den Teilräumen T_j und den Intervallen als Träger der Basisfunktionen im eindimensionalen zu Teilräumen T_{jk} und rechteckigen Trägerflächen im zweidimensionalen Fall. Die Teilräume T_{jk} lassen sich ihrer Erzeugung folgend in einem Schema anordnen. Abbildung 3.4 zeigt das zweidimensionale *Teilraumschema* für äquidistante quadratische, sogenannte *volle Gitter* (Abb. 3.6).

3.2 Die Dünngittermethode

Interpoliert man eine Funktion, die der Glattheitsbedingung, daß $\frac{\delta^4 u}{\delta^2 x \delta^2 y}$ existiert und beschränkt ist, genügt, so kann man zeigen (s. Bungartz in [3]), daß die hierarchischen Überschüsse bestimmte Eigenschaften haben. Innerhalb der Teilräume, die im Teilraumschema auf den Diagonalen liegen, d.h. innerhalb aller Teilräume

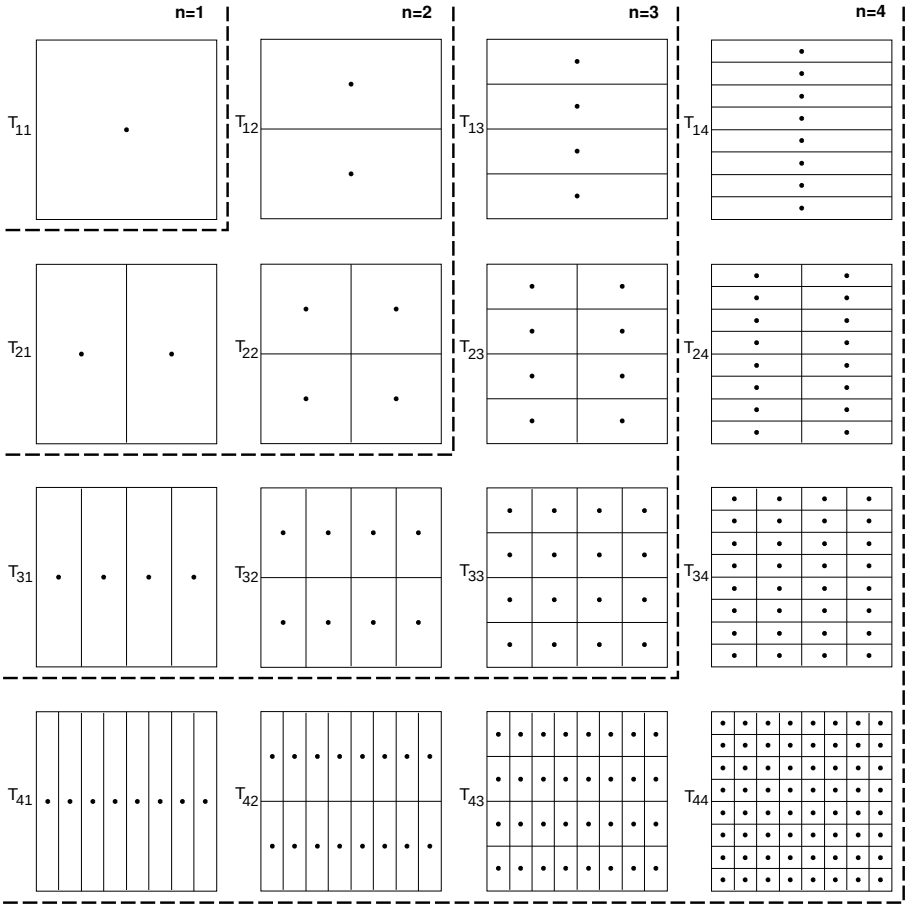


Abbildung 3.4: Teilraumzerlegung im zweidimensionalen Fall

T_{jk}, für die $j + k$ gleich ist, sind die hierarchischen Überschüsse von gleicher Größenordung (Abb. 3.5). Beim Übergang von einer Diagonale zur nächsten fallen die Überschüsse mit einem Faktor $\frac{1}{4}$ ab. Wie von Zenger in [21] vorgeschlagen, bietet es sich an, bei fortschreitender Verfeinerung des Gitters das Teilraumschema nicht jeweils um eine Zeile und Spalte zu erweitern, sondern lediglich um die auf der nächsten Diagonale liegenden Teilräume. Durch die Verwendung der auf diese Weise entstehenden *dünnen Gitter* (Abb. 3.6) verschlechtert sich die Interpolation nur unwesentlich gegenüber der Interpolation auf einem vollen Gitter, das durch Hinzufügen einer vollen Zeile und Spalte erzeugt wurde.

Aus diesem Grund ist es für die weiteren Betrachtungen sinnvoll, als Verfeinerungsstufen die Diagonalen des Teilraumschemas zu betrachten. Diese Verfeinerungsstufen werden im weiteren *Level* l genannt. Das Teilraumschema in Abbildung 3.4 enthält in dieser Betrachtungsweise also $l_{max} = 2n - 1$ Level. Die Anzahl l_{max} bei einem vorgegebenen Gitter der Größe $N \times N$, mit $N = 2^n + 1$,

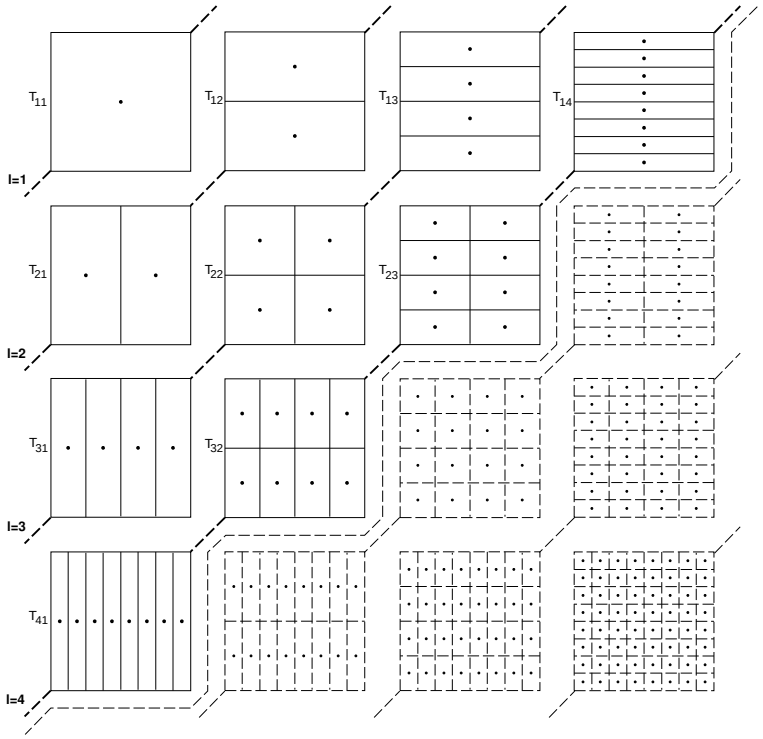


Abbildung 3.5: Teilraumschema mit Dünngitter–Leveln

berechnet sich nach folgender Formel:

$$l_{max} = 2 \cdot \mathrm{ld}(N-1) - 1. \tag{3.12}$$

Für den d–dimensionalen Fall ergibt sich die Formel:

$$l^d_{max} = d \cdot \mathrm{ld}(N-1) - d + 1. \tag{3.13}$$

Die Anzahl der Punkte reduziert sich abhängig von der Stufentiefe n von $O(4^n)$ auf $O(n \cdot 2^n)$. Dabei verschlechtert sich die Ordnung des Approximationsfehlers in der Maximumsnorm von $O(4^{-n})$ auf $O(n \cdot 4^{-n})$. In der Energienorm ist er sogar von derselben Ordnung $O(2^{-n})$ wie im vollen Gitter (Abb. 3.6).

Sowohl beim vollen als auch beim dünnen Gitter besteht die Möglichkeit, adaptiv zu verfeinern. Als Entscheidungskriterium zur Aufnahme eines Punktes dient die Feststellung der Relevanz dieses Punktes, wie sie in 3.1 dargestellt wurde. Dadurch erhält man im Gegensatz zum regulären dünnen Gitter ein *adaptiv verfeinertes dünnes Gitter* (Abb. 3.6).

Eine mögliche Datenstruktur für die Repräsentation eines dünnen Gitters im Rechner ist der Binärbaum. Dieser bietet sich an, da so ein dünnes Gitter effizient im Speicher gehalten werden kann. Außerdem ermöglicht die Verwendung einer dynamischen Datenstruktur eine einfache Realisierung der Adaptivität, da der Baum dort wachsen kann, wo dies erforderlich ist. Abbildung 3.7 zeigt einen

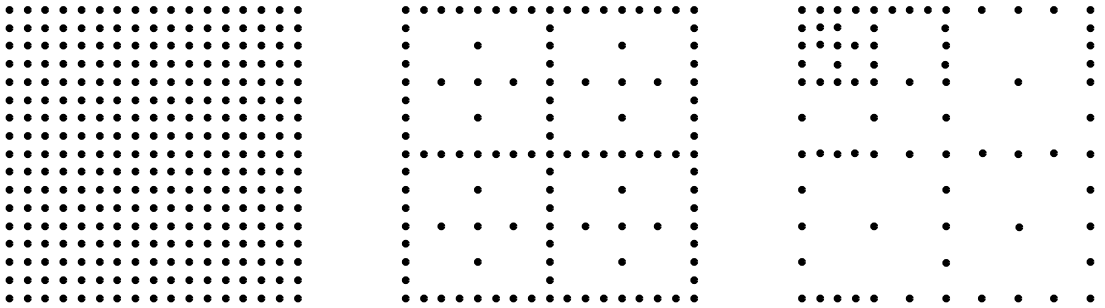


Abbildung 3.6: Reguläres volles und dünnes Gitter für $n = 4$ und adaptives verfeinertes dünnes Gitter

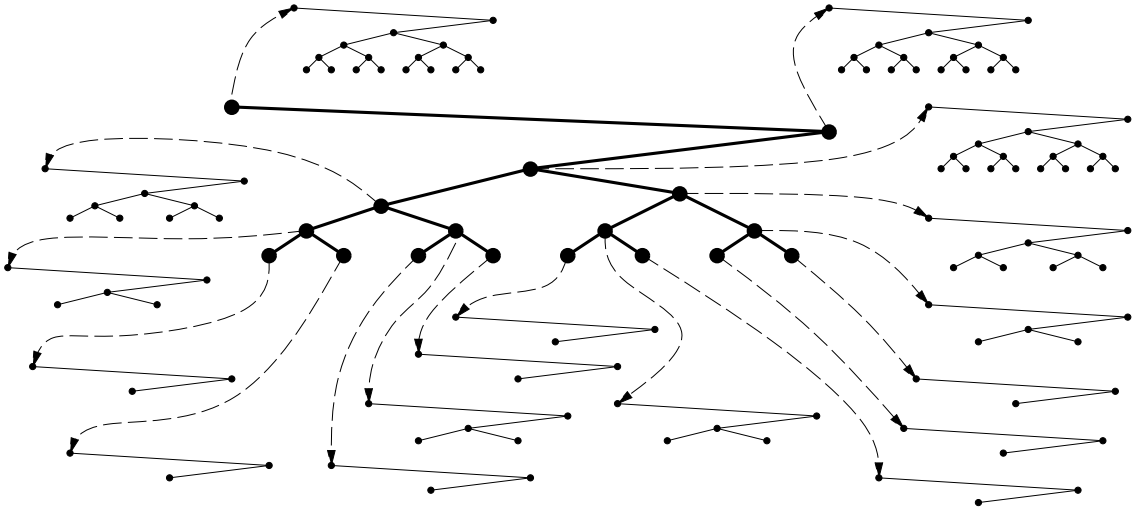


Abbildung 3.7: Zweidimensionaler Binärbaum für ein reguläres dünnes Gitter

Binärbaum für ein dünnes Gitter. Hierbei ist zu beachten, daß die Randpunkte gesondert behandelt werden müssen. Sie befinden sich in dieser Baumstruktur oberhalb der eigentlichen Wurzel. Das ist sinnvoll, da sie ja zur Berechnung des hierarchischen Überschusses im Wurzelknoten herangezogen werden.

Kapitel 4

Datenkompression mit stückweise linearen hierarchischen Basen

Um grundlegende Erfahrungen mit hierarchischen Methoden zur Datenkompression zu gewinnen, haben wir zunächst einfache zweidimensionale diskrete Datensätze betrachtet. Diese lassen sich mit gewissen Einschränkungen leicht als Graustufenbilder visualisieren. Deshalb bietet es sich an, gleich Bilder als Testdatensätze zu verwenden, die sich in der Diplomarbeit von K. Hiller [11]) als gut geeignet für die Anwendung der hierarchischen Datenkompression erwiesen haben. Somit wird der Wertebereich auf ganzzahlige Werte von 0 bis 255 (durch 8 bit darstellbar) eingegrenzt. Dies stellt jedoch keine Beschränkung der Allgemeinheit dar, da eine Erweiterung auf beliebige Wertebereiche ohne großen Aufwand möglich ist. Insbesondere für Daten, die bei der Behandlung von Video- bzw. Audiosignalen auftreten, sind nur geringe Anpassungen nötig. Die Implementierung der Dünngitteralgorithmen für Bilddaten hat darüber hinaus folgende Vorteile: Die Daten sind anschaulich, leicht darzustellen und zu verarbeiten. Vor allem aber fallen Fehler in der Regel sofort ins Auge, was sowohl bei der Entwicklung als auch bei der Bewertung der Algorithmen eine entscheidende Erleichterung darstellt.

4.1 Algorithmen und Datenstrukturen

Bilddaten sind naturgemäß rechteckig und können daher durch ein zweidimensionales Gitter dargestellt werden. Um die Dünngittermethode auf diese Daten anwenden zu können, beschränken wir uns hier auf quadratische $N \times N$–Gitter mit $N = 2^n + 1 (n \in I\!N)$ Punkten in jede Koordinatenrichtung. Im Speicher liegen diese Daten zeilenweise nacheinander angeordnet vor. Man spricht von einer linearisierten Datenstruktur (Abb. 4.1).

Die anfänglichen Betrachtungen diskreter Datensätze geschahen unter Verwendung der herkömlichen Dünngitteralgorithmen. Dazu wurde der vorliegende

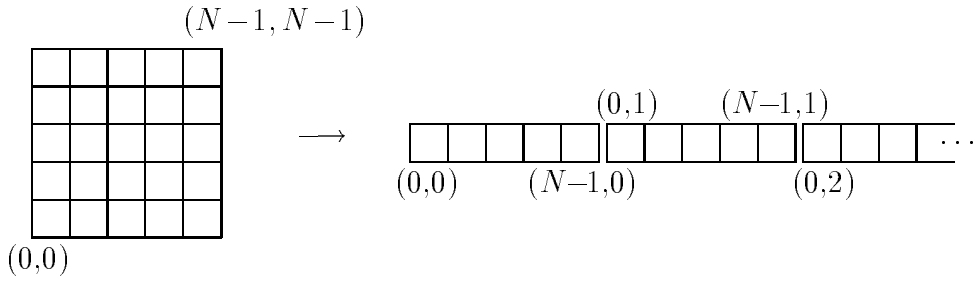


Abbildung 4.1: Linearisierung von Bilddaten

Datensatz vollständig in die hierarchische Datenstruktur eines Binärbaums konvertiert. Das Ausdünnen des Baums erfolgte dann von den Blättern zur Wurzel anhand der während der Konvertierung berechneten hierarchischen Überschüsse, wie von Hiller in [11] vorgeschlagen.

Bei der Implementierung der Algorithmen stellte sich bald heraus, daß der Übergang zu einem Binärbaum als Datenstruktur in unserem Fall ungünstig ist, da wir mit Bilddaten, d.h. auf fest vorgegebenen, nicht weiter verfeinerbaren Gittern arbeiten. Die benötigten Zeigerstrukturen erfordern zusätzlichen Speicherplatz und führen zu komplizierten Speicherzugriffen, die das Laufzeitverhalten negativ beeinflussen. Die entscheidenden Vorteile einer dynamischen Datenstruktur liegen in der uneingeschränkten Erweiterbarkeit und Speicherplatzersparnis bei ihrem sukzessiven Aufbau und haben daher in diesem Fall keine, bisweilen sogar negative Auswirkung. Aus diesem Grund wurden die Dünngitter–Algorithmen direkt auf der linearisierten Datenstruktur implementiert.

Das Festhalten an einer linearen Datenstruktur hat zur Folge, daß das hierarchische Konzept der Dünngittertheorie vollständig in die Algorithmen übernommen werden muß, da es sonst in der Baumdatenstruktur repräsentiert ist. Benötigt wird also eine Zugriffsstrategie, die sämtliche Punkte des Gitters in hierarchischer Reihenfolge erreicht. Ausgangspunkt für die folgenden Überlegungen ist das herkömmliche Teilraumschema (Abb. 3.5). Um sämtliche Punkte des Gitters hierarchisch zu erreichen, müssen alle Teilräume levelweise nacheinander durchlaufen werden. Der Level jedes Gitterpunkts ist durch dessen Koordinaten x, y eindeutig bestimmt. Die von diesem Level abhängigen Gitterweiten h_x und h_y legen die Größe der zur entsprechenden Basisfunktion gehörenden Trägerfläche fest. Da die Trägerflächen jedes Teilraums gleich sind, ändern sich auch h_x und h_y innerhalb eines Teilraums nicht. Es ergibt sich folgender Algorithmus zum Durchlauf eines Teilraums:

```
int N;

void teilraumdurchlauf(data, hx, hy)
    DataType data;
    int      hx, hy;
{
    float u;
    int x, y;

    for (y = hy;  y < N; y = y+2*hy) {
        for (x = hx;  x < N; x = x+2*hx) {
            u = get_point(data, x, y);
        }
    }
}
```

Betrachtet man h_x, h_y aller Teilräume, erhält man folgendes Schema für ein Gitter der Größe $N = 17$:

$$
\begin{array}{lccccccccc}
\mathrm{l} = 1: & & & & & 8,8 & & & & \\
\mathrm{l} = 2: & & & & 4,8 & & 8,4 & & & \\
\mathrm{l} = 3: & & & 2,8 & & 4,4 & & 8,2 & & \\
\mathrm{l} = 4: & & 1,8 & & 2,4 & & 4,2 & & 8,1 & \\
\mathrm{l} = 5: & & & 1,4 & & 2,2 & & 4,1 & & \\
\mathrm{l} = 6: & & & & 1,2 & & 2,1 & & & \\
\mathrm{l} = 7: & & & & & 1,1 & & & &
\end{array}
\tag{4.1}
$$

Das Produkt von h_x und h_y ist für jeden Teilraum eines Levels gleich. Der folgende Algorithmus, in dem der Durchlauf eines Levels durchgeführt wird, basiert auf dieser Eigenschaft.

```
#define L(d)   ((d) * log2(N - 1) - (d) + 1)    /* L = l_max */

void leveldurchlauf(data, l)
    DataType data;
    int      l;
{
    int hx, hy;
    int i;

    for (i = max(0, l - (L(1)-1)); i <= min(l, L(1)-1); i++) {
        hx = exp2( L(2)-L(1)-l-i );
        hy = exp2( L(1)-1-i );
```

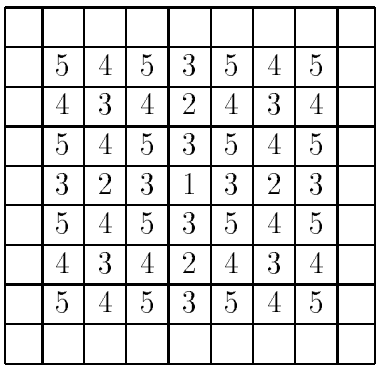


Abbildung 4.2: Levelweise markiertes Gitter

```
        teilraumdurchlauf(data, hx, hy);
    }
}
```

Für einen vollständigen Durchlauf durch das volle Gitter wird nur noch eine Schleife über sämtliche Level benötigt.

```
void gitterdurchlauf(data)
    Data data;
{
    int l;

    for (l = 0;  l < L(2); l++) {
        leveldurchlauf(data, l);
    }
}
```

`exp2` und `log2` seien spezielle Funktionen, die eine Zweierpotenz bzw. den Logarithmus zur Basis 2 berechnen und das Ergebnis in geeigneter Form runden.

Mit Hilfe der oben beschriebenen Funktionen ist ein hierarchischer Durchlauf durch einen linearen Datensatz bis auf den Rand vollständig realisiert (Abb. 4.2). Die Randbehandlung erfolgt prinzipiell genauso, nur eben eindimensional separat für jede Kante.

Die Entscheidung zur Elimination eines Punktes des Gitters erfolgt durch die Berechnung des hierarchischen Überschusses. Die Prozedur `teilraumdurchlauf` ändert sich folgendermaßen:

```
float eps;

void teilraumdurchlauf(data, hx, hy)
    DataType data;
    int      hx, hy;
{
    float u, v;
    int x, y;

    for (y = hy;  y < N; y = y+2*hy) {
        for (x = hx;  x < N; x = x+2*hx) {
            u = get_point(data, x, y);
            v = u - interpoliere(data, x, y, hx, hy);
            if (v < eps) eliminiere(data, x, y);
        }
    }
}
```

Eine in der Baumstruktur inhärente Information ist, ob ein Knoten Nachfolger besitzt oder nicht. Hat er keine Nachfolger, so bezeichnet man den Knoten als *Blatt*. Diese Information muß bei Verwendung einer linearen Struktur separat festgehalten werden. Die einfachste Möglichkeit ist ein Bitfeld, in dem beim Durchlauf durch das Gitter Punkte, die Nachfolger haben, markiert werden. Jedes Bit im Bitfeld repräsentiert dabei einen Wert im Datensatz bzw. einen Knoten im Baum. Ein gesetztes Bit verhindert die Elimination des entsprechenden Punktes.

Bei Bäumen bestand die Möglichkeit, Absolutwert und hierarchischen Überschuß in einem Knoten zu speichern. Da in der von uns verwendeten linearisierten Datenstruktur aber immer nur ein Wert zur Verfügung steht, erfolgt die Speicherung der Überschüsse in einem zusätzlichen Feld. Mit dieser Erweiterung ist die vollständige Übertragung der klassischen Dünngitteralgorithmen auf lineare Datenstrukturen durchgeführt. Man benötigt aber zusätzlich zum Datenfeld der Originaldaten ein entsprechendes Feld für die hierarchischen Überschüsse und ein Bitfeld gleicher Größe. Ebenso wie bei der Implementation auf Bäumen bei Hiller [11] verwendet man die geänderten Algorithmen zur Datenkompression nicht *top–down*, sondern *bottom–up*, damit sichergestellt ist, daß keien relevanten Punkte eliminiert werden. Dazu muß nur die Schleife in `gitterdurchlauf` umgekehrt durchlaufen werden (Abb. 4.3).

4.2 Anpassung der klassischen Algorithmen

Da davon auszugehen ist, daß bei der angestrebten Anwendung der Algorithmen Datenmengen von kritischem Umfang zu verarbeiten sind, war ein vorherrschen-

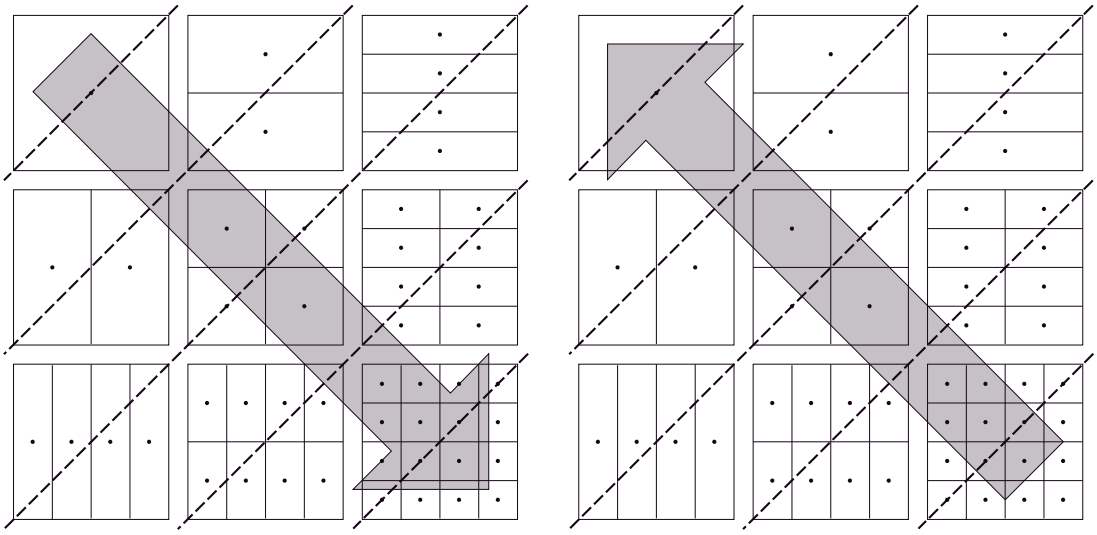


Abbildung 4.3: Top-Down- und Bottom-Up-Durchlauf durch das Teilraumschema

des Ziel, jeden zusätzlich benötigten Speicherplatz zu vermeiden.

Der zur Speicherung der Überschüsse notwendige Speicherplatz bedeutet neben dem Originaldatensatz einen erheblichen zusätzlichen Platzbedarf. Dies kann am besten vermieden werden, indem man die Überschüsse nach ihrer Berechnung am Ort der Originaldaten speichert.

4.2.1 Speicherung der Absolutwerte

Da Grauwertbilder einen Wertebereich von 0 bis 255 besitzen, liegt bei Verwendung stückweise linearer Basisfunktionen der Wertebereich der hierarchischen Überschüsse im schlechtesten Fall zwischen -510 und 510. Zur Darstellung wären also zwei zusätzliche Bits notwendig. Bei genügend glatten Funktionen reduziert sich die Anzahl der benötigten Bits zwar levelweise (s. Thurner [17]), bei Bildern kann aber nicht von dieser Glattheitsvoraussetzung ausgegangen werden. Daher stellt sich die Frage, ob es überhaupt sinnvoll ist, die hierarchischen Überschüsse zu speichern. Da das Ziel nach der Bearbeitung der Daten lediglich eine Rekonstruktion der Ausgangsdaten durch die Interpolation der Werte der verbliebenen Punkte ist, ist es ebenso möglich, die Überschüsse nur als Eliminationskriterium zu verwenden, sie aber nicht zu speichern. Das bedeutet, daß an den nicht eliminierten Punkten die Werte der Ausgangsdaten erhalten bleiben.

Die Identifikation der eliminierten Punkte ist bei Speicherung der hierarchischen Überschüsse dadurch gegeben, daß ein Überschuß der Größe Null genau einem eliminierten Punkt entspricht. Speichert man hingegen die absoluten Werte, so müssen die elimierten Punkte in dem Gitter explizit markiert werden. Leicht erreicht man dies z.B. mittels eines Bitfeldes oder durch eine Zahl außerhalb des Wertebereichs der Originaldaten. Aus Gründen der Speicherplatzersparnis ist die

zweite Variante vorzuziehen. Umfaßt der Wertebereich aber alle durch den Datentyp darstellbaren Werte, so muß eine Zahl innerhalb dieses Wertebereichs zur Identifikation eliminierter Punkte gewählt werden. In diesem Fall wird also der Wertebereich um einen Wert verringert. Jedes Vorkommen dieses ausgezeichneten Wertes muß daher im Originaldatensatz durch einen Wert des eingeschränkten Wertebereichs ersetzt werden. Am besten eignet sich dazu ein Wert, der durch einen anderen ersetzt werden kann, ohne dabei eine signifikante Änderung des Informationsgehaltes hervorzurufen. Erhöht man beispielsweise bei Grauwertbildern alle Werte Null auf Eins, so ist diese Änderung optisch kaum wahrnehmbar. Ein weiterer Vorteil durch die Speicherung der Absolutwerte ist die Tatsache, daß die nicht eliminierten Werte bei der Decodierung nicht mehr berechnet werden müssen. Hierdurch wird eine gewisse Beschleunigung des Decodiervorgangs erreicht.

Die Originaldaten werden bei dem oben beschriebenen Vorgehen an den eliminierten Punkten verändert. Es muß also sichergestellt werden, daß dies erst geschieht, wenn die Werte nicht mehr zur Berechnung des hierarchischen Überschusses noch nicht behandelter Punkte benötigt werden. Da zur Überschußberechnung ausschließlich hierarchische Vorgänger, also Punkte aus höherliegenden Leveln, herangezogen werden, ist dies bei Bottom–Up–Durchlauf sichergestellt.

4.2.2 Elimination hierarchischer Vorgänger

In der Baumstruktur erfolgt das Bottom–Up–Ausdünnen nur solange, bis Blätter aufgrund des Eliminationskriteriums behalten werden müssen. Darüberliegende Level werden dann nicht mehr betrachtet. Dies ist nicht anders möglich, da das Entfernen höherliegender Punkte ihre Nachfolger im Speicher isolieren und damit nicht wiederauffindbar machen würde.

Diese datenstrukturelle Abhängigkeit der Punkte besteht im linearisiert gespeicherten Gitter nicht. Daher ist es möglich, alle Punkte unabhängig von ihren Nachfolgern nur anhand des hierachischen Überschusses zu eliminieren. Es zeigt sich, daß dadurch auf eine nicht unerhebliche Anzahl von Punkten verzichtet werden kann, da diese nur für die Konsistenz der Datenstruktur notwendig sind, aber keine Information tragen.

Somit werden nun ausschließlich Punkte gespeichert, die zur Rekonstruktion des Datensatzes benötigt werden (Abb. 4.4).

4.3 Fehlerakkumulation und Kompensation

In jedem Punkt kann nach 3.1 beim Ausdünnen ein vorgegebener maximaler Interpolationsfehler zugelassen werden. Die Fehlerschranke ε ist zunächst konstant für alle Level. Bei der Interpolation eines eliminierten Wertes kann diese Fehlerschranke jedoch überschritten werden, wenn seine hierarchischen Vorgänger

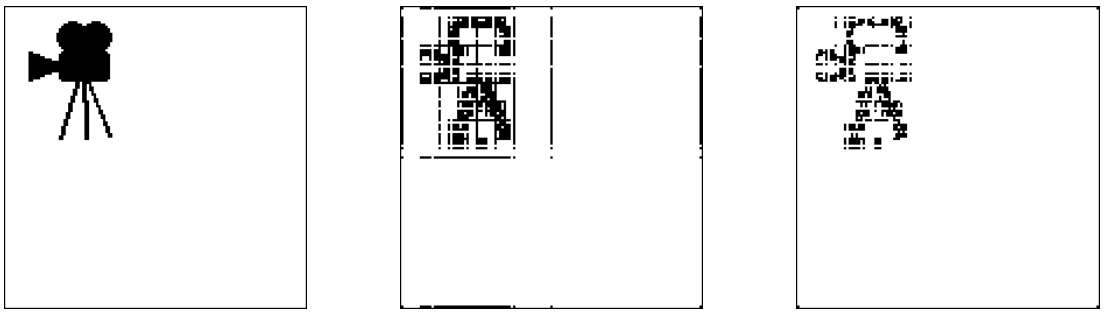


Abbildung 4.4: Originalbild und Gitter mit und ohne hierarchischen Vorgänger

ebenfalls interpoliert wurden. Da die vorangegangene Interpolation bereits mit einem maximalen Fehler ε behaftet sein kann, ist es möglich, daß sich der Fehler mit zunehmender Leveltiefe aufschaukelt. Diese Fehlerakkumulation kann eingeschränkt werden, indem man die Fehlerschranke in hierarchisch höherliegenden Leveln geringer wählt als in tieferliegenden. Am einfachsten ist eine vom Level l abhängige lineare Verringerung durch einen Kompensationsfaktor κ:

$$\varepsilon_l = \kappa^{lmax-l} \cdot \varepsilon \quad \text{für } \kappa \in [0,1]. \tag{4.2}$$

Am zweidimensionalen Fall soll exemplarisch dargestellt werden, wie sich ein geeigneter Faktor κ heuristisch bestimmen läßt. Dazu berechnet man eine obere Schranke für den Betrag e_l des Fehlers, der sich bis zum Level l aufgeschaukelt hat. Der Wert u_i^I eines interpolierten Punktes berechnet sich folgendermaßen:

$$u_i^I = \frac{1}{2}(u_{i_n}^I + u_{i_s}^I + u_{i_w}^I + u_{i_e}^I) - \frac{1}{4}(u_{i_{nw}}^I + u_{i_{ne}}^I + u_{i_{sw}}^I + u_{i_{se}}^I). \tag{4.3}$$

Eine obere Schranke an diesem Punkt ist also:

$$e_l \leq \varepsilon + \frac{1}{2}(4 \cdot e_{l-1}) + \frac{1}{4}(4 \cdot e_{l-2}) = \varepsilon + 2 \cdot e_{l-1} + e_{l-2}. \tag{4.4}$$

Die Aufschauklung von einem Level zum nächsten läßt sich folgendermaßen annehmen:

$$e_l \quad = \quad x \cdot e_{l-1}. \tag{4.5}$$

Setzt man $e_0 = e_1 = \varepsilon$, so ergibt sich für $l \geq 2$:

$$e_l \quad = \quad x^{l-1} \cdot \varepsilon \quad \text{mit } x > 1. \tag{4.6}$$

Damit kann man den Akkumulationsfaktor x folgendermaßen abschätzen:

$$e_l \quad = \quad x^{l-1} \cdot \varepsilon \quad \leq \quad \varepsilon + 2 \cdot e_{l-1} + e_{l-2}$$
$$\Downarrow$$

$$
\begin{aligned}
x^{l-1} \cdot \varepsilon &\leq \varepsilon + 2 \cdot x^{l-2} \cdot \varepsilon + x^{l-3} \cdot \varepsilon \\
x^{l-1} &\leq 1 + 2 \cdot x^{l-2} + x^{l-3} \\
x^{l-1} - 2 \cdot x^{l-2} - x^{l-3} &\leq 1 \\
x^{l-3}(x^2 - 2x - 1) &\leq 1 \\
x^{l-3}(x - (1+\sqrt{2}))(x - (1-\sqrt{2})) &\leq 1 \\
(x - (1+\sqrt{2})) &\leq \frac{1}{x^{l-3}(x - (1-\sqrt{2}))} \\
\text{für } l \to \infty: \quad x - (1+\sqrt{2}) &\leq 0 \\
x &\leq 1 + \sqrt{2}.
\end{aligned}
\tag{4.7}
$$

Es ergibt sich also asymptotisch folgende obere Schranke für die levelweise Akkumulation des Fehlers:

$$\frac{e_l}{e_{l-1}} \leq 1 + \sqrt{2}. \tag{4.8}$$

Der Faktor zur völligen Kompensation des Akkumulationsfehlers ist demnach für den zweidimensionalen Fall folgendermaßen zu wählen:

$$\kappa \geq \frac{1}{1+\sqrt{2}}. \tag{4.9}$$

Bei Bildern stellt man fest, daß es nicht unbedingt von Vorteil ist, den Akkumulationsfehler vollständig zu kompensieren, da der maximale Fehler nur selten eintritt und daher eine so scharfe Kompensation die Effektivität der Kompression negativ beeinflußt. Die Wahl des Kompensationsfaktor sollte sich daher in unserem Fall besser nach Kompressionsrate und Bildqualität richten.

4.4 Fehlerbewertung

Der Fehler, der bei der Kompression eines Bildes entsteht, entspricht der Differenz zwischen Originalbild und interpoliertem Bild. Die Bewertung kann nach mathematischen oder optischen Gesichtspunkten geschehen. Die objektive Beurteilung der Qualität eines Bildes ist ein häufiges Problem bei der Bildverarbeitung, welches bisher nicht zufriedenstellend gelöst ist. Auch die Beurteilung von von Expertengruppen entwickelten Methoden wie dem JPEG–Verfahren muß sich letztendlich auf das subjektive optische Urteil von Testpersonen stützen. Die Problematik wird am besten an folgendem Beispiel klar: Erhöht man sämtliche Werte eines Grauwertbildes um einen nicht zu großen konstanten Wert, so ist der optische Fehler sehr gering — das Bild erscheint lediglich ein wenig heller —, für den mathematischen Fehler kann sich dies aber katastrophal auswirken. Da die optische Beurteilung aber aufwendig ist, werden dennoch mathematische Methoden herangezogen, die aber jeweils nur einen Teilaspekt berücksichtigen und daher nur sehr bedingt zur Gesamtbewertung der Bildqualität geeignet sind.

Als Beispiele seien Fehlermaße wie Signal–Rausch–Abstand (Signal Noise Ratio, SNR), Fouriertransformation, relativer und maximaler Fehler sowie die Entropie eines Bildes genannt.

Auf der Suche nach einem einigermaßen aussagekräftigen Fehlermaß haben wir versucht, mathematische Repräsentationen für den Verlust an Struktur im Bild und den Gesamtfehler mit Berücksichtigung des hierarchischen Konzepts zu finden. Folgende Fehlermaße erwiesen sich unter diesen Gesichtspunkten für sinnvoll:

$$e_h := \frac{\sum_i |u_i^I - u_i| \cdot supp(\varphi_i)}{n^2 \cdot W}, \tag{4.10}$$

$$e_{L_2} := \frac{\sqrt{\sum_i |u_i^I - u_i|^2}}{n^2 \cdot W}. \tag{4.11}$$

Der Wertebereich W beträgt für Grauwertbilder $2^8 = 256$. Das erste Fehlermaß e_h beschreibt den absoluten Fehler an allen Bildpunkten gewichtet gemäß der Trägerfläche der jeweiligen Basisfunktion. Es stützt sich somit auf das hierarchische Prinzip der Dünngittermethode. Fehler hierarchisch höherliegender Punkte werden stärker gewichtet als tieferliegende. Das zweite Fehlermaß e_{L_2} gibt die diskrete L_2–Norm des Fehler wieder.

4.5 Codierung

Zum Zweck der Übertragung der Daten über einen Kanal bzw. der Speicherung auf ein Hintergrundspeichermedium müssen die Daten in ein der Zielsetzung entsprechendes Format gebracht werden. In unserem Fall bedeutet das, daß die Daten, die nach dem Ausdünnen nach wie vor denselben Platz im Speicher belegen, in eine speichereffizientere Form gebracht werden müssen. Da in der Gitter nach dem Ausdünnungsvorgang viele Werte eliminiert und die Stellen markiert worden sind und viele der markierten Stellen direkt aufeinanderfolgen, ist eine Lauflängencodierung die einfachste Form der Kompression solcher Daten. Im Gegensatz zu der in Abschnitt 2.2.3 allgemein beschriebenen Lauflängencodierung wird hier nicht die Länge der Ketten beliebiger Werte, sondern der Einfachheit halber nur die Kettenlänge aufeinanderfolgender Markierungen eliminierter Punkte gespeichert. Da in unserem Fall die eliminierten Werte markiert werden, indem man sie durch einen ausgezeichneten Wert ersetzt, geschieht die Lauflängencodierung in einem linearen Durchlauf durch das Gitter. Das Ergebnis ist eine Folge von Paaren, die aus einem nicht–eliminierten Wert und der Anzahl ihm nachfolgender eliminierter Werte bestehen. Daher benötigt man für die lauflängencodierte Darstellung die doppelte Anzahl an Werten im Vergleich zu den im Gitter verbliebenen Werten. Aus Gründen der Einfachheit verwenden wir 8 bit für die Darstellung der Kettenlängen von Folgen markierter Punkte.

5	**1**	0	0	**4**
1	0	0	0	0
0	0	**3**	0	0
0	**2**	**7**	0	0
1	0	0	**4**	**6**

5	0	**1**	2	**4**	0	**1**	6	**3**	3	**2**	0	**7**	2	**1**	2	**4**	0	**6**

Abbildung 4.5: Ausgedünntes Gitter und lauflängencodierte Darstellung

Abbildung 4.5 zeigt eine ausgedünntes Gitter und ihre lauflängencodierte Darstellung.

Der so erzeugte Datensatz kann nun mit Hilfe anderer Standardverfahren weiter komprimiert werden. Dazu wird hier das UNIX–Tools `gzip` eingesetzt. Eine weitere Kompression um 50% ist hierbei keine Seltenheit.

4.6 Optimierung

Ein angestrebtes Ziel ist u.a. auch der Einsatz der Algorithmen zur Verwendung im Rahmen von Echtzeitanwendungen, z.B. Videokonferenzen. Daher wurden auch Maßnahmen zur Laufzeitoptimierung getroffen. Wie oben bereits erwähnt, werden die Daten linearisiert im Speicher gehalten. Dies ermöglicht bei geeigneter Zugriffsreihenfolge eine Ausnutzung evtl. vorhandener Cachemechanismen.

Bisher wurden Gleitpunktzahlen bei der Berechnung des Eliminationskriteriums, also der Anwendung des Interpolationssterns, benutzt. Eine weitere Möglichkeit, das Laufzeitverhalten der Kompressionsalgorithmen für auf Integerwerte quantisierte Datensätze zu verbessern, ist ein Übergang in der Zahldarstellung zu Integerzahlen. Zu diesem Zweck werden die Brüche, die den Interpolationsstern bilden, mit dem kleinsten gemeinsamen Vielfachen ihrer Nenner multipliziert. In Abbildung 4.6 wird dieser Übergang am Beispiel des Interpolationssterns für die stückweise lineare hierarchische Basis gezeigt. Bei sämtlichen dabei vorkommenden Faktoren handelt es sich um Zweierpotenzen. Das bedeutet, daß Bitshift–Operationen, die einen Geschwindigkeitsvorteil gegenüber herkömmlichen Multiplikationen bzw. Divisionen darstellen, zu ihrer Berechnung verwendet werden können.

4.7 Ergebnisse, Anwendungen und Ausblick

Um beurteilen zu können, wie sich die entwickelten Algorithmen bei der Kompression verschiedener Bilddaten verhalten, wurden mehrere Testdatensätze aus-

$$\begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Abbildung 4.6: Ganzzahliger Interpolationsstern für stückweise bilineare Basis

gewählt, die jeweils repräsentativ für eine bestimmte Klasse von Grauwertbildern sein sollen. Abbildungen der Datensätze befinden sich in Anhang C.1. Die wichtigste Gruppe von Bildern sind sog. natürliche Bilder, die meist durch photographische Aufnahme erzeugt werden. Als Vertreter wurde die Photographie eines Gebirgszuges gewählt (`landscape`). Ein Cartoon, der sich durch große Flächen gleicher Helligkeit auszeichnet, dient als Beispiel für einfache Zeichnungen (`loriot`). Eine im Bereich der Computergraphik wichtige Gruppe stellen die synthetisch erzeugten Bilder dar. Sie sind im Gegensatz zu den vorangegangenen Datensätzen nicht mit einem Meßfehler, der bei der Digitalisierung entsteht, behaftet. Hierfür dient als Vertreter das Beispielbild `schiff`. Als Datensätze, die Extremsituationen für den Algorithmus darstellen, wurden zum einen ein Schachbrettmuster (`chess`), zum anderen ein schwarzes Kreuz auf weißem Hintergrund (`kreuz`) gewählt. Diese lassen sich an den Sprungstellen des Datensatzes mit der hier verwendeten Methode nur schlecht interpolieren.

Zusätzlich wurde noch ein aus Höhendaten generiertes Bild (`hoehen`) untersucht, da sich Höhendaten in der Regel durch besondere Glattheitseigenschaften auszeichnen.

Bewertet werden die Bilder nach den in Abschnitt 4.4 eingeführten Fehlermaßen e_h (4.10) und e_{L_2} (4.11) sowie der größten Differenz zwischen Original- und interpoliertem Wert, dem maximalen Fehler e_{max}. Als Kompressionsrate dient in unserem Fall das Verhältnis von voller Punktanzahl zur Anzahl der nach der Ausdünnung verbliebenen Punkte. Das wichtigste Kriterium bei der Bewertung der Bildqualität bleibt der optische Eindruck, dieser kann hier nur informell wiedergegeben werden.

Für jedes der Testbilder wurden neben zusätzlichen Informationen die Anzahl verbliebener Punkte, die Kompressionsrate in Prozent sowie die Fehlermaße ausgegeben. Diese Daten wurden für unterschiedliche Varianten des Verfahrens für verschiedene Fehlerschranken ε berechnet.

Die ersten Untersuchungen beschäftigen sich mit der Frage, ob die von uns vorgeschlagene Methode bessere Ergebnisse liefert als die klassische Methode von K. Hiller ([11]), bei der die hierarchischen Vorgänger wegen der Benutzung einer Baumstruktur behalten werden müssen. Weil wir diese Grenze überschreiten und auch noch alle möglichen hierarchischen Vorgänger eliminieren, haben wir unsere Methode die **Rubikon**–Methode genannt. Abbildung 4.8 zeigt die Kompressionsraten der beiden Verfahren im Bezug auf die Fehlermaße für das Beispielbild

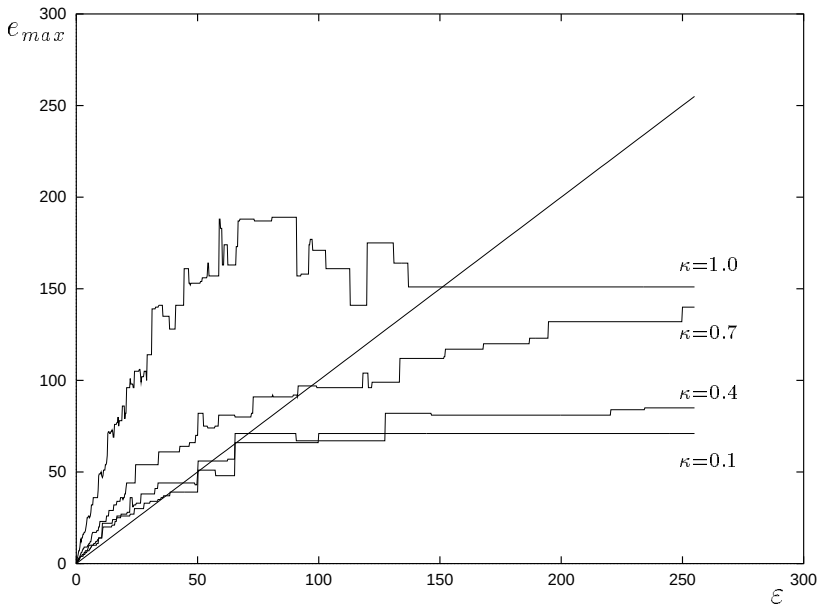


Abbildung 4.7: Fehlerakkumulation bei unterschiedlich starker Kompensation

`landscape`. Abgesehen vom hierarchischen Fehler liefert die Rubikon–Methode bei gleicher Kompressionsrate einen geringeren Fehler. Dies spiegelt sich ebenso in der Qualität der Bilder wider. Auch bei den sämtlichen anderen Testbildern ist dies der Fall. Daher wird die klassische Methode nicht weiter verfolgt.

Nun wird die Stärke der Fehlerakkumulation und ihre bestmögliche Kompensation untersucht. In Abbildung 4.7 ist die Fehlerakkumulation ohne Kompensation ($\kappa = 1.0$) sowie bei einem Kompensationsfaktor κ von 0.1, 0.4 und 0.7 für das Beispielbild `landscape` dargestellt. Ein κ von 0.4 entspricht dabei etwa dem errechneten Wert (4.9). Um den besten Kompensationsfaktor zu erhalten, wird für verschiedene Kompressionsraten der minimale Fehler in den drei Fehlermaßen ermittelt (Abb. 4.8). Hierbei zeigt sich, daß der optimale Kompensationsfaktor für jedes Fehlermaß ein anderer ist. Eine zusätzliche Kontrolle des optischen Eindrucks bestätigt einen Kompromiß zwischen den bezüglich der verschiedenen Fehlermaße optimalen Kompensationsfaktoren. Dieser liegt bei einem κ von ca. 0.7. Vergleicht man die Kompressionsraten dieser kompensierten Variante mit der klassischen Methode, so stellt man fest, daß jetzt alle, auch der hierarchische Fehler, kleiner sind.

Ermittelt man nach optischem Eindruck, unter Berücksichtigung der verschiedenen Fehlermaße, für die anderen Testbilder ein optimales κ, so ergibt sich für `loriot` ein κ von etwa 0.8 und für `schiff` ist $\kappa = 0.4$. Die optischen Beurteilung der Bildqualität für das Testbild `hoehen` gestaltet sich schwierig, da das Bild recht kontrastarm ist, und so Unterschiede durch optische Kontrolle nur schwer bestimmt werden können. Ungefähr bei einem Faktor von $\kappa = 0.9$ schien die Qualität etwas besser zu sein als bei den anderen getesteten Kompensationsfaktoren.

Da bei diesem Datensatz die Fehlermaße, insbesondere der maximale Fehler, von größerem Interesse sind, läßt sich die Aussage machen, daß für Höhendaten ein κ von 0.6 am besten geeignet ist, da hier der maximale Fehler am geringsten war. Für die Testdatensätze `chess` und `kreuz` läßt sich kein sinnvolles optimales κ angeben, da sich aufgrund der Gestaltung dieser Datensätze zu wenig Unterschiede bei Verwendung verschiedener Kompensationsfaktoren ergeben.

Es zeigt sich also, daß sich für jedes Bild ein anderes κ als optimal erweist. Eine allgemeingültige Bestimmung eines für alle Bilder optimalen Kompensationsfaktors ist also nicht möglich. Abbildung 4.9 zeigt die Ergebnisse für die verschiedenen Beispielbilder bei jeweils optimalen Kompensationsfaktoren.

Die Bilddaten des Vertreters für natürliche Bilder (`landscape`) können bei Inkaufnahme einer nur geringfügig schlechteren Bildqualität auf etwa 15% der ursprünglichen Bildpunkte komprimiert werden. Bei dem Cartoon (`loriot`) ergibt sich ein ähnlicher Kompressionsfaktor wie bei den natürlichen Bildern. Bei stärkerer Kompression werden horizontale und vertikale Linien sichtbar („Gobelin–Effekt"), die auf den teilweise sehr langen und schmalen Trägern der Basisfunktionen beruhen. Dies ist eine spezielle Eigenheit dieser Kompressionmethode. Abbildung 4.10 zeigt diesen Effekt bei hoher Fehlerschranke. An sich wäre wegen der großen Flächen gleichmäßiger Helligkeit eine bessere Kompression zu erwarten. Das Auge ist bei solchen Flächen aber Fehlern gegenüber deutlich weniger tolerant, so daß mehr Punkte investiert werden müssen, um den Qualitätsansprüchen zu genügen. Bei synthetischen Bildern (`schiff`) ist sichergestellt, daß Flächen gleicher Helligkeit auch den exakt gleichen Grauwert besitzen. Daher kann eine solche Fläche erheblich besser interpoliert werden. Das Beipielbild läßt sich ohne Qualitätsverlust auf 7.34% der Punkte komprimieren. Andererseits werden ebenso wie bei dem Cartoon geringe Fehler schneller wahrgenommen. Bei den glatteren Höhendaten erreicht man Kompressionsraten von 10 %, ohne eine sichtbare Verschlechterung festzustellen. Aus diesem Grund wird dieser spezielle Typ von Daten zur Bearbeitung mit der Dünngittermethode bereits intensiver untersucht. Die Ergebnisse bezüglich des Testbildes `chess` und der Testfunktion `kreuz` unterscheiden sich, wie zu erwarten, stark von den Ergebnissen der anderen Bilder. Wie bei dem synthetischen Bild lassen sich diese Bilder ohne Qualitätsverlust bereits extrem komprimieren. Der Fehler ändert sich bei wachsendem ε zunächst nicht, dann aber sehr sprunghaft. Meist ist auch der Qualitätsverlust dann sehr hoch.

Für die Speicherung der Daten auf einem Hintergrundspeicher wird das in 4.5 dargestellte Verfahren der Lauflängencodierung verwendet. Dabei werden zwar nur noch die benötigten Punkte abgespeichert, aber zusätzlich ebensoviel Lauflängeninformation. Eine anschließende Kompression durch das UNIX–Tool `gzip` führt bei natürlichen Bildern zu einer Gesamtkompression von etwa einem Fünftel der Ausgangsdaten.

In dieser Form ist die hier verwendete Methode noch nicht konkurrenzfähig, es gibt jedoch zahlreiche Ansätze für Verbesserungen. Dazu gehören z.B. eine

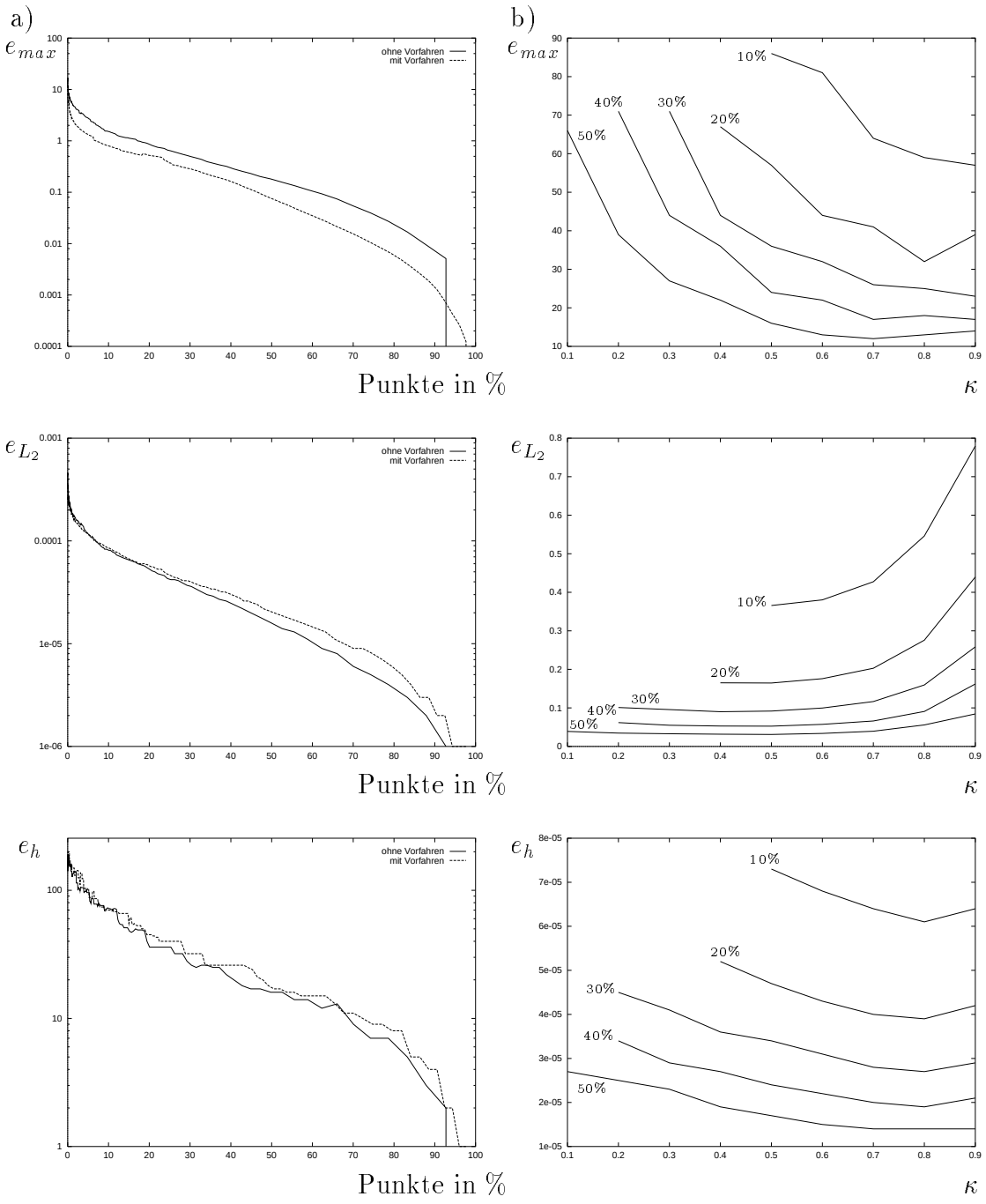


Abbildung 4.8: Vergleich der beiden Verfahren anhand der Fehlermaße (a) und Fehler in Abhängigkeit von κ für verschiedene Kompressionsraten (b)

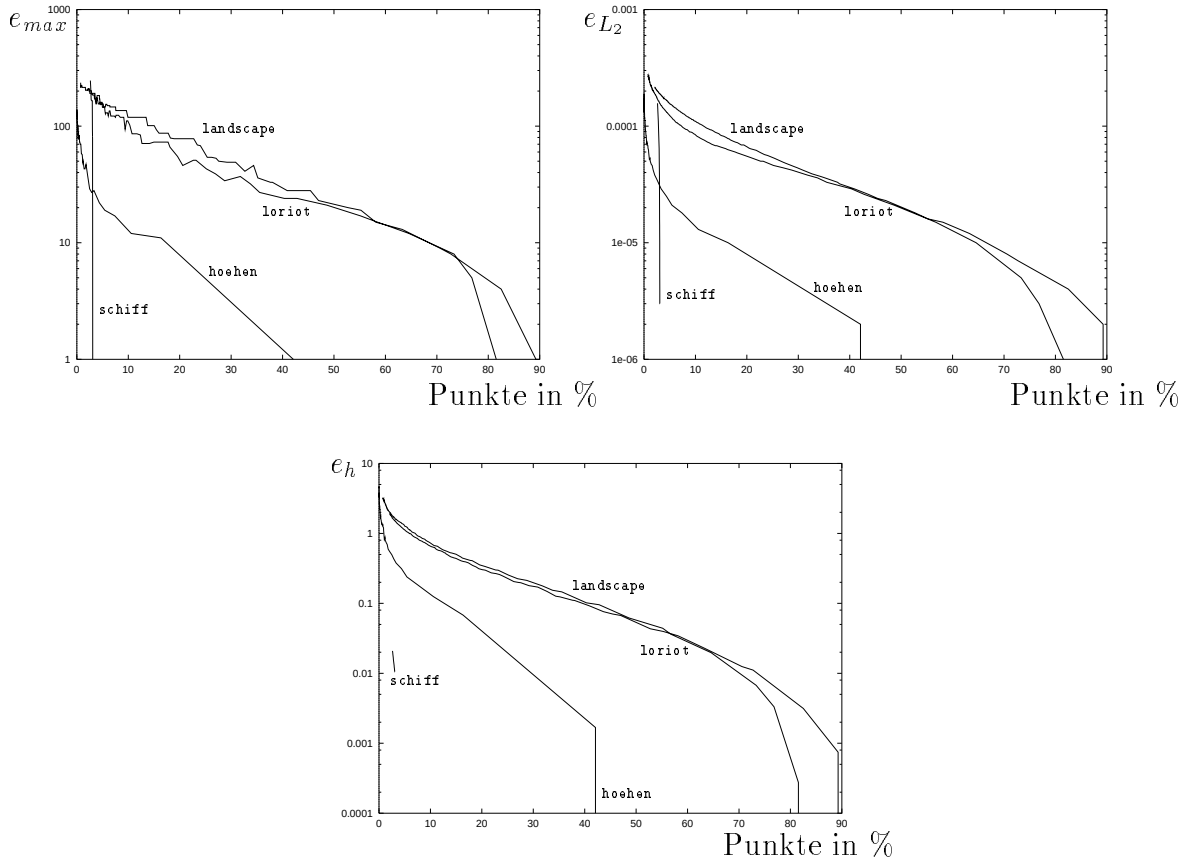


Abbildung 4.9: Vergleich der Testergebnisse für die verschiedenen Beispielbilder

Abbildung 4.10: „Gobelin–Effekt“ (2% der Punkte)

Vorglättung der Daten, andere Codierungsverfahren nach der Ausdünnung, die das hierarchische Prinzip stärker nutzen, oder die Wahl anderer Basisfunktionen oder Hierarchisierungsstrategien.

Als Anwendungsgebiete dieser Methode bieten sich Problemstellungen an, bei denen eine grobe Approximation zu einem frühen Zeitpunkt, die dann sukzessive verfeinert wird, von Interesse ist. Ein solches Vorgehen bietet sich bei der Übertragung von Kamerabildern für Videokonferenzen an, da hier die Bandbreite beschränkt ist und Schwankungen unterliegt. Die einelnen Level der Bilder können sukzessive übertragen und angezeigt werden, so daß zunächst selbst bei geringer Übertragungsrate schnell die groben Umrisse des Gesprächspartners erkennbar werden. Die Anzahl der zu übertragenden Level kann dabei leicht an die aktuelle Bandbreite angepaßt werden. Bei Flugsimulatoren kann nach ähnlichem Prinzip die Darstellung von Feinstrukturen der Umgebung in Abhängigkeit von der Entfernung zum Betrachter gewählt und so der Aufwand für die graphische Darstellung signifikant reduziert werden.

Kapitel 5

Polynome höheren Grades als alternative Basisfunktionen

Bisher hat sich die Untersuchung der Interpolationsmethoden auf stückweise lineare hierarchische Basisfunktionen beschränkt. Aus der numerischen Mathematik aber ist bekannt, daß man eine Fülle unterschiedlicher Basisfunktionen mit ebenso verschiedenen Eigenschaften zur Interpolation von Funktionen heranziehen kann. Daher ist es sicher sinnvoll, sich einige Gedanken über alternative Basisfunktionen zu machen. Zunächst wird die Klasse der Basisfunktionen von Polynomen höheren Grades betrachtet, da sie sich durch eine einfache Funktionsvorschrift auszeichnen. Das Potential hierarchischer Basen zweiten und dritten Grades in Zusammenhang mit Dünngitterinterpolation wurde bereits von Bungartz in [4] aufgezeigt. Der Grund, warum Basisfunktionen höheren Grades bisher nicht weiter untersucht wurden, liegt wohl darin, daß die bisherigen Dünngitter–Anwendungen für hierarchische Basen, wie etwa partielle Differentialgleichungen, unter Zuhilfenahme höhergradiger Polynome sehr schnell kompliziert werden. Mit dem Einsatz der hierarchischen Basen lediglich als Interpolationstechnik zur Datenkompression ist es hingegen leicht möglich, komplexere Basen zu entwickeln und ihre Qualitäten zu bewerten. Durch den Übergang in der Datenstruktur von Bäumen auf linearisierte Felder, wie er in 4.1 erläutert wurde, ist es zudem sehr viel einfacher, Algorithmen zur Berechnung und zum Einsatz solcher Basisfunktionen zu erstellen.

5.1 Hierarchische Basen höheren Grades

5.1.1 Konstruktion hierarchischer Polynombasen

Als einen möglichen Ansatz zur Entwicklung hierarchischer Basen vom Grad $p \geq 1$ fordert man, daß die Basisfunktionen zunächst dieselben Bedingungen

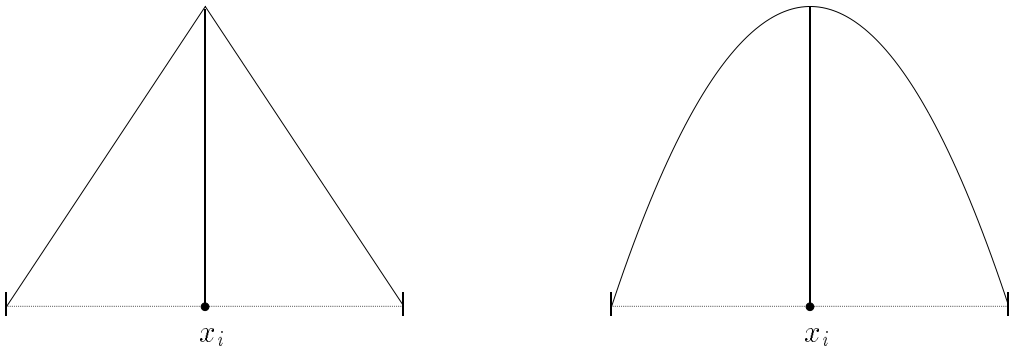


Abbildung 5.1: Stückweise lineare und quadratische Basisfunktion

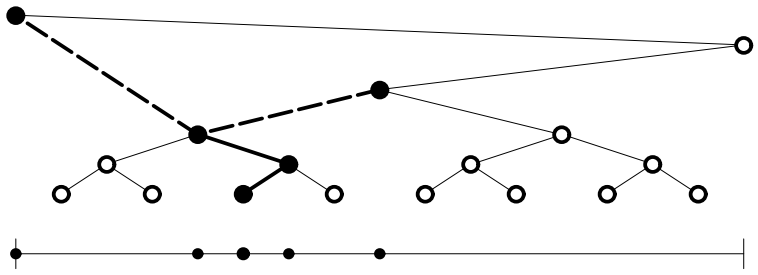


Abbildung 5.2: Wahl der Interpolationspunkte (Beispiel für $p = 4$)

erfüllen sollen, wie bereits die stückweise linearen Basisfunktionen (vgl. 3.5).

$$\begin{aligned} \varphi_i(x_i) &= 1, \\ \varphi_i(x_i \pm h_l) &= 0, \\ \mathrm{supp}(\varphi_i) &= [x_i - h_l, x_i + h_l]. \end{aligned} \tag{5.1}$$

φ_i sei nun allgemein eine Basisfunktion vom Grad p. Für $p = 1$ ist dies gerade die aus 3.1 bekannte stückweise lineare Basisfunktion. Man erhält diese Basisfunktion z.B. durch einen Lagrange–artigen Ansatz zur Erzeugung der Interpolationsfunktion zwischen den durch obige Bedingungen festgelegten Punkten. Da eine lineare Interpolationsfunktion bereits durch zwei Punkte vollständig bestimmt ist, muß man sich in diesem Fall mit auf den Halbintervallen stückweise definierten Funktionen behelfen, um eine volle Basisfunktion zu erhalten. Im quadratischen Fall $p = 2$ ist die Interpolationsfunktion vollständig bestimmt. Abbildung 5.1 zeigt eine stückweise lineare und eine quadratische Basisfunktion.

Für den Fall $p \geq 3$ ist die Interpolation aber unterbestimmt. Die zusätzlichen Freiheitsgrade müssen durch eine Erweiterung der Interpolationsbedingungen fixiert werden. Eine Möglichkeit, dies zu erreichen, ist die Betrachtung von $p + 1$

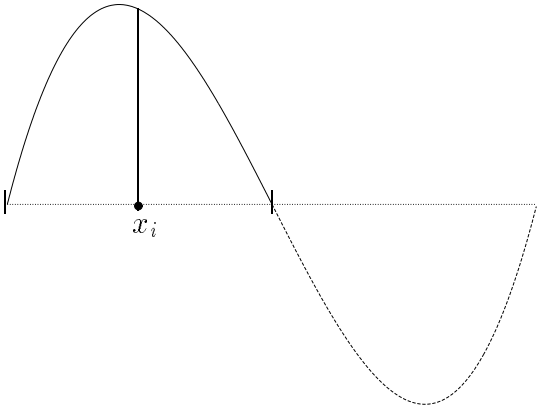


Abbildung 5.3: Basisfunktion dritten Grades

Stützpunkten. Man wählt zusätzlich zu x_i seine $p-2$ hierarchischen Vorgänger sowie die beiden Randpunkte des Trägers des hierarchisch höchsten dieser Punkte (Abb. 5.2). Für diese Punkte sollen nun ebenfalls als Interpolationbedingung die Bedingungen (5.1) gelten, zusätzlich seien die Werte an den hinzugenommenen Stellen ebenfalls gleich Null. Obwohl Punkte außerhalb von $\mathrm{supp}(\varphi_i)$ zur Konstruktion der Basisfunktion herangezogen werden, wird die Trägerfläche auf das ursprüngliche Intervall, den Träger der stückweise linearen Basisfunktion, beschränkt. Dies hat zur Folge, daß sich auch das Teilraumschema für Polynombasen höheren Grades nicht ändert. Die durch Interpolation zwischen diesen Punkten erhaltenen Basisfunktionen werden nun wie in 3.1 entsprechend hierarchisch angeordnet. Eine Besonderheit stellen die Level $l < p-1$ dar. Hier kann keine so konstruierte Basisfunktion p–ten Grades aufgesetzt werden, da für diese nicht ausreichend hierarchische Vorgänger vorhanden sind. Man behilft sich also mit einer Basisfunktion von größtmöglichem Grad $l+1$. Somit erhält man hierarchische Polynombasen beliebigen Grades. Abbildung 5.3 zeigt eine so konstruierte kubische Basisfunktion, Abbildung 5.4 die damit erstellte hierarchische Basis dritten Grades.

5.1.2 Bestimmung des Interpolationsoperators

Um den Übergang von einer absoluten Darstellung der Stützwerte in eine hierarchische Darstellung vollziehen zu können, ist es notwendig, den dazugehörigen Interpolationsoperator zu bestimmen. Bei der stückweise linearen Basis liegt der Operator auf der Hand, bei höhergradigen Polynombasen ist es aufwendiger, diesen Operator zu bestimmen. Um eine allgemeine Methode zur Erstellung der Interpolationsoperatoren höhergradiger Polynombasen zu finden, ist es ratsam, die Zusammenhänge von Basisfunktionen und Interpolationsoperatoren genauer

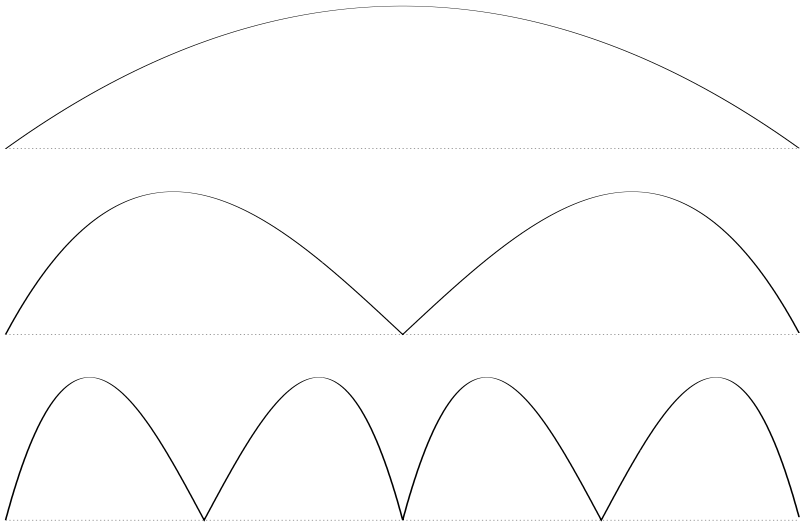


Abbildung 5.4: Hierachische Basis dritten Grades

zu untersuchen. Dazu betrachtet man zunächst den Zusammenhang zwischen den beiden eindimensionalen stückweise linearen Basen. Abbildung 5.5 zeigt die bekannte stückweise lineare hierarchische Basis $\{\varphi_1, \ldots \varphi_N\}$ und die entsprechende Stützpunktbasis $\{\psi_1, \ldots \psi_N\}$, die entsprechend der hierarchischen Basis ebenfalls hierarchisiert angeordnet und numeriert ist.

Gesucht ist eine Basistransformationsmatrix T, die die Darstellung $u \in I\!R^N$ einer stückweise linearen Funktion bezüglich der Stützpunktbasis abbildet auf eine Darstellung $v \in I\!R^N$ derselben Funktion bezüglich der hierarchischen Basis:

$$\begin{aligned} T : I\!R^N &\mapsto I\!R^N, \\ T \cdot u &= v. \end{aligned} \tag{5.2}$$

Die Spalten der Matrix T sind dabei bekanntlich die Bilder der Basisfunktionen ψ_i dargestellt als Summe der φ_i. Ohne Berücksichtigung des Randes, also $N = 2^n - 1$, erhält man folgende Transformationsmatrix für $N = 7$ ($n = 3$):

$$T = \left(\begin{array}{c|cc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline -\frac{1}{2} & 1 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & -\frac{1}{2} & 0 & 1 & 0 & 0 & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & 1 \end{array}\right). \tag{5.3}$$

Die Transformation der Darstellung von der hierarchischen Basis in die Stütz-

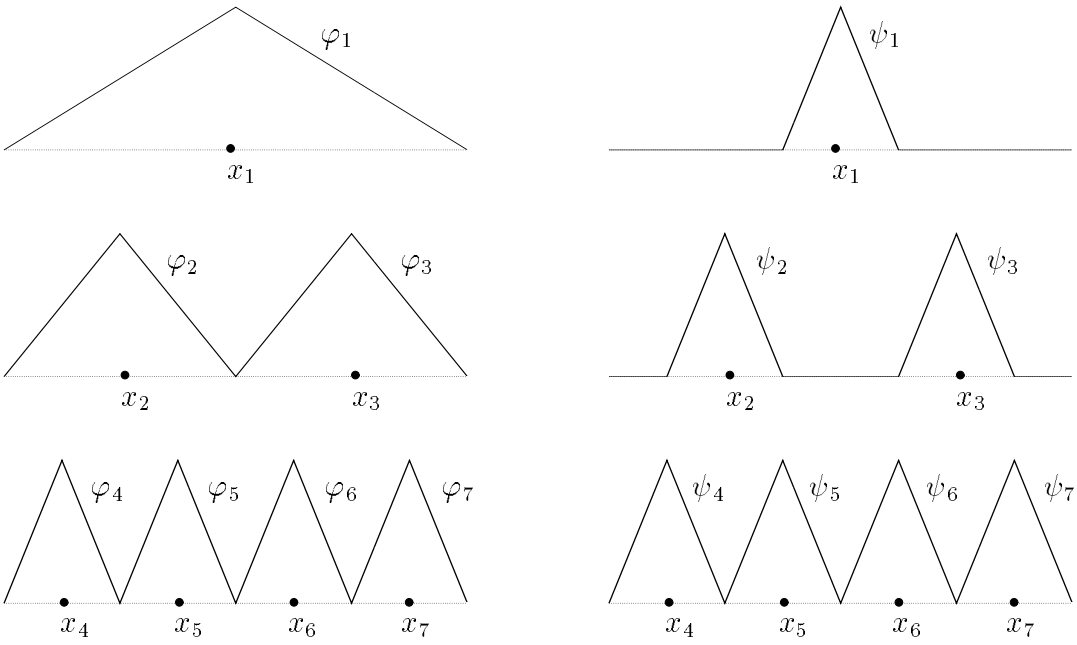


Abbildung 5.5: Stückweise lineare hierarchische Basis und Stützpunktbasis

punktbasis erfolgt durch die inverse Transformationsmatrix:

$$\begin{aligned} T^{-1} : I\!R^N &\mapsto I\!R^N, \\ T^{-1} \cdot v &= u. \end{aligned} \tag{5.4}$$

Wiederum im Fall $n = 3$:

$$T^{-1} = \left(\begin{array}{c|cc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \frac{1}{2} & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline \frac{1}{4} & \frac{1}{2} & 0 & 1 & 0 & 0 & 0 \\ \frac{3}{4} & \frac{1}{2} & 0 & 0 & 1 & 0 & 0 \\ \frac{3}{4} & 0 & \frac{1}{2} & 0 & 0 & 1 & 0 \\ \frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 & 1 \end{array}\right). \tag{5.5}$$

Um die hierarchischen Überschüsse aus den Absolutwerten berechnen zu können, benutzt man den in Formel 3.9 eingeführten Interpolationsoperator.

$$v_i = \left[\begin{array}{ccc} -\frac{1}{2} & 1 & -\frac{1}{2} \end{array} \right] \cdot u. \tag{5.6}$$

Betrachtet man die Zeilen der Matrix T, so ergeben die Koeffizienten der im Gitter benachbarten Punkte die Faktoren des Interpolationsoperators, wie das nach der Definition von T auch zu erwarten war.

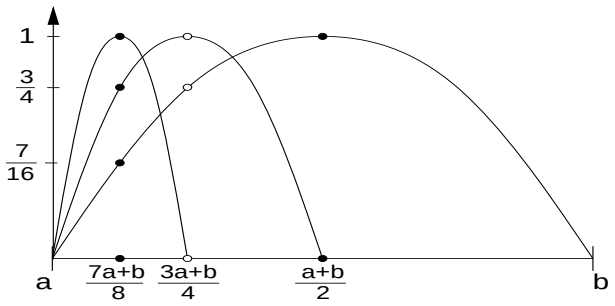


Abbildung 5.6: Berechnung der hierarchischen Überschüsse

Man hat also nun für den linearen Fall ein Verfahren, mit dem es möglich ist, aus vorgegebener Stützpunktbasis und hierarchischer Basis die Basistransformationsmatrix zu konstruieren und daraus den Interpolationsoperator zu exzerpieren. Im Fall $p \geq 2$ ist ausgehend von einer bestimmten hierarchischen Basis die zugehörige Stützpunktbasis jedoch nicht bekannt. Durch Festlegung der hierarchischen Basis sind aber Interpolationsprozeß und -operatoren und damit auch Transformationsmatrix und Stützpunktbasis eindeutig bestimmt, wenn sich wie im linearen Fall die Interpolationsoperatoren in den Zeilen der Matrix T befinden sollen. Daher benutzt man den Interpolationsprozeß, um die Operatoren sowie die Transformationsmatrix zu bestimmen. Man erhält von Level zu Level den hierarchischen Überschuß jedes Punktes, indem man den Wert der Basisfunktion an dieser Stelle nimmt und davon die Anteile sämtlicher hierarchischen Überschüsse der Basisfunktionen höherer Level subtrahiert. Drückt man diese hierarchischen Überschüsse durch die Funktionswerte der hierarchisch benachbarten Punkte aus, so erhält man die Interpolationsvorschrift und damit auch den Interpolationsoperator.

Am Beispiel der quadratischen hierarchischen Basis (Abb. 5.6) wird der Interpolationsprozeß skizziert (O.B.d.A. $u_a = u_b = 0$):

$$
\begin{aligned}
v_{\frac{a+b}{2}} &= u_{\frac{a+b}{2}}, \\
v_{\frac{3a+b}{4}} &= u_{\frac{3a+b}{4}} - \frac{3}{4} v_{\frac{a+b}{2}} = u_{\frac{3a+b}{4}} - \frac{3}{4} u_{\frac{a+b}{2}}, \\
v_{\frac{a+3b}{4}} &= u_{\frac{a+3b}{4}} - \frac{3}{4} v_{\frac{a+b}{2}} = u_{\frac{a+3b}{4}} - \frac{3}{4} u_{\frac{a+b}{2}}, \\
v_{\frac{7a+b}{8}} &= u_{\frac{7a+b}{8}} - (\frac{7}{16} v_{\frac{a+b}{2}} + \frac{3}{4} v_{\frac{3a+b}{4}}) = u_{\frac{7a+b}{8}} + \frac{1}{8} u_{\frac{a+b}{2}} - \frac{3}{4} u_{\frac{3a+b}{4}}, \qquad (5.7) \\
v_{\frac{5a+3b}{8}} &= u_{\frac{5a+3b}{8}} - (\frac{15}{16} v_{\frac{a+b}{2}} + \frac{3}{4} v_{\frac{3a+b}{4}}) = u_{\frac{5a+3b}{8}} - \frac{3}{8} u_{\frac{a+b}{2}} - \frac{3}{4} u_{\frac{3a+b}{4}}, \\
v_{\frac{3a+5b}{8}} &= u_{\frac{3a+5b}{8}} - (\frac{15}{16} v_{\frac{a+b}{2}} + \frac{3}{4} v_{\frac{a+3b}{4}}) = u_{\frac{3a+5b}{8}} - \frac{3}{8} u_{\frac{a+b}{2}} - \frac{3}{4} u_{\frac{a+3b}{4}}, \\
v_{\frac{a+7b}{8}} &= u_{\frac{a+7b}{8}} - (\frac{7}{16} v_{\frac{a+b}{2}} + \frac{3}{4} v_{\frac{a+3b}{4}}) = u_{\frac{a+7b}{8}} + \frac{1}{8} u_{\frac{a+b}{2}} - \frac{3}{4} u_{\frac{a+3b}{4}}, \quad \ldots
\end{aligned}
$$

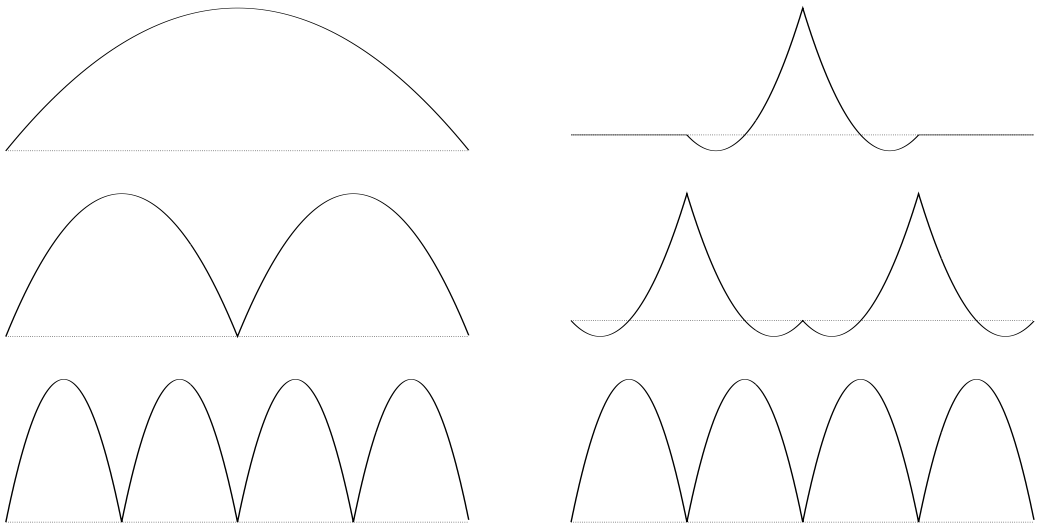


Abbildung 5.7: Quadratische hierarchische Basis und Stützpunktbasis

Die Interpolationsoperatoren lauten somit:

$$\begin{aligned} v_i &= \left[\begin{array}{ccccc} -\frac{3}{8} & 1 & -\frac{3}{4} & 0 & \frac{1}{8} \end{array} \right] \cdot u, \\ \text{bzw. } v_i &= \left[\begin{array}{ccccc} \frac{1}{8} & 0 & -\frac{3}{4} & 1 & -\frac{3}{8} \end{array} \right] \cdot u. \end{aligned} \tag{5.8}$$

Trägt man die Anteile der Überschüsse bzw. Absolutwerte als Zeilen in die Matrizen ein, so erhält man die resultierenden Transformationsmatrizen für die Basistransformation.

$$T^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{3}{4} & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{3}{4} & 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{7}{16} & \frac{3}{4} & 0 & 1 & 0 & 0 & 0 \\ \frac{15}{16} & \frac{3}{4} & 0 & 0 & 1 & 0 & 0 \\ \frac{15}{16} & 0 & \frac{3}{4} & 0 & 0 & 1 & 0 \\ \frac{7}{16} & 0 & \frac{3}{4} & 0 & 0 & 0 & 1 \end{pmatrix}, \; T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{3}{4} & 1 & 0 & 0 & 0 & 0 & 0 \\ -\frac{3}{4} & 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{1}{8} & -\frac{3}{4} & 0 & 1 & 0 & 0 & 0 \\ -\frac{3}{8} & -\frac{3}{4} & 0 & 0 & 1 & 0 & 0 \\ -\frac{3}{8} & 0 & -\frac{3}{4} & 0 & 0 & 1 & 0 \\ \frac{1}{8} & 0 & -\frac{3}{4} & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{5.9}$$

Zur Interpolation ist also die explizite Kenntnis einer Stützpunktbasis nicht mehr vonnöten. Mit Hilfe von T bzw. T^{-1} kann die zugehörige Stützpunktbasis aber leicht gewonnen werden. Durch Umkehrung des oben beschriebenen Vorgangs, also durch Aufsummierung der gewichteten Basisfunktionen, erhält man mittels T die zu den hierarchischen Polynombasen gehörenden höhergradigen Stützpunktbasen. Abbildung 5.7 zeigt hierarchische Basis und Stützpunktbasis für den quadratischen Fall.

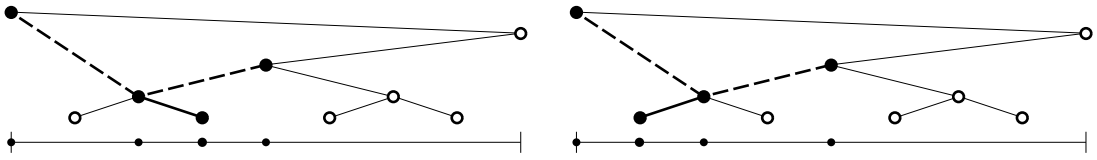


Abbildung 5.8: Lageabhängige Wahl der Interpolationspunkte ($p = 3$)

5.1.3 Eigenschaften der Interpolationsoperatoren und Stützpunktbasen

Die besonderen Eigenschaften und Unterschiede der resultierenden Interpolationsoperatoren und Stützpunktbasen sollen im folgenden eingehender betrachtet werden.

Interpolationsoperatoren

Der zur Berechnung des hierarchischen Überschusses benutzte Interpolationsoperator benötigt $p + 2$ Koeffizienten. Berücksichtigt man auch die zwischen den benötigten Punkten liegenden Nullen im Operator, so ist die Größe des Operators $2^p + 1$. Für jeden Grad gibt es 2^{p-1} Operatoren, die abhängig von der Lage des Punktes x_i zur Überschußberechnung v_i eingesetzt werden müssen. Die unterschiedlichen Operatoren ergeben sich aus dem ebenfalls lageabhängigen Konstruktionsprinzip (Abb. 5.8). Die Operatoren für $p = 1, 2, 3$ haben folgende Gestalt:

$$p = 1: \quad v_{2k+1} = \left[\begin{array}{ccc} -\frac{1}{2} & 1 & -\frac{1}{2} \end{array} \right] \cdot u, \tag{5.10}$$

$$p = 2: \quad \begin{aligned} v_{4k+1} &= \left[\begin{array}{ccccc} -\frac{3}{8} & 1 & -\frac{3}{4} & 0 & \frac{1}{8} \end{array} \right] \cdot u, \\ v_{4k+3} &= \left[\begin{array}{ccccc} \frac{1}{8} & 0 & -\frac{3}{4} & 1 & -\frac{3}{8} \end{array} \right] \cdot u, \end{aligned} \tag{5.11}$$

$$p = 3: \quad \begin{aligned} v_{8k+1} &= \left[\begin{array}{ccccccccc} -\frac{21}{64} & 1 & -\frac{7}{8} & 0 & \frac{7}{32} & 0 & 0 & 0 & -\frac{1}{64} \end{array} \right] \cdot u, \\ v_{8k+3} &= \left[\begin{array}{ccccccccc} \frac{5}{64} & 0 & -\frac{5}{8} & 1 & -\frac{15}{32} & 0 & 0 & 0 & \frac{1}{64} \end{array} \right] \cdot u, \\ v_{8k+5} &= \left[\begin{array}{ccccccccc} \frac{1}{64} & 0 & 0 & 0 & -\frac{15}{32} & 1 & -\frac{5}{8} & 0 & \frac{5}{64} \end{array} \right] \cdot u, \\ v_{8k+7} &= \left[\begin{array}{ccccccccc} -\frac{1}{64} & 1 & 0 & 0 & \frac{7}{32} & 0 & -\frac{7}{8} & 1 & -\frac{21}{64} \end{array} \right] \cdot u. \end{aligned} \tag{5.12}$$

Es sei darauf hingewiesen, daß die Operatoren gespiegelt zueinander liegen. Dies kann bei der Programmierung der Interpolationsvorschrift ausgenutzt werden.

Operatoren höherer Dimension erhält man im linearen Fall durch ein d–maliges Tensorprodukt des eindimensionalen Operators. Wie man leicht sieht, wächst die Größe des Interpolationsoperators vor allem bei höherer Dimension stark an. Dies beeinträchtigt die Geschwindigkeit der Algorithmen erheblich. Die

hierarchsichen Überschüsse einer Darstellung des Interpolanten durch Basisfunktionen p–ten Grades können aber auch aus den Überschüssen der Darstellung durch Funktionen vom Grad $p-1$ berechnet werden. Diese Vorgehensweise wird z.B. von Bungartz in [3] gewählt. Da so auf evtl. vorher berechnete Ergebnisse zurückgegriffen werden kann, werden die Berechnungen wegen der kleinen Operatoren nicht wesentlich gebremst. Für die quadratische hierarchische Basis gilt gemäß (5.8):

$$\begin{aligned} v_i^{(2)} &= u_i - \frac{3}{8}u_{(x_i-h_l)} - \frac{3}{4}u_{(x_i+h_l)} + \frac{1}{8}u_{(x_i+3h_l)} \\ &= u_i - \frac{1}{2}u_{(x_i-h_l)} - \frac{1}{2}u_{(x_i+h_l)} - \frac{1}{4}\left(u_{(x_i+h_l)} - \frac{1}{2}u_{(x_i-h_l)} - \frac{1}{2}u_{(x_i+3h_l)}\right) \\ &= v_i^{(1)} - \frac{1}{4}v_{(x_i+h_l)}^{(1)}. \end{aligned} \tag{5.13}$$

Auch hier ergibt sich also ein Operator, der diesmal aber auf die hierarchischen Überschüsse der Interpolationsfunktionen niedrigeren Grades statt auf die Absolutwerte angewendet wird.

$$v_i^{(2)} = \left[\begin{array}{cc} 1 & -\frac{1}{4} \end{array}\right] \cdot v_i^{(1)}. \tag{5.14}$$

Im Fall $p = 3$ erhält man zwei Operatoren, abhängig von der Lage von x_i:

$$\begin{aligned} v_i^{(3)} &= \left[\begin{array}{ccc} 1 & +\frac{1}{8} \end{array}\right] \cdot v_i^{(2)} = \left[\begin{array}{ccc} 1 & -\frac{3}{8} & +\frac{1}{32} \end{array}\right] \cdot v_i^{(1)}, \\ v_i^{(3)} &= \left[\begin{array}{ccc} 1 & -\frac{1}{8} \end{array}\right] \cdot v_i^{(2)} = \left[\begin{array}{ccc} 1 & -\frac{1}{8} & -\frac{1}{32} \end{array}\right] \cdot v_i^{(1)}. \end{aligned} \tag{5.15}$$

Dabei bezeichnet $v_i^{(p)}$ den Überschuß bezüglich einer hierarchischen Basis vom Grad p.

Dieser Ansatz findet in unseren Arbeiten keine Verwendung, da in unserem Fall hierarchische Überschüsse nicht gespeichert werden, und soll daher nicht weiter verfolgt werden. Von Bungartz ist in [4] diese Vorgehensweise genauer beschrieben.

Stützpunktbasen

Durch die Transformation mittels der Matrix T erhält man aus der vorgegebenen hierarchischen Basis von Grad p eine dazugehörige Stützpunktbasis desselben Grades. Am Beispielbild der quadratischen Basen (Abb. 5.7) kann man bereits einen wichtigen Unterschied der Stützpunktbasen vom Grad $p \geq 2$ im Gegensatz zur stückweise linearen Basisfunktion sehen. Die Stützpunktbasis kommt nun nicht mehr mit einem Basisfunktionstyp aus; man benötigt eine weitere Basisfunktion, um die Basis zu bilden. Sämtliche Basisfunktionen sind aber immer noch stückweise quadratisch und stetig, aber nicht differenzierbar. Ein besonderes Merkmal dieses Ansatzes ist, daß beim Übergang auf höhergradige Polynome die

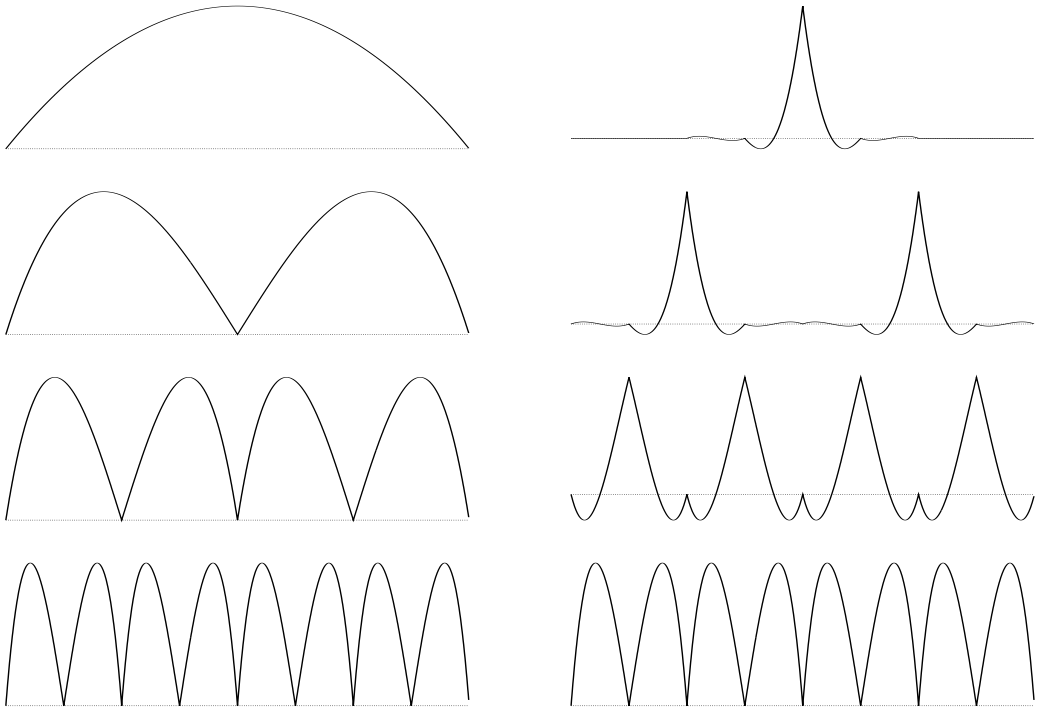


Abbildung 5.9: Kubische hierarchische Basis und Stützpunktbasis

zusätzlichen Freiheitsgrade nicht wie üblich in Glattheitsbedingungen investiert werden, sondern durch die Hinzunahme weiterer Punkte fixiert werden. Im kubischen Fall wird die Situation noch deutlicher. Zur Darstellung der Stützpunktbasis dritten Grades benötigt man bereits vier unterschiedliche Basisfunktionen. Welche Basisfunktion auf einem Punkt aufgesetzt werden muß, hängt von der Position des Punktes ab. Man kann den Basisfunktionen eine Hierarchie zuordnen, wie das bei einer hierarchischen Basis der Fall ist. Abbildung 5.9 zeigt eine solche hierarchische Anordnung für den Fall $p = 3$. Der Begriff der Stützpunktbasis ist trotz dieser Hierarchie gerechtfertigt, da die Größe der Träger der Basisfunktionen wächst, jedoch nach wie vor lokal begrenzt bleibt.

5.1.4 Implementierung

Eine Herleitung des Interpolationsoperators über die Konstruktion der Transformationsmatrix ist nur mit großem Aufwand zu realisieren. Daher stellt sich die Frage, ob es möglich ist, einen einfacheren Weg zu finden, um die zu einer hierarchischen Basis p–ten Grades gehörenden Interpolationsoperatoren zu berechnen. Als Lösung bietet sich die Stützpunktbasis an. Dazu muß sichergestellt sein, daß diese sich auch unter Vermeidung der Konstruktion der Interpolationsmatrix berechnen läßt. Aus einer solchen Stützpunktbasis kann man dann ein einfaches Konstruktionsprinzip zur Bestimmung des Interpolationsoperators gewinnen.

Bei genauer Analyse der Basisfunktionen der Stützpunktbasis stellt man fest, daß diese ebenso wie die Basisfunktionen der hierarchischen Basis durch Polynominterpolation erzeugt werden können. Somit erhält man auf einfache Weise die Funktionsvorschriften für die Stützpunktbasen. Das Konstruktionsprinzip dieser speziellen Stützpunktbasen kann man verallgemeinern, so daß sich die Interpolationsoperatoren direkt als Auswertung dieser Interpolationsfunktionen an den zwischen den Stützstellen liegenden Stellen berechnen lassen. Dies entspricht genau den Anteilen der Absolutwerte, die vom Wert des zu interpolierenden Punktes subtrahiert werden, wie das für den Interpolationsoperator in 3.9 gefordert wurde. Das dazu notwendige Verfahren ist ohne großen Aufwand zu implementieren.

Für die algorithmische Realisierung der Überschußberechnung benötigt man noch die Berechnungsvorschrift für die Wahl des Operators und der Gitterweite h_l aus der Koordinate $i \in 0, \ldots, N-1$ des Punktes x_i, um den richtigen Operator an der richtigen Stelle mit der richtigen Ausdehnung aufsetzen zu können. Dabei ist zu beachten, daß die Ausdehnung des Operators größer oder gleich der Größe des Gitters ist, da man sich sonst mit einem Operator für Funktionen vom Grad $p-1$ behelfen muß. Ein Punkt x_i, der zuerst in dem Level l auftaucht, genügt folgender Bedingung:

$$\frac{i \bmod 2^{l_{max}-l+1}}{2^{l_{max}-l}} = 1. \tag{5.16}$$

Die Zuordnung eines Punktes zu einem Level ist eindeutig. Daher kann man die Berechnung algorithmisch so formulieren:

```
j = i, l = lmax
while not (j mod 2 = 1) and (l > 0)
      j = j div 2
      l = l - 1
end while
```

Auf einem Computer ist die Bestimmung des Levels l effizient durch die Verwendung von Operationen auf Bitebene zu realisieren:

```
j = i; l = lmax;
while !(j & 1) && (l > 0) {
    j >>= 2; l--;
}
```

Zusätzlich bestimmt das bei dieser Rechnung erhaltene j die Wahl für den aufzusetzenden Interpolationsoperator. Alle weiteren Programmstücke zur Anpassung der Algorithmen an beliebige Polynomgrade sind ebenfalls effizient auf Bitebene realisierbar.

Außerdem sind sämtliche Algorithmen ebenso wie in Abschnitt 4.6 optimiert, indem man Ganzzahlarithmetik benutzt. Dies erreicht man, indem man die Koeffizienten des Operators als Brüche, getrennt in Zähler und Nenner, berechnet.

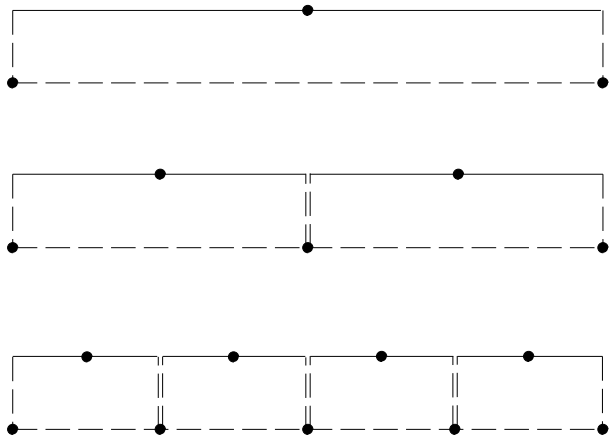


Abbildung 5.10: Stückweise konstante hierarchische Basis

Dann bestimmt man das kleinste gemeinsame Vielfache der Nenner und multipliziert damit die Zähler. Der Interpolationsoperator enthält nur noch ganze Zahlen. Bei der Berechnung des hierarchischen Überschusses ist darauf zu achten, daß auch die Fehlerschranke mit diesem Faktor multipliziert werden muß.

Als weiterer Vorteil ergab sich außer der freien Wahl der Basisfunktion auch, daß die Ränder und Ecken nicht mehr separat behandelt werden mußten. Hier wird automatisch ein Interpolationsstern der Dimension $d-1$ bzw. $d-2$ etc. benutzt.

5.2 Stückweise konstante hierarchische Basisfunktionen

Der Vollständigkeit halber wurde auch untersucht, wie eine stückweise konstante hierarchische Basisfunktion auszusehen hätte. Wieder sollen die Basisfunktionen folgende Bedingungen erfüllen (vgl. 3.5):

$$\begin{aligned} \varphi_i(x_i) &= 1, \\ \varphi_i(x_i \pm h_l) &= 0, \\ supp(\varphi_i) &= [x_i - h_l, x_i + h_l]. \end{aligned} \tag{5.17}$$

Die Basisfunktionen sind auf dem gesamten Träger konstant 1 mit Ausnahme der Randpunkte. Das Ergebnis ist eine konstante hierarchische Basis, wie sie Abbildung 5.10 zeigt.

Der Interpolationsvektor für diese Basis lautet:

$$v_i = \begin{bmatrix} 1 & -1 \end{bmatrix} \cdot u. \tag{5.18}$$

Hier ergibt sich ein zusätzlicher Freiheitsgrad für die Lage des Interpolationsoperators. Diese ist auch beliebig, aber fest zu wählen. Es ist klar, daß es sich

hierbei um nichts anderes handelt als um die Differenz zwischen den Wert an der Stützstelle und dem Wert eines seiner direkten hierarchischen Vorgänger. Im zweidimensionalen Fall ergibt sich ebenso selbstverständlich der Interpolationsstern als das Tensorprodukt dieses Vektors mit sich selbst (vgl. 3.10):

$$v_i = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \cdot u. \tag{5.19}$$

Dieses Vorgehen ähnelt dem Codierungsverfahren der Differenzencodierung (2.2.8). Der interessante Unterschied liegt hier aber wieder in der Hierarchie und Adaptivität.

5.3 Einsatz von Computeralgebra bei der Entwicklung

Zur Erzeugung und Untersuchung von höhergradigen Polynomen zum Zweck der Konstruktion hierarchischer Basen höheren Grades wurden Computeralgebra–Programme eingesetzt. Dies geschah zum einen, um einen besseren Einblick in die Theorie der hierarchischen Polynombasen zu bekommen, zum anderen, um im Rahmen eines größeren Projektes zu studieren, ob und wie gut Computeralgebra–Programme im Bereich des wissenschaftlichen Rechnens (Scientific Computing) eingesetzt werden können. Für diese Untersuchungen wurde das Programmpaket Maple V, das in Anhang A.2 etwas genauer erläutert wird, verwendet.

Zunächst wurde das Programm benutzt, um die Interpolationsfunktionen der Stützpunktbasen symbolisch zu berechnen und damit die Interpolationsoperatoren zu erhalten. Es zeigt sich, daß der ganze Prozeß auf symbolischer Grundlage formuliert werden kann. Der Einsatz von Computeralgebra erfolgt hier nicht zur vollständigen symbolischen Lösung eines mathematischen Problems, was dem eigentlichen Anwendungsgebiet von Computeralgebra–Programmen entspräche. Die symbolische Algebra unterstützt hier den Entwurfsprozeß zur Erstellung von Algorithmen, die viele mathematische Standardprobleme und -berechnungen beinhalten, da eine symbolische Formulierung oft sehr viel einfacher ist als die Darstellung in einer höheren Programmiersprache. So entsteht auf schnelle und einfache Weise ein Prototyp des Algorithmus, der auf Richtigkeit und Effektivität geprüft und eventuell noch geändert werden kann, bevor man ihn in eine höhere Programmiersprache überträgt (*rapid prototyping*). Eine sofortige Programmierung in einer höheren Programmiersprache hätte zur Folge, daß Datenstrukturen und mathematische Berechnungen vorher vollständig ausprogrammiert werden müssen. Stellt sich danach heraus, daß die gewählte Vorgehensweise nicht zum Ziel führt, so war der bisherige Aufwand umsonst. Eine Prototyp–Realisierung läßt auch Freiheiten zu, den Algorithmus durch einfache Veränderung der Rahmenbedingungen genauer zu untersuchen und besser zu verstehen und so eventuell

bereits in einem sehr frühen Entwicklungsstadium Verbesserungen vorzunehmen. Der zusätzliche Schritt, der durch die Übertragung der Algorithmen vom Prototyp in die höhere Programmiersprache entsteht, kann zur Optimierung der bereits verifizierten Algorithmen unter bestimmten Effizienzkriterien benutzt werden. In unserem Fall konnten in der Computeralgebra–Software vorhandene Funktionen wie z.B. Funktionen zur Polynominterpolation von Stützwerten, zur Inversion von Matrizen oder zur Berechnung des kleinsten gemeinsamen Vielfachen genutzt werden, ohne daß man sich zunächst Gedanken über deren Implementation machen mußte. Dabei ergaben sich während der Entwicklung des Prototyps notwendige Änderungen, die bei einer Programmierung in einer höheren Programmiersprache nur unter großem Aufwand zu modifizieren gewesen wären. Außerdem ist der automatische Funktionsplotter eine große Hilfe, da man Zwischen- und Ergebnisse sofort auch graphisch visualisieren kann und sich so sehr viel leichter eine Vorstellung von den Eigenheiten der Methode machen kann.

Die effiziente Nutzung von Computeralgebra–Programmen in der oben beschriebenen Weise ist noch nicht ganz gegeben, da manche Funktionen noch nicht weit genug entwickelt sind. Außerdem spielt die Geschwindigkeit der Software auch in diesem Fall eine nicht unwesentliche Rolle, da es oft von Vorteil wäre, den Prototyp auf umfangreichere Probleme anzuwenden. Dies ist wegen der relativ langen Rechenzeiten des Programms meist noch nicht möglich. Außerdem wäre eine Schnittstelle zu höheren Programmiersprachen wie C eine große Erleichterung bei einer Einbindung oder sukzessiven Übertragung der Algorithmen. Durch eine stärkere Wechselwirkung zwischen Computeralgebra– und Programmiersprachen–Programmteilen könnte der Entwicklungsprozeß weiter beschleunigt werden. Eine Erhöhung der Laufzeiteffizienz der Computeralgebra–Programme könnte sogar dazu führen, daß die Programmteile nicht mehr übertragen werden müßten, sondern direkt eingebunden werden könnten. Dies würde wegen der oft sehr viel einfacheren Programmstruktur der Computeralgebra–Programme auch zu einer besseren Lesbarkeit und einem schnelleren Verständnis von in höheren Programmiersprachen verfaßten Programmen führen.

5.4 Weitere hierarchische Basen

Eine Funktionsklasse, die vor allem in der digitalen Signalverarbeitung immer mehr an Bedeutung gewinnt, ist die Klasse der *Wavelets*. In jüngerer Zeit fanden Wavelets auch immer mehr Verwendung im Bereich der Numerik (z.B. bei Daubechies in [8]). Daher ist es interessant, sich darüber Gedanken zu machen, ob die Vorteile von Wavelets mit den Vorteilen stückweise linearer hierarchischer Basen kombiniert werden können. Wavelets führen im Vergleich zu hierarchischen Basen in der Numerik zu besser konditionierten Problembeschreibungen, bieten jedoch nicht die mit einem kompakten Träger verbundenen Vorteile. Aus diesem Grund wurde die sog. *Pre–Wavelet*–Basisfunktion entwickelt (Abb. 5.11). Sie ist

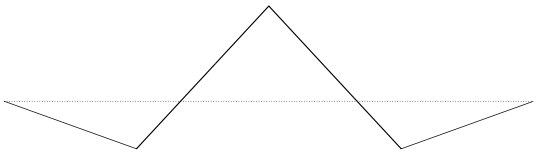


Abbildung 5.11: Pre–Wavelet–Basisfunktion

stückweise linear und auf einem kompakten Träger definiert. Darüber hinaus erfüllt sie einige Bedingungen, die häufig für Wavelets gefordert werden. In einem ersten Schritt wurden die Methoden zur Konstruktion hierarchischer Polynombasen auch auf die einfachste einer solchen Pre–Wavelet–Basisfunktion angewandt. Dabei stellte sich heraus, daß der Interpolationsoperator bei Anwendung dieser Methoden nicht mehr beschränkt bleibt, sondern mit der Leveltiefe wächst. Daher wurde diese Problematik zurückgestellt. Trotzdem ist nicht ausgeschlossen, daß intensivere Forschung auf diesem Gebiet zu interessanten Ergebnissen führt.

Weitere Variationsmöglichkeiten für Basisfunktionen wie Haar–, Hermite– oder Spline–Basen, die eventuell zur Konstruktion hierarchischer Basen herangezogen werden könnten, bieten sich ebenfalls als zukünftige Ansätze an.

5.5 Zusammenfassung, Ergebnisse und Ausblick

Die theoretischen Überlegungen zur Entwicklung von Polynomen höheren Grades geben einen guten Einblick in die Zusammenhänge zwischen hierarchischer Basis, zugehöriger Stützpunktbasis und deren Interpolationsoperatoren. Es wurde ein Konstruktionsprinzip skizziert, das es erlaubt, hierarchische Basen und die dazugehörigen Interpolationsoperatoren und Stützpunktbasen theoretisch bis zu einem beliebigen Grad zu erzeugen und einzusetzen. In der Praxis sind Basisfunktionen von einem Grad höher als 6 bis 8 bei Verwendung von äquidistanten oder wie in unserem Fall auseinanderdriftenden Stützstellen aber nur von untergeordneter Relevanz, da aus der Numerik bekannt ist, daß Polynome mit wachsendem Grad dann immer stärker oszillieren und Probleme wie die Nullstellenbestimmung immer schlechter konditioniert werden. Im Zusammenhang mit der Dünngittermethode treten die positiven Eigenschaften höhergradiger Polynome bereits bei Verwendung von Polynomen niedrigeren Grades auf (s. Bungartz [4]). Somit kann die Qualität einer höhergradigen Polynominterpolation erreicht werden, ohne die oben erwähnten Nachteile in Kauf nehmen zu müssen.

Der zusätzliche Rechenaufwand, den die großen Interpolationsoperatoren verursachen, steigt nicht nur mit höherem Grad sondern auch mit jeder weiteren Dimension erheblich. Es zeigt sich auch, daß sich hierarchische Polynombasen mit wachsendem Grad zunehmend ähneln, so daß hier optisch keine größeren Änderungen zu erwarten sind. Der Einsatz höhergradiger Polynombasen ist nur bei der Anwendung auf Daten sinnvoll, die den sich immer mehr verschärfenden

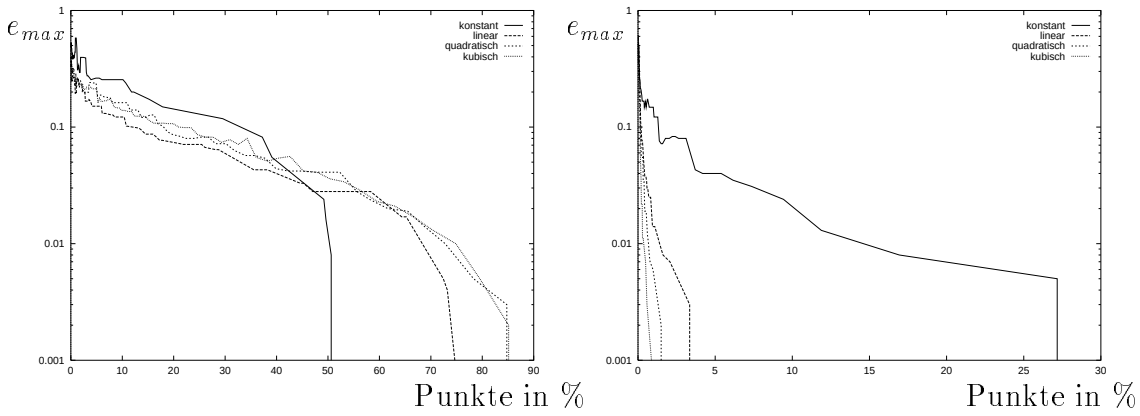


Abbildung 5.12: Maximaler Fehler für Polynombasen verschiedenen Grades

Glattheitsbedingungen genügen.

Die Verwendung von Polynombasen zur Kompression von Bilddaten hat zu keinen besseren Ergebnissen geführt. Dies war auch nicht anders zu erwarten, da bereits bei stückweise linearen Basen die Glattheitsanforderungen nicht ausreichend erfüllt sind. Die Abbildung 5.12 zeigt den maximalen Fehler der auf 129×129 Pixel verkleinerten Testbilder `hoehen` und `hill` für quadratische und kubische Polynombasen sowie für stückweise lineare und konstante Basen. Das Beispiel der Höhendaten wurde gewählt, da man sich hier noch am ehesten eine ausreichende Glattheit der Daten erhoffte. Ein Datensatz, der die Glattheitsbedingungen sicher erfüllt, ist das Grauwertbild `hill` einer Testfunktion (s. Anhang C.1). Wie man deutlich sieht, sind die Höhendaten nicht glatt genug, um sich für eine Interpolation durch höhergradige Polynome zu eignen. Die Testfunktion zeigt aber, daß die Kompressionsrate bei ausreichender Glattheit wie erwartet bei der Verwendung höhergradiger Polynombasen sehr viel besser ausfällt. Weitere Tests haben gezeigt, daß sich für synthetische Bilder wie `schiff` wegen der großen Flächen konstanter Grauwerte die stückweise konstanten Basen am besten eignen.

Kapitel 6

Dünngitterkompression von Bildsequenzen

Die Kompression von Daten mit der Dünngittermethode unter Verwendung von stückweise linearen Basen stützt sich auf lineare Interpolation von Daten. Bei Bildfolgen tritt es häufig auf, daß sich von Bild zu Bild nicht viel ändert und demzufolge relativ viele Punkte linear interpoliert werden können. Daher erscheint die Kompression von Bildfolgen mit der Dünngittermethode vielversprechend.

6.1 Übertragung der Kompressionsalgorithmen

Nachdem die Dünngitteralgorithmen für quadratische Grauwertbilder implementiert worden sind, müssen sie für die Verwendung mit Bildfolgen entsprechend erweitert werden.

6.1.1 Ausnutzung der dreidimensionalen Struktur

Während ein Bild von zweidimensionaler Struktur ist, kann eine Bildfolge als dreidimensional angesehen werden (Abb. 6.1). Dabei ist zu beachten, daß im

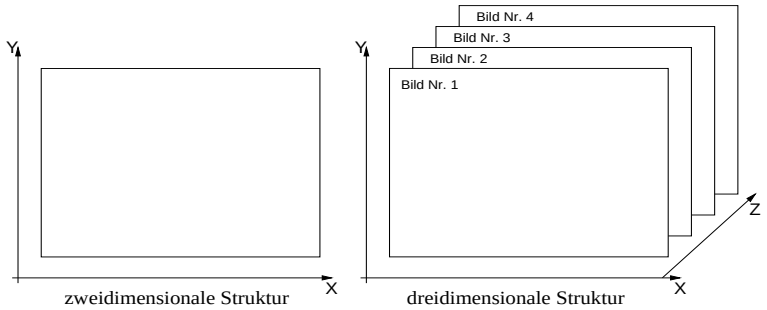


Abbildung 6.1: dreidimensionale Struktur von Bildfolgen

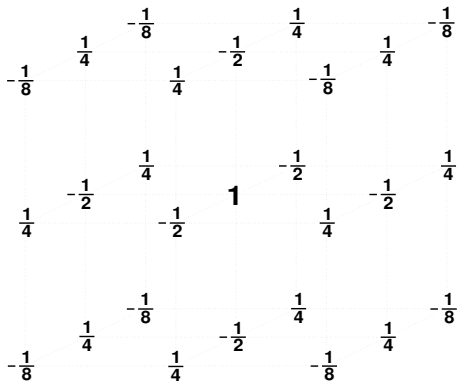


Abbildung 6.2: dreidimensionaler Interpolationsoperator

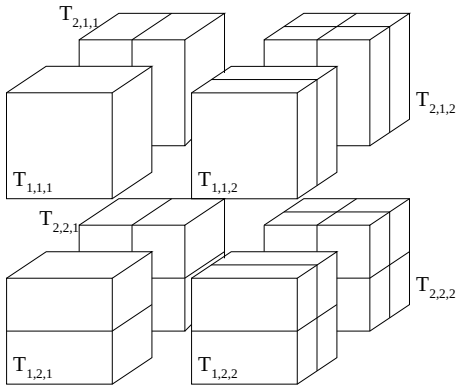


Abbildung 6.3: dreidimensionales Teilraumschema (n=2)

zweidimensionalen der Rand der Daten aus der ersten und letzten Zeile sowie der linken und rechten Spalte des Bildes besteht. Im dreidimensionalen besteht er dagegen aus dem ersten und letzten Bild der Sequenz, sowie den Rändern aller dazwischen liegender Einzelbilder. Die Ränder sind also von zweidimensionaler Struktur, weswegen die in Kapitel 4 entwickelten Algorithmen im dreidimensionalen Fall bei der Randbehandlung verwendet werden können.

Um Abhängigkeiten in aufeinanderfolgenden Bildern für die Interpolation auszunutzen müssen die bisher für zweidimensionale Daten implementierten Algorithmen um eine Dimension erweitert werden. Der demnach erforderliche dreidimensionale Interpolationsoperator ergibt sich wie in Abschnitt 3.1 beschrieben einfach durch ein dreifaches Tensorprodukt des eindimensionalen Operators. Zur Anwendung dieses Operators (Abb. 6.2) auf die Daten muß die in Abschnitt 4.1 entwickelte Durchlaufstrategie so erweitert werden, daß sie alle Komponenten eines jetzt dreidimensionalen Teilraumschemas (Abb. 6.3) erreicht. Es wird wie zuvor im zweidimensionalen eine levelweise Bottom–Up–Durchlaufstrategie verfolgt. Die Level, die zuvor aus den auf den Diagonalen eines quadratischen Teilraumschemas liegenden Teilräumen bestanden, finden sich jetzt auf Diagonalschnitten

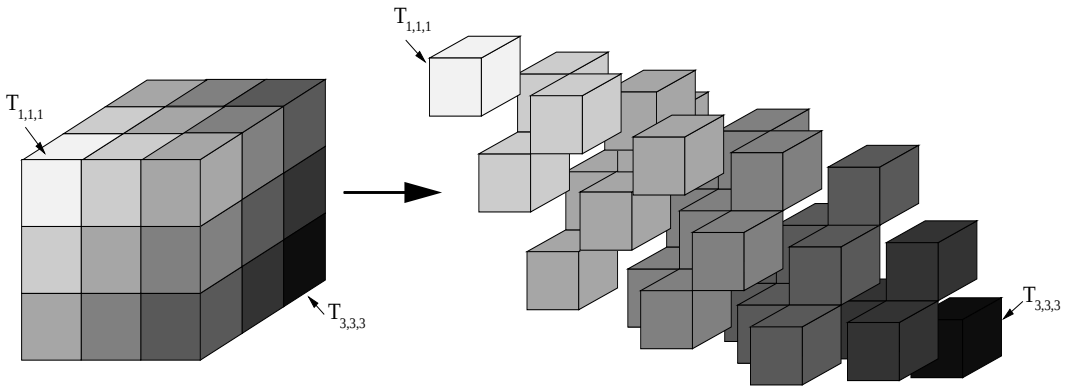


Abbildung 6.4: dreidimensionales Teilraumschema mit Leveln (n=3)

eines Würfels wieder. Abbildung 6.4 zeigt die einzelnen Level eines dreidimensionalen Teilraumschemas für ein Gitter mit $9 \times 9 \times 9$ Gitterpunkten. Für die Anpassung der aus Abschnitt 4.1 bekannten Funktionen `teilraumdurchlauf()`, `leveldurchlauf()` und `gitterdurchlauf()` wird wiederum die Eigenschaft der Teilräume eines Levels genutzt, nach der die Träger dieser Teilräume gleiches Volumen haben.

6.1.2 Farbe

Da Videosequenzen in der Regel farbig sind, ist es nötig, die bisherige Einschränkung auf Grauwertbilder fallenzulassen. Es wird davon ausgegangen, daß die Eingabebilder im RGB–Modell mit 24 bit Farbtiefe vorliegen. Durch separate Behandlung der drei Farbebenen ist es dann ohne größeren Aufwand möglich, Farbbildcodierungen durchzuführen. Für die Kompression erweist es sich allerdings als nachteilig, daß die RGB–Farbkanäle sich in Intensität und Kontrast nur geringfügig voneinander unterscheiden. Werden die Farbkanäle als Grauwertbild dargestellt, so sehen sich diese sehr ähnlich. Bei separater Behandlung der Kanäle wird also dreimal ein etwa gleich aussehendes Grauwertbild komprimiert, um ein Farbbild zu codieren. Betrachtet man dagegen die Komponenten des auch von JPEG und MPEG genutzten Farbmodells YUV als Grauwertbilder, so findet sich im Y–Kanal eine Schwarz–Weiß Version des Farbbildes. Die U– und V–Kanäle sind dagegen eher dunkel und zeigen deutlich weniger Kontraste auf, d.h. sie sind deutlich glatter. Sie lassen sich dementsprechend besser komprimieren als der Y– oder die R–, G– und B–Kanäle. Aufgrund dieses entscheidenden Vorteils wurde bei der Implementierung die Möglichkeit geschaffen, die Eingabedaten in eine YUV–Darstellung zu konvertieren. Es wurde jedoch auf eine Verringerung der Bildauflösung im U– und V–Kanal, wie sie bei der JPEG– und MPEG–Kompression praktiziert wird, verzichtet.

Um weiterhin mit einem Byte pro Pixel und Farbebene auszukommen wird das YCbCr–Farbmodell verwendet, das eine Variante von YUV darstellt. Bei

RGB–Werten im Bereich von 0 bis 255 liegt der Wertebereich von Y ebenfalls zwischen 0 und 255, Cb und Cr liegen zwischen -127 und +127. Diese Wertebereiche lassen sich mit jeweils 8 Bit darstellen. Die beiden Farbmodelle lassen sich nach folgenden Formeln ineinander konvertieren.

$$\begin{aligned} Y &= 0.2989 \cdot R + 0.5866 \cdot G + 0.1145 \cdot B, \\ Cb &= \frac{B - Y}{2 - 2 \cdot 0.1145}, \\ Cr &= \frac{R - Y}{2 - 2 \cdot 0.2989}, \end{aligned} \tag{6.1}$$

$$\begin{aligned} R &= Cr \cdot (2 - 2 \cdot 0.2989) + Y, \\ B &= Cb \cdot (2 - 2 \cdot 0.1145) + Y, \\ G &= \frac{Y - 0.1145 \cdot B - 0.2989 \cdot R}{0.5866}. \end{aligned} \tag{6.2}$$

Eine verlustfreie Konvertierung von RGB nach YCbCr und zurück ist bei Verwendung von 8–Bit–Integerwerten nicht möglich, da der zulässige Wertebereich bei YCbCr gegenüber RGB deutlich eingeschränkt ist. Im RGB–Modell ist jede Kombination aus Werten des Intervalls $[0, 255]$ ein gültiger Farbwert. Setzt man hingegen im YCbCr Modell beispielsweise $Y = 255$ und $Cb = Cr = 127$, was Werten innerhalb des möglichen Wertebereichs entspricht, so ergibt sich nach obiger Formel eine Farbe in RGB–Darstellung von $R = 433$, $G = 120$ und $B = 480$, die jedoch außerhalb des darstellbaren Bereichs liegt. Der Verlust an Information, der bei der Konvertierung auftritt, entspricht dem einer Quantisierung und ist gewöhnlich nicht wahrnehmbar.

Aus den Formeln zur Konvertierung von YCbCr nach RGB wird deutlich, daß die Grünkomponente auf den zuvor berechneten Rot– und Blaukomponenten aufbaut. Fehler, die hier zugelassen wurden, können sich bei der Berechnung des Grünanteils aufschaukeln, was einen gewissen Grünstich der Bilder nach Verlustbehafteter Kompression und Dekompression erklärt.

6.1.3 Beliebige Formate

Die letzte aufzuhebende Einschränkung betrifft das Bildformat. Bei Fernsehbildern ist das Verhältnis von Breite zur Höhe des Bildes 4 zu 3. Das HDTV Format strebt ein Seitenverhältnis von 16:9 an. Im Kino ist die Breite des Bildes ebenfalls deutlich größer als seine Höhe. Quadratische Bildsequenzen treten also in der Praxis nirgendwo auf.

Um nun eine Bildfolge von beliebigem Format und beliebiger Länge mit der Dünngittermethode komprimieren zu können, ist es das einfachste, die Bilder in den nächstgrößeren Würfel mit $(2^n + 1) \times (2^n + 1) \times (2^n + 1)$ Gitterpunkten einzubetten. Dabei ist darauf zu achten, daß die Ausdehnung des Würfels nicht

zu groß im Vergleich zu Länge, Höhe oder Breite des Ausgangsdatensatzes wird. Werden z.B. die Bilder eines 90 minütigen Films in PAL–Auflösung in einen Würfel eingebettet, so führt dies zu einer Kantenlänge von $2^{18} + 1 = 262145$ Punkten in jeder Koordinatenrichtung. Bei einer Bildauflösung von 768×576 Punkten ist dies jedoch nicht besonders sinnvoll, ganz abgesehen davon, daß ein unkomprimierter 90 minütiger Film ca. 166.9 GB Speicherplatz benötigen würde. Bei einem derartigen Mißverhältnis zwischen Länge, Höhe und Breite des auszudünnenden Datensatzes wird es nötig, die Ausgangsdaten in Blöcke sinvollerer Größe zu unterteilen. Am besten wäre es, wenn die Ausdehnungen eines auszudünnenden Blockes von gleicher Größenordnung sind. Stehen dagegen nur wenig Bilder in hoher Auflösung zur Verfügung, sollten die Daten in Blöcke unterteilt werden, deren X– und Y–Größe die Anzahl der Bilder nicht übersteigt.

Auf einem geeigneten, umschreibenden Würfel können die Dünngitteralgorithmen problemlos eingesetzt werden. Es ist klar, daß, obwohl die Algorithmen auf dem ganzen Würfel arbeiten, nur die hierarchischen Überschüsse des tatsächlichen Datensatzes berechnet werden. So ist es möglich, die Einbettung ohne zusätzlichen Speicherbedarf oder Rechenaufwand zu implementieren. Punkten, die außerhalb des tatsächlichen Datensatzes liegen, aber für die Berechnung des Interpolationsoperators an einem Punkt innerhalb des Datensatzes benötigt werden, ordnet man einen einheitlichen, konstanten Wert zu. Dadurch entstehen zwar an den Rändern des Datensatzes relativ scharfe Kanten, diese bewirken aufgrund der Verteilung der Faktoren im Interpolationsoperator jedoch nur selten eine erhöhte Anzahl von Punkten im ausgedünnten Gitter. Betrachtet man die häufigsten Fälle (Abb. 6.5), bei denen für die Berechnung des Operators Punkte außerhalb des Datensatzes herangezogen werden, so stellt man fest, daß in den Fällen a) und b) die Summe der außerhalb liegenden und deswegen gleichgewichteten Faktoren gleich Null ist. Im Fall c) der deutlich seltener auftritt als die anderen beiden Fälle, liegt das Gewicht der äußeren Punkte bei $-\frac{1}{8}$. Abgesehen von den, in Abbildung 6.5 dargestellten Fällen gibt es noch weitere Situationen, in denen, bei der Berechnung des Interpolationsoperators, nicht alle Punkte innerhalb des Datensatzes liegen. Obwohl in diesen Fällen die Summe der außerhalb liegenden Koeffizienten nicht Null ist, führt das Auftreten dieser Situationen nicht zu signifikant höheren Punktanzahlen im ausgedünnten Gitter, da diese nur wenige, hierarchisch sehr hoch liegende Punkte betreffen.

Es treten durch das Einbeschreiben in einen Würfel also hauptsächlich an den Ecken des Datensatzes vermehrt Punkte im ausgedünnten Gitter auf.

Bei der Einbettung in einen größeren Würfel gibt es im wesentlichen zwei Varianten. Zum einen kann der Originaldatensatz in einer Ecke des Würfels plaziert werden, zum anderen kann er ins Zentrum des Würfels gerückt werden (Abb. 6.6). Diese beiden Einbettungsformen unterschieden sich in der Randbehandlung. Sitzt der Originaldatensatz im Zentrum des Würfels, so fällt die Randbehandlung meist ganz weg. Beim Ausdünnen des Würfelinneren wird bereits für alle Punkte des Datensatzes der hierarchische Überschuß berechnet, es sei denn Länge, Breite

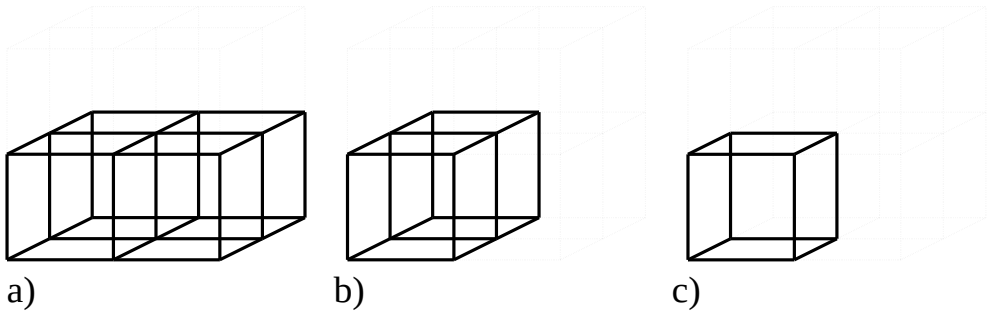


Abbildung 6.5: Randfälle beim Aufsetzen des Interpolationsoperators

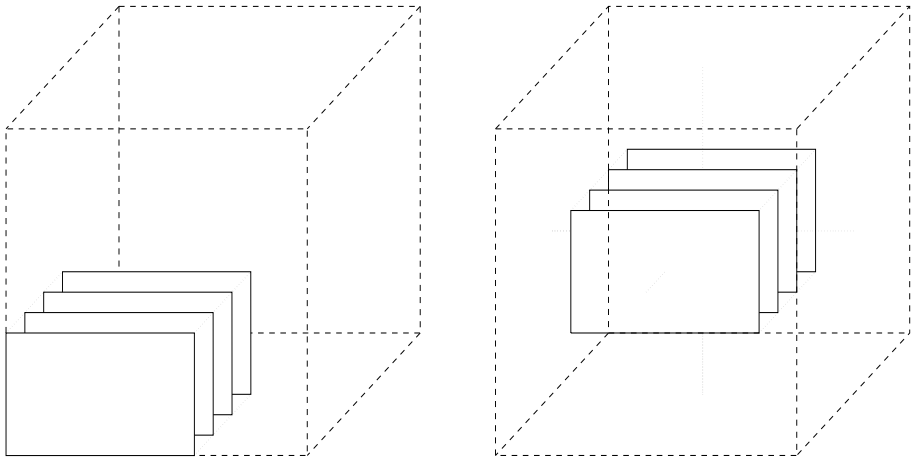


Abbildung 6.6: Einbettung in einen umschreibenden Würfel

oder Höhe des Datensatzes entspricht gerade der Ausdehnung des Würfels. Dann muß der verbleibende Rand separat ausgedünnt werden. Der Nachteil dieser Variante ist, daß dadurch an allen Seiten des Datensatzes die zuvor erwähnten scharfe Kanten auftreten, wodurch die Anzahl der Punkte im ausgedünnten Gitter in der Regel geringfügig höher ist. Von Vorteil ist jedoch, daß keine Randbehandlung nötig ist, was die Implementierung vereinfacht.

Liegt der Originaldatensatz in einer Ecke des Würfels, so müssen die drei Seiten die mit den Seitenflächen des Würfels zusammenfallen separat ausgedünnt werden. Um die Auswirkungen einer völlig separaten Behandlung der Ränder des Datensatzes untersuchen zu können, wurde an dieser Stelle bei der Implementierung auch die im Inneren des Würfels liegenden Randflächen separat behandelt.

Durch die Erweiterung der in Kapitel 4 entwickelten Algorithmen auf die Bearbeitung dreidimensionaler Strukturen, die separate Kompression der drei Farbkanäle und die Einbettung in einen umschreibenden Würfel ist eine Kompression von beliebigen Bildsequenzen mit der Dünngittermethode möglich.

6.2 Echtzeitcodierung

Als typische Anwendung für Echtzeitcodierung soll eine Videokonferenz dienen. Hier müssen Bildfolgen in Echtzeit aufgenommen, komprimiert, übertragen, dekodiert und schließlich angezeigt werden. Eine zu große Verzögerung zwischen Aufnahme und Anzeigen eines Bildes würde zu störenden Gesprächspausen führen und ist daher nicht tolerierbar. Selbst eine Verzögerung von einer Sekunde wird in einem Dialog bereits als unangenehm empfunden. Demnach stehen bei geeigneter Pufferung höchstens 25 bis 30 Einzelbilder zur Verfügung, die von dem auf die dritte Dimension erweiterten Algorithmus durchlaufen werden können. Dabei können die Daten jedoch nicht mehr in ihrer zeitlichen Reihenfolge übertragen werden, da für die Darstellung des ersten Bildes bereits Teile anderer, folgender Bilder decodiert werden müssen. Dies macht die Verwaltung der Daten und die Behandlung eventueller Übertragungsfehler erheblich komplizierter.

Es besteht natürlich die Möglichkeit, die Bilder einzeln, unabhängig voneinander zu komprimieren und zu übertragen, dabei können aber keine Abhängigkeiten zwischen aufeinanderfolgenden Bildern ausgenutzt werden. Auf diese will man in der Regel jedoch nicht verzichten, da sich bei ihrer Ausnutzung deutlich höhere Kompressionsraten erzielen lassen. So bietet es sich an, Differenzencodierung zu verwenden. Dabei wird nicht ein Bild, sondern die Differenz zweier aufeinanderfolgender Bilder komprimiert und übertragen. Das anzuzeigende Bild wird rekonstruiert, indem die dekomprimierten Daten zu dem zuletzt angezeigten Bild hinzuaddiert werden. Abbildung 6.7 zeigt den schematischen Ablauf auf den Seiten des Komprimierers und Dekomprimierers.

Bei einer Videokonferenz ändert sich von einem Bild zum anderen typischerweise nicht viel, also ist die Differenz zwischen zwei Bildern an den meisten Stellen Null und auch sonst sehr glatt. Sie ist also bestens für eine Kompression nach der Dünngittermethode geeignet. Ein Problem stellt der Umstand dar, daß der Wertebereich des Differenzbildes doppelt so groß sein kann wie der ursprüngliche Wertebereich. Um aber nicht mehr als 8 Bit pro Bildpunkt und Farbebene zu benötigen werden die Werte des Differenzbildes halbiert, und vor der abschließenden Addition wieder verdoppelt. Der Verlust, der durch diese Quantisierung entsteht, ist zu gering, um wahrgenommen werden zu können. Da die Dünngitterkompression, auch abgesehen von dieser Quantisierung, verlustbehaftet ist, kann sich ein Fehler bei fortschreitender Differenzencodierung aufschaukeln, was ein weiteres Problem darstellt. Der Grund für die Aufschaukelung ist, daß das zuletzt angezeigte, bereits mit einem Fehler behaftete Bild, als Referenzbild für die Addition des nächsten Differenzbildes dient.

Das Anwachsen von Fehlern kann verhindert werden, indem man auf der Seite des Komprimierers den Mechanismus des Dekomprimierers nachbildet und die so erzeugte Interpolation der komprimierten Daten als Referenzbild für die Berechnung des nächsten Differenzbildes nutzt. Abbildung 6.8 zeigt das entsprechend geänderte Ablaufschema.

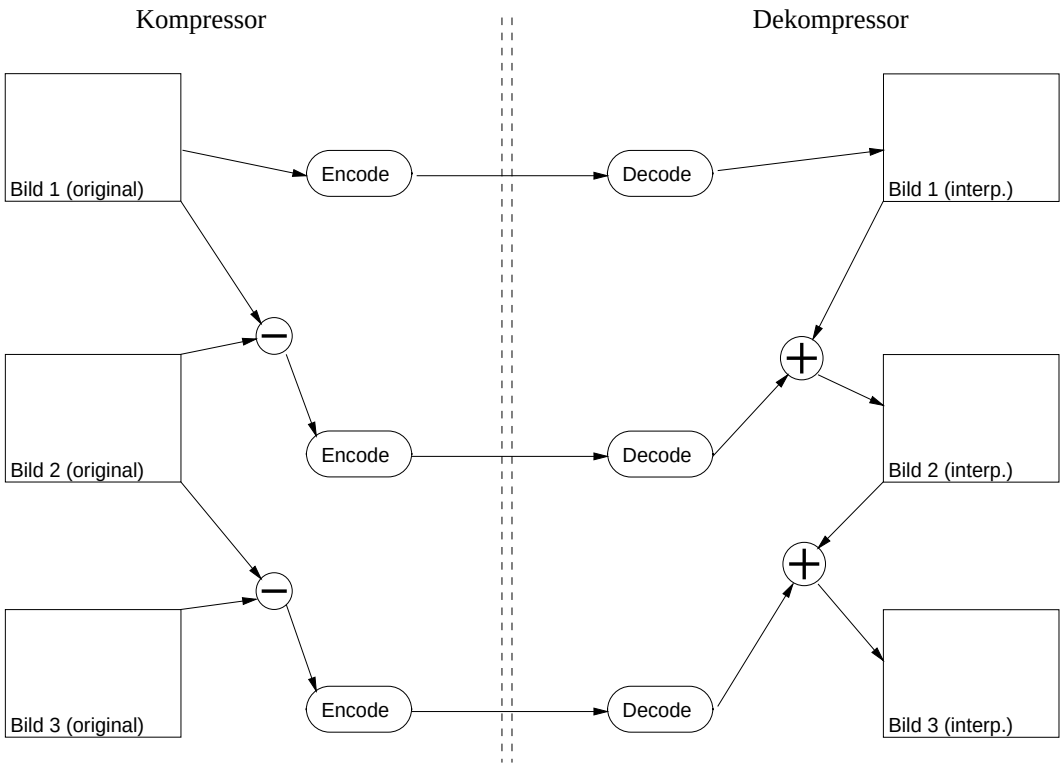


Abbildung 6.7: Kompressor und Dekompressor bei Differenzencodierung

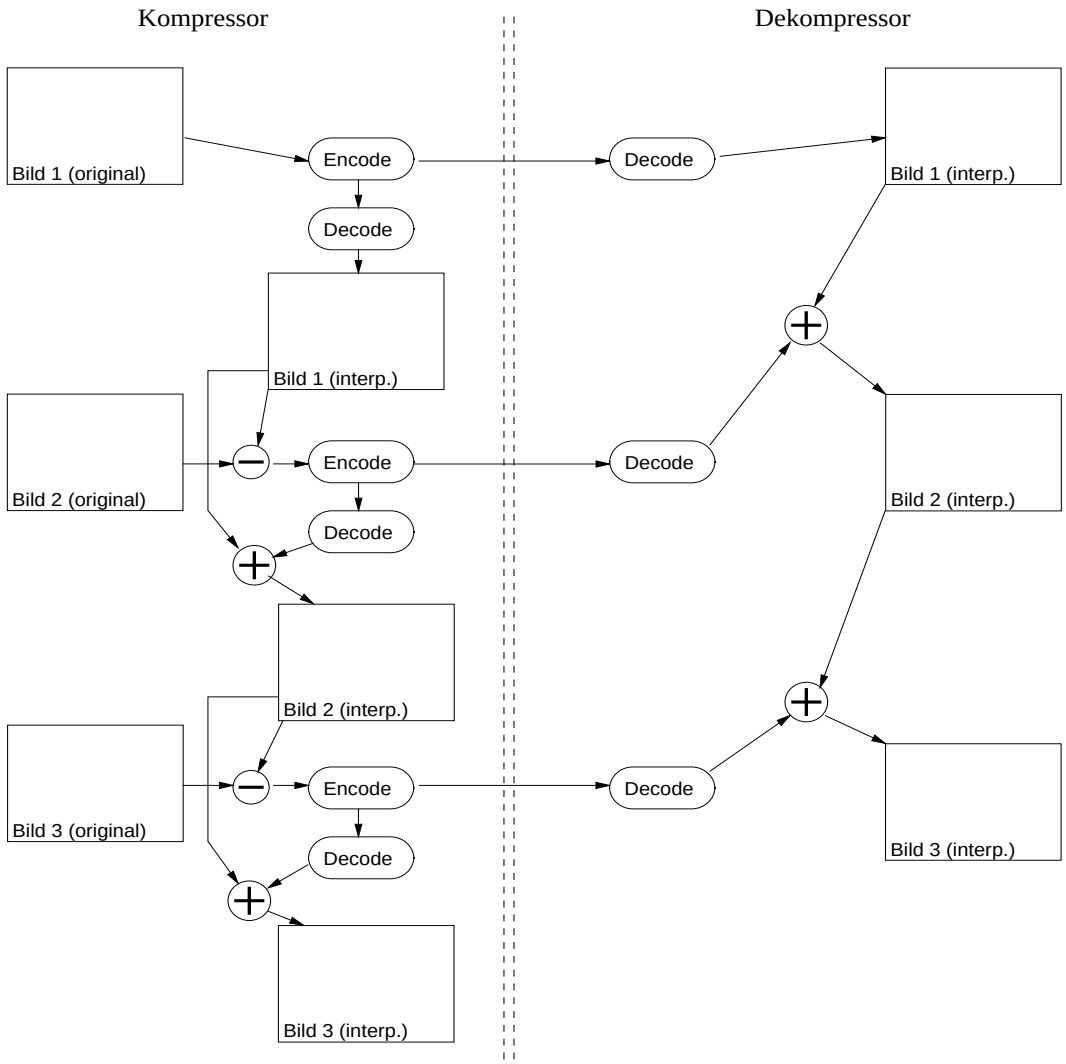


Abbildung 6.8: Kompensation der Fehleraufschaukelung

6.3 Tests, Ergebnisse und Bewertung

Für die Durchführung von Tests wurden, ähnlich wie in Kapitel 4, verschiedene, jeweils für eine bestimmte Klasse von Bildsequenzen repräsentative Beispiele ausgewählt. Als Vertreter für Videokonferenzen dient eine Sequenz aus einer Tagesschau–Sendung (`tagesschau`), in der der Sprecher, ohne sich viel zu bewegen, vor statischem Hintergrund eine Meldung vorliest. Die Aufnahme eines Tennisspiels (`tennis`) zeigt sich schnell bewegende Spieler bei relativ unbewegter Umgebung. Der Ausschnitt einer Szene aus dem Film „Star Wars – Die Rückkehr der Jedi–Ritter“ (`star_wars`) zeigt schnelle Bewegung vor sich stark änderndem Hintergrund sowie schnelle Schnittfolge. Diese Eigenschaften sind für die hier beschriebene Kompressionsmethode eher ungünstig. Die ersten beiden Beispiele stammen von einer PAL–Videoaufnahme im S–VHS–Format, das zuletzt genannte von einer VHS–Aufnahme. Die Qualität der Bilder ist also nicht optimal, was sich ebenfalls eher negativ auf die Kompression auswirkt. Alle von Video überspielten Beispielsequenzen liegen in unterschiedlichen Bildauflösungen vor, einmal in der vollen PAL–Bildauflösung von 768×576 Punkten, zum anderen in Auflösungen von 384×288 Pixeln, $\frac{1}{4}$ der ursprünglichen Daten, und 192×144 Bildpunkten, $\frac{1}{8}$ der Daten. Als letztes Beispiel dient eine Szene aus einem synthetisch erzeugten Film (`dünne_gitter`), die im Gegensatz zu den anderen Beispielen nicht verrauscht ist, große glatte Flächen hat und sich deswegen deutlich besser komprimieren läßt. Die Bildauflösung dieses Beispiels beträgt 360×270 Pixel. Für diese Sequenz wurden nur 10 Bilder pro Sekunde Laufzeit erzeugt, anstelle der 25 Bilder pro Sekunde im Fall der von PAL–Videoaufnahmen stammenden Beispielen. In Anhang C.2 sind einige Bilder aus den einzelnen Sequenzen abgebildet.

In den Testläufen wurden verschiedene Parameter bzw. Strategien des Kompressionsalgorithmus variiert, um die unter dem Gesichtspunkt der Effizienz der Kompression günstigste Methode zu finden. Zu den verschiedenen Variationsmöglichkeiten gehören beispielsweise Kompensationsfaktor und Farbmodell. Für jede einzelne sowie bestimmte interessante Kombinationen dieser Strategien wurde die Fehlerschranke sukzessive erhöht und die in Abschnitt 4.4 eingeführten Fehlermaße ermittelt. Diese wurden, wenn nötig, auf die dritte Dimension erweitert und an die Behandlung mehrerer Farbkanäle angepaßt. Aufgrund des Rechenzeit- und Speicherplatzbedarfs wurden die Testläufe zunächst nur mit den Beispielen mit der jeweils kleinsten Bildauflösung durchgeführt, da es so möglich war, auch eine größere Anzahl von Bildern bei der Kompression zu berücksichtigen. Für `tagesschau`, `tennis`, und `star_wars` wurden jeweils 100 Bilder, für `dünne_gitter` aufgrund der höheren Bildauflösung nur 50 Bilder bei der Kompression berücksichtigt.

Kompensationsfaktor

Dem Vorgehen bei den Testreihen für Bilder im 4. Kapitel folgend sollte als erstes der günstigste Kompensationsfaktor κ für die verschiedenen Beispielsequenzen ermittelt werden. Hierbei ergaben sich die gleichen Probleme, die auch bei der Bestimmung von κ im zweidimensionalen Fall in Abschnitt 4.7 aufgetreten sind. Zudem hat sich gezeigt, daß der Unterschied, der sich durch die Verwendung verschiedener Kompensationsfaktoren ergibt, deutlich geringer ist als im zweidimensionalen Fall. Dadurch wird eine Entscheidung für ein bestimmtes κ aufgrund optischer Vergleiche praktisch unmöglich. Da dieser Vergleich jedoch der entscheidende Qualitätsmaßstab ist, wird für die weitere Durchführung der Tests als Kompromiß ein Kompensationsfaktor von $\kappa = 0.26$ gewählt. Dieser Wert entspricht dem Inversen des Aufschauklungsfaktors, der sich analog zu dem in Abschnitt 4.3 beschriebenen Vorgehen für den dreidimensionalen Fall zu $1+\sqrt[3]{2}+\sqrt[3]{4}$ berechnet. Alle weiteren Tests wurden also jeweils mit einem κ von 0.26 und unkompensierter Fehlerakkumulation durchgeführt.

Einbettung in einen umschreibenden Würfel

Anschließend wurden die Auswirkungen, die sich durch die beiden verschiedenen Einbettungsarten in einen umschreibenden Würfel ergeben, untersucht. Der Unterschied zwischen der Plazierung des Datensatzes in einer Ecke des Würfels mit separater Randbehandlung und dem zentrierten Datensatz ist marginal. Weder durch optischen Vergleich noch durch Vergleich der Fehler läßt sich die eine oder die andere Methode als günstiger klassifizieren. Für die Kompressionsrate ist der Unterschied nicht signifikant.

Farbmodell

Die Verwendung des YCbCr–Farbmodells bei der Kompression führt selbst bei einer Fehlerschranke von $\varepsilon = 0$, d.h. der Ausgangsdatensatz soll verlustfrei rekonstruiert werden, zu geringen Fehlern und somit zu erhöhten Werten in allen Fehlermaßen. Im Fall der Beispielsequenz `tagesschau` steigt der maximale Fehler aller Farbkanäle bei $\varepsilon = 0$ durch Verwendung des YCbCr–Farbmodells von 0 auf 3. Der Grund hierfür liegt, wie in Abschnitt 6.1.2 beschrieben, in dem Umstand, daß die Konvertierung vom RGB–Farbmodell nach YCbCr und anschließend wieder zurück nach RGB einer Quantisierung entspricht. Eine Eigenschaft des YCbCr–Modells ist jedoch, daß der Quantisierungsverlust optisch nicht wahrnehmbar ist. Er beeinflußt aber dennoch die Ergebnisse der verschieden Fehlermaße so stark, daß diese, obwohl praktisch kein Qualitätsverlust festgestellt werden kann, hohe Werte liefern. Daher haben in diesem Fall Vergleiche von Fehlerkurven bei Kompression im RGB– und YCbCr–Farbmodell wenig Sinn. Eine optische Beurteilung der Bildqualität zeigt, daß sich die nach YCbCr konvertierten Bilder besser komprimieren lassen. Die Qualität der Sequenz `tagesschau`

erscheint bei gleicher Kompressionsrate im YCbCr–Farbmodell deutlich besser als im RGB–Modell, der bereits in Abschnitt 4.7 erwähnte „Gobelin–Effekt“ ist deutlich schwächer ausgeprägt. Der in Abschnitt 6.1.2 erwähnte Grünstich wird ab Kompressionsraten von etwa 20% und darunter als störend empfunden. Da sich der Grünstich jedoch auf das gesamte Bild auswirkt, ist es möglich, ihn durch einfache Nachbearbeitung und Farbkorrektur zu kompensieren.

Unterteilung in Blöcke

Bei allen bisherigen Testläufen wurde bei der Einbettung in einen umschreibenden Würfel nicht darauf geachtet, daß Länge, Breite und Höhe des auszudünnenden Datensatzes von gleicher Größenordnung sind. Dies soll bei den folgenden Testläufen berücksichtigt werden. Bei der Implementierung der Aufteilung des Datensatzes in Blöcke wurde die Blockgröße auf das Minimum von Bildauflösung in X– und Y–Richtung und der Anzahl der Bilder festgelegt. Der Testdatensatz, der aus 100 Bildern mit 192×144 Bildpunkte besteht, wird also in Würfel der Größe $100 \times 100 \times 100$ Pixel unterteilt. Bei der separaten Kompression dieser Würfel werden sie jeweils in einen größeren Würfel mit der Kantenlänge 129 Pixel eingebettet. Da, wie in Abschnitt 6.1.3 erklärt wurde, im Bereich der Ecken der Würfel beim Ausdünnen mehr Punkte behalten werden müssen, ist die Anzahl der Punkte im ausgedünnten Gitter bei gleichem Fehler höher als bei einer Kompression desselben Datensatzes ohne Unterteilung in Blöcke. Dies zeigt sich in allen Fehlermaßen, nach denen die Blockbildung keinen Vorteil für die Kompression darstellt. Bei einem optischen Vergleich zeigt sich jedoch, daß bei gleicher Kompressionsrate die Interpolation des in Blöcke unterteilten Datensatzes von besserer Qualität ist als die Rekonstruktion des nicht unterteilten Datensatzes. Der Grund hierfür ist, daß der als ausgesprochen störend empfundene Gobelin–Effekt auf die gebildeten Blöcke beschränkt bleibt. Es liegt also nahe, den Datensatz in noch kleinere Blöcke zu unterteilen, um diesen Effekt weiter zu beschränken. Durch dieses Vorgehen wird jedoch die zeitliche Abhängigkeit aufeinanderfolgender Bilder immer weniger ausgenutzt und so die Kompression immer weniger effektiv. Eine günstige Unterteilung stellt demnach einen Kompromiß zwischen Kompressionseffektivität und der Ausdehnung des Gobelin–Effekts dar. Da auch beim JPEG– und MPEG–Verfahren die zu komprimierenden Daten, wenn auch aus anderen Gründen, in Blöcke unterteilt werden, sind weitere Tests auf diesem Gebiet ausgesprochen vielversprechend.

Kompression von Bilddifferenzen

Die zum Zweck der Echtzeitverarbeitung implementierte separate Kompression von Differenzen aufeinanderfolgender Bilder wurde ebenfalls auf die Testdatensätze angewandt. Dabei wurde die in Abbildung 6.8 dargestellte Strategie zur Verhinderung der zeitlichen Aufschaukelung von Fehlern verwendet. Da sich hier

die Fehler, die bei der Interpolation zugelassen werden auf die Bildung der zu komprimierenden Differenzbilder auswirken, wächst von Bild zu Bild das Fehlerrauschen in der Bilddifferenz an. Dies führt zu einer immer schlechteren Komprimierbarkeit der Bilddifferenzen. So zeigt sich, daß die separate Kompression von Bilddifferenzen zu schlechteren Kompressionsraten führt als das Ausdünnen des gesamten dreidimensionalen Datensatzes.

Die Vorteile dieser Methode beschränken sich also auf die in Abschnitt 6.2 erwähnten Aspekte bei der Echtzeitverarbeitung. Dies wird auch durch die Tatsache deutlich, daß der Vorgang des Komprimierens und Dekomprimierens für den Testdatensatz `tagesschau` ohne Differenzenbildung ca. 150 Sekunden, mit Differenzenbildung jedoch nur etwa 60 Sekunden dauert. Der Grund für diesen Unterschied liegt darin begründet, daß beim zweidimensionalen Ausdünnen der einzelnen Bilder die vorhandenen Cache–Mechanismen wesentlich besser ausgenutzt werden können als beim dreidimensionalen hierarchischen Durchlauf. Bei diesem wird häufig auf Punkte zugegriffen, die im Speicher sehr weit voneinander entfernt sind. Betrachtet man beispielsweise eine Bildfolge von $N = 2^n + 1$ Bildern mit jeweils $N \times N$ Bildpunkten, so sind für das Aufsetzen des Interpolationsoperators auf dem Mittelpunkt des Gitters neben anderen auch der erste und letzte Punkt des gesamten linearisiert gespeicherten Datensatzes nötig. Diese sind im Speicher N^3 Bildpunkte voneinander entfernt. Bei zweidimensionalen Durchläufen durch die Einzelbilder der Sequenz ist der Abstand höchstens N^2 Bildpunkte. Es ist also klar, daß bei einem dreidimensionalen Durchlauf durch größere Datensätze ständig neue Speicherbereiche in den Cache geladen werden müssen. Dies führt gegenüber einem sequentiellen, zweidimensionalen Durchlauf durch die einzelnen Bilder des Datensatzes zu einem erheblichen Geschwindigkeitsverlust. Dieser könnte verhindert werden, wenn mehr Cache–Speicher zur Verfügung stehen würde oder der hierarchische Durchlauf bei der Auswahl der Speicherbereiche, die in den Cache geladen werden, berücksichtigt würde.

Erzielte Kompressionsraten

Aus den Testläufen ergibt sich, daß sich die besten Kompressionsraten bei geringem Qualitätsverlust durch Kombination von Blockbildung und YCbCr–Farbmodells erreichen lassen. Die Sequenz `tagesschau` läßt sich in ihrer kleinsten Auflösung bei Kompression von 100 Bildern ohne störende Qualitätsverluste auf 10% der Daten reduzieren. Für die höheren Auflösungen sind die Kompressionsraten geringer, da hier wegen des größeren Platzbedarfs weniger Bilder für die Kompression berücksichtigt werden können. Bei der vollen PAL–Bildauflösung von 768×576 Pixeln konnten nur noch 25 Bilder in den Hauptspeicher des Rechners geladen werden. Die Kompressionsrate, ab der hierbei Verluste störend zu Tage treten, beläuft sich auf etwa 20%. Das Testbeispiel `tennis` liefert vergleichbare Ergebnisse und ist nur geringfügig schlechter komprimierbar, da hier weniger große glatte Flächen auftreten als im Fall der Sequenz `tagesschau`. Der

Ausschnitt `star_wars`, bei dem innerhalb der berücksichtigten 100 Bilder zwei Schnitte vorkommen, also zweimal völlig verschiedene Bilder aneinandergrenzen, läßt sich erwartungsgemäß nicht so stark komprimieren wie `tennis` oder `tagesschau`. Aufgrund der schnellen Bewegungen und ständig wechselnden Szenen fallen jedoch Fehler beim Abspielen der Sequenz `star_wars` weniger stark auf. Die Kompressionsrate, ab der Fehler sichtbar auftreten liegt etwa bei 15% bzw. bei den höheren Bildauflösungen aus denselben Gründen wie bei den anderen Beispielen entsprechend höher. Ebenso wie bei den Untersuchungen über Bildkompression erweist sich das synthetisch generierte Beispiel `dünne_gitter` als am besten komprimierbar. Verlustfrei kann der Datensatz bei Verwendung des RGB–Farbmodells auf 10%, im YCbCr–Modell auf 7% komprimiert werden. Setzt man noch eine Unterteilung in Würfel mit einer Kantenlänge von 31 Pixeln ein, so ist eine weitere Kompression auf 1.7% der Daten möglich, bevor deutliche Fehler auftreten.

Echtzeitkompression

Abschließende Tests untersuchen die Kompression bei Echtzeitverarbeitung, wie sie im Rahmen einer Videokonferenz erforderlich wird. Hierzu werden Bilder, die von einer an den Rechner angeschlossenen Kamera stammen, als Eingabedaten verwendet. Die Kamera genügt dem NTSC–Standard, d.h. sie liefert 30 Bilder pro Sekunde bei einer Auflösung von 640×480 Pixeln. Da auch kommerzielle Videokonferenzprogramme nicht in der Lage sind, Bilder dieser Größe bei einer Geschwindigkeit von 30 Bildern pro Sekunde zu übertragen, werden ebenso wie bei professionellen Produkten auch hier Bildrate und Auflösung reduziert. Bilder, die von der Kamera aufgenommen werden, während der Kompressionsvorgang noch nicht beendet ist, können bei Echtzeitverarbeitung nicht berücksichtigt werden. So ergibt sich die Bildrate direkt aus der Geschwindigkeit der Kompression, die wiederum direkt von der Bildauflösung abhängt. Bei 320×240 Pixeln liegt die erreichte Bildrate auf den verwendeten Rechner des Typs Indy (Anhang A.1) bei weniger als einem Bild pro Sekunde. Ab einer Anzahl von 212×160 Pixeln pro Bild ist die Bildrate in etwa der gleichen Größenordnung, die auch von kommerziellen Programmen erreicht wird und die für Videokonferenzen gerade erträglich ist.

Nimmt man weiter die für eine Videokonferenz erträglichen Qualitätsverluste in Kauf, d.h. Gesichter müssen erkennbar bleiben, so liegt die Kompression der einzelnen Bilder bei etwa 30%. Verwendet man das YCbCr–Farbmodell, so läßt sich die Kompressionsrate auf bis zu 20% senken. Die stärkste Datenreduktion läßt sich durch eine zusätzliche Verwendung der Differenzenbildung erzielen. Dabei fällt ein charakteristisches Verhalten auf. Die Kompressionsrate des ersten Bildes, für das noch keine Differenz gebildet werden kann, liegt wie zuvor bei etwa 25%. Die Interpolation dieses Bildes ist der Ausgangspunkt für die Bildung der nun folgenden Differenzen und wird im weiteren als Referenzbild erster Ge-

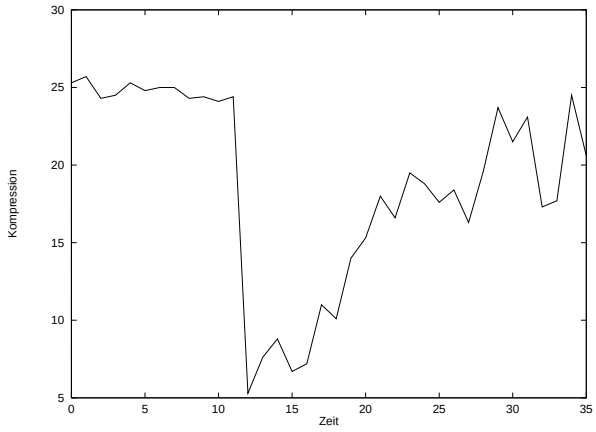


Abbildung 6.9: Kompressionsrate nach Aktivierung der Differenzenbildung

neration bezeichnet. Das unmittelbar folgende Bild, für das nun die Differenz mit dem Referenzbild berechnet werden kann, läßt sich mit etwa 5% am weitesten reduzieren. Diese starke Reduktion läßt sich auf den Umstand zurückführen, daß das Differenzbild an den meisten Stellen den Wert Null hat. Da das Kompressionsverfahren jedoch nicht verlustfrei arbeitet, ergeben sich mit jedem weiteren Bild Fehler in der Interpolation, die bei der Differenzbildung zu einem gewissen Rauschen in den zu komprimierenden Daten führen. Dieses Rauschen muß beim Ausdünnen erfaßt werden, um die zeitliche Aufschaukelung von Fehlern zu verhindern. So wächst nach dem zweiten komprimierten Bild die Anzahl der benötigten Punkte langsam an und pendelt sich auf einem Wert von etwas unterhalb von 20% ein. Längerfristig bringt die Kompression der Bilddifferenzen also keinen gravierenden Vorteil. Da sich die ersten gebildeten Differenzen erheblich besser komprimieren lassen als die folgenden, ist ein Vorgehen denkbar, bei dem, sobald die Anzahl der benötigten Punkte zu hoch wird, ein neues Referenzbild übertragen wird. Da die darauf folgenden Bilder wieder stärker komprimiert werden können ließe sich insgesamt eine Kompressionsrate erreichen, die im Mittel bei etwa 10% liegt. Abbildung 6.9 zeigt den zeitlichen Verlauf der Kompressionsrate. Man erkennt deutlich den Zeitpunkt, zu dem die Differenzenbildung aktiviert wurde.

Bewertung des Verfahrens

Wie die Testergebnisse bestätigen, eignet sich die Dünngittermethode sehr gut zur Kompression von Bildfolgen. Da das hier entwickelte Verfahren noch zahlreiche Ansätze für Optimierung bietet, ist die Entwicklung eines konkurrenzfähigen Kompressionsverfahrens für Bildfolgen möglich. Der beste Ansatz ist voraussichtlich die Kombination der Vorteile von hierarchischen Basen mit den im MPEG-Standard festgelegten Methoden.

Kapitel 7

Zusammenfassung und Ausblick

Die Implementierung der Dünngitteralgorithmen unter dem Gesichtspunkt der Interpolation sowie die Entscheidung, eine Datenstruktur zu verwenden, die für die spezielle Aufgabenstellung am geeignetesten waren, hat sich als ausgesprochen vorteilhaft erwiesen. Dadurch ist nicht nur die Verarbeitung der vorgegebenen Daten in effizienter Form möglich geworden, sondern es konnten auch Überlegungen angestellt werden, die bei einer weniger leicht handzuhabenden Datenstruktur im Rahmen dieser Arbeit kaum möglich gewesen wären. Gemeint ist die Konstruktion und Verwendung von höhergradigen, hierarchischen Polynombasen. Diese haben sich zwar für die Kompression von Bildern oder Bildfolgen in der Praxis wie erwartet als eher ungeeignet erwiesen, dafür hat ihre Untersuchung tiefere Einblicke in die Systematik der Konstruktion von Interpolationsoperatoren gewährt. Aufgrund dieser Erkenntnisse ist es möglich, für eine beliebige, vorgegebene Basisfunktion die hierarchische sowie die zugehörige Stützpunktbasis und den Interpolationsoperator automatisch zu generieren.

Ein wichtiger Vorteil, des in dieser Arbeit entwickelten Kompressionsverfahrens, ist die hierarchische Darstellung der Daten. Diese bietet Vorteile, die bei anderen Verfahren in dieser Form nicht gegeben sind. Denn allein durch die Hierarchisierung sind bereits inhärent Strategien gegeben, die beispielsweise bei der Übertragung der Daten zur adaptiven Anpassung an die zur Verfügung stehenden Kapazitäten dienen können. Darüber hinaus kann, was bei der derzeitigen Geschwindigkeit von Computernetzwerken besonders vorteilhaft ist, bereits nach der Übertragung eines Bruchteils der nötigen Informationen eine grobe Approximation des ganzen Datensatzes dargestellt werden. Aufgrund dieser groben Darstellung kann eine Entscheidung getroffen werden kann, ob der Rest der Daten von weiterem Interesse ist. So wird es im World Wide Web oft praktiziert, daß, bevor ein größeres Bild übermittelt wird, zunächst eine stark verkleinerte Version des Bildes übertragen wird. Der Benutzer kann dann anhand dieser Version entscheiden, ob das eigentliche Bild übertragen werden soll. Ist dies der Fall, so können die, für die verkleinerte Version des Bildes übertragenen Daten nicht weiter genutzt werden und müssen deswegen redundant übertragen werden. Würden

hierarchisch komprimierte Daten übertragen, so wäre die frühzeitige Darstellung einer verkleinerten Version des Bildes mit Hilfe von Daten möglich, die gleichermaßen zur Darstellung des eigentlichen Bildes genutzt werden könnten.

Die Untersuchungen und Versuche zur Verbesserung der Effizienz bei der Kompression von Bildsequenzen haben gezeigt, daß die im MPEG–Verfahren eingesetzten Methoden wie z.B. YUV–Farbmodell und Unterteilung der Daten in Blöcke, auch hier zu erheblichen Verbesserungen führten. Ein sehr vielversprechender Ansatzpunkt für die Verbesserung des in dieser Arbeit entwickelten Kompressionsverfahrens ist demnach eine Kombination des MPEG–Verfahrens mit der Darstellung von Daten durch hierarchischer Basen. Die einfachste Möglichkeit hierfür ist die Ersetzung der im MPEG–Verfahren verwendeten diskreten Cosinustransformation durch die Transformation in die Darstellung mittels einer geeigneten hierarchischen Basis. Dadurch wäre die Bildung größerer als der bisherigen 8×8 Blöcke problemlos möglich, die dann auch adaptiv an die zu komprimierenden Daten angepaßt werden können.

Ein Hindernis, das sich im Verlauf der Arbeit immer wieder zeigte, ist die mangelnde Hardwareunterstützung für hierarchische Verfahren. Cache–Strategien oder komplexere, hardwareoptimierte Zeichenfunktionen können nur schlecht genutzt werden, da sie meist auf lineares Vorgehen ausgelegt sind und hierarchische Verfahren für sie in manchen Fällen sogar den „worst–case“–Fall darstellen.

Trotz der derzeit mangelnden Hardwareunterstützung ist die Verwendung hierarchischer Verfahren zur Darstellung und Kompression von Bildfolgen und anderer Daten ein sehr vielversprechender Ansatzpunkt, dem in weiteren Forschungen größeres Augenmerk geschenkt werden sollte. Auch wäre es wünschenswert, daß das an Bedeutung gewinnende hierarchische Prinzip auch bei der Entwicklung von Spezialhardware berücksichtigt wird.

Anhang A

Benutzte Hardware und Software

A.1 Hardware

SGI–Workstations

Alle Arbeiten wurden auf Workstations der Firma Silicon Graphics Inc. (SGI) durchgeführt. Auf dieser Plattform stehen Hard- und Software zur Verfügung, die die Handhabung der zu komprimierenden Daten wesentlich erleichtert. Die entwickelten Kompressionsalgorithmen an sich sind hardwareunabhängig und sind ohne Probleme auf andere Architekturen übertragbar sein.

Die genutzten Rechner Indigo, Indy und Indigo^2 sind mit MIPS–Prozessoren vom Typ R4000, R4600 bzw. R4400 ausgestattet. Der Hauptspeicher beträgt jeweils 64 MB.

Die Rechner vom Typ Indy sind mit Video–Hardware ausgerüstet, die das Einspielen von Videosequenzen ermöglicht und an die eine digitale Kamera angeschlossen ist.

Zusätzliche Ausrüstung

Die Digitalisierung der Testbilder erfolgte mit einem Scanner der Firma Hewlett–Packard. Testvideosequenzen wurden mit Hilfe eines S–VHS–Videorecorders der Firma Panasonic und der oben erwähnten Video–Hardware eingespielt.

A.2 Software

Alle Programme wurden in der Programmiersprache C entwickelt. Zum Zweck der Visualisierung von Ergebnissen und zur Handhabung von Benutzereingaben wurde auf die von SGI entwickelte Graphikbibliothek GL und die Public–Domain–Bibliothek Forms zurückgegriffen.

Graphics Library (GL)

Diese Programmbibliothek von SGI stellt alle zur Programmierung von 2D–, 3D– und Bitmap–Graphik auf SGI–Workstations notwendigen Funktionen bereit. Diese Funktionen haben den Vorteil, daß sie von der Hardware unterstützt werden und somit eine bessere Graphikleistung erreichen. Weitere Informationen entnehme man dem Handbuch [2].

Forms Library

Speziell zur einfachen Erzeugung graphischer Benutzeroberflächen wurde die *Forms Library* verwendet. Sie stellt die notwendigen Bedienelemente wie Schalter, Schieber, etc. bereit und erzeugt C–Unterprogramme, die eingebunden werden können. Über die in Programmen verwendbaren Funktionen hinaus wird der Editor `fdesign` mitgeliefert, der es ermöglicht, eine graphische Benutzeroberfläche interaktiv zu erzeugen und zu bearbeiten. Dieses Programm erleichtert die Erstellung einer Benutzerschittstelle erheblich und ist somit eine große Zeitersparnis. Eine genauere Beschreibung findet sich im Handbuch [13].

Digital Media Development Option (DMDEV)

Dieses von SGI entwickelte und vertriebene Softwarepaket enthält Programmbibliotheken zur Entwicklung von Software im Audio– und Video–Bereich, der Kompression solcher Daten, sowie Funktionen zum Ansprechen von Spezialhardware wie Videodigitizer, CD–ROM, etc.

SGI–RGB–Format

Als Eingabeformat für Einzelbilder wurde das SGI–RGB–Format verwendet. Es handelt sich um ein Format, das einzelne Bildpunkte mit 24 bit Farbtiefe codiert. Jeder Bildpunkt wird jedoch durch 4 Bytes (32 bit) dargestellt, wobei das zusätzliche Byte den Blendfaktor α beinhaltet. Dieser wird in den hier entwickelten Anwendungen nicht berücksichtigt. Es werden vom System mehrere Funktionen zur Verfügung gestellt, die den Umgang mit diesem Datenformat erleichtern.

SGI–Movie–File–Format

SGI–Movie–Dateien die Sequenzen von Einzelbildern und Audiodaten beinhalten wurden genutzt um Bildsequenzen effizient einzulesen. Der Zugriff auf die Datei erfolgt über Bibliotheksfunktionen, die Bilder im SGI–RGB–Format zurückliefern.

Eine weitere genutzte Quelle für Bildsequenzen ist eine digitale Kamera, die mit Hilfe der DMDEV–Bibliothek konfiguriert und abgefragt werden kann.

Für weitere Informationen zur DMDEV–Bibliothek siehe [1].

Maple V

Zur schnellen Entwicklung und Überprüfung der bei der Untersuchung von Polynombasen anfallenden mathematischen Algorithmen wurde Maple V verwendet. Es handelt sich dabei um ein Computeralgebra–Programm zur symbolischen Manipulation mathematischer Formeln. Dadurch wird das Rechnen mit mathematischen Ausdrücken vereinfacht. Hier wurden neben den mathematischen Standardfunktionen vor allem die Funktionen `piecewise` zur Definition stückweiser Funktionen, `interp` zur Berechnung der Interpolationsfunktion zwischen gegebenen Stützpunkten, `lcm` zur Berechnung des kleinsten gemeinsamen Vielfachen und `inverse` zur Berechnung einer inversen Matrix verwendet. Einzelheiten zu Maple V findet man in den Handbüchern [5] und [6].

Anhang B

Erstellte Programme

B.1 Bildkompression

Zunächst wurde eine Programm- und Benutzerumgebung erstellt, die es ermöglichen sollte, Dünngitter–Datenkompression nach verschiedenen Vorgehensweisen anhand unterschiedlicher Testfunktionen und -bilder zu analysieren und zu bewerten. Das Programm, in dem die Dünngitter–Algorithmen entwickelt und untersucht wurden, heißt `dgbk`.

Das Programm öffnet nach dem Start mehrere Fenster, in denen die Eingabedaten, das ausgedünnte Gitter, die Interpolation der Eingabedaten und die Differenz von Eingabedaten und deren Interpolation angezeigt wird. Ein fünftes Fenster stellt ein Kontrollfeld dar und dient zur Anzeige und Änderung verschiedener Einstellungen.

Das Kontrollfeld enthält einen Schieber und ein Eingabefeld, mit denen der zulässige Interpolationsfehler ε eingestellt werden kann. Der Auto–Knopf startet ein automatisches Weiterschalten der Fehlerschranke.

Durch Knopfdruck kann aus verschiedenen Varianten der Interpolationsstrategie ausgewählt werden. Normal bezeichnet das rein lokale Eliminationskriterium, das bei gleichbleibender Fehlerschranke unabhängig von der Existenz irgendwelcher Nachfolger Punkte ausdünnt. Keep Father entspricht der Vorgehensweise bei Verwendung von Bäumen als Datenstruktur, Punkte mit nichtausgedünnten Nachfolgern werden ebenfalls nicht ausgedünnt. Bei Compensate wird im Gegensatz zu Normal die Kompensationsstrategie verwendet, in der die Fehlerschranke von Level zu Level angepaßt wird . Der Kompensationsfaktor κ kann über einen Schieber eingestellt werden. Ein κ von 1.0 entspricht dabei der Methode Normal. Regular schaltet auf die Verwendung regulärer dünner Gitter um, die nicht adaptiv verfeinern. Über einen Schieber kann die Anzahl der Verfeinerungsstufen eingestellt werden.

Im Gegensatz zu den bisherigen Methoden verwenden die folgenden Methoden keine stückweise linearen Basisfunktionen. Const. Basis benutzt zur Interpolation

konstante Basisfunktionen. Hier kann die Fehlerakkumulation durch Angabe eines κ ebenfalls kompensiert werden. Setzt man den Kompensationsfaktor auf 1.0, so entspricht dies der unkompensierten Variante (vgl. Normal). Variante Polyn. Basis benutzt Polynombasisfunktionen zur Berechnung der Interpolation. Der Grad der Basisfunktionen kann in einem Eingabefeld angegeben werden.

Ein weiterer Schieber legt fest, bis zu welchem Level nichtausgedünnte Daten bei der Decodierung verwendet werden sollen. Dies ermöglicht es, durch Weglassen der unteren Level zu beurteilen, welchen Beitrag die Punkte dieser Level zur Bildqualität liefern. Ein Selektorfeld ermöglicht die Auswahl verschiedener vordefinierter Testfunktionen, die anstatt eines Bildes als Eingabedaten verwendet werden können. Weitere Knöpfe aktivieren Zeitmessungen, die Lauflängencodierung der ausgedünnten Daten, das Schreiben der RLE–codierten Daten, das Schreiben der interpolierten Daten als Bild und die Skalierung der Differenzanzeige zum besseren Erkennen von geringen Fehlern.

Bei jeder Änderung der Parameter werden die Daten automatisch komprimiert und interpoliert. Abhängig von den gewählten Varianten werden folgende Daten dabei jedesmal über die Standardausgabe ausgeben:

`EPS:`	Fehlerschranke ε
`DEG:`	Grad der Polynombasis
`KOMP:`	Kompensationsfaktor κ
`REG_LEV:`	Anzahl der Level im regulären Gitter
`PTS:`	Anzahl der Punkte im Gitter
`PTS_%`	Anzahl der Punkte im Gitter in Prozent
`PTS_USED:`	Anzahl der zur Decodierung verwendeten Punkte
`PTS_USED_%`	Anzahl der zur Decodierung verwendeten Punkte in Prozent
`RLE:`	Anzahl der Bytes nach der Lauflängencodierung
`RLE_%`	Anzahl der Bytes nach der Lauflängencodierung in Prozent
`GZIP:`	Anzahl der Bytes nach einer Kompression mit `gzip -9`
`GZIP_%`	Anzahl der Bytes nach der `gzip`–Kompression in Prozent
`MAX_ERROR:`	Maximaler Interpolationsfehler
`ERR_VAR:`	Fehler in der L_2–Norm
`H_ERR:`	Hierarchischer Fehler

Durch Aufrufparameter können die meisten Wahlmöglichkeiten voreingestellt werden. Darüber hinaus kann festgelegt werden, welche Bilddatei eingeladen werden soll und in welche Dateien die Ausgaben (RLE–codiertes Bild, interpoliertes Bild) geschrieben werden sollen. Für die Verwendung von Testfunktionen kann die gewünschte Größe voreingestellt werden. Die Aufrufparameter haben folgende Bedeutung:

`-l <filename>`	Eingabebild
`-s <size>`	Größe des Datenfeldes bei Testfunktionen
`-e <epsilon>`	Fehlerschranke ε
`-t <number>`	Nummer der Testfunktion
`-g`	Abschaltung der graphischen Ausgabe
`-a`	Automatisches Weiterschalten von ε
`-z <factor>`	Zoomfaktor für graphische Ausgabe
`-T`	Zeitmessung
`-K <kappa>`	Kompensation mit Faktor κ
`-F`	Klassische Variante
`-R <level>`	Verwendung regulärer dünner Gitter
`-w <filename>`	Festlegen des Ausgabedateinamens für das interpolierte Bild
`-W <filename>`	Festlegen des Ausgabedateinamens und Schreiben des interpolierten Bildes
`-o <filename>`	Festlegen des Ausgabedateinamens für das RLE–codierte Bild
`-O <filename>`	Festlegen des Ausgabedateinamens und Schreiben des RLE–codierten Bildes
`-r`	Aktivieren der Lauflängencodierung
`-S`	Skalierte Darstellung der Differenz
`-P <degree>`	Verwendung von Polynombasisfunktionen vom Grad `<degree>`
`-G`	Nachfolgende Kompression der RLE–codierten Daten mittels `gzip -9`
`-L`	Nichtlineare Fortschaltung bei `-a`

Das Programm wird durch Drücken der Escape–Taste beendet.

Durch geeignete `#define`–Anweisungen in einem zentralen Programmteil wurde eine gewisse Datentypunabhängigkeit erreicht, d.h. durch Rekompilierung kann diese Einstellung geändert werden. Dies ermöglicht die Identifikation von Effekten, die aufgrund der Rundungsfehler bei den Berechnungen durch Ganzzahlarithmetik entstehen.

B.2 Basis–Editor

Zur leichteren Konstruktion von hierarchischen Basen wurde der Editor `bed` entwickelt, der auf verschiedenen Leveln die individuelle Skalierung einzelner Basisfunktionen zuläßt, diese aufsummiert und anzeigt. Es besteht an jedem Punkt die Möglichkeit, zwischen einer stückweise linearen oder quadratischen bzw. einem Prewavelet als Basisfunktion zu wählen. Diese Wahl kann durch drei Knöpfe auch global für alle nicht bereits verwendeten Schieber getroffen werden. Der Editor besitzt vier Ebenen von Schiebern, mit denen die Basisfunktionen der entsprechenden Level manipuliert werden können. Durch Drücken der Space–Taste wird der zuletzt betätigte Schieber auf Null gesetzt. Die Return–Taste setzt alle Schieber auf Null zurück. Die Escape–Taste beendet das Programm.

B.3 Video–Kompression

Für die Erweiterung des zur Bildkompression entwickelten Programms auf die Verarbeitung von Bildfolgen wurden zunächst Funktionen implementiert, die das Einladen von Bildsequenzen erlauben. Es ist möglich mehrere Einzelbilder im SGI-RGB-Format oder SGI-Movie-Dateien als Datenquelle zu verwenden. Dabei kann beim Aufruf des Programms durch Angabe der Nummer des ersten und letzten Bildes sowie des Abstandes zwischen den einzuladenden Bildern festgelegt werden wieviele und welche Bilder geladen werden sollen. Auf Rechnern des Typs Indy mit angeschlossener Kamera kann zusätzlich über Kommandozeilenparameter die Nutzung dieser Kamera als Eingabemedium aktiviert werden. Im Übrigen wurde im Rahmen der Erweiterungen das Programm von `dgbk` auf den Namen `sgvc` umgetauft.

Die vier Fenster zur Anzeige der Originalsequenz, des ausgedünnten Gitters, der Interpolation und der Differenz zwischen Original und Interpolation wurde beibehalten. Das Bedienfeld wurden um Knöpfe und Schieber erweitert, die das Abspielen der eingeladenen Bildsequenz sowie Einzelbildweiterschaltung vorwärts und rückwärts erlauben. Ein Knopf zum Unterbrechen des Abspielens wurde ebenso hinzugefügt wie ein Eingabefeld und ein Schieber, die das direkte Anspringen von bestimmten Bildern der Sequenz erlauben. Der Schieber wird darüberhinaus genutzt um die Position des aktuell angezeigten Bildes innerhalb der Sequenz zu zeigen.

Ist der Knopf Do Border aktiviert, so wird der Datensataz in einer Ecke des umschriebenden $2^N + 1$–Würfels plaziert und eine separate Randbehandlung durchgeführt, es sei denn diese wurde durch Drücken des Keep Border Knopfes verhindert. Ist Do Border inaktiv, so wird der Datensatz in das Zentrum des umschreibenden Würfels gerückt. YCbCr schaltet auf die Verwendung des entsprechenden Farbmodells um. Blocking unterteilt den Datensatz in Würfel, die separat ausgedünnt werden. Die Kantenlänge dieser Würfel entspricht, wenn nicht über Kommandozeilenparameter ander festgelegt dem Minimum aus X– und Y–Auflösung der Bilder sowie der Anzahl der Bilder. Durch drücken des Knopfs Seq.Diffs wird auf die Kompression von Differenzen aufeinanderfolgender Bilder umgeschaltet.

Verschiedene Bedienelemente wurden beibehalten und erfüllen dieselbe Funktion wie im Program `dgbk`. Dazu gehören der Schieber zur Wahl der Fehlerschranke ε zusammen mit dem zugehörigen Eingabefeld und dem Auto–Knopf, der Knopf Compensate, der Schieber zur Wahl des Kompensationsfaktors, die Knöpfe zur Aktivierung der Zeitmessungen, der Lauflängencodierung, des Schreibens der RLE–codierten Daten und die Skalierung der Differenzanzeige. Es wurde auch das Auswahlfeld zur Bestimmung von Testfunktionen beibehalten, hinter den einzelnen Auswahlmöglichkeiten verbergen sich jedoch andere Funktionen. Die in `dgbk` vorhandene Einstellung Normal stellt bei `sgvc` die Standardeinstellung dar, die nur inaktiv ist, wenn Compensate gedrückt wurde. Aus diesem Grund

wurde auf diesen Knopf in der Bedienoberfläche verzichtet. Ein neu hinzugekommener Knopf verhindert das erneute Ausdünnen der Daten bei der Änderung von kompressionsrelevanten Parametern, um das gleichzeitige Ändern verschiedener Parameter zu erlauben. Ein Knopf verhindert die Rückinterpolation der Daten. Diese Option wurde im Zusammenhang mit der Implementierung der Differenzenbildung implementiert und dient im wesentlichen der Fehlersuche. Ein Auswahlfeld erlaubt es bei der Anzeige des ausgedünnten Gitters die drei verschiedenen Farbkanäle getrennt voneinander zu untersuchen. Auf der linken Seite des Eingabefeldes zeigt ein vertikaler roter Balken die erzielte Kompression an. Ist der Balken ganz oben, so wurden 100% der Daten für die Interpolation des Datensatzes verwendet.

Als Ergebnis jedes Kompressionsvorgangs werden folgende Werte auf die Standardausgabe ausgegeben:

`Eps:`	Fehlerschranke ε
`Komp:`	Kompensationsfaktor κ
`Pts:`	Anzahl der Punkte im Gitter
`Pts_%`	Anzahl der Punkte im Gitter in Prozent
`RLE:`	Anzahl der Bytes nach der Lauflängencodierung
`RLE_%`	Anzahl der Bytes nach der Lauflängencodierung in Prozent
`ERR_MAX:`	Maximaler Interpolationsfehler
`ERR_VAR:`	Fehler in der L_2–Norm
`ERR_H:`	Hierarchischer Fehler
`ERR_MW:`	Mittelwert des Fehler

Falls es angebracht erschien, so wurde jeweils in Klammern hinter dem Wert, der sich auf den gesammten Datensatz bezieht, die Werte für die einzelnen Farbkanäle ausgegeben.

Ebenso wie bei `dgbk` ist auch bei `sgvc` die Voreinstellung der meisten Parameter durch Komandozeilenparameter möglich. Die Aufrufparameter haben folgende Bedeutung:

`-l <filename>`	einzelnes Eingabebild
`-f <filename>`	Dateiname für Bildfolgen
`-m <filename>`	Eingabe-SGI-Movie-Datei
`-C`	Verwendung der Kamera als Datenquelle
`-r <range>`	Nummer des ersten und letzten zu ladenden Bildes
`-s <size>`	Größe des Datenfeldes bei Testfunktionen
`-e <epsilon>`	Fehlerschranke ε
`-t <number>`	Nummer der Testfunktion

`-g`	Abschaltung der grapischen Ausgabe
`-a`	Automatisches Weiterschalten von ε
`-z <factor>`	Zoomfaktor für graphische Ausgabe
`-T`	Zeitmessung
`-K <kappa>`	Kompensation mit Faktor κ
`-Y`	Verwendung des YCbCr–Farbmodells
`-b`	entspricht dem Do Border Knopf
`-E`	aktiviert die Lauflängencodierung
`-R <filename>`	setzt den Dateinamen für das Schreiben der RLE–Daten
`-W`	Schreiben der RLE–Daten beim ersten Kompressionsdurchgang
`-D`	Skalierung der Differenzenanzeige
`-c`	verhindert die Kompression
`-G <kanal>`	Setzen des Farbkanals für die Gitterausgabe
`-Z 1/<denom>`	Setzen des Verkleinerungsfaktors für die Videokamera
`-M`	Ausdünnen der Daten am Ort, es wird außer für den Datensatz kein zusätzlicher Speicher benötigt
`-S`	Kompression von Differenzen aufeinanderfolgender Bilder
`-B <size>`	Setzen der Blockgröße und Aktivierung von Blocking
`-O`	der Datensatz wird für Testläufe nur zweimal im Speicher gehalten
`-L`	nichtlineare Epsilonweiterschaltung
`-H <filename>`	Schreiben der unkomprimierten Daten

Das Programm wird durch Drücken der Escape–Taste beendet.

Anhang C

Beispieldatensätze

C.1 Bild–Testdaten

Folgende Testbilder wurden für die Untersuchung der Kompression von Grauwertbildern verwendet:

- Eine Photographie aus dem Salzkammergut (`landscape`, Abb. C.1).
- Ein Cartoon von Loriot (`loriot`, Abb. C.1).
- Die Computergraphik eines Pinguins auf einem Schiff (`schiff`, Abb. C.2).
- Ausschnitt aus Höhendaten des Voralpenlandes (`hoehen`, Abb. C.2).
- Ein Schachbrettmuster aus Quadraten von 16×16 Pixeln mit den Grauwerten 143 und 255 (`chess`, Abb. C.3).
- Das Grauwertbild der Testfunktion

  $$(-32x^4 + 64x^3 - 42x^2 + 10x) \cdot (-32y^4 + 64y^3 - 42y^2 + 10y)$$

  (`hill`, Abb. C.3).
- Ein ein Pixel breites diagonales Kreuz mit den Grauwerten 0 und 255 (`kreuz`, Abb. C.4).

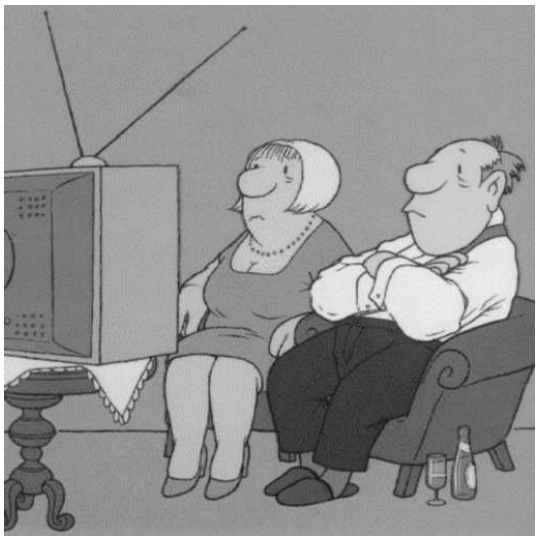



Abbildung C.1: Testbild `landscape`, Testbild `loriot`

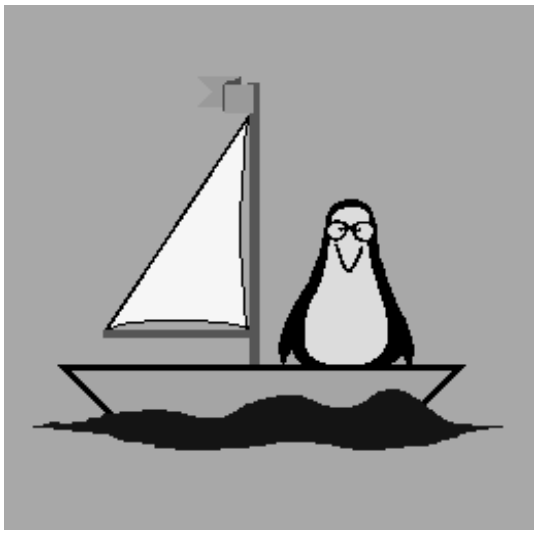
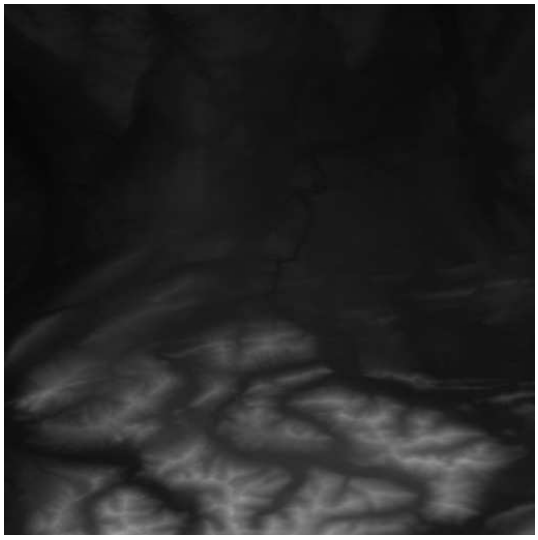



Abbildung C.2: Testbild `schiff`, Testbild `hoehen`

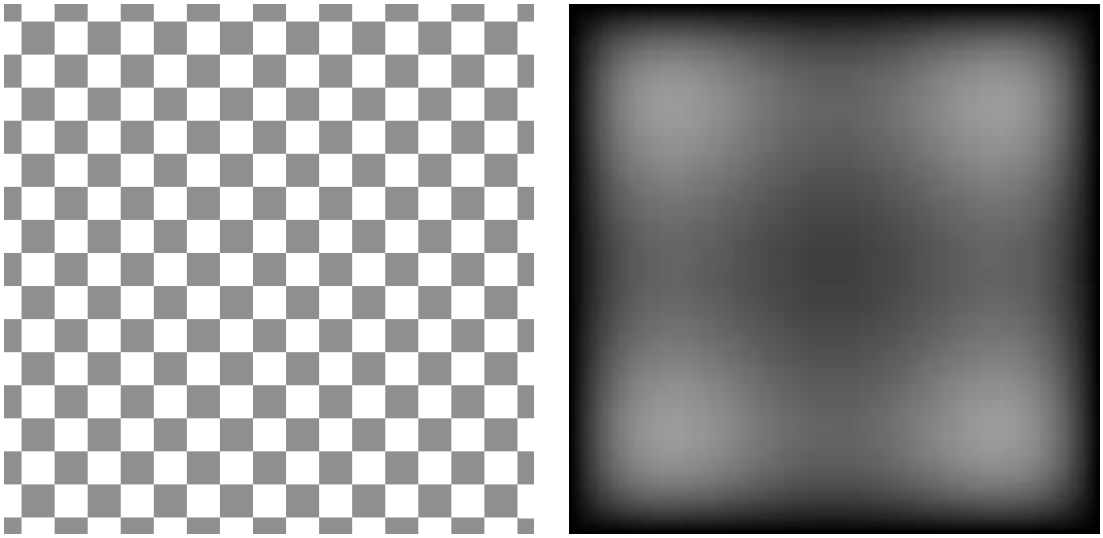


Abbildung C.3: Testbild `chess`, Testbild `hill`

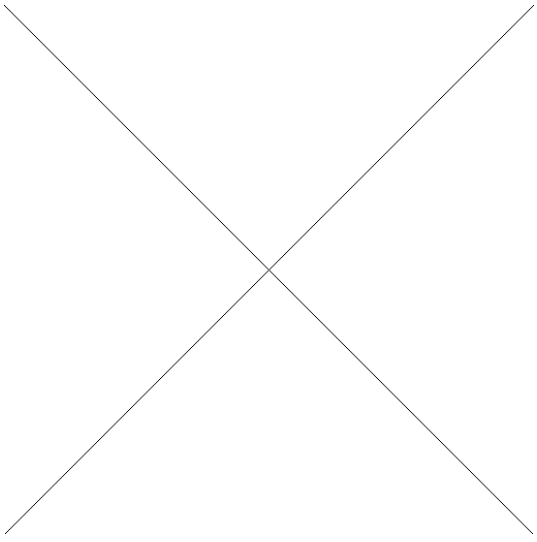


Abbildung C.4: Testfunktion `kreuz`

C.2 Video–Testdaten

Folgende Bildsequenzen wurden für die Untersuchung der Kompression von Bildfolgen verwendet:

- Die Aufnahme einer Tagesschau-Sendung in der ARD, vom Sonntag den 21. Mai 1995 (`tagesschau`, Abb. C.5).
- Der Ausschnitt eines Tennisspiels (`tennis`, Abb. C.6).
- Eine Szene aus dem Film „Star Wars 3 – Die Rückkehr der Jedi-Ritter“ (`star_wars`, Abb. C.7).
- Eine Szene aus dem synthetisch generierten Film „Dünne Gitter“, der im Rahmen von [10] entstanden ist. (`dünne_gitter`, Abb. C.8).

Die Beispiele `tagesschau` und `tennis` stammen von einer S-VHS-Aufnahme, `star_wars` von einer VHS-Videokasette. Die ursprüngliche Überspielung der Sequenzen auf den Rechner erfolgte mit Hilfe einer JPEG-Kompressionskarte, wobei die Bildqualität so hoch wie möglich (ca. 85%) gewählt wurde. Die Digitalisierung lieferte Bilder in PAL-Auflösung (768×576 Pixel, 25 Bilder pro Sekunde) die für die schnellere Verarbeitung auf $\frac{1}{4}$ und $\frac{1}{8}$ der ursprünglichen Bildauflösung verkleinert worden sind. Die Sequenz `dünne_gitter` hat eine Auflösung von 360×270 Pixeln und eine Bildrate von 10 Bildern pro Sekunde.

Abbildung C.5: Testsequenz `tagesschau`, Bild 1, 50 und 100

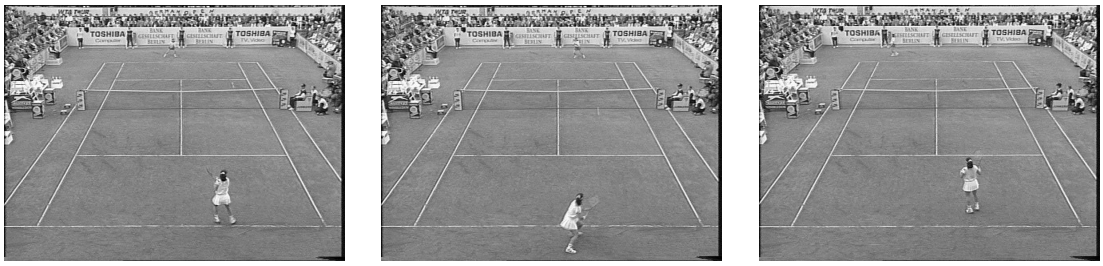


Abbildung C.6: Testsequenz `tennis`, Bild 1, 50 und 100

Abbildung C.7: Testsequenz `star_wars`, Bild 1, 60 und 100

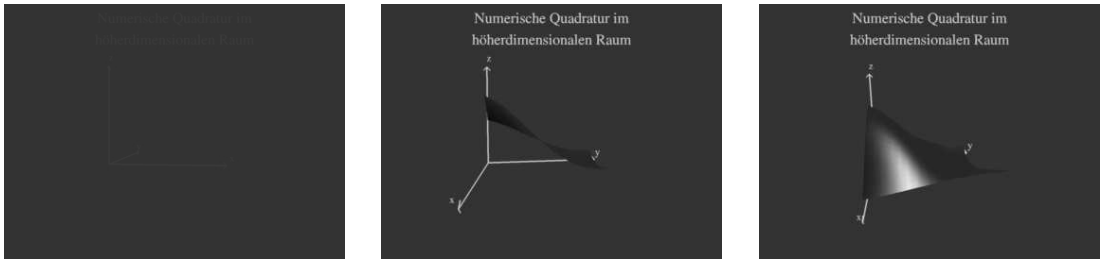


Abbildung C.8: Testsequenz `dünne_gitter`, Bild 1, 25 und 50

Abbildungsverzeichnis

Literaturverzeichnis

[1] *Digital Media Programming Guide: Audio, MIDI, Video, and Compression.* Silicon Graphics, Inc. 10 1993. Document Number: 007-1799-010.

[2] *Graphics Library Programming Guide.* Silicon Graphics, Inc. 3 1991. Document Number: 007-1210-040.

[3] Bungartz, H.: *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung.* Dissertation. Institut für Informatik, TU München. 6 1992.

[4] Bungartz, H.: *Higher Order Finite Elements on Sparse Grids.* SFB Bericht 342/01/95 A. Institut für Informatik, TU München. 2 1995.

[5] Char, B.W., K.O. Geddes, G.H. Gonnet, G.L. Leong, M.B. Monagan und S.M. Watt: *Maple V Language Reference Manual.* 3 1991.

[6] Char, B.W., K.O. Geddes, G.H. Gonnet, G.L. Leong, M.B. Monagan und S.M. Watt: *Maple V Library Reference Manual.* 3 1991.

[7] Crandall, R.E.: *Projects in Scientific Computation.* Springer Verlag, New York: 1994.

[8] Daubechies, I.: *Ten lectures on wavelets.* Society for Industrial and Applied Mathematics, Philadelphia: 1992.

[9] Foley, J.D., A. van Dam, S.K. Feiner und J.F. Hughes: *Computer Graphics – Principle and Practice.* 2. Aufl. Addison-Wesley Publishing Company: 1992.

[10] Gerstner, T., und A. Paul: *Erstellung eines Lehrfilms über die Dünngittermethode.* Fortgeschrittenenpraktikum. Institut für Informatik, TU München. 1994.

[11] Hiller, K.: *Datenkompression mit dem Dünn–Gitter–Verfahren.* Diplomarbeit. Institut für Informatik, TU München. 1993.

[12] Jackson, R., L. MacDonald und K. Freeman: *Computer Generated Color.* 1. Aufl. John Wiley & Sons, Chichester: 1994.

[13] Overmars, M. H.: *Forms Library, A Graphical User Interface Toolkit for Silicon Graphics Workstations.* Department of Computer Science, Utrecht University. November 1992.

[14] Pennebaker, W., und J. L. Mitchell: *JPEG Still Image Data Compression Standard.* Van Nostrand Reinhold, New York: 1993.

[15] Riedel, K.: *Datenreduzierende Bildcodierung: über 50 verschiedene Verfahren.* Franzis-Verlag, München: 1986.

[16] Sedgewick, R.: *Algorithmen in C.* Addison-Wesley Publishing Company: 1992.

[17] Thurner, V.: *Eine ganzzahlige Arithmetik zur adaptiven Lösung der Poisson-Gleichung mit hierarchischen Basen.* Diplomarbeit. Institut für Informatik, TU München. 1994.

[18] Webers, J.: *Handbuch der Film- und Videotechnik.* 2. Aufl. Franzis-Verlag, München: 1988.

[19] Yserentant, H.: „Hierarchical Bases". In O'Malley, R.E. et al., Jr. (Ed.): *ICIAM 91: Proceedings of the Second International Conference on Industrial and Applied Mathematics.* SIAM, Philadelphia: 1992.

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

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