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
<table>
<thead>
<tr>
<th>Kapitel</th>
<th>Titel</th>
<th>Seitenzahl</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Einleitung</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Gebietsaufteilungen</td>
<td>15</td>
</tr>
<tr>
<td>2.1</td>
<td>Gebietsaufteilungen</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>N-Bäume</td>
<td>17</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Verfeinerungen und zugehörige duale Bäume</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>Transformationen</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>Arten von Transformationen</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Fraktale Transformation</td>
<td>21</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Fourierttransformation</td>
<td>25</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Wavelettransformation</td>
<td>27</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Hierarchische Finite Elemente</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Übergang zu höheren Dimensionen</td>
<td>35</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Tensorprodukt-Ansatz</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Triangulierung</td>
<td>38</td>
</tr>
<tr>
<td>3.3</td>
<td>Vergleich der Transformationen</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>Kompressionsverfahren</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>Arten von Kompressionsverfahren</td>
<td></td>
</tr>
<tr>
<td>4.1.1</td>
<td>Thresholding</td>
<td>41</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Quantisierung</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Vergleich der Kompressionsverfahren</td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td>Fehlerbeschränkung</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>A priori Fehlerbeschränkung</td>
<td></td>
</tr>
<tr>
<td>5.1.1</td>
<td>Orts- und Level-abhängiges Koeffizientenabschneiden</td>
<td>48</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Koeffizientenabschneiden bei endlicher Leveltiefe</td>
<td>48</td>
</tr>
<tr>
<td>5.2</td>
<td>Übergang zu 2-D mit Tensorprodukt-Ansatz</td>
<td>48</td>
</tr>
<tr>
<td>5.2.1</td>
<td>A priori Fehlerbeschränkung in 2-D</td>
<td>49</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Adaptive Fehlerbeschränkung</td>
<td>49</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Greedy</td>
<td>50</td>
</tr>
<tr>
<td>5.3</td>
<td>Übergang zu 2-D mit Triangulierung</td>
<td>50</td>
</tr>
<tr>
<td>5.3.1</td>
<td>1-level-look-ahead</td>
<td>51</td>
</tr>
<tr>
<td>5.3.2</td>
<td>n-level-look-ahead</td>
<td>53</td>
</tr>
<tr>
<td>5.4</td>
<td>Vergleich der Fehlerbeschränkungs-Verfahren</td>
<td>55</td>
</tr>
</tbody>
</table>
6 Struktur-Kodierungsverfahren 57
   6.1 Struktur durch Null-Setzen der Koeffizienten . . . . . . . . . . 57
   6.2 Baumspeicherung . . . . . . . . . . . . . . . . . . . . . . . 58
      6.2.1 $4m$-Verfahren . . . . . . . . . . . . . . . . . . . . . . . 58
      6.2.2 $3m$-Verfahren . . . . . . . . . . . . . . . . . . . . . . . 59
      6.2.3 $2m$-Verfahren . . . . . . . . . . . . . . . . . . . . . . . 59
      6.2.4 Doppelte Knoten . . . . . . . . . . . . . . . . . . . . . . 62
   6.3 Embedded Zerotree for Wavelets (EZW) . . . . . . . . . . . . 62
   6.4 Set Partitioning in Hierarchical Trees (SPIHT) . . . . . . . . . 64
   6.5 Vergleich der Struktur-Kodierungsverfahren . . . . . . . . . . 66

7 Daten-Kodierungsverfahren 69
   7.1 Arten von Kodierungsverfahren . . . . . . . . . . . . . . . . . 69
      7.1.1 Explizite Speicherung . . . . . . . . . . . . . . . . . . . 69
      7.1.2 Run Length Encoding (RLE) . . . . . . . . . . . . . . . 70
      7.1.3 Lempel-Ziv-Welsh (LZW) . . . . . . . . . . . . . . . . . 71
      7.1.4 Huffman-Kodierung . . . . . . . . . . . . . . . . . . . . . 74
      7.1.5 Arithmetische Kodierung . . . . . . . . . . . . . . . . . . 75
      7.1.6 Dynamisches mehrstufiges Wörterbuch . . . . . . . . . . 79
      7.1.7 Burrows-Wheeler Transformation (BWT) . . . . . . . . . 80
   7.2 Vergleich der Kodierungsverfahren . . . . . . . . . . . . . . . 82

8 Erweiterungen 85
   8.1 Rechteckige Gebiete . . . . . . . . . . . . . . . . . . . . . . . 85
   8.2 Beliebige Gebiete . . . . . . . . . . . . . . . . . . . . . . . . 86
   8.3 Fokussierung . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
   8.4 Farbe . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88
      8.4.1 RGB-Farbmodell . . . . . . . . . . . . . . . . . . . . . . . 88
      8.4.2 YCbCr-Farbmodell . . . . . . . . . . . . . . . . . . . . . 88

9 Ergebnisse 91
   9.1 Tensorprodukt-Ansatz . . . . . . . . . . . . . . . . . . . . . . 91
   9.2 Hierarchische Triangulierung . . . . . . . . . . . . . . . . . . 96
      9.2.1 Raumfüllende Kurven . . . . . . . . . . . . . . . . . . . . 96
      9.2.2 Look-Ahead . . . . . . . . . . . . . . . . . . . . . . . . . 99
      9.2.3 Farbraume . . . . . . . . . . . . . . . . . . . . . . . . . . 99
      9.2.4 Histogramme . . . . . . . . . . . . . . . . . . . . . . . . . 101
      9.2.5 Fokussierung . . . . . . . . . . . . . . . . . . . . . . . . . 101
      9.2.6 Kodierung . . . . . . . . . . . . . . . . . . . . . . . . . . 107
   10 Praktik . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
<table>
<thead>
<tr>
<th>Kapitel</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Zusammenfassung und Ausblick</td>
<td>119</td>
</tr>
<tr>
<td>A Anhang</td>
<td>121</td>
</tr>
<tr>
<td>A.1 Test-Bilder</td>
<td>121</td>
</tr>
<tr>
<td>A.2 Kompressionstabellen</td>
<td>125</td>
</tr>
<tr>
<td>A.3 Symbolverzeichnis</td>
<td>130</td>
</tr>
<tr>
<td>A.3.1 Variablen und Konstanten</td>
<td>130</td>
</tr>
<tr>
<td>A.3.2 Abkürzungen</td>
<td>131</td>
</tr>
</tbody>
</table>
1 Einleitung


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.


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-

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.

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


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 erreichen 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\times300.

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:

\[
\begin{array}{c|c|c|c}
\text{Ausgangsdaten} & T & \text{transformierte Daten} & K & \text{relevanten Daten} & C & \text{nicht redundante Daten} \\
\hline
\end{array}
\]

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 \( f \) die zu komprimierenden Daten enthält und \( T \), \( K \) und \( C \) die oben aufgeführten Operatoren

\[
\begin{array}{l|c|c}
\text{redundant} & \text{relevant} \\
\hline
\text{keine Kompression} & \text{ja} & \text{ja} \\
\text{verlustlose Kompression} & \text{nein} & \text{ja} \\
\text{verlustbehafette Kompression} & \text{ja} & \text{nein} \\
\text{verlustbehaftete Kompression} & \text{nein} & \text{nein} \\
\end{array}
\]

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

\[ \hat{f} = CKTf, \]

wobei \( \hat{f} \) 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 irrevante Daten wegläßt. Die Pseudo-Inverse \( K^{-1}_x \) 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:

\[ \tilde{f} = T^{-1}K^{-1}C^{-1}f = T^{-1}K^{-1}C^{-1}CKTf. \]

Der Fehler der bei dieser Kompression gemacht wird ist:

\[ \varepsilon = ||\tilde{f} - f||. \]

Nun müssen nur noch geeignete Operatoren \( T, K \) und \( C \) gefunden werden. Hierbei ist zu beachten, daß die geeigneten Operatoren von den vorher ange-wandten abhängen können.


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
1 EINLEITUNG
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 Raumauftteilungen vorgestellt werden, wobei der Einfachheit halber zunächst von einem Quadrat als Grundfläche ausgegangen wird. In Kapitel wird erläutert, wie diese Einschränkung aufgehoben werden kann.

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

\[
\Omega = \bigcup_i T_i \\
T_i \cap T_j = \emptyset \text{ oder gemeinsame Kante oder Ecke für } i \neq j \\
T = \{T_1, \ldots, T_n\}.
\]

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 \( T_{ij} \) ersetzt. Für die neu eingesetzten Teilgebiete gelten ebenso die oben aufgeführten Bedingungen, wobei \( \Omega \) 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 (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 (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 (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.


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 2 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 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 \( T_0 \) bis zur endgültigen Verfeinerung \( T_n \) zurückgelegt wird. Jeder Knoten in dem entsprechenden Baum repräsentiert eine Ersetzung eines Teilgebietes \( T_i \in T_l \) durch mehrere neue Teilgebiete \( T_{i1}, \ldots, T_{ik} \). Hierbei wird aus der Zerlegung \( T_l \) die verfeinerte Aufteilung \( T_{l+1} \). 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).
2 GEBIETSAUFTeilungen
3 Transformationen


3.1 Arten von Transformationen

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


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 $F$ wird jeweils in Domänen ($domains, D_i$) und Regionen ($regions, R_i$) aufgeteilt. Die Domänen agieren dabei als Bildmaterialsponder, 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 $F = \bigcup_i R_i$. 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 $F$ in Regionen $R_i$ zu finden und für jedes der $R_i$ eine passende Domäne $D_{sd}$ 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 $R_i \in \mathcal{F}$. 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

$$\tilde{\mathcal{F}} = \lim_{l \to \infty} \bigcup_i T_{st,i} \cdot D_{sd,i},$$

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.


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).
3.1 Arten von Transformationen

Abbildung 4: Aufteilung eines Bildes in Domänen und Regionen. Für die in Abbildung 5 dargestellte Kopiermaschine besteht das Bild 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

1: \( F = \text{Irgend ein Bild} \neq \emptyset \ (F = \cup_i R_i) \)
2: \textbf{for} \( l = 0 \) to \textbf{maxit} \textbf{do}
3: \textbf{for} \( R_i \in F \) \textbf{do}
4: \( R_i = T_{st_i} \cdot D_{sd_i} \)
5: \textbf{end for}
6: \textbf{end for}

Algorithmus 2: Einfache Fraktale Kompression

1: \( O = \text{Original Bild} \ (F = \cup_i R_i) \)
2: \textbf{for} \( R_i \in F \) \textbf{do}
3: \( m = \text{maximaler Wert} \)
4: \textbf{for} \( D_j \in F \) \textbf{do}
5: \textbf{for} \( T_k \in \text{mögliche Transformationen} \) \textbf{do}
6: \( d = |R_i - T_k \cdot D_j| \)
7: \textbf{if} \( d < m \) \textbf{then}
8: \( sd_i = j, st_i = k \)
9: \textbf{end if}
10: \textbf{end for}
11: \textbf{end for}
12: \textbf{end for}
13: \( \text{Speichere alle } R_i \in R \text{ und dazugehörige } T_{st_i}, D_{sd_i} \)
3.1 Arten von Transformationen

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 $O(\log N)$ 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 $O(n^2 \log^2 n)$.

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 $\mathbf{F}$ 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 3).

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:

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

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 $O(n \log n)$. 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 variert werden, z. B. durch ein hartes Abschneiden oder ein weiches Einblenden der Basisfunktionen wie in Abbildung 8 dargestellt.
3.1 Arten von Transformationen


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


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.


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 skaliierten 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:

\[ f(x) = \sum_{l=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} d_{l,i} \psi_{l,i}(x). \]

(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.
3.1 Arten von Transformationen

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:

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

Ein anderes Beispiel für ein Wavelet ist der in Abbildung 9 rechts dargestellte Mexikanische Hut (Kumar und Foutoula-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 $f$ gegeben sind, in zwei orthogonale Räume $V_0$ und $W_0$ auf. In diesen beiden Räumen liegen die Vektoren $g$ und $h$. Es gilt:

$$V_1 = V_0 \oplus W_0$$

$$f = g + h.$$  

Im Fall des Haar-Wavelets kann $g$ als die Mittelwerte und $h$ als die Differenz zu den Mittelwerten aufgefaßt werden. Dies entspricht auch einem Hochpaß und einem Tiefpaß, der zu $h$ respektive $g$ 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: $f \in V_1$ (Original), $g \in V_0$ (Tiefpaß) und $h \in W_0$ (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 $\varphi$ gilt

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

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 $\varphi$ auch die Verfeinerungsgleichung

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

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 $\varphi$ 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.
3.1 Arten von Transformationen

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 $h$. 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:

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

Analog zu der Skalierungsfunktion $\varphi$ heißt der Vektor $b$ Maske des Wavelets $\psi$.

Falls die Skalierungsfunktion $\varphi_{j,k}$ orthogonal auf jedem Level $j$ ist, so läßt sich zeigen, daß die Einträge von $b$ aus dem Datenvektor $a$ 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 $\psi$ die Werte $b_0 = 1/\sqrt{2}$ und $b_1 = -1/\sqrt{2}$; alle anderen Werte von $b$ 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 $d_l$ dargestellt werden. Es gilt also:

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

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

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

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

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

wobei die Matrizen $M_{l,0}$ und $M_{l,1}$ aus den Masken $a$ und $b$ 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 1 und 2 kann der Vektor $v$ als

$$v = c^T\Phi_L$$

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 $O(N)$ 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 $a$ und $b$ 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 Arten von Transformationen

3.1.4 Hierarchische Finite Elemente


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 $O(N)$.

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 $\psi_{li}$ 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

$$
d_{l,i} = \left[ -\frac{1}{2}, 1 - \frac{1}{2} \right]_{x_{l,i}, h_{l,i}} \cdot f
$$

$$
d_{l,i} = -\frac{1}{2} f(x_{l,i} - h_{l,i}) + f(x_{l,i}) - \frac{1}{2} (x_{l,i} + h_{l,i})
$$

$$
d_{l,i} = f(x_{l,i}) - \frac{1}{2} (f(x_{l,i} - h_{l,i}) + f(x_{l,i} + h_{l,i})),
$$
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.
3.2 Übergang zu höheren Dimensionen


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:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & 1 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & 0 & 1 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{1}{2} & 0 & 1 & 0 & 0 \\
\frac{3}{4} & \frac{1}{2} & 0 & 0 & 1 & 0 \\
\frac{3}{4} & 0 & \frac{1}{2} & 0 & 0 & 1 \\
\frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 & 1
\end{pmatrix}
\]

\[
T^{-1} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{1}{2} & 1 & 0 & 0 & 0 & 0 & 0 \\
-\frac{1}{2} & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & -\frac{1}{2} & 0 & 1 & 0 & 0 & 0 \\
-\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 1 & 0 & 0 \\
-\frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 1 & 0 \\
0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & 1
\end{pmatrix}
\]

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 bewerkstel-

ligen. Die eindimensionale Transformation wird nacheinander in allen Rich-
tungen auf die Daten angewandt. Dieser Tensorprodukt-Ansatz ist der Nor-
malfall, wenn höhere Dimensionen bearbeitet werden sollen. Standardkom-
pressions-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 Hütenfunktionen 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 Hypotenusen 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(x₁, x₂, x₃, level)

1: if level ≤ endLevel then
2: \( x_n = \frac{x_2 + x_3}{2} \) \{Zur Numerierung der Eckpunkte siehe Abbildung 17\}
3: recTrafo(xₙ, x₁, x₂, level+1)
4: recTrafo(xₙ, x₃, x₁, 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 10000:1 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.


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.
3 TRANSFORMATIONEN
4 Kompressionsverfahren

Nachdem die Daten \( f \) durch die Transformation \( T \) verarbeitet worden sind, soll die Kompression \( K \) für die Auswahl der relevanten Informationen aus \( \hat{f} = Tf \) 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 \( K_s^{-1} \) soll möglichst sinnvole Werte für die Näherung \( \hat{f} = T^{-1}K_s^{-1}K\hat{f} \) ergeben. Denn nur so können die komprimierten Daten später wiederverwendet werden.

4.1 Arten von Kompressionsverfahren


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 \( \varepsilon \) für alle Eingabewerte \( s_i \) gleich. Für die Kompressionsmatrix \( K \) ergibt sich dann

\[
K_{ii} = \begin{cases} 
1 & : \|s_i\| \geq \varepsilon \\
0 & : \text{sonst}
\end{cases}
\]

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


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).


### 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
4.1 Arten von Kompressionsverfahren


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 wegge lassen. 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 \( \lfloor \cdot \rfloor \) - und \( \lceil \cdot \rceil \)-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
Betrachtet 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 verwendeten 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 $\tilde{f}$ zu den Ausgangsdaten $f$. 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 $||\cdot||_\infty$ ist dabei wie folgt definiert:

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

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 $K$ dargestellt werden, wobei

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

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

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

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 $\varepsilon_{\text{cut}}$ sind. In der Matrixschreibweise entspricht dies einer Diagonalmatrix $K = 0$ und $\tilde{f} = 0$. Der Fehler $e$ läßt sich dann direkt aus der worst case-Funktion $f$ ableiten und beträgt

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

Die worst case-Funktion $f$ ist die Rücktransformation von $\hat{f}$, wobei $\hat{f}$ aus $\varepsilon_{\text{cut}}$ 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 $\varepsilon_{\text{cut}}$ vom Level $l$ und eventuell auch von der Position $i$ abhängig gemacht. $\varepsilon_{\text{cut}}$ wird damit zu einer zweiparametrischen 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 $\varepsilon$ global nicht zu überschreiten.

Wenn $\varepsilon_{\text{cut}}$ 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 $\varepsilon$ 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 Levels dargestellt.

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

\[ \varepsilon_{\text{cut}}(l, i) = c \cdot \xi^{l+1}, \quad \frac{1}{2} \leq \xi < 1, \quad (8) \]

gefunden mit

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

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

\[
\begin{align*}
a_{-1} & = a_{-2} = 0 \\
\quad a_l & = \frac{a_{l-1} + a_{l-2}}{2} + \xi^{l+1}.
\end{align*}
\]

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

\[
\begin{align*}
a_L & = \sum_{l=1}^{L} \left[ \sum_{n=0}^{L-l} \left( \frac{1}{2} \right)^n \right] \cdot \xi^l \\
& = \sum_{l=1}^{L} \left( \frac{2}{3} + \frac{1}{3} \cdot (-2)^{L-l} \right) \cdot \xi^l \\
\Rightarrow \lim_{L \to \infty} a_L & = \frac{2}{3} \cdot \frac{\xi}{1 - \xi}.
\end{align*}
\]
5.1 A priori Fehlerbeschränkung

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 $\frac{1}{2} \leq \xi < 1$, das an den Stellen $\frac{1}{3}$ und $\frac{2}{3}$ erreicht wird, entspricht. Das Maximum der worst case-Funktion für $0 \leq \xi \leq \frac{1}{4}$ hingegen liegt bei $\frac{1}{2}$ und ist damit ganz einfach der erste Koeffizient auf dem Level 0. Im Falle $0 \leq \xi \leq \frac{1}{4}$ lautet also die Formel

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

Die Maxima für $\frac{1}{4} < \xi < \frac{1}{2}$ wandern relativ chaotisch von $\frac{1}{2}$ nach $\{\frac{1}{3}, \frac{2}{3}\}$. Die Wahl des Parameters $\xi$ 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 $\xi$ gewählt werden.

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

Abbildung 23: Level-abhängige und Orts-unabhängige worst case-Funktion für $\xi = \frac{1}{2}$ (links) und Orts- und Level-abhängige worst case-Funktion $\varepsilon_{\text{cut}}(l, i), \xi = \frac{1}{2}$ (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 \( \varepsilon^l \cdot \frac{3}{8} \) gesetzt. Es gilt also für \( \frac{1}{2} \leq \xi < 1 \):

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

Die dadurch erhaltene \textit{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 \textit{worst case}-Funktion liegt ebenfalls bei \( \{ \frac{1}{2}, \frac{2}{3} \} \) für \( \frac{1}{2} \leq \xi < 1 \) und beträgt \( \varepsilon \).

5.1.2 Koeffizientenabschneiden bei endlicher Levetiefe

Die oben angestellten Betrachtungen gelten für \( l \to \infty \), wohingegen in der Praxis meist mit endlicher Levetiefe gearbeitet wird. Es ist also zu überprüfen, ob bei bekannter Levetiefe die \( \varepsilon_{\text{cut}} \) 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 Levetiefe 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 Levetiefte ist dies eine notwendige Eigenschaft. Für endliche Levetiefte können die \( \varepsilon_{\text{cut}}(l, i) \) mit größer werdender Levetiefe 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
5.2 Übergang zu 2-D mit Tensorprodukt-Ansatz

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 \( \frac{2}{3} \cdot \frac{1}{(1-\xi)} \) zu \( \left( \frac{2}{3} \cdot \frac{1}{(1-\xi)} \right)^2 \) wird und der Level sowohl in \( x \)- als auch in \( y \)-Richtung beachtet werden muß. Es ergibt sich also für \( \frac{1}{2} \leq \xi < 1 \):

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

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 \( \alpha \) 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


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

5.3 Übergang zu 2-D mit Triangulierung

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.


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.


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


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:

$$k_v = |c_v| + \frac{2^{L-l}-1}{2^{L-l}} \cdot \max \{|k_{s_l}|, |k_{s_r}|\}$$
5.3 Übergang zu 2-D mit Triangulierung

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 \( w_2 := f(x_2) \):

\[
\begin{align*}
  k(x_1, x_2, w_1, w_2) &= \max \left\{ f\left( \frac{x_1 + x_2}{2} \right) - \frac{w_1 + w_2}{2}, \right. \\
  & \quad \left. k\left( \frac{x_1 + x_2}{2}, w_1, \frac{w_1 + w_2}{2} \right), \right. \\
  & \quad \left. k\left( \frac{x_1 + x_2}{2}, x_2, \frac{w_1 + w_2}{2}, w_2 \right) \right\}.
\end{align*}
\]

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 58 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:

\[
k(x_1, x_2, x_3, w_1, w_2, w_3) = \max \left\{ \left| f \left( \frac{x_2 + x_3}{2} - \frac{w_2 + w_3}{2} \right) \right|, \quad k \left( \frac{x_2 + x_3}{2}, x_3, x_1, \frac{w_2 + w_3}{2}, w_3, w_1 \right), \quad k \left( \frac{x_2 + x_3}{2}, x_1, x_2, \frac{w_2 + w_3}{2}, w_1, w_2 \right) \right\}.
\]

Bei der 2-D-Variante ist zu beachten, daß der Punkt \( \frac{1}{2}(x_2 + x_3) \) 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.

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 Glättigkeit 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.


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.
5  FEHLERBESCHRÄNKUNG
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 \( \varepsilon \) sind, als Nullen abzuspeichern (siehe Abbildung 27). Der nachgeschaltete Daten-Kodierer kann dann diese Nullen speicherplatzsparend ablegen.

Für kleines \( \varepsilon \) ist diese Art der Struktur-Kodierung nicht nur einfach zu implementieren, auch der Speicherplatzbedarf ist gering.

6.2 Baumspeicherung


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

11010011101001110000.

Abbildung 28: Adaptive Triangulierung und entsprechender Binärbaum.
6.2 Baumspeicherung

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).


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 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 gehörende Binärbbaum vom grössten 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:

$$2 \ 100101011.$$


Abbildung 29: Numerierung der Dreiecke und dazugehörige raumfüllende Kurve für das $2m$-Verfahren.
Algorithmus 4: Rekursionsdurchlauf für das 2m-Verfahren

1: Aufruf durch \texttt{rekDurchlauf}(x_1, x_2, x_3, level)
2: \textbf{if} level = endlevel \textbf{then}
3: \hspace{1em} lese Änderungsbit ein
4: \hspace{1em} \textbf{if} Änderungsbit = 1 \textbf{then}
5: \hspace{2em} \textbf{if} Maschine steht auf Doppelkringel \textbf{then}
6: \hspace{3em} endlevel = endlevel - 1
7: \hspace{2em} \textbf{else}
8: \hspace{3em} endlevel = endlevel + 1
9: \hspace{1em} \textbf{end if}
10: \textbf{end if}
11: \textbf{Ende der Funktion}
12: \textbf{if} level gerade \textbf{then}
13: \hspace{1em} \texttt{rekDurchlauf}(x_n, x_1, x_2, level + 1) \{0\}
14: \hspace{1em} \texttt{rekDurchlauf}(x_n, x_3, x_1, level + 1) \{1\}
15: \textbf{else}
16: \hspace{1em} \texttt{rekDurchlauf}(x_n, x_3, x_1, level + 1) \{0\}
17: \hspace{1em} \texttt{rekDurchlauf}(x_n, x_1, x_2, level + 1) \{1\}
18: \textbf{end if}

Abb. 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 vergrößert wird, wohingegen der Level bei einem Einfachkringel verfeinert wird (Gerstner 2000).
6.2.4 Doppelte Knoten


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

6.3 Embedded Zerotree for Wavelets (EZW)

6.3 Embedded Zerotree for Wavelets (EZW)

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.


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

$$T_0 = 2^{\lfloor \log c_{max} \rfloor}$$

mit

$$c_{max} = \max_{i=0,...,N} \{|c_i|\}, \quad N \text{ Anzahl der Koeffizienten}$$
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)


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 $(i, j)$.
6.4 Set Partitioning in Hierarchical Trees (SPIHT)

$\mathcal{D}(i,j)$: 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 $2 \times 2$-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 insignifikan- ten 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 $2^{\lfloor \log c_{\text{max}} \rfloor}$ gesetzt. Zusätzlich wird dem Quantisierungswert $n$ der Wert $\lfloor \log c_{\text{max}} \rfloor$ 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 größten 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 $O(i,j)$ 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 $L(i,j)$ an LIS angereiht und $D(i,j)$ 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 $(k,l) \in O(i,j)$ als $D(i,j)$ – also Typ A – an LIS angefügt und $L(i,j)$ 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.


### 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-
6.5 Vergleich der Struktur-Kodierungsverfahren


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.


<table>
<thead>
<tr>
<th>Verfahren</th>
<th>kodierte Strukturgröße (Bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4m</td>
<td>(n \cdot 1 = n)</td>
</tr>
<tr>
<td>3m</td>
<td>(n+1\cdot \frac{1}{2} = \frac{3}{4}(n + 1))</td>
</tr>
<tr>
<td>2m</td>
<td>(n+1\cdot \frac{1}{2} = \frac{n+1}{2})</td>
</tr>
</tbody>
</table>

Tabelle 2: Vergleich der Struktur-Kodierungsverfahren.
6 STRUKTUR-KODIERUNGSVERFAHREN
7 Daten-Kodierungsverfahren


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


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

**Beispiel:**
Zu komprimierende Daten: 

\[ aaaaaabaacaa \]

Kodierung:

<table>
<thead>
<tr>
<th>Position</th>
<th>Wert</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>( b )</td>
</tr>
<tr>
<td>11</td>
<td>( c )</td>
</tr>
</tbody>
</table>

7.1.2 Run Length Encoding (RLE)

7.1 Arten von Kodierungsverfahren


**Beispiel:**
Zu kodierende Daten: 

\[
\text{aaaaaabaaca}
\]

Kodierung:

ohne ESCAPE-Zeichen: \[6a1b3a1c1a\]
mit ESCAPE-Zeichen: \[Esc6aE6bEsc3aca\]

---

Algorithmus 5: Run Length Encoding (RLE)

1: altesZeichen=''
2: \[z=0\]
3: \[\text{while } \neg \text{EOF do} \]
4: \[\text{neuesZeichen}=\text{leseZeichen()}\]
5: \[\text{if } \text{neuesZeichen}=\text{altesZeichen} \text{ then}\]
6: \[z=z+1\]
7: \[\text{else}\]
8: \[\text{gibAus}(z)\]
9: \[\text{gibAus(} \text{altesZeichen)}\]
10: \[z=1\]
11: \[\text{altesZeichen}=\text{neuesZeichen}\]
12: \[\text{end if}\]
13: \[\text{end while}\]

---

7.1.3 Lempel-Ziv-Welsh (LZW)


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.


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.

Beispiel:
Wörterbuch zu Anfang:

<table>
<thead>
<tr>
<th>Index</th>
<th>Wert</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
</tbody>
</table>

Zu komprimierende Daten: \(aabaabbaa\)

Ein- und Ausgabe des Algorithmus:

<table>
<thead>
<tr>
<th>lese</th>
<th>akt. Wort</th>
<th>anh.</th>
<th>schreibe</th>
<th>akt. Wörterbuch</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>–</td>
<td>a</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>1</td>
<td>3 (\rightarrow) aa</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>b</td>
<td>1</td>
<td>4 (\rightarrow) ab</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>2</td>
<td>5 (\rightarrow) ba</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>b</td>
<td>aa</td>
<td>b</td>
<td>3</td>
<td>6 (\rightarrow) aab</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>2</td>
<td>7 (\rightarrow) bb</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>a</td>
<td>ba</td>
<td>a</td>
<td>5</td>
<td>8 (\rightarrow) baa</td>
</tr>
<tr>
<td>–</td>
<td>a</td>
<td>–</td>
<td>1</td>
<td>–</td>
</tr>
</tbody>
</table>

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


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


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.


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


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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Bau des Baumes:

<table>
<thead>
<tr>
<th>Schritt</th>
<th>Felder mit Teilbäumen</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A:0.1 ( \Sigma \text{a})</td>
</tr>
<tr>
<td>1</td>
<td>A:0.1 ( \Sigma \text{a})</td>
</tr>
<tr>
<td>2</td>
<td>A:0.1 ( \Sigma \text{a})</td>
</tr>
</tbody>
</table>

Wörterbuch:

<table>
<thead>
<tr>
<th>Wert</th>
<th>Bitcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>00</td>
</tr>
<tr>
<td>c</td>
<td>01</td>
</tr>
</tbody>
</table>

Sequenz:

<table>
<thead>
<tr>
<th>Eingabe</th>
<th>Ausgabe</th>
</tr>
</thead>
<tbody>
<tr>
<td>aabac</td>
<td>1100101</td>
</tr>
</tbody>
</table>

Abbildung 35: Beispiel einer Huffman-Kodierung mit vorgegebenen Statistiken.
Algorithmus 7: Huffman

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

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 Arten von Kodierungsverfahren

7.1.6 Dynamisches mehrstufiges Wörterbuch


7.1.7 Burrows-Wheeler Transformation (BWT)


\[ S_0 = \begin{bmatrix} B & L & U & B & B \\ S_1 = \begin{bmatrix} L & U & B & B & B \\ S_2 = \begin{bmatrix} U & B & B & B & L \\ S_3 = \begin{bmatrix} B & B & B & L & U \\ S_4 = \begin{bmatrix} B & B & L & U & B \\ \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \]

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.

7.1 Arten von Kodierungsverfahren

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>S4</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>S0</td>
<td>B</td>
<td>L</td>
</tr>
<tr>
<td>S1</td>
<td>L</td>
<td>U</td>
</tr>
<tr>
<td>S2</td>
<td>U</td>
<td>B</td>
</tr>
</tbody>
</table>

$i = 2 \Rightarrow$

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).

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>L</th>
<th>$T[i]$</th>
<th>$T[i]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S7</td>
<td>B</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

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 $T[i]$ gesetzt wird

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.


\[\text{Daß der Transformationsvektor } T \text{ auf diese Weise erstellt werden kann, hängt mit der Sortierung der Ausgangsmatrix zusammen.}\]

Eine so erhaltene Verteilung der Indizes ist dann ideal mit Entropiekodierern 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 $500 \times 500$ 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 $\varepsilon$ der Anteil an Nullen im Datenstrom stark ansteigen. Wenn hingegen die anderen Baum-Kodierer eingesetzt werden, so wird bei zunehmendem Schwellenwert $\varepsilon$ 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 enthält und die Transformation $T$ versagt hat oder daß der vorgegebene Schwellenwert $\varepsilon$ so hoch eingestellt wurde, daß keine Informationen übrig sind.
7.2 Vergleich der Kodierungsverfahren

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


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)).


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.
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.


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 Blocks 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.


\section*{8.3 Fokussierung}


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.

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.


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 reflektier- ten 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 \( \text{rot} \), \( \text{grün} \) und \( \text{blau} \) (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 Wertebereiche der einzelnen Komponenten sind dabei wie folgt:

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

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 verwendet, 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.


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:

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
= \begin{bmatrix}
1,000 & 0,000 & 1,402 \\
1,000 & -0,344 & -0,714 \\
1,000 & 1,772 & 0,000
\end{bmatrix} \cdot \begin{bmatrix}
Y \\
C_b \\
C_r
\end{bmatrix}
\]

Der Fehler soll im RGB-Farbraum in jeder einzelnen Farbkomponente auf \( \varepsilon \) beschränkt sein, also ergibt sich ein Gleichungssystem 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 Fehlerschanke 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.


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 $\varepsilon$ 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.


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.
<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>Erft Kompr. (%)</th>
<th>Fehler</th>
<th>Lena Kompr. (%)</th>
<th>Fehler</th>
<th>X-Wing Kompr. (%)</th>
<th>Fehler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>26,50</td>
<td>0,00</td>
<td>3,26</td>
<td>0,00</td>
<td>72,04</td>
<td>0,00</td>
</tr>
<tr>
<td>2</td>
<td>26,50</td>
<td>0,25</td>
<td>3,26</td>
<td>0,00</td>
<td>72,04</td>
<td>0,00</td>
</tr>
<tr>
<td>4</td>
<td>26,50</td>
<td>1,88</td>
<td>3,26</td>
<td>1,00</td>
<td>72,04</td>
<td>0,00</td>
</tr>
<tr>
<td>8</td>
<td>26,50</td>
<td>11,32</td>
<td>3,26</td>
<td>2,38</td>
<td>72,04</td>
<td>2,00</td>
</tr>
<tr>
<td>16</td>
<td>26,50</td>
<td>23,33</td>
<td>3,26</td>
<td>9,00</td>
<td>72,04</td>
<td>2,00</td>
</tr>
</tbody>
</table>

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

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

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

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

Tabelle 5: Kompression und Fehler für einfach adaptives Koeffizientenabschneiden (top down).
### 9.1 Tensorprodukt-Ansatz

<table>
<thead>
<tr>
<th>Erft</th>
<th>Lena</th>
<th>X-Wing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>Kompr. (%)</td>
<td>Fehler</td>
</tr>
<tr>
<td>1</td>
<td>42,39</td>
<td>1,00</td>
</tr>
<tr>
<td>2</td>
<td>50,05</td>
<td>2,00</td>
</tr>
<tr>
<td>4</td>
<td>58,66</td>
<td>1,95</td>
</tr>
<tr>
<td>8</td>
<td>60,53</td>
<td>2,75</td>
</tr>
<tr>
<td>16</td>
<td>62,42</td>
<td>3,60</td>
</tr>
<tr>
<td>32</td>
<td>80,93</td>
<td>4,29</td>
</tr>
<tr>
<td>64</td>
<td>85,98</td>
<td>13,10</td>
</tr>
</tbody>
</table>

Tabelle 6: Kompression und Fehler für multi adaptives Koeffizientenabschneiden.

<table>
<thead>
<tr>
<th>Erft</th>
<th>Lena</th>
<th>X-Wing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>Kompr. (%)</td>
<td>Fehler</td>
</tr>
<tr>
<td>1</td>
<td>26,50</td>
<td>0,00</td>
</tr>
<tr>
<td>2</td>
<td>40,95</td>
<td>0,50</td>
</tr>
<tr>
<td>4</td>
<td>58,65</td>
<td>1,50</td>
</tr>
<tr>
<td>8</td>
<td>68,17</td>
<td>2,56</td>
</tr>
<tr>
<td>16</td>
<td>81,21</td>
<td>3,90</td>
</tr>
<tr>
<td>32</td>
<td>91,28</td>
<td>5,86</td>
</tr>
<tr>
<td>64</td>
<td>95,34</td>
<td>9,05</td>
</tr>
</tbody>
</table>

Tabelle 7: Kompression und Fehler für bottom up Koeffizientenabschneiden.

<table>
<thead>
<tr>
<th>Erft</th>
<th>Lena</th>
<th>X-Wing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>Kompr. (%)</td>
<td>Fehler</td>
</tr>
<tr>
<td>1</td>
<td>72,75</td>
<td>1,00</td>
</tr>
<tr>
<td>2</td>
<td>86,17</td>
<td>2,00</td>
</tr>
<tr>
<td>4</td>
<td>93,29</td>
<td>4,00</td>
</tr>
<tr>
<td>8</td>
<td>97,18</td>
<td>8,00</td>
</tr>
<tr>
<td>16</td>
<td>98,51</td>
<td>15,94</td>
</tr>
<tr>
<td>32</td>
<td>99,07</td>
<td>30,40</td>
</tr>
<tr>
<td>64</td>
<td>99,08</td>
<td>33,16</td>
</tr>
</tbody>
</table>

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.


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 $\varepsilon$ 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.
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 Hierarchische Triangulierung

9.2.2 Look-Ahead


9.2.3 Farbräume

Die Farbbilder in den Bildern 47 und 48 wurden in den $Y C_b C_r$-Farbraum transformiert, bevor die Farbkomponenten dann einzeln wie Graustufenbilder komprimiert wurden.

In Bild 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 größere Triangulierung zulassen.

9.2 Hierarchische Triangulierung


9.2.4 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-
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 49: Histogramme der Koeffizienten der Bilder Lena (links) und Erft (rechts) mit Fehlerschranken $\varepsilon$ 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 $\varepsilon$ von 1, 2, 4, 8, 16, 32 und 64 (von oben nach unten).
9.2 Hierarchische Triangulierung


9.2.6 Kodierung

Um die Kodierungsverfahren zu vergleichen, wurden 14 Bilder ausgesucht (siehe Abbildungen [56] in Abschnitt [A.1]) und jeweils unterschiedlich komprimiert. Bei den Struktur-Kodierern sind nur die $2^m$-Verfahren und das Null-Setzen ausgewählt worden. Die $2^m$-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.


---

2Das Programm `gzip` kann die Daten auch mit einem Huffman-Verfahren kombiniert speichern.
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 59–61 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 61 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 \( \varepsilon \) 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 \( \varepsilon \) grundsätzlich ungeeigneter als die BWT.

Daß die Kompressionsraten in Abbildung 54 für die durch Null-Setzen erhaltenen Daten größer ist als für die \( 2^m \)-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 \( \varepsilon \) kann das \( 2^m \)-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 \( \varepsilon \) wird, um so besser wird das \( 2^m \)-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.
<table>
<thead>
<tr>
<th>Datei</th>
<th>k</th>
<th>$\varepsilon = 1$</th>
<th>$\varepsilon = 2$</th>
<th>$\varepsilon = 4$</th>
<th>$\varepsilon = 8$</th>
<th>$\varepsilon = 16$</th>
<th>$\varepsilon = 32$</th>
<th>$\varepsilon = 64$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belle h</td>
<td>18,76</td>
<td>23,66</td>
<td>32,79</td>
<td>54,10</td>
<td>83,94</td>
<td>94,21</td>
<td>*98,29</td>
<td></td>
</tr>
<tr>
<td>Belle b</td>
<td>44,65</td>
<td>47,73</td>
<td>53,29</td>
<td>*66,71</td>
<td>*86,33</td>
<td>*94,49</td>
<td>98,25</td>
<td></td>
</tr>
<tr>
<td>Berlin h</td>
<td>-16,40</td>
<td>-12,86</td>
<td>-6,09</td>
<td>6,43</td>
<td>27,77</td>
<td>*58,08</td>
<td>*84,47</td>
<td></td>
</tr>
<tr>
<td>Berlin b</td>
<td>-13,76</td>
<td>-10,51</td>
<td>-4,22</td>
<td>7,60</td>
<td>28,07</td>
<td>57,88</td>
<td>84,32</td>
<td></td>
</tr>
<tr>
<td>Canyon h</td>
<td>15,28</td>
<td>15,70</td>
<td>19,08</td>
<td>24,87</td>
<td>36,79</td>
<td>59,29</td>
<td>*87,32</td>
<td></td>
</tr>
<tr>
<td>Canyon b</td>
<td>19,94</td>
<td>20,32</td>
<td>23,35</td>
<td>28,54</td>
<td>39,25</td>
<td>*60,00</td>
<td>86,98</td>
<td></td>
</tr>
<tr>
<td>CR Bein h</td>
<td>58,83</td>
<td>61,26</td>
<td>65,96</td>
<td>75,24</td>
<td>90,90</td>
<td>99,34</td>
<td>*99,82</td>
<td></td>
</tr>
<tr>
<td>CR Bein b</td>
<td>68,74</td>
<td>70,44</td>
<td>73,77</td>
<td>*80,58</td>
<td>*92,53</td>
<td>*99,40</td>
<td>99,82</td>
<td></td>
</tr>
<tr>
<td>CR Thorax h</td>
<td>1,32</td>
<td>19,55</td>
<td>51,49</td>
<td>85,39</td>
<td>98,49</td>
<td>99,57</td>
<td>99,69</td>
<td></td>
</tr>
<tr>
<td>CR Thorax b</td>
<td>35,43</td>
<td>46,32</td>
<td>*66,30</td>
<td>*88,97</td>
<td>*98,70</td>
<td>*99,60</td>
<td>*99,71</td>
<td></td>
</tr>
<tr>
<td>CT Kopf h</td>
<td>59,88</td>
<td>63,76</td>
<td>69,66</td>
<td>79,85</td>
<td>87,86</td>
<td>*93,02</td>
<td>*96,25</td>
<td></td>
</tr>
<tr>
<td>CT Kopf b</td>
<td>65,14</td>
<td>68,21</td>
<td>*73,00</td>
<td>*81,47</td>
<td>*88,38</td>
<td>92,95</td>
<td>96,03</td>
<td></td>
</tr>
<tr>
<td>Erft h</td>
<td>64,73</td>
<td>90,31</td>
<td>96,81</td>
<td>98,98</td>
<td>99,72</td>
<td>*99,93</td>
<td>*99,96</td>
<td></td>
</tr>
<tr>
<td>Erft b</td>
<td>75,60</td>
<td>92,11</td>
<td>97,11</td>
<td>99,02</td>
<td>*99,73</td>
<td>99,89</td>
<td>99,91</td>
<td></td>
</tr>
<tr>
<td>Lena h</td>
<td>-6,88</td>
<td>7,15</td>
<td>32,95</td>
<td>67,08</td>
<td>85,97</td>
<td>94,40</td>
<td>*98,24</td>
<td></td>
</tr>
<tr>
<td>Lena b</td>
<td>15,37</td>
<td>24,92</td>
<td>43,42</td>
<td>*69,98</td>
<td>*86,44</td>
<td>*94,42</td>
<td>98,19</td>
<td></td>
</tr>
<tr>
<td>Mandrill h</td>
<td>-13,69</td>
<td>-9,01</td>
<td>-0,28</td>
<td>16,32</td>
<td>40,66</td>
<td>*65,03</td>
<td>*86,44</td>
<td></td>
</tr>
<tr>
<td>Mandrill b</td>
<td>-8,14</td>
<td>-4,05</td>
<td>3,56</td>
<td>18,39</td>
<td>*41,11</td>
<td>64,91</td>
<td>86,31</td>
<td></td>
</tr>
<tr>
<td>Mona Lisa h</td>
<td>7,35</td>
<td>9,86</td>
<td>20,92</td>
<td>38,51</td>
<td>63,89</td>
<td>87,93</td>
<td>*98,49</td>
<td></td>
</tr>
<tr>
<td>Mona Lisa b</td>
<td>26,77</td>
<td>28,68</td>
<td>36,42</td>
<td>*49,11</td>
<td>*68,56</td>
<td>*88,61</td>
<td>98,44</td>
<td></td>
</tr>
<tr>
<td>Parabel h</td>
<td>64,74</td>
<td>95,67</td>
<td>98,56</td>
<td>98,58</td>
<td>98,59</td>
<td>98,59</td>
<td>99,29</td>
<td></td>
</tr>
<tr>
<td>Parabel b</td>
<td>80,26</td>
<td>96,73</td>
<td>*98,67</td>
<td>98,69</td>
<td>98,70</td>
<td>98,70</td>
<td>99,30</td>
<td></td>
</tr>
<tr>
<td>Peppers h</td>
<td>-11,40</td>
<td>-0,57</td>
<td>19,60</td>
<td>53,67</td>
<td>82,71</td>
<td>92,85</td>
<td>*97,32</td>
<td></td>
</tr>
<tr>
<td>Peppers b</td>
<td>12,32</td>
<td>19,75</td>
<td>34,06</td>
<td>*59,74</td>
<td>*83,71</td>
<td>*92,99</td>
<td>97,30</td>
<td></td>
</tr>
<tr>
<td>Vancouver h</td>
<td>4,03</td>
<td>10,57</td>
<td>23,79</td>
<td>47,27</td>
<td>75,03</td>
<td>91,22</td>
<td>*97,85</td>
<td></td>
</tr>
<tr>
<td>Vancouver b</td>
<td>17,75</td>
<td>22,81</td>
<td>33,10</td>
<td>*52,31</td>
<td>*76,41</td>
<td>*91,36</td>
<td>97,81</td>
<td></td>
</tr>
<tr>
<td>X-Wing h</td>
<td>76,30</td>
<td>76,86</td>
<td>78,84</td>
<td>82,62</td>
<td>86,67</td>
<td>90,33</td>
<td>94,29</td>
<td></td>
</tr>
<tr>
<td>X-Wing b</td>
<td>80,76</td>
<td>81,15</td>
<td>*82,45</td>
<td>*85,10</td>
<td>*88,12</td>
<td>*91,17</td>
<td>*94,74</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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 $\varepsilon$-Werte kennzeichnen die Schwellenwerte für das Abschneidekriterium. Die Sterne markieren das beste Verfahren. (Werte kleiner Null stellen eine Vergrößerung der Ausgangsdatei dar)

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

9.3 Fraktale


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 $0^\circ$, $90^\circ$, $180^\circ$, $270^\circ$, eine horizontale Spiegelung sowie eine Helligkeits- und Kontrastanpassung zur Verfügung.


In Abbildung 55 rechts oben ist gut zu sehen, wie die fraktale Transformation sich den lokalen Eigenschaften des Ausgangsbildes anpaßt.
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 $\varepsilon$ gelegt wurde. Die Kompression bestand dabei aus drei Komponenten: der Transformation $T$, der Kompression $K$ und der Kodierung $C$.


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.


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 $2^m$ (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-

Als Erweiterungen wurden für die hierarchische Triangulierung die Farbbild-Kodierung im $RGB$- und $YC_{b}C_{r}$-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 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).


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).


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).


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 2\(m\) mit Werten und Huffman, 2\(m\) mit Werten und BWT, Null-Setzen mit Werten und Huffman, Null-Setzen mit Werten und BWT, 2\(m\) mit Koeffizienten und Huffman, 2\(m\) 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 \(\varepsilon\). Daher wurden diese Werte nur im Bild für eine Fehlerschwelle \(\varepsilon\) 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 $\varepsilon$ 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 $\epsilon$ 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 $\epsilon$ konstant, wurden aber zur besseren Visualisierung der anderen Werte in den unteren Bildern abgeschnitten.
### A.2 Kompressionstabellen

<table>
<thead>
<tr>
<th>Dateigröße [KByte]</th>
<th>$\varepsilon$ = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dateigröße [KByte]</th>
<th>$\varepsilon$ = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dateigröße [KByte]</th>
<th>$\varepsilon$ = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dateigröße [KByte]</th>
<th>$\varepsilon$ = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dateigröße [KByte]</th>
<th>$\varepsilon$ = 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dateigröße [KByte]</th>
<th>$\varepsilon$ = 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dateigröße [KByte]</th>
<th>$\varepsilon$ = 64</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

Abbildung 62: Dateigrößen für verschiedene Bilder bei Fehlerschwellen $\varepsilon$ 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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Bedeutung</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
<tr>
<td>$.$</td>
<td>Nächstgrößere ganze Zahl</td>
</tr>
<tr>
<td>$.$</td>
<td>Nächstkleinere ganze Zahl</td>
</tr>
<tr>
<td>$a, b$</td>
<td>Masken für Waveletes</td>
</tr>
<tr>
<td>$C, C^{-1}$</td>
<td>Kodierung, Inverse der Kodierung</td>
</tr>
<tr>
<td>$C_d$</td>
<td>Daten-Kodierung</td>
</tr>
<tr>
<td>$C_s$</td>
<td>Struktur-Kodierung</td>
</tr>
<tr>
<td>$D_i$</td>
<td>Domänen</td>
</tr>
<tr>
<td>$D(i, j)$</td>
<td>Alle Nachfahren bei SPIHT</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>Fehlerwert</td>
</tr>
<tr>
<td>$f$</td>
<td>Datenvektor</td>
</tr>
<tr>
<td>$\hat{f}$</td>
<td>Transformierter Datenvektor</td>
</tr>
<tr>
<td>$\tilde{f}$</td>
<td>Verlustbehaftet rücktransformierter Datenvektor</td>
</tr>
<tr>
<td>$F$</td>
<td>Bild</td>
</tr>
<tr>
<td>$I$</td>
<td>Identität</td>
</tr>
<tr>
<td>$K, K^{-1}$</td>
<td>Kompression, Pseudo-Inverse der Kompression</td>
</tr>
<tr>
<td>$L_2(\mathbb{R})$</td>
<td>Raum der quadratintegrierbaren Funktionen über $\mathbb{R}$</td>
</tr>
<tr>
<td>$L(i, j)$</td>
<td>Nachfahren ohne Söhne bei SPIHT</td>
</tr>
<tr>
<td>$N$</td>
<td>Menge der natürlichen Zahlen</td>
</tr>
<tr>
<td>$O(N)$</td>
<td>Komplexität</td>
</tr>
<tr>
<td>$O(i, j)$</td>
<td>Direkte Söhne bei SPIHT</td>
</tr>
<tr>
<td>$\mathbb{R}$</td>
<td>Menge der reellen Zahlen</td>
</tr>
<tr>
<td>$R_i$</td>
<td>Regionen</td>
</tr>
<tr>
<td>$T, T^{-1}$</td>
<td>Transformation, Inverse der Transformation</td>
</tr>
<tr>
<td>$T_{(j)}$</td>
<td>Teilgebiete</td>
</tr>
<tr>
<td>$T$</td>
<td>Menge aller Teilgebiete</td>
</tr>
<tr>
<td>$V_l, W_l$</td>
<td>Wavelets-Räume</td>
</tr>
<tr>
<td>$\mathbb{Z}$</td>
<td>Menge der ganzen Zahlen</td>
</tr>
<tr>
<td>$\xi$</td>
<td>Glattheitsparameter</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>Skalierungsfunktion</td>
</tr>
<tr>
<td>$\psi$</td>
<td>Wavelet</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>Gebiet</td>
</tr>
</tbody>
</table>
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
$Y_C b C_r$ Farbkodierung
Literatur


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


**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 Publishers, 2000


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


Danksagung