Bilddatenkompression

Semesterarbeit
der Informatik
an der Fachhochschule Zürich

vorgelegt von

Samuel Benz

Leiter der Arbeit: Dr. N. Ari
Fachhochschule Zürich

Dieses Dokument ist in LaTeX geschrieben,

Zürich, 17.09.2002 Samuel Benz

Inhaltsverzeichnis

Kapitel 1

Einführung

Die digitale Kommunikationstechnik hat in den letzten Jahrzehnten eine enorme Entwicklung erfahren und ist dabei, nach und nach jene Verfahren zu verdrängen, die zeit- und wertekontinuierliche Signale verarbeiten. Digitales Telefonieren ist mitlerweile der Standart, digitales Fernsehen wird bereits Praktiziert. Für den zunehmenden Einsatz von Digitaltechnik gibt es gute Gründe. Erstens ist dadurch meist eine bessere Qualität der Signale realisierbar, vor allem weil die digitale Übertragung einen besseren Schutz gegen Übertragungsfehler ermöglicht. Zweitens ist die Verarbeitung digitaler Signale oft auch einfacher als die von analogen Signalen. Diese Vorteile müssen allerdings durch einen Nachteil erkauft werden: digitale Signalquellen produzieren sehr grosse Datenmengen.

Dies ist der Grund, warum Informationstechnologien zur effizienten Datenkompression für die Speicherung und Übertragung von Signalen immer wichtiger werden. Insbesondere die Bild- und Video-Codierung hat eine wachsende Bedeutung, da hier zwei- und sogar mehrdimensionale Signale verarbeitet werden müssen. Typische Anwendungen der Datenkompression sind: Archivierung von Daten jeglicher Art, digitales Fernsehen, Videoaufzeichnung, Bildtelefonie, Videokonferenzen, digitale Fotografie u.v.a.m.[1].

In dieser Semesterarbeit möchte ich einen Einblick in die Theorie der Bilddatenkomression geben. Ich werde dies anhand vom JPEG Verfahren (Kapitel: 2.1) erläutern.

Kapitel 2

JPEG Kompression

2.1 Einführung

Seit 1989 wurde das JPEG Verfahren zur Komprimierung von unbewegten Bildern entwickelt und 1990 standartisiert. Er ist bekannt geworden nach der Gruppe, die ihn verabschiedet hat. Die *Joint Picture Expert Group*, kurz JPEG, entsand aus einer Zusammenarbeit con CCITT[1] und ISO[2]. Das Ziel diser Gruppe war es einen Algorithmus zur Komprimierung von "natürlichen" Grauton- oder Farbbildern zu entwickeln. Besondere Rücksicht wure dabei auf eine hohe Kompressionsrate und hohe Geschwindigkeit zum Codieren und Decodieren gelegt. Schnell stellte sich jedoch heraus, dass es nicht möglich war einen einzigen Algorithmus für alle Anforderungen zu finden. So wurde das Verfahren in vier unterschiedliche Modi gegliedert:

Sequentiell mode: Ein einzelner Durchgang durch das Bild von links oben nach rechts unten decodiert das Bild. Dieser Modus ist für die meisten Anwendungen gut geeignet, liefert die besten Kompressionsraten und ist am leichtesten zu implementieren.

Progressiv mode: Das Bild wird in mehreren Durchgängern codiert bzw. decodiert. Dabei wird das Bild von Durchgang zu Durchgang schärfer. Das Einsatzgebiet für dieses Verfahren ist beispielsweise die Datenübertragung von Bildern. Sobald das Bild einen ausreichende Schärfe erreicht hat, kann die Übertragung abgebrochen werden.

Hirachical mode: Das Bild wird erst in einer geringen Auflösung gespeichert und dann in der vollen Auflösung. Das Bild mit der geringen Auflösung kann bedeutend schneller dekodiert werden, ist somit gut als schnelles Preview des Bildes geeignet. Dies ist sinnvoll für den Einsatz in Bilddatenbanken, bei denen das Bild in geringer Auflösung als Entscheidungshilfe gilt.

[1] Consulative Committee on International Telegraph and Telephone
[2] International Standards Organisation

Lossless mode: In diesem Modus wird, im Gegensatz zu den anderen Modi, verlustfrei codiert und decodiert. Jedes Bit wird genau so wiederhergestellt wie es codiert wurde. Dies verlustfreie Codierung wird benötigt um Bilder zu komprimieren, die keinerlei Abweichungen zulassen oder nicht von Menschen ausgewertet werden. Dieser Modus bietet verständlicherweise eine geringe Kompressionsrate, als die anderen Modi, da bei verlustfreier Kompression der Informationsgehalt nicht reduziert werden kann.

Bis auf den letzten Modus, die verlustfreie Kodierung[3], sind alle Modi verlustbehaftet[4]. Es findet während der Kodierung folglich nicht nur eine Kompression, sondern auch eine Reduktion der Daten statt. Da der Informationsgehalt der Daten verringert wird, kann das Orginalbild nicht identisch wiederhergestellt werden. Durch Codierung kann sich daher die Qualität des Bildes verschlechtern. Diese ist jedoch in der Regel für das menschliche Auge nur wenig zu erkennen.

Für die meisten Anwendungen ist der sequentielle Modus gut geeignet. Die anderen Modi sind nur für spezielle Einsatzgebiete gedacht. Sie basieren aber grösstenteils auf dem gleichen Grundprinzip. Daher wird im weiteren nur der sequentielle Modus betrachtet.

[3]Kompressionsrate ca. 1:2

[4]Kompresisonsrate bis ca. 1:20 möglich

2.2 Funktionsweise

Die Funktionsweise von JPEG kann Grundsätzlich in vier Teile gegliedert werden. Die ersten drei sind Verlustbehaftet *(lossy compression)*, die Daten werden also reduziert. Der Letzte ist eine verlustlose Kompression *(lossless compression)*.

- Farbraumtransformation
- DCT *Discret Cosinus Transformation*
- Quantisierung
- Hufmann Codierung

Um die Testbilder (Abbildung: A.1,A.2,A.3) in Matlab zu laden verwenden sie das Matlab Programm *initpic.m (C.1)*. Dies gibt drei Vektoren von 300x400x3, was der Höhe, Länge und den Werten R,G und B entspricht.

2.2.1 Farbraumtransformation

Das menschliche Auge benutzt zur Wahrnehmung von Licht zwei veschiedene Rezeptoren. Zum einen werden Stäbchen genutzt, die es erlauben Helligkeit zu sehen und schon bei geringer Lichtstärke einsetzbar sind. Zum anderen besitzt das Auge Zapfen, die nur bei guter Beleuchtung das Farbsehen erlauben. Die Zapfen sind unterschiedlich sensibel für die Farben: Rot, Grün, Blau. Mit diesen drei Grundfarben lassen sich additiv alle Farben des RGB-Farbraums darstellen. Bei der Entwicklung des Fernsehsystems führte man einige Tests durch und fand heraus, dass sich das menschliche Helligkeitsempfinden zu

$$Y = 0.3 * R + 0.59 * G + 0.11 * B \tag{2.1}$$

ergibt. Dies weil unser Auge am meisten auf Grün reagierende Zapfen besitzt. So entstand das Y-Signal[5] welches bei Scharz-Weiss-TV übertagen wird. So kann ein Farbbild sehr einfach mit der Formel (2.1) in Graustufen gewandelt werden (*Matlab: togray.m C.2*). Um die Helligkeitsverteilung eins Bildes zu zeigen habe ich das Matlab Programm *histo.m (C.3)* geschrieben (*Histogramme: Abbildung A.4, A.5*). Bei der Einführung vom Farbfernseher musste das neue System kompatibel zum Alten sein. Man übertrug also neu nicht alle RGB-Werte sondern nur Farbdifferenzsignale zum Helligkeitssignal.

$$\begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} 0.493 * (B - Y) \\ 0.877 * (R - Y) \end{pmatrix} \tag{2.2}$$

[5] Y-Signal = Helligkeitssignal

Bei diesem sogennanten YUV-Farbraum spart man das G-Y Signal, da es durch Addition der anderen wieder rekonstruiert werden kann. Die Übertragungsbandbreite kann somit reduziert werden.

Bei JPEG werden die selben Ansätze verwendet. Der Farbraum heisst hier allerdings YCbCr (2.3). Das Prinzip ist das gleiche, jedoch werden die Farbkomponenten von 0...255 nach -128...127 verschoben.

Die RGB Werte eines Bildes mit der Formel (2.3) in den YCbCr Farbraum transformieren:

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.3}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{pmatrix} * \begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} \tag{2.4}$$

Matlab: YCbCr.m (C.4) und *Matlab: RGB.m (C.5).*

Das Trennen von Helligkeit- und Farbinformation bietet zusätzlich den Vorteil, dass ein Farb-Unterabtastung einfach möglich ist. Dies bedeutet, dass man eine andere Helligkeitsauflösung als Farbauflösung verwendet um Daten zu reduzieren. Das menschliche Auge merkt dies kaum, da eine Farbtonschwankung im Gegensatz zu einer Helligkeitsschwankung kaum wahrgenommen wird.

Die Farb-Unterabtastung *(Subsampling)* kann entweder auf aueinanderfolgende Bildpunkte oder auf definierte Blöcke angewandt werden. Zum Besispiel: 1 Chroma-Wert für 2 Y-Bildpunkte, 1 Chroma-Wert für 4 Y-Bildpunkte oder 2 Chroma-Werte für ein 2x2 Y-Bildblock (*Matlab: subsamp.m (C.6)*).

Um dies zu demonstrieren[6] habe ich zusätzlich ein Matlab Programm *tocolor.m (C.7)* geschrieben, welches Y- und Chroma-Werte in Blockform unterabtastet (*2x2: Abbildung A.6; 4x4: Abbildung A.7; 8x8: Abbildung A.8*).

[6]Da nicht ich nicht alle Teile des JPEG-Verfahrens, insbesondere die Binäre-File-Codierung, programiert habe, wird dieses Programm zur Darstellung verwendet.

2.2.2 DCT *Discret Cosinus Transformation*

Nach der Farbraumtransformation und Indexverschiebung[7] der Chroma Werte wird die Diskrete Kosinus Transformation *(Forward DCT)* angewendet. Die DCT ist änlich der Fourier Transformation ohne Sinus Werte. Dies hat der Vorteil, dass die Koeffizienten reell Wertig sind. Dazu wird das Bild in kleine Teilbereiche der Grösse 8x8 zerlegt (*Matlab: block8.m C.8*). Der 8x8 Block wird als ein diskretes Signal mit 64 Werten aufgefasst. Diese Werte sind abhängig von zwei räumlichen Dimensionen x und y. Sie werden in ihr Spektrum transformiert, indem die FDCT einen Basiswechsel auf eine Basis von 64 orthogonalen, diskreten Signalen durchführt. Die Koeffizienten für diese Basissignale bilden die Ausgabewerte.

Der Wert der horizontalen und der vertikalen räumlichen Frequenz von 0 wird als der Gleichstromkoeffizient *(direct current, DC)* bezeichnet, die restlichen 63 Werte als die Wechselstromkoeffizienten *(alternating current, AC.* Der DC Wert ist der durchschnittliche Wert der 64 Punkte mal 8. In einem "natürlichen" Bild sind scharfe Linien und abrupte Farbwechsel eher selten. Daher ist der Unterschied zwischen zwei benachbarten Punkten meist sehr gering. Nach der FDCT haben die Werte meist folgende Eigenschaften:

- Der DC-Koeffizient ist mit Abstand der grösste Wert.
- Die AC-Koeffizienten werden kleiner und kleiner.
- Die meisten AC-Koeffizienten sind sehr klein, nahe bei 0.

$$F(u,v) = \frac{1}{4} * C(u) * C(v) * \left[\sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) * cos \frac{(2x+1)u\pi}{16} * cos \frac{(2y+1)v\pi}{16} \right] \tag{2.5}$$

$$f(x,y) = \frac{1}{4} * \sum_{x=0}^{7} \sum_{y=0}^{7} C(u) * C(v) * F(u,v) * cos \frac{(2x+1)u\pi}{16} * cos \frac{(2x+1)v\pi}{16} \tag{2.6}$$

mit

$$C(t) = \begin{cases} 0 & t \neq 0 \\ \frac{1}{\sqrt{2}} & sonst \end{cases} \tag{2.7}$$

[7] Die DCT hat geringere Fehler bei Werten zwischen -128. . . 127

Matlab: dctfunk.m (C.9).

Mathematisch gesehen ist sowohl die Transformation (2.5) wie auch die Rücktransformation (2.6) eindeutig. Das heisst, dass die Punkte des Ortsbereiches nach der Abbildung in den Frequenzbereich identisch wiederhergestellt werden können und umgekehrt. Es geht keine Information verloren, also ist dieser Schritt theoretisch verlustfrei.

In der Praxis gelingt dies jedoch nicht. Heutige Rechenanlagen können bedingt durch ihren physikalischen Aufbau und ihre Arbeitsweise manche Funktionen, wie hier den Kosinus, zwar mit grosser Genauigkeit, aber doch nicht hundertprozentig exakt ermitteln. Dadurch ergeben sich Abweischungen der praktisch ermittelten zu den mathematisch korekten Werten. Diese Abweichungen finden sich später als Informationsverlust im Bild wieder.

Um eine bestimmte Bildqualität garantieren zu können, hat die JPEG-Gruppe nicht den Algorithmus zur Berechnung der FDCT und IDCT normiert, sonder festgelegt wie gross der Fehler der berechneten Werte sein darf.

2.2.3 Quantisierung

Die Diskrete Kosinus Transformation hat als Ausgabe 64 Werte geliefert, die in einen DC und 63 AC Koeffizienten unterschieden werden. Die 63 AC-Werte variieren nur wenig, werden immer kleiner und sind nahe bei 0. Diese Werte werden nun quantisiert. Es können dabei mehrere Werte auf eine Zahl abgebildet werden. Es geht dadurch Information verloren. Die Quantisierung jedes der 64 Werte geschieht mittles Dividierung durch einen Wert aus einer Quantisierungsmatrix. Der Wert der Posistion $F(u, v)$ wird dividiert durch den Wert $Q(u, v)$ der Quantisierungstabelle. Das Ergebnis wird auf die nächste Integerzahl gerundet.

$$\tilde{F}(u, v) = Integer - Round\left(\frac{F(u, v)}{Q(u, v)}\right) \tag{2.8}$$

Matlab: quant.m (C.10).

Die Quantisierungstabelle benutz feinere Quantisierung für Koeffizienten der niedrigen Frequenzen und eine gröbere Quantisierung für Koeffizienten der höheren Frequenzen. Deshalb werden die höhren Frequenzen, die meist nahe bei 0 sind, zu 0 quantisiert. Das JPEG-Gremium hat Quantisierungstabellen als Richtlinien vorgegeben (Tabelle B.1,B.2), dies können aber durch eigene ersetzt werden.

Die Umkehrung der Quantisierung ist eine einfache Multiplikation des Wertes mit dem entsprechenden Eintrag in der Qunatisierungstabelle.

$$F(u,v) = \tilde{F}(u,v) * Q(u,v) \tag{2.9}$$

Hierbei wird deutlich, dass der Orginalwert nicht wiederhergestellt werden kann. Abgesehen von Rechenungenauigkeiten bei der DCT ist hier der eigentliche verlustbehaftete Teil des Verfahrens. Gleichzeitig bildet er durch die Reduktion der Daten die Grundlage zur effizienten Komprimierung der Daten. Es ist möglich eine Quantisierungstabelle zu wählen, deren Einträge aus lauter 1 bestehen. Dadurch wird Datenverlust bei der Qunatisierung eliminiert.

2.2.4 Huffman Codierung

Nach der Quantisierung ist der Informationsgehalt der Daten gesunken. Das bietet eine gute Ausgangsbasis zum verlustlosen komprimieren der Daten. Werte, die häufiger auftreten als andere, können mit kurzen Symbolen, seltenere Werte mit längeren Symbolen codiert werden. Um den ersten Durchgang zum Zählen der Häufigkeiten zu sparen, hat das JPEG-Gremium mehrere Tabellen zum Codieren der DC und AC Werte mitgeliefert. Diese Tabellen basieren auf empirischen Erfahrungen und stellen somit keine optimale Codierung dar. Daher sind diese Tabellen kein Standard im JPEG-Verfahren, sondern dienen nur als günstige Default-Tabellen.

Codierung eines 8x8 Blockes

Die DC-Koeffizineten werden unabhängig von den AC-Koeffizienten mit einem prädikativen Verfahren codiert. Man geht davon aus, dass benachbarte Blöcke einen ähnlichen Gleichanteil besitzen. Auf dieser Basis kann ein DC-Wert aus seinem Vorgänger vorausgesagt werden (2.10). Für den ersten Block wird der DC Wert des vorhergehenden Blockes mit 0 initialisiert, das heisst die Differenz ist der DC Wert selbst.

$$diff(k) = DC(k) - DC(k-1) \tag{2.10}$$

Beim Decodieren wird die Differenz ermittelt und durch hinzuaddieren des DC Wertes des vorhergehenden Blockes der aktuelle DC Wert ermittelt (2.11).

$$DC(k) = diff(k) + DC(k-1) \tag{2.11}$$

Der theoretische Wertebereich der Prädiktionsfehler ist sehr gross. Deshalb werden die DIFF-Werte in 12 Kategorien eingeteilt und jeder Kategorie ein Codewort zugewiesen. Tabelle 2.1 enthält die vorgeschlagenen Huffmann-Codes. Es steht dem Anwender jedoch frei, selbst einen Huffmann-Code zu konstruieren um optimale Kompressionsergebnisse zu erzielen. Die verwendeten Codes werden in jedem Fall zum Empfänger übertragen.

Die Kategorienummer legt gleichzeitig fest, wie viele Bits für die Übertragung des DIFF-Wertes folgen. Decodiert der Empfänger zum Beispiel den Huffmann-Code für die Kategorie 2, so weiss er damit, dass weitere 2 Bits kommen, die einen Wert con -3-2,2 und 3 identifizieren.[1]

Kategorie	DIFF-Werte	Code (Lum)	Code (Chrom)
0	0	00	00
1	-1,1	010	01
2	-3,-2,2,3	011	10
3	-7,. . . ,-4,4,. . . 7	011	110
4	-15,. . . ,-8,8,. . . 15	101	1110
5			
6			

Tabelle 2.1: Kategorien für die Codierung von DC-Koeffizineten und Huffman-Code-Beispiele für die Codierung von Luminanz und Chrominanz

Die AC Werte werden getrennt von den DC Werten behandelt. Die hohen Frequenzen werden dann durch die Quantisation weitgehend auf 0 geändert. Um die niedrigen Frequenzen vor den hohen Frequenzen abzuarbeiten werden die AC-Werte in einer Zig-Zag Reihenfolge aufgegriffen.

Alle AC-Koeffizienten ungleich Null werden wie die DC-Koeffizienten in Kategorien unterteilt und Codiert. Nullen werden nicht Codiert, bei ihnen wird die Anzahl Nullen hintereinander *(Lauflänge)* Codiert.

Beispiel eines Codierten Blockes

Der ursprüngliche Datenblock Tabelle 2.2 wird mittels der FDCT (2.5) zu Tabelle 2.3 umgewandelt. Mit Hilfe der Quantisierungstabelle (B.1) ergibt sich Tabelle 2.4 mit vielen 0-Werten.

Zuerst wird der DC-Wert verarbeitet. War der vorhergehende Block zum Beispiel 12, ist nun die Differenz nach (2.10) 3. Aus Tabelle 2.1 liest man, dass 2 Bits benötigt werden um diese 3 zu codieren. Somit ergibt sich die Zwischendarstellung zu (2)(3).

Die Reihe der AC-Werte fängt mit einer 0 an, auf die -2 folgt. Es ergibt sich somit eine Lauflänge von 1 und für die -2 aus der Tabelle 2.1, dass ebenfals 2 Bit benötigt werden. Die Zwischendarstellung ist also (1,2)(-2).

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

Tabelle 2.2: 8x8 Y-Block eines Bildes

235.8	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3
-22.8	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2
-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1
-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3
-0.8	-0.8	1.5	1.8	-0.1	-0.7	0.6	1.3
1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0
-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8
-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4

Tabelle 2.3: 8x8 Block nach FDCT

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabelle 2.4: 8x8 Block nach Quantisierung

Als nächstes folgen mehrere -1, die jeweils die Lauflänge von 0 haben. Damit ergibt sich die Zwischendarstellung (0,1)(-1) mehrmals. Als letztes folgt eine -1 mit der Lauflänge 2, also (2,1)(-1). Danach folgen nur noch Nullen, also eine End-Of-Block Markierung. Die gesammte Zwischendarstellung des Blockes sieht wie folgt aus:

(2)(3),(1,2)(-2),(0,1)(-1),(0,1)(-1),(0,1)(-1),(2,1)(-1),(0,0)

Diese Zwischendarstellung muss in die eigentlichen Huffman-Codes umgewandelt werden. In diesem Beispiel werden Standardtabellen[8] benutzt, die vom JPEG-Gremium empfohlen wurden.

(011)(11),(11011)(01),(00)(0),(00)(0),(00)(0),(11100)(0),(1010)

Dies ergibt für den 8x8 Block:

01111110110100000000011100010 10

Es werden insgesamt 31 Bit benötigt für die 64 Werte, man hat den Block auf unter 0.5 Bit pro Wert komprimiert.

[8] Sind im Buch [1] zu finden.

Anhang A

Testbilder

Abbildung A.1: Meine Freundin Nina mit Hund Matilde (400x300)

Abbildung A.2: Meine Katze Linda (400x300)

Abbildung A.3: Regenwald Kilimanjaro (400x266)

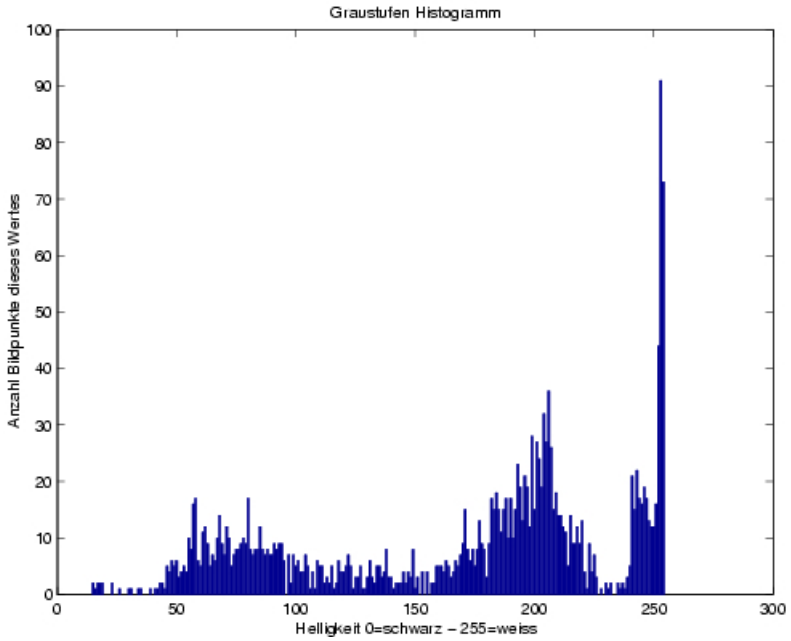

Abbildung A.4: Histogramm von Testbild A.1: Dieses Bild ist Überbelichtet.

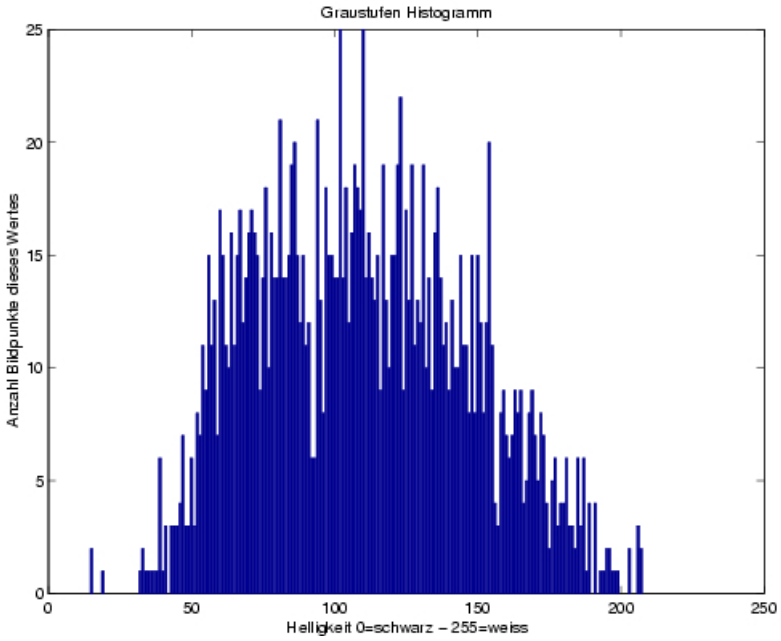

Abbildung A.5: Histogramm von Testbild A.3: Dieses Bild ist optimal ausgeleuchtet.

Abbildung A.6: Testbild A.2 in 2x2 Blöcken

Abbildung A.7: Testbild A.2 in 4x4 Blöcken

Abbildung A.8: Testbild A.2 in 8x8 Blöcken

Anhang B

Quantisierungs-Tabellen

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Tabelle B.1: Qunatisierungstabelle für Luminanz

17	18	24	99	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Tabelle B.2: Qunatisierungstabelle für Chrominanzen

Anhang C

Quelltexte

C.1 initpic.m

```
% init Testbilder

pic1=double(imread('testbild1.png'));
pic2=double(imread('testbild2.png'));
pic3=double(imread('testbild3.png'));

% für andere Tests
yblock=[139 144 149 153 155 155 155 155;...
      144 151 153 156 159 156 156 156;...
      150 155 160 163 156 156 156 156;...
      159 161 162 160 160 159 159 159;...
      159 160 161 162 162 155 155 155;...
      161 161 161 161 160 157 157 157;...
      162 162 161 163 162 157 157 157;...
      162 162 161 161 163 158 158 158];

ydct=[235.8 -1 -12.1 -5.2 2.1 -1.7 -2.7 1.3;...
      -22.8 -17.5 -6.2 -3.2 -2.9 -0.1 0.4 -1.2;...
      -10.9 -9.3 -1.6 1.5 0.2 -0.9 -0.6 -0.1;...
      -7.1 -1.9 0.2 1.5 0.9 -0.1 0.0 0.3;...
      -0.8 -0.8 1.5 1.8 -0.1 -0.7 0.6 1.3;...
      1.8 -0.2 1.6 -0.3 -0.8 1.5 1.0 -1.0;...
      -1.3 -0.4 -0.3 -1.5 -0.5 1.7 1.1 -0.8;...
      -2.6 1.6 -3.8 -1.8 1.9 1.2 -0.6 -0.4];
```

C.2 togray.m

```
function [] = togray(pic,filename)

% Convert pic to gray image
%
%
% Y = togrey(pic,'filename')

% calulate Y Values
vert=length(pic(:,1,1));
horz=length(pic(1,:,1));
Z=1;

for H=1:1:horz
   for V=1:1:vert
      outpic(V,H)=(0.299*pic(V,H,1)+0.587*pic(V,H,2)+0.114*pic(V,H,3));

   end
end

if exist('filename')
  filename=strcat(filename,'.png');
  imwrite(uint8(outpic),filename,'png');
else
  imwrite(uint8(outpic),'gray.png','png');
end
```

C.3 histo.m

```
function [Y3] = histo(pic)

% Histogramm Plot
%
%
% Y = histo(pic)

% block Grösse ex. 8 = 8x8 pixel Blöcke
block=8;

% calulate Y Values mean for blocks
vert=length(pic(:,1,1));
horz=length(pic(1,:,1));
Z=1;

for H=1:block:horz
   for V=1:block:vert
      % H V geben mir die Startposition jedes z.b 8x8er Werte Blockes
      ZZ=1;
      for HH=0:1:(block-1)
         if H+HH <= horz
          for VV=0:1:(block-1)
            if V+VV <= vert
               YY(ZZ)=(0.299*pic(V+VV,H+HH,1)+0.587*pic(V+VV,H+HH,2)...
                      +0.114*pic(V+VV,H+HH,3));
               ZZ=ZZ+1;
            end
          end
         end
      end
     Y(Z)=mean(YY);
     Z=Z+1;
   end
end

% round, sort, caculate same values and plot
Y2=round(sort(Y));

ylength=(length(Y2));
anzahl=1;
m=1;
for i=1:1:(ylength-1)
```

```
   if Y2(i) == Y2(i+1)
     anzahl=anzahl+1;
     Y3(m,1)=Y2(i);
     Y3(m,2)=anzahl;
   else
     Y3(m,1)=Y2(i);
     Y3(m,2)=anzahl;
     anzahl=1;
     m=m+1;
   end
end

bar(Y3(:,1),Y3(:,2))
title('Graustufen Histogramm')
xlabel('Helligkeit 0=schwarz - 255=weiss')
ylabel('Anzahl Bildpunkte dieses Wertes')
```

C.4 YCbCr.m

```
function [outpic] = YCbCr(pic)

% Zerlegt Bild in Y,Cb,Cr
%
%
% use [outpic] = YCbCr(pic)

vert=length(pic(:,1,1));
horz=length(pic(1,:,1));

for H=1:1:horz
   for V=1:1:vert
       outpic(V,H,1)=(0.299*pic(V,H,1)+0.587*pic(V,H,2)+0.114*pic(V,H,3));
       outpic(V,H,2)=(-0.169*pic(V,H,1)-0.331*pic(V,H,2)+0.500*pic(V,H,3));
       outpic(V,H,3)=(0.500*pic(V,H,1)-0.419*pic(V,H,2)-0.081*pic(V,H,3));
   end
end
```

C.5 RGB.m

```
function [outpic] = RGB(pic)

% Zerlegt Bild in R,G,B
%
%
% use [outpic] = RGB(pic)


vert=length(pic(:,1,1));
horz=length(pic(1,:,1));

for H=1:1:horz
   for V=1:1:vert
       outpic(V,H,1)=pic(V,H,1)+1.402*pic(V,H,3);
       outpic(V,H,2)=pic(V,H,1)-0.71414*pic(V,H,3)-0.34414*pic(V,H,2);
       outpic(V,H,3)=pic(V,H,1)+1.7772*pic(V,H,2);
   end
end
```

C.6 subsamp.m

```
function [outpic] = subsamp(pic,sample)

% Subsample the picture
%
% 4:4:4 -> for 4Y = 4Cb,4Cr
% 4:2:1 -> for 4Y = 2Cb,2Cr
% 4:1:1 -> for 4Y = 1Cb,1Cr
%
% use outpic=subsamp(pic,sample)
%

vert=length(pic(:,1,1));
horz=length(pic(1,:,1));

outpic(:,:,1)=pic(:,:,1);

if exist('sample')
    sample=sample;
else
    sample=2;
end

if sample == 2

 for H=1:1:horz
    for V=1:2:vert
       outpic(V,H,2)=(pic(V,H,2)+pic(V+1,H,2))/2;
       outpic(V+1,H,2)=(pic(V,H,2)+pic(V+1,H,2))/2;
       outpic(V,H,3)=(pic(V,H,3)+pic(V+1,H,3))/2;
       outpic(V+1,H,3)=(pic(V,H,3)+pic(V+1,H,3))/2;
    end
 end

elseif sample == 1

 for H=1:1:horz
    for V=1:4:vert
       outpic(V,H,2)=(pic(V,H,2)+pic(V+1,H,2)+pic(V+2,H,2)+pic(V+3,H,2))/4;
       outpic(V+1,H,2)=(pic(V,H,2)+pic(V+1,H,2)+pic(V+2,H,2)+pic(V+3,H,2))/4;
       outpic(V+2,H,2)=(pic(V,H,2)+pic(V+1,H,2)+pic(V+2,H,2)+pic(V+3,H,2))/4;
       outpic(V+3,H,2)=(pic(V,H,2)+pic(V+1,H,2)+pic(V+2,H,2)+pic(V+3,H,2))/4;
```

```
        outpic(V,H,3)=(pic(V,H,3)+pic(V+1,H,3)+pic(V+2,H,3)+pic(V+3,H,3))/4;
        outpic(V+1,H,3)=(pic(V,H,3)+pic(V+1,H,3)+pic(V+2,H,3)+pic(V+3,H,3))/4;
        outpic(V+2,H,3)=(pic(V,H,3)+pic(V+1,H,3)+pic(V+2,H,3)+pic(V+3,H,3))/4;
        outpic(V+3,H,3)=(pic(V,H,3)+pic(V+1,H,3)+pic(V+2,H,3)+pic(V+3,H,3))/4;
    end
 end

else

    outpic(:,:,2)=pic(:,:,2);
    outpic(:,:,3)=pic(:,:,3);

end
```

C.7 tocolor.m

```
function [outpic] = tocolor(pic,block,filename)

% Zerlegt Bild in Y,Cb,Cr Bloecke -> export als png
%
%
% use [outpic] = tocolor(pic,blocksize,filename)

% block Grösse ex. 8 = 8x8 pixel Blöcke
if ~exist('block')
   block=8;
end


% calulate Y Values mean for blocks
vert=length(pic(:,1,1));
horz=length(pic(1,:,1));
Z=1;

for H=1:block:horz
   for V=1:block:vert
      % H V geben mir die Startposition jedes ex. 64er Werte Blockes
      ZZ=1;
      for HH=0:1:(block-1)
         if H+HH <= horz
          for VV=0:1:(block-1)
            if V+VV <= vert
               YY(ZZ)=(0.299*pic(V+VV,H+HH,1)+0.587*pic(V+VV,H+HH,2)...
                      +0.114*pic(V+VV,H+HH,3));
               CCB(ZZ)=(-0.169*pic(V+VV,H+HH,1)-0.331*pic(V+VV,H+HH,2)...
                      +0.500*pic(V+VV,H+HH,3));
               CCR(ZZ)=(0.500*pic(V+VV,H+HH,1)-0.419*pic(V+VV,H+HH,2)...
                      -0.081*pic(V+VV,H+HH,3));
               ZZ=ZZ+1;
            end
          end
         end
      end
     Y(Z)=mean(YY);
     CB(Z)=mean(CCB);
     CR(Z)=mean(CCR);
     Z=Z+1;
   end
```

```
end

Z=1;
for H=1:block:horz
   for V=1:block:vert

      % H V geben mir die Startposition jedes ex. 64er Werte Blockes
      for HH=0:1:(block-1)
         if H+HH <= horz
          for VV=0:1:(block-1)
            if V+VV <= vert
               outpic(V+VV,H+HH,1)=Y(Z)+1.402*CR(Z);
               outpic(V+VV,H+HH,2)=Y(Z)-0.71414*CR(Z)-0.34414*CB(Z);
               outpic(V+VV,H+HH,3)=Y(Z)+1.7772*CB(Z);
            end
          end
       end
     end
    Z=Z+1;

 end
end
if exist('filename')
  filename=strcat(filename,'.png');
  imwrite(uint8(outpic),filename,'png');
else
  imwrite(uint8(outpic),'color.png','png');
end
```

C.8 block8.m

```
function [yblock,cbblock,crblock] = block8(pic)

% Zerlegt das Bild in 8x8 Blöcke
%
%
% use [yblock,cbblock,crblock] = block(pic)

block=8;

vert=length(pic(:,1,1));
horz=length(pic(1,:,1));
Z=1;

for H=1:block:horz
   for V=1:block:vert
      % H V geben mir die Startposition jedes 64er Werte Blockes
      ZZ=1;
      for HH=0:1:(block-1)
        if H+HH <= horz
          for VV=0:1:(block-1)
            if V+VV <= vert
               yblock(Z,VV+1,HH+1)=pic(V+VV,H+HH,1);
               cbblock(Z,VV+1,HH+1)=pic(V+VV,H+HH,2);
               crblock(Z,VV+1,HH+1)=pic(V+VV,H+HH,3);
            end
          end
        end
     end
     Z=Z+1;
   end
end
```

C.9 dctfunk.m

```
function [dctblock] = dctfunk(block)

% Berechnet die FDCT von einem 8x8 Block
%
%
% use [dctblock] = dctfunk(block)

dctblock=zeros(8,8);

for i=0:7
 for ii=0:7
  for n1=0:7
    for n2=0:7

      dctblock(i+1,ii+1) = dctblock(i+1,ii+1) + ...
         block(n1+1,n2+1)*cos(pi*(2*n1+1)*(i/16))...
         *cos(pi*(2*n2+1)*(ii/16));
    end
   end

      if i==0
         ci=(1/2^(1/2));
      else
         ci=1;
      end

      if ii==0
         cii=(1/2^(1/2));
      else
         cnii=1;
      end

     dctblock(i+1,ii+1) = dctblock(i+1,ii+1) *  (ci*cii/4);

 end
end
```

C.10 quant.m

```
function [qmat] = quant(block,type)

% Quantisiert einen Block von 64 Werten
%
%
% type= Y=1 od. C=2
%
% use [qmat] = quant(block,type)

Ymat=[16 11 10 16 24 40 51 61;...
      12 12 14 19 26 58 60 55;...
      14 13 16 24 40 57 69 56;...
      14 17 22 29 51 87 80 62;...
      18 22 37 56 68 109 102 77;...
      24 35 55 64 81 104 113 92;...
      49 64 78 87 103 121 120 101;...
      72 92 95 98 112 100 103 99];

Cmat=[17 18 24 99 99 99 99 99;...
      18 21 26 66 99 99 99 99;...
      24 26 56 99 99 99 99 99;...
      47 66 99 99 99 99 99 99;...
      99 99 99 99 99 99 99 99;...
      99 99 99 99 99 99 99 99;...
      99 99 99 99 99 99 99 99;...
      99 99 99 99 99 99 99 99];

if type == 1
   for i=1:8
      for ii=1:8
         qmat(ii,i)=round(block(ii,i)/Ymat(ii,i));
      end
   end
elseif type == 2
     for i=1:8
      for ii=1:8
         qmat(ii,i)=round(block(ii,i)/Cmat(ii,i));
      end
   end
end
```

Literaturverzeichnis

[1] Tilo Strutz. *Bildatenkompression.* Vieweg ISBN: 3-528-13922-, 2002.