Diplomarbeit

Entwurf von Quadtree–Videocodecs

Bearbeitung: Klaus Pieper
Betreuung: Dipl.-Ing. Thomas Wiegand

22. Juli 1997

Lehrstuhl für Nachrichtentechnik
Universität Erlangen-Nürnberg
Prof. Dr.-Ing. B. Girod

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 22. Juli 1997

Inhaltsverzeichnis

Überblick **vii**

Liste der verwendeten Formelzeichen und Abkürzungen **ix**

Abbildungsverzeichnis **xi**

1 Einleitung **1**

2 Quadtree-Videocodec **3**
2.1 Vektor-Quantisierung mit Entropie-Nebenbedingung 3
2.1.1 Die Lagrange-Kostenfunktion 4
2.1.2 Der iterative Entwurfsalgorithmus 5
2.2 Aufbau des Quadtree-Codecs 7
2.2.1 Strukturierung des Codier-Algorithmus 8
2.2.1.1 Segmentbasierte Bewegungs- und Intensitätskompensation 9
2.2.1.2 Rekonstruktion mit Segmentüberlappungen 9
2.2.2 Prädiktion der Bewegungsvektoren 10
2.2.3 Algorithmen zur Suche optimaler Baumstrukturen 11
2.2.3.1 Pruning-Algorithmus 11
2.2.3.2 Growing-Algorithmus 12
2.3 Unrestricted Motion 12

3 Bewegungskompensation mit Halb-Pel-Genauigkeit **15**
3.1 Implementierung der Halb-Pel-Bewegungskompensation 15
3.2 Untersuchungsergebnisse 16

4 Langzeitprädiktion **19**
4.1 Implementierung der Langzeitprädiktion 19
4.2 Untersuchungsergebnisse 20
4.2.1 Vergleich zwischen konventioneller Bewegungskompensation und Bewegungskompensation mit Langzeitspeicher 20
4.2.2 Vergleich zwischen Langzeitprädiktion mit separater Zeittabelle und mit in die Shapetabelle integriertem Zeitparameter 21

5 Kombination mit DCT-Codierung **23**
5.1 Implementierung der DCT-Codierung 23
5.2 Untersuchungsergebnisse 24

6 Weitere Schritte zur Optimierung des Codecs **27**
6.1 Quantisierung der Intensitätsdifferenz unter Berücksichtigung der Lagrange-Kosten . . . 27
6.1.1 Einschränkung des Suchbereichs . . . 28
6.1.2 Untersuchungsergebnisse . . . 29
6.2 Ansatz zur Beschleunigung der Bewegungsschätzung . . . 32

7 Zusammenfassung und Ausblick **33**

A Programmbeschreibung **35**
A.1 Codec . . . 35
A.1.1 Coder . . . 35
A.1.2 Decoder . . . 37
A.2 Programme zur Optimierung des Codebuchs . . . 38
A.2.1 Erzeugung des Ausgangscodebuchs . . . 39
A.2.2 Häufigkeitsverteilung . . . 40
A.2.3 Erzeugung des Codebuchs aus der Häufigkeitsverteilung . . . 41
A.2.4 Sortierung des Codebuchs . . . 42
A.3 Analysetool . . . 42

B Syntax der Codesequenz **45**
B.1 Sequenzebene . . . 45
B.2 Bildebene . . . 46
B.3 Makroblockebene . . . 46

C Simulationsbedingungen **49**

Literaturverzeichnis **52**

Überblick

In dieser Arbeit wird der Entwurf von Quadtree-Videocodecs nach verschiedenen Aspekten hin untersucht. Zunächst wird der aus der Vektorquantisierung mit Entropie-Nebenbedingung bekannte iterative Entwurfs-Algorithmus erläutert. Der Aufbau eines Quadtree-Codecs wird in kurzer Form dargelegt. Es erfolgt die Erweiterung der Bewegungskompensation auf Halbpixel-Genauigkeit. Die Implementierung der Langzeitprädiktion in den Quadtree-Codec wird untersucht. Weiterhin wird der Quadtree-Codec mit einer DCT-Codierung kombiniert. Schließlich werden Optimierungsmöglichkeiten am Codec untersucht die auf der Separierbarkeit der Lagrange-Kosten eines Bewegungsvektors von den übrigen Kosten innerhalb des Lagrange-Funktionals beruhen. Die Untersuchungsergbnisse werden in Form von PSNR-Messungen präsentiert.

Liste der verwendeten Formelzeichen und Abkürzungen

Signaldarstellungen

$\mathbf{C}$	Code
$\mathbf{I}$	Index
$\mathbf{X}$	Quelle
$\mathbf{Y}$	Senke
$\mathbf{x}$	Videosequenz im Original
$\mathbf{y}$	Rekonstruierte Videosequenz

Funktionen

$P_{\mathbf{X}}(\mathbf{x})$	Verteilungsfunktion der Zufallsvariablen $\mathbf{X}$
$p_{\mathbf{X}}(\mathbf{x})$	Wahrscheinlichkeitsfunktion der Zufallsvariablen $\mathbf{X}$
$d(\mathbf{x}, \mathbf{y})$	Quadratischer Fehler zwischen $\mathbf{x}$ und $\mathbf{y}$
α	Abbildung der Quelle auf den Index
β	Abbildung des Index auf das quantisierte Signal
γ	Abbildung des Index auf den Code

Funktionale

D	Distortion-Funktional
H	Entropie-Funktional
J	Lagrange-Funktional
R	Raten-Funktional

Operatoren

$E\{\cdot\}$	Erwartungswert-Operator

Symbole

λ	Lagrange-Parameter

Abkürzungen

CIF	Common Intermediate Format
ECVQ	Entropy Constrained Vector Quantization
IE	Intensity Estimation
MC	Motion Compensation
ME	Motion Estimation
MIC	Motion and Intensity Compensation
PSNR	Peak Signal to Noise Ratio
QCIF	Quarter Common Intermediate Format
SSE	Summed Squared Error
VBS	Variable Block Size
VLC	Variable Length Code
VQ	Vector Quantization

Abbildungsverzeichnis

2.1 *Modell eines Vektorquantisierers mit variabler Rate als Kommunikationssystem* 4

2.2 *Eine typische realisierbare Distortion-Rate-Funktion für eine diskrete Verteilung. Jeder Punkt stellt ein Rate-Distortion-Paar dar; die auf der konvexen Hülle liegenden, werden durch Minimierung des Lagrange-Funktionals gefunden.* 5

2.3 *Blockschaltbild des Quadtree-Videocodecs* 7

2.4 *Beispiel für eine Quadtree-Struktur* 8

2.5 *Sechzehn verschiedene Möglichkeiten einen Block in vier Teilblöcke zu unterteilen. Die übergeordneten Teilblöcke werden zu Segmenten zusammengefaßt (weiß), während die untergeordneten Teilblöcke (grau) weiter zerlegt werden können.* 9

2.6 *Masken zur Gewichtung eines im Inneren, am Bildrand und in der Bildecke liegenden Bildpunktes* 10

2.7 *Der Bewegungsvektor des aktuellen Makroblocks wird durch die Bewegungsvektoren der benachbarten Makroblöcke prädiziert* 10

2.8 *Testsequenz „Car Phone“ (7.5 fps, 10 s) mit und ohne Unrestricted-Motion-Modus* 13

2.9 *Testsequenz „Salesman“ (7.5 fps, 10 s) mit und ohne Unrestricted-Motion-Modus* 13

3.1 *Vergleich zwischen Half-Pel- und Full-Pel-Genauigkeit anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)* 16

3.2 *Vergleich zwischen Half-Pel- und Full-Pel-Genauigkeit anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)* 16

3.3 *Vergleich zwischen Half-Pel-Genauigkeit auf allen Leveln und nur auf Level 0 und 1 aller Komponenten und Half-Pel-Genauigkeit auf allen Leveln der Y-Komponente anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)* 17

3.4 *Vergleich zwischen Half-Pel-Genauigkeit auf allen Leveln und nur auf Level 0 und 1 aller Komponenten und Half-Pel-Genauigkeit auf allen Leveln der Y-Komponente anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)* 17

4.1 *Vergleich zwischen Langzeitspeichertiefe 1 und 10 anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)* 21

4.2 *Vergleich zwischen Langzeitspeichertiefe 1 und 10 anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)* 21

4.3 *Vergleich zwischen Langzeitprädiktion mit separater Zeittabelle und mit in die Shapetabelle integriertem Zeitparameter anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)* 22
4.4 *Vergleich zwischen Langzeitprädiktion mit separater Zeittabelle und mit in die Shapetabelle integriertem Zeitparameter anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)* 22

5.1 *Testsequenz „Car Phone“ (7.5 fps, 10 s) mit und ohne DCT-Modus* 25
5.2 *Testsequenz „Salesman“ (7.5 fps, 10 s) mit und ohne DCT-Modus* 25

6.1 *Wahrscheinlichkeitsdichtefunktionen der Intensitätsdifferenzen aus dem Codebuchentwurf für $\lambda = 100$ für Level 1, 2 und 3 (im Uhrzeigersinn) der Y-Komponente.* 29
6.2 *Vergleich zwischen fester und variabler Intensitätsdifferenzquantisierung anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)* 30
6.3 *Vergleich zwischen fester und variabler Intensitätsdifferenzquantisierung anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)* 30
6.4 *Vergleich zwischen fester und variabler Intensitätsdifferenzquantisierung anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)* 31
6.5 *Vergleich zwischen fester und variabler Intensitätsdifferenzquantisierung anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)* 31

B.1 *Struktur der Codesequenz auf Sequenzebene* 45
B.2 *Struktur der Codesequenz auf Bildebene* 46
B.3 *Struktur der Codesequenz auf Makroblockebene* 46

Kapitel 1

Einleitung

Der Mensch ist in der Lage große Mengen visueller Informationen aufzunehmen und zu verarbeiten. Es ist daher kaum verwunderlich, daß die Verbreitung visueller Informationen in den Industrienationen mehr eine Rolle spielt denn je. Neben aufwendigen und hochqualitativen Videokommunikationseinrichtungen sind vor allem auch preiswerte Lösungen gefragt, die vorhandene Datennetze mit geringer Bandbreite nutzen können. Anwendungen sind die Videotelefonie über herkömmliche analoge Netze oder die Videoübertragung über das Internet.

Standards wie der H.263-Standard oder der zukünftige MPEG-4-Standard zielen auf die Videocodierung mit niedriger Bitrate ab. Ihnen liegt ein Hybrid-Coder zugrunde, der einen DPCM-Algorithmus entlang einer Bewegungstrajektorie des Sequenzinhaltes mit einer Fehlerbildcodierung verbindet. Ein in [1] vorgestellter und in [2] weiterentwickelter Codec führt die Bewegungskompensation mit Blöcken variabler Größe aus. Hierbei wird zugunsten einer Intensitätskompensation auf die Fehlerbildcodierung verzichtet. Der Bewegungs- und Intensitätskompensation ist eine Quadtree-Struktur überlagert, deren Knoten einfache Segmentformen variabler Größe adressieren. Die Einteilung des Bildes in Segmente führt zur einer Suche nach der optimalen Baumstruktur. Dabei bildet die Segmentierung, sowie die Bewegungs- und Intensitätskompensation eine Einheit, so daß die Optimierung der Quadtree-Struktur zu einem, im Sinne der Rate-Distortion-Theorie, optimalen Ergebnis führt. Als Optimalitätskriterium dient hier die Minimierung des Lagrange-Funktionals.

Dieser Quadtree-Videocodec wird im Rahmen der vorliegenden Arbeit nach verschiedenen Aspekten hin untersucht. Von Interesse sind hierbei die Erweiterung der Bewegungskompensation auf Halbpixel-Genauigkeit, die Einbindung einer bewegungskompensierten Langzeitprädiktion und die Kombination des Codecs mit der DCT-Codierung. Ferner wird die Auswirkung einer besseren Einbindung der Intensitätskompensation in den Optimierungsprozeß untersucht.

Der Entwurf des Quadtree-Codecs stellt ein nichtlineares Optimierungsproblem dar. Gelöst wird das Problem durch einen aus der Vektorquantisierung mit Entropie-Nebenbedingung bekannten iterativen Entwurfs-Algorithmus. Zu dieser Problematik gibt Kapitel 2 eine kurze Einführung. Weiterhin wird dort der Aufbau des Quadtree-Codecs erläutert.

In Kapitel 3 wird die Erweiterung der Bewegungskompensation des Codecs auf Halbpixel-Genauigkeit untersucht.

Kapitel 4 beschäftigt sich mit der Implementierung der bewegungskompensierten Langzeitprädiktion. Die Untersuchungen beziehen sich hierbei auf die Bewegungskompensation mit Pixel- und Halbpixel-Genauigkeit.

Die Erweiterung des Quadtree-Codecs um die DCT-Codierung und deren Ergebnisse werden in Kapitel 5 dargestellt.

In Kapitel 6 werden weitere Optimierungsmöglichkeiten am Codec untersucht, die aufgrund der Separierbarkeit des quadratischen Fehlers eines bewegungskompensierten Blocks von den übrigen Lagrange-Kosten durchführbar sind.

Schließlich erfolgt in Kapitel 7 eine Zusammenfassung der am Quadtree-Codec vorgenommenen Untersuchungen. Ein Ausblick zeigt noch weitere erfolgversprechende Untersuchungsmöglichkeiten auf.

Kapitel 2

Quadtree-Videocodec

Der hier vorgestellte Codec erlaubt aufgrund seiner Quadtree-Struktur die Videocodierung mit einer Bewegungskompensation von Blöcken variabler Größe. Zunächst sieht das Konzept des Quadtree-Videocodecs nur die Bewegungs- und Intensitätskompensation von Segmenten variabler Größe vor, ohne eine weitere Prädiktionsfehlerbildcodierung anzuschließen.

Die Bewegungskompensation kann als ein Spezialfall der Vektor-Quantisierung mit Entropie-Nebenbedingung (ECVQ) angesehen werden. Dabei stellt das in der Videosequenz vorangegangene, decodierte Bild das Codebuch dar, wo hingegen die entropiecodierten translatorischen Bewegungsparameter als Indizes, die auf einen Codebucheintrag verweisen, betrachtet werden. Konkret besteht hier das Codebuch aus den einzelnen Bildblöcken, die durch die Bewegungsvektoren adressiert werden. Die Anzahl der möglichen Bewegungsvektoren ist daher gleich der Codebuchgröße eines Vektorquantisierers. Der vorliegende Quadtree-Videocodec entspricht also einer baumstrukturierten ECVQ und es ist deshalb naheliegend, den Entwurfsprozeß in den aus der ECVQ bekannten, iterativen Entwurfsalgorithmus einzubetten.

Bevor in diesem Kapitel der Aufbau des Quadtree-Codecs erläutert wird, erfolgt zunächst eine kurze Darlegung der nötigen theoretischen Grundlagen aus dem Gebiet der Vektorquantisierung mit Entropie-Nebenbedingung.

2.1 Vektor-Quantisierung mit Entropie-Nebenbedingung

Ein Vektorquantisierer faßt n Quellsymbole einer diskreten Quelle zu einem Vektor zusammen. Diesem wird aus einem Satz von m Reproduktionsvektoren, dem Codebuch, ein Codevektor zugeordnet. Werden die Indizes der Codevektoren mit Codeworten fester Länge codiert und über einen binären Kanal übertragen, so ergibt sich eine Rate von $R = (\log_2 m)/n$ bits pro Sample. Nach Shannons Qellencodierungstheorem kann bei einer Übertragungsrate R die minimale Distortion $D(R)$ beliebig nahe erreicht werden, wenn die Vektorlänge n ausreichend groß ist. Natürlich sind in der Praxis einer willkürlichen Vektorlänge Grenzen gesetzt. Bei typischen Vektorlängen kann eine Verringerung der

durchschnittlichen Übertragungsrate R erreicht werden, wenn die Index-Sequenz entropiecodiert wird [3].

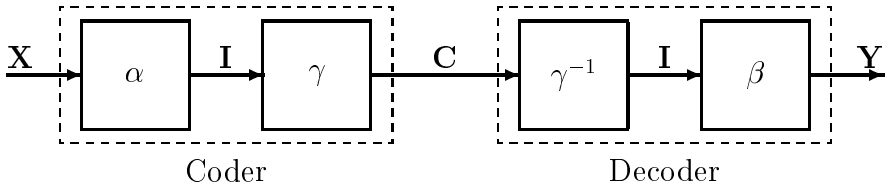

Abbildung 2.1: *Modell eines Vektorquantisierers mit variabler Rate als Kommunikationssystem*

Abb. 2.1 zeigt das Modell eines Vektorquantisierers mit variabler Rate. Der Coder besteht aus der Komposition $\gamma \circ \alpha$. Der Quellsymbolvektor $\mathbf{X}$ wird durch α auf den Index $\mathbf{I}$ abgebildet. Dieser Schritt ist im Allgemeinen mit einem Informationsverlust verbunden. Die Entropie-Codierung des Index $\mathbf{I}$ erfolgt durch die Abbildung γ. Hierbei wird dem Index $\mathbf{I}$ ohne Informationsverlust ein eindeutiger präfix-freier Code $\mathbf{C}$ zugewiesen. Der Übertragungskanal sei als verlustlos angenommen, so daß der Decoder $\beta \circ \gamma^{-1}$ das Codesymbol $\mathbf{C}$ über γ^{-1} auf den Index $\mathbf{I}$ abbildet und dieser durch β schließlich auf das quantisierte Signal $\mathbf{Y}$ abgebildet wird. Die Decodierung (Abbildung β) erfolgt mit Hilfe einfacher Tabellendecodierung der Indizes.

Das Ziel ist die optimale Quellencodierung, d.h. zu einer bestimmten mittleren Bitrate ist eine minimale mittlere Distortion zu erzielen. Die Formulierung des Optimierungsproblems und der Entwurfsalgorithmus eines optimalen Vektorquantisierers ist Thema der folgenden Abschnitte.

2.1.1 Die Lagrange-Kostenfunktion

Zunächst muß eine Festlegung getroffen werden, wie das Qualitätsmaß für die Rekonstruktion bestimmt wird. Der quadratische Fehler

$$d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||^2 = \sum_{k=0}^{n-1} |x_k - y_k|^2 \tag{2.1}$$

wird am häufigsten als Fehlerkriterium verwendet und findet auch innerhalb dieser Arbeit Anwendung. Hierbei stellt $\mathbf{x}$ den Eingangsvektor der Länge n und $\mathbf{y}$ die Rekonstruktion des Eingangsvektors dar. Für das System aus Abb. 2.1 ergibt sich daher für die mittlere Distortion

$$D(\alpha, \beta) = E\left[d(\mathbf{X}, \beta(\alpha(\mathbf{X})))\right]. \tag{2.2}$$

Durch die Operation γ des Coders ergibt sich ein Kommunikationssystem mit variabler Codewortlänge für die Eingangsvektoren $\mathbf{x}$. Für die mittlere Anzahl der Kanalsymbole in bit oder einfach mittlere Rate erhält man

$$R(\alpha, \gamma) = E\left[\gamma(\alpha(\mathbf{X}))\right]. \tag{2.3}$$

Offenbar besteht der Wunsch, die Distortion bei einer gegebenen Rate zu minimieren. Dieses Optimierungsproblem mit Nebenbedingung

$$\hat{D}(R) = \inf_{\alpha,\gamma,\beta} \{D(\alpha,\beta) | R(\alpha,\gamma) \leq R\}, \tag{2.4}$$

läßt sich durch Einführung des Lagrange-Funktionals

$$J_\lambda(\alpha,\gamma,\beta) = D(\mathbf{X}, \beta(\alpha(\mathbf{X}))) + \lambda|\gamma(\alpha(\mathbf{X}))| \tag{2.5}$$

in ein Optimierungsproblem ohne Nebenbedingung überführen. Die Lagrange-Kosten J sind bei festem Lagrange-Parameter λ zu minimieren. Jede Wahl von λ führt zu einem Distortion-Rate-Paar (D, R) auf der konvexen Hülle von $\hat{D}(R)$, die zugleich die untere Schranke der realisierbaren Rate-Distortion-Funktion darstellt. Die Lösungen die sich aus der Minimierung der Lagrange-Kostenfunktion ergeben, sind auch Lösungen des Optimierungsproblems mit Nebenbedingung (2.4). Zudem stellt λ den negativen Anstieg in den gefundenen Distortion-Rate Paaren dar [3].

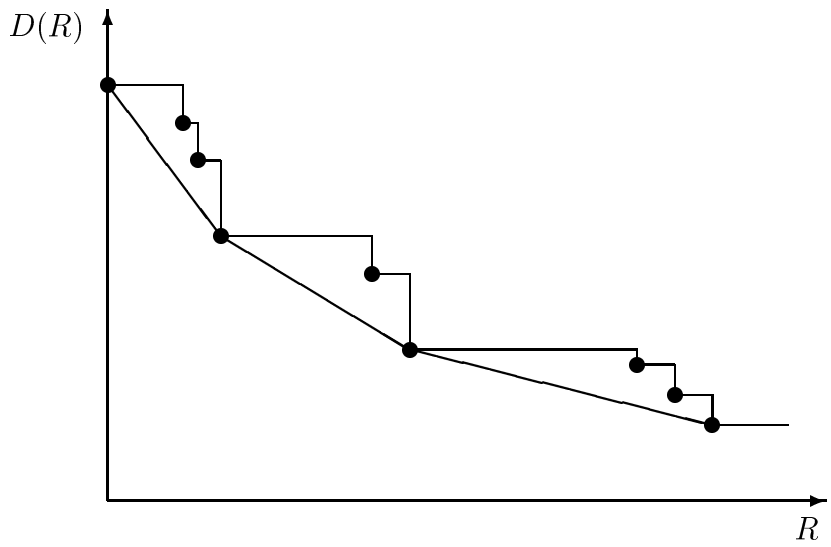

Abbildung 2.2: *Eine typische realisierbare Distortion-Rate-Funktion für eine diskrete Verteilung. Jeder Punkt stellt ein Rate-Distortion-Paar dar; die auf der konvexen Hülle liegenden, werden durch Minimierung des Lagrange-Funktionals gefunden.*

Abb. 2.2 zeigt eine typische realisierbare Distortion-Rate-Funktion für eine diskrete Verteilung. Die auf der konvexen Hülle liegenden Distortion-Rate-Paare werden durch Minimierung des Lagrange-Funktionals (2.5) gefunden. Der positive reelle Lagrange-Multiplikator λ gibt die Gewichtung zwischen R und D an. Große λ-Werte führen zu einer niedrigen Rate, wogegen kleine λ-Werte gleichbedeutend mit einer hohen Rate sind.

2.1.2 Der iterative Entwurfsalgorithmus

Es ist nun ein Codec (α, β, γ) zu finden, der das Lagrange-Funktional (2.5) minimiert. Diese Aufgabe löst der iterative Algorithmus nach Chou, Lookabaugh, und Gray [3]:

Es stehe ein initialer Coder $(\alpha^0, \gamma^0, \beta^0)$ zur Verfügung, sowie eine feste Verteilungsfunktion der Quelle $P_{\mathbf{X}}(\mathbf{x})$. Bei unbekannter Verteilung behilft man sich mit einem Satz von Trainingsvektoren. Der Lagrange-Parameter λ wird auf einen konstanten Wert festgelegt.

1. Das Reproduktionscodebuch und die korrespondierenden Codewortlängen (Abbildungen β und γ) werden konstant gehalten. Es ist eine Abbildung $\alpha : \mathbf{X} \to \mathbf{I}$ zu suchen, die das Lagrange-Funktional minimiert.

$$\alpha(\mathbf{x}) = \underset{i \in \mathbf{I}}{\operatorname{argmin}} \left[d(\mathbf{x}, \beta(i)) + \lambda |\gamma(i)| \right] \tag{2.6}$$

   Ist die Distanz zwischen $\mathbf{x}$ und $\beta(i)$ plus der mit λ gewichteten Codewortlänge $|\gamma(i)|$ minimal, so bildet α den Eingangsvektor $\mathbf{x}$ auf den Index i ab.

2. Jetzt werden α und β konstant gehalten und eine Abbildung $\gamma : \mathbf{I} \to \mathbf{C}$ gesucht, die das Lagrange-Funktional in der Form

$$J_\lambda(\alpha, \gamma, \beta) = \sum_{i \in \mathbf{I}} p(i) E \left[d(\mathbf{X}, \beta(i)) + \lambda |\gamma(i)| \middle| \alpha(\mathbf{X}) = i \right] \tag{2.7}$$

   minimiert. Hierbei steht

$$p(i) = P\{\alpha(\mathbf{X}) = i\} \tag{2.8}$$

   für die Wahrscheinlichkeit der Verwendung des Repräsentanten i. Optimal wird das Problem durch

$$|\gamma(i)| = -\log_2 p(i) \tag{2.9}$$

   gelöst. Die mittlere Codewortlänge entspricht genau der Indexentropie

$$H(p) = -\sum_{i \in \mathbf{I}} p(i) \log_2 p(i) \tag{2.10}$$

   des erzeugten Codes. Allerdings stehen praktisch nur ganzzahlige Codewortlängen zur Verfügung, so daß hier der Huffman-Algorithmus angewendet wird. Die mit Hilfe des Huffman-Algorithmus erzielbare mittlere Codewortlänge ist durch

$$H(p) \leq \sum_{i \in \mathbf{I}} p(i) |\gamma(i)| \leq H(p) + 1 \tag{2.11}$$

   eingeschränkt.

3. Im letzten Schritt wird das Reproduktionscodebuch optimiert, hierzu werden α und γ konstant gehalten. Es wird die Abbildung $\beta : \mathbf{I} \to \mathbf{Y}$ gesucht, die die Gleichung (2.7) minimiert.

$$\beta(i) = \underset{\mathbf{y} \in \mathbf{Y}}{\operatorname{argmin}} E[d(\mathbf{X}, \mathbf{y}) | \alpha(i) = i] \tag{2.12}$$

   bestimmt den Zentroiden $\beta(i)$ der Voronoi-Region $\alpha(\mathbf{X}) = i$, der sich hier einfach aus dem arithmetischen Mittelwert der in der Voronoi-Region liegenden Eingangsvektoren $\mathbf{x}_i$ berechnen läßt.

Die Schritte 1 bis 3 werden solange wiederholt, bis das Abbruchkriterium

$$\frac{J^{(k)} - J^{(k+1)}}{J^{(k+1)}} < \epsilon \tag{2.13}$$

für z.B. $\epsilon = 0.005$ erfüllt ist.

Bei der Implementierung des iterativen Entwurfsalgorithmus für den Quadtree-Videocodec entfällt der Schritt 3, der für die Verbesserung des Reproduktionscodebuches sorgt. Eingangs wurde schon erwähnt, daß die Codebucheinträge aus dekodierten Bildblöcken bestehen und somit ständig variieren. Es besteht daher keine Möglichkeit der direkten Einflußnahme auf das Reproduktionscodebuch. Für die Intensitätsdifferenzen wird von vornherein ein eindimensionales Integer-Lattice verwendet.

2.2 Aufbau des Quadtree-Codecs

Der dieser Arbeit zugrunde liegende Videocodec ermöglicht die Bewegungs- und Intensitätskompensation von Bildblöcken variabler Größe (VBS-Codec). Die hierarchische Struktur eines Quadtrees läßt nicht nur die Verwendung effizienter Algorithmen zur Quantisierung zu, vor allem macht der einfache Zugriff auf die Lagrange-Kosten den, in den vorangegangenen Abschnitten beschriebenen, iterativen Entwurfsalgorithmus der ECVQ für diesen Codec praktikabel. Ein weiterer Aspekt ist, daß die Quadtree-Struktur auch die Verknüpfung von einfachen Segmentformen mit nur einem Bewegungsvektor zuläßt.

Auf eine Prädiktionsfehlercodierung wird vorerst zugunsten der blockbasierten Intensitätskompensation verzichtet. Es erfolgt also lediglich eine Übertragung der bewegungs- und intensitätskompensierten Prädiktionsparameter.

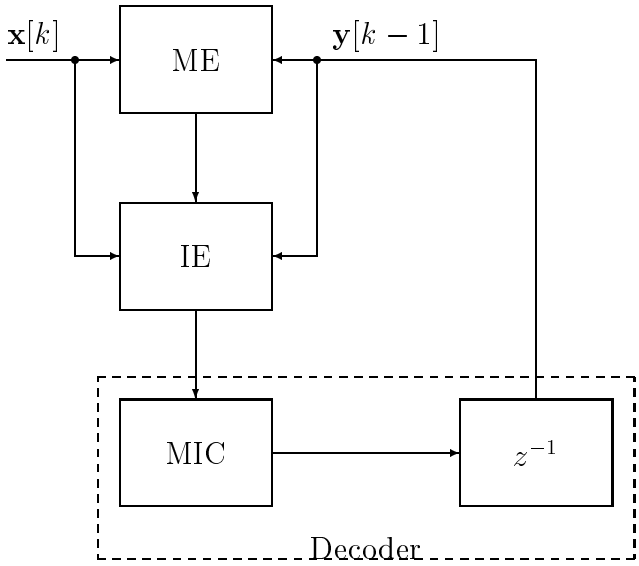

Abbildung 2.3: *Blockschaltbild des Quadtree-Videocodecs*

Das Blockschaltbild des Quadtree-Videocodecs ist in Abb. 2.3 dargestellt. Das aktuelle Bild der Viedeosequenz wird durch $\mathbf{x}[k]$ repräsentiert. Die Rekonstruktion des quantisierten Bildes $\mathbf{x}[k]$ stellt $\mathbf{y}[k]$ dar. Die Decodierung erfolgt anhand der translatorischen

Bewegungsvektoren $\mathbf{\Delta m}$, $\mathbf{\Delta n}$, den quantisierten Intensitätsdifferenzen $\hat{\mathbf{q}}$ und dem zuletzt decodierten Bild.

2.2.1 Strukturierung des Codier-Algorithmus

Der Coder erwartet Video-Sequenzen im 4:2:0-YUV-Format. Für QCIF-Bilder[1] ergeben sich Bildgrößen von 176×144 Pixel für die Y-Komponente und 88×72 Pixel für die U- und V-Komponente. Das aktuelle Einzelbild wird zeilenweise in Makroblöcke unterteilt. Die Makroblockgröße ist ein vielfaches der kleinsten verwendeten Blockgröße. Die kleinste zulässige Blockgröße ist bei diesem Codec 2×2. Für QCIF-Bilder werden für die Y-Komponente Makroblöcke der Größe 16×16 verwendet (für die U- und V-Komponente gilt entsprechend 8×8). Bei CIF-Bildern ist dagegen eine Makroblockgröße von 32×32 für die Y-Komponente üblich. Zur Verringerung des Aufwandes, verwaltet ein Makroblock sowohl die Helligkeitskomponente, als auch die Farbkomponenten. Es ergibt sich somit eine Aufteilung des Bides in 99 Makroblöcke.

Die Blöcke unterliegen einer hierarchischen Anordnung, d.h. ein Makroblock der Größe $2^\nu \times 2^\nu$ mit $\nu = 1, 2, \ldots$ wird in vier Blöcke der der Größe $2^{\nu-1} \times 2^{\nu-1}$ unterteilt und solch ein Teilblock wird wiederum in vier kleinere Blöcke aufgeteilt. Dies setzt sich fort, bis die kleinste zulässige Blockgröße erreicht ist.

Diese hierarchische Anordnung, läßt sich mit Hilfe eines Quadtrees (Abb. 2.4) einfach verwalten und effizient übertragen. Der Wurzel-Knoten stellt den Level-0-Block, den Makroblock dar. Jede weitere Unterteilung der Blöcke wird durch einen höheren Level im Quadtree dargestellt, wobei jedem Block ein Knoten zugeteilt ist. Die Anzahl k der möglichen Blockgrößen bestimmt die Tiefe $d = k - 1$ des Quadtree.

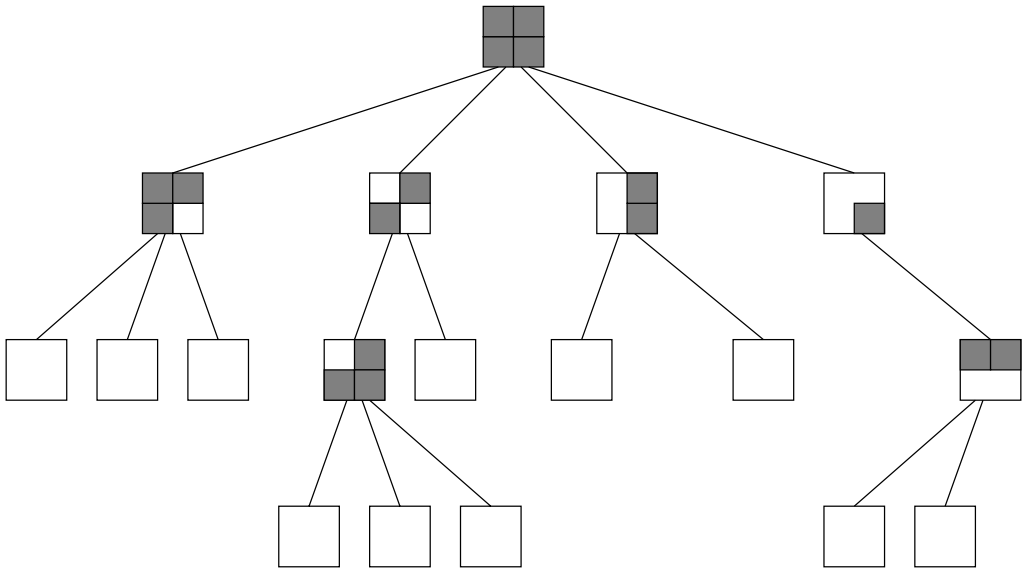

Abbildung 2.4: *Beispiel für eine Quadtree-Struktur*

Alle Teilblöcke die einem übergeordneten Knoten zugeteilt sind, werden zu einem Segment zusammengefaßt und gemeinsam codiert (jedem Segment wird eine örtliche Verschiebung

[1] Im weiteren erfolgen Erklärungen am Beispiel von QCIF-Bildern.

und eine Intensitätsdifferenz zugewiesen). Für einen Block ergeben sich 2^4 verschiedene Möglichkeiten der Aufteilung und Zusammensetzung zu Segmenten (Abb. 2.5).

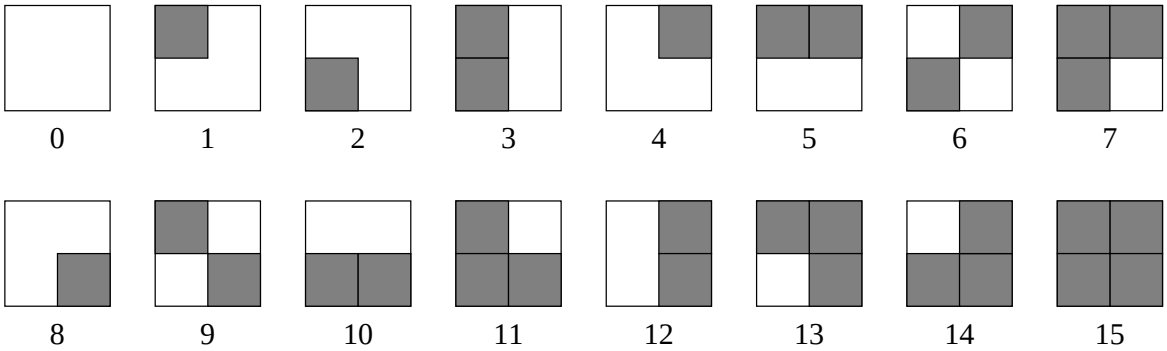

Abbildung 2.5: *Sechzehn verschiedene Möglichkeiten einen Block in vier Teilblöcke zu unterteilen. Die übergeordneten Teilblöcke werden zu Segmenten zusammengefaßt (weiß), während die untergeordneten Teilblöcke (grau) weiter zerlegt werden können.*

2.2.1.1 Segmentbasierte Bewegungs- und Intensitätskompensation

Die Segmentierung des aktuellen Bildes erfolgt vollständig. Den einzelnen Segmenten wird jeweils ein Tripel bestehend aus örtlicher Verschiebung und Intensitätsdifferenz zugeordnet. Die Bewegungskompensation erfolgt mittels Segmentmatching. Hierzu wird ausgehend von der Segmentposition m, n im aktuellen Bild $\mathbf{x}[k]$ die örtliche Verschiebung $\Delta m, \Delta n$ des Segmentes S bezüglich des rekonstruierten vorangegangenen Bildes $\mathbf{y}[k-1]$ ermittelt. Diese Verfahrensweise wird auch Rückwärtsschätzung genannt. Die Intensitätskompensation erfolgt hierzu in umgekehrter Richtung, nämlich ausgehend von $\mathbf{y}[k-1]$ zu $\mathbf{x}[k]$. Nach Quantisierung ergibt sich die Intensitätsdifferenz

$$\hat{q} = Q\left\{\frac{1}{|S[k]|}\sum_{(m,n)\in S[k]} \mathbf{x}[m,n,k] - \mathbf{y}[m+\Delta m, n+\Delta n, k-1]\right\} \tag{2.14}$$

für das betrachtete Segment. Schließlich erhält man für die rekonstruierten Bildelemente des Segments $S[k]$

$$\mathbf{y}[m,n,k] = \mathbf{y}[m+\Delta m, n+\Delta n, k-1] + \hat{q} \qquad (m,n) \in S[k]. \tag{2.15}$$

Die Aufteilung des Bildes in Segmente erfolgt durch einen Algorithmus zur Suche optimaler Baumstrukturen im Verbund mit der Bewegungs- und Intensitätskompensation. Als Optimalitätskriterium wird die Minimierung des Lagrange-Funktionals (2.5) eingesetzt.

2.2.1.2 Rekonstruktion mit Segmentüberlappungen

Bei der Bewegungskompensation von größeren Blöcken kann es zu Blockeffekten kommen, die subjektiv als sehr störend empfunden werden. Um diesen Effekt zu mindern, ist der Quadtree-Videocodec in der Lage innerhalb der Bildrekonstruktion mit Segmentüberlappung zu arbeiten. Die implementierte Methode gewichtet den zu kompensierenden

Bildpunkt sowie seine acht Nachbarn mit einer Maske (Abb. 2.6) und addiert diese in das zu rekonstruierende Bild. Bei größeren Segmenten ergänzen sich die Gewichtungen der benachbarten Bildelemente im Inneren des Segments zu eins. Dagegen kommt die Gewichtung an den überlappenden Segmenträndern zur Geltung. An Bildkanten erfolgt durch Verwendung spezieller Masken keine Segmentüberlappung.

$$\begin{matrix} \frac{2}{25} & \frac{3}{25} & \frac{2}{25} \\ \frac{3}{25} & \boxed{\frac{5}{25}} & \frac{3}{25} \\ \frac{2}{25} & \frac{3}{25} & \frac{2}{25} \end{matrix} \qquad \begin{matrix} 0 & \frac{5}{25} & \frac{2}{25} \\ 0 & \boxed{\frac{8}{25}} & \frac{3}{25} \\ 0 & \frac{5}{25} & \frac{2}{25} \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 \\ 0 & \boxed{\frac{13}{25}} & \frac{5}{25} \\ 0 & \frac{5}{25} & \frac{2}{25} \end{matrix}$$

Abbildung 2.6: *Masken zur Gewichtung eines im Inneren, am Bildrand und in der Bildecke liegenden Bildpunktes*

Zur Gewichtung größerer Blöcke lassen sich die in Abb. 2.6 dargestellten Masken leicht durch versetzte Überlagerung erweitern.

2.2.2 Prädiktion der Bewegungsvektoren

Durch prädiktive Codierung der Bewegungsparameter läßt sich nochmals ein Ratengewinn erzielen. Zwischen benachbarten Bildblöcken besteht eine hohe Korrelation bezüglich ihrer Bewegungsparameter. Werden diese Parameter zur Vorhersage des Bewegungsvektors eines anschließenden Blocks verwendet, ergibt sich oftmals eine erhebliche Verkürzung des nun zu übertragenden Differenzvektors gegenüber dem Bewegungsvektor ohne Prädiktion.

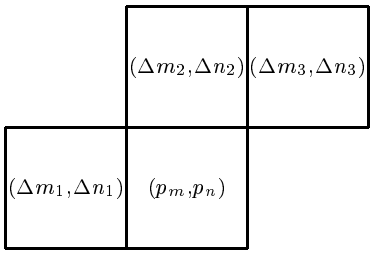

Abbildung 2.7: *Der Bewegungsvektor des aktuellen Makroblocks wird durch die Bewegungsvektoren der benachbarten Makroblöcke prädiziert*

Zur Gewinnung eines Schätzwertes (p_m, p_n) für den aktuellen Makroblock, werden die Top-Level-Bewegungsvektoren der in Abb. 2.7 dargestellten benachbarten Makroblöcke herangezogen. Liegt der aktuelle Makroblock in einer Bildecke oder am Bildrand, ergeben sich Ausnahmen. Für die verschiedenen Positionen des aktuellen Makroblocks berechnet sich der Schätzwert wie folgt:

1. Linke obere Bildecke:

$$(p_m, p_n) = (0, 0) \tag{2.16}$$

2. Oberste Bildzeile:

$$(p_m, p_n) = (\Delta m_1, \Delta n_1) \tag{2.17}$$

3. Linker oder rechter Bildrand:

$$(p_m, p_n) = (\Delta m_2, \Delta n_2) \tag{2.18}$$

4. Bildinneres:

$$(p_m, p_n) = (\mathrm{Med}(\Delta m_1, \Delta m_2, \Delta m_3), \mathrm{Med}(\Delta n_1, \Delta n_2, \Delta n_3)) \tag{2.19}$$

Weist ein Makroblock die Segmentform 15 auf, wird für dessen Top-Level-Bewegungsvektor der Wert $(0, 0)$ angenommen. Der zu übertragende Bewegungsvektor $(\Delta\bar{m}, \Delta\bar{n})$ ergibt sich aus der Differenz von tatsächlichen Bewegungsvektor $(\Delta m, \Delta n)$ und geschätzten Bewegungsvektor (p_m, p_n)

$$(\Delta\bar{m}, \Delta\bar{n}) = (\Delta m, \Delta n) - (p_m, p_n). \tag{2.20}$$

2.2.3 Algorithmen zur Suche optimaler Baumstrukturen

Die vollständige Suche nach einer optimalen Baumstruktur ist aufgrund der Komplexität ausgeschlossen[2]. Es werden daher Algorithmen angewandt, die die Komplexität der Optimierung erheblich verringern. Im Quadtree-Codec sind der Pruning- und der Growing-Algorithmus implementiert. Detailierte Informationen dazu findet man in [4] und [13]. Hier werden beide Algorithmen nur in kürzester Form dargestellt.

2.2.3.1 Pruning-Algorithmus

Der Pruning-Algorithmus wendet eine lokale Bottom-Up-Strategie an. Beginnend von der Wurzel wird folgender Algorithmus angewandt:

1. Sind die Lagrange-Kosten aller Kind-Knoten bekannt, gehe zu Schritt 2. Ansonsten suche unbestimmten Kind-Knoten auf und wiederhole Schritt 1.

2. Bestimme die optimale Segmentform, sowie Bewegungs- und Intensitätsparameter bezüglich der Lagrange-Kostenfunktion.

3. Ist die Wurzel des Baumes erreicht, breche Algorithmus ab. Ansonsten suche Eltern-Knoten auf und gehe zu Schritt 1.

[2] Für einen Quadtree der Tiefe 3 ergeben sich $4.8 \cdot 10^{19}$ Möglichkeiten [2].

2.2.3.2 Growing-Algorithmus

Der Growing-Algorithmus wendet eine lokale Top-Down-Strategie an. Beginnend von der Wurzel wird der folgende Algorithmus angewandt:

1. Bestimme die Lagrange-Kosten der Kind-Knoten des aktuellen Knotens. Die Kind-Knoten werden hierbei als Endknoten betrachtet.
2. Bestimme die optimale Segmentform, sowie Bewegungs- und Intensitätsparameter bezüglich der Lagrange-Kostenfunktion.
3. Suche erlaubte Kind-Knoten auf und gehe zu Schritt 1. Sind alle erlaubten Zweige bestimmt, ist der Algorithmus beendet.

2.3 Unrestricted Motion

Im bisherigen Quadtree-Videocodec werden Bewegungsvektoren zu Null gesetzt, die dazu führen, daß bewegungskompensierte Blöcke teilweise oder vollständig außerhalb des Bildes liegen. Beim Unrestricted-Motion-Modus wird der Suchbereich des vorangegangenen rekonstruierten Bildes durch Wiederholen der Kantenpixel rings um das ursprüngliche Bild vergrößert. Dies ermöglicht die Bewegungs- und Intensitätskompensation von Blöcken die teilweise außerhalb des eigentlichen Bildes liegen.

Im vorliegenden Codec werden die Bildränder im Urestricted-Motion-Modus um jeweils sechzehn Pixel erweitert.

Abb. 2.8 zeigt für die Testsequenz „Car Phone“ im Unrestricted-Motion-Modus einen Gewinn im Bereich von $0.1 \ldots 0.2dB$. Erwartungsgemäß decken sich beide PSNR-Messungen der Testsequenz „Salesman“ für den Fall mit und ohne Unrestricted-Motion-Modus. Der Grund hierfür liegt darin, daß im Bildrandbereich dieser Videosequenz keinerlei Bewegung existiert.

Der Unrestricted-Motion-Modus ist eine einfache Maßnahme, die nur geringfügig mehr Speicherplatz benötigt und sich, wenn überhaupt, nur verbessernd auswirkt. In allen zukünftigen Messungen wird daher der Unrestricted-Motion-Modus eingeschaltet.

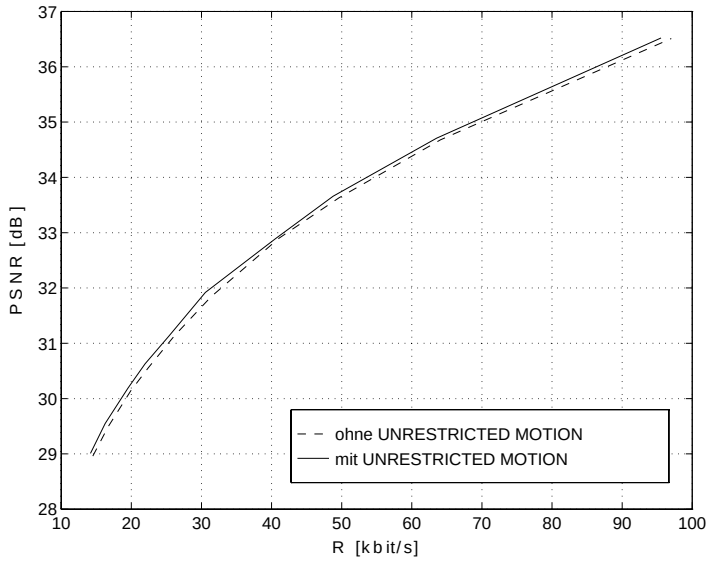

Abbildung 2.8: *Testsequenz „Car Phone“ (7.5 fps, 10 s) mit und ohne Unrestricted-Motion-Modus*

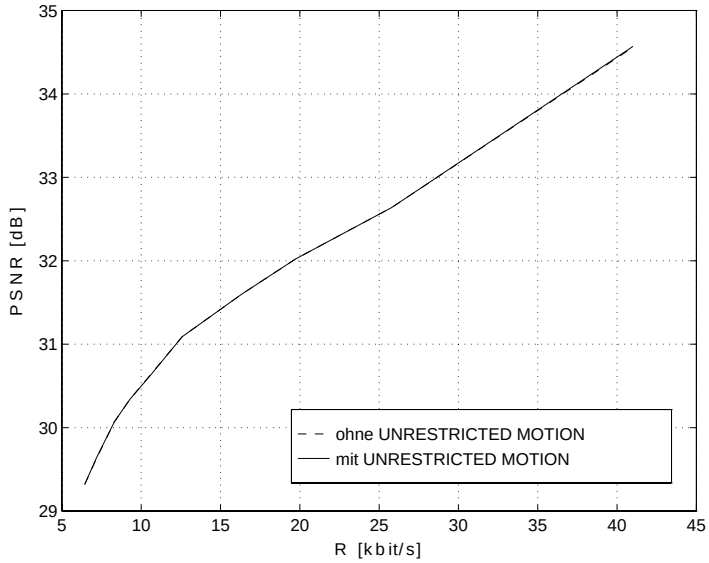

Abbildung 2.9: *Testsequenz „Salesman“ (7.5 fps, 10 s) mit und ohne Unrestricted-Motion-Modus*

Kapitel 3

Bewegungskompensation mit Halb-Pel-Genauigkeit

Bisher erfolgte die Bewegungskompensation des Quadtree-Codecs mit Integer-Pel-Genauigkeit. Reale Bewegungen verlaufen aber kontinuierlich und werden daher selten das Pixelraster treffen. Eine Ausweitung der Bewegungsschätzung auf Sub-Pixel-Positionen bei Aufhebung der Ratenrestriktion führt zu einer Erhöhung des Prädiktionsgewinns [6], [7]. Hierfür ist einerseits die erhöhte Genauigkeit der Bewegungsschätzung und andererseits der Filtereffekt der notwendigen Interpolation mit seiner Rauschminderung verantwortlich.

Die Genauigkeit der Bewegungskompensation des Quadtree-Videocodecs wird auf Halb-Pixel-Positionen erhöht. Ein bilineares Interpolationsfilter erzeugt die Zwischenpixelpositionen. Es wird nun untersucht, inwieweit die Erhöhung der Genauigkeit und der Filtereffekt bei niedrigen Bitraten den Verlust, durch die zu erwartende Erhöhung der mittleren Rate für die Bewegungsvektoren, entgegenwirken kann.

3.1 Implementierung der Halb-Pel-Bewegungskompensation

Damit die Bewegungskompensation mit Halb-Pixel-Genauigkeit möglich ist, muß das vorangegangene rekonstruierte Bild auch die Werte auf den Halb-Pixel-Positionen enthalten. Dies wird erreicht, in dem das im Integer-Pixel-Raster vorliegende Vorgängerbild einer örtlichen Interpolation unterzogen wird. Im Quadtree-Videocodec wird hierzu die bilineare Interpolation verwendet. Durch die bilineare Interpolation erfolgt eine Tiefpaßfilterung des Videobildes, was nebenbei eine Rauschreduktion bewirkt.

Die Bewegungskompensation mit Halb-Pel-Genauigkeit wird durch eine vollständige Suche realisiert. Weiterhin erfolgt die Implementierung so, daß Halb-Pel-Genauigkeit auf allen Leveln der Helligkeits- und der Farbkomponenten möglich ist.

3.2 Untersuchungsergebnisse

Für die Bewegungskompensation mit Halb-Pel-Genauigkeit wurde ein Codebuch trainiert, bei dem die Bewegungsschätzung auf allen Leveln der drei Komponenten mit Halb-Pel-Genauigkeit erfolgt. Zusätzlich wurde die Quantisierungsstufenhöhe der Intensitätsdifferenzen von vier auf zwei verringert. Die weiteren Codebücher die im Laufe der Untersuchungen verwendet wurden, sind durch Kombination aus anderen Codebüchern entstanden.

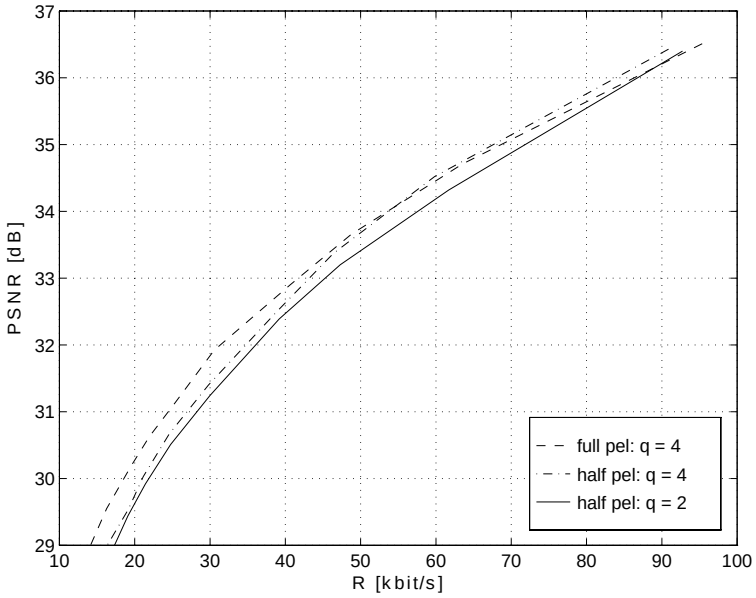

Abbildung 3.1: *Vergleich zwischen Half-Pel- und Full-Pel-Genauigkeit anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)*

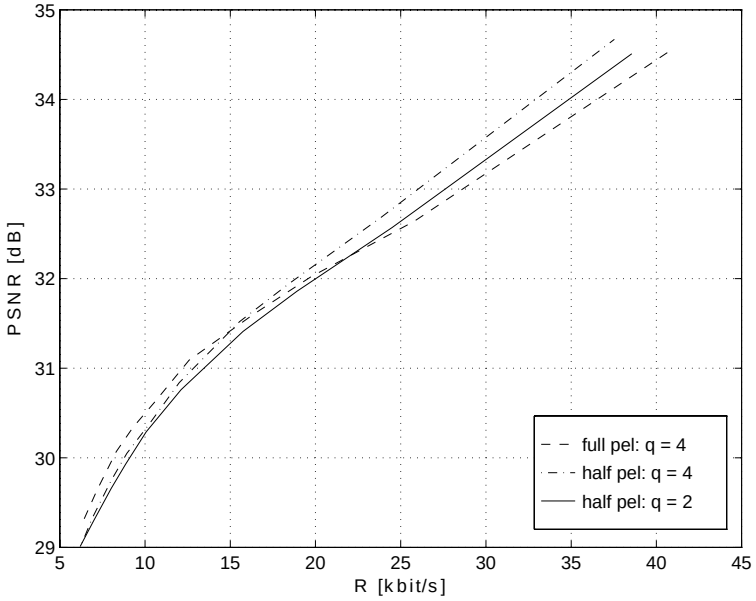

Abbildung 3.2: *Vergleich zwischen Half-Pel- und Full-Pel-Genauigkeit anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)*

Zunächst interessiert, wie sich die erhöhte Genauigkeit auf das Codierergebnis auswirkt. Die Abb. 3.1 und 3.2 zeigen eine Verschlechterung der PSNR-Werte im Bereich niedriger Bitraten gegenüber der Integer-Pel-Genauigkeit, wohingegen die Bewegungskompensation

mit Halb-Pel-Genauigkeit im Bereich höherer Bitraten zu einem verbesserten Ergebnis führen.

Das schlechtere Abschneiden der Bewegungskompensation mit Halb-Pel-Genauigkeit im Bereich niedriger Bitraten ist damit zu begründen, daß sich die Anzahl der Bewegungsvektoren je nach Level um den Faktor zwei bis vier erhöht. Dies ist zwangsläufig mit einer Erhöhung der mittleren Rate verbunden. Offenbar kann im Bereich niedriger Bitraten kein so hoher Qualitätszuwachs erzielt werden, so daß der erhöhte Bedarf an Bitrate zumindest ausgeglichen werden kann.

Visuell ergeben sich für den unteren Bitratenbereich deutliche Unterschiede zwischen den beiden Genauigkeitsklassen. Während bei pixelgenauer Bewegungskompensation bewegte Objekte zum Wabern neigen, verschwindet das Wabern bei halbpixelgenauer Bewegungskompensation nahezu, dafür verliert das Bild an Schärfe. Dies wird durch den Filtereffekt der halbpixelgenauen Bewegungskompensation verursacht.

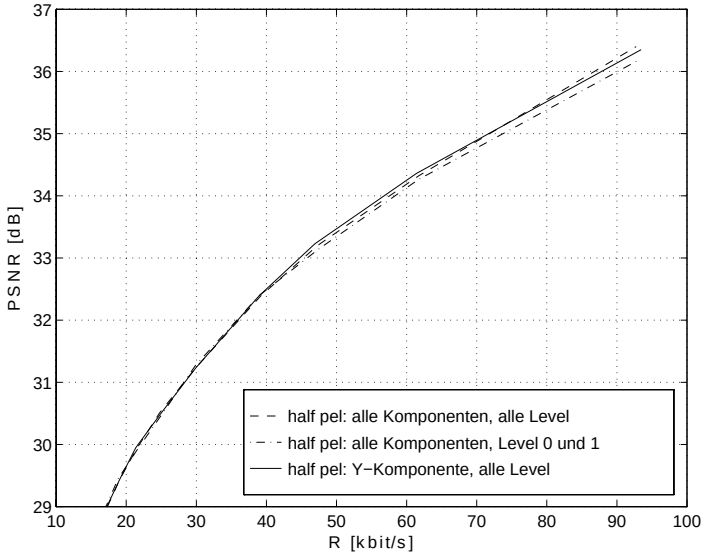

Abbildung 3.3: *Vergleich zwischen Half-Pel-Genauigkeit auf allen Leveln und nur auf Level 0 und 1 aller Komponenten und Half-Pel-Genauigkeit auf allen Leveln der Y-Komponente anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)*

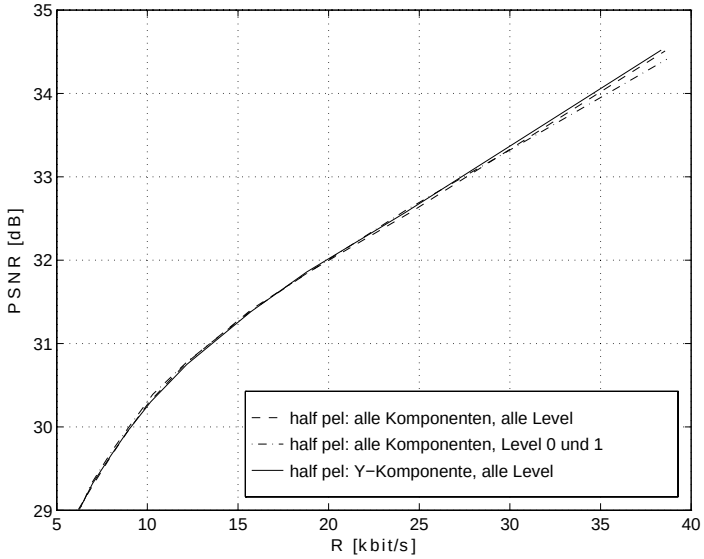

Abbildung 3.4: *Vergleich zwischen Half-Pel-Genauigkeit auf allen Leveln und nur auf Level 0 und 1 aller Komponenten und Half-Pel-Genauigkeit auf allen Leveln der Y-Komponente anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)*

Die Abb. 3.3 und 3.4 zeigen Meßergebnisse, die sich für zwei Varianten der Aufwandverminderung ergeben. Bei der ersten Variante erfolgt die Bewegungskompensation mit Halb-Pel-Genauigkeit nur auf den Leveln 0 und 1 aller Komponenten. Bei der zweiten Variante wird die Bewegungskompensation mit Halb-Pel-Genauigkeit nur für die Y-Komponente, aber dafür auf allen Leveln, verwendet. Im unteren Bitratenbereich ergibt sich kein Unterschied zwischen den beiden Varianten und der Bewegungskompensation mit Halb-Pel-Genauigkeit auf allen Leveln. Soll der Aufwand für die Bewegungsschätzung verringert werden, ist jedoch im oberen Bitratenbereich der zweiten Variante der Vorzug zu geben.

Kapitel 4

Langzeitprädiktion

Die Bewegungskompensation des Quadtree-Videocodecs beruht auf einem translatorischen Bewegungsmodell. Hierbei werden Bildsegmenten des zu codierenden Bildes örtliche Bewegungsvektoren bezüglich des zuletzt rekonstruierten Bildes, zugewiesen. Dieses einfache Bewegungsmodell wird umso ungenauer, je geringer die Bildfolgefrequenz ist. Dies ist dadurch begründet, daß Bewegungen wie Rotation oder Kamerazoom immer schlechter durch translatorische Bewegungen angenähert werden können. Aber auch das Aufdecken von vorher verdeckten oder neuen Objekten wirkt sich in stärkerem Maße aus.

Die bewegungskompensierte Langzeitprädiktion ist eine Methode den Prädiktionsfehler zu verringern [17]. Das translatorische Bewegungsmodell wird hierbei beibehalten. Allerdings werden zur Bewegungsschätzung weitere in der Vergangenheit liegende Bilder herangezogen. Man kann sich leicht vorstellen, daß sich Objekte oder Hintergrund der zwischenzeitlich verdeckt bzw. aus dem Bildbereich herausgeschoben war, ausgehend von früheren Bildern mit geringem Fehler und Aufwand prädizieren lassen. Trotz dieser intuitiven Erklärung handelt es sich hier lediglich um eine Erweiterung des Suchraumes für das Blockmatching.

4.1 Implementierung der Langzeitprädiktion

Zur Langzeitprädiktion wird eine vorgegebene maximale Anzahl vorangegangener decodierter Bilder benötigt. Diese werden in einem Langzeitspeicher zum unmittelbaren Zugriff aufbewahrt. Der Langzeitspeicher wird als Ringspeicher realisiert. Dies bietet den Vorteil einer einfachen Handhabung der Zugriffssteuerung auf die Bilddaten. Eine Prädiktion des Zeitparameters ist im Rahmen dieser Arbeit nicht vorgesehen. Weiterhin erfolgt die Untersuchung der Langzeitprädiktion nur auf den beiden obersten Hierarchiestufen des Quadtrees der Y-Komponente. Eine Erweiterung auf andere Level oder Komponenten ist aber bei Verwendung eines entsprechenden Codebuchs ohne weiteres möglich.

Der Zeitparameter wird in die Tabellen der Segmentformen integriert. Dies ändert jedoch nichts an der bisherigen hierarchischen Codestruktur des Codecs, da der Zeitparameter als weiteres Ereignis zugefügt wurde. Naheliegend wäre zwar die Tabelle der örtlichen Verschiebungen um den Zeitparameter zu erweitern, aber die Tabelle würde sich derart

vergrößern, daß der Satz vorhandener Trainingssequenzen keine ausreichende statistische Basis mehr für ein Codebuchtraining bieten würde. Die Integration des Zeitparameters in die Shapetabelle hat durchaus Sinn. Ohnehin besteht zwischen Segmentform und Bewegungsparameter eine enge Verknüpfung, es ist daher sicher kein Fehler, einer Segmentform die Zeitbasis zuzuweisen, aus der die Bewegungsschätzung erfolgen soll.

Um eine Vorstellung über den Gewinn gegenüber einem Produktcodebuch zu bekommen, wird auch die Langzeitprädiktion mit separater Codebuchtabelle für den Zeitparameter untersucht.

4.2 Untersuchungsergebnisse

Für die Untersuchung der Langzeitprädiktion erfolgte ein Codebuchtraining für die Bewegungskompensation mit Pixel-Genauigkeit, einer Langzeitspeichertiefe von zehn, wobei der Zeitparameter in die Shape-Tabelle integriert wurde und Quantisierungsstufenhöhe vier für die Intensitätskompensation. Die weiteren Codebücher entstanden durch Kombination vorhandener Codebücher. Für Halb-Pel-Genauigkeit fand das in Kapitel 3 entworfene Codebuch Verwendung. Zur Untersuchung des Unterschiedes, der sich ergibt, wenn für den Zeitparameter eine separate Codebuchtabelle verwendet wird, wurde die Codetabelle aus [17] übernommen.

4.2.1 Vergleich zwischen konventioneller Bewegungskompensation und Bewegungskompensation mit Langzeitspeicher

Die Abb. 4.1 und 4.2 zeigen, daß die Langzeitprädiktion gegenüber der konventionellen Bewegungskompensation anhand nur eines vorangegangenen Bildes deutlich überlegen ist. Bei der Sequenz „Salesman“ liegen die Gewinne für Pixel- und Halb-Pixel-Auflösung etwa im gleichen Rahmen von $0.2dB$ bis $0.4dB$. Hier ist der ansonsten ruhende Bildhintergrund flächenmäßig nur wenig von Verdeckung und Aufdeckung betroffen, so daß nur im geringen Maße von der Langzeitprädiktion profitiert werden kann.

Für die Sequenz „Car Phone“ sind wesentlich höhere Gewinne auszumachen. Mit Pixel-Genauigkeit liegen sie im Bereich von $0.4dB$ bis $0.7dB$, wohingegen bei Halb-Pixel-Genauigkeit der Gewinn bei ca. $0.8dB$ liegt. Der Hintergrund dieser Sequenz befindet sich ständig in Bewegung. Hier kann mit Hilfe der Langzeitprädiktion besonders effizient codiert werden.

Im Falle der Langzeitspeichertiefe 1 muß das Bild öfter feiner segmentiert werden, was eine Erhöhung der erforderlichen Codewortanzahl bedeutet, wogegen bei großer Bilddatenbasis (höhere Langzeitspeichertiefe) eher ein größeres Segment mit geringerem Codeaufwand, aber ähnlichem qualitativen Ergebnis gefunden werden kann. Hiervon profitiert insbesondere die Bewegungskompensation mit Halb-Pel-Genauigkeit in stark bewegten Bildern, da für deren Bewegungsvektoren eine höhere mittlere Rate in Kauf genommen werden muß.

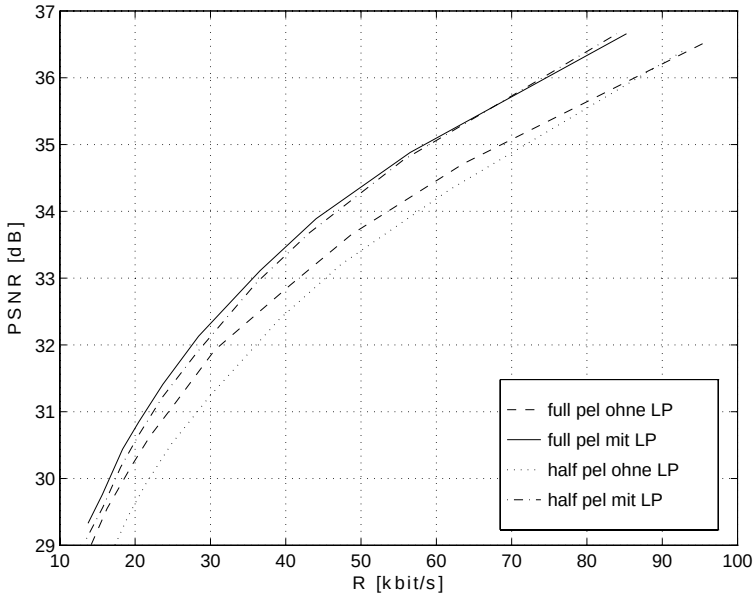

Abbildung 4.1: *Vergleich zwischen Langzeitspeichertiefe 1 und 10 anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)*

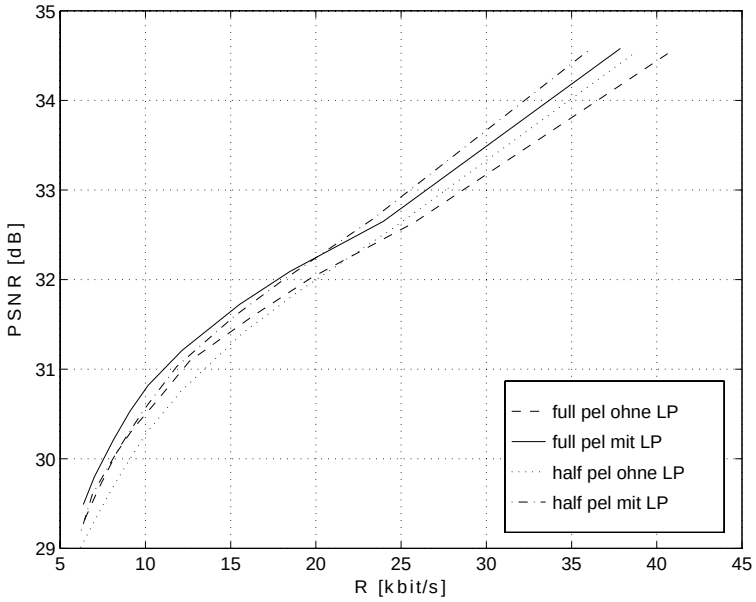

Abbildung 4.2: *Vergleich zwischen Langzeitspeichertiefe 1 und 10 anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)*

4.2.2 Vergleich zwischen Langzeitprädiktion mit separater Zeittabelle und mit in die Shapetabelle integriertem Zeitparameter

Die Erwartung, daß ein Codebuch, bei dem mehrere Ereignisse in einer Tabelle zusammengefaßt sind, eher bessere Resultate aufweist als ein Produktcodebuch, wird durch die Meßergebnisse in den Abb. 4.3 und 4.4 bestätigt. Zwar liegen die PSNR-Kurven bei der Sequenz „Salesman“ über den gesamten betrachteten Bereich fast aufeinander, aber bei der Sequenz „Car Phone“ ist im oberen Ratenbereich ein Gewinn von $0.3dB$, bei Integration des Zeitparameter in die Shapetabelle, auszumachen.

Die Integration des Zeitparameters in die Shapetabelle des Quadtree-Codecs ist also zu

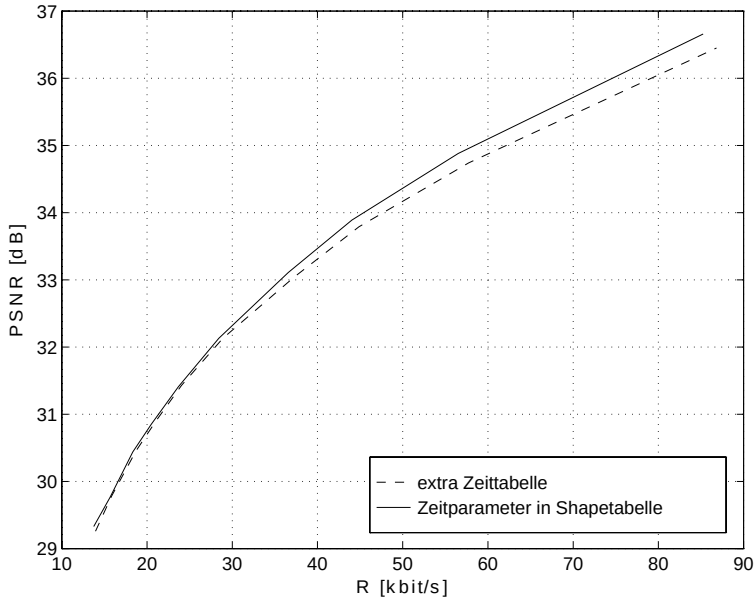

Abbildung 4.3: *Vergleich zwischen Langzeitprädiktion mit separater Zeittabelle und mit in die Shapetabelle integriertem Zeitparameter anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)*

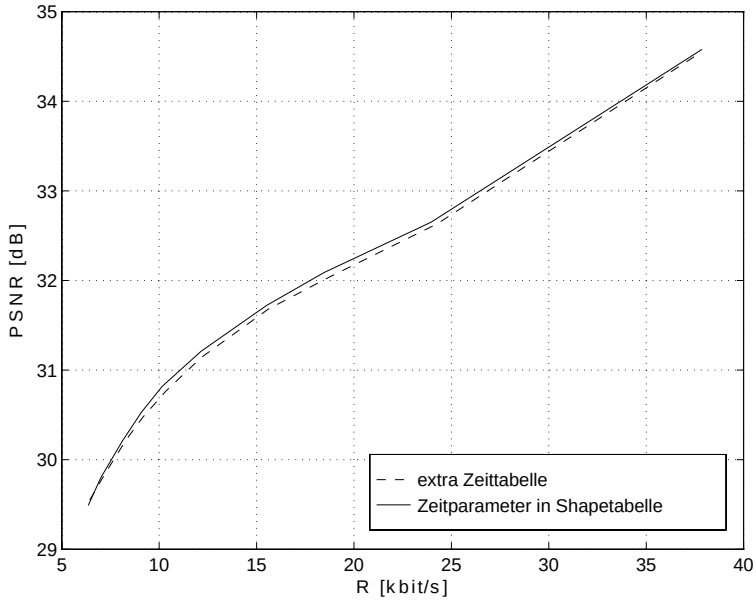

Abbildung 4.4: *Vergleich zwischen Langzeitprädiktion mit separater Zeittabelle und mit in die Shapetabelle integriertem Zeitparameter anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)*

empfehlen. Im Bereich niedriger Bitraten erzielt man aber auch durch Codierung des Zeitparameters mit Hilfe einer separaten Codetabelle kaum nennenswert schlechtere Ergebnisse.

Kapitel 5

Kombination mit DCT-Codierung

In der Videocodierung sind hybride Coder derzeit weit verbreitet. Bei diesen wird häufig entlang einer Bewegungstrajektorie des Sequenzinhaltes eine bewegungskompensierende Prädiktion vorgenommen. Das Prädiktionsfehlerbild wird schließlich transformationscodiert. Bei dem hier vorgestellten Quadtree-Videocodec wurde bisher, zugunsten der Intensitätskompensation, auf eine Fehlerbildcodierung verzichtet. Es soll jetzt untersucht werden, inwieweit sich das Codierergebnis durch die Einbeziehung einer Transformationscodierung des Prädiktionsfehlers verbessern läßt.

In Anlehnung an den H.263-Standard wird die weitverbreitete diskrete Kosinustransformation (DCT) eingesetzt. Neben dem Interframe-Codierverfahren, ermöglicht jetzt ein Intraframe-Modus die Codierung von Bildbereichen, bei dem die Bewegungskompensation bisher zu schlechten Ergebnissen geführt hat. Man denke hier z.B. an das erste Bild einer Videosequenz, oder Szenenwechsel innerhalb der Sequenz. Der Quadtree-Videocodec soll anhand eines Optimalitätskriteriums entscheiden, ob für einen Bildblock, wie bisher, eine Intensitätskompensation oder die Inter- bzw. Intra-DCT-Codierung vorgenommen wird.

5.1 Implementierung der DCT-Codierung

Im natürlichen Bildmaterial weisen benachbarte Pixel eine hohe Korrelation auf. Eine geeignete Transformation erzeugt hieraus dekorrelierte Koeffizienten, wobei wenige Koeffizienten die Energie konzentrieren. Viele Koeffizienten liegen nahe bei Null oder sind gleich Null. Nach Quantisierung und Lauflängencodierung (mit hoher Wahrscheinlichkeit folgen viele Nullen einander) und anschließender Entropiecodierung kann ein hoher Kompressionsgewinn erzielt werden.

Von entscheidender Bedeutung ist jedoch die Segmentierung des Bildmaterials [1]. Werden zu große Blöcke gewählt, kann es zum Überlappen aktiver in inaktive Bereiche innerhalb eines Blockes kommen. Andererseits nutzen zu kleine Blöcke die Nachbarschaftsbeziehungen weiter entfernt liegender Bildelemente nicht aus. In der Praxis haben sich daher Blockgrößen von 8×8 und 16×16 durchgesetzt.

[1] Meist werden quadratische Blöcke verwendet.

Prädiktionsfehlerbilder sind schon stark dekorreliert, weisen aber häufig immer noch Korrelation auf, weshalb auch hier oftmals eine Transformationscodierung sinnvoll ist.

In der Bild- und Videocodierung hat sich die DCT als Transformationsvervahren durchgesetzt. Sie überzeugt wegen ihrer einfachen Handhabung und den sehr guten Ergebnissen, die der aufwendigen Karhunen-Loeve-Transformation (KLT) nahe kommen.

Die Implementierung der DCT in den Quadtree-Codec erfolgt in Anlehnung an den H.263-Standard [8]. Die DCT wird nur innerhalb der Y-Komponente verwendet. Für die U- und V-Komponente ist keine Transformationscodierung vorgesehen. Der Codieralgorithmus unterscheidet für die 8×8-Blöcke der Helligkeitskomponente drei Varianten. Im Intra-Modus werden I-Blöcke ohne Prädiktion DCT-codiert. Im Inter-Modus werden entweder P-Blöcke bewegungskompensiert und der resultierende Prädiktionsfehler wird DCT-codiert oder Q-Blöcke werden wie bisher Quadtree-codiert. Es wird die Codierung vorgenommen, die im Sinne der Lagrange-Kosten optimal ist.

Die Koeffizienten eines transformierten Blocks werden über den gesamten Frequenzbereich mit konstanter Quantisierungsstufenhöhe quantisiert. Es gibt 31 Quantisierer, die geradzahlige Quantisierungsstufenhöhen im Bereich von 2 bis 62 zulassen. Eine Ausnahme gibt es jedoch: Im Intra-Modus wird der Gleichanteil immer mit Stufenhöhe 8 im Bereich von 8 bis 2032 quantisiert.

Eine im Sinne der Datenkompression erfolgreiche Transformation hat jetzt die Energie auf wenige Koeffizienten konzentriert. Nach der Quantisierung haben viele Koeffizienten den Wert Null. Es ist also wieder Redundanz vorhanden, die sich aber leicht durch Lauflängencodierung der zu Null gesetzten Koeffizienten beseitigen läßt. Die meiste Energie konzentriert sich im niederfrequenten Bereich, also in der linken oberen Ecke des Koeffizienten-Blocks. Werden die Koeffizienten zickzackförmig ausgelesen, so ergibt sich näherungsweise eine Sortierung der Koeffizienten gemäß ihrer Amplitude. Die Werte sind oft nach wenigen Schritten so klein, daß diese durch lange Nullfolgen getrennt werden. Am Ende der Kette befindet sich meist eine lange Nullfolge. Die Nullfolgen können jetzt effizient durch Lauflängencodierung zusammengefaßt werden. Die Codierung der Koeffizienten erfolgt gemäß dem H.263-Standard.

5.2 Untersuchungsergebnisse

Das Codebuchtraining wurde für die Bewegungskompensation mit Pixel-Genauigkeit vorgenommen. Die weiteren Codebücher entstanden durch Kombination vorhandener. Für Halb-Pel-Genauigkeit diente das in Kapitel 3 trainierte Codebuch als Basis.

Die Abb. 5.1 und 5.2 zeigen die Meßergebnisse der Testsequenzen „Car Phone“ und „Salesman“. Für die Bewegungskompensation mit Pixel-Genauigkeit läßt sich feststellen, daß der DCT-Modus keinen weiteren Gewinn bringt. Anders sieht es dagegen bei der Bewegungskompensation mit Halb-Pixel-Genauigkeit aus. Hier ist bei der Sequenz „Car Phone“ ein Gewinn von ca. $0.4dB$ und bei der Sequenz „Salesman“ ein Gewinn von ca. $0.2dB$ zu verzeichnen [2]. Die Ursache für den Gewinn des DCT-Modus bei der Bewegungskom-

[2] Im Bereich kleiner Bitraten ($< 10kbit/s$) ist zu beachten, daß die mit der DCT-Codierung eingeführten Header (s. Anhang B) eine Verschiebung der Meßkurve für den DCT-Modus nach rechts verursachen.

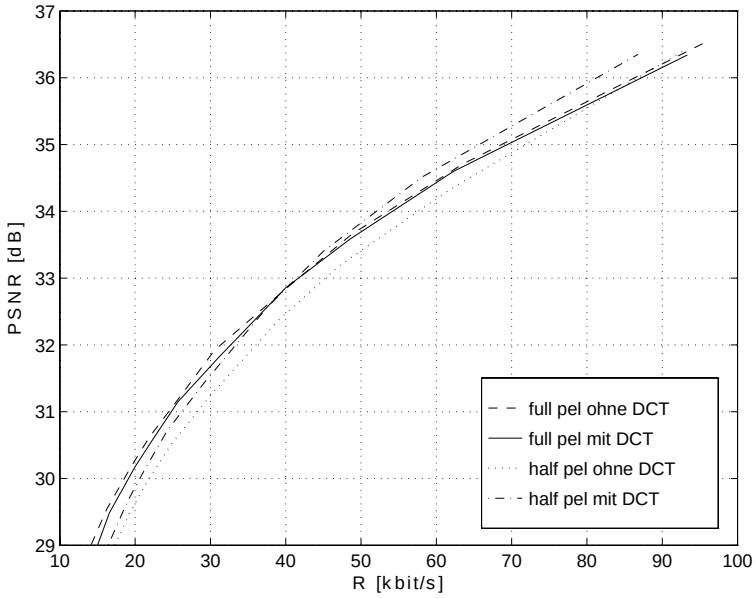

Abbildung 5.1: *Testsequenz „Car Phone“ (7.5 fps, 10 s) mit und ohne DCT-Modus*

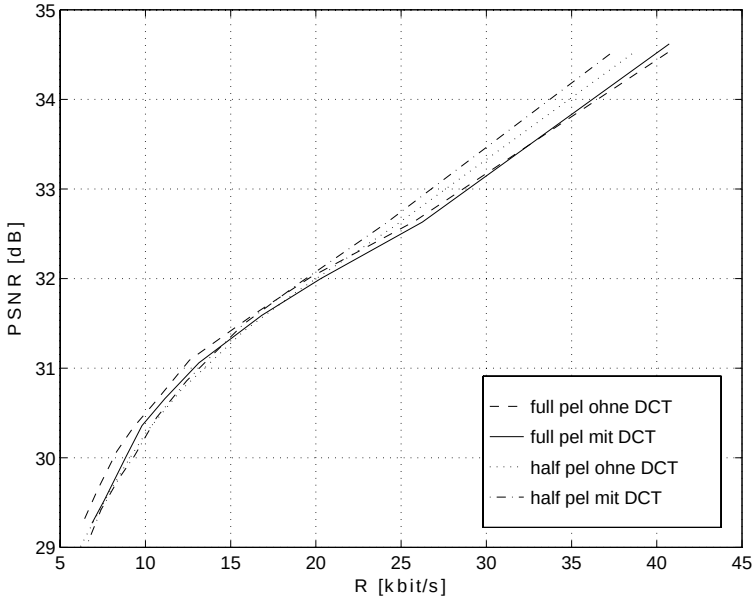

Abbildung 5.2: *Testsequenz „Salesman“ (7.5 fps, 10 s) mit und ohne DCT-Modus*

pensation mit Halb-Pixel-Genauigkeit liegt einerseits in der erhöhten Genauigkeit der Bewegungsschätzung und andererseits im Filtereffekt, der der Bewegungskompensation mit Sub-Pixel-Genauigkeit eigen ist. Offenbar sind die höherfrequenten Spektralanteile weniger stark bei der Bewegungskompensation mit Halb-Pixel-Genauigkeit ausgeprägt.

Kapitel 6

Weitere Schritte zur Optimierung des Codecs

Die in diesem Kapitel untersuchten Methoden, den Quadtree-Codec zu optimieren, beruhen auf der Separierbarkeit der Lagrange-Kosten eines Bewegungsvektors von den übrigen Kosten innerhalb des Lagrange-Funktionals.

Bisher erfolgte die Intensitätskompensation anhand eines eindimensionalen Integer-Lattice. Die Differenz der Mittelwerte von Originalsegment und bewegungskompensierten Segment ergab durch einfache Berechnung den Index des Codewortes für die zu übertragende Intensitätsdifferenz. Die Überprüfung, ob das Codewort einer benachbarten Intensitätsdifferenz zu günstigeren Lagrange-Kosten führt, war nicht vorgesehen. Es wird eine Intensitätsquantisierung auf der Basis einer Intensitätsdifferenzsuche unter Berücksichtigung der minimalen Lagrange-Kosten untersucht.

Bei der Bewegungsschätzung ist die Berechnung des quadratischen Fehlers für jeden möglichen Bewegungsvektor das rechenintensivste Element. Es wird daher am Ende des Kapitels die Frage geklärt, ob durch blockweises Abspeichern und Wiederverwenden der SSE-Werte eine signifikante Beschleunigung des Coders erreicht werden kann.

6.1 Quantisierung der Intensitätsdifferenz unter Berücksichtigung der Lagrange-Kosten

Die Quantisierung der Intensitätsdifferenz erfolgte bisher durch Verwendung der ihr am nächsten gelegenen Gitterposition. Das entsprechende Codewort konnte durch eine einfache Indexberechnung ermittelt werden. Jetzt werden die Intensitätsdifferenzen derart quantisiert, daß sich minimale Lagrange-Kosten ergeben. Der Gedanke der sich hier verbirgt ist, daß eventuell neben der Quantisierung mit dem geringsten Fehler eine Quantisierung mit zwar größeren Fehler aber geringerem Ratenaufwand existiert, so daß sich geringere Lagrange-Kosten ergeben. Damit die Suche nach der, im Sinne der Lagrange-Kosten, besten Quantisierung nicht unnötig aufwendig wird, ist der Suchbereich einzuschränken.

6.1.1 Einschränkung des Suchbereichs

Die Lagrange-Kosten zur Kompensation der Intensitätsdifferenz für einen Block der Größe $N = hw$ ergeben sich zu

$$\begin{aligned} J_q &= \sum_{k=1}^{N} (x_k - y_k - (q + \Delta q))^2 + \lambda R \\ &= \sum_{k=1}^{N} \left(x_k^2 + y_k^2 + (q + \Delta q)^2 - 2x_k y_k - 2(q + \Delta q)(x_k - y_k) \right) + \lambda R. \end{aligned} \tag{6.1}$$

Unter Berücksichtigung der Intensitätsdifferenz

$$q = \frac{1}{N} \sum_{k=1}^{N} (x_k - y_k) \tag{6.2}$$

erhält man für Gleichung (6.1)

$$J_q = \sum_{k=1}^{N} (x_k - y_k)^2 + N(q + \Delta q)^2 - 2Nq(q + \Delta q) + \lambda R. \tag{6.3}$$

Für die quantisierte Intensitätsdifferenz wird $\hat{q}$ eingeführt. Einsetzen der Differenz $\Delta q = \hat{q} - q$ in Gleichung (6.3) führt schließlich auf eine andere Darstellung von Gleichung (6.1)

$$J_q = \sum_{k=1}^{N} (x_k - y_k)^2 + N\hat{q}^2 - 2Nq\hat{q} + \lambda \hat{R}. \tag{6.4}$$

Zur Einschränkung des Suchbereichs nach einem bezüglich der Lagrange-Kosten optimalen Wert q_{opt} wird zunächst ein q_i mit dem geringsten Abstand zu q gesucht. Mit q_i ist auch R_i bekannt und es kann eine obere Kostengrenze

$$J_{q,max} = \sum_{k=1}^{N} (x_k - y_k)^2 + Nq_i^2 - 2Nqq_i + \lambda R_i \tag{6.5}$$

bestimmt werden. Für q_{opt} muß also

$$J_{q_{opt}} \leq J_{q,max} \tag{6.6}$$

gelten oder anders formuliert

$$q_{opt}^2 - 2qq_{opt} \leq q_i^2 - 2qq_i + \frac{\lambda}{N}(R_i - 1). \tag{6.7}$$

Damit Schranken für den Suchbereich bestimmt werden können, wird Ungleichung (6.7) als Gleichung betrachtet und deren Nullstellen gesucht. Es ergeben sich somit

$$\bar{q}_{0,1} = q \pm \sqrt{(q - q_i)^2 + \frac{\lambda}{N}(R_i - 1)} \tag{6.8}$$

als obere und untere Grenze für den Suchbereich nach einer geeigneten Quantisierung der Intensitätsdifferenz.

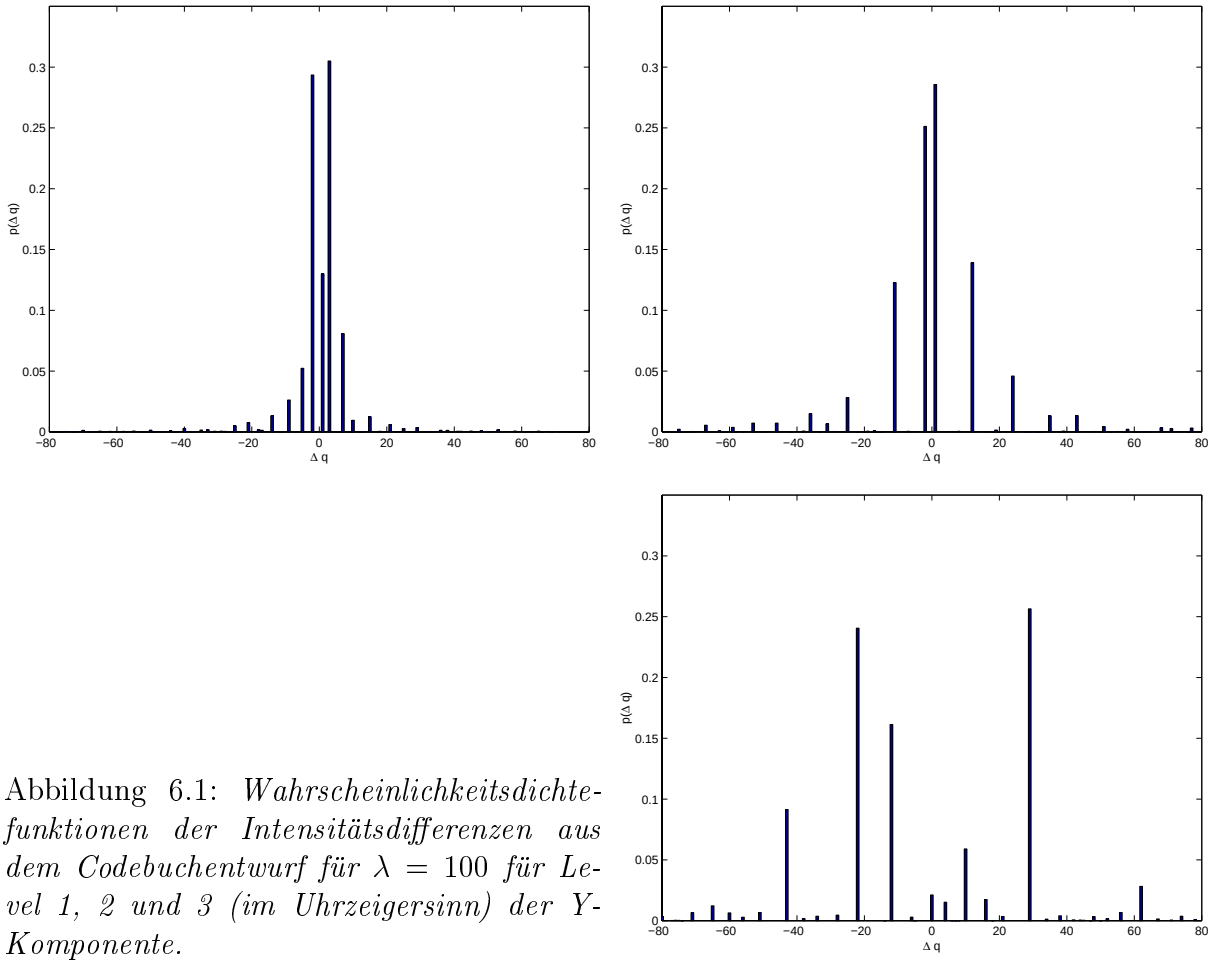

Abbildung 6.1: *Wahrscheinlichkeitsdichtefunktionen der Intensitätsdifferenzen aus dem Codebuchentwurf für $\lambda = 100$ für Level 1, 2 und 3 (im Uhrzeigersinn) der Y-Komponente.*

6.1.2 Untersuchungsergebnisse

Das Codebuchtraining ging von einem gleichverteilten Integer-Lattice mit Abstand eins für die Intensitätsdifferenzen aus. Weiterhin erfolgte das Codebuchtraining für Halb-Pel-Genauigkeit.

Abb. 6.1 zeigt die Wahrscheinlichkeitsdichten der Intensitätsdifferenzen für die Y-Komponente, die sich aus dem Codebuchtraining ergeben haben. Die Wahrscheinlichkeitsdichtefunktionen werden mit zunehmender Baumtiefe „flacher“. Für Level 0 wird eine Wahrscheinlichkeitsdichtefunktion in Form eines Dirac-Impulses bei Null vorausgesetzt. Das etwas unregelmäßige Erscheinungsbild der Wahrscheinlichkeitsdichtefunktion der untersten Ebene ist auf die offenbar nicht ganz ausreichende statistische Basis der Trainingssequenzen zurückzuführen.

Der Vergleich zwischen fester und variabler Intensitätsquantisierung (Abb. 6.2 und 6.3) zeigt bei der Sequenz „Car Phone“ lokal einen geringfügigen Gewinn von $0.2dB$ der optimalen Quantisierung gegenüber der Quantisierung mit fester Schrittweite. Dieser Bereich liegt um den Wert $\lambda = 100$ (Markierung in den Abbildungen), für den das Codebuchtraining erfolgte. Die optimale Quantisierung der Intensitätsdifferenz erzielt bei der Sequenz „Salesman“ keinen nennenswerten Gewinn gegenüber der Quantisierung mit Schrittweite vier. Für höhere Bitraten werden mit dem trainierten Codebuch sogar schlechtere Ergeb-

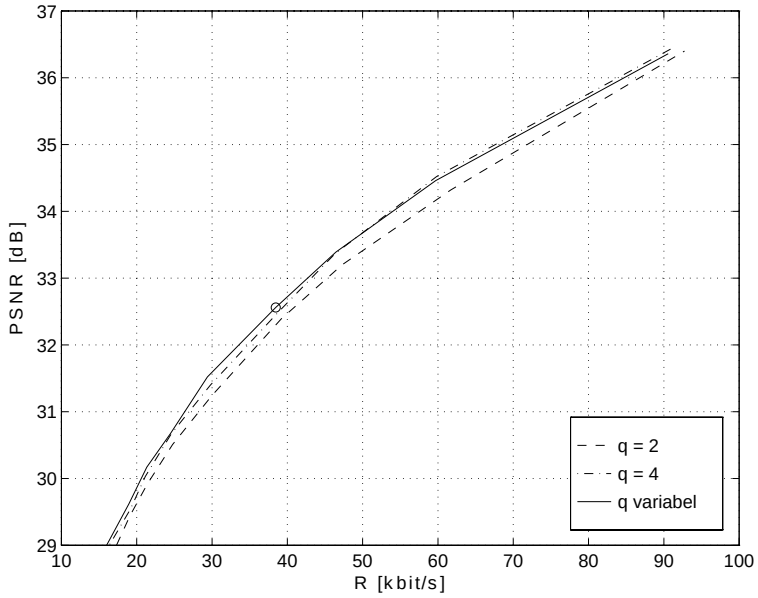

Abbildung 6.2: *Vergleich zwischen fester und variabler Intensitätsdifferenzquantisierung anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)*

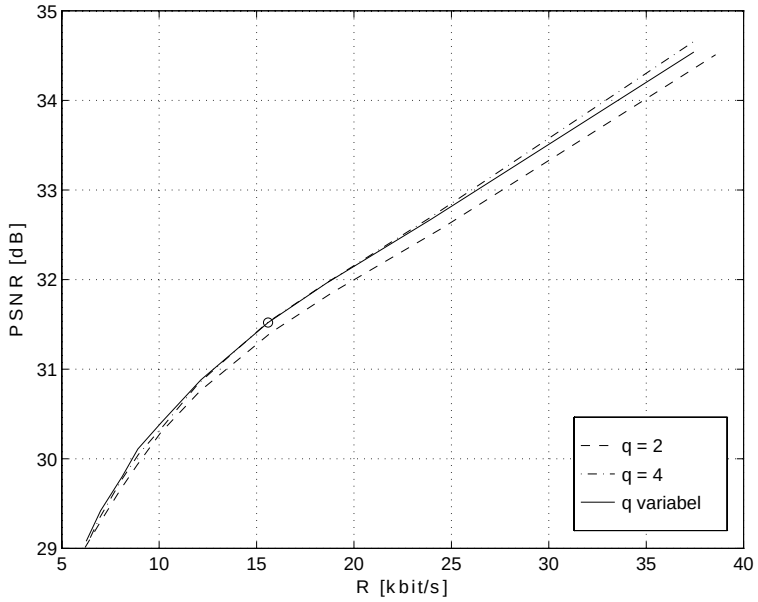

Abbildung 6.3: *Vergleich zwischen fester und variabler Intensitätsdifferenzquantisierung anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)*

nisse erzielt als bei Verwendung eines Integer-Lattice mit Abstand vier.

Die Abb. 6.4 und 6.5 zeigen, daß die Quantisierung der Intensitätsdifferenzen bezüglich optimaler Lagrange-Kosten bei Verwendung eines Codebuchs mit Integer-Lattice zu keinem besseren Ergebnis führt.

Zusammengefaßt kann gesagt werden, daß die bezüglich der Lagrange-Kosten optimale Quantisierung der Intensitätsdifferenz allenfalls im Bereich des Lagrange-Parameters für den das Codebuch trainiert wurde zu einem Gewinn gegenüber der Quantisierung anhand eines Integer-Lattice mit Abstand vier führt. Hier kann die Anwendung der Zentroiden-Regel beim Codebuchentwurf zu einem geringfügig verbesserten Ergebnis führen. Hierbei bildet der Mittelwert über alle $n(i)$ Intensitätsdifferenzen $q_k(i)$ die zu einem bestimmten

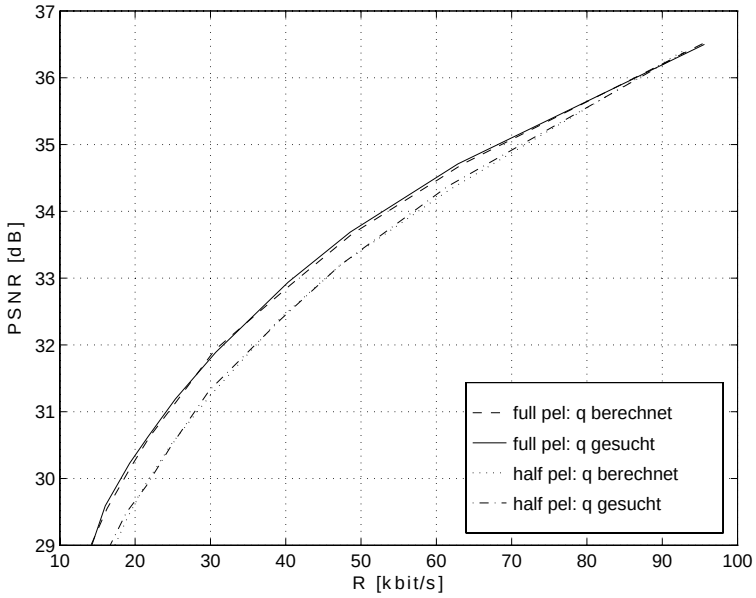

Abbildung 6.4: *Vergleich zwischen fester und variabler Intensitätsdifferenzquantisierung anhand der Testsequenz „Car Phone“ (7.5 fps, 10 s)*

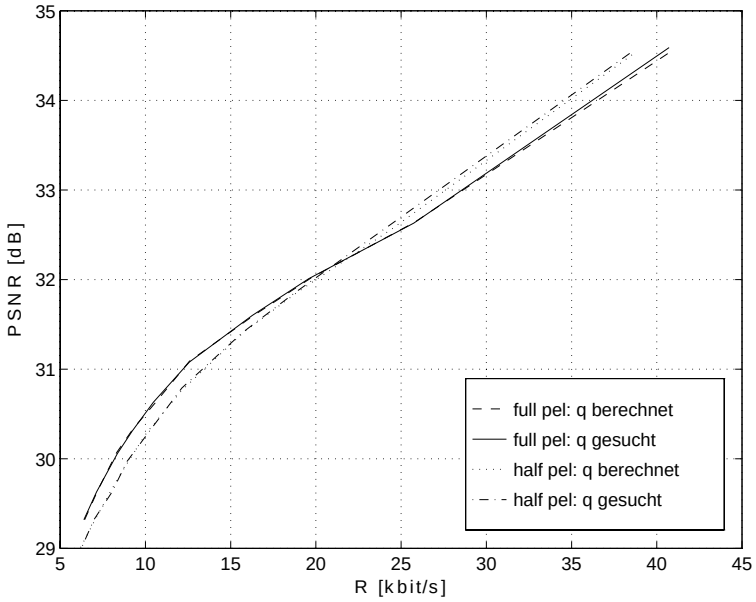

Abbildung 6.5: *Vergleich zwischen fester und variabler Intensitätsdifferenzquantisierung anhand der Testsequenz „Salesman“ (7.5 fps, 10 s)*

Index i geführt haben den neuen Repräsentanten

$$\hat{q}(i) = \frac{1}{n(i)} \sum_{k=1}^{n(i)} q_k(i) \tag{6.9}$$

innerhalb des Codebuchs.

Betrachtet man den Aufwand für die optimale Quantisierung der Intensitätsdifferenzen gegenüber der Verwendung eines Integer-Lattice mit einfacher Indexberechnung, so stellt die letzte Variante bei einem Gitterabstand von vier einen guten Kompromiß hinsichtlich des Aufwands und des Ergebnisses über den untersuchten Ratenbereich dar.

6.2 Ansatz zur Beschleunigung der Bewegungsschätzung

Der Quadtree-Codec besitzt eine ausgefeilte Codierstrategie, um eine hohe Recheneffizienz zu erreichen. So sind einerseits die Shape- und Displacement-Tabellen des Codebuchs nach den Auftrittswahrscheinlichkeiten sortiert und andererseits werden im Codieralgorithmus an geeigneten Stellen Abbruchkriterien abgefragt, die bei ungünstigen Schape-, Displacement- bzw. Intensitätswerten zu einem vorzeitigem Abbruch der Suche führen. Konkret führt die Sortierung der Codebuchtabellen nach Auftrittswahrscheinlichkeiten dazu, daß meistens die optimalen Bewegungsparameter frühzeitig gefunden werden. Damit sich daraus ein Vorteil ergibt, werden an geeigneten Stellen die Lagrange-Kosten des bisherigen besten Match mit den bis dahin aktuellen Lagrange-Kosten verglichen. Übersteigen die aktuellen Lagrange-Kosten die des bisher besten Match, wird der Suchprozeß des aktuellen Parameters abgebrochen und mit dem nächsten Parameter fortgefahren. Obwohl mit dieser Strategie erheblich Rechenzeit eingespart wird, soll dennoch untersucht werden, ob eine weitere Ersparnis möglich ist, indem schon berechnete Distortionwerte abgespeichert und gegebenenfalls wiederverwendet werden.

Die Implementierung erfolgt so, daß für jede Baumebene, mit Ausnahme der obersten, eine Matrix abgespeichert wird. Jedes Matrixelement korrespondiert mit einem Knoten der Baumebene und verweist wiederum auf eine Matrix deren Größe dem Suchbereich der Bewegungsvektoren entspricht. Der Suchbereich wird hier auf eine Größe von $[-16, 16]$ begrenzt. In diese Matrix werden die Distortionwerte, die sich aus der Bewegungsvektorensuche ergeben, eingetragen. Der Pruning-Algorithmus benutzt eine Bottom-Up-Strategie, so daß die Distortionwerte der untersten Ebene zuerst vorliegen. Gelangt der Algorithmus in eine übergeordnete Ebene, wird für jeden Bewegungsvektor überprüft, ob für die Teilblöcke schon berechnete Distortionwerte vorliegen. Sind Distortionwerte vorhanden, werden diese addiert und in die entsprechende Matrix der aktuellen Ebene eingetragen. Für fehlende Teilblöcke müssen die partiellen Distortionwerte natürlich noch separat berechnet werden. Im günstigsten Fall brauchen nur vier Distortionwerte der untergeordneten Teilblöcke addiert werden.

Der Nachteil dieser Strategie ist der erheblich vergrößerte Speicherbedarf und natürlich ist noch der erhöhte Verwaltungsaufwand zu berücksichtigen. Tatsächlich zeigt sich auch, daß mit dieser Methode keine Rechenzeitersparnis zu erzielen ist. Testläufe haben ergeben, daß die Codierzeiten mit und ohne dieser Strategie etwa gleich sind. Offenbar arbeitet der Coder schon so effizient, daß mit dieser speicherintensiven Methode keine weitere Rechenzeitersparnis erreicht werden kann.

Kapitel 7

Zusammenfassung und Ausblick

Die Untersuchungen am Quadtree-Codec haben gezeigt, daß dieser im Bereich niedriger Bitraten seine Möglichkeiten nahezu ausschöpft. Allein mit Hilfe der Langzeitprädiktion konnte in diesem Bereich noch ein nennenswerter Gewinn erzielt werden. Für höhere Bitraten bringt die Bewegungskompensation mit Halb-Pel-Genauigkeit, insbesondere in Kombination mit der DCT-Codierung, eine Qualitätssteigerung bei gleicher Rate. Dagegen führt die DCT-Codierung bei nur pixelgenauer Bewegungskompensation zu keinerlei Gewinn, so daß in diesem Fall die DCT eingespart werden kann. Mit Hilfe der Langzeitprädiktion wird immer ein besseres Codierergebnis erzielt, dies gilt besonders für Sequenzen mit aktiven Bildinhalt.

Die Einbeziehung der Intensitätsdifferenzquantisierung in den Optimierungsprozeß hat nicht den erhofften Erfolg gebracht. Zwar wird bei Anwendung der Zentroidenregel des iterativen Entwurfsalgorithmus lokal ein Optimum erzielt, aber über den betrachteten Ratenbereich stellt die Verwendung des eindimensionalen Integer-Lattice mit Abstand vier einen guten Kompromiß dar. Hinzu kommt, daß die Intensitätsdifferenzsuche unter Berücksichtigung der Lagrange-Kosten zu einem erheblich gesteigerten Aufwand gegenüber der einfachen Berechnung des Codewortindex eines Integer-Lattice führt.

Die Untersuchungen am Quadtree-Codec können noch nicht als abgeschlossen gelten. Es gibt noch einige interessante Aspekte, deren Untersuchung sich durchaus lohnt:

- Für die Prädiktion der Bewegungsparameter könnte eine ausgefeiltere Strategie verwendet werden. So sollte auch die Möglichkeit bestehen, die Bewegungsparameter einzelner Segmente separat zu schätzen.
- Die implementierte DCT-Codierung verwendet einen gleichbleibenden durch λ bestimmten Quantisierer. Für optimale Codierergebnisse sollte die Auswahl des Quantisierers mit in den Optimierungsprozeß einbezogen werden.
- Für DCT-codierte Blöcke wird kein Overlapping verwendet. Um Block-Effekte zu vermindern, sollte eine geeignete Overlapping-Methode eingebaut werden.
- Der Quadtree-Codec ist von seiner Struktur her für eine skalierbare Codierung geeignet.
- In den Codier-Algorithmus sollte eine Ratenregulierung implementiert werden.

Anhang A

Programmbeschreibung

Die Untersuchungen dieser Arbeit erfolgen an einem realen Codec. Hierbei bilden die in [2] entworfenen Programm-Module das Ausgangsmaterial. In [2] ist zu diesen Modulen eine ausführliche Programmbeschreibung zu finden. Hier werden nur die Erweiterungen bzw. Änderungen beschrieben, die sich im Laufe der Arbeit ergeben haben.

A.1 Codec

A.1.1 Coder

Das Programm *QtEncode* stellt den Coder dar, der eine Videosequenz in eine Codesequenz mit deutlich reduzierter Datenrate überführt. Der Coder ist in der Lage während des Codiervorgangs das decodierte Video anzuzeigen bzw. zu schreiben. Der Coder erlaubt die Angabe folgender Parameter:

```
VBS coder version 6.0
(C) 1996 University of Erlangen-Nuremberg
    T. Wiegand, M. Flierl, K. Pieper

QtEncode [-<help>hiorR<RFF>t<info>CK<ff>nmabsfkDSMdOlzG
          <um><hp><hpr><qO><LTPM><DCT>c<Hd>]
    -help        verbose help
    -h           help
    -i    [name] original sequence file name
                 []
    -o    [name] code file name
                 [bits.itd]
    -r    [name] reconstructed sequence file name
                 []
    -R    [num]  target bit rate [0 bps]
    -RFF  [num]  bits used for first frame [0 bits]
    -t    [name] trace file name
```

```
-info [name] information file name
-C    [name] codebook file name
             [/POOL_HOME/stud/pieper/quadtree/Codec/tab/itd1.0.cbk]
-K    [name] mask file name
             [/POOL_HOME/stud/pieper/quadtree/Codec/tab/mask.tab]
-ff   [name] first frame file name
-n    [num]  input image width                    [176]
-m    [num]  input image height                   [144]
-a    [num]  start                                [0]
-b    [num]  end                                  [99]
-s    [num]  frame step                           [1]
-f    [num]  frequency                            [30]
-k    [num]  image file format                    [0]
               0: LNT format
               1: catenate sequence file format
               2: khoros format
-D    [num]  output mode                          [2]
               0: write only code file
               1: write reconstruction
               2: display reconstruction
               3: display imposed reconstruction
-S    [num]  code buffer size (byte)              [100]
-M    [num]  macroblock size                      [16]
-d    [0/1]  differencing                         [1]
-O    [0/1]  overlapping                          [1]
-l    [num]  lambda                               [100]
-z    [num]  zoom factor                          [1]
-G           use growing algorithm
-um   [0/1]  unrestricted motion flag             [1]
-hp   [num]  half pel accuracy                    [1]
-hpr  [0/1]  half pel refinement flag             [0]
-qO   [0/1]  optimal intensity quantization flag [0]
-LTPM [num]  long time prediction memory          [1]
-DCT  [0/1]  DCT                                  [1]
-c    [name] DCT data path
             [/SUN_HOME/mit/cids/H263/tab/]
-Hd   [0/1]  code file header flag                [1]
```

Der Coder arbeitet im Unrestricted-Motion-Modus. Der Parameter `-um 0` schaltet den Unrestricted-Motion-Modus aus.

Mit dem Parameter `-hp n` kann eingestellt werden, für welche Komponenten der Coder die Bewegungskompensation mit Halb-Pel-Genauigkeit durchführen soll. Die Halb-Pel-Genauigkeit wird mit `-hp 1` für die Y-Komponente, mit `-hp 2` für die U-Komponente und mit `-hp 4` für die V-Komponente eingeschaltet. Soll für mehrere Komponenten die Bewegungskompensation mit Halb-Pel-Genauigkeit erfolgen, so werden die entsprechenden Werte addiert. Durch Angabe von `-hp 7` wird die Bewegungskompensation mit erhöhter

Genauigkeit für alle Komponenten ermöglicht. Das Ausschalten der Halb-Pel-Genauigkeit für einzelne Komponenten, bei Verwendung eines entsprechenden Codebuchs, ist sinnvoll, da der Speicherbedarf des Coders dadurch erheblich vermindert werden kann. Der Refinement-Modus, mit `-hpr 1` einschaltbar, wird nur vom Growing-Algorithmus unterstützt. Hier werden zunächst alle Pixelpositionen nach dem besten Match abgesucht. Ist der pixelgenaue Bewegungsvektor mit den niedrigsten Lagrange-Kosten gefunden, werden die benachbarten Halb-Pixelpositionen nach einem besseren Bewegungsvektor abgesucht.

Die Suche nach einer bezüglich der Lagrange-Kosten optimalen quantisierten Intensitätsdifferenz kann mit dem Parameter `-qO 1` eingeschaltet werden.

Der Parameter `-LTPM n` gibt die Tiefe des Langzeitspeichers an. Die Standardeinstellung `-LTPM 1` verwendet die konventionelle Bewegungskompensation anhand des zuletzt decodierten Bildes. Mit der Einstellung `-LTPM 10` werden die zehn vorangegangenen decodierten Bilder zur Bewegungskompensation herangezogen.

Die DCT-Codierung wird mit dem Parameter `-DCT n` ein- bzw. ausgeschaltet. Für die DCT-Codierung ist die Angabe des Verzeichnises erforderlich in dem die Codetabellen zur DCT-Codierung enthalten sind. Mit `-c /SUN_HOME/mit/cids/H263/tab/` erfolgt ein entsprechender Verweis. Wird die DCT-Codierung verwendet, sollte auf jeden Fall das Header-Flag gesetzt sein. Mit `-Hd 0` kann jedoch der Header innerhalb der Codesequenz abgeschaltet werden.

A.1.2 Decoder

Das Programm *QtDecode* decodiert eine gegebene Codesequenz. Der Decoder ist in Lage die rekonstruierte Videosequenz anzuzeigen oder in ein File zu schreiben. Folgende Parameter können angegeben werden:

```
VBS coder version 6.0
(C) 1996 University of Erlangen-Nuremberg
    T. Wiegand, M. Flierl, K. Pieper

QtDecode [-<help>hio<orig><info>CK<ff><RFF>nmabsfkDzMdO<sc>
          <um><hp><LTPM><DCT>c<Hd>]
    -help        verbose help
    -h           help
    -i    [name] code file name
                 [bits.itd]
    -o    [name] reconstructed sequence file name
                 []
    -orig [name] original sequence file name
                 []
    -info [name] information file name
    -C    [name] codebook file name
                 [/POOL_HOME/stud/pieper/quadtree/Codec/tab/itd1.0.cbk]
    -K    [name] mask file name
```

```
                [/POOL_HOME/stud/pieper/quadtree/Codec/tab/mask.tab]
-ff    [name] first frame file name
-RFF   [num]  bits used for first frame          [0]
-n     [num]  input image width                  [176]
-m     [num]  input image height                 [144]
-a     [num]  start                              [0]
-b     [num]  end                                [99]
-s     [num]  frame step                         [1]
-f     [num]  frequency                          [30]
-k     [num]  image file format                  [0]
                0: LNT format
                1: catenate sequence file format
                2: khoros format
-D     [num]  output mode                        [2]
                0: write imposed reconstruction file
                1: write reconstruction file
                2: display reconstruction
                3: display imposed reconstruction
-z     [num]  zoom factor                        [1]
-M     [num]  macroblock size                    [16]
-d     [0/1]  differencing                       [1]
-O     [0/1]  overlapping                        [1]
-sc    [num]  MV scale factor                    [1]
-um    [0/1]  unrestricted motion flag           [1]
-hp    [num]  half pel accuracy                  [1]
-LTPM  [num]  long time prediction memory        [1]
-DCT   [0/1]  DCT                                [1]
-c     [name] DCT data path
              [/SUN_HOME/mit/cids/H263/tab/]
-Hd    [0/1]  code file header flag              [1]
```

Die Parameter `-um n`, `-hp n`, `-LTPM n`, `-c name` und `-Hd n` haben eine identische Bedeutung, wie die des Coders.

A.2 Programme zur Optimierung des Codebuchs

Der Entwurf des Codecs ist in einen iterativen Algorithmus eingebettet. Hier werden den Repräsentanten innerhalb der Codebuchtabellen Huffman-Codeworte zugewiesen.

Zunächst wird ein Ausgangscodebuch erzeugt. Danach wird die Häufigkeitsverteilung der einzelnen Codeworte anhand eines Satzes von Trainingssequenzen bestimmt. Aus dem bisherigen Codebuch und der Häufigkeitsverteilung seiner Codeworte wird ein neues Codebuch berechnet. Die Wiederholung der letzten beiden Schritte führt zu einem verbesserten Codebuch.

A.2.1 Erzeugung des Ausgangscodebuchs

Mit dem Programm *InitialCB* kann ein Ausgangscodebuch erzeugt werden. Den jeweiligen Segmentformen, örtlichen Verschiebungen und Intensitätsdifferenzen werden Codeworte fester Länge zugeordnet. Folgende Parameter sind erlaubt:

```
InitialCB [-help h C Y U V qQ hp DCT dL]
    -help        verbose help
    -h           help
    -C    [name] codebook file name
                 [uniform.cbk]
    -qQ   [no no no ..] list of intensity quantization steps
                 [3 4 4 4 ]
    -hp   [num]  half pel accuracy          [1]
    -DCT  [0/1]  DCT                        [1]
    -dL   [num]  level for DCT              [1]
    -Y    [no no no ..] list of Y codebook parameters of size 'no'
                 [12 162 4356 0 162 4356 77 16 4356 97 0 4356 93 ]
    -U    [no no no ..] list of U codebook parameters of size 'no'
                 [9 16 0 0 16 289 41 0 289 53 ]
    -V    [no no no ..] list of V codebook parameters of size 'no'
                 [9 16 0 0 16 289 41 0 289 53 ]
```

Für jede Komponente kann die Schrittweite der Intensitätsdifferenzen separat angegeben werden. Mit `-qQ 3 4 4 4` wird den Intensitätsdifferenzen aller Komponenten eine Schrittweite von vier zugewiesen. Die erste Ziffer gibt die Länge des folgenden Vektors an.

Der Parameter `-hp n` gibt an, für welche Komponenten Tabellen der örtlichen Verschiebung mit Halb-Pixel-Positionen generiert werden sollen. Die Verwendung dieses Parameters erfolgt in der gleichen Weise, wie beim Coder beschrieben.

Mit dem Parameter `-DCT n` wird bestimmt, ob das Codebuch für eine DCT-Codierung vorgesehen ist. Soll das Codebuch für eine DCT-Codierung verwendet werden, ist noch anzugeben auf welchen Level die DCT erfolgt. Im Codec ist die DCT-Codierung für 8×8-Blöcke implementiert. Im Falle einer Makroblockgröße von 16×16 wird mit `-dL 1` die DCT auf Level 1 festgelegt.

Aufgrund der Erweiterung des Codecs um die Bewegungskompensation mit Halb-Pel-Genauigkeit, die Langzeitprädiktion und die DCT-Codierung wird die Anzahl der Dimensionen der Code-Tabellen für die Segmentform von 1 auf 3 und für die örtliche Verschiebung von 2 auf 4 erhöht.

```
[Title] Shape-Code [Size] 162 [Event Dimension] 3
0 0 0 0000000
0 1 0 0000001
0 2 0 0000010
0 3 0 0000011
```

```
0 4 0 0000100
0 5 0 0000101
0 6 0 0000110
0 7 0 0000111
0 8 0 0001000
0 9 0 0001001
1 0 0 0001010
1 1 0 0001011
...
15 9 0 11111100
0 0 1 11111101
0 0 2 11111110
```

Die erste Spalte der Shape-Tabelle gibt die Segmentform wieder. Die zweite Spalte enthält für die jeweilige Segmentform den Zeitparameter. Die dritte Spalte bestimmt den Blocktyp. Hier steht null für einen Q-Block, eins für einen P-Block und zwei für einen I-Block.

```
[Title] Spatial-Displacement-Code [Size] 4356 [Event Dimension] 4
-16 -16 0 0 000000000000
-16 -16 0 1 000000000001
-16 -16 1 0 000000000010
-16 -16 1 1 000000000011
-16 -15 0 0 000000000100
...
```

Die ersten beiden Spalten der Tabelle für die örtliche Verschiebung enthalten den x- bzw. den y-Parameter des Bewegungsvektors bezogen auf Pixelpositionen. Spalte 3 zeigt an, ob eine Verschiebung um eine halbe Pixelposition in x-Richtung vorliegt. Für Spalte 4 gilt das Gleiche jedoch auf die y-Richtung bezogen.

A.2.2 Häufigkeitsverteilung

Die Häufigkeitsverteilung der einzelnen Codeworte die für eine Videosequenz bei verschiedenen λ-Werten gewählt werden, kann mit dem Programm *Histogram* bestimmt werden.

```
Histogram [-<help>hi<ff>ot<info>CK<RFF>nmabskSMdOGlH
           <um><hp><hpr><qO><LTPM><DCT>c<Hd>]
    -help        verbose help
    -h           help
    -i    [name] original sequence file name
                 []
    -ff   [name] first frame file name
    -o    [name] code file name
                 [bits.itd]
    -t    [name] trace file name
```

```
-info [name] information file name
             [histogram.info]
-C    [name] codebook file name
             [/POOL_HOME/stud/pieper/quadtree/Codec/tab/itd1.0.cbk]
-K    [name] mask file name
             [/POOL_HOME/stud/pieper/quadtree/Codec/tab/mask.tab]
-RFF  [num]  bits for first frame                [0 bits]
-n    [num]  input image width                   [176]
-m    [num]  input image height                  [144]
-a    [num]  start                               [0]
-b    [num]  end                                 [99]
-s    [num]  frame step                          [1]
-k    [num]  image file format                   [0]
               0: LNT format
               1: catenate sequence file format
               2: khoros format
-S    [num]  code buffer size (byte)             [100]
-M    [num]  macroblock size                     [16]
-d    [0/1]  differencing                        [1]
-O    [0/1]  overlapping                         [1]
-G           use growing algorithm
-um   [0/1]  unrestricted motion flag            [1]
-hp   [num]  half pel accuracy                   [1]
-hpr  [0/1]  half-pel refinement flag            [0]
-qO   [0/1]  optimal intensity quantization flag [0]
-LTPM [num]  long time prediction memory [0]
-l    [no num num ..]   list of lambda values of size 'no'
             [1 100.000000 ]
-H    [name] histogram file name
             [a.hst]
-DCT  [0/1]  DCT                                 [1]
-c    [name] DCT data path
             [/SUN_HOME/mit/cids/H263/tab/]
-Hd   [0/1]  code file header flag               [1]
```

Das Programm *Histogram* ist von *QtEncode* abgeleitet und läßt sich daher fast identisch bedienen.

A.2.3 Erzeugung des Codebuchs aus der Häufigkeitsverteilung

Die Häufigkeitsverteilungen der Codeworte, die anhand eines Satzes von Trainingssequenzen erstellt wurden, können mit dem Programm *UpdateCodebook* summiert werden. Hieraus wird dann ein neues Codebuch erstellt.

```
UpdateCodebook [-help h info C NewC K H qO]
```

```
    -help        verbose help
    -h           help
    -info [name] information file name
                 [update.info]
    -C    [name] codebook file name
                 [/POOL_HOME/stud/pieper/quadtree/Codec/tab/itd1.0.cbk]
    -NewC [name] file name of new codebook
                 [new.cbk]
    -K    [name] mask file name
                 [/POOL_HOME/stud/pieper/quadtree/Codec/tab/m.tab]
    -qO   [0/1]  optimal intensity quantization flag [0]
    -H    [no name name ..] list of histogram file names of size 'no'
                 [1 a.hst ]
    -S    [name] accumulated histogram file name
                 [sum.hst]
```

Wird das Codebuchtraining mit der Option der Quantisierung der Intensitätsdifferenzen bezüglich minimaler Lagrange-Kosten durchgeführt, so muß auch hier der Parameter `-qO 1` angegeben werden. Andernfalls kann ein Abschneiden des Randbereichs der Intensitätsdifferenzverteilung erfolgen.

A.2.4 Sortierung des Codebuchs

Das Programm *SortCodebook* erlaubt die Sortierung der Shape- und Displacement-Tabellen bezüglich der Codewortlängen eines bisher unsortierten Codebuchs.

```
SortCodebook [-help h C NewC qO]
    -help        verbose help
    -h           help
    -C    [name] codebook file name
                 [unsorted.cbk]
    -NewC [name] file name of new codebook
                 [sorted.cbk]
    -qO   [0/1]  optimal intensity quantization flag [0]
```

Für den Parameter `-qO` gilt dasselbe, wie im Falle des Programms *UpdateCodebook*.

A.3 Analysetool

Mit dem Analysetool *Analyse* (vorher *Freq*) läßt sich eine Codesequenz nach bestimmten Aspekten hin untersuchen.

```
VBS coder version 6.0
(C) 1996 University of Erlangen-Nuremberg
    T. Wiegand, M. Flierl, K. Pieper

Analyse [-<help>hio<orig><info>CK<ff><RFF>nmabsfkDzMdO<sc>
         <um><hp><LTPM><DCT>c<Hd>]
    -help        verbose help
    -h           help
    -i    [name] code file name
                 [bits.itd]
    -o    [name] reconstructed sequence file name
                 []
    -orig [name] original sequence file name
                 []
    -info [name] information file name
    -C    [name] codebook file name
                 [/POOL_HOME/stud/pieper/quadtree/Codec/tab/itd1.0.cbk]
    -K    [name] mask file name
                 [/POOL_HOME/stud/pieper/quadtree/Codec/tab/mask.tab]
    -H    [name] histogram file name
                 [out.hst]
    -ff   [name] first frame file name
    -RFF  [num]  bits used for first frame          [0]
    -n    [num]  input image width                  [176]
    -m    [num]  input image height                 [144]
    -a    [num]  start                              [0]
    -b    [num]  end                                [99]
    -s    [num]  frame step                         [1]
    -f    [num]  frequency                          [25]
    -k    [num]  image file format                  [0]
                   0: LNT format
                   1: catenate sequence file format
                   2: khoros format
    -D    [num]  output mode                        [2]
                   0: write imposed reconstruction file
                   1: write reconstruction file
                   2: display reconstruction
                   3: display imposed reconstruction
    -z    [num]  zoom factor                        [1]
    -M    [num]  macroblock size                    [16]
    -d    [0/1]  differencing                       [1]
    -O    [0/1]  overlapping                        [1]
    -sc   [num]  MV scale factor                    [1]
    -um   [0/1]  unrestricted motion flag           [1]
    -hp   [num]  half pel accuracy                  [1]
    -LTPM [num]  long time prediction memory        [1]
    -DCT  [0/1]  DCT                                [1]
```

```
-c     [name] DCT data path
              [/SUN_HOME/mit/cids/H263/tab/]
-Hd    [0/1]  code file header flag              [1]
```

Das Programm *Analyse* wurde vom Decoder abgeleitet. Die Bedeutung der Parameter ist genau dieselbe, wie die des Programms *QtDecode*. Am Ende des Info-Files werden jedoch weitere summarische Daten angefügt.

```
FF-Rate:        0.000000
Partial Rates for Encoded Frames:
Skip-Rate       9900.000000
Skip:   8927341.000000  747264  1377280.000000  186816  508120.000000   186816
Partial Rates (Component) (Level) (Shape) (S-XY-Q-Rate) (SSE) (P-Rate):
C-L-S: 0 0 0    5433    14551   0       9041564.000000  463616
C-L-S: 0 0 1    1725    2303    0       1476016.000000  66240
C-L-S: 0 0 2    1690    1977    0       1274484.000000  64896
C-L-S: 0 0 3    832     1466    0       832378.000000   26624
C-L-S: 0 0 4    1600    2087    0       1401296.000000  61440
C-L-S: 0 0 5    2160    4514    0       2011604.000000  69120
C-L-S: 0 0 6    312     363     0       291646.000000   6656
C-L-S: 0 0 7    0       0       0       0.000000        0
C-L-S: 0 0 8    1640    2020    0       1345522.000000  62976
C-L-S: 0 0 9    273     312     0       203970.000000   4992
C-L-S: 0 0 10   2170    2955    0       1217333.000000  55552
C-L-S: 0 0 11   0       0       0       0.000000        0
C-L-S: 0 0 12   736     1198    0       782435.000000   23552
C-L-S: 0 0 13   0       0       0       0.000000        0
C-L-S: 0 0 14   0       0       0       0.000000        0
C-L-S: 0 0 15   2382    0       0       0.000000        0
C-L-S: 0 1 0    11842   54226   18187   13475841.000000 378944
C-L-S: 0 1 1    3876    7946    2769    2671107.000000  46512
C-L-S: 0 1 2    3148    6351    1936    2105999.000000  37776
...
```

Die erste Zeile gibt die Größe eines eventuell am Anfang geladenen Bildes in bit an. In der dritten Zeile wird die Anzahl der COD-Bits angegeben. Die nächste Zeile stellt komponentenweise, in der Reihenfolge YUV, den aufsummierten quadratischen Fehler (SSE) und die Anzahl der beteiligten Pixel für den Skip-Fall (COD = 0) dar. Die folgenden Zeilen geben für jeden Knoten des Quadtree die Anzahl der benötigten Bits und den SSE an. Eine Zeile gliedert sich in die Bestandteile Komponente, Level, Segmentform, Bit-Rate jeweils für Segmentform, örtliche Verschiebung und Intensitätsdifferenz, SSE der Segmentform und die Anzahl der beteiligten Pixel.

Anhang B

Syntax der Codesequenz

Die Codesequenz des Quadtree-Videocodecs ist hierarchisch in Bild-, Makroblock- und Quadtree-Code gegliedert. Mit Einführung der DCT-Codierung wird auch ein optionaler Sequenz- und Bildheader zur Verfügung gestellt.

B.1 Sequenzebene

Auf Sequenzebene ergibt sich die in Abb. B.1 gezeigte Struktur, bestehend aus Sequenzheader, der Codesequenz der Bilder und dem Codewort für das Sequenzende (EOS).

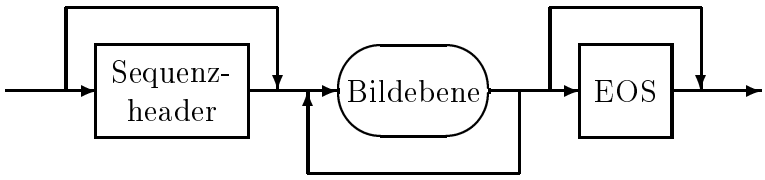

Abbildung B.1: *Struktur der Codesequenz auf Sequenzebene*

Der Sequenzheader enthält Informationen zum Format der Videosequenz und den verwendeten Modi und hat eine Länge von 96 bit. Folgende Informationen sind enthalten:

Bit 1-16	Bildhöhe
Bit 17-32	Bildbreite
Bit 33-64	Bildfrequenz als Festkommazahl im Format XX.XX
Bit 65-72	Bildschrittweite
Bit 73-80	Makroblockgröße
Bit 81-88	Ringbuffergröße
Bit 89	Unrestricted-Motion-Mode, „0“ aus, „1“ ein
Bit 90-92	Halb-Pel-Genauigkeit komponentenweise in der Reihenfolge YUV, „0“ aus, „1“ ein
Bit 93	DCT-Modus, „0“ aus, „1“ ein
Bit 94-96	000 (frei)

Das EOS-Codewort zeigt das Ende der Codesequenz an und besteht aus 22 bit. Der Wert des Codeworts ist 00000000 00000000 111111.

B.2 Bildebene

Die Bildebene der Codesequenz setzt sich aus Bildheader, der Makroblockebene und den Füllbits (STUF) zusammen. Der Bildheader, bestehend aus Bildstartcode (PSC), Zeitreferenz (TR), Bildtyp (PTYPE) und Quantisierungsstufe (QUANT), lehnt sich an den H.263-Standard [8] an. Der Bildstartcode soll an Bytepositionen beginnen. Dies wird durch Anhängen von 0 bis 7 Füllbits (STUF) mit dem Wert „0“ im Anschluß an die Makroblockebene erreicht.

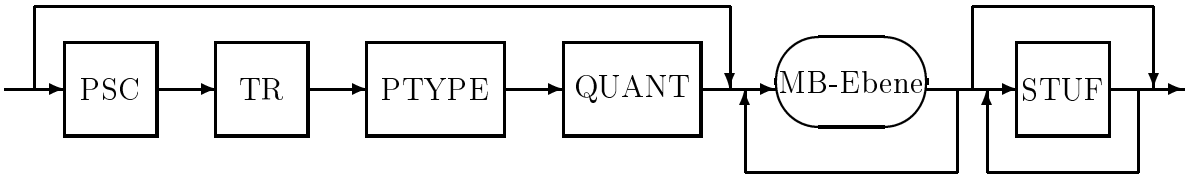

Abbildung B.2: *Struktur der Codesequenz auf Bildebene*

Der Bildstartcode ist ein 22 bit langes Codewort mit dem Wert 00000000 00000000 100000. Die Zeitreferenz ist ein 8-bittiges Codewort, welches hier jedoch nicht unterstützt wird. Das Codewort für den Bildtyp setzt sich aus

Bit 1	Bildcodiermodus: „0“ I-Bild, „1“ P-Bild
Bit 2	Prädiktions-Modus, „0“ aus, „1“ ein
Bit 3	Überlappungsmodus, „0“ aus, „1“ ein
Bit 4-5	00 (frei)

zusammen. QUANT ist ein 5 bit langes Codewort, welches den im DCT-Modus verwendeten Quantisierer angibt. Für QUANT sind Werte zwischen 1 bis 31 erlaubt. Die Quantisierungsstufenhöhe ergibt sich hieraus durch Multiplikation mit zwei und nimmt einen geradzahligen Wert im Bereich von 2 bis 62 an.

B.3 Makroblockebene

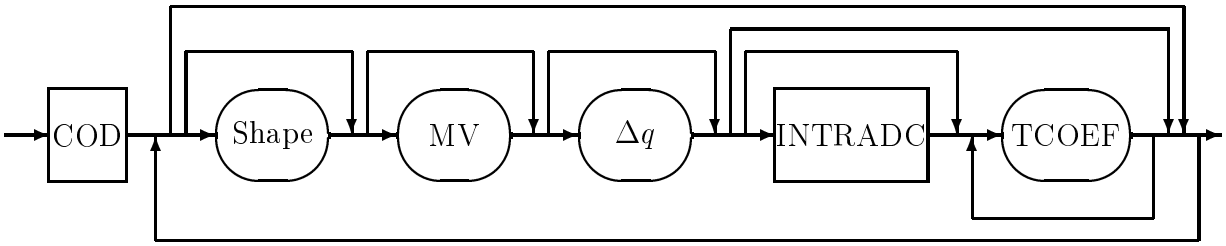

Abbildung B.3: *Struktur der Codesequenz auf Makroblockebene*

Die Makroblockebene beginnt mit dem COD-bit. Ist dieses Null, wird kein Quadtree-Code gesendet. Der Decoder kopiert die Bilddaten des vorangegangen Bildes. Ist das COD-bit gesetzt, folgt der Quadtree-Code jeweils für die Y-, U- und V-Komponente. Der

Quadtree-Code richtet sich nach der Depth-First-Regel, d.h. jeder Zweig des aktuellen Knotens wird vor dem nächsten Zweig des aktuellen Knotens geschrieben. Erfolgt keine DCT-Codierung, wird für jeden Knoten ein Codewort für die Segmentform (Shape), die örtliche Verschiebung (MV) und die Intensitätsdifferenz (Δq) vorgesehen. Es existieren jedoch vier Ausnahmen von der Regel:

- Die Segmentform 15 (der Knoten hat vier Zweige) besitzt keinen Bewegungsvektor und keine Intensitätsdifferenz. Die Codeworte für örtliche Verschiebung und Intensitätsdifferenz entfallen.
- Auf der obersten Hierarchiestufe des Quadtrees wird für das Segment immer eine Intensitätsdifferenz von Null angenommen. Das Codewort für die Intensitätsdifferenz entfällt.
- Auf der letzten Hierarchiestufe des Quadtree sind die Knoten immer Endknoten (Segmentform 0). Das Codewort für die Segmentform entfällt.
- Den Farbkomponenten wird der Bewegungsvektor auf der obersten Hierarchiestufe des Quadtree von der Helligkeitskomponente vererbt. Das Codewort für die örtliche Verschiebung entfällt.

Wird ein Block DCT-codiert, sind noch zwei weitere Fälle zu unterscheiden.

- Beim I-Block entfällt das Codewort für die örtliche Verschiebung und die Intensitätsdifferenz. Dem Codewort für die Segmentform folgt das Codewort für den DC-Koeffizienten (INTRADC) und anschließend die Codewörter der AC-Koeffizienten (TCOEF).
- Beim P-Block entfällt das Codewort für die Intensitätsdifferenz. An dessen Stelle treten die Codewörter der AC-Koeffizienten (TCOEF).

Die Codierung der Transformationskoeffizienten richtet sich nach dem H.263-Standart. Der DC-Koeffizient (INTRADC) ist ein Codewort der festen Länge 8. Es repräsentiert den Wertebereich von 8 bis 2032 mit der Schrittweite 8. Die AC-Koeffizienten werden nach einer Tabelle lauflängencodiert und durch Codeworte variabler Länge (TCOEF) repräsentiert.

Anhang C

Simulationsbedingungen

Das Codebuchtraining und sämtliche Untersuchungen dieser Arbeit wurden anhand von QCIF-Videosequenzen bei einer Bildrate von $7.5Hz$ durchgeführt. Die Laufzeit der Videosequenzen betrug jeweils $10s$. Für das Codebuchtraining wurden 18 Videosequenzen verwendet. Die im Rahmen dieser Arbeit erzeugten Codebücher wurden bei $\lambda = 100$ trainiert und weisen für die Bewegungsschätzung einen Suchbereich von $[-16, 16]$ auf. Die verwendeten Codebücher haben alle die gleiche Baumtiefe. Die Anzahl der Level beträgt für die Y-Komponente 4 und für die U- und V-Komponente jeweils 3.

Die Untersuchungsergebnisse werden in Form von PSNR-Messungen [1] dargestellt. Die PSNR-Werte

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{MSE_C} \tag{C.1}$$

der Y-Komponente ($D = C$) werden über alle Bilder gemittelt und über der Gesamtbitrate dargestellt.

Sofern nichts anderes ausgesagt wird, wird das erste Bild der Testsequenzen „Car Phone“ und „Salesman“ im Intra-Mode des TMN 1.6-Coders codiert. Die weiteren Bilder der Testsequenzen werden bei konstantem λ codiert.

[1] PSNR = **P**eak **S**ignal to **N**oise **R**atio

Literaturverzeichnis

[1] ISO/IEC JTC1/SC29/WG11 MPEG 96/0553, *Iterated Systems MPEG-4 Video Submission Technical Description*, Jan. 1996.

[2] M. Flierl, *Untersuchung von Algorithmen zur Bestimmung von Quadtree-Strukturen bei der Videocodierung mit Blöcken variabler Größe*, Studienarbeit. Universität Erlangen–Nürnberg, Nachrichtentechnik, 1997.

[3] P. A. Chou, T. Lookabaugh und R. M. Gray, "Entropy-constrained vector quantization", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 1, pp. 31–42, Jan. 1989.

[4] P. A. Chou, T. Lookabaugh und R. M. Gray, "Optimal pruning with application to tree-structured source coding and modeling", *IEEE Transactions on Information Theory*, pp. 299–315, März 1989.

[5] A. Gersho und R. M. Gray, *Vector quantization and signal compression*, Kluwer Academic Publishers, Boston, 1992.

[6] B. Girod, "Motion compensation: Visual aspects, accuracy, and fundamental limits", in *Motion Analysis and Image Sequence Processing*. Kluwer Academic Publishers, 1993.

[7] B. Girod, "Motion-compensating prediction with fractional-pel accuracy", *IEEE Transactions on Communications*, vol. 41, no. 4, pp. 604–612, Apr. 1993.

[8] ITU-T, *Video Coding for Low Bitrate Communication: Draft Recommendation H.263*, Okt. 1995.

[9] J. Lee, "Optimal quadtree for variable block size motion estimation", in *Proceeding of the IEEE International Conference on Image Processing*, Washington D.C., Okt. 1995, vol. II, pp. 480–483.

[10] K. Rose und S.K. Mitra M. Lightstone, "Locally optimal codebook design for quadtree-based vector quantization", in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1995, pp. 2497–2482.

[11] J. Ohm, *Digitale Bildcodierung*, Springer Verlag, 1995.

[12] G.J. Sullivan M.T. Orchad, "Overlapped block motion compesation: An estimation-theoretic approach", *IEEE Transactions on Image Processing*, vol. 5, no. 3, pp. 693–699, Sep. 1994.

[13] E. A. Riskin und R. M. Gray, "A greedy tree growing algorithm for the design of variable rate veqtor quantizers", *IEEE Transactions on Signal Processing*, vol. 39, no. 11, pp. 2500–2507, Nov. 1991.

[14] P. Strobach, "Tree-structured scene adaptive coder", *IEEE Transactions on Communications*, vol. 38, no. 4, pp. 477–486, Apr. 1991.

[15] G.J. Sullivan und R.L. Baker, "Efficient quadtree coding of images and video", *IEEE Transactions on Image Processing*, vol. 3, pp. 327–331, Mai 1991.

[16] A. M. Tekalp, *Digital video processing*, Prentice Hall, London, 1995.

[17] X. Zhang, *Statistische Untersuchung blockbasierter Langzeitprädiktion*, Diplomarbeit. Universität Erlangen–Nürnberg, Nachrichtentechnik, 1996.