Diplomarbeit

Adaptiv hierarchische Datenkompression mit Fehlerkontrolle basierend auf raumfüllenden Kurven

angefertigt am Institut für Angewandte Mathematik
vorgelegt der Math.-Nat. Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

November 2000

von Felix Schumacher
aus Bonn

Referent: Prof. Dr. M. Griebel
Koreferent: Prof. Dr. M. Rumpf

Ich versichere, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Bonn, November 2000

Inhaltsverzeichnis

1 Einleitung

Schon immer wollten die Menschen mehr Informationen speichern als Kapazitäten vorhanden waren. Heutige Wissenschaften häufen allerdings immer größere Datenmengen in immer kürzeren Zeitintervallen an. Diese Daten müssen alle gespeichert werden.

In den Geowissenschaften sind dafür Satellitenbilder ein gutes Beispiel. Noch vor 100 Jahren waren die Gewinne an Kenntnissen über die Landmassen der Welt durch die Geschwindigkeit der Landvermesser beschränkt. Ein Landvermesserteam könnte pro Tag nur ein paar Quadratkilometer vermessen. Auch sollte ein solches Team nicht jeden Busch erfassen, wenn der zeitliche Aufwand im Rahmen bleiben soll.

Heutzutage können große Teile der Vermessung mit Hilfe von Satellitenbildern, Radarerkundungen, sonstigen Luftaufnahmen und nicht zuletzt Computern genauer und schneller durchgeführt werden. Auf einer Luftaufnahme sind etwa nicht nur die Entfernungen zwischen zwei Landpunkten abzulesen, auch die Farben der Flächen sind zu erkennen.

Nicht nur in den Geowissenschaften werden moderne Meßtechniken eingesetzt und damit Meßgenauigkeit und Meßgeschwindigkeit erhöht. In der Medizin wird ständig daran gearbeitet, Messungen zu ermöglichen und zu verbessern. So sind aus den alten Röntgenbildern heutzutage dreidimensionale Computerscans wie Computer Tomographie (CT) oder Magnet Resonanz Tomographie (MRT) geworden.

Da nicht alle Daten gespeichert werden können, muß eine Auswahl getroffen werden. Dies wurde bei solchen Problemen schon immer gemacht. Noah durfte auf seiner Arche von jeder Tierart nur ein Paar mitnehmen. Und auch bei Zahlen wird normalerweise nur eine Auswahl gespeichert. Als Beispiel könnte hierfür die Zahl π dienen. Niemand kann sich alle Stellen von π merken, dies ist aber in den meisten Fällen auch nicht nötig. Wenn mit der Zahl π gerechnet wird, so geschieht dies meist mit einer Näherung auf acht Stellen hinter dem Komma. Dies ist für einen Taschenrechner völlig ausreichend, denn dessen Rechengenauigkeit liegt genau in diesem Bereich. Auch werden später von dem Ergebnis meist nur die ersten drei Stellen hinter dem Komma mit ausgewertet. Es werden also Informationen weggelassen. Und zwar solche Informationen, die unwichtig, irrelevant erscheinen. Daß die unwichtigen Stellen bei Zahlen in unserem Dezimalsystem rechts stehen, hat seinen Ursprung in der hierarchischen Anordnung der Zahlen.

Angenommen es stünden nur Striche zur Darstellung von Stückzahlen zur Verfügung. Für kleine Mengen ist das durchaus praktisch und wird auch eingesetzt, aber bei einer Stückzahl von mehr als 10 oder 20 wird dieses Verfahren schnell unübersichtlich. Daher werden diese Striche häufig zu Fün-

fergruppen gepaart. Durch dieses neue Symbol wird auch eine etwas größere Menge noch übersichtlich dargestellt. Bei einer Stückzahl von mehr als 50 wird aber auch diese Methode schnell unübersichtlich. An dieser Stelle treten die Zahlen von Null bis Neun auf.

Im Dezimalsystem werden den Positionen der Ziffern noch Werte zugeordnet. Diese Werte werden mit den dort stehenden Zahlwerten multipliziert und alle so erhaltenen Werte addiert ergeben die endgültige Zahl. Die Werte werden so ausgewählt, daß eine Eins an einer beliebigen Stelle mehr Bedeutung hat als alle rechts von ihr stehenden Zahlwerte zusammen. Wenn also die Ziffern von Null bis Neun die Basis bilden, so sind die Positionswerte Exponenten zur Basis 10, also z. B.

$$372,5 \quad = \quad 3 \cdot 10^2 + 7 \cdot 10^1 + 2 \cdot 10^0 + 5 \cdot 10^{-1}$$

Je ungenauer eine Zahl dargestellt werden soll, um so mehr Ziffern werden auf der rechten Seite auf Null gesetzt, bei eventueller Rundungskorrektur der davon links stehenden Zahlen. So könnte beispielsweise aus der 372,5 eine 400 werden. Die Differenz von 27,5 wäre für den Betrachter irrelevant gewesen und deswegen weggelassen worden. Von den ursprünglich vier zu merkenden Zahlwerten ist nur noch eine Stelle übrig geblieben.

Eine andere Möglichkeit Daten zu komprimieren, ist das Weglassen von redundanten Daten. Auch diese Art der Komprimierung ist im Alltag häufig zu beobachten. So werden Teile gleicher Art zusammengefaßt und nur die Summe und ein Teil der Art genannt, z.B drei Kühe. Bei dieser Art von Kompression gehen keine Informationen verloren. Es ist das gleiche zu sagen, daß auf einer Weide drei Kühe stehen oder jede Kuh einzeln aufzuzählen. Die erste Darstellungsart ist nur erheblich kürzer. Neben dem Zusammenfassen von gleichen Teilen wird auch noch häufig mit Abkürzungen gearbeitet. Diese haben nur in ihrem Kontext einen definierten Inhalt. Wenn in einem Text *„z. B.“* auftaucht, wird es als *zum Beispiel* gelesen. Taucht *„iff“* in einem Kinderbuch anstatt in einem englischen Text über Mathematik auf, denkt jeder wohl eher an einen Tippfehler als an die eigentliche Bedeutung *„if and only if“*.

Solche Abkürzungen werden sehr häufig automatisch eingesetzt. Der Weg zum nächsten Supermarkt wird vielleicht zweimal erklärt, danach reicht die Angabe des Ortes, um den Weg zu finden. Hier wird also immer ein kurzes Wort für eine längere Beschreibung oder ein langes Wort eingesetzt, um Platz oder Zeit zu sparen.

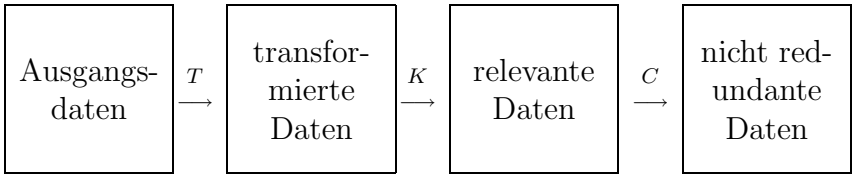
Der Unterschied der beiden oben vorgestellten Kompressionsverfahren liegt in der Art der Daten, die reduziert werden. Das erste Beispiel nähert die vorgegebenen Zahlen an, damit sie einprägsamer, also kürzer zu speichern

sind. Es läßt *irrelevante* Daten weg. Das zweite Verfahren ersetzt häufig wiederkehrende Daten, die viel Platz verbrauchen, durch kürzer zu speichernde Daten. Dieses Verfahren reduziert die *Redundanz* in den vorgegebenen Daten.

Häufig können beide Arten der Kompression, wenn sie hintereinandergeschaltet werden, noch deutlich bessere Kompressionsraten erzielen als einzeln. Als Beispiel sei die Zahlenfolge 310, 295, 305, 302, 297 gegeben. Zuerst werden die Zahlen auf den Hunderterbereich genähert. Die Folge lautet jetzt 300, 300, 300, 300, 300. Es ist deutlich zu sehen, daß jetzt Platz gespart werden kann, wenn ein Wort für die 300 kürzer wäre als die 300 selbst oder Wiederholungen an sich in kurzer Form aufgeschrieben werden könnten; etwa wie 5×300.

Die oben aufgeführten Arten der Kompression sind in Tabelle 1 noch einmal kurz aufgeführt.

Insgesamt kann die verlustbehaftete Kompression von Daten mit folgendem Modell beschrieben werden:



Die Ausgangsdaten werden mittels des Operators T in ein anderes Darstellungssystem überführt. Die Striche aus dem ersten Beispiel also in die Dezimaldarstellung. Die Transformation T ist eine Basistransformation. Die so erhaltenen Daten werden auf *Relevanz* getestet. Die irrelevanten Daten werden durch den Operator K weggelassen. (In obigem Zahlenbeispiel also die Rundung auf die Hunderterstelle.) Zum Schluß werden noch alle *redundanten* Daten aus den jetzt erhaltenen Daten herausgefiltert. Diese Filterung geschieht durch den Operator C. Wenn jetzt ein Datenvektor $\mathbf{f}$ die zu komprimierenden Daten enthält und T, K und C die oben aufgeführten Operatoren

	redundant	$\neg$*relevant*
keine Kompression	ja	ja
verlustlose Kompression	nein	ja
verlustbehaftete Kompression	ja	nein
verlustbehaftete Kompression	nein	nein

Tabelle 1: Arten der Kompression, wenn redundante oder nicht relevante Daten weggelassen werden. Relevante und nicht redundante Daten müssen immer behalten werden.

darstellen, so kann die Kompression als eine Verkettung dieser Operatoren angesehen werden. Also

$$\hat{\mathbf{f}} = CKT\mathbf{f},$$

wobei $\hat{\mathbf{f}}$ die verlustbehafteten Daten in der komprimierten Form sind. Um die ursprünglichen Daten wiederzuerlangen, müssen die Operatoren invertierbar sein. Dies kann aber nur für C und T gelten. Für K war gefordert, daß es irrelevante Daten wegläßt. Die Pseudo-Inverse K_*^{-1} von K muß aber für die durch die Transformation von K weggelassenen nicht relevanten Daten vernünftige Werte einsetzen. Für weggelassene Dezimalstellen hinter dem Komma wären das z. B. Nullen. Die Rücktransformation – also die Dekomprimierung – lautet dann:

$$\begin{aligned} \tilde{\mathbf{f}} &= T^{-1}K_*^{-1}C^{-1}\hat{\mathbf{f}} \\ &= T^{-1}K_*^{-1}C^{-1}CKT\mathbf{f}. \end{aligned}$$

Der Fehler der bei dieser Kompression gemacht wird ist:

$$\varepsilon = ||\tilde{\mathbf{f}} - \mathbf{f}||.$$

Nun müssen nur noch geeignete Operatoren T, K und C gefunden werden. Hierbei ist zu beachten, daß die geeigneten Operatoren von den vorher angewandten abhängen können.

In dieser Arbeit werden mögliche Operatoren T, K und C vorgestellt und verglichen. Ziel ist eine gute Datenkomprimierung unter Einhaltung eines vorgegebenen Fehlers in der Maximumsnorm zu erreichen. Dazu werden für die Transformation T die Fraktale-, Fourier-, Wavelet- und hierarchische Finite Elemente-Transformation besprochen. Für die Kompression K werden a priori- und adaptive Fehlerrechner vorgestellt. Die Kodierung C wird in zwei Bereiche aufgeteilt, die Struktur-Kodierer C_s und die Daten-Kodierer C_d. Bei den Struktur-Kodierern werden einige an die Hierarchische Basis angepaßte Baum-Kodierer vorgestellt. Für die Daten-Kodierung werden gängige Verfahren wie Huffman, Lempel-Ziv-Welsh, Burrows-Wheeler-Transformation, Run-Length-Encoding und arithmetische Kodierung verglichen.

Diese Arbeit ist dazu wie folgt aufgebaut: Im folgenden Kapitel 2 werden Gebietszerlegungen und dazu duale Bäume vorgestellt. Diese sind als Grundlage für die folgenden Kapitel notwendig. Das Kapitel 3 beschäftigt sich mit möglichen Transformationen T. Die Kompression K wird in Kapitel 4 behandelt. Die notwendigen Fehlerschranken werden in Kapitel 5 diskutiert. Die Kodierung C wird in zwei Kapiteln behandelt. Das Kapitel 6 beschreibt Möglichkeiten der Struktur-Kodierung und Kapitel 7 geht auf die eigentliche

Daten-Kodierung ein. Erweiterungen zu den vorgestellten Verfahren befinden sich in Kapitel 8. Die Ergebnisse dieser Arbeit sind in Kapitel 9 zusammengestellt. Eine Zusammenfassung und ein Ausblick findet sich in Kapitel 10.

2 Gebietsaufteilungen

Zum Verständnis dieser Arbeit sind einige Grundlagen erforderlich. Dies gilt vor allem für die Begriffe Gebietszerlegungen und N-Bäume. Diese werden im nächsten Abschnitt vorgestellt.

2.1 Gebietsaufteilungen

In den folgenden Abschnitten werden immer wieder Gebiete in Teilgebiete aufgeteilt, da diese Teilgebiete meist leichter bearbeitet werden können (*divide and conquer*).

Hier sollen die verschiedenen Raumaufteilungen vorgestellt werden, wobei der Einfachheit halber zunächst von einem Quadrat als Grundfläche ausgegangen wird. In Kapitel 8 wird erläutert, wie diese Einschränkung aufgehoben werden kann.

Von einer Aufteilung wird verlangt, daß die Teilgebiete T_i zusammen das gesamte Gebiet Ω überdecken. Die Teilgebiete selber dürfen sich dabei aber außer an gemeinsamen Ecken oder Kanten nicht überlappen:

$$\begin{aligned} \Omega &= \bigcup_i T_i \\ T_i \cap T_j &= \emptyset \text{ oder gemeinsame Kante oder Ecke} \qquad \text{für } i \neq j \\ \mathcal{T} &= \{T_1, \ldots, T_n\}\,. \end{aligned}$$

Eine Gebietsaufteilung kann direkt gegeben sein oder durch eine Verfeinerung einer gegebenen Aufteilung erreicht werden. Bei einer Verfeinerung wird also ein T_i durch mehrere kleinere Teilgebiete T_{i_j} ersetzt. Für die neu eingesetzten Teilgebiete gelten ebenso die oben aufgeführten Bedingungen, wobei Ω durch T_i ersetzt wird.

Bei den Aufteilungen kann zwischen zwei Gruppen unterschieden werden: den strukturierten und den unstrukturierten Zerlegungen.

Ein einfaches Beispiel für die Gruppe der strukturierten Aufteilungen ist die Überlagerung durch gleich große Quadrate. Diese werden einfach von oben links nach unten rechts in das Ausgangsquadrat gepackt (in Abbildung 1 (a) sind diese Quadrate noch durch jeweils vier Dreiecke geteilt). Die Quadrate könnten auch durch Rechtecke ersetzt werden, deren Kantenlänge durch die Position im Ausgangsgebiet bestimmt wird. Ein Beispiel einer solchen Aufteilung ist in Abbildung 1 (b) zu sehen. Der Aufwand zur Speicherung solcher strukturierten Aufteilungen ist sehr gering.

Desweiteren gibt es auch strukturierte, aber unregelmäßige Aufteilungen wie es in Abbildung 1 (c) dargestellt ist.

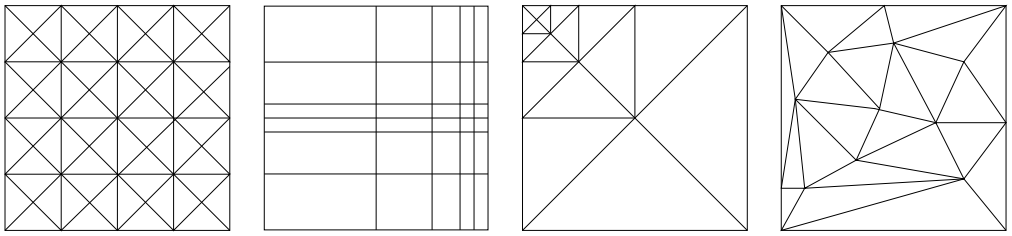


Abbildung 1: Verschiedene Gebietszerlegungen auf Basis von Dreiecken und Rechtecken: gleich große Dreiecke (a), positionsabhängig skalierte Rechtecke (b), rechtwinklige unterschiedlich große Dreiecke (c), unregelmäßige Dreieckszerlegung (d).

Ein extremes Beispiel für unstrukturierte Aufteilungen ist die Überlagerung des Gebietes mit beliebigen Teilgebieten wie es in Abbildung 1 (d) zu sehen ist. Hier ist offensichtlich keine Struktur mehr vorhanden. Entsprechend hoch ist der Aufwand zur Speicherung der Positionen und der Formen der Teilgebiete.

In vorliegender Arbeit wird mit Verfeinerungen gearbeitet. Verfeinerungen werden in vielen Bereichen von der Lösung von partiellen Differentialgleichungen (Rivara 1984) bis zur Visualisierung von großen Datenmengen benutzt (Gross u. a. 1996; Lindstrom u. a. 1996; Ohlberger und Rumpf 1998). Als Teilgebiete sollen nur Rechtecke oder nur Dreiecke verwandt werden. Dabei sollen die Verfeinerungen aus entweder nur Dreiecken oder nur Rechtecken bestehen.

Bei einer Aufteilung $\mathcal{T}$, die nur aus Dreiecken besteht, wird von einer Triangulierung gesprochen. Die hier verwendeten Triangulierungen sollen folgende Bedingungen erfüllen:

zulässig: Der Durchschnitt zweier Dreiecke ist entweder leer, ein gemeinsamer Eckpunkt oder eine gemeinsame Seite.

quasiuniform: Das Verhältnis der Dreiecksgrößen (längste Seite) bleibt beschränkt.

(form-)regulär: Für alle Dreiecke bleibt das Verhältnis Außen- zu Innenkreisradius gleichmäßig beschränkt.

Die Bedingung der Zulässigkeit sorgt dafür, daß keine *hängenden Knoten* auftreten. Ein hängender Knoten ist ein Eckpunkt, der auf einer gemeinsamen

Kante mit einem anderen Dreieck liegt, aber nicht auf einem Eckpunkt des zugehörigen Dreieckes (siehe Abbildung 25 auf Seite 53).

Die Forderungen der Quasiuniformität und der (Form-)Regularität sorgen dafür, daß die Dreiecke der Triangulierung nicht ausarten, also fast auf einer Linie zusammenfallen.

Bei den Rechteck-Verfeinerungen werden von den für die Triangulierungen geltenden Forderungen nur die letzten beiden (Quasiuniformität und (Form-)Regularität) in angepaßter Form übernommen, d.h. die Dreiecke werden durch Rechtecke ersetzt. Dadurch sollen auch hier wieder ausgeartete Rechtecke verhindert werden.

Die Strukturen der Aufteilungen, die durch Verfeinerungen entstehen, besitzen oft einen repräsentierenden N-Baum.

2.2 N-Bäume

In der Informatik werden zur Speicherung von strukturierten Daten häufig Bäume eingesetzt. Bäume erlauben es, Daten schnell zu sortieren und auf die gespeicherten Daten zuzugreifen. Gleichzeitig erleichtern sie es, Strukturen in den Daten zu erhalten oder zu schaffen.

Ein Baum oder auch N-Baum besteht immer aus einem Wurzelknoten (siehe Abbildung 2). Von diesem Wurzelknoten gehen bis zu $N\,(N \in \mathbb{N})$ Äste ab, die auf weitere Knoten verweisen. Jeder dieser Knoten kann wieder bis zu N Äste haben und damit auf bis zu N Knoten verweisen. Diese Knoten heißen Söhne. Der Knoten, von dem auf einen Sohn verwiesen wird, wird auch dessen Vaterknoten genannt (siehe Abbildung 2). Knoten, die keine Söhne haben, heißen Blätter. Unter der Tiefe eines Baumes wird der längste direkte Weg von der Wurzel bis zu einem Blatt verstanden. Dabei entspricht die Länge des Weges der Anzahl der durchlaufenen Äste.

Häufig benutzte N-Bäume sind der Binärbaum, der Ternärbaum und der *quadtree*. Diese Bäume besitzen zwei, drei, respektive vier Äste an jedem Knoten. Die Söhne eines Binärbaumes werden häufig als linker und rechter Sohn bezeichnet. In Abbildung 2 bildet n_2 den linken und n_6 den rechten Sohn von n_1.

Ein Baum kann in verschiedenen Weisen abgearbeitet werden. Häufig verwandte Durchlaufreihenfolgen sind:

inorder: Besuche den linken Teilbaum des Knotens n_i; besuche den Knoten n_i; besuche den rechten Teilbaum von n_i.

preorder: Besuche den Knoten n_i; besuche den linken Teilbaum von n_i; besuche den rechten Teilbaum von n_i.

postorder: Besuche den linken Teilbaum des Knotens n_i; besuche den rechten Teilbaum von n_i; besuche n_i.

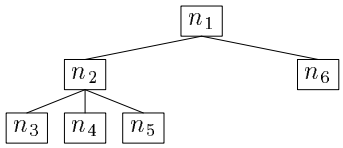


Abbildung 2: N-Baumaufbau. Dabei ist n_1 der Wurzelknoten. n_3, n_4, n_5 und n_6 sind die Blätter dieses Baumes. Alle n_i sind Knoten. Die Äste sind durch Linien dargestellt. n_1 ist der Vaterknoten zu den Söhnen n_2 und n_6.

2.2.1 Verfeinerungen und zugehörige duale Bäume

Die Verfeinerungen durch Drei- und Rechtecke besitzen jeweils duale Bäume. Die Bäume stellen dabei den Weg dar, der von der Ausgangszerlegung $\mathcal{T}_0$ bis zur endgültigen Verfeinerung $\mathcal{T}_n$ zurückgelegt wird. Jeder Knoten in dem entsprechenden Baum repräsentiert eine Ersetzung eines Teilgebietes $T_i \in \mathcal{T}_l$ durch mehrere neue Teilgebiete $T_{i_1}, \ldots, T_{i_k}$. Hierbei wird aus der Zerlegung $\mathcal{T}_l$ die verfeinerte Aufteilung $\mathcal{T}_{l+1}$. Mit den Informationen, daß es sich um eine Triangulierung oder eine Aufteilung durch Rechtecke handelt und wie die Aufteilung der Gebiete von statten geht, kann die Zerlegung aus den Bäumen rekonstruiert werden. Umgekehrt kann auch aus der Aufteilung des Gebietes der Baum mit den entsprechenden Informationen generiert werden. Beispiele von Bäumen und deren Zerlegungen sind in Abbildung 3 zu sehen. Mehr zu Raumaufteilungen und den dazugehörigen Bäumen findet sich in (Samet 1985) und (Samet 1989).

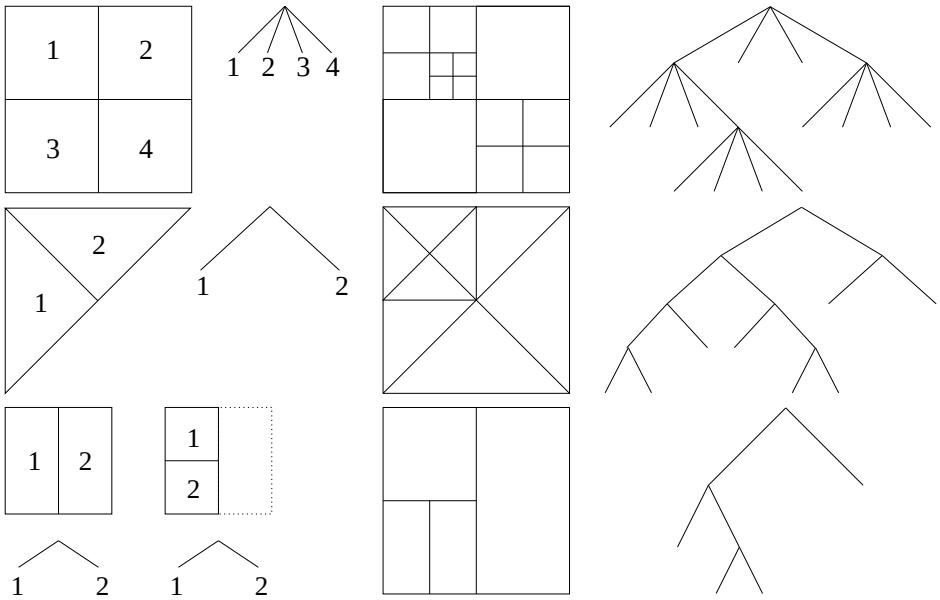


Abbildung 3: Gebietszerlegungen und deren zugehörige Bäume. Von oben nach unten: ein *quadtree* mit Quadraten, Binärbaum mit Dreiecken und Binärbaum mit Rechtecken.

3 Transformationen

Die Transformation ist bei der Datenkompression für die Vorbereitung der Daten zuständig. Sie soll die Ausgangsdaten in eine Darstellung bringen, die die darauffolgenden Operatoren gut bearbeiten können, also die relevanten gut von den irrelevanten Daten trennen und eventuell schon Redundanzen sichtbar machen. Eine weitere Anforderung, die für große Bilder sinnvoll ist, ist die Bereitstellung von unterschiedlich fein aufgelösten Bildern des Ausgangsbildes. So kann zuerst ein Überblick gewonnen werden und dann können bestimmte Stellen genauer dargestellt werden. Oder es kann zuerst eine grobe Version schnell geladen und anschließend die verfeinerte Version nachgeladen werden. Dies sollte in einer Art und Weise geschehen, daß für die feinere Version auf die Daten der gröberen, also bereits vorhandenen Daten zurückgegriffen wird. Diese Vorgehensweise wird progressive Verfeinerung genannt.

3.1 Arten von Transformationen

Für die Transformation gibt es eine Auswahl an Standardtransformationen, von denen hier einige vorgestellt und verglichen werden sollen.

Bei den Transformationen gibt es im großen und ganzen zwei verschiedene Gruppen. Die klassische Gruppe basiert auf einer Frequenzanalyse und enthält die Fourier-, Wavelet- und Hierarchische Basis-Transformation. Die andere Gruppe besteht aus den fraktalen Transformationen.

3.1.1 Fraktale Transformation

Die fraktale Transformation (Fisher 1992; Fisher u. a. 1994; Saupe 1995) ist eigentlich eine Transformation mit integrierter Kompression, d.h. es wird direkt bei der Transformation schon eine Relevanzüberprüfung vorgenommen. Bei der fraktalen Transformation wird die Selbstähnlichkeit von Bildern ausgenutzt. Das Bild $\mathbf{F}$ wird jeweils in Domänen (*domains*, D_i) und Regionen (*regions*, R_i) aufgeteilt. Die Domänen agieren dabei als Bildmaterialspender, die in die Regionen hineinkopiert werden (siehe Abbildung 4). Damit das gesamte Bild wieder rekonstruiert werden kann, müssen die Regionen das gesamte Bild abdecken. Es gilt also $\mathbf{F} = \bigcup_i R_i$. Zusätzlich soll gelten, daß sich die Regionen nicht überlappen, also $R_i \cap R_j = \emptyset$ für $i \neq j$.

Die große Kunst bei der fraktalen Kompression liegt nun darin, eine gute Aufteilung des Bildes $\mathbf{F}$ in Regionen R_i zu finden und für jedes der R_i eine passende Domäne D_{sd_i} zu finden. Die Domäne muß dann skaliert, translatiert, rotiert und gespiegelt die entsprechende Region möglichst gut nachahmen.

Für diese Zwecke kann auch noch eine Helligkeits- und Kontrastanpassung vorgenommen werden.

All diese Transformationen für die Domänen werden in einer Transformation T_{st_i} zusammengefaßt. Es soll also gelten, daß $|R_i - T_{st_i} \cdot R_{sd_i}|$ minimal ist für jede $R_i \in \mathbf{F}$. Ist für jede Region eine Transformation und eine Domäne gefunden, die einen minimalen Fehler bietet, so kann die Rücktransformation, also die Synthese des Bildes, durch Iterieren der Transformationen der Domänen erreicht werden. Die Synthese lautet

$$\tilde{\mathbf{F}} = \lim_{l \to \infty} \bigcup_i T_{st_i} \cdot D_{sd_i},$$

wobei l der Zähler der Iterationen ist. In der Praxis wird dieser Zähler nur wenige Iterationen durchlaufen müssen.

Alle Transformationsanweisungen zusammengenommen ergeben eine fraktale Kopiermaschine, die in die Region die zugehörige Domäne mittels der zuvor bestimmten Transformationen kopiert.

Ein Beispiel einer solchen Kopiermaschine ist in Abbildung 5 zu sehen. Die Transformationen beschränken sich hierbei auf eine Skalierung und Translation. Eine fraktale Kopiermaschine, die auch die anderen oben angesprochenen Transformationen ausnutzt, ist in Abbildung 55 auf Seite 117 zu sehen.

In der Praxis kann natürlich nicht unendlich oft iteriert werden. Hier kann entweder eine feste Anzahl von Iterationen gewählt werden oder aufeinanderfolgende Iterationsergebnisse miteinander verglichen werden. Ist die Differenz klein genug, so wird die Iteration abgebrochen.

Bei den Transformationen ist aber bei der Skalierung darauf zu achten, daß eine Kontraktion stattfinden muß. Sonst kann keine Konvergenz zu einem Fixpunkt garantiert werden. Wenn keine Konvergenz erreicht wird, so ist das Verfahren nicht geeignet, da das Bild mit jedem Iterationsschritt völlig unterschiedlich aussehen kann.

Die Aufteilung von $\mathbf{F}$ in Regionen bzw. Domänen kann auf einfache Weise erfolgen. Das gesamte Bild $\mathbf{F}$ wird mit einem Gitter überdeckt. Die Felder des Gitters entsprechen dann den Regionen. Je feiner das Gitter ist, um so größer ist die Wahrscheinlichkeit, daß alle Feinheiten des Bildes erfaßt werden. Auf der anderen Seite steigt aber auch der Aufwand sowohl zum Komprimieren als auch zum Dekomprimieren stark an.

Für die Domänen des Bildes kann eine ebenso einfache Aufteilung benutzt werden. Wenn die Gitterweite der Domänen größer ist als die der Regionen, so ist zusätzlich auf triviale Art dafür gesorgt, daß eine Kontraktion stattfindet, die somit die Konvergenz des Verfahrens garantiert (siehe hierzu auch Algorithmus 2).

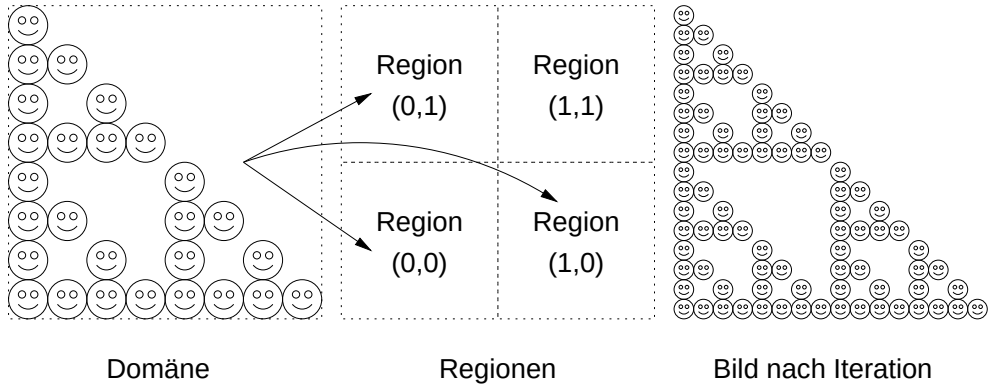


Abbildung 4: Aufteilung eines Bildes in Domänen und Regionen. Für die in Abbildung 5 dargestellte Kopiermaschine besteht das Bild **F** nur aus einer Domäne und vier Regionen. Die Transformation besteht aus einer Skalierung der Domäne auf ein Viertel und kopieren in die Regionen [(0,0),(1,0),(0,1)]. Die letzte noch verbleibende Region wird mit dem skalierten Inhalt der einzigen Domäne gefüllt, wobei der Kontrast minimal und die Helligkeit maximal eingestellt sind.

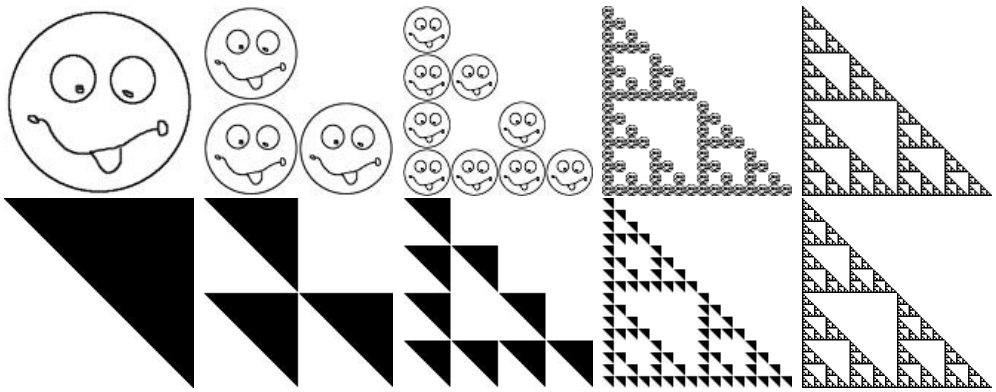


Abbildung 5: Kopiermaschine für das Sierpiński-Dreieck. Das Ausgangsbild wird auf ein Viertel der ursprünglichen Größe verkleinert und in die linke obere, linke untere und untere rechte Ecke kopiert. Dieser Vorgang ist oben für ein *Smiley* und unten für ein *Dreieck* dargestellt. Von links nach rechts sind die Iterationen 0, 1, 2, 4 und 8 zu sehen. Wie zu erkennen ist, ist das Ausgangsbild unerheblich für das dekomprimierte Endbild.

Algorithmus 1: Fraktale Dekompression

F =Irgendein Bild $\neq \emptyset$ $\{\mathbf{F} = \cup_i R_i\ \}$
for $l = 0$ to maxit **do**
  for $R_i \in \mathbf{F}$ **do**
    $R_i = T_{st_i} \cdot D_{sd_i}$
  end for
end for

Algorithmus 2: Einfache Fraktale Kompression

O =Original Bild $\{\mathbf{F} = \cup_i R_i\}$
for $R_i \in \mathbf{F}$ **do**
  m = maximaler Wert
  for $D_j \in \mathbf{F}$ **do**
    for $T_k \in$ mögliche Transformationen **do**
      $d = |R_i - T_k \cdot D_j|$
      if $d < m$ **then**
        $sd_i = j, st_i = k$
      end if
    end for
  end for
end for
Speichere alle $R_i \in \mathbf{R}$ und dazugehörige T_{st_i}, D_{sd_i}

Der Aufwand für die fraktale Kompression hängt von der Menge der gewählten Domänen ab. Wenn N die Anzahl der zur Verfügung stehenden Domänen ist, so kann der Aufwand zur Suche der besten Domäne für eine Region auf den Aufwand von $\mathcal{O}(\log N)$ beschränkt werden (Saupe 1995). Bei einem $n \times n$-Pixel großen Bild gibt es n^3 verschiedene Möglichkeiten Domänen aus dem Bild zu wählen. Selbst wenn nur n^2 verschiedene Domänen ausgewählt werden, so ergibt sich ein Gesamtaufwand von $\mathcal{O}(n^2 \log^2 n)$.

Ein gravierender Nachteil der oben vorgestellten Methode ist, daß lokale Eigenschaften des Bildes hierbei nicht berücksichtigt werden. Das bedeutet, daß auf der einen Seite feine Strukturen des Bildes nur schlecht aufgelöst werden können und auf der anderen Seite für Gebiete mit wenigen Details ein unnötig großer Aufwand getrieben wird.

Um das zu ändern, kann eine adaptive Auswahl der Größe der Regionen erfolgen. Dafür kann z. B. ein *quadtree* benutzt werden. Falls für eine Region keine passende Domäne gefunden wird, so kann die Region in vier kleine Regionen aufgeteilt werden. Die neuen Regionen entsprechen dann neuen Blättern in dem *quadtree*. Als Domänen für die Regionen können z. B. die Teilgebiete von **F** genommen werden, die eine doppelt so lange Kantenlänge haben wie die aktuell untersuchte Region.

Statt eines *quadtree*, bei dem normalerweise die Ausgangsregionen in gleich große Gebiete aufgeteilt werden, kann auch die Aufteilung der Gebiete adaptiv an das Bild angepaßt werden. Auch kann statt einer Aufteilung in Vierecke eine Triangulierung des Gebietes vorgenommen werden.

In (Davis 1997) wird darauf hingewiesen, daß zwischen den hier vorgestellten blockorientierten fraktalen Kodierern und den in Abschnitt 3.1.3 vorgestellten Wavelets deutliche Zusammenhänge bestehen.

3.1.2 Fouriertransformation

Die Fouriertransformation stellt die vorgegebenen Daten aus Sinus- und Cosinus-Funktionen mit unterschiedlicher Frequenz und Amplitude dar. Daten, die durch eine Sinuskurve repräsentiert werden, können so durch einen einzigen Wert dargestellt werden (siehe hierzu Abbildung 6).

Bei Musikdaten, die hauptsächlich aus periodischen, sinusartigen Schwingungen bestehen, funktioniert die Trennung der relevanten von den irrelevanten Daten durch die Fouriertransformation relativ gut. Die Fouriertransformation ist eigentlich auf kontinuierliche Daten ausgelegt. Sie lautet:

$$\hat{f}(\omega) = \sqrt{\frac{1}{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt.$$

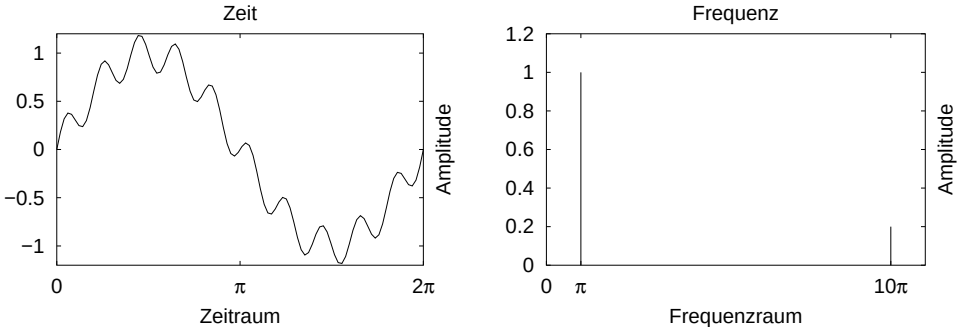


Abbildung 6: Zeit- und Frequenzraum für die Funktion $\sin(x)+0,2\cdot\sin(10x)$. Die sinusartige Kurve links kann im Frequenzraum mit der Sinustransformation durch zwei Werte dargestellt werden.

Für unsere diskreten Daten müssen wir auf die diskrete Form der Fouriertransformation zurückgreifen:

$$\begin{aligned} j &= 0,1,\ldots,N-1 \\ N &= 2^n, n \in \mathbb{N} \\ \hat{\mathbf{f}}_j &= \frac{1}{N}\sum_{k=0}^{N-1} \mathbf{f}_k e^{-2\pi i \frac{jk}{N}}. \end{aligned}$$

Diese diskrete Formel erfordert einen unvertretbar hohen Aufwand. Für Daten der Länge 2^n läßt sich unter Ausnutzung von Symmetrien der Basisfunktionen die *fast fourier transformation* (fft) ableiten. Diese hat nur noch einen Aufwand von $\mathcal{O}(n \log n)$. Die Einschränkung auf Datenvektoren der Länge 2^n spielt bei den meisten Anwendungen keine Rolle.

Die größere Einschränkung der Fouriertransformation ist eher die schlechte Lokalisierbarkeit der Frequenz im Zeitraum (siehe Abbildung 7), da die Basisfunktionen keinen lokalen, sondern einen globalen Träger besitzen.

Ist eine solche Lokalisierung aber erwünscht, so muß eine Änderung an der Fouriertransformation vorgenommen werden. Dies kann z. B. durch eine Beschränkung der Basisfunktionen auf bestimmte Zeitabschnitte geschehen. Dies ist dann die gefensterte Fouriertransformation. Bei dieser wird der vorgegebene Datenvektor in kleinere Teile zerteilt, diese kleinen Teile werden einzeln transformiert und bearbeitet. Dabei kann die Art des Fensters beliebig variiert werden, z. B. durch ein hartes Abschneiden oder ein weiches Einblenden der Basisfunktionen wie in Abbildung 8 dargestellt.

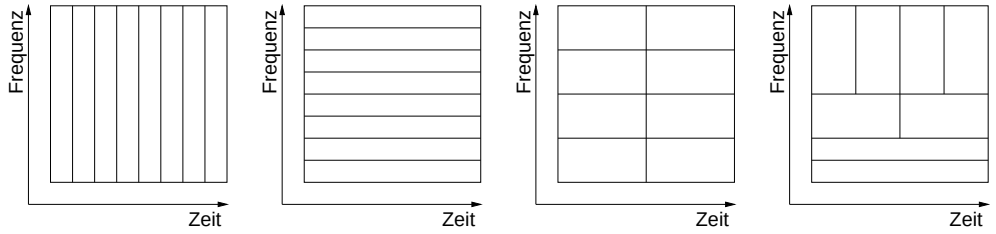


Abbildung 7: Unterteilung des Zeit- Frequenzraumes für verschiedene Transformationsarten. Von links nach rechts: ohne Transformation, Fouriertransformation, gefensterte Fouriertransformation und Wavelettransformation.

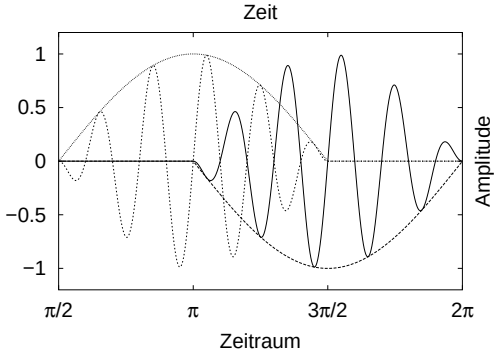


Abbildung 8: Basisfunktionen und einhüllende Fensterfunktionen für eine gefensterte Fouriertransformation.

Wenn auf die Phase der Frequenzen kein großer Wert gelegt wird, so kann statt der normalen Fouriertransformation auch eine Cosinus- oder Sinustransformation benutzt werden. Dies hat den Vorteil, daß statt der komplexen Koeffizienten, die bei der Fouriertransformation entstehen, reale Koeffizienten benutzt werden können. Die diskrete gefensterte Cosinustransformation wird z. B. bei der Bildkompression nach dem JPEG-Standard benutzt.

3.1.3 Wavelettransformation

Die gefensterte Fouriertransformation erlaubt die gleichzeitige Auflösung von Frequenz und Zeit oder Ort bei Funktionen. Die Güte der Auflösung ist jedoch beschränkt durch die Breite des benutzten Fensters. Tiefe Frequenzen brauchen aber ein großes Fenster, wohingegen hohe Frequenzen mit einem kleinen Fenster auskommen (siehe Abbildung 7). Die Größe des gewählten Fensters sollte möglichst an die zu beobachtende Frequenz angepaßt sein. Was ist aber, wenn diese Frequenz zu Beginn nicht bekannt ist? Die gefensterte

Fouriertransformation müßte eventuell mehrere Male durchgeführt werden. Aber das triebe den Aufwand erheblich in die Höhe.

Um dies zu vermeiden, kann auf Basen mit beschränktem (kompaktem) Träger zurückgegriffen werden. Günstig ist es, wenn die Größe des Trägers von der Frequenz der Basisfunktion abhängt. Wavelets haben genau diese Eigenschaft. Sie bieten

- Multiskalen-Repräsentation und
- Lokalitätserhaltung.

Die Multiskalen-Repräsentation ist eine effiziente Art, einen Satz von Daten in verschiedenen Verfeinerungsstufen darzustellen und bearbeiten zu können. Dies ist z. B. bei geophysikalischen Daten wichtig, bei denen nur Strukturen bestimmter Größenordnungen von Interesse sind. Bei der Betrachtung eines Flußverlaufes ist es vielleicht nicht nötig, jede kleine Unebenheit darzustellen oder zu berücksichtigen, die durch Kieselsteine am Flußufer verursacht wird. Bei einer anderen Anwendung des gleichen Datensatzes können aber diese Strukturen sehr wohl eine Rolle spielen. Hier wäre es also von Vorteil, wenn die Daten beide Größenordnungen oder besser noch beliebig viele verschiedene Verfeinerungsstufen anbieten können.

Lokalitätserhaltung bedeutet, daß lokale Strukturen in den Daten auch in den transformierten Daten nur lokale Strukturen zur Folge haben. Der Kieselstein aus obigen Beispiel soll in der transformierten Darstellung nicht die Dimension eines Flußbettes erlangen.

Wavelets zeichnen sich dadurch aus, daß ihre Basis-Funktionen durch Translation und Skalierung eines Mutter-Wavelets dargestellt werden können. Wenn zur Vereinfachung der Schreibweise für diese translatierten und skalierten Basisversionen

$$\psi_{l,i}(x) := 2^{l/2}\psi(2^l x - i), \quad l, i \in \mathbb{Z}$$

definiert wird, so kann f als Doppelsumme solcher Funktionen geschrieben werden:

$$f(x) = \sum_{l=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} d_{l,i}\psi_{l,i}(x). \tag{1}$$

An dieser Summe ist genau zu sehen, daß wirklich eine Multiskalen-Repräsentation erreicht ist. Die innere Summe verfolgt die Basen einer Frequenzauflösung über den gesamten Ort x. Die äußere Summe hingegen sorgt für die Skalierung der Wavelets, also deren Frequenz und Ausdehnung.

Sollen nur bestimmte Frequenzen beobachtet werden, so kann die äußere Summe auf diese Frequenzen beschränkt werden. Die Funktion $\tilde{f}$ als Approximation von f kann durch Einschränkung der Laufweite von l erreicht werden.

Auf diese Weise können Frequenzfilter einfach realisiert werden. Im Gegensatz zur Fouriertransformation können diese Bandfilter sehr leicht ortsabhängig variiert werden.

Ein einfaches und bekanntes Beispiel eines Wavelets ist das Haar-Wavelet (siehe Abbildung 9 links). Dieses Wavelet ist stückweise konstant und lautet:

$$\psi(x) = \begin{cases} 1, & x \in [0, 1/2) \\ -1, & x \in [1/2, 1) \\ 0, & \text{sonst} \end{cases} .$$

Ein anderes Beispiel für ein Wavelet ist der in Abbildung 9 rechts dargestellte Mexikanische Hut (Kumar und Foufoula-Georgiou 1997).

Wie für die Fouriertransformation gibt es auch für die Wavelettransformation eine schnelle diskrete Variante.

Die schnelle diskrete Wavelettransformation spaltet den Raum V_1, auf dem die Funktionsdaten $\mathbf{f}$ gegeben sind, in zwei orthogonale Räume V_0 und W_0 auf. In diesen beiden Räumen liegen die Vektoren $\mathbf{g}$ und $\mathbf{h}$. Es gilt:

$$\begin{aligned} V_1 &= V_0 \oplus W_0 \\ \mathbf{f} &= \mathbf{g} + \mathbf{h}. \end{aligned}$$

Im Fall des Haar-Wavelets kann $\mathbf{g}$ als die Mittelwerte und $\mathbf{h}$ als die Differenz zu den Mittelwerten aufgefaßt werden. Dies entspricht auch einem Hochpaß und einem Tiefpaß, der zu $\mathbf{h}$ respektive $\mathbf{g}$ führt (siehe hierzu auch Abbildung 10).

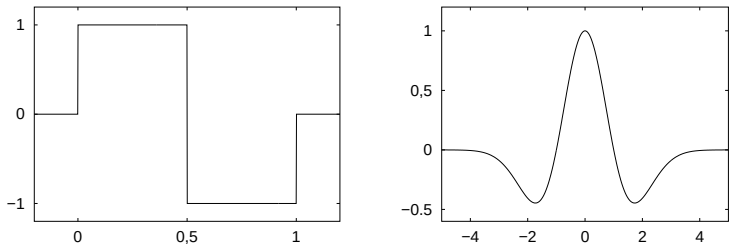


Abbildung 9: Das Haar-Wavelet (links) und die sogenannte Mexikanische Hut-Funktion ($f(x) = (1 - x^2)e^{-x^2/2}$) (rechts).

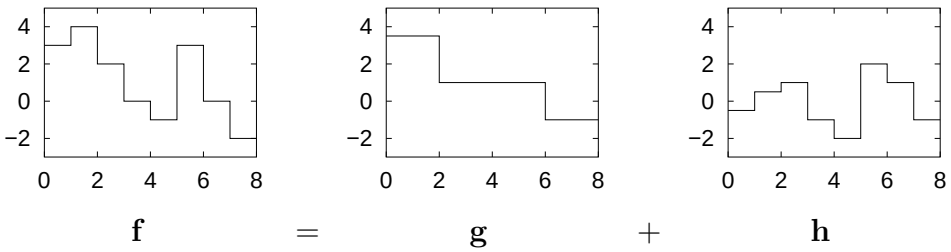


Abbildung 10: Zerlegung des Datenvektors **f** in **g** und **h**. Von links nach rechts: $\mathbf{f} \in V_1$ (Original), $\mathbf{g} \in V_0$ (Tiefpaß) und $\mathbf{h} \in W_0$ (Hochpaß).

Sowohl **f** als auch **g** können durch eine Skalierungsfunktion dargestellt werden, wobei **f** auf einem feineren Level ($l = 0$) als **g** ($l = 1$) lebt. Die Skalierungsfunktion $\varphi(x)$ ist Eins für $[0, 1)$ und sonst Null. Wenn für die skalierten und translatierten Versionen von φ gilt

$$\varphi_{l,i}(x) = 2^{l/2}\varphi(2^l x - i) \qquad l, i \in \mathbb{Z},$$

so können **f** und **g** als Summe solcher Funktionen geschrieben werden. Also $f(x) = \sum_{i\in\mathbb{Z}} f_i\varphi_{0,i}(x)$ und $g(x) = \sum_{i\in\mathbb{Z}} g_i\varphi_{1,i}(x)$. Wenn für φ auch die Verfeinerungsgleichung

$$\varphi_{l,i}(x) = \sum_{m\in\mathbb{Z}} a_{m-2i}\varphi_{l+1,m}(x) \qquad l, i \in \mathbb{Z} \tag{2}$$

erfüllt ist, können aus der charakteristischen Funktion $\varphi(x)$ mit einfachen Mitteln alle Basen der Räume V_j berechnet werden. Für das Haar-Wavelet sind die Koeffizienten a_0 und a_1 gleich $1/\sqrt{2}$, alle anderen Koeffizienten sind gleich 0. Der Koeffizientenvektor **a** wird auch die Maske von φ genannt.

Die Räume V_j bestehen aus Translationen von $\varphi_{l,i}$:

$$V_l = \left\{ v \in L_2(\mathbb{R}) : v(x) = \sum_{i\in\mathbb{Z}} c_i\varphi_{l,i}(x) \right\}.$$

Wegen der Verfeinerungsgleichung sind die Räume V_l ineinandergeschachtelt, d.h. $V_l \subseteq V_{l+1}$. Insgesamt gilt sogar, daß die Menge der $\{V_l\}_{l\in\mathbb{Z}}$ dicht in $L_2(\mathbb{R})$ liegt, es gilt:

$$\ldots \subseteq V_{-2} \subseteq V_{-1} \subseteq V_0 \subseteq V_1 \subseteq V_2 \subseteq \ldots \subseteq L_2(\mathbb{R}).$$

Die Menge $\{V_l\}_{l\in\mathbb{Z}}$ wird auch *multiresolution analysis* von $L_2(\mathbb{R})$ genannt.

Da $V_l \subseteq V_{l+1}$ ist, gibt es einen zu V_l orthogonalen Raum W_l, der V_l zu V_{l+1} komplettiert. Auf dem Raum W_l lebt der oben schon erwähnte Vektor $\mathbf{h}$. Da aber V_l wieder in $V_{l-1} \oplus W_{l-1}$ aufgeteilt werden kann, gilt insgesamt für festes $L > 0$:

$$V_L = V_0 \bigoplus_{l=0}^{L-1} W_l.$$

Da $W_l \subseteq V_{l+1}$ ist, kann auch das in W_l lebende Wavelet $\psi_{l,i}$ als Linearkombination von Basisfunktionen aus dem Raum V_{l+1} geschrieben werden. Es gilt also:

$$\psi_{l,i}(x) = \sum_{m \in \mathbb{Z}} b_{m-2i} \varphi_{l+1,m}(x) \qquad l, i \in \mathbb{Z}. \tag{3}$$

Analog zu der Skalierungsfunktion φ heißt der Vektor $\mathbf{b}$ Maske des Wavelets ψ.

Falls die Skalierungsfunktion $\varphi_{j,k}$ orthogonal auf jedem Level j ist, so läßt sich zeigen, daß die Einträge von $\mathbf{b}$ aus dem Datenvektor $\mathbf{a}$ berechnet werden können. Die Formel hierzu lautet:

$$b_m = (-1)^{m-1} a_{1-m} \quad m \in \mathbb{Z}.$$

Für die oben angeführte Skalierungsfunktion des Haar-Wavelets ergeben sich somit für die Maske von ψ die Werte $b_0 = 1/\sqrt{2}$ und $b_1 = -1/\sqrt{2}$; alle anderen Werte von $\mathbf{b}$ verschwinden.

Wie oben erwähnt, kann für eine gewählte feinste Auflösung ein $L \in \mathbb{N}$ gewählt werden, bei der für V_L gilt: $V_L = V_0 \bigoplus_{l=0}^{L-1} W_l$.

Der Raum V_L kann entweder durch die Skalierungsfunktionen $\varphi_{l,i}(x)$ und c_i oder durch die Wavelets $\psi_{l,i}(x)$ und d_i^l dargestellt werden. Es gilt also:

$$v(x) = \sum_{i \in \mathbb{N}} c_i \varphi_{L,i}(x) \text{ und} \tag{4}$$

$$v(x) = \sum_{l=0}^{L-1} \sum_{i \in \mathbb{N}} d_i^l \psi_{l,i}(x). \tag{5}$$

Die Koeffizienten c_i und d_i^l können als Vektoren $\mathbf{c}$ und $\mathbf{d} = (d^0, d^1, \ldots, d^{L-1})^T$ geschrieben werden. Gesucht ist eine lineare Transformation, die es ermöglicht, schnell und einfach von $\mathbf{c}$ nach $\mathbf{d}$ und zurück zu gelangen, also

$$T_L : \mathbf{d} \to \mathbf{c}, \quad T_L^{-1} : \mathbf{c} \to \mathbf{d}.$$

Um dieses Ziel zu erreichen, werden die $\varphi_{l,i}(x)$ zu einem Spaltenvektor $\Phi_l(x)$ zusammengefaßt.

$$\Phi_l(x) = (\ldots, \varphi_{l,i-1}(x), \varphi_{l,i}(x), \varphi_{l,i+1}(x), \ldots)^T$$

Das gleiche wird mit $\Psi_l(x)$ gemacht. Dann können die Gleichungen 2 und 3 in Matrixschreibweise formuliert werden:

$$\Phi_l(x) = M_{l,0}^T \Phi_{l+1}(x) \text{ und} \quad (6)$$

$$\Psi_l(x) = M_{l,1}^T \Phi_{l+1}(x), \quad (7)$$

wobei die Matrizen $M_{l,0}$ und $M_{l,1}$ aus den Masken $\mathbf{a}$ und $\mathbf{b}$ geformt werden. Für die Matrizen gilt:

$$(M_{l,0})_{i,m} = a_{m-2i}, \quad (M_{l,1})_{i,m} = b_{m-2i}.$$

Wenn nun Gleichung 5 in Matrizenschreibweise dargestellt wird, so gilt:

$$v = (d^0)^T \Psi_0 + (d^1)^T \Psi_1 + \ldots + (d^{L-1})^T \Psi_{L-1}$$

und unter Ausnutzung der Gleichungen 6 und 7 kann der Vektor v als

$$v = \mathbf{c}^T \Phi_L$$

geschrieben werden. Damit ist aber auch die Transformation

$$T_L = T_{L,L-1} \cdot \ldots \cdot T_{L,0}$$

gefunden. Hierbei haben die $T_{L,l}$ die Form

$$T_{L,l} = \begin{pmatrix} M_l & 0 \\ 0 & I \end{pmatrix}.$$

Die M_l sind dabei aus den beiden *Hälften* von $M_{l,0}$ und $M_{l,1}$ zusammengesetzt und sind quadratische Matrizen. I ist die Identität. T_L wird die (diskrete) Wavelettransformation genannt. Sie kann mittels T_L in $\mathcal{O}(N)$ berechnet werden, wenn die Matrizen $M_{l,0}$ und $M_{l,1}$ eine feste endliche Anzahl von Einträgen besitzen. N ist hierbei die Anzahl der Unbekannten auf dem Level L. Wenn die Masken $\mathbf{a}$ und $\mathbf{b}$ endlich viele Koeffizienten besitzen, so sind die Bedingungen für die Komplexität erfüllt. Die auf T_L basierende Wavelettransformation wird deshalb auch schnelle Wavelettransformation (*fast wavelet transformation* (fwt)) genannt.

Wenn orthogonale Wavelets, wie das Haar-Wavelet, verwandt werden, so ist die Rücktransformation $T_L^{-1} = T_L^T$ und hat damit die gleiche Komplexität wie die Transformation.

Aber auch wenn auf die Orthogonalität verzichtet wird, kann die Transformation auf ein Pyramiden-Schema (Dahmen 1997) zurückgeführt werden und behält so die gleiche Komplexität bei.

3.1.4 Hierarchische Finite Elemente

Ein Spezialfall der oben vorgestellten Wavelets ist die sogenannte Hierarchische Basis (Faber 1909). Diese bietet keine verschwindenden Momente und keine Orthogonalität der Basen. Immerhin sind die hierarchischen Finiten Elemente biorthogonal. Das Fehlen der Eigenschaften stört bei den in dieser Arbeit behandelten Problemen aber nicht, da diese in der Regel die Glattheitsbedingungen, die an Wavelets höherer Ordnung gestellt werden, nicht erfüllen.

Die hierarchischen Finiten Elemente sind stückweise linear und stetig. Dadurch können Bilddaten für das menschliche Auge angenehmer approximiert werden als mit den stückweise konstanten Haar-Wavelets.

Im Gegensatz zu Wavelets höherer Ordnung treten bei den hierarchischen Finiten Elementen keine Probleme bei der Randbehandlung auf. Sie ist einfach zu implementieren und behält die sehr gute Komplexität von $\mathcal{O}(N)$.

Wenn für den Gesamt-Level $L \in \mathbb{N}$, den Level-Index $1 \leq l \leq L$ und den Orts-Index $1 \leq i \leq 2^{l-1}$ gilt, kann die von den Wavelets schon bekannten Funktionen $\psi_{l,i}$ für die Hierarchische Basis als

$$\psi_{l,i}(x_{j,k}) = \begin{cases} \delta_{ij} & \text{, für } k = l \\ 0 & \text{, für } k < l \end{cases}$$

definiert werden und die Funktion $f(x)$ in Anlehnung an Gleichung 1 aus Abschnitt 3.1.3 als

$$f(x) = \sum_{l=1}^{L} \sum_{i=1}^{2^{l-1}} d_{l,i} \cdot \psi_{l,i}(x)$$

geschrieben werden.

Wird als Basis für die ψ_{li} die hierarchischen linearen Splinefunktionen

$$\psi_{li}(x - x_{li}) := \max(1 - 2^l |x - x_{li}|, 0)$$

eingesetzt, können die Koeffizienten d_{li} einfach mit der Maske $[-1/2\, 1\, -1/2]$ mittels Vektormultiplikation durch

$$\begin{aligned} d_{l,i} &= \left[-\frac{1}{2}\, 1\, -\frac{1}{2}\right]_{x_{l,i}, h_{l,i}} \mathbf{f} \\ &= -\frac{1}{2} f(x_{l,i} - h_{l,i}) + f(x_{l,i}) - \frac{1}{2}(x_{l,i} + h_{l,i}) \\ &= f(x_{l,i}) - \frac{1}{2}(f(x_{l,i} - h_{l,i}) + f(x_{l,i} + h_{l,i})), \end{aligned}$$

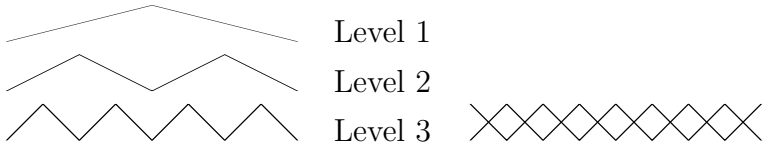


Abbildung 11: Links: Hierarchische Basis. Die Basiselemente sind skalierte und translatierte Versionen der Basis auf Level 1; Rechts: nodale Basis. Alle Basen sind gleichwertig, es gibt nur einen Level.

errechnet werden. Die Basisfunktionen für $L = 3$ sind in Abbildung 11 gezeigt.

Die Koeffizienten d_{li} können als die Differenzen zu den linearen Interpolanden zwischen den beiden Stützstellen des nächstgröberen Levels aufgefaßt werden (siehe Abbildung 12). Deshalb werden die Koeffizienten auch *hierarchischer Überschuß* genannt. Da alle Basis-Funktionen stückweise linear sind, ist auch die interpolierende Funktion ebenfalls stückweise linear.

Bei den hier untersuchten Fällen kann das Problem auf den diskreten Fall beschränkt werden. Es interessieren also nur die Werte an den ursprünglichen Stützstellen. Das Problem kann im diskreten Fall, wie schon bei den Wavelets vorgestellt, auch in Matrixschreibweise formuliert werden:

$$T\mathbf{f} = \hat{\mathbf{f}}$$

wobei T die Transformationsmatrix, $\mathbf{f}$ der ursprüngliche Datenvektor und $\hat{\mathbf{f}}$ der in die Hierarchische Basis transformierte Datenvektor ist. T ist eine untere Dreiecksmatrix. Die Rücktransformation ist einfach T^{-1} und kann in diesem Fall direkt angegeben werden. Die Matrizen sehen dabei für $L = 3$ wie in Abbildung 13 aus.

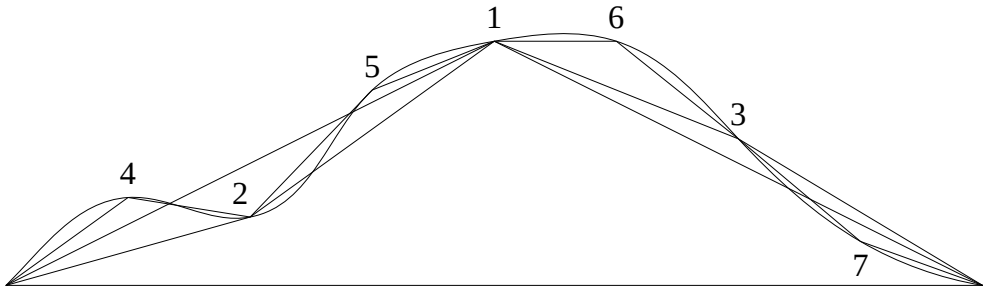


Abbildung 12: Sortierung der Hierarchischen Basisfunktionen.

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1/4 & 1/2 & 0 & 1 & 0 & 0 & 0 \\ 3/4 & 1/2 & 0 & 0 & 1 & 0 & 0 \\ 3/4 & 0 & 1/2 & 0 & 0 & 1 & 0 \\ 1/4 & 0 & 1/2 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1/2 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1/2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1/2 & 0 & 1 & 0 & 0 & 0 \\ -1/2 & -1/2 & 0 & 0 & 1 & 0 & 0 \\ -1/2 & 0 & -1/2 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1/2 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Abbildung 13: Matrizen T und T^{-1} für die Hierarchische Basis.

3.2 Übergang zu höheren Dimensionen

Bisher sind – bis auf die fraktale – die Transformationen T nur eindimensional vorgestellt worden. Bilder und andere bildähnliche Daten, wie DHM oder MRT, verlangen aber höherdimensionale Transformationen. Hier sollen zwei Wege vorgestellt werden, wie die oben vorgestellten Verfahren auf 2-D zu übertragen sind.

3.2.1 Tensorprodukt-Ansatz

Beim Tensorprodukt-Ansatz wird die eindimensionale Transformation hintereinander auf die Spalten und Zeilen der Datenmatrix angewandt. Für die Hierarchische Basis wird dann aus der Maske für 1-D:

$$\begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \end{bmatrix}_{x_{li}, y_{li}, h^x_{li}, h^y_{li}} = \begin{bmatrix} -\frac{1}{2} \\ 1 \\ -\frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}.$$

Einige Basisfunktionen der Hierarchischen Basis im Tensorprodukt-Ansatz sind in Abbildung 14 abgebildet. Zur Veranschaulichung wird häufig eine Umsortierung der Koeffizienten vorgenommen. So etwa für das Haar-Wavelet in h- und l-Koeffizienten (*highpass*, *lowpass*). Die vollständige Transformation

in die tensorierte *multiresolution analysis* für das Haar-Wavelet zeigt Abbildung 15. Ein Beispiel für einen Schritt der Zeilen- und Spaltentransformation für die Hierarchische Basis liefert Abbildung 16.

Der Tensorprodukt-Ansatz wird auch beim JPEG-Verfahren für die DCT angewandt, wobei diese dann noch auf 8×8-Pixel große Blöcke angewandt wird.

Die Erweiterung auf n-D läßt sich durch Tensorieren leicht bewerkstelligen. Die eindimensionale Transformation wird nacheinander in allen Richtungen auf die Daten angewandt. Dieser Tensorprodukt-Ansatz ist der Normalfall, wenn höhere Dimensionen bearbeitet werden sollen. Standardkompressions-Verfahren wie JPEG, SPIHT und EZW arbeiten alle mit dem Tensorprodukt-Ansatz.

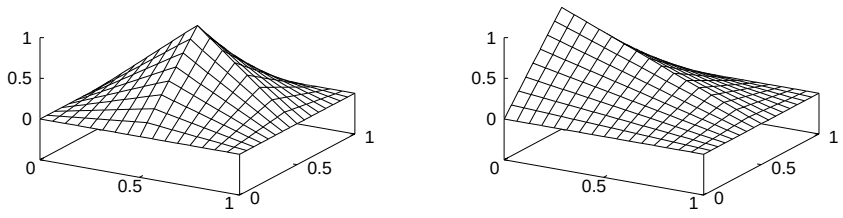


Abbildung 14: Einige Basisfunktionen für den Tensorprodukt-Ansatz für die Hierarchische Basis in 2-D.

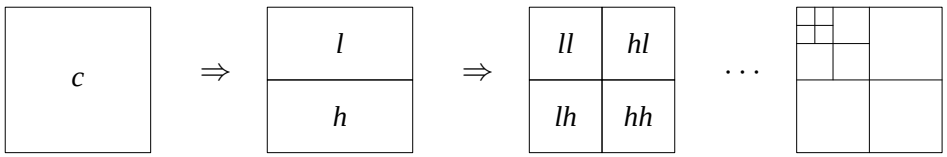


Abbildung 15: *Multiresolution* Darstellung von 2-D Daten.

Abbildung 16: Größenordnung der Koeffizienten der Hierarchischen Basis im Tensorprodukt-Ansatz. Der Nullpunkt der Koeffizienten wurde zur Darstellung verschoben, so daß schwarz den größten und weiß den kleinsten Wert eines Koeffizienten darstellt.

3.2.2 Triangulierung

Ein anderer Ansatz, um zu höheren Dimensionen zu gelangen, ist die Suche nach Basisfunktionen in diesen höheren Dimensionen. Für die Hierarchische Basis kann hierfür auf die zweidimensionalen linearen Hütchenfunktionen zurückgegriffen werden, wie sie auch bei der Finiten Elemente Methode eingesetzt werden (Gerstner 1995).

Die transformierten Koeffizienten lassen sich recht einfach durch einen rekursiven Algorithmus (siehe Algorithmus 3) errechnen. Ausgehend von einem großen rechtwinkligen und gleichschenkligen Dreieck wird dieses in zwei gleich große Dreiecke entlang der Winkelhalbierenden des rechten Winkels geteilt (siehe Abbildung 17). Der Koeffizient liegt dann auf der Mitte der Hypotenuse des großen Dreieckes und wird unter Einhaltung der Numerierung aus Abbildung 17 folgendermaßen berechnet:

$$\hat{f}(x_n) = \frac{f(x_2) + f(x_3)}{2} - f(x_n).$$

Die beiden neuen Dreiecke werden auf die gleiche Weise rekursiv weiter aufgeteilt, bis der feinste Level erreicht ist.

Algorithmus 3: recTrafo(x_1, x_2, x_3,level)

1: **if** level≤endLevel **then**
2: $x_n = \frac{x_2+x_3}{2}$ {Zur Numerierung der Eckpunkte siehe Abbildung 17}
3: recTrafo(x_n, x_1, x_2,level+1)
4: recTrafo(x_n, x_3, x_1,level+1)
5: $\hat{f}[x_n] = \frac{f[x_2]+f[x_3]}{2} - f[x_n]$
6: **end if**

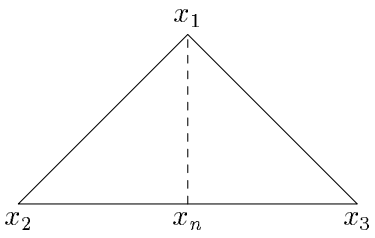


Abbildung 17: Aufteilung und Numerierung der Dreiecke der hierarchischen Triangulierung.

3.3 Vergleich der Transformationen

Bei einem Vergleich der vorgestellten Transformationen können die Verfahren in zwei Gruppen eingeteilt werden; die Gruppe der fraktalen Transformationen und die der Frequenz-Transformationen.

Bei der fraktalen Kompression sollen Kompressionsraten von 10 000:1 erreicht worden sein (Saupe und Hamzaoui 1994; Barnsley und Sloan 1987); allerdings handelte es sich dabei um sehr speziell ausgesuchte Daten, und die Transformation wurde von Hand nachgebessert (als Beispiel könnte ein Bild eines Sierpiński-Dreieckes, wie es in den Abbildungen 4 und 5 gezeigt ist, dienen). Der Aufwand zur Suche der Fixpunktabbildung ist auf jeden Fall ein Argument gegen die Verwendung von Fraktalen. Für die Transformation der *Mona Lisa* aus Abbildung 55 auf Seite 117 rechnete ein Pentium II mit 266 MHz mit dem oben vorgestellten einfachen adaptiven Algorithmus etwa eine Stunde. Ein anderer Punkt ist die Erzeugung von Strukturen, die im Ausgangsmaterial nicht vorhanden sind.

Die andere Gruppe besteht aus Fourier-, Wavelet- und hierarchische Finite Elemente-Transformation. Gegen die Fouriertransformation spricht, daß die Eingangsdaten periodisch sein müssen. Sind sie es nicht, so müssen diese erst durch Spiegelung an den Enden oder Ähnlichem periodisiert werden. Desweiteren ist die Komplexität mit $\mathcal{O}(N \log N)$ höher als die der beiden anderen Verfahren und die Errechnung der Basisfunktionen recht aufwendig.

Bleiben noch die Wavelet- und die hierarchische Finite Elemente-Transformationen. Die Vorteile, die die Wavelets durch ihre Orthogonalität und der verschwindenden Elemente auf Daten mit bestimmten Eigenschaften haben, sind bei Bildern nicht ausschlaggebend, da diese i. allg. keine Glattheitsbedingungen erfüllen. Die Komplexität der beiden Transformationen ist dank des Pyramiden-Schemas mit $\mathcal{O}(N)$ dieselbe.

Bei der Hierarchischen Basis mit stückweise linearen Spline-Funktionen als Basis ist keine besondere Randbehandlung notwendig, was der einfachen Implementierung zugute kommt. Ein weiteres Argument für die hierarchische Finite Elemente-Transformation ist die einfache Möglichkeit der Fehlerkontrolle in der Maximumsnorm.

Welche Erweiterung auf höhere Dimensionen gewählt wird, hängt von der weiteren Verarbeitung ab. Ein Vergleich in Hinsicht auf Kompression und Fehlerbeschränkung wird deswegen in den entsprechenden Abschnitten vorgenommen.

4 Kompressionsverfahren

Nachdem die Daten $\mathbf{f}$ durch die Transformation T verarbeitet worden sind, soll die Kompression K für die Auswahl der relevanten Informationen aus $\hat{\mathbf{f}} = T\mathbf{f}$ sorgen.

Diese Auswahl erzeugt i. allg. einen Fehler, da Informationen vernichtet werden, die nicht unbedingt redundant sind. Der Operator K ist also nicht invertierbar. Die Pseudo-Inverse K_*^{-1} soll möglichst sinnvolle Werte für die Näherung $\tilde{\mathbf{f}} = T^{-1}K_*^{-1}K\hat{\mathbf{f}}$ ergeben. Denn nur so können die komprimierten Daten später wiederverwendet werden.

4.1 Arten von Kompressionsverfahren

Hier sollen nur zwei Kompressionsverfahren vorgestellt werden. Diese beiden Verfahren ergänzen sich bei der Reduzierung der irrelevanten Informationen. Bei einem Vergleich ist also nicht die Frage nach dem besseren Verfahren zu stellen, sondern es sind Möglichkeiten festzustellen, beide Kompressionsarten sinnvoll zu ergänzen.

4.1.1 Thresholding

Das *thresholding* ist eine einfache Art die relevanten, also die wichtigen Daten von den unwichtigen oder weniger wichtigen Daten zu trennen. Beim *harten thresholding* wird eine Grenze festgesetzt, die ein Wert überschreiten muß, um als relevant eingestuft zu werden. Im einfachsten Fall ist diese Grenze ε für alle Eingabewerte s_i gleich. Für die Kompressionsmatrix $\mathbf{K}$ ergibt sich dann

$$K_{ii} \;=\; \begin{cases} 1 & : \quad ||s_i|| \geq \varepsilon \\ 0 & : \quad \text{sonst} \end{cases} .$$

In Abbildung 18 links ist ein *hartes thresholding* für $f(x) = x$ mit einer Grenze von Eins abgebildet.

Bei Wavelets wird in der Signalanalyse häufig auch ein *weiches thresholding* angewandt (Vidaković und Müller 1993). Diese Art des Verwerfens von Werten beruht auf der Beobachtung, daß die Unstetigkeit beim *harten thresholding* ungünstige Einwirkungen auf die Darstellung der relevanten Daten hat. Beim *weichen thresholding* wird also darauf geachtet, daß bei dem Verwerfen der Daten keine Unstetigkeiten künstlich hinzugefügt werden. Ein Beispiel eines solchen *thresholdings* ist in Abbildung 18 rechts zu sehen.

Bei der tensorierten Hierarchischen Basis werden häufig nur die Koeffizienten auf einem *dünnen Gitter* (Zenger 1991) behalten. Diese Art von

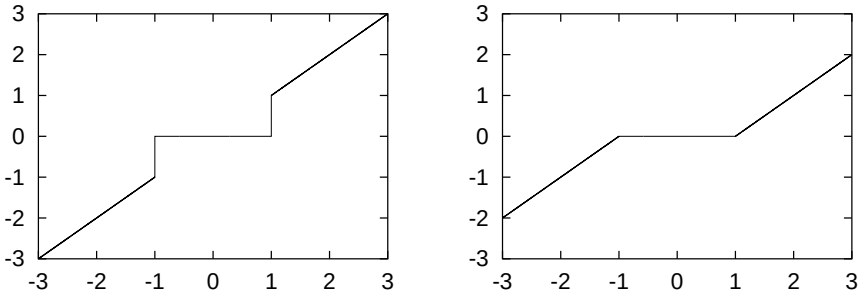


Abbildung 18: *Hartes thresholding* mit Grenze Eins und Funktion $f(x) = x$ (links). *Weiches thresholding* mit Grenze Eins ($\lambda = 1$) und Funktion $f(x) = x$. Als *thresholding* Funktion wurde $f(x) = \text{sign}(x)(|x| - \lambda)_+$ gewählt (rechts).

thresholding verwirft Koeffizienten aufgrund ihrer Position und nicht ihrer Größe. Ein Schema zur Bildung eines dünnen Gitters ist in Abbildung 19 zu sehen. Alle dort abgebildeten Basisfunktionen zusammen ergeben ein volles Gitter. Werden alle Koeffizienten der Basisfunktionen rechts unter der eingezeichneten Grenzlinie auf Null gesetzt, so ergibt sich das dünne Gitter.

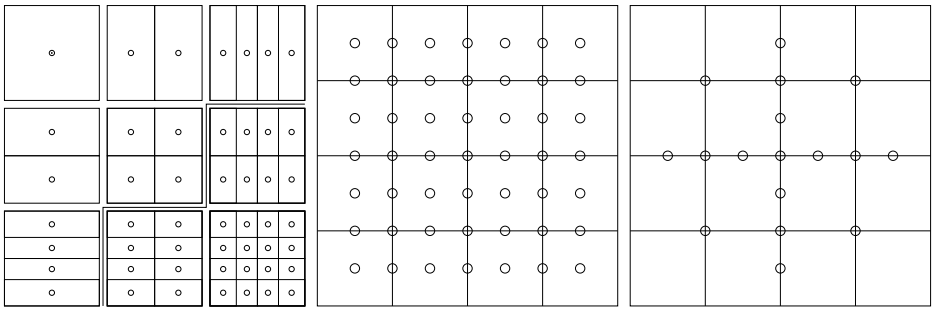


Abbildung 19: Dünnes Gitter als Beispiel eines positionsabhängigen Kompressionsverfahrens. Von links nach rechts: Schema zur Erzeugung des dünnen Gitters: nur die Basisfunktionen links oberhalb der Grenzlinie werden genutzt; volles Gitter und dünnes Gitter.

4.1.2 Quantisierung

Während beim *thresholding* alle Werte verworfen werden, die kleiner als eine bestimmte Schranke sind, wird bei der Quantisierung (Sayood 2000) ein Teil von jedem Wert weggelassen. Dies kann mit der Fischbearbeitung auf einem

Fischerboot verglichen werden. Zuerst werden dort alle Fische, die zu klein sind, ins Meer zurückgeworfen. Das entspricht dem *thresholding*. Dann werden aus jedem Fisch die Eingeweide herausgenommen und nur der ausgeweidete Fisch wird behalten. Das entspricht dem Quantisieren.

Das Entfernen von nicht relevanten Teilen eines jeden Wertes ist die Quantisierung. Eine erste Quantisierung findet meist schon bei der Erfassung der Eingabewerte statt. Denn i. allg. können die zu verarbeitenden Werte nicht beliebig genau erfaßt werden. Es wird also von jedem Wert ein Teil weggelassen. Diese Quantisierung, die bei der Meßwerterfassung nicht unbedingt gewollt ist, kann bei der Speicherung der Werte durchaus gewollt sein. Außerdem ist die Quantisierung besser steuerbar. Eine einfache Form der Quantisierung sind die $\lfloor . \rfloor$- und $\lceil . \rceil$-Klammern. Also etwa die Funktion $f(x) = \lfloor x \rfloor$ wie sie in Abbildung 20 zu sehen ist.

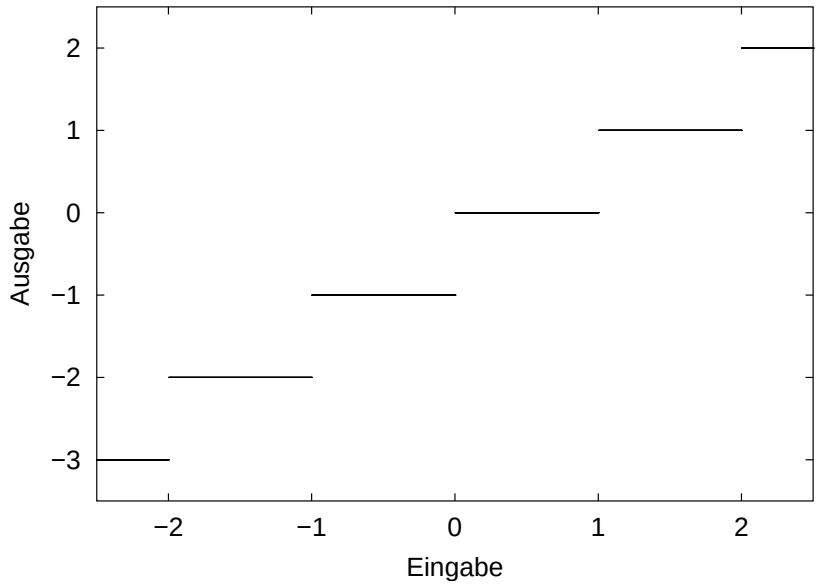


Abbildung 20: Beispiel einer Quantisierung durch $f(x) = \lfloor x \rfloor$.

Durch Einsatz dieser einfachen Quantisierungsfunktion werden alle Werte gleich behandelt. Vielleicht ist es aber erwünscht, unterschiedlich große Werte oder Werte aufgrund ihrer Position im Eingabestrom oder Raum unterschiedlich zu behandeln. Die angewandte Quantisierung muß auf den Anwendungsfall optimiert sein. So wird z. B. die Z-Koordinate in OpenGL logarithmisch quantisiert (Woo u. a. 1997). Größere Z-Werte werden also ungenauer gespeichert. Die Z-Koordinate entspricht der Achse, die vom Betrachter weg ins Bild hineinzeigt. Objekte mit großen Z-Werten liegen also weit entfernt vom

Betrachter im Bild. Diese Objekte nehmen aber in der Projektion nur wenig Platz ein und Fehler fallen mit der Entfernung immer weniger auf. So ist es z. B. einfach zu unterscheiden, ob ein Baum 1 oder 10 Meter vom Betrachter entfernt ist, aber ob er 200 oder 210 Meter entfernt steht, ist fast unmöglich zu beurteilen. Die Z-Koordinatenquantisierung ist in Abbildung 21 zu sehen.

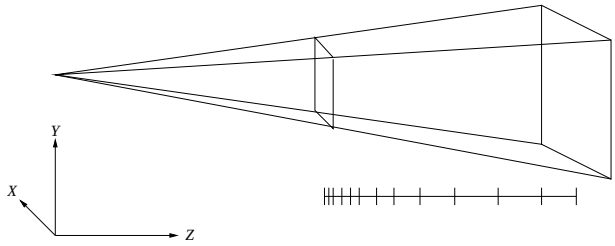


Abbildung 21: Z-Koordinatenquantisierung unter OpenGL. Mit zunehmender Tiefe werden die Z-Werte ungenauer gespeichert.

4.2 Vergleich der Kompressionsverfahren

Wie oben schon erwähnt, ist ein Vergleich zwischen den beiden vorgestellten Verfahren nicht sinnvoll, da beide Methoden gleichzeitig angewandt werden können und sich so ergänzen. Eine Quantisierung der Daten erfolgt meist schon bei der Erfassung der Daten, da nur mit einer bestimmten Genauigkeit gemessen werden kann. So sind in allen in dieser Arbeit verwandten Bilder die Grauwerte auf ganze Zahlen im Bereich von 0 bis 255 quantisiert.

Um *thresholding* vernünftig einsetzen zu können, muß der dadurch entstehende Fehler errechnet werden können. Da die Kompression in dieser Arbeit erst nach der Transformation T angewandt wird, muß der Fehler nach der Rücktransformation ermittelt werden. Diese Fehlerrechnung wird im nächsten Abschnitt genauer behandelt.

5 Fehlerbeschränkung

Sobald relevante Daten weggelassen werden, entsteht ein Fehler der genäherten Daten $\tilde{\mathbf{f}}$ zu den Ausgangsdaten $\mathbf{f}$. Bei den gängigen Verfahren wird eine Fehlerschätzung in der L_2-Norm vorgenommen. Dies ist aber für viele Anwendungen, z. B. medizinische Daten, häufig nicht genug. In dieser Arbeit wird deshalb die Maximumsnorm zugrunde gelegt. Außerdem wird der Fehler nicht nur geschätzt, sondern garantiert beschränkt. Die Maximumsnorm $||.||_\infty$ ist dabei wie folgt definiert:

$$||\mathbf{x}||_\infty := \max\{|x_1|, \ldots, |x_n|\}.$$

In den folgenden Abschnitten wird gezeigt, wie eine Fehlerschranke für diese Norm und die Hierarchische Basis erzeugt werden kann.

5.1 A priori Fehlerbeschränkung

Wenn der Fehler gleichmäßig auf alle Level verteilt wird, so kann für einen vorgegebenen Fehler die Fehlerschwelle für alle Level errechnet werden. Wichtig ist hierbei, daß eine endliche Leveltiefe vorausgesetzt wird. Dies ist aber bei Bilddaten auf jeden Fall gegeben und in der Praxis somit keine Einschränkung. Im folgenden Abschnitt wird eine a priori-Fehlerschranke für die eindimensionale Hierarchische Basis hergeleitet.

In der Matrixschreibweise kann das Abschneiden der Koeffizienten durch eine Multiplikation mit einer Diagonalmatrix $\mathbf{K}$ dargestellt werden, wobei

$$\mathbf{K} = \begin{cases} 1 & \text{für } |c_{li}| > \varepsilon_{\text{cut}} \\ 0 & \text{für } |c_{li}| \leq \varepsilon_{\text{cut}} \end{cases}$$

ist und c_{li} die Einträge aus $\mathbf{Tf}$ sind. Sei $\tilde{\mathbf{f}} = \mathbf{T}^{-1}\mathbf{KTf}$, dann ist der Fehler, der nun gemacht wird

$$||\mathbf{e}|| = ||\mathbf{f} - \tilde{\mathbf{f}}||.$$

Der Fehler, der bei einer Funktion höchstens gemacht werden kann, entsteht durch Null-Setzen von Koeffizienten, die alle gleiches Vorzeichen haben und die gerade kleiner als ε_{cut} sind. In der Matrixschreibweise entspricht dies einer Diagonalmatrix $\mathbf{K} = 0$ und $\tilde{\mathbf{f}} = 0$. Der Fehler $\mathbf{e}$ läßt sich dann direkt aus der *worst case*-Funktion $\mathbf{f}$ ableiten und beträgt

$$||\mathbf{e}|| = ||\mathbf{f}|| = ||\mathbf{T}^{-1}\mathbf{c}|| = ||\mathbf{T}^{-1}\mathbf{1}|| \cdot \varepsilon_{\text{cut}}.$$

Die *worst case*-Funktion $\mathbf{f}$ ist die Rücktransformation von $\hat{\mathbf{f}}$, wobei $\hat{\mathbf{f}}$ aus ε_{cut} als Koeffizienten besteht.

Die obige Methode hat den Nachteil, daß alle Level gleich behandelt werden. Es wird also nicht ausgenutzt, daß die Koeffizienten i. d. R. mit dem Level abfallen. Um hier eine Verbesserung zu erreichen, wird ε_{cut} vom Level l und eventuell auch von der Position i abhängig gemacht. ε_{cut} wird damit zu einer zweiparametrigen Funktion $\varepsilon_{\text{cut}}(l,i)$.

$\varepsilon_{\text{cut}}(l,i)$ ist die Abschneidefunktion, die abhängig von Level l und Ort i den maximalen Betrag ermittelt, den ein Koeffizient auf Level l und Ort i haben darf, um den Schwellenwert ε global nicht zu überschreiten.

Wenn ε_{cut} von Level l und Position i abhängt, der Schwellenwert also $\varepsilon_{\text{cut}}(l,i)$ ist, so kann der Fehler mittels einer *worst case*-Funktion beschränkt werden (siehe Abbildung 23 links). Wenn nun die Koeffizienten auf ein solches $\varepsilon_{\text{cut}}(l,i)$ gesetzt werden, daß die daraus in die nodale Basis transformierte Funktion den eigentlichen Schwellenwert ε nicht überschreitet, so ist ein geeignetes $\varepsilon_{\text{cut}}(l,i)$ gefunden. In Abbildung 22 sind die abfallenden $\varepsilon_{\text{cut}}(l,i)$ auf den verschiedenen Leveln dargestellt.

Bei Vernachlässigung des Ortes i ist ein – vom Glattheitsparameter ξ abhängiges – geeignetes $\varepsilon_{\text{cut}}(l,i)$ durch

$$\varepsilon_{\text{cut}}(l,i) = c \cdot \xi^{(l+1)}\,,\ 1/2 \le \xi < 1, \tag{8}$$

gefunden mit

$$c = \varepsilon \cdot \left(\frac{2}{3} \cdot \frac{\xi}{1-\xi}\right)^{-1}.$$

Denn für die Fehlerwerte a_l auf den Leveln $l-1$ ergibt sich folgende rekursive Fehler-Formel:

$$\begin{aligned} a_{-1} = a_{-2} &= 0 \\ a_l &= \frac{a_{l-1} + a_{l-2}}{2} + \xi^{(l+1)}\,. \end{aligned}$$

Die Fehlerwerte auf den Leveln $l < 0$ sind natürlich alle Null. Nach Auflösen der Rekurrenzrelation ergibt sich:

$$\begin{aligned} a_L &= \sum_{l=1}^{L} \left[\sum_{n=0}^{L-l} \left(-\frac{1}{2}\right)^n\right] \cdot \xi^l \\ &= \sum_{l=1}^{L} \left(\frac{2}{3} + \frac{1}{3} \cdot (-2)^{L-l}\right) \cdot \xi^l \\ \Rightarrow \lim_{L\to\infty} a_L &= \frac{2}{3} \cdot \frac{\xi}{1-\xi}\,, \end{aligned}$$

c muß also als $\varepsilon \cdot \left(\frac{2}{3} \cdot \frac{\xi}{1-\xi}\right)^{-1}$ gewählt werden, wobei $\lim_{L\to\infty} a_L$ gerade dem Wert des Maximums der *worst case*-Funktion mit $1/2 \leq \xi < 1$, das an den Stellen $1/3$ und $2/3$ erreicht wird, entspricht. Das Maximum der *worst case*-Funktion für $0 \leq \xi \leq 1/4$ hingegen liegt bei $1/2$ und ist damit ganz einfach der erste Koeffizient auf dem Level 0. Im Falle $0 \leq \xi \leq 1/4$ lautet also die Formel

$$\varepsilon_{\text{cut}}(l,i) = \varepsilon \cdot \frac{1}{\xi^{(l+1)}} \tag{9}$$

Die Maxima für $1/4 < \xi < 1/2$ wandern relativ chaotisch von $1/2$ nach $\{1/3, 2/3\}$. Die Wahl des Parameters ξ sagt etwas über die Glattheit der *worst case*-Funktion aus. So ergibt sich z. B. für $\xi = 0,25$ eine Parabel und für $\xi = 0$ die Gerade. Der Parameter kann damit aber an die Glattheit des Bildes angepaßt werden. Je glatter das Ausgangsbild ist, um so näher an Null kann ξ gewählt werden.

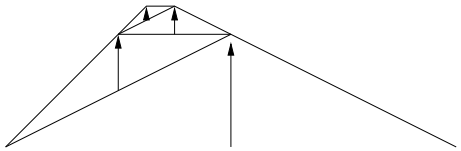


Abbildung 22: Abfallen der $\varepsilon_{cut}(l,i)$.

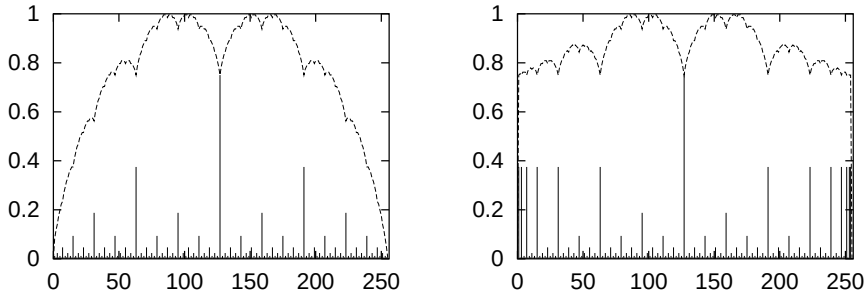


Abbildung 23: Level-abhängige und Orts-unabhängige *worst case*-Funktion für $\xi = 1/2$ (links) und Orts- und Level-abhängige *worst case*-Funktion $\varepsilon_{cut}(l,i)\,, \xi = 1/2$ (rechts). Es sind jeweils die Koeffizienten und der an den Positionen entstehende Fehler dargestellt.

5.1.1 Orts- und Level-abhängiges Koeffizientenabschneiden

In Abbildung 23 rechts sind die äußersten $\varepsilon_{\text{cut}}(l,i)$ mit Level $l > 1$ auf $\varepsilon^l \cdot 3/8$ gesetzt. Es gilt also für $1/2 \leq \xi < 1$:

$$\varepsilon_{\text{cut}}(l,i) = \begin{cases} (l>1) \text{ und } ((i=1) \text{ oder } (i=2^l-1)) & : \varepsilon^l \cdot \frac{3}{8} \\ \text{sonst} & : \varepsilon \cdot \frac{2}{3} \cdot \frac{1-\xi}{\xi} \cdot \xi^l \end{cases} . \quad (10)$$

Die dadurch erhaltene *worst case*-Funktion nutzt den Fehler auf dem gesamten Intervall besser aus und kann insgesamt mehr Koeffizienten auf Null-Setzen als die ortsunabhängige Version.

Das Maximum dieser *worst case*-Funktion liegt ebenfalls bei $\{1/2, 2/3\}$ für $1/2 \leq \xi < 1$ und beträgt ε.

5.1.2 Koeffizientenabschneiden bei endlicher Leveltiefe

Die oben angestellten Betrachtungen gelten für $l \to \infty$, wohingegen in der Praxis meist mit endlicher Leveltiefe gearbeitet wird. Es ist also zu überprüfen, ob bei bekannter Leveltiefe die ε_{cut} nicht noch etwas schärfer beschränkt werden können. Bei Verwendung der Version 9 kann sicher nicht genauer beschränkt werden, da der größte Fehler bereits mit dem ersten Koeffizienten gemacht wird. Übrig bleibt Version 8. Hier muß nur die Summe mit Leveltiefe L berechnet werden. Es zeigt sich aber, daß bei einer Leveltiefe von $L = 10$ der Wert von Methode 8 schon bis auf die dritte Stelle hinter dem Komma genau erreicht wird.

Bisher wurde immer davon ausgegangen, daß die $\varepsilon_{\text{cut}}(l,i)$ mit zunehmendem Level kleiner werden. Für unendliche Leveltiefe ist dies eine notwendige Eigenschaft. Für endliche Leveltiefe können die $\varepsilon_{\text{cut}}(l,i)$ mit größer werdender Leveltiefe allerdings auch zunehmen, da hier nur auf eine endliche Summe Rücksicht genommen werden muß. Diese kann ausgerechnet und somit der Fehler beschränkt werden. Gegenüber den grundsätzlich auch verwendbaren gleichbleibenden Koeffizienten haben zunehmende Koeffizienten den Vorteil, daß die Koeffizienten von hohen Leveln eher abgeschnitten werden, da bei genügend glatten Funktionen die Koeffizienten mit dem Level abnehmen und so von den ansteigenden $\varepsilon_{\text{cut}}(l,i)$ mit zunehmender Wahrscheinlichkeit auf Null gesetzt werden.

5.2 Übergang zu 2-D mit Tensorprodukt-Ansatz

Die oben vorgestellten adaptiven Verfahren in 1-D können nicht ohne weiteres in 2-D übernommen werden, da es keine einfache Rekursionsstruktur mehr gibt. Es kann in 2-D der Fall eintreten, daß das Maximum in der nodalen

Basis an einer Stelle liegt, an der der hierarchische Koeffizient Null ist. Dies führt z. B. bei dem Greedy-Algorithmus dazu, daß überprüft werden muß, ob eine solche Situation vorliegt und korrigiert werden muß.

Die hier vorgestellten Verfahren orientieren sich alle an den *Dünnen Gittern*, d.h. die Blöcke, die sich aus dem Tensorprodukt-Ansatz ergeben, werden in Diagonalen zusammengefaßt.

5.2.1 A priori Fehlerbeschränkung in 2-D

Für den zweidimensionalen Fall gilt für $\varepsilon_{\text{cut}}(x, y, l_x, l_y)$ eine ähnliche Formel wie in 1-D. Es muß nur darauf geachtet werden, daß die Konstante c von $2/3 \cdot 1/(1-\xi)$ zu $(2/3 \cdot 1/(1-\xi))^2$ wird und der Level sowohl in x- als auch in y-Richtung beachtet werden muß. Es ergibt sich also für $1/2 \leq \xi < 1$:

$$\varepsilon_{\text{cut}}(x, y, l_x, l_y) = \varepsilon \cdot \left(\frac{2}{3} \cdot \frac{\xi}{1-\xi}\right)^{-2} \cdot \xi^{1+l_x+l_y}.$$

Oder allgemeiner: Für n-D wird $\varepsilon_{\text{cut}}(x_1, \ldots, x_n, l_{x_1}, \ldots, l_{x_n})$ zu:

$$\varepsilon_{\text{cut}}(x_1, \ldots, x_n, l_{x_1}, \ldots, l_{x_n}) = \varepsilon \cdot \left(\frac{2}{3} \cdot \frac{\xi}{1-\xi}\right)^{-n} \cdot \xi^{(1+\sum_{i=1}^{n} l_{x_i})}.$$

5.2.2 Adaptive Fehlerbeschränkung

Die vorangegangenen Fehlerschranken haben keine lokalen Glattheitseigenschaften der zu bearbeitenden Daten nutzen können. Um den Fehler auf dem ganzen Intervall möglichst uniform zu beschränken, wird ein zusätzlicher Fehlervektor benötigt. In diesem Fehlervektor wird gespeichert, wie viele Koeffizienten auf Null gesetzt werden und welchen Einfluß sie auf die synthetisierte Funktion haben werden, also das Produkt der auf Null gesetzten Koeffizienten mal der Gewichtung der zugehörigen Basisfunktion. Der Aufwand dieser Methode ist zwar sehr viel höher als bei den oben beschriebenen Beschränkungs-Methoden, dafür paßt sich aber diese Methode den Funktionen besser an. Für Daten, die gespeichert werden sollen und sich danach nicht mehr verändern, ist der Speicherplatzgewinn das auschlaggebende Argument. Adaptive Verfahren im Tensorprodukt-Ansatz sind:

Einfach Adaptiv (oder *top down*): Bei diesem Verfahren werden die Diagonalen von links oben nach rechts unten abgearbeitet. Auf jeder Diagonalen wird der maximale Schwellenwert bestimmt, der noch vergeben werden darf. Dieser Schwellenwert wird mit einem Sicherheitsparameter α multipliziert. Jetzt werden alle hierarchischen Koeffizienten

der Diagonale, die kleiner als der Schwellenwert sind, auf Null gesetzt. Dieses Verfahren wird auch *top down* genannt.

Multi Adaptiv: Dieses Verfahren ist eine Verfeinerung des Verfahrens *top down*. Hier wird ausgenutzt, das die Diagonalen rechts unterhalb der Mitteldiagonalen in unabhängige Teildiagonalen zerfallen; sie können also unterteilt betrachtet werden. Der Nachteil ist allerdings ein höherer Speicheraufwand im Vergleich zu *top down*.

***bottom up*:** Wenn die Durchlaufrichtung der Diagonalen von *top down* umgedreht wird, die Diagonalen also von rechts unten nach links oben durchlaufen werden, so wird von *bottom up* gesprochen.

5.2.3 Greedy

Beim Greedy-Verfahren wird in jedem Schritt aus den hierarchischen Knotenwerten die Rücktransformation in die nodale Basis berechnet. Der hierarchische Koeffizient, der an der Stelle des größten absoluten nodalen Wertes liegt, wird auf Null gesetzt. Danach wird wieder in die nodale Basis umgerechnet und wieder auf Null gesetzt, bis letztendlich kein Wert in der nodalen Basis mehr über dem vorgegebenen ε liegt. Nun werden alle ursprünglich berechneten hierarchischen Koeffizienten auf Null gesetzt, die der Algorithmus nicht auf Null gesetzt hat. Der Fehler ist damit auf ε beschränkt.

Bei dieser Methode können noch einige Tricks eingebaut werden, die das Verfahren beschleunigen. So können die Koeffizienten-Positionen mit ihren Fehlereinflüssen in einer Warteschlange gespeichert werden. Aus dieser Warteschlange können dann gleichzeitig mehrere Koeffizienten entnommen werden, deren Träger sich nicht überlappen. Hierbei verliert die Methode allerdings sehr stark in Bezug auf ihre Kompressionseigenschaft.

Beim Tensorprodukt-Ansatz muß noch beachtet werden, daß die Koeffizienten an Stellen maximalen Fehlers schon auf Null gesetzt sein können. In solchen Fällen muß der nächste Koeffizient mit dem größten Einfluß auf den oben genannten Koeffizienten gesucht werden.

5.3 Übergang zu 2-D mit Triangulierung

Bei dem Tensorprodukt-Ansatz hindern die überlappenden und stark gestreckten Basisfunktionen einfache adaptive Verfahren. Das Greedy-Verfahren kann zwar gute Kompressionsraten erreichen, hat dafür aber auch eine hohe Komplexität. Beim Übergang zu 2-D mittels Triangulierung können einfache adaptive Verfahren eingesetzt werden. In den folgenden Abschnitten werden zwei Methoden vorgestellt.

5.3.1 1-level-look-ahead

Wenn einige Koeffizienten in der Hierarchischen Basis weggelassen werden, entsteht ein Fehler zur ursprünglichen Darstellung. Um den Fehler dieser Approximation genau angeben zu können, kann mit verschiedenen Methoden gearbeitet werden.

Eine sehr einfache Methode ist der *1-level-look-ahead* oder auch *bounding box*. Diese Methode versieht jeden Koeffizienten mit einem Fehlerwert. Dieser Wert gibt den Fehler an, der gemacht wird, wenn dieser Koeffizient und alle im Baum nach ihm abgehenden weggelassen werden. Da die weggelassenen Koeffizienten einen Teilbaum bilden, wird diese Methode auch Teilbaumabschneiden genannt.

Der entstehende Fehler auf dem gesamten Gebiet ist genau gegeben durch eine Rücktransformation, bei der alle nicht weggelassenen Koeffizienten auf Null gesetzt werden. Der maximale Fehler liegt aber i. allg. nicht auf dem hierarchisch höchsten Koeffizienten. Es muß also noch eine Suche über den gesamten Träger eines jeden Koeffizienten durchgeführt werden, um den maximalen Fehler des jeweils zugehörigen Gebietes zu finden.

Wenn dies aus Zeitgründen nicht erwünscht ist und der Fehler auch etwas zu grob beschränkt werden kann, so ist es einfacher für jeden Koeffizienten nur eine Fehlerrechnung anhand der direkten Söhne im Baum vorzunehmen, also nur einen Level weiterzusuchen. Der Baum wird dazu von den Blättern her bearbeitet. Die Blätter bekommen als Fehlerwert die Koeffizienten zugeteilt, denn das ist genau der Fehler, der durch Weglassen der feinsten Koeffizienten entsteht. Daraufhin wird der Baum bis zur Wurzel abgearbeitet. Jeder Knoten erhält einen Fehlerwert, der aus dem jeweiligen Koeffizienten und den beiden Sohn-Fehlerinformationen berechnet wird.

Zur Fehlerberechnung wird jetzt die *bounding box* verwandt. Es wird eine Kiste um jeden Koeffizienten gebaut, die die Rücktransformation der bis zu diesen Koeffizienten ausgelassenen Koeffizienten umschließt. Diese Kiste ist bei den Blättern genauso hoch wie der Koeffizient selber. Die Größe der Kiste für einen Knoten ist die Summe der absoluten Beträge des Koeffizienten und der maximale absolute Betrag der beiden Sohn-Fehlerinformationen (siehe hierzu auch Abbildung 24).

$$
\begin{aligned}
k_v &= |c_v| + \max\{|k_{s_l}|, |k_{s_r}|\} \text{ , mit} \\
k_v &= \text{Fehlerinformation des Vaterknotens} \\
k_{s_{l,r}} &= \text{Fehlerinformationen der Sohnknoten (links und rechts)} \\
c_v &= \text{Koeffizient des Vaterknotens}
\end{aligned}
$$

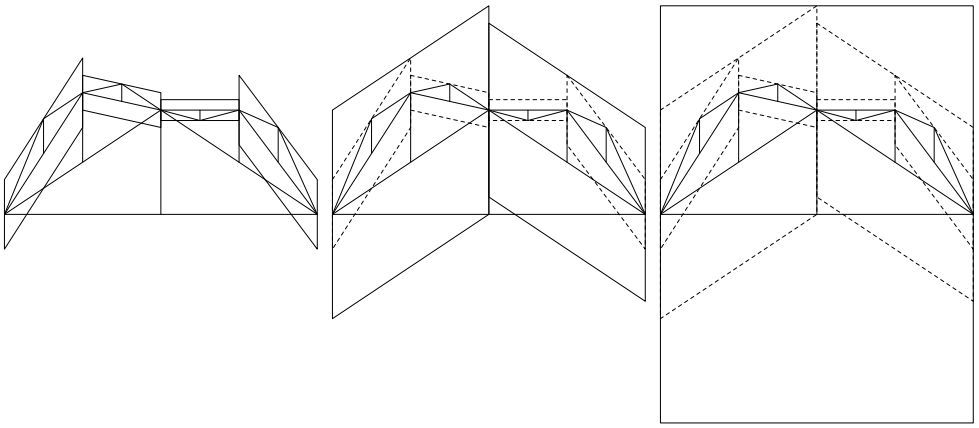


Abbildung 24: Aufbau der *bounding box* für *1-level-look-ahead*. Gestrichelte Linien zeigen die Schranken für den Fehler auf den nächstfeineren Leveln.

Da der angegebene Fehler in dem Knoten immer für den gesamten Träger der Koeffizienten gilt, ist diese Methode auch auf den Triangulierungs-Ansatz in 2-D (n-D) zu übertragen. Allerdings muß in 2-D darauf geachtet werden, daß keine hängenden Knoten entstehen. Hängende Knoten sorgen bei Bildern für Unstetigkeiten in der Darstellung. Wenn hängende Knoten bei digitalen Höhenmodellen vorkommen, so können Löcher in der Darstellung der Landschaft auftauchen. Ein hängender Knoten kann in 2-D bei dieser Methode entstehen, wenn wie in Abbildung 25 der Knoten a noch verfeinert wird, Knoten b aber nicht. Dies kann passieren, wenn die für die Verfeinerung von Knoten a zuständige Fehlerinformation nur auf dem Träger von Knoten a, aber nicht auf dem von Knoten b liegt. Um solche Fälle zu vermeiden, muß der Baum nicht nur einmal von den Blättern zur Wurzel hin abgearbeitet werden, sondern für jeden Level einmal. Das heißt, es müssen zuerst alle Koeffizienten wie oben beschrieben bearbeitet werden; danach alle bis auf die Blätter. In den darauffolgenden Schritten werden die Väter der jeweiligen Blätter als neue Blätter behandelt. Dies wird so lange wiederholt, bis nur noch der Wurzelknoten als einziges Blatt übrigbleibt. Dieser Vorgang, der die hängenden Knoten entfernt, heißt auch Saturation.

Die *bounding box* kann noch etwas genauer an den wirklichen Fehler angepaßt werden, wenn der Level l in den Algorithmus mit einbezogen wird. Es wird dann nicht der volle Fehler der feineren Level aufgerechnet, sondern nur bis zum nächsten feinsten Koeffizienten anteilig addiert. Die oben genannte Formel erweitert sich somit zu:

$$k_v = |c_v| + \frac{2^{L-l} - 1}{2^{L-l}} \cdot \max\{|k_{s_l}|, |k_{s_r}|\}$$

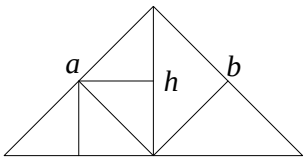


Abbildung 25: Entstehung eines hängenden Knotens h. Knoten a wird verfeinert, da auf dem Träger von a ein großer Fehlerinformationswert ist, auf dem Träger von b hingegen nicht.

5.3.2 n-level-look-ahead

Bei der *1-level-look-ahead* Methode wachsen die Schranken für den Fehler sehr rasch an. Dies führt zu schlechten Kompressionsergebnissen, da viele Koeffizienten abgespeichert werden, die einen realen Fehler kleiner als den vorgegebenen Schwellenwert produzieren. Um dies zu vermeiden, muß der Fehler für die einzelnen Koeffizienten genauer ausgerechnet werden. Wie bei der *1-level-look-ahead* Variante kann auch hier eine rekursive Formel angegeben werden. Diese Formel lautet für 1-D mit den Startwerten $w_1 := f(x_1)$ und $w_2 := f(x_2)$:

$$\begin{aligned} k(x_1, x_2, w_1, w_2) = \max \Bigg\{ & \left| f\left(\frac{x_1 + x_2}{2}\right) - \frac{w_1 + w_2}{2} \right|, \\ & k\left(x_1, \frac{x_1 + x_2}{2}, w_1, \frac{w_1 + w_2}{2}\right), \\ & k\left(\frac{x_1 + x_2}{2}, x_2, \frac{w_1 + w_2}{2}, w_2\right) \Bigg\}. \end{aligned}$$

Die *bounding box* wird für jeden Koeffizienten auf dem Träger der entsprechenden Basisfunktion rekursiv berechnet. Dafür wird die Funktion k mit den Eckpunkten x_1 und x_2 und den jeweiligen Werten an den Stellen als w_1 und w_2 initialisiert. Die Funktion k selber ermittelt das Maximum aus der absoluten Differenz des Wertes in der Mitte des aktuellen Abschnittes mit dem Interpolanden von w_1 und w_2 an derselben Stelle und den Ergebnissen der rekursiven Aufrufe von k für die linke und die rechte Hälfte. Für diese Aufrufe werden die neuen Abschnitts-Eckpunkte als neue x_1 und x_2 und die interpolierten Werte von w_1 und w_2 als neue w_1 und w_2 verwandt.

Für den 2-D-Fall müssen noch zwei weitere Werte x_3 und w_3 hinzugenommen werden, um die Dreiecke der Triangulierung interpolieren zu können. Die Numerierung der Ecken der Dreiecke entspricht dabei der in Abbildung 17 auf Seite 38 angegebenen Numerierung. Die Initialisierung der neuen Funktion k

erweitert sich so zu:

$$w_1 := f(x_1); \quad w_2 := f(x_2); \quad w_3 := f(x_3).$$

Die Funktion k sieht dann folgendermaßen aus:

$$\begin{aligned} k(x_1, x_2, x_3, w_1, w_2, w_3) = \max \Bigg\{ & \left| f\left(\frac{x_2 + x_3}{2} - \frac{w_2 + w_3}{2} \right) \right|, \\ & k\left(\frac{x_2 + x_3}{2}, x_3, x_1, \frac{w_2 + w_3}{2}, w_3, w_1 \right), \\ & k\left(\frac{x_2 + x_3}{2}, x_1, x_2, \frac{w_2 + w_3}{2}, w_1, w_2 \right) \Bigg\}. \end{aligned}$$

Bei der 2-D-Variante ist zu beachten, daß der Punkt $1/2 \cdot (x_2 + x_3)$ nicht in der Mitte des Dreieckes liegt, sondern auf der Mitte der Hypotenuse des aktuellen Dreieckes.

Als Abbruchbedingung kann im diskreten Fall der Vergleich $x_1 = x_2$ für 1-D und $x_2 = x_3$ für 2-D genommen werden, denn dann trägt nur noch der eigentliche Koeffizient zum Gesamtfehler bei. Die Auswirkungen der genaueren Berechnung auf die Größe der *bounding box* ist in Abbildung 26 dargestellt. Die Größe der *bounding box* paßt sich mit dem *n-level-look-ahead*-Verfahren jeweils dem größten Wert auf dem Träger der zugrunde liegenden Basisfunktion an.

Der Aufwand für dieses vollständige Ausrechnen des Fehlers ist viel größer als für die einfache *1-level-look-ahead* Variante (siehe Abbildung 46 auf Seite 101), aber der Fehler wird dafür auch viel besser ausgenutzt, wie in Abbildung 45 auf Seite 100 zu sehen ist.

Für große Datenmengen, bei denen der Aufwand für die *n-level-look-ahead* Methode nicht mehr vertretbar ist, kann auch eine Mischform der beiden Extrema *1-level-* und *n-level-look-ahead* benutzt werden.

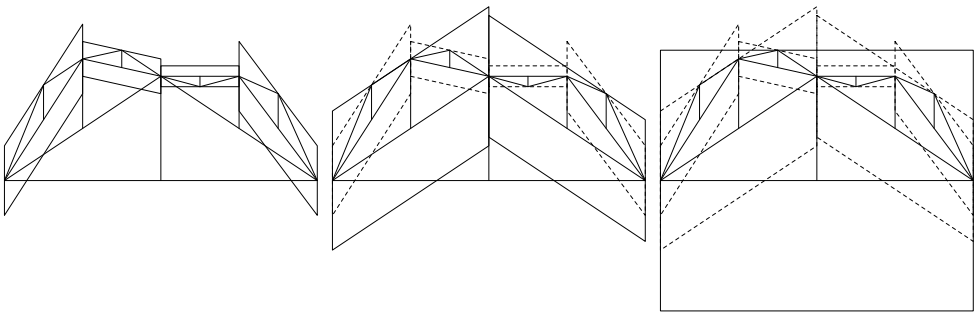


Abbildung 26: Aufbau der *bounding box* für *n-level-look-ahead*. Gestrichelte Linien deuten die Schranken für den Fehler auf den feineren Leveln an.

Dafür wird insgesamt die *1-level-look-ahead* Variante eingesetzt, aber die feinsten n Level werden mit der *n-level-look-ahead* Methode errechnet. Dadurch, daß n variiert werden kann, ist ein guter Kompromiß zwischen Genauigkeit der Beschränkung und dem dafür nötigen Rechenaufwand möglich.

5.4 Vergleich der Fehlerbeschränkungs-Verfahren

Bei einem Vergleich der oben vorgestellten Fehlerbeschränkungs-Methoden müssen zuerst die adaptiven und die a priori-Methoden verglichen werden. Die a priori-Methoden sind unabhängig von den Eingabedaten und sehr schnell. Dafür können sie aber auch keine allzu guten Kompressionsraten erreichen, da sie vom schlechtesten Fall ausgehen müssen.

Die adaptiven Fehlerbeschränkungs-Varianten können sich an die Glattheit der Funktionen anpassen. Ihre Güte zeigt sich in der Fähigkeit, sich lokalen Unstetigkeiten gut anzupassen.

Die Tensorprodukt-Fehlerbeschränkungs-Verfahren können sich lokalen Dateneigenschaften nicht so gut anpassen wie die Triangulierungen, da die Träger der Basisfunktionen sehr gestreckt sein können und sich für die Fehlerschranken ungünstig überlappen.

Für den Tensorprodukt-Ansatz kann das Greedy-Verfahren mindestens gleich gute Werte wie die anderen adaptiven Tensorprodukt-Fehlerbeschränkungs-Varianten erzielen, da Greedy beendet wird, sobald die Fehlergrenze unterschritten wird, also eine minimale Anzahl von Koeffizienten weggelassen wird. In vielen Fällen kann daher auf bessere Kompressionsraten durch Greedy gehofft werden. Dazu wird allerdings ein erheblicher Mehraufwand benötigt.

Bei den Triangulierungs-Fehlerbeschränkungs-Methoden ist die *1-level-look-ahead* Variante recht schnell und zudem auch einfach zu implementieren. Aber schon das Beispiel aus Abbildung 24 zeigt, daß die Fehlerschranken schnell viel zu grob berechnet werden.

Alle diese Überlegungen zusammen lassen das *n-level-look-ahead*-Verfahren bei dem Triangulierungs-Ansatz als das beste Verfahren erscheinen, sofern der höhere Rechenaufwand in Kauf genommen werden kann.

Abschließend sollte erwähnt werden, daß jedes verlustbehaftete Kompressions-Verfahren in eines mit garantiertem Fehler verwandelt werden kann, wenn es einen Qualitäts-Parameter hat. Über diesen kann iteriert werden, wobei der Parameter kleiner eingestellt wird, wenn der Fehler des dekomprimierten Bildes zum Original-Bild zu groß ist.

6 Struktur-Kodierungsverfahren

Bei der Struktur-Kodierung kommt es darauf an, Strukturen in Daten aufzuspüren und diese möglichst effizient abzuspeichern.

Bei der oben vorgestellten hierarchischen Triangulierung ergibt sich eine Baumstruktur. Kodierer für generelle Bäume und solche, die bei der hierarchischen Triangulierung auftreten, werden in Abschnitt 6.2 beschrieben. Bei dem Tensorprodukt-Ansatz für die Hierarchische Basis ist auch eine Baumstruktur gegeben, dort ist an jedem Knoten ein weiterer Baum angebracht.

Eine Möglichkeit generelle Strukturen zu kodieren, ist in Abschnitt 6.1 skizziert.

6.1 Struktur durch Null-Setzen der Koeffizienten

Eine einfache Möglichkeit eine Struktur in den Daten vorzugeben, ist es alle Knoten, deren Schwellenwerte kleiner als das vorgegebene ε sind, als Nullen abzuspeichern (siehe Abbildung 27). Der nachgeschaltete Daten-Kodierer kann dann diese Nullen speicherplatzsparend ablegen.

Für kleines ε ist diese Art der Struktur-Kodierung nicht nur einfach zu implementieren, auch der Speicherplatzbedarf ist gering.

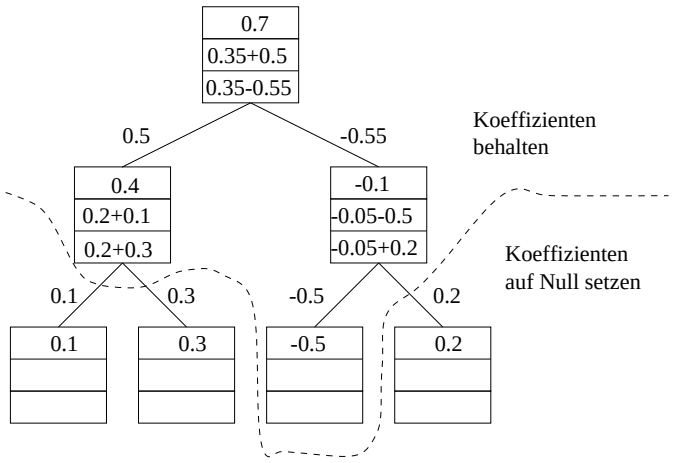


Abbildung 27: Abtrennung von Teilbäumen aufgrund von Fehlerschranken in den Knoten. Die abgeschnittenen Knoten können beim Null-Setzen auf Null gesetzt werden.

6.2 Baumspeicherung

Die hierarchische Triangulierung kann als Binärbaum aufgefaßt werden. Das ursprünglich quadratische Gebiet wird in zwei Dreiecke aufgeteilt, die die ersten beiden Söhne des Wurzelknotens bilden. Jedes Dreieck wird nun rekursiv wieder in zwei Dreiecke gemäß Abbildung 17 auf Seite 38 aufgeteilt. Diese zwei neuen Dreiecke sind dann die neuen Söhne des Vaterdreieckes. Nun muß dieser Baum effizient abgespeichert werden. Es ergibt sich also bei der hierarchischen Triangulierung eine implizite Baumstruktur. Auch bei den unten vorgestellten *Embedded Zerotree for Wavelets* (EZW) und *Set Partitioning in Hierarchical Trees* (SPIHT) Verfahren wird eine implizite Baumstruktur ausgenutzt, die ähnlich kodiert wird wie die Bäume der hierarchischen Triangulierung.

6.2.1 $4m$-Verfahren

Das $4m$-Verfahren speichert für m Koeffizienten im Baum etwa vier Bits ab. Der Baum wird *preorder* rekursiv durchlaufen. Wenn der Baum weiter durchlaufen werden muß, so wird eine Eins gespeichert, sonst eine Null. Ob der Baum weiter durchlaufen werden muß, kann in jedem Knoten anhand des dort gespeicherten Schwellenwertes entschieden werden (siehe Abbildung 27). Der Schwellenwert gibt den Fehler an, der gemacht wird, wenn alle Söhne dieses Knotens weggelassen werden. Für den Baum aus Abbildung 28 ergibt sich somit ein Bitmuster von

$$1101001101001110000.$$

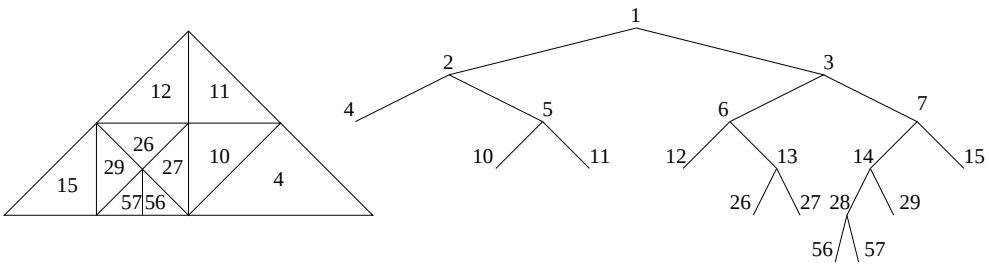


Abbildung 28: Adaptive Triangulierung und entsprechender Binärbaum.

6.2.2 $3m$-Verfahren

Im $4m$-Verfahren wird der Baum von der Wurzel aus verfeinernd abgespeichert. Für bestimmte Bäume kann mit dem $3m$-Verfahren eine noch bessere Kodierung erreicht werden. Dazu muß ein Baum folgende Bedingungen erfüllen:

1. Zwei nebeneinanderliegende Blätter dürfen sich höchstens um einen Level unterscheiden.

2. Ein Knoten hat immer genau zwei Söhne oder ist ein äußerster Knoten (Blatt).

Das Kodieren solcher Bäume geschieht durch ein Abtasten der untersten Baumlinie, also des Blattwerks. Hierfür wird zuerst die Leveltiefe des ersten Blattes, also des äußersten Astes, auf der linken Seite des Baumes ermittelt und abgespeichert. Von diesem Blatt ausgehend wird die Blattlinie nach rechts weiterverfolgt und es werden nur noch die Änderungen zum Vorgänger abgespeichert. Wenn der Baum alle oben aufgeführten Bedingungen erfüllt, sind nur drei Zustände möglich. Entweder liegt das vorhergehende Blatt auf gleichem Level oder genau einen Level tiefer oder höher. Diesen drei Zuständen können drei Werte zugeordnet werden. Es könnte zum Beispiel eine Null für keine Leveländerung und eine Eins für eine solche kodiert werden. Im Falle einer Änderung kann dann durch eine Eins ein Ansteigen und durch eine Null ein Abfallen des Levels angezeigt werden. Dann würde 11 Anstieg und 10 Abfall des Levels bedeuten.

Ein Stoppzeichen, das das Ende der Baum-Kodierung anzeigt, ist nicht nötig. Der Baum hat auf jedem Level eine fest definierte Breite und sowohl der Level als auch die Position auf dem Level ist zu jeder Zeit bekannt.

Da der Baum in Abbildung 28 alle oben geforderten Bedingungen erfüllt, kann eine Kodierung für diesen Baum wie oben beschrieben angegeben werden. Sie lautet:

$$2 \quad 11001101101010.$$

Hierbei muß die führende Zwei noch in eine Bitsequenz umgewandelt werden; das geschieht aber abhängig von der vorher zu bestimmenden Baumtiefe.

6.2.3 $2m$-Verfahren

Das $2m$-Verfahren speichert die einzelnen Dreiecke der Triangulierung entlang einer raumfüllenden (Sierpiński-)Kurve (Sierpiński 1912; Sagan 1994). Die Kurve ist dabei so gewählt, daß die Struktur des Baumes für m Koeffizienten in etwa $2m$ Bits abgespeichert werden kann. Hierzu muß zunächst

der Anfangslevel der Kurve gespeichert werden. Von dort ab wird nur noch gespeichert, ob das nächste auf der Kurve liegende Dreieck auf dem gleichen Level wie das aktuelle Dreieck liegt oder nicht. Dabei steht eine Eins für eine Leveländerung und eine Null für eine gleichbleibende Leveltiefe. Die raumfüllende Kurve ergibt sich ganz von alleine, wenn der zur Triangulierung gehörige Binärbaum vom gröbsten Level aus rekursiv durchlaufen wird, wobei auf jedem Level die beiden neuen Teildreiecke in anderer Reihenfolge bearbeitet werden (siehe hierzu Abbildung 28, Abbildung 29 und Algorithmus 4).

Der Baum in Abbildung 28 kann mit dem $2m$-Verfahren folgendermaßen kodiert werden:

$$2 \quad 100101011.$$

Um beim Laden einfach entscheiden zu können, ob die Leveltiefe erhöht oder erniedrigt wird, wird ein Automat (Gerstner 2000) benutzt. Der benutzte Automat ist in Abbildung 30 zu sehen. Die Kreise geben in der Abbildung an, ob der Level kleiner oder größer wird. Bei einem einfachen Kreis wird verfeinert, bei einem doppelten Kreis vergröbert. Der Weg, der in dem Automaten genommen wird, hängt ab von dem Pfad im Baum, der genommen werden muß, um auf direktem Wege von der Wurzel bis zu dem Ast zu gelangen, der das aktuelle Dreieck repräsentiert. Ein linker Ast bedeutet dabei ein Null und ein rechter eine Eins. An der Wurzel wird der Automat auf den Kreis ganz links initialisiert. Anhand der *Position* des Dreieckes in dem zur Triangulierung gehörenden Binärbaum kann so bestimmt werden, ob das *nächste* Dreieck auf einem höheren oder einem niedrigeren Level liegt, sollte es eine Änderung der Level-Tiefe geben. Die Nummern in den Dreiecken in Abbildung 29 entsprechen den Nummern der Äste des nebenstehenden Baumes.

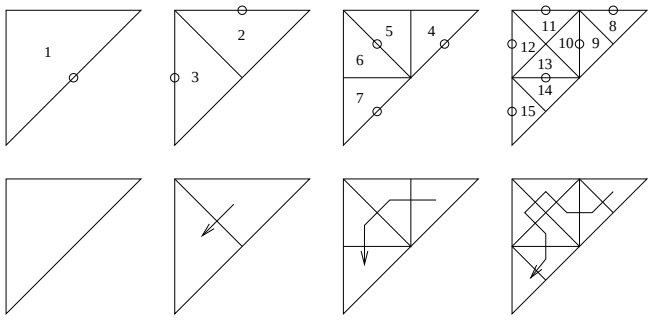


Abbildung 29: Numerierung der Dreiecke und dazugehörige raumfüllende Kurve für das $2m$-Verfahren.

Algorithmus 4: Rekursionsdurchlauf für das $2m$-Verfahren

```
1: Aufruf durch rekDurchlauf (x1, x2, x3, level)
2: if level = endlevel then
3:    lese Änderungsbit ein
4:    if Änderungsbit=1 then
5:       if Maschine steht auf Doppelkringel then
6:          endlevel=endlevel-1
7:       else
8:          endlevel=endlevel+1
9:       end if
10:   end if
11:   Ende der Funktion
12: end if
13: xn = (x2+x3)/2
14: if level gerade then
15:   rekDurchlauf(xn, x1, x2, level+1) {0}
16:   rekDurchlauf(xn, x3, x1, level+1) {1}
17: else
18:   rekDurchlauf(xn, x3, x1, level+1) {0}
19:   rekDurchlauf(xn, x1, x2, level+1) {1}
20: end if
```

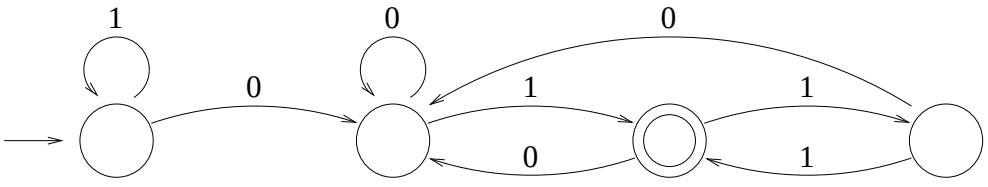


Abbildung 30: Dreiecks-Richtungs-Erkennungs-Automat, der zur Erkennung der Richtung eines Levelwechsels entlang der raumfüllenden Kurve des $2m$-Verfahrens dient. Dabei deutet ein Doppelkringel an, daß der Level vergröbert wird, wohingegen der Level bei einem Einfachkringel verfeinert wird (Gerstner 2000).

6.2.4 Doppelte Knoten

Ein Strukturproblem bei den Bäumen der hierarchischen Triangulierung ist das doppelte Vorkommen der Koeffizienten in den Knoten. Bis auf die Randkoeffizienten sind davon alle Koeffizienten betroffen, da ein Koeffizient dann zu zwei Dreiecken der Triangulierung gehört. Durch einen rekursiven Algorithmus können aber die doppelten Knoten leicht erkannt werden und bei der Speicherung übersprungen werden. Dafür müssen nur die Kanten, auf denen Koeffizienten schon abgespeichert wurden, markiert werden (siehe Abbildung 31). Da die $2m$-, $3m$- und $4m$-Verfahren sowieso schon rekursiv auf dem Baum zur Triangulierung agieren, kann der Algorithmus zur Kantenmarkierung sehr einfach eingebaut werden. Dazu muß nur die Funktion `rek-Durchlauf` aus Algorithmus 4 um drei neue Schalter erweitert werden. Diese Schalter geben Auskunft über die drei Kanten des jeweils aktuellen Dreieckes. Bei der Teilung des Dreieckes wird nur dem zuerst bearbeitetem Dreieck die neuentstandene Kante zugewiesen. Das zweite Dreieck bekommt mitgeteilt, daß die Koeffizienten auf der neuen Kante schon gesichert sind. Der Status der restlichen Seiten wird einfach übernommen.

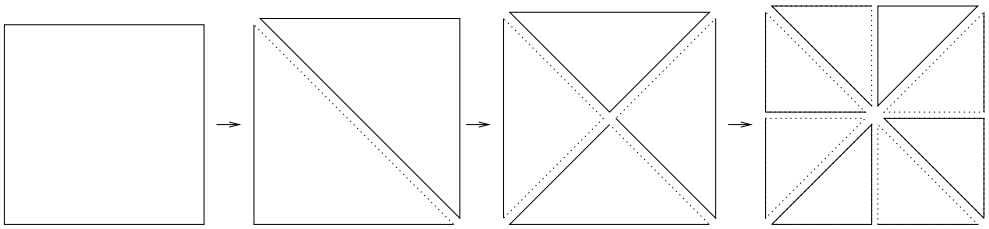


Abbildung 31: Kantenmarkierung zur Erkennung von doppelten Knoten. Gestrichelte Kanten deuten an, daß dort schon alle Koeffizienten bearbeitet wurden.

6.3 Embedded Zerotree for Wavelets (EZW)

Bei dem *Embedded Zerotree for Wavelets*-Verfahren (EZW) (Shapiro 1993; Valens 1999) handelt es sich um eine Mischform aus Struktur-Kodierung und Quantisierung. Das Verfahren basiert auf einer Haar-Wavelettransformation wie sie in Abschnitt 3.1.3 erklärt wurde. Die Wavelettransformation wird mittels Tensorprodukt-Ansatz (siehe Abbildung 15 auf Seite 36) auf 2-D erweitert. Dadurch ergibt sich eine implizite Baumstruktur. Die Koeffizienten auf den feineren Leveln hängen von den entsprechenden Koeffizienten auf den

gröberen Leveln ab. Vor allem bei natürlichen Bildern konzentriert sich deren Energie in den Koeffizienten der gröberen Leveln. Die Abhängigkeit der Koeffizienten auf den unterschiedlichen Leveln ist in Abbildung 32 zu sehen.

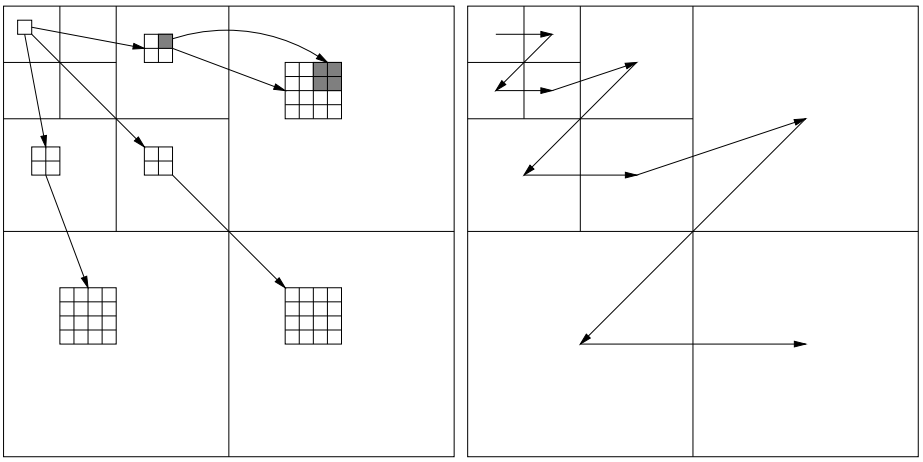


Abbildung 32: Abhängigkeit und Reihenfolge der Teilbäume bei EZW (von links nach rechts).

Bei EZW werden die Koeffizienten von der Wurzel aus abgearbeitet. Jeder Knoten wird dabei in eine von vier Kategorien eingeteilt:

zerotree (zt): Der zugehörige Koeffizient ist kleiner als ein vorgegebener *threshold* T_i und seine Söhne sind ebenfalls *zerotrees* zu diesem *threshold*.

isolated zero (iz): Der zugehörige Koeffizient ist kleiner T_i, aber nicht alle seine Söhne sind *zerotrees* zu diesem *threshold*.

significant positive, negative (sp, sn): Der zugehörige Koeffizient ist größer, respektive kleiner als T_i bzw. $-T_i$.

Die Reihenfolge, in der die Koeffizienten von der Wurzel aus getestet werden, ist in Abbildung 32 zu sehen. Ein Koeffizient wird in dieser Reihenfolge nur getestet, wenn sein Vater im Baum kein *zt* ist und er selber nicht schon *sp* oder *sn* getestet wurde. Die implizite Quantisierung wird bei EZW durch den *threshold* T_i erreicht. Dieser wird zu Beginn auf den Wert

$$\begin{aligned} T_0 &= 2^{\lfloor \log c_{max} \rfloor} \text{ mit} \\ c_{max} &= \max_{i=0\ldots N} \{|c_i|\}, \quad N \text{ Anzahl der Koeffizienten} \end{aligned}$$

gesetzt. Die Wahl dieses T_0 sorgt dafür, daß alle Koeffizienten innerhalb $[-2T_0, 2T_0]$ liegen und die absoluten Werte der Koeffizienten damit aus den beiden Bereichen $[T_0, 2T_0)$ (*sp*, *sn*) oder $(-T_0, T_0)$ (*iz*, *zt*) stammen.

Das EZW-Verfahren besteht aus zwei Schritten. Der erste wird der dominante und der zweite der subordinierte Schritt genannt.

Im dominanten Schritt werden die Koeffizienten in der oben beschriebenen Weise für den *threshold* T_0 abgearbeitet. Wenn ein Koeffizient signifikant (*sp* oder *sn*) ist, so wird dieser Wert in eine Liste L_s aufgenommen. Auf jeden Fall wird ein Code für die Kategorie des Koeffizienten (*sp*, *sn*, *zt*, *iz*) ausgegeben.

Hierauf folgt der subordinierte Schritt, der die in der Liste L_s gespeicherten Werte verfeinert. Verfeinert wird mit dem Wert $\pm T_i/4$.

Für diese Verfeinerung werden alle Werte aus der Liste L_s der Reihenfolge nach mit der Rekonstruktion der entsprechenden Koeffizienten verglichen und nur die Vorzeichen der Differenz ausgegeben.

Abschließend wird der neue *threshold* T_i auf die Hälfte des ursprünglichen Wertes gesetzt:

$$T_i = \frac{T_{i-1}}{2}.$$

Nun werden die beiden Schritte solange wiederholt, bis entweder keine Informationen mehr zu übertragen sind oder eine bestimmte Datenmenge erreicht wird. Die Eigenschaften des Haar-Wavelets sorgen dafür, daß das Bild zu jeder Zeit für die Datenmenge optimal übertragen wird.

In (Barthel u. a. 1997) ist ein Versuch, das EZW-Verfahren mit fraktalen Methoden zu kombinieren; es konnten aber letztlich nicht die Kompressionsraten des Original-EZW bei gleicher Darstellungsqualität erreicht werden.

6.4 Set Partitioning in Hierarchical Trees (SPIHT)

Amir Said und William Pearlman haben 1993 eine Erweiterung von EZW vorgestellt, die auf Mengen von Bäumen beruht. Dieses Verfahren nannten sie *Set Partitioning in Hierarchical Trees* (SPIHT) (Said und Pearlman 1993; Said und Pearlman 1996). Wie bei EZW besteht dieses Verfahren aus einem dominanten und einem subordinierten Schritt und vereint die Quantisierung und die Struktur-Kodierung in einem Prozeß.

Bei SPIHT wird mit Mengen gearbeitet, die auf die Koordinaten der Koeffizienten bezogen werden. Diese Koordinaten werden aber nur zur Identifikation während des Algorithmus' gebraucht und nie explizit übertragen.

Die Mengen, mit denen gearbeitet wird, sind:

$\mathcal{O}(i, j)$**:** Direkte Söhne der Koeffizienten an der Position (i, j).

$\mathcal{D}(i,j)$**:** Alle Nachfahren des Koeffizienten an der Stelle (i,j).

$\mathcal{L}(i,j)$**:** $\mathcal{D}(i,j) - \mathcal{O}(i,j)$.

Im Gegensatz zu EZW arbeitet SPIHT immer auf 2×2-Blöcken, d.h. $\mathcal{O}(i,j)$ besteht entweder aus vier Elementen oder ist die leere Menge. Die Abhängigkeiten der Mengen und der Blöcke sind in Abbildung 33 zu sehen.

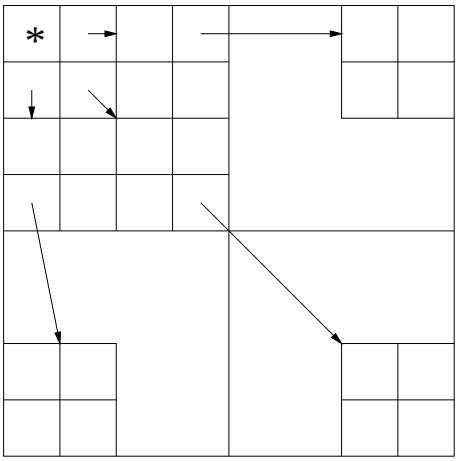


Abbildung 33: Abhängigkeiten der Blöcke bei SPIHT.

Das SPIHT-Verfahren führt eine Liste neu ein: die Liste der insignifikanten Mengen (LIS). Insgesamt hat SPIHT damit drei Listen:

LIP: Liste der insignifikanten Pixel. Insignifikant wird dabei wie bei EZW bezüglich eines *thresholds* T_i aufgefaßt.

LIS: Liste der insignifikanten Mengen. In dieser Liste werden zwei Arten von Mengen aufgenommen: $\mathcal{D}(i,j)$ (Typ A) und $\mathcal{L}(i,j)$ (Typ B).

LSP: Liste der signifikanten Pixel.

Wie bei EZW wird der *threshold* T_0 zu Beginn auf $2^{\lfloor \log c_{max} \rfloor}$ gesetzt. Zusätzlich wird dem Quantisierungswert n der Wert $\lfloor \log c_{max} \rfloor$ zugewiesen. Dann wird eine Initialisierung der drei Listen vorgenommen. LIP bleibt zunächst leer, da noch keine Koeffizienten auf Signifikanz getestet wurden. In LIP kommen die vier Koeffizienten aus dem gröbsten Levelblock (in Abbildung 33 links oben). Die LIS wird mit den Mengen $\mathcal{D}(i,j)$ der drei Koeffizienten mit Nachfolgern aus LIP gefüllt (in Abbildung 33 sind dies die Koeffizienten aus dem Block links oben ohne den mit Stern gekennzeichneten Koeffizienten).

Nach der Initialisierung kann nun das Iterieren über die dominanten und subordinierten Schritte erfolgen.

Im dominanten Schritt wird dafür zuerst die LIP abgearbeitet. Dazu wird für jedes Element der Liste ausgegeben, ob es signifikant ist. Ist es das, so wird das Vorzeichen des Koeffizienten ausgegeben, das Element aus der LIP gelöscht und an die LSP hinten angefügt.

Nach der LIP wird die LIS durchgearbeitet. Bei dieser Liste sind zwei Arten von Elementen zugelassen mit denen unterschiedlich verfahren werden muß.

Ist das Element vom Typ A, so wird erst dessen Signifikanz ausgegeben. Ist es signifikant, so werden die Koeffizienten aus $\mathcal{O}(i,j)$ auf Signifikanz getestet. Diese Entscheidung wird ebenfalls ausgegeben und im Falle der Signifikanz noch das Vorzeichen des Koeffizienten. Außerdem werden die Koeffizienten entsprechend ihrer Signifikanz an die LIP oder LSP angehängt. Dann wird noch $\mathcal{L}(i,j)$ an LIS angereiht und $\mathcal{D}(i,j)$ aus LIS gelöscht.

Handelt es sich bei dem Element um Typ B, so wird zuerst das Element auf Signifikanz getestet und diese ausgegeben. Fällt diese positiv aus, so werden die Elemente $(k,l) \in \mathcal{O}(i,j)$ als $\mathcal{D}(i,j)$ – also Typ A – an LIS angefügt und $\mathcal{L}(i,j)$ gelöscht.

Zu bemerken ist, daß die neu angefügten Elemente an LIS noch in demselben dominanten Schritt bearbeitet werden. Im subordinierten Schritt, in der die LSP abgearbeitet wird, werden allerdings nur die Elemente bearbeitet, die zu Beginn des Iterationsschrittes schon vorhanden waren.

Der subordinierte Schritt arbeitet die LSP ab. Für jedes Element aus LSP wird das n-höchste Bit übertragen.

Nun wird n um eins verringert und der *threshold* wie bei EZW halbiert ($T_i = T_{i-1}/2$). Jetzt kann die Schleife wieder bei dem dominanten Schritt fortfahren.

Wie bei EZW kann die Iteration über die beiden Schritte jederzeit abgebrochen werden, um bestimmte Anforderungen an Übertragungsvolumen oder ähnliches einzuhalten. Die Bitströme, die von SPIHT erzeugt werden, können noch zusätzlich durch einen Daten-Kodierer, wie etwa arithmetische Kodierung, komprimiert werden. Said und Pearlman weisen aber darauf hin, daß dadurch nicht mehr viel an Kompressionsrate gewonnen werden kann, der Aufwand zur Kodierung und Dekodierung aber deutlich zunimmt (Said und Pearlman 1996).

6.5 Vergleich der Struktur-Kodierungsverfahren

In diesem Vergleich werden die Verfahren EZW und SPIHT nicht weiter einbezogen, denn sie können nicht mit einem garantierten Fehler in der Ma-

ximumsnorm aufweisen. Eine entsprechende Erweiterung würde aber die guten Kompressionsraten dieser Verfahren zerstören, da bei diesen Verfahren der Quantisierer dafür sorgt, daß alle Koeffizienten gleichzeitig die gleiche Größenordnung erreichen. Der allein für die eigentliche Struktur-Kodierung notwendige Speicheraufwand liegt aber in etwa in der Größenordnung des $4m$-Verfahrens. Desweiteren zeigen die beiden Verfahren wie Transformation, Kompression, Struktur- und Daten-Kodierung optimal in einem Algorithmus vereint werden können.

Der Speicheraufwand für das $4m$-Verfahren beträgt genau ein Bit pro Knoten. Daraus ergibt sich für einen Baum der Größe n ein Bitvolumen von n. Für das Speichervolumen spielt es keine Rolle, welches Aussehen der Baum hat. Für die $2m$- und $3m$-Verfahren ist der Speicheraufwand für einen voll gefüllten Baum am größten. Es müssen dann für einen Baum der Größe n in etwa $(n+1)/2$ Informationen gespeichert werden. Die Größe dieser Information beträgt bei dem $2m$-Verfahren genau ein Bit und bei dem $3m$-Verfahren 1,5 Bit. Daraus ergibt sich Tabelle 2. Da für fast alle Koeffizienten zwei Dreiecke zu speichern sind, verdoppelt sich der Speicherverbrauch für die Koeffizienten noch einmal, daher die Namen $2m$-, $3m$- und $4m$-Verfahren.

Aus der Tabelle kann abgelesen werden, daß für allgemeine Binärbäume das $4m$-Verfahren das beste ist. Falls der Baum aber die in Sektion 6.2.2 geforderten Bedingungen erfüllt, ist das $3m$-Verfahren gegenüber dem $4m$-Verfahren leicht im Vorteil. Das $2m$-Verfahren setzt auf noch speziellere Bäume, kann diese dann aber auch erheblich besser kodieren als die anderen Verfahren.

Wenn nur kleine Teile eines Baumes weggelassen werden, kann es sinnvoller sein, einen vollen Baum anzunehmen und die wegzulassenden Knotenwerte durch Nullen zu ersetzen. Der nachgeschaltete Daten-Kodierer ist dann für die Kodierung zuständig. Daher kann an dieser Stelle keine Größe für die Struktur-Kodierung durch zu Null-Setzen angegeben werden.

Numerische Vergleiche zu den Verfahren Null-Setzen und $2m$ sind in Abschnitt 7.2, in den Tabellen 9–12 und in den Abbildungen 53–62 zu finden.

Verfahren	kodierte Strukturgröße (Bit)
$4m$	$n \cdot 1 = n$
$3m$	$\frac{n+1}{2} \cdot 1\frac{1}{2} = \frac{3}{4}(n+1)$
$2m$	$\frac{n+1}{2} \cdot 1 = \frac{n+1}{2}$

Tabelle 2: Vergleich der Struktur-Kodierungsverfahren.

7 Daten-Kodierungsverfahren

Die Daten, die nach der Kompression K und der Struktur-Kodierung übrig geblieben sind, müssen noch nach redundanten Daten abgesucht werden. Dies soll durch die Kodierung C geschehen. Wie schon im Kapitel 1 dargelegt, gibt es verschiedene Möglichkeiten, die Redundanzen aus den Daten zu filtern. Möglicherweise können auch verschiedene Verfahren kombiniert werden. Das Komprimierungsprogramm `gzip` kombiniert z. B. das LZW- und das Huffman-Verfahren, um höhere Komprimierungsraten zu erzielen.

Die unten vorgestellten Verfahren erkennen nur Redundanzen, die aus verschiedenartigen Wiederholungen bestehen. Andersartige Redundanzen sollten schon von der vorhergehenden Transformation T in Wiederholungen transformiert oder ganz entfernt worden sein. Ein Trend, in der nodalen Basis etwa eine konstant steigende Gerade, besteht in der Hierarchischen Basis nur aus Anfangs- und Endpunkten. Alle Koeffizienten auf der Geraden sind Null, da keine Änderung zu der Geraden vorhanden ist. Es ist also zu erwarten, daß die auf Wiederholungen ausgelegte Redundanzfilterung ausreicht, um im Zusammenspiel mit der Transformation T und der Kompression K gute Komprimierungsraten zu erzielen.

Ein Kodierungsverfahren kann nicht alle Daten reduzieren. Es wird immer bei einigen Eingabedaten eine Vermehrung der Daten geben. Um solche Daten für einen gegebenen Algorithmus zu finden, muß das am meisten unerwartete Zeichen als nächstes zur Bearbeitung genommen werden.

Es sollte also in jedem Programm, das die unten beschriebenen Algorithmen implementiert, möglichst eine Überprüfung stattfinden, ob eine Datenreduktion erfolgt. Wenn dies nicht der Fall ist, ist es besser die Daten unkodiert oder mit einem anderen Verfahren kodiert zu speichern.

7.1 Arten von Kodierungsverfahren

In den folgenden Abschnitten wird eine Auswahl an üblichen Kodierungsverfahren für Daten vorgestellt. Die Daten liegen dabei als Strom von Eingabedaten vor und werden kontinuierlich von vorne nach hinten abgearbeitet. Bei einigen Algorithmen ist es eventuell notwendig, diesen Strom von Daten zweimal abzuarbeiten.

7.1.1 Explizite Speicherung

Ein Exot unter den hier vorgestellten Verfahren ist die explizite Speicherung. Sie versucht eine Strukturanalyse der Daten vorzunehmen. Die explizite Speicherung sichert neben den eigentlichen Daten zusätzlich deren Position im

Strom der Eingabedaten. Das Guinness-Buch der Rekorde ist ein Beispiel für eine explizite Speicherung. Um die Leistungen aller Menschen zu speichern, wäre zu wenig Speicherplatz vorhanden. Außerdem ist es auch nicht nötig zu speichern, daß mehrere Milliarden Menschen keine zwei Meter hohe Latte überspringen können oder mehr als zehn Sekunden brauchen, um allein mit einer Küchenschere bewaffnet ein Klavier zu zerstören. Daher werden nur die Menschen erwähnt, die zu solchen Leistungen fähig sind. Von allen anderen Menschen wird angenommen, daß sie diese Leistungen nicht erbringen können. Eine solche explizite Speicherung ist auch bei dünnbesiedelten Matrizen sinnvoll, da hier viele Einträge Null sind und nur einige wenige von Null verschieden.

Beispiel:
Zu komprimierende Daten:

aaaaaabaaacaa

Kodierung:

Überall a

Außer

Position	Wert
7	b
11	c

7.1.2 Run Length Encoding (RLE)

Eine einfache Möglichkeit Daten zu reduzieren besteht darin, aufeinanderfolgende gleiche Werte zusammenzufassen. Bei Bildern mit wenigen Farben, z.B. Fax-Bildern, kommen häufig gleiche Werte mehrmals hintereinander. Hier kann es sinnvoll sein, diese Werte zu vereinen. Es wird dann die Häufigkeit gemeinsam mit dem Wert abgespeichert. Die einfachste *Run Length Encoding* (RLE) (Capon 1959; Sayood 2000) speichert vor jeden Wert zusätzlich wie viele dieser Werte hintereinander vorkommen. Wenn jetzt verschiedene Werte immer nur einmal vorkommen, dann vermehrt dieser Algorithmus die Daten nur unnötig. Ein Gegenmittel hierfür ist die Verwendung von speziellen ESCAPE-Zeichen die signalisieren, daß der nächste Wert die Anzahl der Wiederholungen darstellt. Dieses Zeichen wird nur benutzt, wenn ein Wert mehr als zwei Mal wiederholt wird. Bei der Verwendung muß darauf geachtet werden, daß das Zeichen nicht aus dem Wertebereich der eigentlichen Werte sein darf. Sollte dies doch zutreffen, so muß dieser Wert als Wiederholung kodiert werden, auch wenn er nur einmal vorkommt. Damit die Komprimierungsrate

nicht durch solche Wiederholungskodierungen unnötig verschlechtert wird, sollte das spezielle Zeichen möglichst selten in den zu komprimierenden Daten vorkommen. Um noch bessere Komprimierungsraten zu erreichen, kann auch nach Wiederholungen von Mustern gesucht werden. Bei Bildern können hierfür einfach aufeinanderfolgende Zeilen auf Gleichheit untersucht werden. Diese Zeilen- oder Musterwiederholung kann erneut durch ein spezielles Zeichen dargestellt werden. Wenn aber auf Zeilenbasis gearbeitet wird, so kann auch jeder Zeile ein Muster zugewiesen werden, wobei jedes einzelne Muster dann RLE-kodiert wird. Ein solcher Algorithmus wird z. B. in dem *RGB*-Bildformat von SGI benutzt. Für eine einfache Implementierung eines RLE-Algorithmus siehe auch Algorithmus 5.

Beispiel:
Zu kodierende Daten:

$$aaaaaabaaaca$$

Kodierung:

ohne ESCAPE-Zeichen:	$6a1b3a1c1a$
mit ESCAPE-Zeichen:	ESC$6ab$ESC$3aca$

Algorithmus 5: Run Length Encoding (RLE)

```
 1: altesZeichen=' '
 2: z = 0
 3: while ¬EOF do
 4:    neuesZeichen=leseZeichen()
 5:    if neuesZeichen=altesZeichen then
 6:       z = z + 1
 7:    else
 8:       gibAus(z)
 9:       gibAus(altesZeichen)
10:       z = 1
11:       altesZeichen=neuesZeichen
12:    end if
13: end while
```

7.1.3 Lempel-Ziv-Welsh (LZW)

Die Vorgänger des weit verbreiteten Lempel-Ziv-Welsh-Algorithmus (LZW), die LZ77/LZ78-Algorithmen wurden 1977 bzw. 1978 von Jacob Ziv und Abraham Lempel vorgestellt (Ziv und Lempel 1977; Ziv und Lempel 1978). Der

LZW-Algorithmus wurde 1984 von Terry Welsh aus den LZ7x-Varianten entwickelt (Welsh 1984).

Beim LZW-Verfahren wird ein Wörterbuch zur Kodierung der Daten benutzt. Dieses Wörterbuch wird zur Laufzeit dynamisch an die zu komprimierenden Daten angepaßt. Dies geschieht auf eine solche Art, daß das eigentliche Wörterbuch nicht abgespeichert werden muß. Es kann beim Dekomprimieren aus den Daten wiederhergestellt werden.

Das Wörterbuch wird durch eine Tabelle dargestellt. In dieser Tabelle wird jedem Wort ein Index zugeordnet. Dabei ist darauf zu achten, daß zu jedem Index genau ein Wort gehört und die Wörter nur einmal vorkommen.

Zu Anfang wird das Wörterbuch mit der möglichen Wertemenge initialisiert. Für einen 7-Bit ASCII Text wären das also alle 128 ASCII Zeichen. Diesen Wörtern wird außerdem noch ein Index zugeordnet.

Nun werden die zu komprimierenden Daten nacheinander eingelesen. Das erste Datum wird jetzt als Wort in dem Wörterbuch gesucht. Da das Wörterbuch mit dem gesamten Wertebereich der Daten initialisiert wurde, wird es natürlich auch gefunden. Jetzt werden so lange Daten eingelesen und an das aktuelle Wort angehängt, bis das so entstandene neue Wort nicht mehr in dem Wörterbuch gefunden wird. An dieser Stelle wird der Index des alten Wortes ausgegeben und das eingelesene Datum bildet das neue Wort. Außerdem wird das nicht gefundene Wort mit einem eindeutigen Index versehen und in das Wörterbuch eingefügt.

Der Algorithmus bricht ab, sobald keine Daten mehr eingelesen werden können. Beim Abbruch muß noch der Index des aktuellen Wortes ausgegeben werden. Ein Beispiel für die LZW-Kodierung findet sich in Abbildung 34.

Eine Komprimierung tritt dann ein, wenn sich Muster in dem Datenstrom häufig wiederholen. Dies ist bei Texten der Fall und hier komprimiert LZW auch besonders gut. Damit der Index später wieder eingelesen werden kann, wird bei LZW mit einer festen Bit-Länge für den Index gearbeitet. Diese Bit-Länge bestimmt die Größe des Wörterbuches. Für eine Bit-Länge von l kann das Wörterbuch $2^l - 1$ Einträge aufnehmen. Die Größe des Wörterbuches ist mitverantwortlich für die Komprimierungsrate der Daten. Je größer es ist, um so mehr Wörter, also unabhängige Muster, können in das Wörterbuch aufgenommen werden. Gleichzeitig wird aber auch der Speicherplatzbedarf für jeden Index größer. Hier muß also ein Kompromiß gemacht werden. Desweiteren können sich die Muster lokal mit dem Datenstrom verändern. Es kann also sinnvoll sein, das Wörterbuch zur Laufzeit neu zu initialisieren und es so den lokalen Eigenschaften der Daten anzupassen.

Beispiel:
Wörterbuch zu Anfang:

Index	1	2
Wert	a	b

Zu komprimierende Daten:

$$aabaabbaa$$

Ein- und Ausgabe des Algorithmus:

lese	akt. Wort	anh.	schreibe	akt. Wörterbuch
a	–	a	–	–
a	a	a	1	$3 \rightarrow aa$
b	a	b	1	$4 \rightarrow ab$
a	b	a	2	$5 \rightarrow ba$
a	a	a	–	–
b	aa	b	3	$6 \rightarrow aab$
b	b	b	2	$7 \rightarrow bb$
a	b	a	–	–
a	ba	a	5	$8 \rightarrow baa$
–	a	–	1	–

Abbildung 34: Beispiel einer LZW-Kodierung.

Algorithmus 6: Lempel-Ziv-Welsh (LZW)

```
 1: Initialisiere Wörterbuch
 2: wort=„“
 3: while ¬EOF do
 4:   neuesZeichen=leseZeichen()
 5:   if wort+neuesZeichen ∉ Wörterbuch then
 6:     füge wort+neuesZeichen in Wörterbuch ein
 7:     gibAus(Code für Wort)
 8:     wort=neuesZeichen
 9:   else
10:     wort=wort+neuesZeichen
11:   end if
12: end while
```

7.1.4 Huffman-Kodierung

Bei der Huffman-Kodierung (Huffman 1951; Press u. a. 1992) wird wie bei der LZW-Kodierung ein Wörterbuch zur Komprimierung der Daten benutzt. Das Wörterbuch besteht allerdings nur aus den eigentlichen Daten und nicht noch zusätzlich aus zusammengesetzten Wörtern. Der Index ist bei Huffman ein eindeutiger Bitcode, wobei unter eindeutig gemeint ist, daß kein Bitcode in einem anderen enthalten sein darf. Dies ist wichtig, da bei der Huffman-Kodierung die Länge der Bitcodes variabel ist. Der Algorithmus muß aber beim Dekodieren der Daten feststellen können, daß ein Bitcode vollständig gelesen wurde und der nächste eingelesen werden kann.

Die Länge des Bitcodes wird durch die Häufigkeit des zugehörigen Datums bestimmt. Je häufiger ein Wert ist, um so kürzer sollte sein Bitcode sein.

Eine einfache Art an einen eindeutigen Bitcode zu gelangen, der auch die Häufigkeit der Werte berücksichtigt, ist das Aufbauen eines Binärbaumes. Der Baum besteht aus zwei verschiedenen Arten von Knoten. Die erste Art hat keine Sohnäste. In diesen Blättern werden die Werte und ihre Häufigkeit gespeichert. Die andere Art von Knoten hat genau zwei Söhne. In diesen Zweigknoten werden keine Werte gespeichert, außer der Summe der Häufigkeit ihrer Sohnknoten. Der Aufbau eines solchen Baumes gelingt recht einfach, wenn die Werte nach ihrer Häufigkeit sortiert vorliegen.

Zu Anfang gibt es nur Knoten mit Werten und Häufigkeiten, aber ohne Sohnäste.

Aus den zur Verfügung stehenden Knoten werden die zwei seltensten heraus gesucht, also die Knoten mit der geringsten Häufigkeit. Diese werden an einen neuen Zweigknoten als Sohnknoten angehängt. Der neue Knoten erhält

die Summe der Häufigkeit der Sohnknoten als eigene Häufigkeit zugeordnet.

Das Verbinden von zwei Knoten wird solange fortgeführt, bis nur noch ein Knoten zur Auswahl steht. Dieser Knoten ist nun die Wurzel des gesuchten Baumes, aus dem die Bitcodes für die Wörter des Huffman-Wörterbuches generiert werden.

Um den Bitcode eines Wortes zu ermitteln, wird der direkte Weg von der Wurzel bis zum Wert verfolgt. Einer linken Astverzweigung wird eine Null, einer rechten eine Eins zugeordnet. Je tiefer ein Wert im Baum hängt, um so länger ist auch sein Bitcode. Abbildung 35 zeigt ein Beispiel für eine Huffman-Kodierung mit vorgegebenen Statistiken.

Das Kodieren der Daten erfolgt nun durch Ausgabe des Bitcodes für jeden einzelnen Wert.

Damit die Bitcodes beim Dekomprimieren wieder den Worten zugeordnet werden können, muß das Wörterbuch samt Bitcodes mit den komprimierten Daten abgespeichert werden. Alternativ kann auch mit vorher vereinbarten Wörterbüchern gearbeitet werden. Das Dekodieren besteht nun einfach darin, die Bitcodes einzulesen und die zugehörigen Werte auszugeben.

Ein verwandtes Beispiel für ein vereinbartes Wörterbuch ist der Morse-Code. Der Morse-Code ist aber kein Huffman-Code, da einige Bitcodes in anderen enthalten sind. Die einzelnen Bitcodes müssen also bei der Übertragung durch ein extra vereinbartes Zeichen getrennt werden.

7.1.5 Arithmetische Kodierung

Bei der Huffman-Kodierung wird jeder Wert durch einen Bitcode mit ganzzahliger Länge dargestellt. Die arithmetische Kodierung (Witten u. a. 1987) kann hier noch bessere Kompressionsraten erreichen, da sie auch gebrochenzahlige Längen für Bitcodes erlaubt. Dies wird dadurch erreicht, daß der gesamte Datenstrom als ein Intervall aus dem Bereich $[0, 1)$ dargestellt wird. Dabei reicht die Übermittlung eines Repräsentanten aus diesem Intervall als Darstellung aus.

Das entsprechende Intervall für den Datenstrom wird auf folgende Art bestimmt: Zunächst wird wie bei Huffman eine Tabelle mit den Werten und ihrer Häufigkeitsverteilung angelegt. Dann wird jedem Wert ein Teilintervall aus $[0, 1)$ zugeordnet, wobei die Größe des Teilintervalls proportional zur Wahrscheinlichkeit des Wertes ist. Bei der Vergabe der Teilintervalle ist darauf zu achten, daß sie sich nicht überlappen und die Vereinigung das gesamte Intervall $[0, 1)$ überdeckt.

Zum Kodieren wird nun das Startintervall auf $[0, 1)$ gesetzt und die Daten werden eingelesen. Jeder Wert schränkt das Intervall anhand seines eigenen Teilintervalls weiter ein. Dabei wird das Teilintervall, welches auf $[0, 1)$ ska-

Beispiel:
Werte und ihre Häufigkeiten

a	b	c
10	2	5

Bau des Baumes:

Schritt	Felder mit Teilbäumen
0	A:0,1 B:0,4 C:0,5
1	∑:0,5 C:0,5 A:0,1 B:0,4
2	∑:1,0 ∑:0,5 C:0,5 A:0,1 B:0,4

Wörterbuch:

Wert	Bitcode
a	1
b	00
c	01

Sequenz:

Eingabe	Ausgabe
aabac	1100101

Abbildung 35: Beispiel einer Huffman-Kodierung mit vorgegebenen Statistiken.

Algorithmus 7: Huffman

1: Sammle Häufigkeiten für alle Werte und fasse beides in Knoten zusammen. Diese Knoten werden in einem Feld gespeichert.
2: **while** Mehr als ein Knoten in dem Feld **do**
3: Wähle Knoten1 und Knoten2 als die Knoten mit der geringsten Häufigkeit
4: KnotenNeu.links=Knoten1
5: KnotenNeu.rechts=Knoten2
6: KnotenNeu.häufigkeit=Knoten1.häufigkeit+Knoten2.häufigkeit
7: Lösche Knoten1 und Knoten2 aus dem Feld und hänge stattdessen KnotenNeu ein
8: **end while**
9: WurzelKnoten=Feld[0]
10: codebuch=neues Feld
11: rekCodebuch(codebuch, WurzelKnoten, 1)
12: Speichere Codebuch
13: **while** ¬EOF **do**
14: neuesZeichen=leseZeichen()
15: gib Code für neuesZeichen aus
16: **end while**

Algorithmus 8: Rekursiver Aufbau des Codebuches

1: Aufruf durch `rekCodebuch`(*codebuch*, *Knoten*, *zweig*)
2: **if** Knoten hat Söhne **then**
3: rekCodebuch(codebuch, Knoten.links, zweig $\cdot\, 2 + 0$)
4: rekCodebuch(codebuch, Knoten.rechts, zweig $\cdot\, 2 + 1$)
5: **else**
6: codebuch[Knoten.wert]=zweig
7: **end if**

liert ist, auf das aktuelle Intervall projiziert. So schränkt jedes Datum das aktuelle Teilstück ein (siehe hierzu auch Abbildung 36). Zum Schluß muß noch eine Zahl als Repräsentantin aus dem Intervall übertragen werden. Damit die Zahl nicht alle Datenströme darstellt, die mit den gleichen Daten anfangen, muß noch ein spezielles EOM-Zeichen (*end of message*) in die Wertetabelle mit aufgenommen werden. Dieses Zeichen sollte nur eine geringe Wahrscheinlichkeit haben und damit auch nur ein kleines Teilintervall zugewiesen bekommen.

Das größte Problem bei der arithmetischen Kodierung ist allerdings, daß die heutigen Rechner und Programmiersprachen standardmäßig keine beliebig genaue Darstellung von Fließkommazahlen bereitstellen. Es muß also dafür gesorgt werden, daß die Repräsentantin übertragen werden kann.

Der Ansatz, der bei der arithmetischen Kodierung gemacht wird, besteht darin, daß die Ober- und Untergrenze des Intervalls gespeichert wird. Diese Grenze wird in den oberen (höherwertigen) Bits auf Übereinstimmungen überprüft. Stimmen die oberen Bits überein, so werden diese ausgegeben und die beiden Grenzen um die Anzahl der ausgegebenen Bits nach links verschoben (*Bitshift*). Außerdem werden noch die gleiche Anzahl niederwertiger Bits an die Grenzen passend angefügt.

Auf diese Weise ist es einfach möglich, die Ausgabe-Kodierung auch auf andere Bitlängen als die üblichen 8-Bit zu bringen.

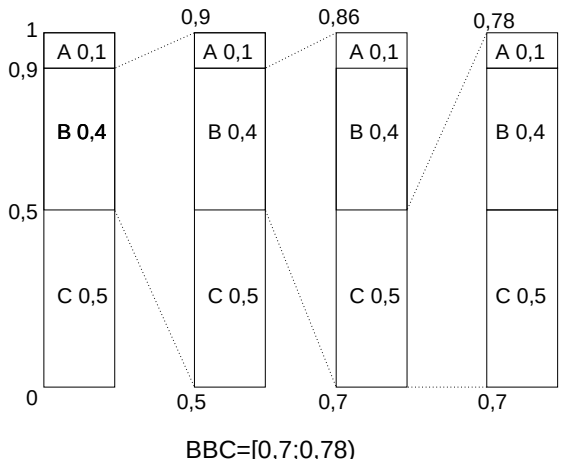


Abbildung 36: Intervallzuweisung bei der Arithmetischen Kodierung.

7.1.6 Dynamisches mehrstufiges Wörterbuch

Bisher wurde sowohl bei der Huffman-Kodierung als auch bei der arithmetischen Kodierung von einem statischen Wörterbuch ausgegangen. Dies bringt ein paar Nachteile mit sich. Einmal muß vor dem Kodieren und dem Dekodieren das gesamte Wörterbuch bekannt sein. Wenn also ein den Daten angepaßtes Wörterbuch benutzt werden soll, so muß dieses mit den Daten gespeichert werden. Zum anderen muß die Wahrscheinlichkeitsverteilung der Daten vor dem Kodieren bekannt sein. Ist dies nicht der Fall, so ist ein weiterer Durchlauf nötig, um diese zu bestimmen. Auch kann sich ein statisches Wörterbuch den lokalen Änderungen in der Wahrscheinlichkeitsverteilung der Werte im Datenstrom nicht anpassen. Desweiteren ist in Datenströmen häufig eine Abhängigkeit der Wahrscheinlichkeit für die verschiedenen Werte von den vorhergehenden zu beobachten. So ist in deutschen Texten z. B. nach einem *„q“* sehr häufig ein *„u“* zu finden. Um diese Abhängigkeiten modellieren zu können, wird mit einer mehrstufigen Datenstruktur gearbeitet. Diese Abhängigkeiten werden auch als Markov-Modelle bezeichnet (Shannon 1948; Sayood 2000).

Jetzt erhält jeder Wert nicht nur einen Bitcode bzw. ein Teilintervall, er wird zusätzlich auch noch in seinem Kontext betrachtet. Das heißt, jeder Wert erhält einen Verweis auf eine Tabelle mit der Wahrscheinlichkeitsverteilung für die nachfolgenden Werte in Bezug auf sich und eventuell seine Vorgänger. Damit die Modellierung der Wahrscheinlichkeiten nicht unendlich viel Speicher verbrauchen kann, werden die Tabellen nur n Stufen tief aneinandergehängt. Um dynamisch arbeiten zu können, wird jedes Wörterbuch zu Anfang nur mit zwei speziellen Zeichen gefüllt (Knuth 1985; Vitter 1987; Sayood 2000). Einmal das schon bekannte EOM-Zeichen und zum anderen ein Escape-Zeichen, das benutzt wird, wenn im Datenstrom ein in dieser Tabelle noch nicht bekannter Wert vorkommt. An dieser Stelle wird das Escape-Zeichen gesendet und die Kontextstufe um Eins verringert, wenn sie größer Null ist. Ist die Kontextstufe dann noch größer Null, so wird der Wert in der jetzt aktuellen Tabelle gesucht. Wird er gefunden, so wird er anhand seines Teilintervalls kodiert. Falls er nicht gefunden wird, wird erneut das Escape-Zeichen kodiert und die Kontextstufe um Eins erniedrigt. Dies wird so lange wiederholt, bis die Kontextstufe gleich Null ist. Findet sich das Datum auch hier nicht, so wird das Escape-Zeichen ein letztes Mal kodiert und im Anschluß direkt der Wert übertragen. Daraufhin wird der Wert in alle Kontexte eingetragen, aus denen per Escape-Zeichen gewechselt wurde. Zusätzlich werden nach jedem kodierten Wert die kontextabhängigen Häufigkeitsverteilungen neu angepaßt.

7.1.7 Burrows-Wheeler Transformation (BWT)

Eine weitere Kodierungsmethode ist 1994 von Burrows und Wheeler vorgestellt worden (Burrows und Wheeler 1994; Nelson 1996). Die nach ihnen benannte Burrows-Wheeler Transformation (BWT) arbeitet auf Blöcken von Daten. Diese Datenblöcke werden transformiert und dann erst gepackt. Deshalb wurde dieses Verfahren von den Entwicklern auch als *block-sorting lossless data compression algorithm* bezeichnet.

Die BWT kann als kontextbasiertes Verfahren verstanden werden. Als Kontext dienen aber nicht die vorhergehenden Zeichen – wie bei den oben vorgestellten dynamischen Wörterbüchern oder LZW – sondern die nachfolgenden Zeichen. Dies ist möglich, da die BWT auf Blöcken anstatt auf einem Datenstrom arbeitet. Je größer die zu bearbeitenden Blöcke sind, um so besser ist i. allg. auch die Komprimierung.

Die BWT geht dabei folgendermaßen vor: Zuerst wird für einen Block der Länge N ($N \in \mathbb{N}$) eine Matrix der Größe $N \times N$ erstellt. In dieser Matrix sind der Ausgangsblock und alle Rotationen des Ausgangsblockes in den Zeilen angeordnet (siehe Abbildung 37). Die Zeilen der Matrix werden dann nach dem jeweiligen Zeileninhalt sortiert (siehe Abbildung 38). Die erste Spalte der so erhaltenen Matrix wird mit **F** (first) und die letzte Spalte mit **L** (last) bezeichnet. Die Spalte **F** enthält jetzt alle Werte des Ausgangsblockes in sortierter Reihenfolge. In der Spalte **L** sind jeweils deren Vorgänger enthalten. Die Ausgabe der BWT besteht aus der Spalte **L** und einem Index i, der die momentane Position des Ausgangsblockes in der sortierten Matrix angibt.

S_0	B	L	U	B	B
S_1	L	U	B	B	B
S_2	U	B	B	B	L
S_3	B	B	B	L	U
S_4	B	B	L	U	B

Abbildung 37: BWT Ausgangsmatrix mit Originalblock in der obersten Zeile.

Zur Dekodierung muß aus der Spalte **L** und dem Index i die gesamte Matrix wieder aufgebaut und der Ausgangsblock rekonstruiert werden.

Zur Rekonstruktion kann zunächst die Spalte **F** auf einfache Weise wieder hergestellt werden. Die Spalte **F** enthält die Spalte **L** in sortierter Reihenfolge (siehe Abbildung 39). Mittels eines Transformationsvektors **T** und den nun bekannten Spalten **F** und **L** kann die komplette Matrix wieder hergestellt werden. Dazu werden die Werte aus der Spalte **F** von oben nach unten in der Spalte **L** gesucht. In den Vektor **T** wird als Wert der Index der obersten

		F				**L**
	S_3	B	B	B	L	U
	S_4	B	B	L	U	B
$i = 2 \Rightarrow$	S_0	B	L	U	B	B
	S_1	L	U	B	B	B
	S_2	U	B	B	B	L

Abbildung 38: BWT nach Zeilen sortierte Matrix.

Zeile aus Spalte **L** eingetragen, die den gesuchten Wert aus Spalte **F** aufweist und noch nicht als Index in **T** eingetragen wurde (siehe Abbildung 39).

		F				**L**	
	$S_?$	B	?	?	?	U	$\mathbf{T}[i] \Rightarrow i$
	$S_?$	B	?	?	?	B	$\mathbf{T}[1] \Rightarrow 2$
$i = 2 \Rightarrow$	S_0	B	?	?	?	B	$\mathbf{T}[2] \Rightarrow 3$
	$S_?$	L	?	?	?	B	$\mathbf{T}[3] \Rightarrow 4$
	$S_?$	U	?	?	?	L	$\mathbf{T}[4] \Rightarrow 0$

Abbildung 39: BWT Rekonstruktion der Ausgangsmatrix.

Der ursprüngliche Datenblock kann jetzt ausgegeben werden, indem ab Zeile i die Werte aus der Spalte **F** ausgegeben werden und i auf den Wert $\mathbf{T}[i]$ gesetzt wird[1].

Bisher hat noch keine Komprimierung stattgefunden. Das oben beschriebene Vorgehen ist nur die Transformation, die dafür sorgen soll, daß die Spalte **L** sich gut komprimieren läßt.

In der Spalte **L** sind die Vorgänger der kompletten Zeile enthalten. Wenn die Nachfolger von **L**, also Spalte **F** und folgende, in einem ähnlichen Kontext stehen, so sind diese Zeilen in der Matrix nahe beisammen. In deutschen Texten werden Zeilen, in deren Spalten **L** ein *„q“* steht, sehr wahrscheinlich mit einem *„u“* anfangen und deshalb relativ nahe beisammen stehen.

Eine solche Anordnung der Spalte **L** kann nun z. B. mit RLE komprimiert werden. Burrows und Wheeler (Burrows und Wheeler 1994) schlagen allerdings vor, die Spalte **L** mit einem *move to front*-Kodierer (MTF) zu bearbeiten und dessen Ausgabe dann mit einem Entropie-Kodierer, wie z. B. einem Huffman- oder einem arithmetischen Kodierer, zu bearbeiten.

[1] Daß der Transformationsvektor **T** auf diese Weise erstellt werden kann, hängt mit der Sortierung der Ausgangsmatrix zusammen.

Ein MTF-Kodierer arbeitet mit einem Vektor der Größe aller möglichen Werte. In diesem Vektor stehen anfangs alle Werte in ihrer natürlichen Reihenfolge. Wird jetzt ein Zeichen kodiert, so gibt der Kodierer dessen Index in dem Vektor aus und stellt das Zeichen an den Anfang des Vektors. Auf diese Weise ist die Ausgabe des MTF-Kodierers sehr stark auf die Null und Eins konzentriert, wenn viele gleiche Werte wie bei der BWT hintereinander kodiert werden.

Eine so erhaltene Verteilung der Indizes ist dann ideal mit Entropie-Kodierern zu komprimieren.

Als letztes ist noch anzumerken, daß die Matrix der BWT nie wirklich aufgestellt werden muß. Wäre das der Fall, so würde schnell der Hauptspeicher knapp, denn schon für einen 500 KByte großen Block würde die Matrix auf 500×500 KByte = 250 MByte anwachsen.

7.2 Vergleich der Kodierungsverfahren

Bei einem Vergleich der Daten-Kodierungsverfahren muß auf die Charakteristik der Eingabedaten Rücksicht genommen werden. Jedes der oben vorgestellten Verfahren ist für bestimmte Dateneigenschaften entwickelt worden.

Für die Datenkompression, wie sie in dieser Arbeit vorgestellt wird, ist die Daten-Charakteristik durch die Transformation T, die Kompression K und die Struktur-Kodierung C_s bestimmt.

Wenn die Struktur-Kodierung aus Null-Setzen besteht, so wird bei zunehmendem Schwellenwert ε der Anteil an Nullen im Datenstrom stark ansteigen. Wenn hingegen die anderen Baum-Kodierer eingesetzt werden, so wird bei zunehmendem Schwellenwert ε die Wertigkeit der Daten immer zufälliger.

Da die Knoten in dieser Arbeit rekursiv abgearbeitet werden, also jedes neue Datum im (Ausgabe-)Strom auch einen Levelsprung beinhaltet, haben aufeinanderfolgende Daten sehr wahrscheinlich auch andere Größenordnungen. Wiederkehrende Datenmuster werden wahrscheinlich nicht auftreten.

Die RLE arbeitet nur bei wenigen unterschiedlichen Werten gut, sie wird deshalb bei den hier anfallenden Daten überfordert sein. Außerdem müßten aufeinanderfolgende Daten gleiche Werte haben, dies ist aber wegen der rekursiven Abarbeitung nicht gegeben.

Auch die explizite Speicherung wird für die anfallenden Daten nicht gut geeignet sein. Wenn ein Bild durch explizite Speicherung effizient abgespeichert werden kann, bedeutet das, daß es entweder von vornherein sehr wenige Informationen enthielt und die Transformation T versagt hat oder daß der vorgegebene Schwellenwert ε so hoch eingestellt wurde, daß keine Informationen übrig sind.

Die Huffman-Kodierung braucht zwar in der statischen Variante zwei Durchläufe, kann dann aber auch für die eher zufällige Reihenfolge der Werte gute Kompression erreichen. Zudem kann die Huffman-Kodierung für den kompletten Wertebereich mit einem Codebuch auskommen. Für die Huffman-Kodierung spricht auch, daß sie recht einfach zu implementieren ist und eine niedrige Komplexität hat.

Das gleiche gilt für die arithmetische Kodierung, die sogar noch bessere Ergebnisse liefern kann, da die Codewörter für die unterschiedlichen Daten auch eine gebrochene Bitlänge erlauben. Allerdings ist die Implementation der arithmetischen Kodierung nicht ganz so einfach, wie die der Huffman-Kodierung. Unterschiede zwischen Huffman-Kodierung und arithmetischer Kodierung können sich durch die Implementation der Wörterbücher ergeben.

Die LZx-Kodierer brauchen Muster im Datenstrom, die sich wiederholen. Diese müssen aus Rücksicht auf die Speicheranforderungen im Datenstrom möglichst nahe zusammenliegen. Dies wird aber bei den hier anfallenden Daten vermutlich nicht der Fall sein.

Die BWT arbeitet wie die LZx-Verfahren auf Abhängigkeiten im Datenstrom. Da die BWT aber mit viel größeren Datenblöcken arbeitet als die LZx-Verfahren, kann diese die Daten vermutlich besser komprimieren.

8 Erweiterungen

Bisher wurden nur Graustufen-Bilder auf quadratischen Gebieten behandelt. Um auch beliebige Gebiete und farbige Bilder bearbeiten zu können, sind einige Änderungen der Algorithmen nötig. Diese Erweiterungen werden im folgenden Kapitel behandelt.

8.1 Rechteckige Gebiete

Rechteckige Gebiete können durch Quadrate überdeckt werden. Dabei gibt es zwei Grundarten, die beliebig gemischt werden können. Die erste Möglichkeit ist, das Rechteck durch ein großes Quadrat zu überdecken (Abbildung 40 (a)). Die zweite Möglichkeit bettet in das Rechteck das nächstkleinere Quadrat ein und füllt den verbleibenden Platz rekursiv durch kleine Quadrate auf (Abbildung 40 (b)).

Die erste Möglichkeit hat den Vorteil, daß nur ein Quadrat bearbeitet werden muß. Dafür muß aber zusätzlicher Speicher reserviert werden, der keinem Teil des eigentlichen Bildes entspricht. Dieser Platz muß außerdem geeignet gefüllt werden.

Die zweite Möglichkeit braucht keinen zusätzlichen Speicherplatz, da wirklich nur das Bild überdeckt wird. Dafür hat diese Methode den Nachteil der Berechnung vieler Quadrate. Dies kann bei der Kompression zu Einbußen führen.

Allerdings kann auch eine Mischstrategie gewählt werden. Hierbei wird das größte im Rechteck noch enthaltene Quadrat genommen. Dann werden die kleinsten Quadrate genommen, die den restlichen Platz des Rechteckes überdecken und eventuell über den Rand hinausragen (Abbildung 40 (c)). Hier muß der zusätzliche Platz zwar auch wie in der ersten Methode reserviert und geeignet aufgefüllt werden, aber der zusätzliche Speicheraufwand ist sehr viel geringer. Gegenüber der zweiten Methode müssen weniger Quadrate bearbeitet werden, was sich in besseren Kompressionsraten niederschlägt.

Eine weitere Möglichkeit rechteckige Gebiete bearbeiten zu können, ist das Strecken der Träger der Basisfunktion (Abbildung 40 (d)). Auf diese Art wird statt der quadratischen Grundfläche direkt die zu bearbeitende rechteckige Grundfläche benutzt. Im rekursiven Algorithmus muß dann noch die Abbruchbedingung für die Rekursion entsprechend angepaßt werden.

Auf den feineren Leveln kann es passieren, daß der Mittelpunkt der Diagonalen nicht mehr einem Feld zugeordnet werden kann, da das zugrundeliegende Rechteck zu sehr gestreckt ist. In diesen Fällen muß die Baumstruktur erweitert werden und dieses Rechteck in mehrere eher quadratische Rechtecke unterteilt werden.

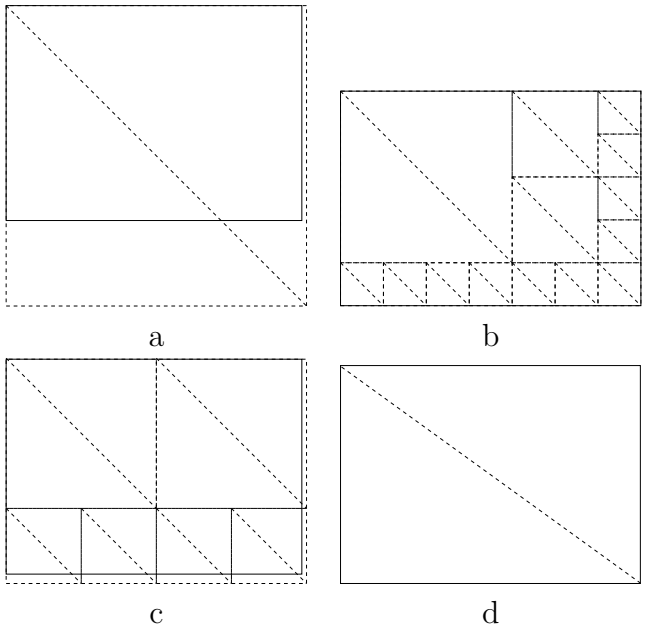


Abbildung 40: Links unten und obere Reihe: Mögliche Überdeckungen von Rechtecken durch Quadrate. Rechts unten: Streckung der Basisfunktionen.

8.2 Beliebige Gebiete

Rechteckige Gebiete tauchen zwar in vielen Gebieten alleine schon wegen der verwendeten Meßverfahren wie Photographie oder dergleichen auf. Häufig sind aber komplexere Gebiete innerhalb der Rechtecke die Gebiete des eigentlichen Interesses.

Bei einer Satellitenaufnahme sollen eventuell nur die Landmassen vermessen werden, bei einem CT-Bild interessiert nur der zu vermessende Körper.

Damit für die uninteressanten Informationen nicht auch noch Speicherplatz verschwendet wird, ist es wünschenswert, mit beliebigen Gebieten arbeiten zu können, innerhalb deren die Informationen erhalten bleiben.

Damit beliebige Gebiete abgespeichert werden können, muß ein etwas anderes Datenformat benutzt werden. Bisher wurden die Dreiecke der Triangulierung entlang einer raumfüllenden Kurve abgespeichert. Wenn allerdings Gebiete ausgelassen werden sollen, so werden Dreiecke entlang der raumfüllenden Kurve weggelassen. Dies kann auf zwei unterschiedlichen Wegen geschehen.

Die erste Möglichkeit besteht darin, die Kurve weiterlaufen zu lassen mit allen Leveländerungen der Triangulierung und zusätzlich die Anzahl und die Position des jeweils ersten ausgelassenen Dreieckes eines Blocks der über-

sprungenen Dreiecke zu behalten, um später bei der Datenspeicherung die entsprechenden Dreiecke einfach auslassen zu können.

Bei der zweiten Methode wird die Position des ersten Dreieckes eines auszulassenden Blockes mitsamt der Länge des Blockes und dem Level für das erste Dreieck nach dem Block bei der Struktur-Kodierung mitgespeichert. Hierbei kann der neue Level entweder relativ oder absolut gespeichert werden.

Beide Methoden haben Vor- und Nachteile. Bei der ersten Methode kann das $2m$-Verfahren zur Struktur-Kodierung unverändert übernommen werden. Als weitere Struktur müssen dann nur noch die Anfänge und Länge der auszulassenden Blöcke gespeichert werden. Wenn große Gebiete ausgelassen werden, kann aber die $2m$-Struktur-Kodierung zu viel Platz verbrauchen, da diese alle Informationen über die Lage der auszulassenden Dreiecke mit abspeichert. Eine Unterbrechung der Kurve ist hier also wünschenswert. Die zweite Methode erlaubt zwar eine derartige Unterbrechung, aber hier besteht das Problem, das erste Dreieck nach dem auszulassenden Block zu finden. Entweder muß die Position explizit gespeichert werden oder die Länge der Kurve muß anhand des feinsten Levels der Triangulierung bestimmt werden.

8.3 Fokussierung

Anstatt ganze Gebiete mittels beliebiger Gebiete wegzulassen, kann es auch erwünscht sein, bestimmte Gebiete mit unterschiedlicher Genauigkeit abzuspeichern bzw. zu selektieren. Dies ist z. B. für Flugsimulatoren interessant. Bei einer solchen Anwendung wird besonderer Wert auf Genauigkeit in den Bereichen gelegt, auf die der Anwender gerade achtet. Bereiche außerhalb des Fokusses können hingegen ungenauer gezeichnet werden.

Die Fokussierung kann zusätzlich dazu dienen, einen Überblick zu schaffen und gleichzeitig nicht zu sehr von den eigentlich wichtigen Daten abzulenken. Dies wird häufig bei Wetterkarten im Fernsehen angewandt, bei denen nur das aktuell besprochene Land genau gezeichnet ist und die umliegenden Gebiete ungenauer dargestellt werden.

Um einen solchen Fokus zu implementieren, muß nur der Fehlerbeschränkungs-Algorithmus entsprechend der Vorlage des Fokus angepaßt werden. Dies kann z. B. mit einer einfachen Multiplikation geschehen. Zur Verhinderung von hängenden Knoten muß dies allerdings vor der Saturation geschehen. Der Fehlerwert wird mit einem Fokuswert multipliziert, wobei eine Eins bedeutet, daß der Fehlerwert nicht verändert werden soll und eine Null dafür sorgt, daß der Fehlerwert an dieser Position nicht beachtet wird und das entsprechende Dreieck vergröbert werden kann.

8.4 Farbe

Bisher wurden die Daten immer nur als Grauwerte eines Bildes angesehen. Das bedeutet, daß jeder Wert eines Bildpunktes dessen Helligkeit repräsentierte. Dabei wurde davon ausgegangen, daß die Helligkeitswerte normalerweise aus dem Bereich $[0, 255]$ stammen. Um mit den vorgestellten Algorithmen auch Farbbilder verarbeiten zu können, muß zuerst eine brauchbare Farbdarstellung der Bilder im Computer gefunden werden.

Farben setzen sich aus Lichtwellen unterschiedlicher Wellenlänge und Intensität zusammen. Es würde also Sinn machen, die stärksten Intensitäten der entsprechenden Wellenlängen zusammen als eine Annäherung an die entsprechende Farbe anzunehmen. Die Farbrezeptoren des menschlichen Auges sind für die komplette Farbvorstellung zuständig. Da der Mensch nur drei verschiedene Farbrezeptoren hat, reicht es aus, nur drei Farbkomponenten als Näherung einer Farbe zu bestimmen. Aus dem gleichen Grund müssen es aber auch mindestens drei Komponenten sein, wenn alle Farben dargestellt werden sollen, die ein Mensch sehen kann.

Bei den Farbmodellen kann grundsätzlich zwischen additivem und subtraktivem Farbmodell unterschieden werden.

Das subtraktive Farbmodell geht davon aus, daß aus einem alle Farben enthaltenen Lichtstrahl einzelne Farben absorbiert werden und die reflektierten Lichtanteile im Auge als Farbe erkannt werden. Dieses Farbmodell wird vor allem bei Druckern und beim Malen verwendet.

Das additive Farbmodell, das in diesem Abschnitt benutzt wird, beruht hingegen auf einer Mischung verschiedenfarbiger Lichtquellen; es wird z. B. bei RGB-Monitoren verwandt.

8.4.1 RGB-Farbmodell

Ein übliches Farbmodell mit drei Komponenten ist das RGB-Farbmodell (Poynton 1997). Es zählt zu den additiven Farbmodellen. Die einzelnen Komponenten dieses Farbmodells sind *rot*, *grün* und *blau* (RGB). Jeder moderne Farbmonitor arbeitet mit diesem Modell.

Soll ein Bild in diesem Farbmodell komprimiert werden, so reicht es aus, jede Komponente einzeln zu bearbeiten. Der Fehler ist dann in jeder Farbkomponente gleich groß. Übliche Werte für die einzelnen Farbkomponenten liegen im Bereich $[0, 255]$.

8.4.2 YC_bC_r-Farbmodell

Ein anderes additives Farbmodell ist das YC_bC_r-Farbmodell. Dieses Modell geht von einer Helligkeitskomponente Y aus. Diese Komponente wird auch

als Luminanz bezeichnet. Die Helligkeit einer im RGB-System angegebenen Farbe errechnet sich nach (Poynton 1997) durch:

$$Y = 16 + 65{,}481 \cdot R + 128{,}553 \cdot G + 24{,}966 \cdot B$$

Die C_b- und C_r-Komponenten sind als Differenz zur Luminanz definiert. Die komplette Formel zur Errechnung der YC_bC_r-Komponenten aus RGB-Daten lautet:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65{,}481 & 128{,}553 & 24{,}966 \\ -37{,}797 & -74{,}203 & 112{,}000 \\ 112{,}000 & -93{,}786 & -18{,}214 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Die Wertebreiche der einzelnen Komponenten sind dabei wie folgt:

$$Y \in [16, 253]; \qquad C_b, C_r \in [16, 240]; \qquad R, G, B \in [0, 1]$$

Diese Wertebereiche wurden gewählt, um für die Signalverarbeitung noch Spielraum in beide Richtungen der Skala zu lassen.

Der YC_bC_r-Farbraum wird bei der JPEG-Kompression verwandt, wobei die Farbkomponenten unterschiedlich wichtig genommen werden. Das menschliche Auge ist besonders anfällig für Fehler in der Luminanz, also der Y-Komponente. Farbfehler hingegen fallen nicht so stark auf. JPEG erreicht diese Gewichtung durch eine Einschränkung der Auflösung bei den C_b- und C_r-Komponenten.

Ein historisches Beispiel für eine Übertreibung dieser Methode sind alte handkolorierte Filme. Bei diesen wurde der Film in Schwarz-Weiß aufgenommen. Später wurde in diese Luminanz-Komponente noch grob von Hand Farbe in die Kopien gemalt.

Bei dem hier vorgestellten Algorithmus reicht es, die Fehlertoleranz für die Werte einzeln auszuwählen. Es muß dabei beachtet werden, daß die Fehlernorm im YC_bC_r-Farbraum nicht gleich der Fehlernorm im RGB-System ist. Soll der Fehler in RGB angegeben werden, so müssen die Fehlertoleranzen entsprechend angepaßt werden.

Wenn dazu auf den oben genannten Spielraum verzichtet wird und der Wertebereich für alle Farbkomponenten auf $[0, 255]$ vereinheitlicht werden soll, so lautet die Rücktransformation von YC_bC_r nach RGB:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1{,}000 & 0{,}000 & 1{,}402 \\ 1{,}000 & -0{,}344 & -0{,}714 \\ 1{,}000 & 1{,}772 & 0{,}000 \end{bmatrix} \cdot \left(\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} - \begin{bmatrix} 0{,}000 \\ 127{,}000 \\ 127{,}000 \end{bmatrix} \right)$$

Der Fehler soll im RGB-Farbraum in jeder einzelnen Farbkomponente auf ε beschränkt sein, also ergibt sich ein Gleichungssytem mit drei Gleichungen

und drei Unbekannten. Daraus ergibt sich, daß bei gleicher Wertung aller Farbkomponenten aus dem YC_bC_r-Raum, ein $\varepsilon_{YC_bC_r}$ als Fehlerschranke im YC_bC_r-Raum eingesetzt werden muß mit:

$$\varepsilon_{YC_bC_r} = \frac{\varepsilon}{2{,}722}.$$

9 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der verschiedenen Kombinationen der Verfahren vorgestellt.

9.1 Tensorprodukt-Ansatz

Für einen Vergleich der verschiedenen Fehlerbeschränkungs-Methoden im Tensorprodukt-Ansatz wurden die Bilder der Abbildungen 56–58 aus Abschnitt A.1 verwandt.

In den Abbildungen 41 und 42 sind die dekomprimierten Bilder und die Differenzbilder zum Original *X-Wing* abgebildet. Als Schwellenwert ε wurde 64 gewählt. Die Kompressionsergebnisse und realen Fehler sind in den folgenden Tabellen 3–8 zu sehen. Aus diesen Tabellen geht hervor, daß die a priori-Beschränker die schlechtesten Kompressionsraten haben. Dies ist nicht verwunderlich, da diese keine Eigenheiten der Bilder ausnutzen können. Das Orts- und Level-abhängige Verfahren nutzt zwar den Fehler besser aus als das Orts-unabhängige, kann aber nur wenig bessere Kompressionsraten erzielen.

Die adaptiven Verfahren sind allesamt besser als die a priori-Fehlerbeschränkungs-Varianten, nutzen aber den vorgegebenen Fehler auch nicht komplett aus. Abgesehen vom Greedy-Verfahren ist das *bottom up*-Verfahren das beste. Bei Schwellenwerten ε von 1 bis 4 kann das multi adaptive Verfahren noch etwas bessere Kompressionswerte als die *bottom up*-Methode erreichen. Das Greedy-Verfahren kann zwar den vorgegebenen Fehler voll ausnutzen und hat bessere Kompressionsraten als die anderen Tensorprodukt-Ansatz-Methoden, aber es verbraucht unverhältnismäßig viel Rechenzeit.

Beim visuellen Vergleich der Verfahren (siehe Abbildungen 41 und 42) fällt besonders bei dem Greedy-Fehlerbeschränkungs-Algorithmus der Gobelin-Effekt auf. Dieser entsteht durch die langgestreckten Basisfunktionen des Tensorprodukt-Ansatzes und ist bei der Greedy-Methode wegen des am besten ausgenutzten Fehlers am deutlichsten zu erkennen. Wie auf den Differenzbildern zu sehen ist, tritt dieser Effekt bei allen Verfahren auf, aber auch beim visuellen Vergleich hinterläßt das *bottom up*-Verfahren den besten Eindruck.

Wenn also ein schnelles fehlerbeschränktes Verfahren im Tensorprodukt-Ansatz eingesetzt werden soll, ist die *bottom up*-Fehlerbeschränkungs-Variante die erste Wahl.

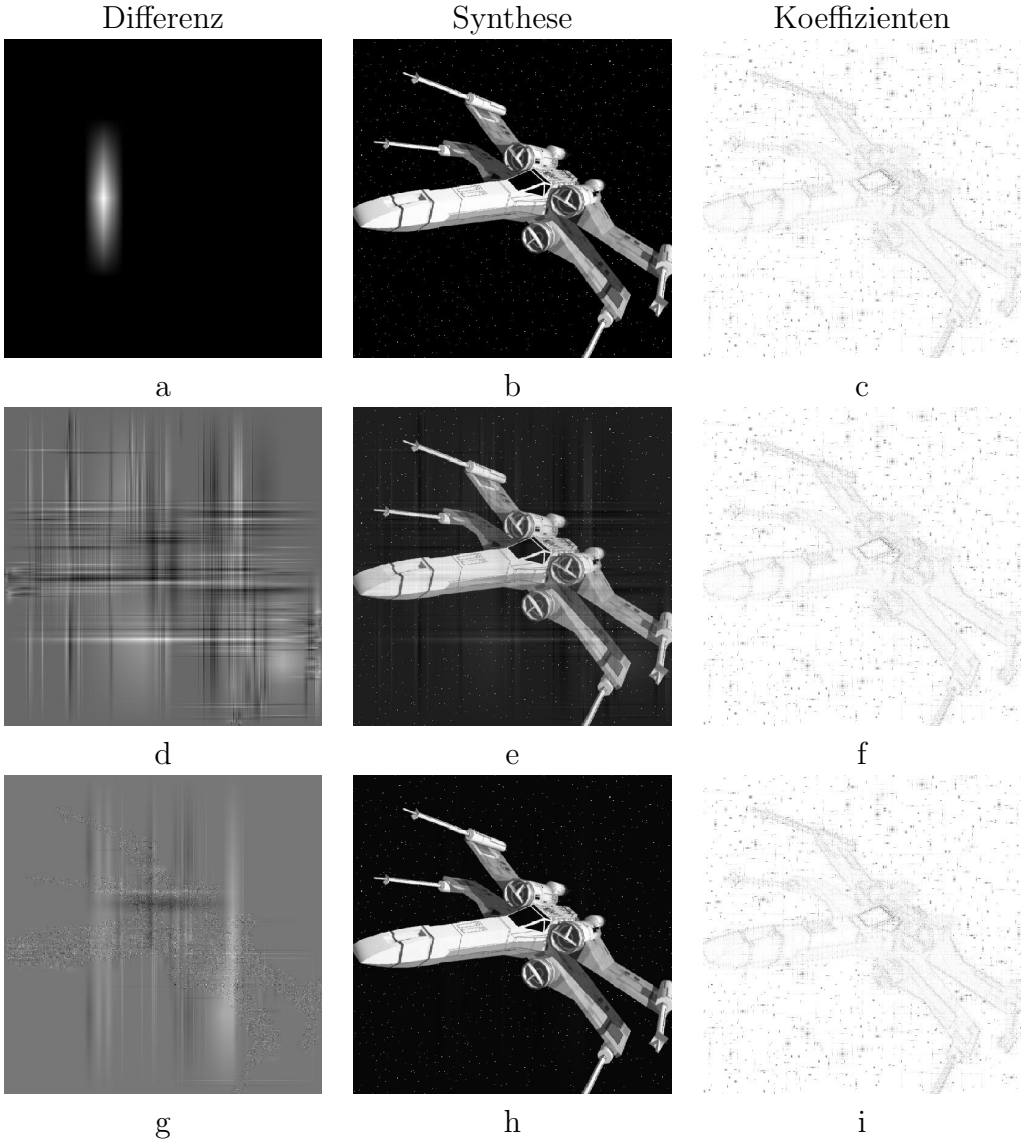


Abbildung 41: Differenz-, Synthese- und Koeffizientenbilder für den *X-Wing* bei $\varepsilon = 64$ im Tensorprodukt-Ansatz. Die Methoden sind: Level-abhängig (a, b und c), Orts- und Level-abhängig (d, e und f) und Einfach Adaptiv (g, h und i). Bei den Koeffizienten sind kleine Beträge weiß und große Beträge schwarz eingefärbt.

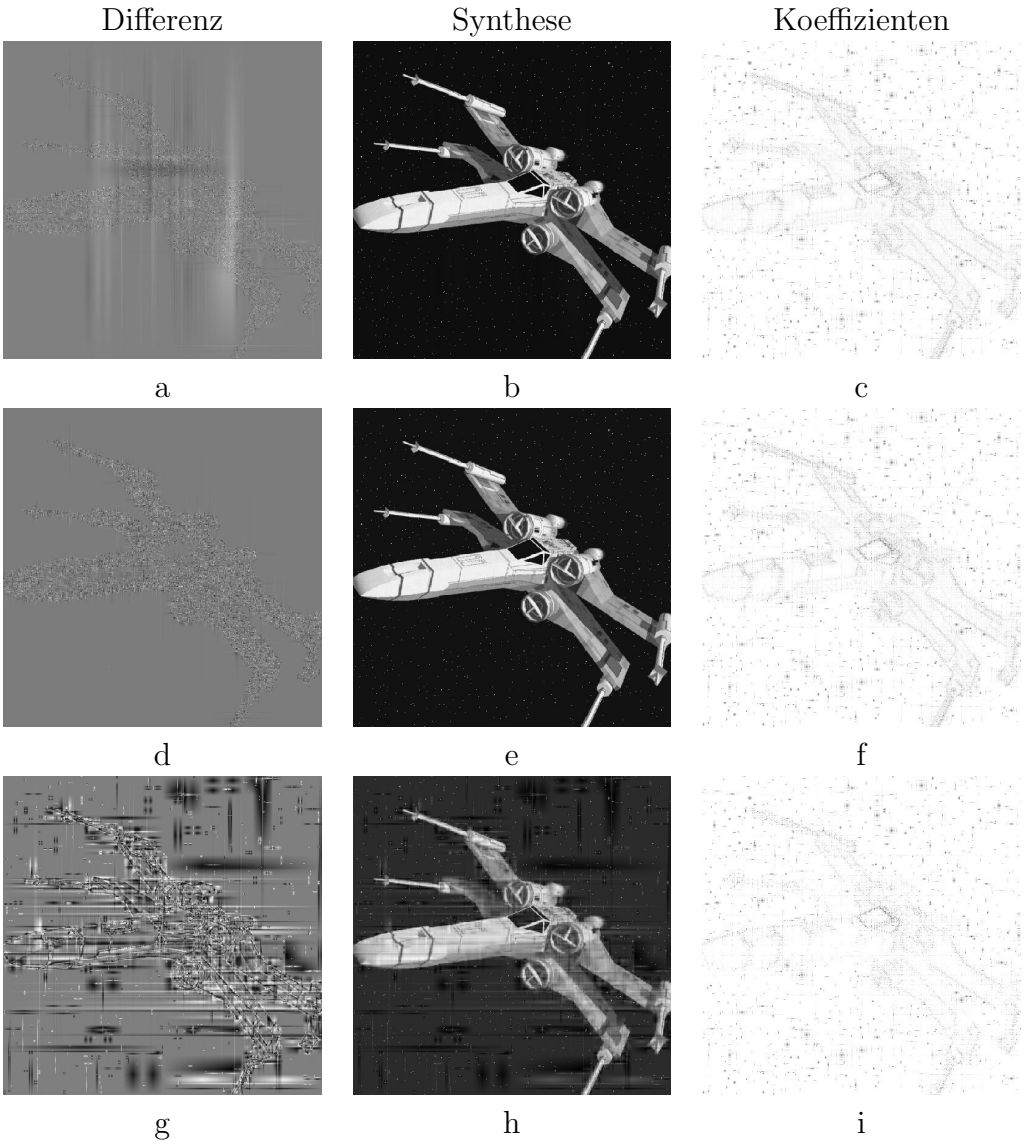


Abbildung 42: Differenz-, Synthese- und Koeffizientenbilder für den *X-Wing* bei $\varepsilon = 64$ im Tensorprodukt-Ansatz. Die Methoden sind: Multi Adaptiv (a, b und c), *bottom up* (d, e und f) und Greedy (g, h und i). Bei den Koeffizienten sind kleine Beträge weiß und große Beträge schwarz eingefärbt.

	Erft		Lena		X-Wing	
ε	Kompr. (%)	Fehler	Kompr. (%)	Fehler	Kompr. (%)	Fehler
1	26,50	0,00	3,26	0,00	72,04	0,00
2	26,50	0,00	3,26	0,00	72,04	0,00
4	26,50	0,25	3,26	0,00	72,04	0,00
8	26,51	1,75	3,26	0,75	72,04	0,00
16	26,56	1,88	3,26	1,00	72,04	0,00
32	26,70	11,32	3,26	2,38	72,04	2,00
64	27,19	23,33	3,28	9,00	72,04	2,00

Tabelle 3: Kompression und Fehler für Level-abhängiges Koeffizientenabschneiden.

	Erft		Lena		X-Wing	
ε	Kompr. (%)	Fehler	Kompr. (%)	Fehler	Kompr. (%)	Fehler
1	29,33	1,00	3,54	0,76	72,05	0,35
2	31,40	1,82	3,83	1,39	72,06	1,06
4	33,35	3,56	4,38	3,10	72,11	2,12
8	34,42	6,71	5,60	7,57	72,25	5,57
16	34,83	12,62	7,01	16,00	72,40	13,22
32	35,05	31,65	8,11	32,00	72,55	21,14
64	35,18	36,80	8,62	64,00	72,72	57,52

Tabelle 4: Kompression und Fehler für Orts- und Level-abhängiges Koeffizientenabschneiden.

	Erft		Lena		X-Wing	
ε	Kompr. (%)	Fehler	Kompr. (%)	Fehler	Kompr. (%)	Fehler
1	42,39	1,00	8,94	1,00	72,53	1,00
2	50,04	2,00	14,03	2,00	72,90	1,88
4	58,65	4,00	16,60	2,47	73,02	2,01
8	60,53	2,75	15,80	2,51	73,90	3,34
16	62,41	3,60	16,96	3,16	74,27	4,17
32	79,79	4,79	26,85	5,89	74,19	4,84
64	82,49	12,89	34,52	11,95	76,36	10,96

Tabelle 5: Kompression und Fehler für einfach adaptives Koeffizientenabschneiden (*top down*).

	Erft		Lena		X-Wing	
ε	Kompr. (%)	Fehler	Kompr. (%)	Fehler	Kompr. (%)	Fehler
1	42,39	1,00	8,94	1,00	72,53	1,00
2	50,05	2,00	14,91	2,00	73,20	2,00
4	58,66	1,95	19,07	3,13	73,81	4,00
8	60,53	2,75	18,17	3,72	74,99	6,38
16	62,42	3,60	22,15	4,62	75,21	8,25
32	80,93	4,29	29,41	7,59	75,57	8,88
64	85,98	13,10	36,99	13,82	77,81	16,77

Tabelle 6: Kompression und Fehler für multi adaptives Koeffizientenabschneiden.

	Erft		Lena		X-Wing	
ε	Kompr. (%)	Fehler	Kompr. (%)	Fehler	Kompr. (%)	Fehler
1	26,50	0,00	3,26	0,00	72,04	0,00
2	40,95	0,50	6,05	0,50	72,19	0,38
4	58,65	1,50	11,03	1,25	72,52	1,00
8	68,17	2,56	19,21	2,69	73,33	2,50
16	81,21	3,90	33,42	5,62	75,04	5,50
32	91,28	5,86	52,13	10,95	77,83	10,25
64	95,34	9,05	67,58	22,21	81,47	21,38

Tabelle 7: Kompression und Fehler für *bottom up* Koeffizientenabschneiden.

	Erft		Lena		X-Wing	
ε	Kompr. (%)	Fehler	Kompr. (%)	Fehler	Kompr. (%)	Fehler
1	72,75	1,00	12,83	1,00	73,91	1,00
2	86,17	2,00	23,05	2,00	75,05	2,00
4	93,29	4,00	41,69	4,00	77,19	4,00
8	97,18	8,00	67,30	8,00	80,43	8,00
16	98,51	15,94	83,84	16,00	84,14	16,00
32	99,07	30,40	92,33	32,00	88,36	32,00
64	99,08	33,16	96,64	64,00	93,56	64,00

Tabelle 8: Kompression und Fehler für Koeffizientenabschneiden durch Greedy.

9.2 Hierarchische Triangulierung

In diesem Abschnitt werden Ergebnisse und Vergleiche zur hierarchischen Triangulierung zusammengetragen.

Die dafür verwandten Bilder wurden alle mit der hierarchischen Triangulierung transformiert. Dafür wurde das Ausgangsquadrat jeweils in ein linkes oberes und ein rechtes unteres Dreieck aufgeteilt. Diese Dreiecke wurden dann rekursiv transformiert wie in Abschnitt 3.2.2 beschrieben.

Auf den so erhaltenen Koeffizienten wurde dann ein *n-level-look-ahead* berechnet. Einzige Ausnahme hierzu ist Abbildung 45, bei der zum Vergleich das Bild sowohl mit *n-level-look-ahead* als auch mit *1-level-look-ahead* berechnet wurde.

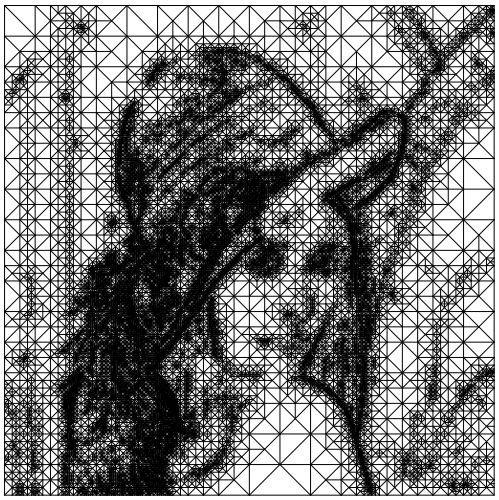
Die Koeffizienten wurden entlang der Kurve des $2m$-Verfahrens abgespeichert. Die doppelten Koeffizienten wurden, wie in Abschnitt 6.2.4 beschrieben, ausgelassen. Die restlichen Koeffizienten wurden mit einem Huffman-Verfahren kodiert. Das Huffman-Verfahren wurde dazu mit den Statistiken der jeweiligen Bildkoeffizienten initialisiert. Auf die Qualität der Synthese hat die Speicherung der Koeffizienten oder Werte keinen Einfluß. Sie bestimmt lediglich die Kompressionsrate.

In Abbildung 43 ist das Bild *Lena* mit einem Schwellenwert ε von 20 komprimiert worden. Die Kompressionsrate beträgt hierbei 88,76%. Die Ausschnittsvergrößerung der Synthese unten links zeigt, daß noch keine sichtbaren Artefakte auftreten. Auch der direkte Vergleich des Originals (rechts unten) mit der Synthese (rechts oben) zeigt, daß keine wichtigen Informationen verlorengegangen sind. An der Triangulierung (links oben) ist die Adaptivität des Verfahrens gut zu erkennen.

9.2.1 Raumfüllende Kurven

In Abbildung 44 sind die raumfüllenden (Sierpiński-)Kurven des $2m$-Verfahrens für die Bilder *X-Wing*, *CT-Kopf*, *Lena* und *Peppers* dargestellt. Der Schwellenwert ε wurde erneut auf 20 gesetzt.

Deutlich zu sehen ist die adaptive Verfeinerung der Triangulierung an Kanten in den Ausgangsbildern. Obwohl für die raumfüllende Kurve nur einige Linien weniger zu zeichnen sind als bei der direkten Darstellung der Triangulierung, sind die zugrundeliegenden Strukturen besser zu erkennen. Vergleiche hierzu die Triangulierung aus Abbildung 43 links oben mit der in Abbildung 44 dargestellten Kurve der *Lena* links unten.



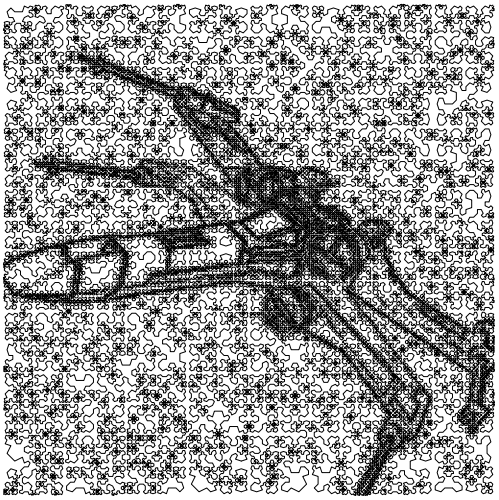
Triangulierung

Synthese

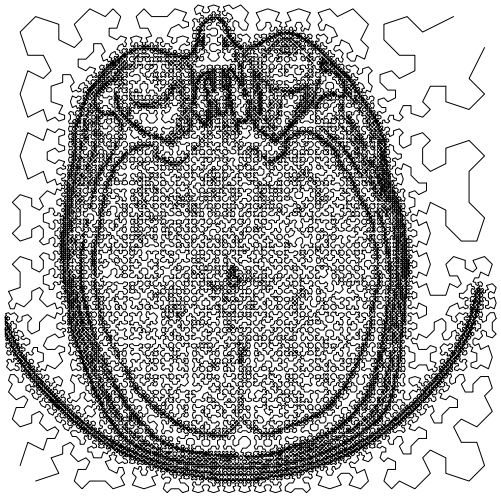
Ausschnitt

Original

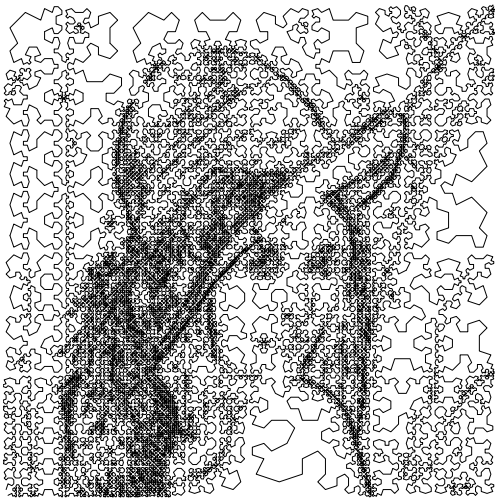
Abbildung 43: Triangulierung, Synthese, eine Ausschnittsvergrößerung und Original des Bildes *Lena* bei $\varepsilon = 20$ mit *n-level-look-ahead* (von links oben nach rechts unten).



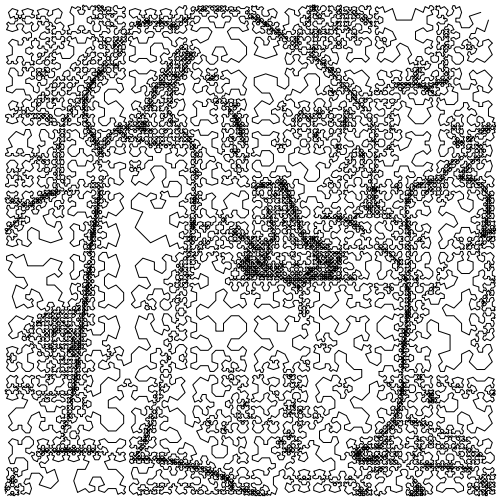
X-Wing Kurve



CT-Kopf Kurve



Lena Kurve



Peppers Kurve

Abbildung 44: Sierpiński-Kurve entlang der Triangulierung bei $\varepsilon = 20$ für die Bilder *X-Wing*, *CT-Kopf*, *Lena* und *Peppers* (von links oben nach rechts unten).

9.2.2 Look-Ahead

Ein Vergleich der Verfahren *1-level-look-ahead* und *n-level-look-ahead* ist in Abbildung 45 zu sehen. Das *n-level-look-ahead* Verfahren kann den vorgegebenen Fehler ε von 32 besser ausnutzen als die *1-level-look-ahead* Methode. Die Triangulierung muß für das *1-level-look-ahead* deutlich feiner ausfallen, damit die Fehlerschranken nicht überschritten werden. Auch das Differenzbild der Synthese zum Original zeigt deutlich weniger Kontraste als das *n-level-look-ahead* Pendant, was auf eine schlechtere Ausnutzung des Fehlers hinweist.

Für die Transformation mittels *1-level-look-ahead* brauchte ein AMD K6 mit 450 MHz 1,92 Sekunden, für den *n-level-look-ahead* 8,21 Sekunden. Die Kompressionsraten betragen 87,21%, respektive 93,87%. Die Kompressionsrate für das *1-level-look-ahead* Verfahren ist bei einem Schwellenwert ε von 32 schlechter als die Kompressionsrate bei einem Schwellenwert ε von 20 mit der *n-level-look-ahead* Methode.

In Abbildung 46 ist die Zeit, die zur Komprimierung mit *1-level-look-ahead* und *n-level-look-ahead* für verschiedene Bildgrößen benötigt wurde, aufgetragen. Deutlich ist der höhere Aufwand für das *n-level-look-ahead*-Verfahren zu sehen. Für die Rechenzeit der beiden Verfahren ist ausschließlich die Größe der Bilder verantwortlich.

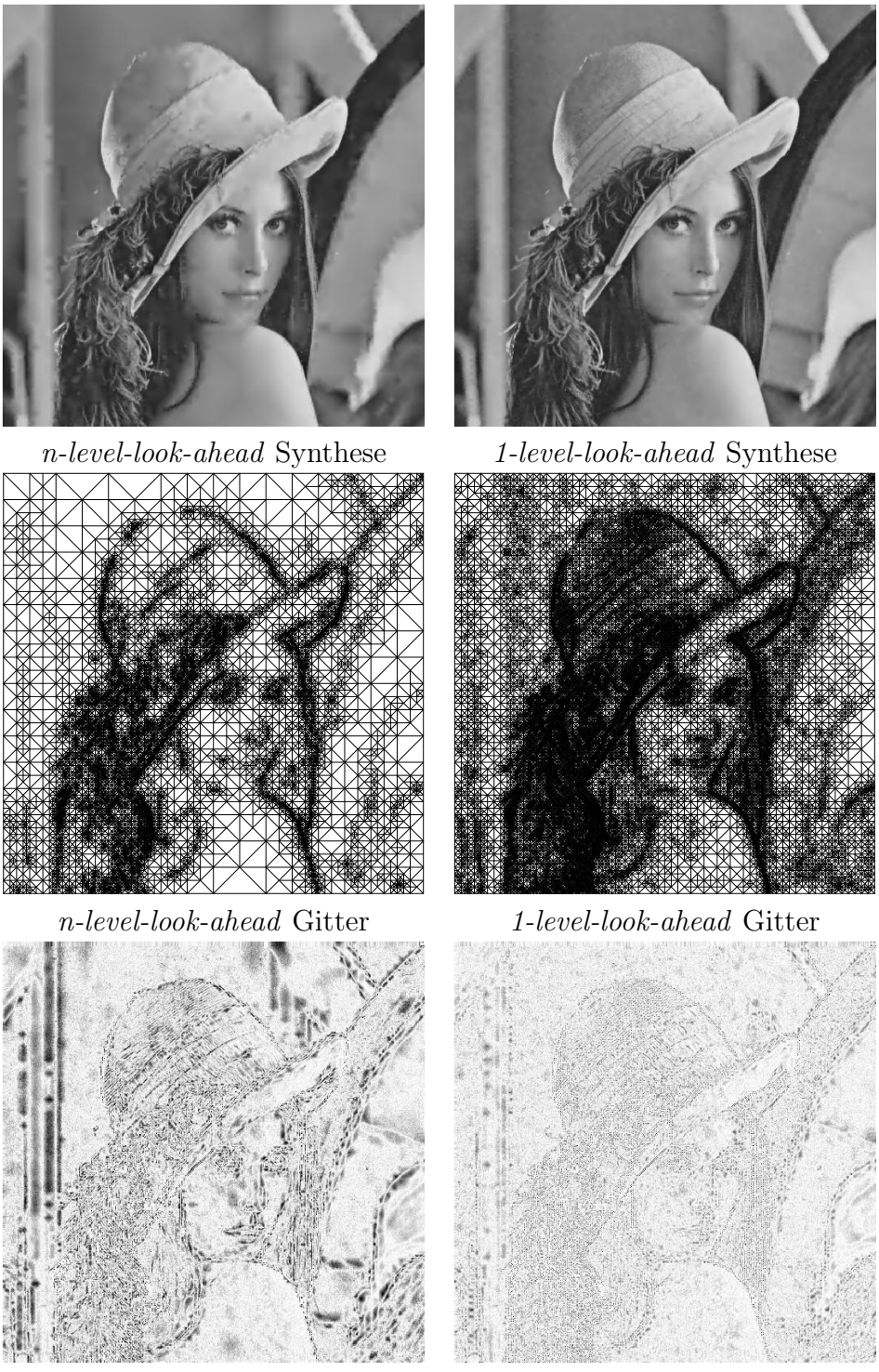
Wie in Abbildung 45 aber erkennbar ist, kann das *n-level-look-ahead*-Verfahren den vorgegebenen Fehler viel besser ausnutzen. Daher wurden alle Ergebnisse in diesem Abschnitt mit *n-level-look-ahead* berechnet.

9.2.3 Farbräume

Die Farbbilder in den Abbildungen 47 und 48 wurden in den YC_bC_r-Farbraum transformiert, bevor die Farbkomponenten dann einzeln wie Graustufenbilder komprimiert wurden.

In Abbildung 47 ist deutlich zu sehen, wie sich die Farbinformationen in der Y-Komponente konzentrieren. Die beiden anderen Komponenten weisen deutlich geringere Kontrastwerte auf. Die feinen Haare des *Mandrill* forcieren eine feine Triangulierung vor allem in der Y-Komponente, wohingegen die C_b- und C_r-Komponenten wie erwartet eine deutlich gröbere Triangulierung zulassen.

Bei der Statue aus Abbildung 48 ist die Konzentration noch besser zu erkennen, da hier nur die Kugel und der Lorbeerkranz golden gefärbt sind. Die raumfüllenden Kurven der C_b- und C_r-Komponenten – dargestellt ist nur die der C_r sehr ähnlich sehende C_b-Komponente – werden auch nur zu diesen farbigen Punkten hinverfeinert. Auch der Kontrast der C_b-Werte zeigt an, wie wenige Informationen hier benötigt werden.



n-level-look-ahead Synthese | *1-level-look-ahead* Synthese

n-level-look-ahead Gitter | *1-level-look-ahead* Gitter

n-level-look-ahead Differenz | *1-level-look-ahead* Differenz

Abbildung 45: Das Bild *Lena* als Synthese, Gitter und Differenzbild zum Original (von oben nach unten) bei $\varepsilon = 32$ mit links *n-level-look-ahead*- und rechts *1-level-look-ahead*-Algorithmus. Es ist deutlich zu sehen, daß der *1-level-look-ahead*-Algorithmus erheblich mehr Informationen speichern muß.

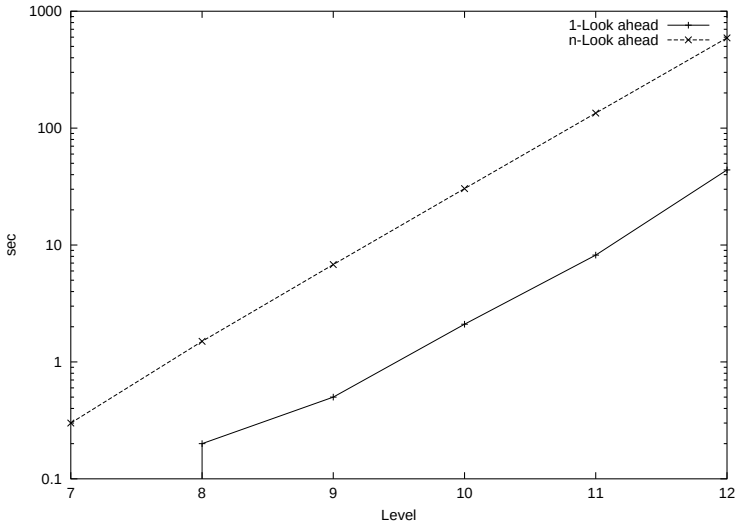


Abbildung 46: Vergleich von Rechengenauigkeit gegenüber Zeitaufwand bei Verwendung von verschiedenen *look-ahead* Varianten und unterschiedlich großen Bildern. Die Größe des Bildes wird durch den Level angedeutet.

9.2.4 Histogramme

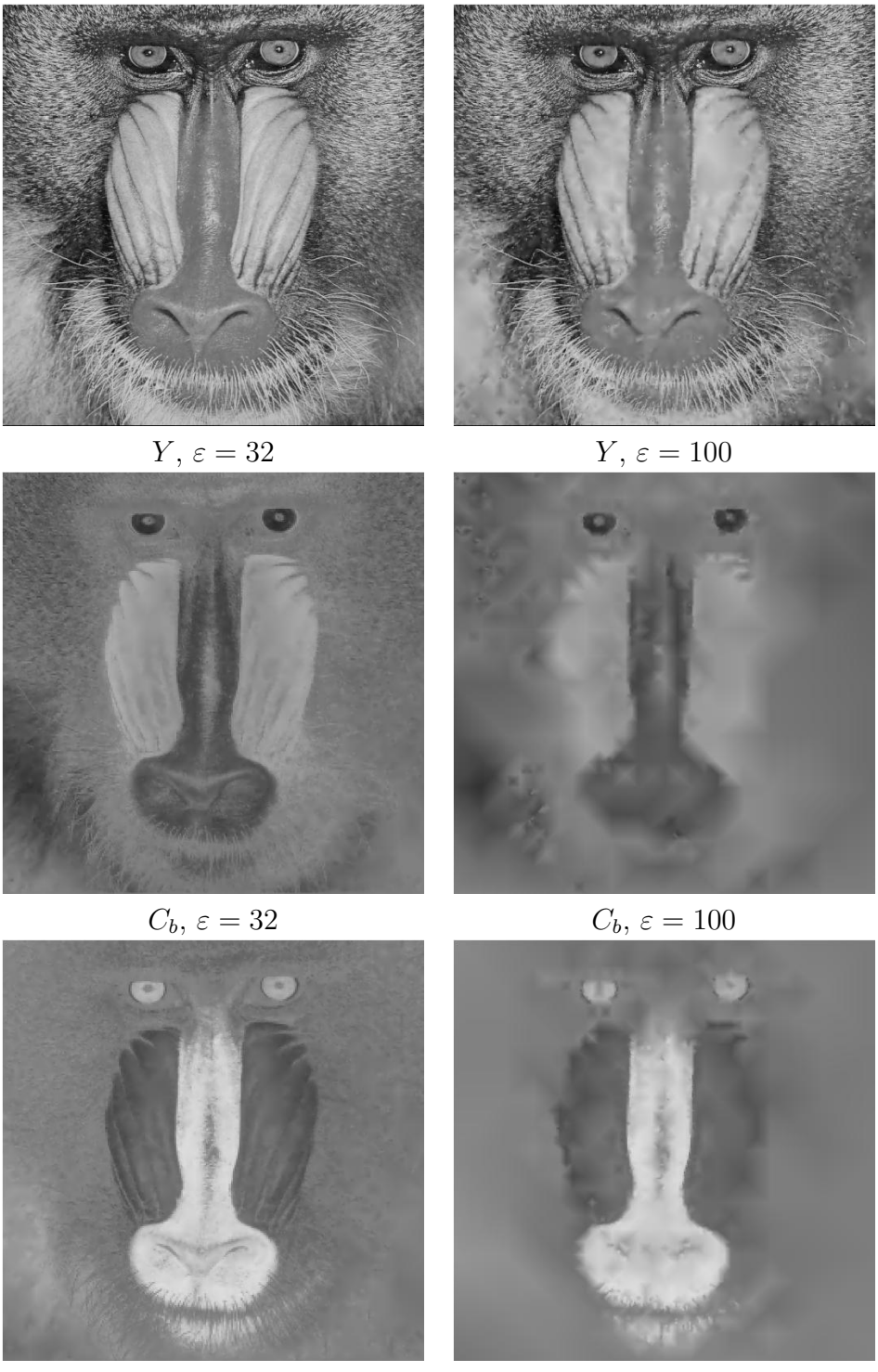
Die Histogramme in den Abbildungen 49–51 stellen die Häufigkeit der Werte bzw. Koeffizienten der komprimierten Bilder dar, wie sie der Huffman-Kodierer nach Entfernung der doppelten Knoten als Statistik erhält. In Abbildung 49 ist zu sehen, daß zu Beginn viele kleine Koeffizienten existieren, diese aber bei zunehmendem Schwellenwert ε wegfallen.

Bei den Werten in Abbildung 50 ist diese Konzentration nicht zu beobachten. Die Häufigkeitsverteilung ist deutlich homogener. Auch die Reduzierung bestimmter Werte bei zunehmendem Schwellenwert ε ist nicht zu beobachten.

Abbildung 51 zeigt noch einmal für verschiedene Bilder sowohl Werte- als auch Koeffizienten-Histogramme.

9.2.5 Fokussierung

Für die in Abbildung 52 dargestellte Fokussierung, wurde die dort oben rechts abgebildete Fokussiermaske verwandt. Diese ist in der Mitte weiß und in den Außenbereichen dunkelgrau. Die Farbwerte liegen im Bereich $[0, 255]$ und werden für den Fokussiervorgang auf $[0, 1]$ skaliert. Dieser Wert wird vor der Saturation mit den Fehlerschrankenwerten multipliziert. Ein dunkler Farbpunkt hat einen Wert nahe Null, so daß ein Koeffizient an der entsprechen-



Y, $\varepsilon = 32$ Y, $\varepsilon = 100$

C_b, $\varepsilon = 32$ C_b, $\varepsilon = 100$

C_r, $\varepsilon = 32$ C_r, $\varepsilon = 100$

Abbildung 47: YC_bC_r Komponenten des Bildes *Mandrill* einzeln aufgetragen für links $\varepsilon = 32$ und rechts $\varepsilon = 100$. Von oben nach unten Y, C_b und C_r. Die Informationskonzentration auf den Y-Kanal ist deutlich erkennbar.

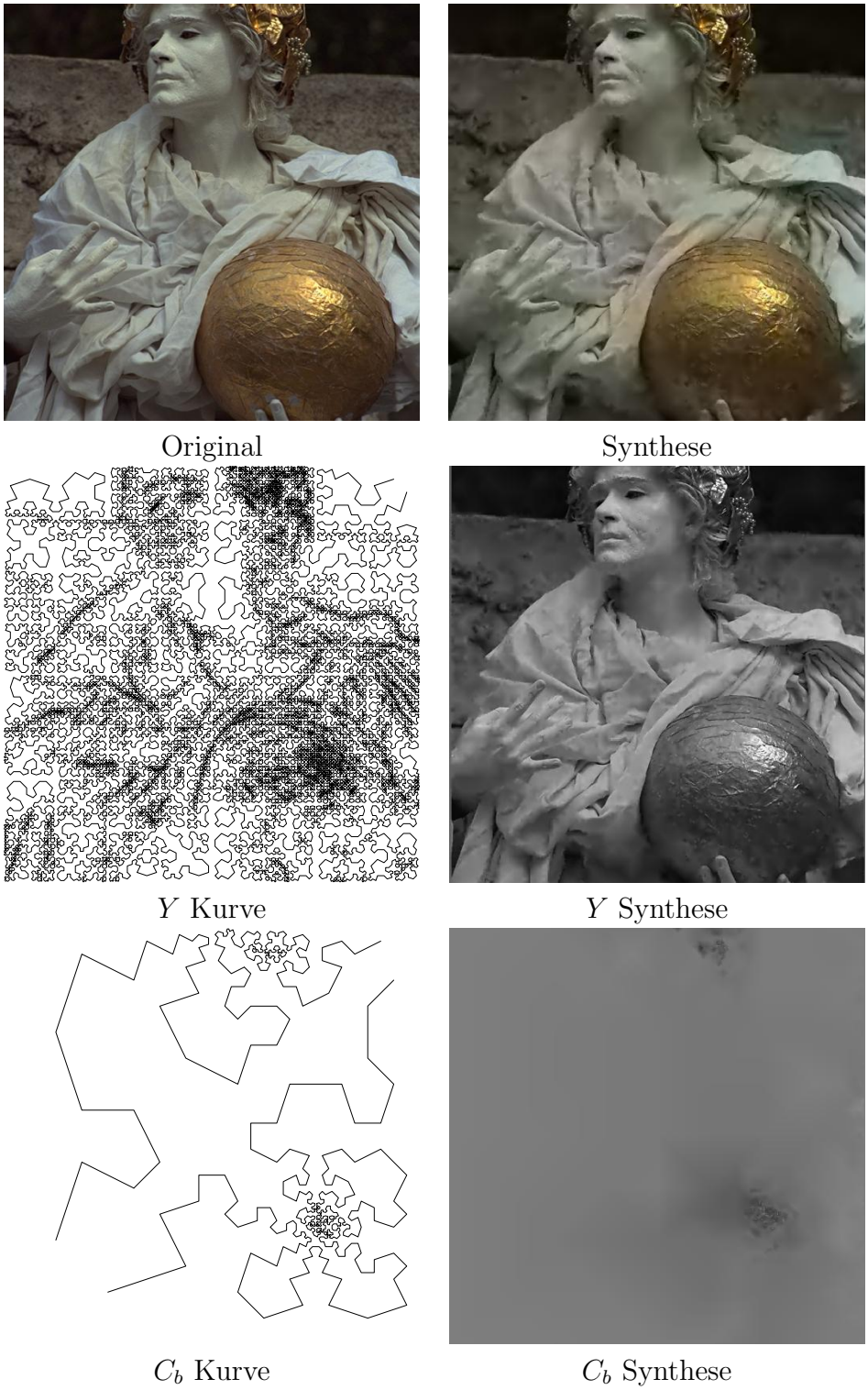


Abbildung 48: Bild einer Statue mit goldener Kugel und goldenem Lorbeerkranz. Von links oben nach rechts unten: Original, Synthese bei einem Epsilon von 64, raumfüllende Kurve und Synthese der Y- und der C_b-Komponente. Die hier nicht dargestellte C_r-Komponente sieht der C_b-Komponente sehr ähnlich.

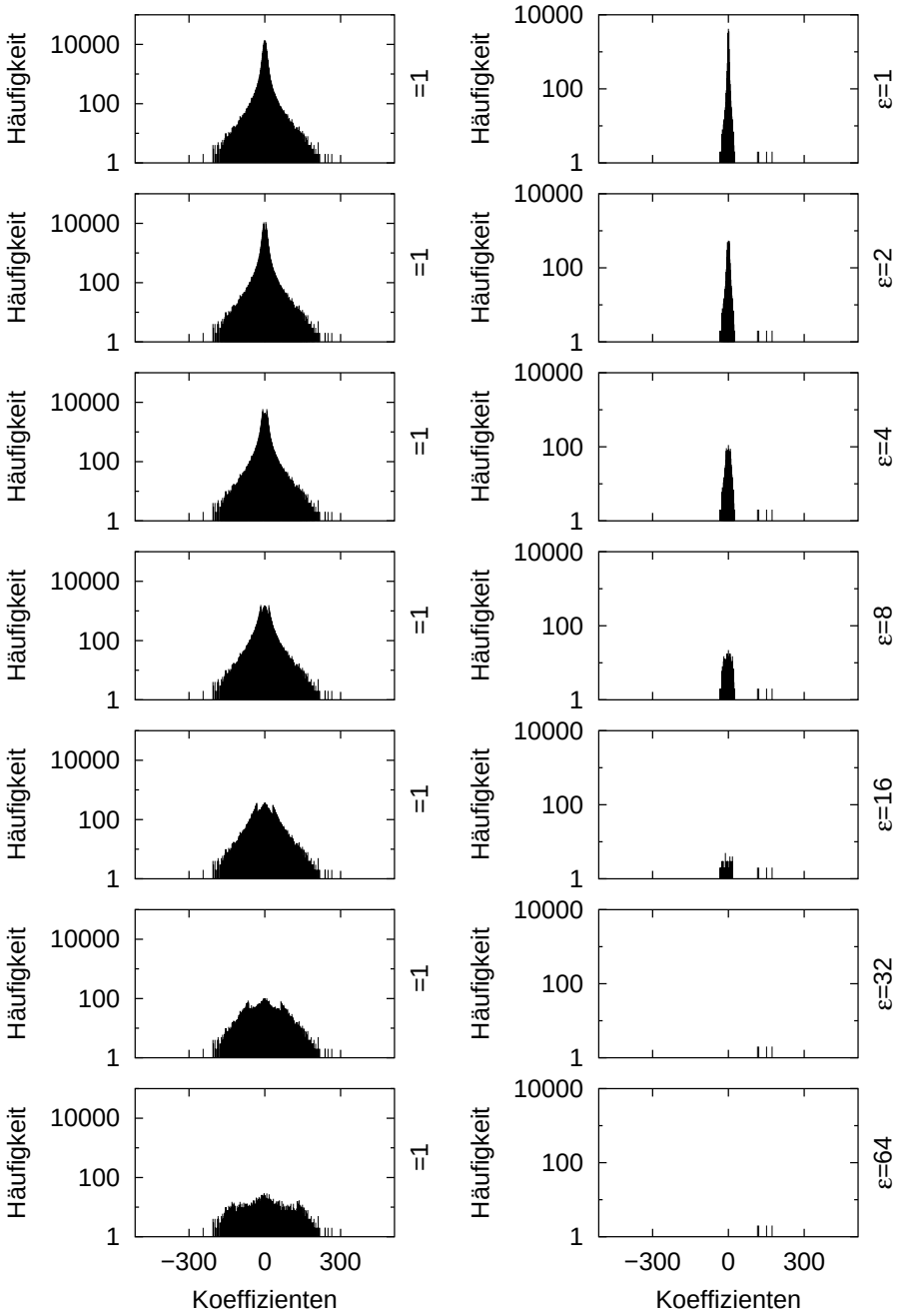


Abbildung 49: Histogramme der Koeffizienten der Bilder *Lena* (links) und *Erft* (rechts) mit Fehlerschranken ε von 1, 2, 4, 8, 16, 32 und 64 (von oben nach unten).

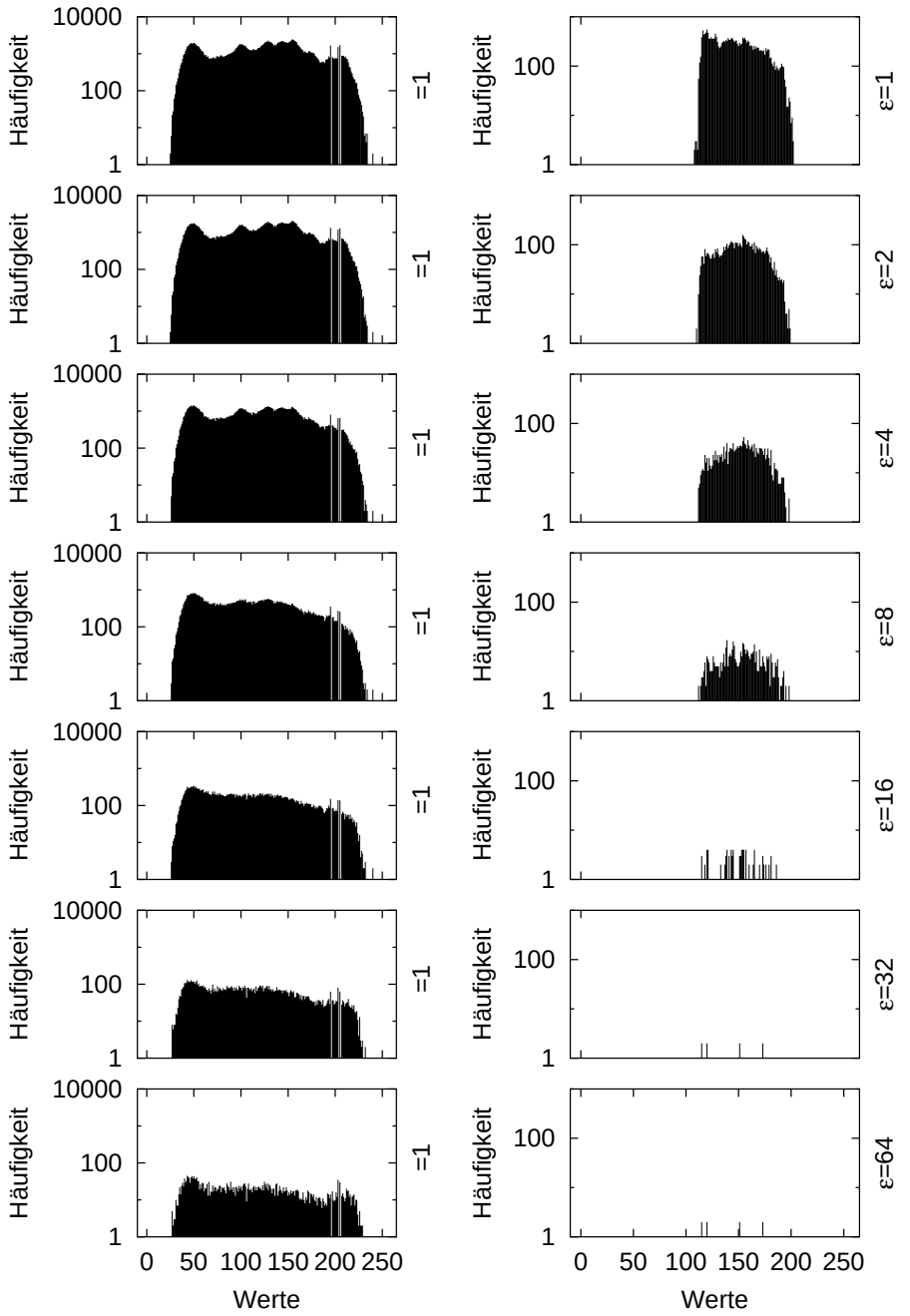


Abbildung 50: Histogramme der Werte der Bilder *Lena* (links) und *Erft* (rechts) mit Fehlerschranken ε von 1, 2, 4, 8, 16, 32 und 64 (von oben nach unten).

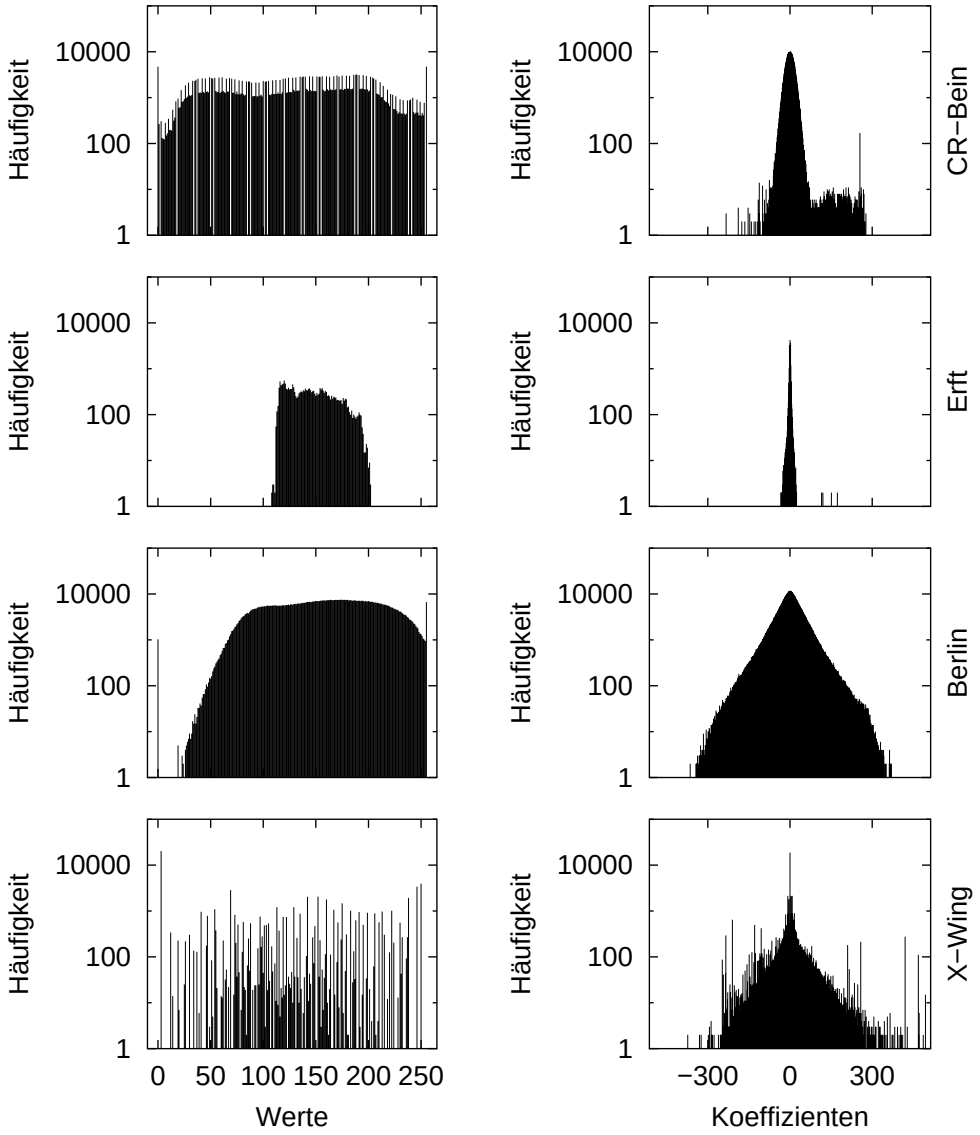


Abbildung 51: Histogramme der Werte (links) und Koeffizienten (rechts) der Bilder *CR-Bein*, *Erft*, *Berlin* und *X-Wing* (von oben nach unten). Alle Histogramme gelten für einen Fehlerwert ε von Eins.

den Stelle einen größeren Fehler erreichen kann als vorgegeben. Ein weißer Farbwert hingegen hält den Fehler des zugehörigen Koeffizienten in den vorgeschriebenen Grenzen. Bei realen Anwendungen, wie einem Flugsimulator, würde die Fokussiermaske sehr viel glatter und stetiger sein. Der starke Kontrast wurde gewählt, damit der Effekt an sich besser zu erkennen ist.

9.2.6 Kodierung

Um die Kodierungsverfahren zu vergleichen, wurden 14 Bilder ausgesucht (siehe Abbildungen 56–58 in Abschnitt A.1) und jeweils unterschiedlich komprimiert. Bei den Struktur-Kodierern sind nur die $2m$-Verfahren und das Null-Setzen ausgewählt worden. Die $2m$-Methode wurde ausgewählt, da sie für die vorliegenden Daten auf jeden Fall besser ist als die anderen Baum-Kodierer. Die einzige verbleibende Alternative war somit die Strukturierung durch Null-Setzen.

Bei der Daten-Kodierung wurden die explizite Speicherung und RLE nicht getestet, da – wie in Kapitel 7.2 beschrieben – die Daten für diese Verfahren nicht geeignet sind. Für das LZW-Verfahren wurden die Programme `compress` und `gzip`[2] ohne spezielle Optionen verwandt. Für die Burrows-Wheeler-Transformation kam das Programm `bzip2` zum Einsatz. Die Huffman-Kodierung wurde selbst implementiert und konnte so speziell an die Größe der Koeffizienten der hierarchischen Triangulierung – diese betragen bei 8 Bit Graustufenbildern 10 Bit – angepaßt werden. Für die arithmetische Kodierung wurde das Programm `arith_coder` (Carpinelle u. a. 1999) verwandt. Die Daten für die oben genannten Programme wurden entlang der raumfüllenden Kurve für das $2m$-Verfahren gespeichert bzw. für das Null-Setzen gemäß der Baumreihenfolge gespeichert. Desweiteren wurden von Bildern einmal die Koeffizienten an den entsprechenden Knoten und einmal die Werte an den Knoten gespeichert. Dabei ergab sich, daß die arithmetische Kodierung, `compress` und `gzip` der Burrows-Wheeler-Transformation und dem handangepaßten Huffman-Kodierer in allen Fällen unterlegen waren. Deswegen werden diese Kompressionsergebnisse hier nicht aufgeführt. In den Tabellen 9–12 sind die Kompressionsraten der Bilder für die acht übriggebliebenen Methoden aufgeführt. Die jeweils besten Kompressionsraten sind in den Tabellen durch Sterne (*) gekennzeichnet. Die Kompressionsrate bezieht sich dabei immer auf die ursprüngliche Größe der Bilder. Diese ist durch Breite×Höhe×Bits pro Pixel/8 in KByte gegeben. Die Bilder haben alle 8 Bits pro Pixel (*bpp*). In Abbildung 61 und 62 sind die Größen der Bilder

[2]Das Programm `gzip` kann die Daten auch mit einem Huffman-Verfahren kombiniert speichern.

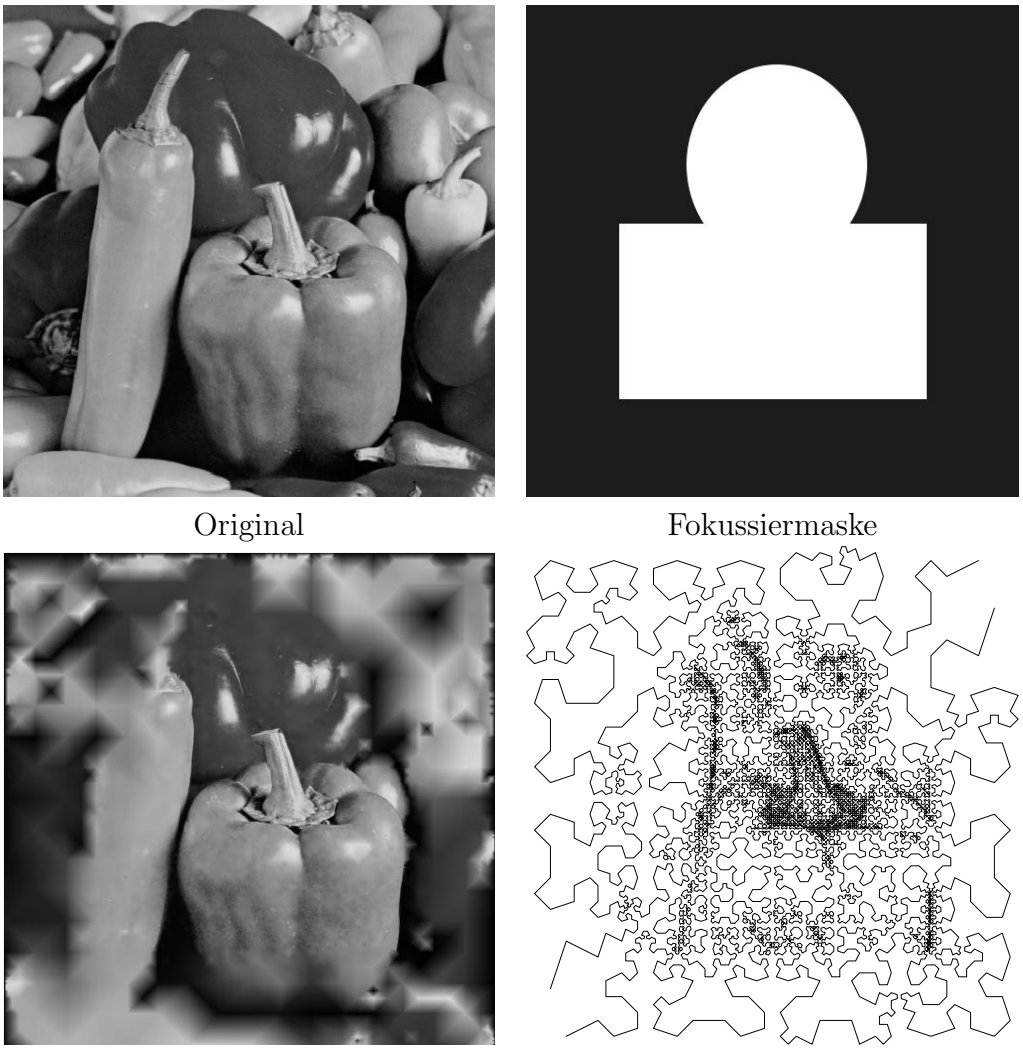


Abbildung 52: Fokussierung mittels einer Bildmaske. Epsilon ist auf 16 gesetzt. Weiß in der Maske bedeutet, daß der Fehler voll mit berechnet werden soll, schwarz bedeutet, daß der Fehler ignoriert werden soll. Von links oben nach rechts unten: Original, Fokussiermaske (in der Mitte weiß und außen dunkelgrau), Synthese und Kurve.

in KByte eingetragen. In den Abbildungen 59 und 60 sind die Kompressionsraten für die gleichen Bilder aufgetragen, die durch die Kodierungen erreicht werden. Hier wurden die unkodierten Größen der Daten als Maß genommen. Die Ergebnisse aus den Abbildungen 59 und 60 sind in Abbildung 54 noch einmal zusammengefaßt. In Abbildung 53 sind die Werte aus den folgenden Tabellen 9–12 zusammengetragen.

Aus Abbildung 54 ist zu ersehen, daß die BWT im Durchschnitt besser als die Huffman-Kodierung ist. Dies trifft vor allem zu, wenn es um die Kodierung der Werte und nicht die der Koeffizienten geht. In den Abbildungen 49–51 sind Histogramme der Werte und der Koeffizienten diverser Bilder dargestellt. Dort ist zu sehen, daß sich bei den Bildern, bei denen die Huffman-Kodierung besonders effizient ist, die meisten Koeffizienten um den Nullpunkt sammeln. Je breiter die Streuung wird, um so schlechter wird die Komprimierung der Koeffizienten durch Huffman. In Abbildung 51 sind die oberen beiden Bilder besser über die Koeffizienten zu komprimieren und die unteren beiden durch die entsprechenden Werte. Dies hängt mit der Kodierung durch das Huffman-Verfahren zusammen. Dieses Verfahren ist besonders geeignet für ungleiche Verteilungen der Eingangsdatenhäufigkeit. Das erklärt, daß die Huffman-Kodierung für die Werte wie in Abbildung 50 oben sehr viel schlechtere Ergebnisse liefert als die BWT, die nicht auf die Häufigkeit der Werte in dem Datenstrom angewiesen ist. Wenn der Schwellenwert ε größer gewählt wird, so ähnelt das Histogramm für die Koeffizienten (Abbildung 49) immer mehr dem Histogramm der Werte (Abbildung 50). Die Huffman-Kodierung wird also mit zunehmendem ε grundsätzlich ungeeigneter als die BWT.

Daß die Kompressionsraten in Abbildung 54 für die durch Null-Setzen erhaltenen Daten größer ist als für die $2m$-Verfahren, liegt an der großen Häufigkeit der Null in den Daten, die sowohl Huffman als auch BWT gut reduzieren können. Allerdings ist auch hier der Huffman-Kodierer ein wenig im Nachteil, da dieser für jede Null mindestens ein Bit speichern muß.

Insgesamt ergibt sich, wie in Abbildung 53 zu sehen ist, folgendes Bild: Bei geringen Fehlerwerten ε kann das $2m$-Verfahren seine Stärke noch nicht ausspielen. Es erzeugt mehr Overhead als es einzusparen vermag. Hier ist es am besten, die BWT auf die mit Nullen aufgefüllten Werte anzuwenden. Je größer der Schwellenwert ε wird, um so besser wird das $2m$-Verfahren. Aber auch hierbei ergeben sich die besten Ergebnisse, wenn Burrows-Wheeler auf die Werte angewandt wird. Wie in den Tabellen 9–12 zu sehen ist, kann aber in einigen Fällen auch die Huffman-Kodierung bessere Werte als die BWT erzielen.

Datei	k	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$	$\varepsilon = 16$	$\varepsilon = 32$	$\varepsilon = 64$
Belle	h	18,76	23,66	32,79	54,10	83,94	94,21	*98,29
Belle	b	44,65	47,73	53,29	*66,71	*86,33	*94,49	98,25
Berlin	h	-16,40	-12,86	-6,09	6,43	27,77	*58,08	*84,47
Berlin	b	-13,76	-10,51	-4,22	7,60	28,07	57,88	84,32
Canyon	h	15,28	15,70	19,08	24,87	36,79	59,29	*87,32
Canyon	b	19,94	20,32	23,35	28,54	39,25	*60,00	86,98
CR Bein	h	58,83	61,26	65,96	75,24	90,90	99,34	*99,82
CR Bein	b	68,74	70,44	73,77	*80,58	*92,53	*99,40	99,82
CR Thorax	h	1,32	19,55	51,49	85,39	98,49	99,57	99,69
CR Thorax	b	35,43	46,32	*66,30	*88,97	*98,70	*99,60	*99,71
CT Kopf	h	59,88	63,76	69,66	79,85	87,86	*93,02	*96,25
CT Kopf	b	65,14	68,21	*73,00	*81,47	*88,38	92,95	96,03
Erft	h	64,73	90,31	96,81	98,98	99,72	*99,93	*99,96
Erft	b	75,60	92,11	97,11	99,02	*99,73	99,89	99,91
Lena	h	-6,88	7,15	32,95	67,08	85,97	94,40	*98,24
Lena	b	15,37	24,92	43,42	*69,98	*86,44	*94,42	98,19
Mandrill	h	-13,69	-9,01	-0,28	16,32	40,66	*65,03	*86,44
Mandrill	b	-8,14	-4,05	3,56	18,39	*41,11	64,91	86,31
Mona Lisa	h	7,35	9,86	20,92	38,51	63,89	87,93	*98,49
Mona Lisa	b	26,77	28,68	36,42	*49,11	*68,56	*88,61	98,44
Parabel	h	64,74	95,67	98,56	98,58	98,59	98,59	99,29
Parabel	b	80,26	96,73	*98,67	98,69	98,70	98,70	*99,30
Peppers	h	-11,40	-0,57	19,60	53,67	82,71	92,85	*97,32
Peppers	b	12,32	19,75	34,06	*59,74	*83,71	*92,99	97,30
Vancouver	h	4,03	10,57	23,79	47,27	75,03	91,22	*97,85
Vancouver	b	17,75	22,81	33,10	*52,31	*76,41	*91,36	97,81
X-Wing	h	76,30	76,86	78,84	82,62	86,67	90,33	94,29
X-Wing	b	80,76	81,15	*82,45	*85,10	*88,12	*91,17	*94,74

Tabelle 9: Kompressionsraten in Prozent für verschiedene Bilder bei Speicherung der Werte entlang der raumfüllenden Kurve des $2m$-Verfahrens. Die Kodierungsverfahren sind BWT mittels `bzip2` (b) und eine angepaßte Huffman-Variante (h). Die ε-Werte kennzeichnen die Schwellenwerte für das Abschneidekriterium. Die Sterne markieren das beste Verfahren. (Werte kleiner Null stellen eine Vergrößerung der Ausgangsdatei dar.)

Datei	k	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$	$\varepsilon = 16$	$\varepsilon = 32$	$\varepsilon = 64$
Belle	h	32,38	34,41	39,36	53,56	75,33	83,08	86,19
Belle	b	*55,58	*55,50	*56,37	62,87	82,44	92,58	97,54
Berlin	h	5,96	*7,00	*9,80	*16,56	*30,47	54,40	75,28
Berlin	b	*6,91	6,98	7,59	10,86	23,00	50,86	81,36
Canyon	h	36,95	37,05	37,53	39,26	*44,54	58,96	78,56
Canyon	b	*39,93	*39,94	*39,82	*40,16	42,16	54,90	84,25
CR Bein	h	55,30	57,20	60,88	68,14	80,41	87,01	87,38
CR Bein	b	*75,30	*75,32	*75,70	78,65	90,72	99,17	99,78
CR Thorax	h	13,03	24,43	49,13	76,02	86,35	87,19	87,29
CR Thorax	b	*47,17	*48,33	60,70	85,74	98,17	99,51	99,67
CT Kopf	h	59,84	62,68	66,95	74,27	80,07	83,64	85,75
CT Kopf	b	*70,81	*70,97	72,41	79,55	87,25	92,32	95,76
Erft	h	60,63	80,08	85,02	86,68	87,27	87,45	87,47
Erft	b	69,63	88,79	95,65	98,48	99,55	99,87	99,92
Lena	h	9,85	17,18	34,61	61,52	76,41	83,06	86,09
Lena	b	28,58	29,58	38,63	64,35	83,00	92,74	97,53
Mandrill	h	8,02	9,54	*13,38	*22,95	40,68	59,87	76,75
Mandrill	b	*12,11	*12,18	13,01	19,30	36,66	59,08	82,98
Mona Lisa	h	25,56	26,42	30,80	40,81	59,83	78,20	86,33
Mona Lisa	b	*43,52	*43,45	*43,60	47,96	62,91	85,31	97,79
Parabel	h	60,73	84,67	86,80	86,82	86,83	86,83	87,14
Parabel	b	81,43	95,88	98,66	*98,69	*98,71	*98,70	99,19
Peppers	h	7,15	12,26	24,58	50,94	73,90	81,92	85,42
Peppers	b	*27,66	28,07	32,19	52,59	79,56	90,97	96,46
Vancouver	h	23,75	26,07	32,59	47,82	68,59	80,83	85,88
Vancouver	b	*35,09	*35,13	*36,83	47,69	71,63	89,22	97,13
X-Wing	h	70,54	70,97	72,45	75,25	78,34	81,16	84,11
X-Wing	b	*82,24	*82,22	82,38	83,59	85,91	88,85	92,97

Tabelle 10: Kompressionsraten in Prozent für verschiedene Bilder bei Speicherung der Werte. Die Baumstruktur wurde durch Nullen aufgefüllt. Die Kodierungsverfahren sind BWT mittels `bzip2` (b) und eine angepaßte Huffman-Variante (h). Die ε-Werte kennzeichnen die Schwellenwerte für das Abschneidekriterium. Die Sterne markieren das beste Verfahren.

Datei	k	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$	$\varepsilon = 16$	$\varepsilon = 32$	$\varepsilon = 64$
Belle	h	16,50	21,32	30,25	51,17	81,70	92,90	97,68
Belle	b	18,73	23,37	31,96	52,12	81,52	92,99	97,79
Berlin	h	-19,86	-16,44	-9,91	2,21	23,27	54,18	82,34
Berlin	b	-20,64	-17,07	-10,29	2,35	24,34	56,23	83,81
Canyon	h	-14,99	-14,37	-9,91	-2,30	13,15	42,97	81,53
Canyon	b	-16,95	-16,33	-11,71	-3,81	12,31	43,32	82,29
CR Bein	h	65,56	67,49	71,21	78,64	91,79	99,29	99,78
CR Bein	b	64,41	66,42	70,30	78,09	91,61	99,32	99,79
CR Thorax	h	30,53	42,20	63,58	88,11	98,53	99,53	99,68
CR Thorax	b	26,91	39,58	62,21	87,74	98,56	99,55	99,67
CT Kopf	h	54,53	58,26	64,15	74,60	83,30	89,45	93,85
CT Kopf	b	54,78	58,70	64,79	75,59	84,38	90,37	94,43
Erft	h	*76,65	*92,36	*97,20	*99,04	99,70	99,93	*99,96
Erft	b	74,63	91,76	96,97	98,97	99,69	99,88	99,91
Lena	h	12,22	22,09	41,10	68,63	85,56	93,87	97,91
Lena	b	8,22	19,03	39,27	67,79	85,44	94,04	98,06
Mandrill	h	-15,10	-10,89	-2,97	12,44	36,24	61,43	84,61
Mandrill	b	-16,32	-11,86	-3,51	12,67	37,16	62,87	85,66
Mona Lisa	h	1,77	4,41	15,75	33,94	60,69	86,33	98,07
Mona Lisa	b	-0,18	2,48	13,82	32,06	59,58	86,08	98,11
Parabel	h	79,86	96,66	98,19	98,20	98,21	98,21	99,11
Parabel	b	83,01	*96,87	98,34	98,35	98,36	98,36	99,14
Peppers	h	7,71	15,56	30,63	57,65	82,63	92,25	96,89
Peppers	b	3,89	12,39	28,44	56,81	82,71	92,59	97,14
Vancouver	h	1,10	7,15	19,52	42,64	72,07	89,88	97,36
Vancouver	b	-3,34	3,05	16,44	40,85	71,52	89,90	97,48
X-Wing	h	71,40	71,99	74,03	77,98	82,34	86,75	92,01
X-Wing	b	73,70	74,28	76,25	80,14	84,49	88,62	93,32

Tabelle 11: Kompressionsraten in Prozent für verschiedene Bilder bei Speicherung der Koeffizienten entlang der raumfüllenden Kurve des $2m$-Verfahrens. Die Kodierungsverfahren sind BWT mittels `bzip2` (b) und eine angepaßte Huffman-Variante (h). Die ε-Werte kennzeichnen die Schwellenwerte für das Abschneidekriterium. Die Sterne markieren das beste Verfahren. (Werte kleiner Null stellen eine Vergrößerung der Ausgangsdatei dar)

Datei	k	$\varepsilon = 1$	$\varepsilon = 2$	$\varepsilon = 4$	$\varepsilon = 8$	$\varepsilon = 16$	$\varepsilon = 32$	$\varepsilon = 64$
Belle	h	33,25	35,65	40,49	54,36	73,88	81,98	85,64
Belle	b	33,54	34,91	39,13	52,47	78,71	91,00	96,89
Berlin	h	2,72	3,74	6,42	12,68	26,24	50,63	73,19
Berlin	b	1,72	1,92	3,00	7,69	22,06	50,81	80,82
Canyon	h	6,84	7,14	8,77	12,34	21,24	42,79	72,71
Canyon	b	4,11	4,27	4,76	6,65	14,31	38,99	78,57
CR Bein	h	62,27	63,68	66,37	71,74	81,37	86,95	87,33
CR Bein	b	71,56	71,69	72,62	77,37	90,18	99,04	99,73
CR Thorax	h	43,51	48,28	62,32	79,03	86,40	87,16	87,28
CR Thorax	b	40,41	44,13	59,91	85,43	98,07	99,46	99,64
CT Kopf	h	54,21	56,88	61,14	68,74	75,25	79,91	83,21
CT Kopf	b	61,49	62,25	65,40	74,20	82,83	89,11	93,54
Erft	h	75,46	82,65	85,55	86,78	87,26	87,45	87,47
Erft	b	75,06	89,58	95,53	98,25	99,48	99,85	99,92
Lena	h	*30,13	*33,64	*44,13	63,54	76,13	82,56	85,77
Lena	b	25,40	27,68	39,13	64,13	82,30	92,12	97,23
Mandrill	h	6,91	8,04	11,20	19,69	36,61	56,44	74,98
Mandrill	b	5,53	5,77	7,56	15,64	34,22	57,91	82,23
Mona Lisa	h	21,53	22,72	27,79	38,16	57,43	76,81	85,94
Mona Lisa	b	18,68	19,05	22,30	32,65	55,65	82,86	97,28
Parabel	h	77,87	85,74	86,42	86,43	86,43	86,44	86,96
Parabel	b	*88,16	96,72	98,14	98,17	98,18	98,17	98,89
Peppers	h	27,11	*29,60	*36,79	55,62	73,97	81,36	85,01
Peppers	b	22,36	23,61	30,83	53,15	79,33	90,48	96,14
Vancouver	h	21,36	23,38	29,14	43,98	65,87	79,54	85,41
Vancouver	b	16,49	17,13	21,61	38,31	67,62	87,56	96,60
X-Wing	h	68,45	68,88	70,29	72,96	76,07	79,25	83,00
X-Wing	b	77,77	77,94	78,77	81,23	84,67	88,48	93,30

Tabelle 12: Kompressionsraten in Prozent für verschiedene Bilder bei Speicherung der Koeffizienten. Die Baumstruktur wurde durch Nullen aufgefüllt. Die Kodierungsverfahren sind BWT mittels `bzip2` (b) und eine angepaßte Huffman-Variante (h). Die ε-Werte kennzeichnen die Schwellenwerte für das Abschneidekriterium. Die Sterne markieren das beste Verfahren.

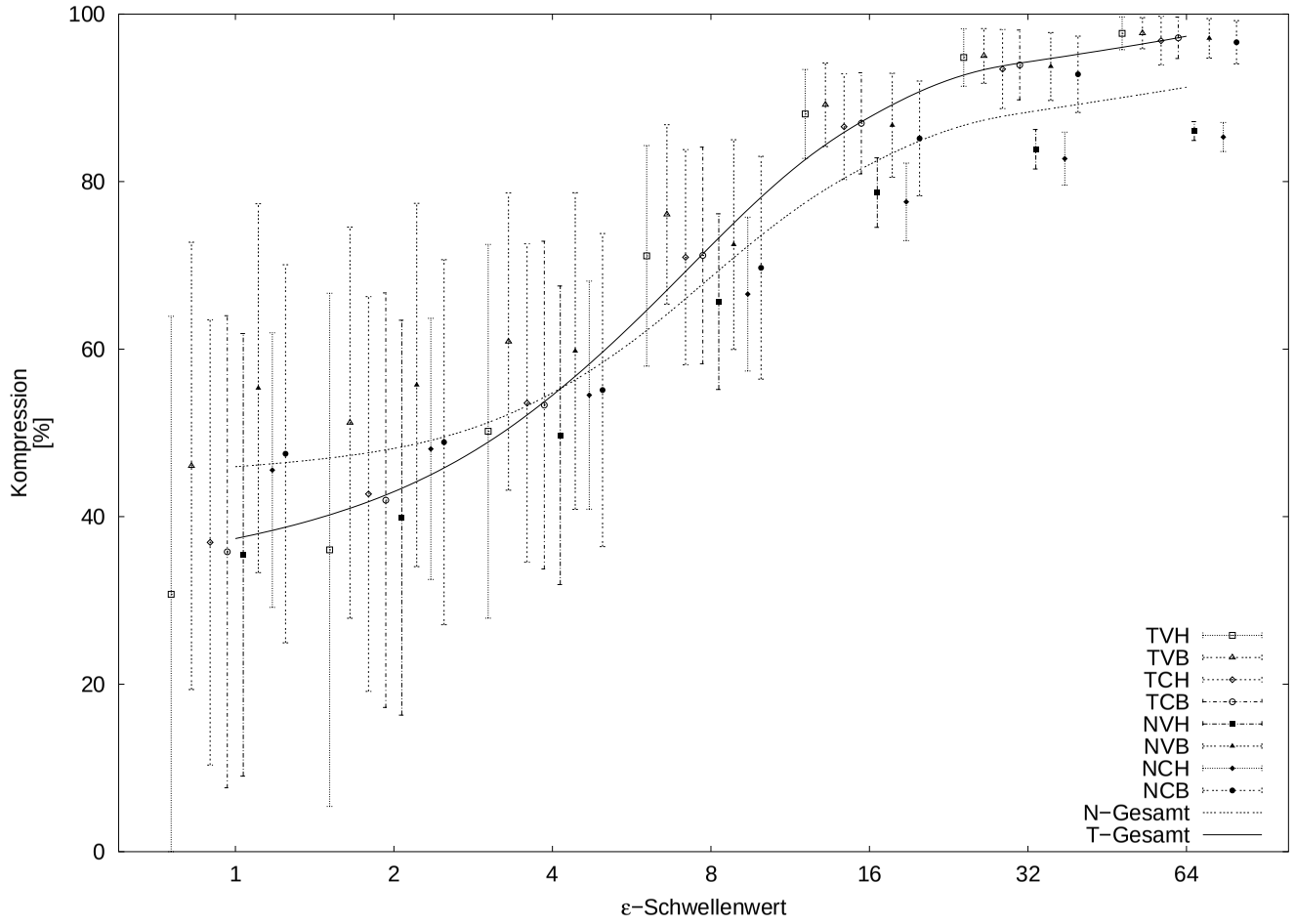


Abbildung 53: Vergleich der verschiedenen Kompressionsverfahren für die hierarchische Triangulierung. Dabei steht T für das $2m$-Verfahren, N zeigt an, daß der Baum durch Nullen aufgefüllt wurde; V,C deutet Werte- bzw. Koeffizienten-Abspeicherung an. Die Kodierungsverfahren sind Burrows-Wheeler B und Huffman H. Die dargestellten Werte sind die Mittelwerte aus den Dateien *Belle*, *CT Kopf*, *CR Bein*, *CR Thorax*, *Lena*, *Peppers* und *X-Wing* mit der Standardabweichung als Fehlerbalken. T- und N-Gesamt sind die gemittelten Mittelwerte der T- bzw. N-Verfahren.

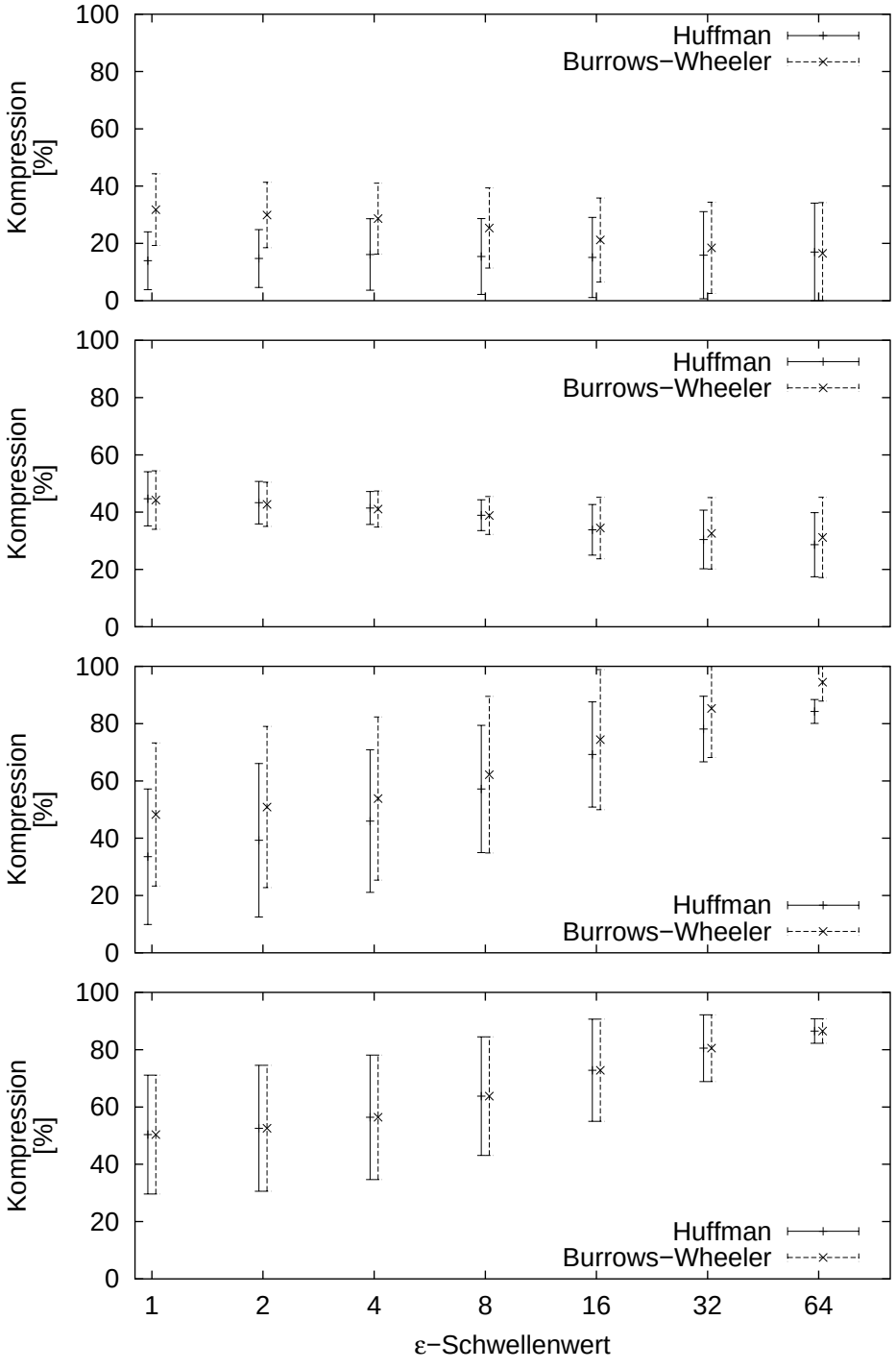


Abbildung 54: Vergleich zwischen Burrows-Wheeler-Transformation und Huffman-Kodierung. Die Prozentangaben beziehen sich auf die Größe der nicht komprimierten Ausgangsdaten. Die Daten wurden folgendermaßen berechnet (von oben nach unten): Werte mit $2m$-Verfahren, Koeffizienten mit $2m$-Verfahren, Werte mit Nullen aufgefüllt und Koeffizienten mit Nullen aufgefüllt. Die dargestellten Werte sind die Mittelwerte aus 14 Dateien (siehe Abbildungen 59 und 60) mit der Standardabweichung als Fehlerbalken.

9.3 Fraktale

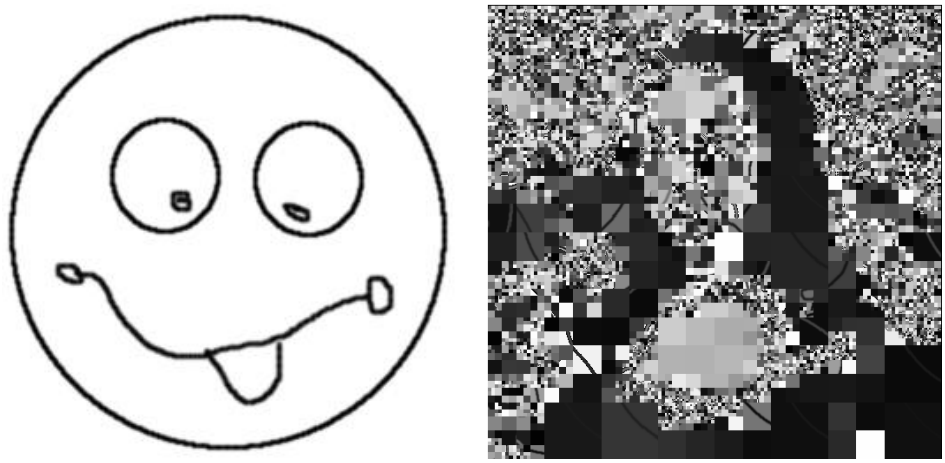
Für die in Abbildung 55 dargestellten Bilder wurde eine adaptive fraktale Kompression verwandt. Diese geht von vier gleich großen, das ganze Bild erfassenden, quadratischen Regionen aus. Die dazugehörigen Domänen besitzen jeweils die doppelte Kantenlänge der Regionen und sind nichtüberlappend auf das gesamte Bild verteilt.

Die Regionen werden rekursiv in vier gleich große Quadrate aufgeteilt, wenn keine Domäne mit den möglichen Transformationen die Region mit einem Fehler kleiner als 15 approximiert. Der Fehler wird in der L_2-Norm gemessen.

Als Transformationen stehen in dem verwandten Programm Drehungen um 0°, 90°, 180°, 270°, eine horizontale Spiegelung sowie eine Helligkeits- und Kontrastanpassung zur Verfügung.

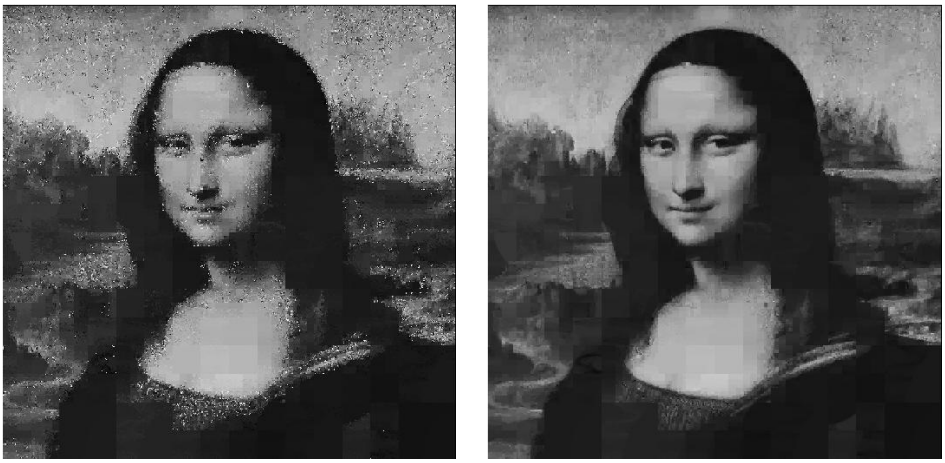
Die Synthese beginnt mit einem beliebigen Ausgangsbild und bildet die ausgesuchte Domäne mit den oben erwähnten Transformationen in die zugehörige Region ab. Die Domänen werden in jedem Iterationsschritt in ein neues Bild kopiert, das als Ausgangsbild für den nächsten Iterationsschritt dient. Die Iterationstiefe wird beim Programmstart vorgegeben.

In Abbildung 55 rechts oben ist gut zu sehen, wie die fraktale Transformation sich den lokalen Eigenschaften des Ausgangsbildes anpaßt.



Iteration 0 Iteration 1

Iteration 2 Iteration 3



Iteration 4 Iteration 15

Abbildung 55: Fraktale Kopiermaschine mit 6856 Regionen für die *Mona Lisa*. Ausgangsbild ist ein *Smiley* (oben links). Abgebildet sind die Iterationen 0, 1, 2, 3, 4 und 15 (von links oben nach rechts unten). Eine Iteration von Null entspricht dem Ausgangsbild. Der Fehler für die Kompression liegt bei 15.

10 Zusammenfassung und Ausblick

In dieser Arbeit wurden Daten-Kompressionsverfahren vorgestellt, bei denen besonderer Wert auf die Beschränkung des Fehlers in der Maximumsnorm durch einen vorgegebenen Schwellenwert ε gelegt wurde. Die Kompression bestand dabei aus drei Komponenten: der Transformation T, der Kompression K und der Kodierung C.

Für die Transformation T wurden vier Verfahren vorgestellt: die fraktale, Fourier-, Wavelet- und Hierarchische Basis-Transformation. Von diesen vier Transformationen hat sich die Hierarchische Basis durch sehr gute $\mathcal{O}(N)$-Komplexität, gute Frequenz-Lokalisierbarkeit, Multiskalen-Repräsentation und einfache Randbehandlung hervorgetan. Die weiteren Kompressions-Komponenten K und C wurden daher auf diese Transformation abgestimmt.

Bei der Kompression K wurden Quantisieren und *thresholding*-Methoden vorgestellt, die kombiniert angewandt werden können. Für die *thresholding*-Methoden wurden a priori und adaptive Fehlerbeschränkungs-Varianten sowohl für den Tensorprodukt-Ansatz als auch für die hierarchische Triangulierung konstruiert.

Die Kodierer C wurden in zwei Bereichen getrennt behandelt. Die Struktur-Kodierer C_s und die Daten-Kodierer C_d. Für die hierarchische Triangulierung wurden als Struktur-Kodierer C_s drei Baum-Kodierer vorgestellt, von denen das $2m$-Verfahren auf Basis einer raumfüllenden Kurve die beste Speicherkomplexität bot. Es wurde zusätzlich eine generelle Variante der Struktur-Kodierung, das Null-Setzen, vorgestellt. Dieses kann sowohl für den Tensorprodukt-Ansatz als auch für die hierarchische Triangulierung verwandt werden.

An gleicher Stelle wurden noch die Wavelet-Basierten Kompressions-Verfahren Embedded Zerotree for Wavelets (EZW) und Set Partitioning in Hierarchical Trees (SPIHT) kurz vorgestellt. Bei diesen waren die Kompressions-Komponenten T, K und C in einem komplexen Algorithmus vereint. Sie weisen sehr gute Kompressionsraten auf, eine Fehlerkontrolle in der Maximumsnorm ist allerdings nicht möglich.

Als Daten-Kodierer C_d wurden die explizite Speicherung, Run Length Encoding (RLE), Lempel-Ziv-Welsh (LZW), Huffman- und arithmetische Kodierung und die Burrows-Wheeler Transformation (BWT) vorgestellt. Besonders bewährt haben sich die Huffman-Kodierung und die BWT als Daten-Kodierer. Bei den Struktur-Kodierern waren je nach vorgegebenem Fehler die Verfahren $2m$ (Fehler im Bereich von 0 bis 4) oder Null-Setzen (Fehler 4 und mehr) im Vorteil.

Die hierarchische Triangulierung war insgesamt dem Tensorprodukt-Ansatz überlegen, sowohl in Bezug auf die Kompressionsraten als auch der vi-

suellen Eigenschaften. Dies zeigte sich in einer Reihe von Experimenten.

Als Erweiterungen wurden für die hierarchische Triangulierung die Farbbild-Kodierung im RGB- und YC_bC_r-Farbraum gezeigt. Ebenso wurden einfache Möglichkeiten gezeigt, die quadratischen Grundgebiete auf Rechtecke oder beliebige Gebiete zu erweitern. Zudem wurde vorgestellt, wie ausgewählte Bereiche eines Bildes mit einer Maske fokussiert werden können.

Über die in dieser Arbeit vorgestellten Verfahren wäre eine Erweiterung der Fehlerbeschränkungs-Methoden im Triangulierungs-Fall auf höhere Dimensionen, etwa 3-D oder sogar 4-D, interessant. Die dreieckigen Träger der Basisfunktionen könnten durch Tetraeder (Gerstner und Rumpf 2000) ersetzt werden. Mit einer solchen Erweiterung wäre es möglich 3-D-Daten wie sie in der Medizin bei CT- oder MRT-Bildern vorkommen auch mit Fehlerschranken in der Maximumsnorm zu speichern oder darzustellen.

Eine Erweiterung auf allgemeinere und vor allem größere Gebiete als die in dieser Arbeit untersuchten, würde die Anwendungsmöglichkeiten der hier vorgestellten Methoden erheblich vergrößern. So ist in (Gerstner 2000) ein Verfahren vorgestellt worden, das große digitale Höhenmodelle in Echtzeit zu visualisieren ermöglicht. Auch hier wäre eine Beschränkung des Fehlers in der Maximumsnorm interessant.

Bei den Struktur-Kodierern der hierarchischen Triangulierung kann durch Symmetrie-Ausnutzung die Kompressionsrate noch etwas gesteigert werden.

Bei den Daten-Kodierern könnten einige der vorgestellten Verfahren, insbesondere die Burrows-Wheeler-Transformation und die arithmetische Kodierung, besser an die Anforderungen der hierarchischen Triangulierung angepaßt werden. Auch dynamische Wörterbücher, die auf die Level-Struktur der Hierarchischen Basis abgestimmt sind, könnten für höhere Kompressionsraten von Vorteil sein.

A Anhang

A.1 Test-Bilder

In den Abbildungen 56–58 sind alle Bilder dargestellt, die für das Testen der Kompressionsverfahren verwandt wurden. Die Größe der Bilder ist unter den Bildern neben deren Namen vermerkt. Die bearbeiteten Bilder sind:

Belle: Bild aus Disney's Zeichentrickfilm „Die Schöne und das Biest“ (`belle`).

Berlin: Satellitenaufnahme von Berlin (`SatBerlin`).

Canyon: Foto einer Gebirgslandschaft (`canyon_513`).

CR-Bein: CR-Aufnahme eines Beines (`cr_bein`).

CR-Thorax: CR-Aufnahme eines Brustkorbes (`cr_thorax`).

CT-Kopf: CT-Aufnahme eines Kopfes (`ct_kopf`).

Erft: Graustufen-Darstellung eines DHM der Eifel (`erft_demo`).

Lena: Frau mit Hut. (`lena_513`).

Mandrill: Gesicht eines Mandrills. (`mandrill_513`).

Mona Lisa: Bildnis der Mona Lisa von Leonardo da Vinci (`mona_513`).

Parabel: Graustufen-Darstellung einer 2-D-Parabel eingeschränkt auf eine Kreisfläche (`para2d`).

Peppers: Verschiedene Paprika-Sorten. (`peppers`).

Vancouver: Satellitenbild von Vancouver (`SatVancouver`).

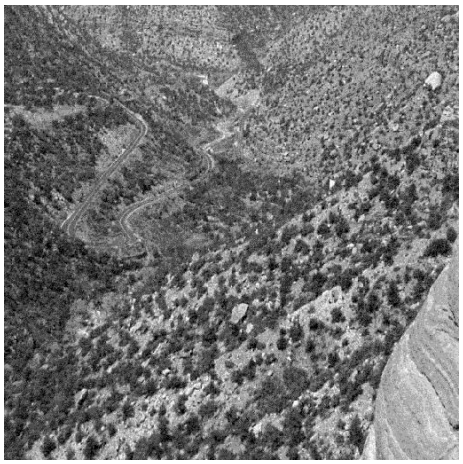
X-Wing: Weltraumschiff aus Star Wars (`xwing_513`).

Die oben aufgeführten Bilder wurden ausgewählt, um eine möglichst breite Streuung von verschiedenen Bildtypen zu erhalten. So sind verschieden stark detaillierte Fotos vorhanden mit *Canyon*, *Lena*, *Mandrill* und *Peppers*. Als Vertreter von gemalten Bildern sind die Bilder *Belle* und *Mona Lisa* ausgewählt worden. Wobei die *Mona Lisa* sehr viel feinere Strukturen aufweist als *Belle*. Der *X-Wing* stellt ein mit dem Computer erstelltes Bild dar. Die *Parabel* ist ein vollkommen synthetischer Testfall. Die Bilder *Berlin* und *Vancouver* sind Luftaufnahmen und *Erft* ein DHM. Als medizinische Bilder

wurden *CR-Bein*, *CR-Thorax* und *CT-Kopf* ausgewählt. Diese unterscheiden sich sowohl in der Größe als auch in der Konzentration der Informationen in den Bildern.

Belle 513 × 513 Pixel

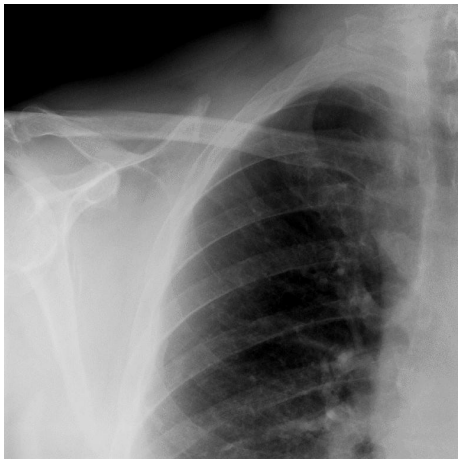
Berlin 1024 × 1024 Pixel



Canyon 513 × 513 Pixel

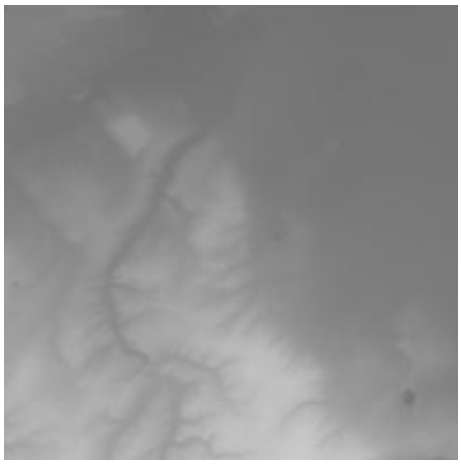
CR-Bein 1024 × 1024 Pixel

Abbildung 56: In dieser Arbeit verwandte Bilder (von oben links nach unten rechts): *Belle*, *Berlin*, *Canyon* und *CR-Bein*.



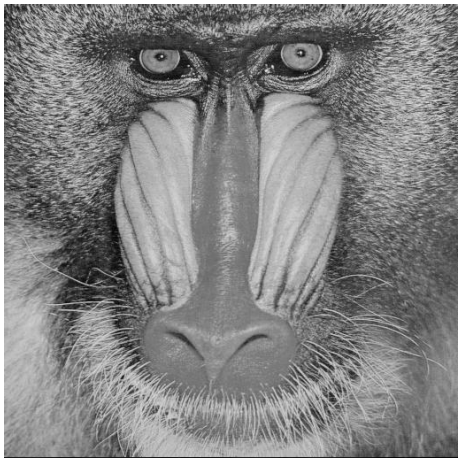
CR-Thorax 1024 × 1024 Pixel

CT-Kopf 513 × 513 Pixel

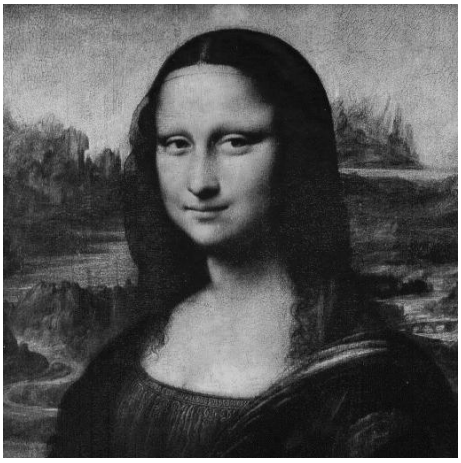


Erft 257 × 257 Pixel

Lena 513 × 513 Pixel

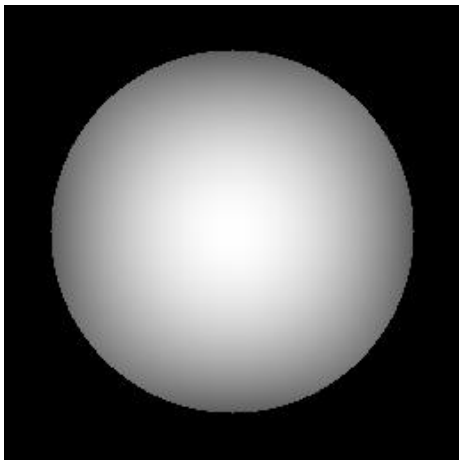


Mandrill 513 × 513 Pixel



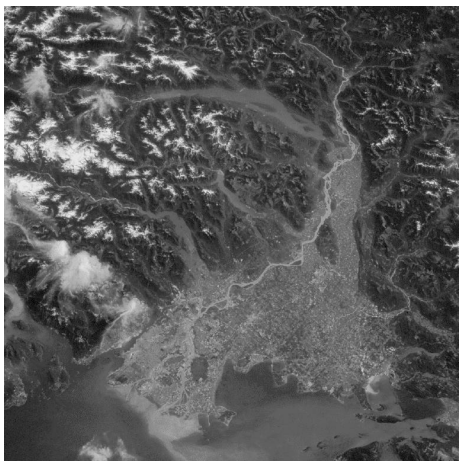
Mona Lisa 513 × 513 Pixel

Abbildung 57: In dieser Arbeit verwandte Bilder (von oben links nach unten rechts): *CR-Thorax*, *CT-Kopf*, *Erft*, *Lena*, *Mandrill* und *Mona Lisa*.

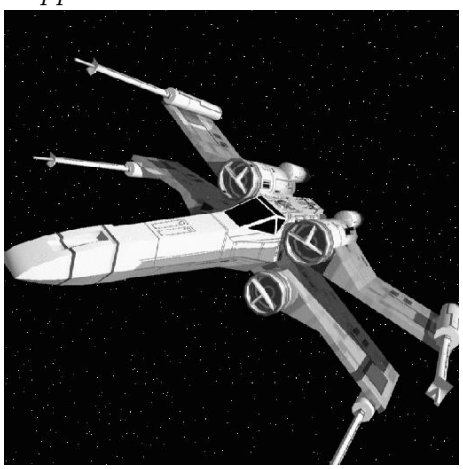


Parabel 257×257 Pixel

Peppers 513×513 Pixel



Vancouver 1024×1024 Pixel



X-Wing 513×513 Pixel

Abbildung 58: In dieser Arbeit verwandte Bilder (von oben links nach unten rechts): *Parabel*, *Peppers*, *Vancouver* und *X-Wing*.

A.2 Kompressionstabellen

Die Tabellen 59–62 stellen die prozentualen Kompressionsraten bzw. Dateigrößen für die in den Abbildungen 56–58 vorgestellten Bilder dar. Die einzelnen Abschnitte in den Tabellen sind dabei wie folgt aufgebaut: für jedes Bild sind acht (für die Kompressionsraten) bis zwölf Balken (für die Dateigrößen) aufgetragen. Bei den prozentualen Größen sind die dargestellten Verfahren $2m$ mit Werten und Huffman, $2m$ mit Werten und BWT, Null-Setzen mit Werten und Huffman, Null-Setzen mit Werten und BWT, $2m$ mit Koeffizienten und Huffman, $2m$ mit Koeffizienten und BWT, Null-Setzen mit Koeffizienten und Huffman und Null-Setzen mit Koeffizienten und BWT. Bei den Dateigrößen sind noch die Dateigrößen ohne Kodierung jeweils vor den entsprechenden Kompressionsverfahren dargestellt. Hierbei ist zu beachten, daß das Null-Setzen alle Koeffizienten bzw. Werte abspeichert und erst der anschließende Kodierer diese reduziert, d. h. diese Größen ändern sich nicht mit zunehmender Fehlerschwelle ε. Daher wurden diese Werte nur im Bild für eine Fehlerschwelle ε von Eins unbeschnitten gezeigt.

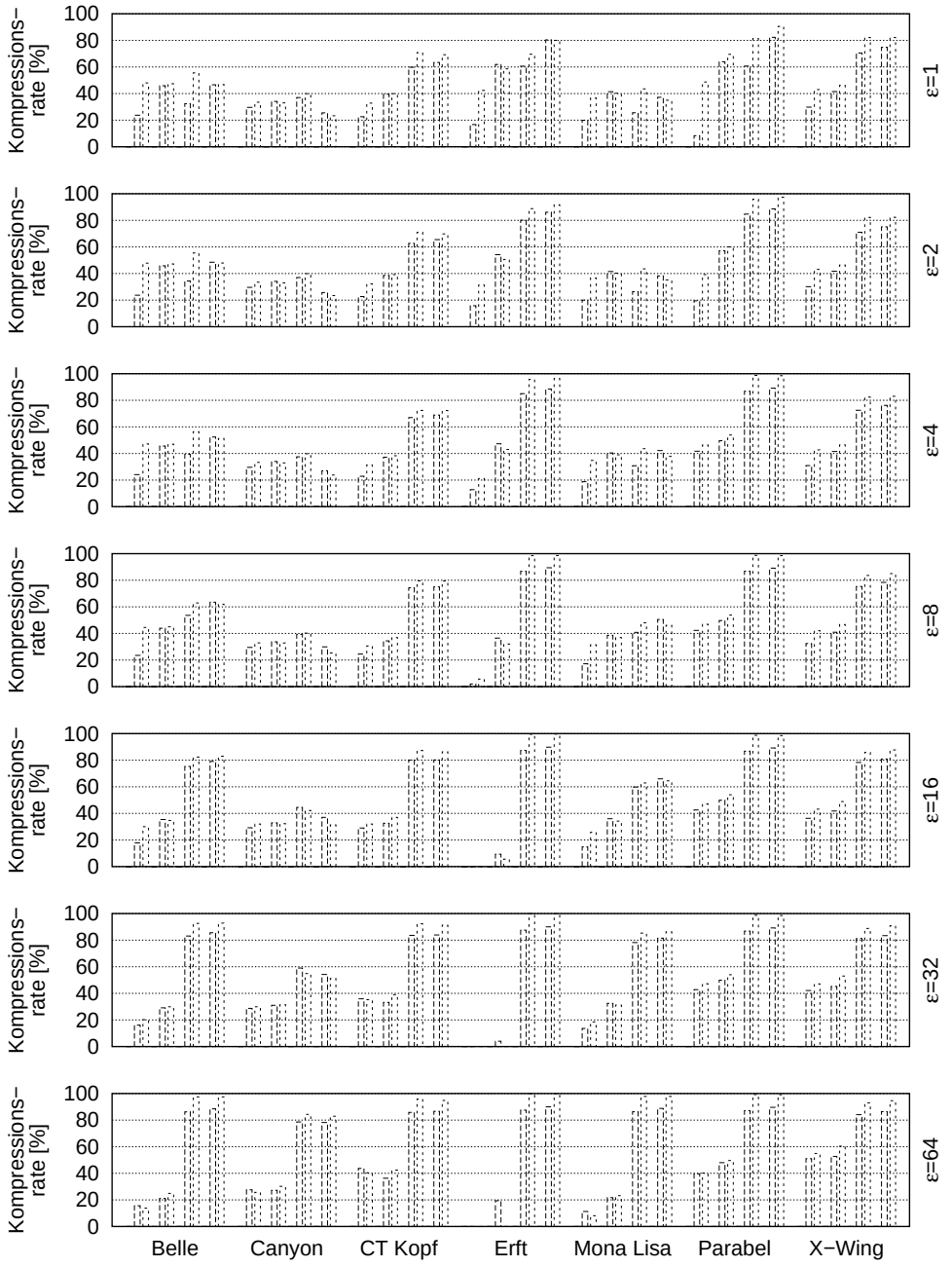


Abbildung 59: Prozentuale Kompressionsraten für verschiedene Bilder bei Fehlerschwellen ε von 1, 2, 4, 8, 16, 32 und 64. Für jedes Bild sind 8 Werte aufgetragen. Die zu diesen Werten führenden Verfahren sind (jeweils von links nach rechts): $2m$-Verfahren mit Werten und Huffman bzw. BWT; Nullen mit Werten und Huffman bzw. BWT; $2m$-Verfahren mit Koeffizienten und Huffman bzw. BWT und Nullen mit Koeffizienten und Huffman bzw. BWT.

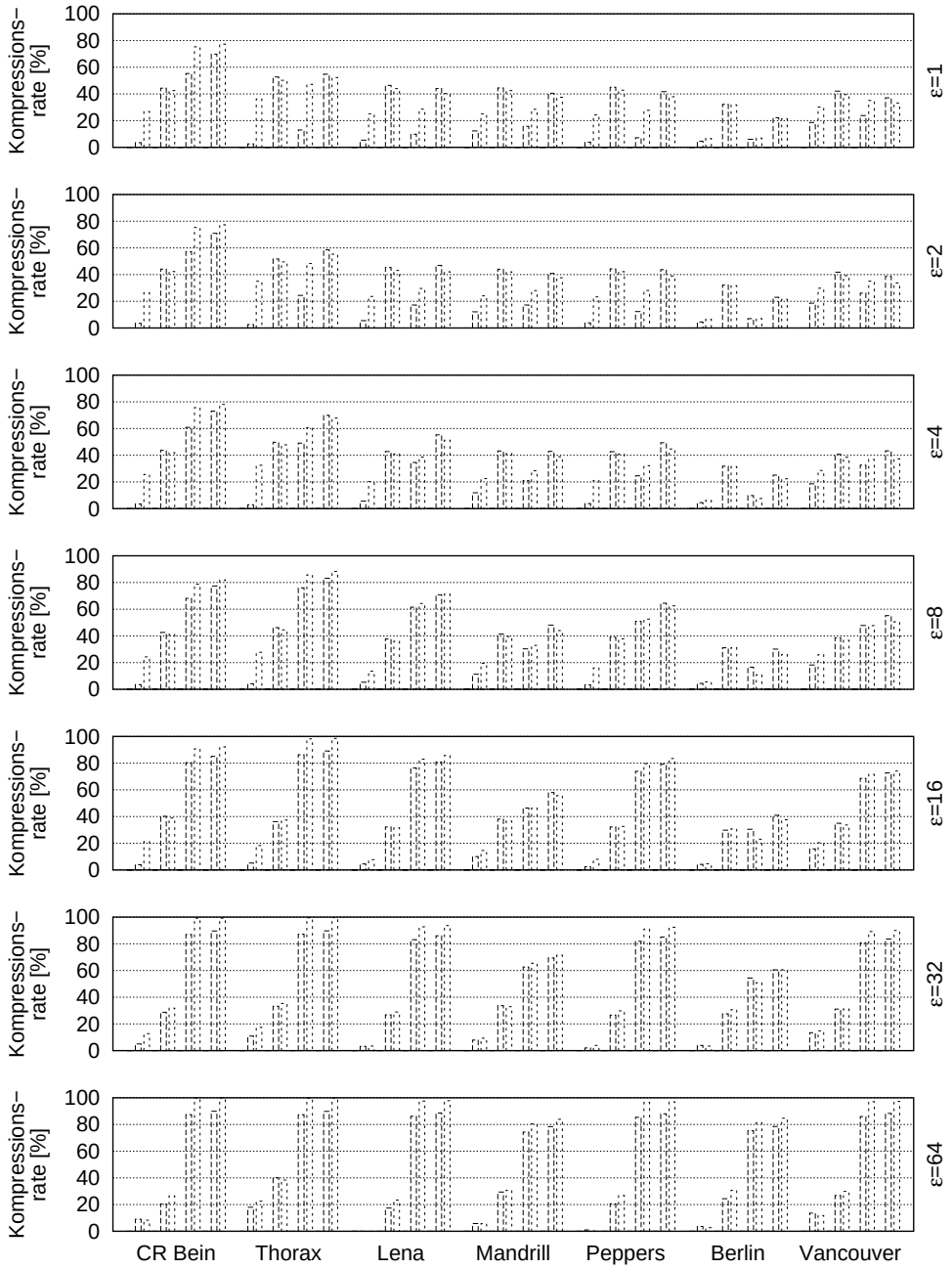


Abbildung 60: Prozentuale Kompressionsraten für verschiedene Bilder bei Fehlerschwellen ε von 1, 2, 4, 8, 16, 32 und 64. Die zu diesen Werten führenden Verfahren sind (jeweils von links nach rechts): $2m$-Verfahren mit Werten und Huffman bzw. BWT; Nullen mit Werten und Huffman bzw. BWT; $2m$-Verfahren mit Koeffizienten und Huffman bzw. BWT und Nullen mit Koeffizienten und Huffman bzw. BWT.

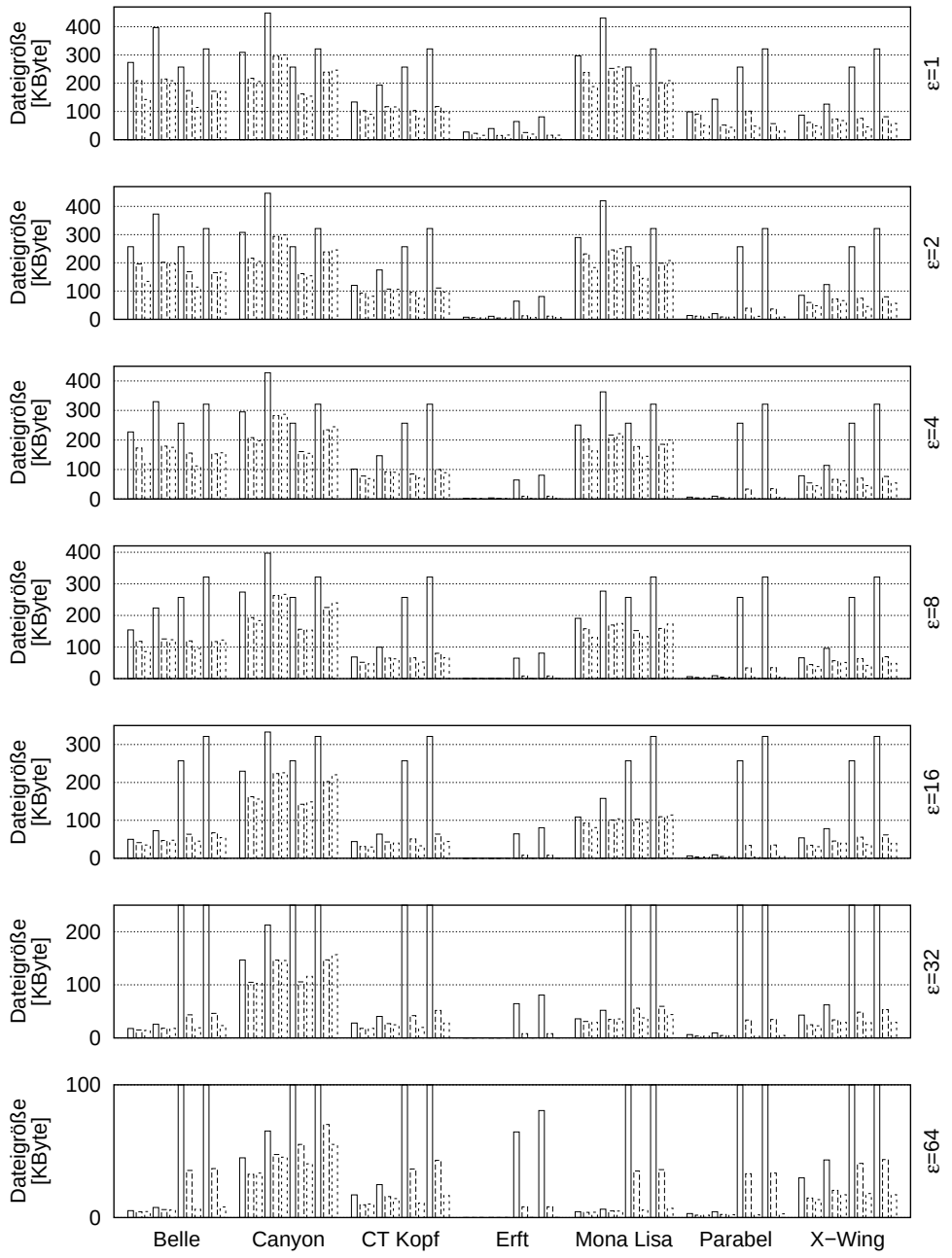


Abbildung 61: Dateigrößen für verschiedene Bilder bei Fehlerschwellen ε von 1, 2, 4, 8, 16, 32 und 64. Für jedes Bild sind 12 Werte aufgetragen. Die zu diesen Werten führenden Verfahren sind (jeweils von links nach rechts): $2m$-Verfahren mit Werten und keiner Kodierung, Huffman bzw. BWT; Nullen mit Werten und keiner Kodierung, Huffman bzw. BWT; keiner Kodierung, $2m$-Verfahren mit Koeffizienten und keiner Kodierung, Huffman bzw. BWT und Nullen mit Koeffizienten und keiner Kodierung, Huffman bzw. BWT. Die Dateigröße der Werte der Null-Auffüllverfahren ohne Kodierung bleiben bei allen Fehlerschwellen ε konstant, wurden aber zur besseren Visualisierung der anderen Werte in den unteren Bildern abgeschnitten.

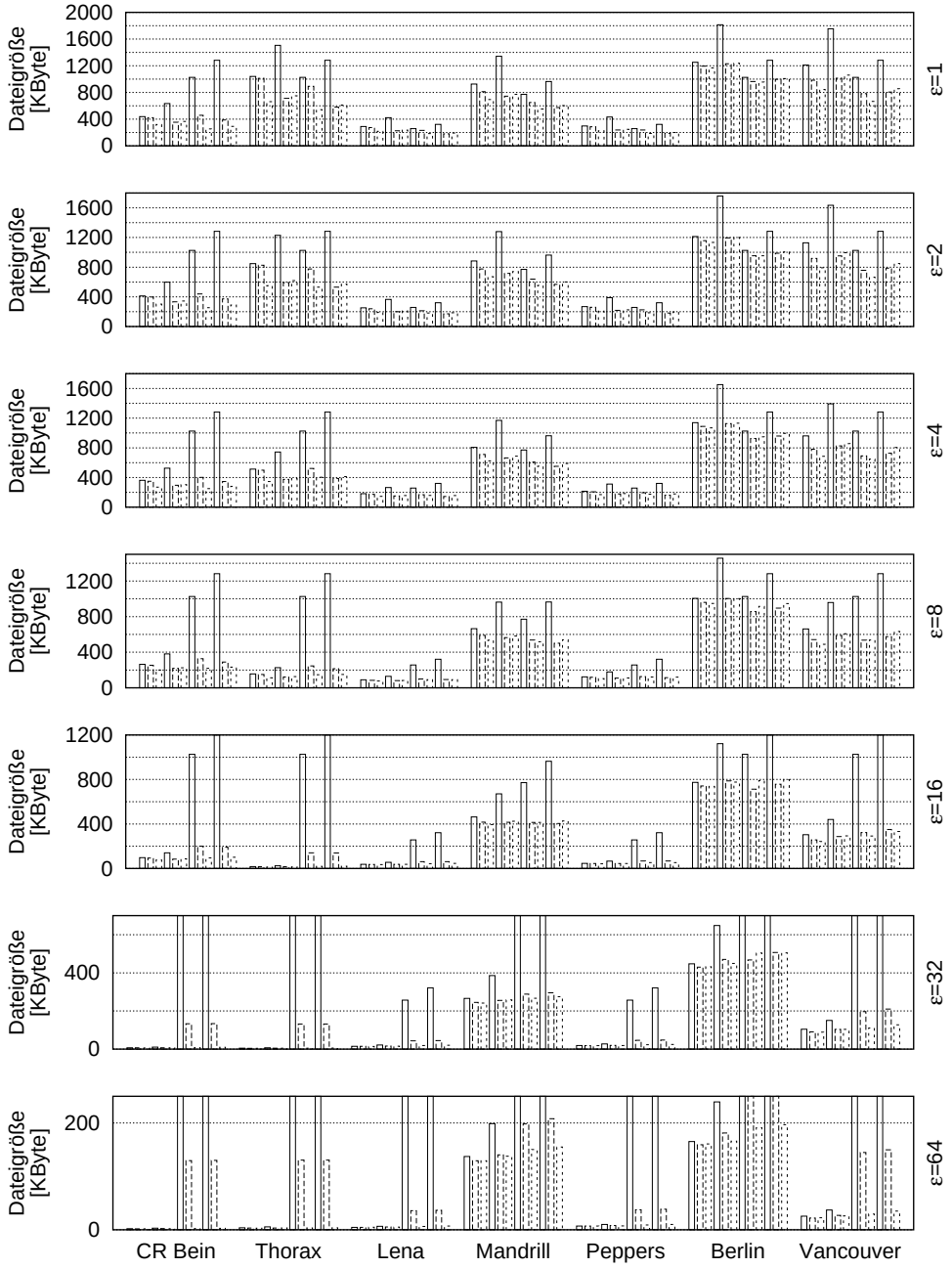


Abbildung 62: Dateigrößen für verschiedene Bilder bei Fehlerschwellen ε von 1, 2, 4, 8, 16, 32 und 64. Für jedes Bild sind 12 Werte aufgetragen. Die Anordnung der Werte entspricht der in Abbildung 61.

A.3 Symbolverzeichnis

A.3.1 Variablen und Konstanten

$\lvert . \rvert$	Absolutbetrag
$\lvert\lvert . \rvert\rvert$	Norm
$\lceil . \rceil$	Nächstgrößere ganze Zahl
$\lfloor . \rfloor$	Nächstkleinere ganze Zahl
$\mathbf{a}, \mathbf{b}$	Masken für Waveletes
C, C^{-1}	Kodierung, Inverse der Kodierung
C_d	Daten-Kodierung
C_s	Struktur-Kodierung
D_i	Domänen
$\mathcal{D}(i,j)$	Alle Nachfahren bei SPIHT
ε	Fehlerwert
$\mathbf{f}$	Datenvektor
$\hat{\mathbf{f}}$	Transformierter Datenvektor
$\tilde{\mathbf{f}}$	Verlustbehaftet rücktransformierter Datenvektor
$\mathbf{F}$	Bild
I	Identität
K, K_*^{-1}	Kompression, Pseudo-Inverse der Kompression
$L_2(\mathbb{R})$	Raum der quadratintegrierbaren Funktionen über $\mathbb{R}$
$\mathcal{L}(i,j)$	Nachfahren ohne Söhne bei SPIHT
$\mathbb{N}$	Menge der natürlichen Zahlen
$\mathcal{O}(N)$	Komplexität
$\mathcal{O}(i,j)$	Direkte Söhne bei SPIHT
$\mathbb{R}$	Menge der reellen Zahlen
R_i	Regionen
T, T^{-1}	Transformation, Inverse der Transformation
$T_{i_{(j)}}$	Teilgebiete
$\mathcal{T}$	Menge aller Teilgebiete
V_l, W_l	Wavelets-Räume
$\mathbb{Z}$	Menge der ganzen Zahlen
ξ	Glattheitsparameter
φ	Skalierungsfunktion
ψ	Wavelet
Ω	Gebiet

A.3.2 Abkürzungen

bpp	Bit per Pixel
BWT	Burrows-Wheeler-Transformation
CT	Computer Tomographie
DCT	Diskrete Cosinus-Transformation
DHM	Digitale Höhenmodelle
EOM	End of Message
EZW	Embedded Zerotrees for Wavelets
fft	fast fourier transformation
fwt	fast wavelet transformation
JPEG	Joint Photographic Expert Group
LIP	Liste der insignifikanten Pixel
LIS	Liste der insignifikanten Mengen
LSP	Liste der signifikanten Pixel
MRT	Magnet Resonanz Tomographie
MTF	move to front
RGB	Rot-Grün-Blau-Farbkodierung
RLE	Run Length Encoding
SGI	Silicon Graphics, Inc.
SPIHT	Set Partitioning in Hierachical Trees
YC_bC_r	Farbkodierung

Literatur

Barnsley und Sloan 1987 Barnsley, M. F. ; Sloan, A. D.: *Chaotic compression.* Computer Graphics World, November 1987

Barthel u. a. 1997 Barthel, Kai U. ; Brandau, Sven ; Hermesmeier, Wolfgang ; Heising, Guido: Zerotree Wavelet Coding Using Fractal Prediction. In: *Proceedings ICIP-97 (IEEE International Conference on Image Processing)*, 1997

Burrows und Wheeler 1994 Burrows, M. ; Wheeler, D. J.: A Block-Sorting Lossless Data Compression Algorithm / Digital System Research Center. 1994. – Forschungsbericht

Capon 1959 Capon, J.: A Probabilistic Model for Run-Length Coding of Pictures. In: *IRE Transactions on Information Theory.* 1959, S. 157–163

Carpinelle u. a. 1999 Carpinelle, John ; Moffat, Alistair ; Neal, Radford ; Salamonsen, Wyne ; Stuiver, Lang ; Turpin, Andrew ; Witten, Ian. *arith_coder.* http://www.cs.mu.oz.au/~alistair. 1999

Dahmen 1997 Dahmen, W.: Wavelet and Multiscale Methods for Operator Equations. In: *Acta Numerica.* 1997, S. 55–228

Davis 1997 Davis, Geoffrey M.: A Wavelet–Based Analysis of Fractal Image Compression. In: *IEEE Transactions on Image Processing.* 1997

Faber 1909 Faber, G.: Über stetige Funktionen. In: *Mathematische Annalen* 66 (1909), S. 81–94

Fisher 1992 Fisher, Y.: Fractal Image Compression. In: *SIGGRAPH Course Notes*, 1992

Fisher u. a. 1994 Fisher, Y. ; Rogovin, D. ; Shen, T. P.: A Comparison of Fractal Methods with DCT and Wavelets / Institute for Nonlinear Science. 1994. – Forschungsbericht

Gerstner 2000 Gerstner, T.: Multiresolution Visualization and Compression of Global Topographic Data. In: *GeoInformatica* (2000). – in Revision (gekürzte Version in Proc. Spatial Data Handling 2000, P. Forer (Hrsg.), A.G.O. Yeh (Hrsg.), J. He (Hrsg.), 2000, auch als SFB 256 Report 29, Univ. Bonn, 1999)

Gerstner und Rumpf 2000 Gerstner, T. ; Rumpf, M.: Multiresolutional Parallel Isosurface Extraction based on Tetrahedral Bisection. In: Chen, M. (Hrsg.) ; Kaufman, A. (Hrsg.) ; Yagel, R. (Hrsg.): *Volume Graphics*, Springer, 2000. – (auch als SFB 256 report 23, Univ. Bonn, 1999), S. 267–278

Gerstner 1995 Gerstner, Thomas: *Ein adaptives hierarchisches Verfahren zur Approximation und effizienten Visualisierung von Funktionen und seine Anwendung auf digitale 3-D Höhenmodelle*, Technische Universität München Institut für Informatik, Diplomarbeit, Juni 1995

Gross u. a. 1996 Gross, M. ; Staadt, O. G. ; Gatti, R.: Efficient Triangular Surface Approximations using Wavelets and Quadtree Data Structures. In: *IEEE Trans. on Visualization and Computer Graphics* Bd. 2. 1996

Huffman 1951 Huffman, D. A.: A Method for the Construction of Minimum Redundancy Codes. In: *Proceedings of the IRE* Bd. 40. 1951, S. 1098–1101

Knuth 1985 Knuth, D. E.: Dynamic Huffman Coding. In: *Journal of Algorithms* 6 (1985), S. 163–180

Kumar und Foufoula-Georgiou 1997 Kumar, Praven ; Foufoula-Georgiou, Efi: Wavelet Analysis For Geographical Applications. In: *Reviews of Geophysics* 35 (1997), November, S. 385–412

Lindstrom u. a. 1996 Lindstrom, P. ; Koller, D. ; Ribarsky, W. ; Hodges, L. F. ; Faust, N.: Real-Time, Continuous Level of Detail Rendering of Height Fields. In: *Computer Graphics (Proc. SIGGRAPH '96)*. 1996

Nelson 1996 Nelson, Mark: Data Compression with the Burrows–Wheeler Transformation. In: *Dr. Dobb's Journal* (1996), September

Ohlberger und Rumpf 1998 Ohlberger, M. ; Rumpf, M.: Adaptive Projection Methods in Multiresolutional Scientific Visualization. In: *IEEE Transactions on Visualization and Computer Graphics* Bd. 4. 1998

Poynton 1997 Poynton, Charles A. *Frequently Asked Questions about Color*. http://www.inforamp.net/~poynton. März 1997

Press u. a. 1992 Kap. 20 Less–Numerical Algorithms In: Press, William H. ; Teukolsky, Saul A. ; Vetterling, William T. ; Flannery,

Brian P.: *Numerical Recipes in C: The Art of Scientific Computing*. Second Edition. Cambridge University Press, 1992

Rivara 1984 Rivara, M. C.: Algorithms for Refining Triangular Grids Suitable for Adaptive and Multigrid Techniques. In: *International Journal for Numerical Methods in Engineering* (1984), Nr. 20, S. 745–756

Sagan 1994 Sagan, H.: *Space-filling Curves*. Springer, 1994

Said und Pearlman 1993 Said, Amir ; Pearlman, William A.: An Image Multiresolution Representation for Lossless and Lossy Compression. In: *SPIE Symposium on Visual Communications and Image Processing*. Cambridge, MA, November 1993

Said und Pearlman 1996 Said, Amir ; Pearlman, William A.: A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees. In: *IEEE Transaction on Circuits and Systems for Video Technology* Bd. 6. 1996

Samet 1985 Samet, Hanan: Data Structures For Quadtree Approximation And Compression. In: *Comm. ACM* (1985), Nr. 28, S. 973–993

Samet 1989 Samet, Hanan: *Applications of spatial Data Structures*. Addison Wesley, 1989

Saupe 1995 Saupe, D.: Accelerating Fractal Image Compression by Multi–Dimensional Nearest Neighbour Search. In: *Proceedings DCC' 95 Data Compression Conference*, 1995

Saupe und Hamzaoui 1994 Saupe, Dietmar ; Hamzaoui, Raouf: *A Guided Tour of the Fractal Image Compression Literature*. Universität Freiburg, July 1994

Sayood 2000 Sayood, Khalid: *Introduction to Data Compression*. Morgan Kaufmann Puplishers, 2000

Shannon 1948 Shannon, C. E.: A Mathematical Theory of Communication. In: *Bell System Technical Journal* 27 (1948), S. 379–423, 623–656

Shapiro 1993 Shapiro, J.: Embedded Image Coding Using Zerotrees Of Wavelet Coefficients. In: *IEEE Trans. Signal Proc.* Bd. 41. 1993, S. 3445–3462

Sierpiński 1912 Sierpiński, W.: *Sur une nouvelle courbe continue qui remplit toute une aire plane*. Bull. Acad. Sci. de Cracovie, 1912

Valens 1999 Valens, C.: Embedded Zerotree Wavelet Encoding / mindless.com. 1999. – Forschungsbericht

Vidaković und Müller 1993 Vidaković, Brani ; Müller, Peter: Wavelets for Kids / Duke University. 1993. – Forschungsbericht

Vitter 1987 Vitter, J. S.: Design and Analysis of Dynamic Huffman Codes. In: *Journal of ACM* Bd. 34(4). 1987, S. 825–845

Welsh 1984 Welsh, T. A.: A Technique for High-Performance Data Compression. In: *IEEE Computer*. 1984, S. 8–19

Witten u. a. 1987 Witten, I. H. ; Neal, R. M. ; Cleary, J. G.: Arithmetic coding for data compression. In: *Commun. ACM* Bd. 20. 1987, S. 520–540

Woo u. a. 1997 Woo, Mason ; Neider, Jackie ; Davis, Tom: *OpenGL Programming Guide*. Second Edition. Addison Wesley Developers Press, Oktober 1997

Zenger 1991 Zenger, C.: Sparse grids. In: *Parallel Algorithms for Partial Differential Equations: Proceedings of the 6th GAMM-Seminar*. Vieweg, 1991

Ziv und Lempel 1977 Ziv, J. ; Lempel, A.: A Universal Algorithm for Data Compression. In: *IEEE Transactions on Information Theory* Bd. 23(3). 1977, S. 337–343

Ziv und Lempel 1978 Ziv, J. ; Lempel, A.: Compression of Individual Sequences via Variable-Rate Coding. In: *IEEE Transactions on Information Theory* Bd. 24(5). 1978, S. 530–536

Danksagung

Mein Dank gilt Herrn Prof. Dr. M. Griebel für die Aufgabenstellung und die Betreuung sowie Herrn Prof. Dr. M. Rumpf für die Übernahme des Koreferates. Danken möchte ich auch der Arbeitsgruppe von Herrn Prof. Dr. Griebel, insbesondere Herrn Dipl.-Inform. Th. Gerstner, der mir jederzeit mit Rat und Tat zur Seite stand. Meinen Eltern danke ich für die finanzielle Unterstützung während meines Studiums sowie Herrn Dipl.-Phys. S. Arnold und Herrn Dipl.-Phys. C. Kurz für wertvolle Ratschläge. Mein allerherzlichster Dank aber gilt Frau Dipl.-Phys. U. Herrmann.