



Technisch-Naturwissenschaftliche  
Fakultät

# **Die JPEG-Kompression**

## **Eine Simulation für Lehrerinnen und Lehrer**

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magistra der Naturwissenschaften

im Diplomstudium

Mathematik Lehramt

Eingereicht von:

Alicia Hofstätter, BSc.

Angefertigt am:

Institut für Algebra

Beurteilung:

Assoz. Prof. Dr. Erhard Aichinger

Linz, November 2015

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, am 18. November 2015

Alicia Hofstätter, BSc.

# Inhaltsverzeichnis

<b>Eidesstattliche Erklärung</b>	<b>ii</b>
<b>Kurzfassung</b>	<b>v</b>
<b>Abstract</b>	<b>vi</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 Grundbegriffe</b>	<b>3</b>
2.1 Farbräume . . . . .	3
2.1.1 RGB-Farbraum . . . . .	3
2.1.2 YCbCr-Farbraum . . . . .	4
2.2 Digitale Bilder . . . . .	5
2.2.1 Informationsgehalt . . . . .	7
2.2.2 Entropie . . . . .	8
2.3 Codierung . . . . .	9
2.3.1 Lauffängencodierung . . . . .	9
2.3.2 Huffman-Codierung . . . . .	11
2.4 Quantisierung . . . . .	14
<b>3 Diskrete Kosinustransformation</b>	<b>16</b>
3.1 Eindimensionale DCT . . . . .	17
3.1.1 Grauwertberechnung . . . . .	22
3.2 Zweidimensionale DCT . . . . .	23
<b>4 Die JPEG-Kompression</b>	<b>26</b>
4.1 Farbraumkonversion . . . . .	27
4.2 Diskrete Kosinustransformation . . . . .	27
4.3 Quantisierung . . . . .	28
4.4 Codierung . . . . .	30
4.5 Decodierung und Rekonstruktion . . . . .	32
4.6 Vom Bild zum Binärcode . . . . .	36

<b>5</b>	<b>Der JPEG-Simulator</b>	<b>38</b>
5.1	Beschreibung der Eingabe . . . . .	38
5.2	Beschreibung der Ausgabe . . . . .	40
5.3	Eigenschaften der JPEG-Kompression . . . . .	42
<b>6</b>	<b>Analyse der Vermittelbarkeit an Schulen und höheren Bildungsanstalten</b>	<b>47</b>
<b>7</b>	<b>Dokumentation des Quellcodes</b>	<b>49</b>
7.1	Klasse ImageComponent . . . . .	49
7.2	Klasse Gui . . . . .	49
7.3	Klasse jpegSimulator . . . . .	49
7.3.1	Globale Variablen . . . . .	50
7.3.2	Private Methoden . . . . .	50
7.3.3	Öffentliche Methoden . . . . .	50
<b>A</b>	<b>Sourcecode</b>	<b>55</b>
	<b>Abbildungsverzeichnis</b>	<b>57</b>
	<b>Tabellenverzeichnis</b>	<b>59</b>
	<b>Literaturverzeichnis</b>	<b>60</b>

# Kurzfassung

Die JPEG-Kompression ist eine weit verbreitete Kompressionsart, die in vielen Bereichen der digitalen Bildverarbeitung eingesetzt wird. Da das Verfahren umfangreich und mathematisch komplex ist, gestaltet sich auch eine didaktische Aufbereitung schwierig und zeitaufwendig. Diese Arbeit erklärt die zum Verständnis notwendigen Grundlagen und Begriffe sowie die wesentlichen Schritte und Formeln der JPEG-Kompression mit dem Ziel, Lehrerinnen und Lehrern die Aufbereitung zu erleichtern und eine Behandlung des Themas in der Schule zu ermöglichen. In dieser Arbeit wird ein Simulationsprogramm – der JPEG-Simulator – vorgestellt, das die einzelnen Schritte der JPEG-Kompression grafisch und tabellarisch darstellt und die Eingabe verschiedener Parameter ermöglicht. Der JPEG-Simulator dient einerseits Lehrerinnen und Lehrern als didaktisches Werkzeug und andererseits Schülerinnen und Schülern sowie Studentinnen und Studenten als Basis für experimentelles Lernen und Verstehen.

# Abstract

The JPEG compression is a widespread type of compression that is commonly used in digital image processing. Because of the sophisticated and mathematically complex process a didactic preparation is difficult and time-consuming. This thesis explains the necessary principles and terminology as well as the main steps and formulas of the JPEG compression. The goal is to simplify didactic preparation for teachers and make approaching this topic in school possible. In the course of this thesis a simulation software – called JPEG-Simulator – is introduced, which illustrates the steps of the JPEG compression graphically and tabularly and also allows various user input. The JPEG-Simulator is meant as a didactic tool for teachers and as a basis for experimental learning and understanding for students.

# Kapitel 1

## Einleitung

In Zeiten fortschreitender Digitalisierung und Technisierung sind wir immer mehr und früher mit technischen Geräten und Fachbegriffen konfrontiert. Diese Entwicklung spiegelt sich auch in der Lebensweise wieder. Viele Menschen besitzen ein Smartphone, eine Digitalkamera oder einen Computer und nutzen täglich das Internet. Mit verschiedenen Geräten können digitale Fotos aufgenommen und gespeichert werden und es wird dem Benutzer abgenommen, sich um die Technik oder den Speicherbedarf Gedanken zu machen. Den meisten Menschen ist die Abkürzung „JPEG“ zwar bekannt, sie wird jedoch oft als Synonym für *Foto* erachtet oder zumindest mit Fotografie in Zusammenhang gebracht. Tatsächlich steht „JPEG“ für *Joint Photographic Experts Group* [1] und bezeichnet eine Arbeitsgruppe, die sich unter anderem mit der Entwicklung von Codierungen und Kompressionen für Bilder beschäftigt. Seit 1992 wird die sogenannte JPEG-Kompression entwickelt, die sich zur wichtigsten und meist verbreiteten Kompression für Fotos etabliert hat. Fotos, die mit der JPEG-Kompression komprimiert wurden, erhalten zur Kennzeichnung die Dateierweiterung *.jpg* oder *.jpeg*. Die meisten Geräte im Amateurbereich komprimieren die aufgenommenen Fotos automatisch und speichern diese nur noch als JPEG-Dateien. Bei hochwertigeren Geräten, wie z. B. digitalen Spiegelreflexkameras, ist es fast immer möglich, Fotos unkomprimiert zu speichern. Auch beim Hochladen von Fotos ins Internet, z. B. in soziale Plattformen, werden diese automatisch komprimiert. Der Sinn jeder Kompression liegt in der effizienten Speicherung von Information. Dies geschieht jedoch meist auf Kosten der Bildqualität, da Kompression in der Regel mit Qualitätsverlust verbunden ist.

Da die Digitalisierung auch vor der Schule nicht Halt macht, ist es durchaus sinnvoll, Schülerinnen und Schülern einen Einblick in diese Technik zu gewähren. Die JPEG-Kompression ist ein aufwendiges und mathematisch komplexes Verfahren, das in vielen Büchern nur oberflächlich behandelt wird. In berufsbildenden höheren Schulen mit informatischem oder technischem Schwerpunkt sowie in technisch orientierten, universitären Einrich-

tungen werden Kompressionen natürlich schon längst gelehrt. Eine didaktische Aufbereitung der JPEG-Kompression ist jedoch schwierig und zeitaufwendig. Auf den ersten Blick scheint dieses Thema für die Schule ungeeignet zu sein. Bei genauerer Betrachtung lässt sich die JPEG-Kompression aber von zwei Seiten behandeln – von der mathematischen einerseits und von der optisch-qualitativen andererseits. Dieser Arbeit ist zum Ziel gesetzt, die JPEG-Kompression umfassend und ausführlich zu erklären, beide Seiten zu betrachten und die Inhalte für Schülerinnen und Schüler, Studentinnen und Studenten sowie Lehrerinnen und Lehrer greifbar zu machen. In erster Linie soll diese Arbeit Lehrerinnen und Lehrer ansprechen und ihnen als umfassende Grundlage zur didaktischen Aufbereitung dienen.

In Kapitel 2 werden die Grundbegriffe erläutert, die zum Verständnis der JPEG-Kompression notwendig sind. Die sogenannte Kosinustransformation bildet das Kernelement der Kompression und wird in Kapitel 3 ausführlich beschrieben. Anschließend wird in Kapitel 4 der vollständige Ablauf der JPEG-Kompression behandelt und beispielhaft vorgeführt. Den praktischen Teil dieser Arbeit bildet der in Kapitel 5 beschriebene JPEG-Simulator – ein Programm, das die Funktionsweise der JPEG-Kompression und die Auswirkungen der Eingabeparameter auf das Kompressionsergebnis verdeutlichen soll. Der JPEG-Simulator ist als didaktisches Werkzeug gedacht und soll einerseits Lehrerinnen und Lehrer bei der Aufbereitung des Themas unterstützen, andererseits auch Lernenden als Grundlage für experimentelles Lernen dienen. Abschließend werden in Kapitel 6 die Vermittelbarkeit der JPEG-Kompression sowie die Einsatzmöglichkeiten des JPEG-Simulators im Unterricht diskutiert. Der Quellcode des JPEG-Simulators ist in Anhang A bzw. auf der beiliegenden CD zu finden. Der JPEG-Simulator befindet sich ebenfalls auf der CD und ist online unter <http://www.aliciakrenn.at/da.html> zu finden. Die wesentlichen Elemente des Programms sind in Kapitel 7 dokumentiert.

# Kapitel 2

## Grundbegriffe

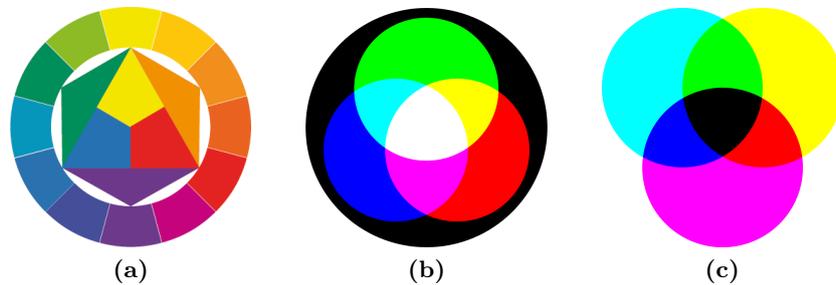
### 2.1 Farbräume

Die klassische Farbenlehre hat verschiedene Varianten hervorgebracht, Farben zu ordnen und zu systematisieren. Die bekanntesten sind die Farbenlehren von Johannes Itten (Abbildung 2.1 (a)) und Johann Wolfgang von Goethe. Im digitalen Zeitalter sind einige weitere Varianten der Farbdarstellung hinzu gekommen. Sie werden Farbräume genannt [9, S. 252ff.], da sie im mathematischen Sinn als dreidimensionale Objekte interpretiert werden. Je nach Anwendung sind verschiedene Farbräume gebräuchlich, die sich in ihren Eigenschaften sowie der Anzahl und Art der Komponenten unterscheiden.

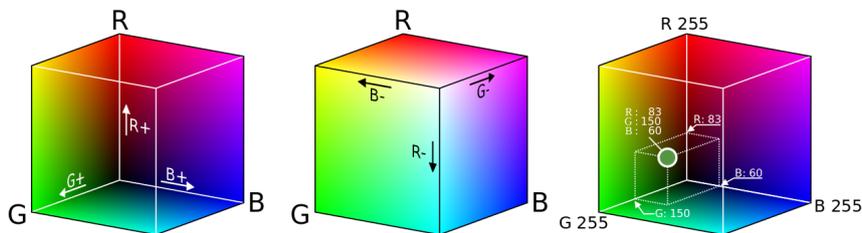
Der grundlegendste Farbraum ist der RGB-Raum [9, S. 252], der aus den Grundfarben Rot, Grün und Blau alle weiteren Farben erzeugt (Abbildung 2.1 (b)). Im digitalen Druck werden alle Farben aus den Grundfarben Cyan, Magenta, Gelb (Yellow) und Schwarz (Key) des CMYK-Raums [9, S. 273] gebildet (Abbildung 2.1 (c)). In der Fernsehtechnik wird der YUV-Raum [9, S. 268] eingesetzt, da dieser die Helligkeitsinformation (Y) von der Farbinformation (UV) trennt. Dadurch war u. a. die Rückwärtskompatibilität zu Schwarz-Weiß-Fernsehern gegeben. Dies sind nur einige kurze Beispiele, um die Vielfalt der digitalen Farbräume zu verdeutlichen. In der JPEG-Kompression spielen zwei Farbräume eine wichtige Rolle, die im Folgenden genauer betrachtet werden.

#### 2.1.1 RGB-Farbraum

Der RGB-Raum [15, S. 179f.] ist ein additives System, das meist dann zur Anwendung kommt, wenn Farben durch Licht erzeugt werden. Dies ist beispielsweise bei Bildschirmen und Projektoren der Fall. Die Buchstaben RGB stehen für die Farbkomponenten Rot, Grün und Blau (Abbildung 2.1 (b)). Dieses System ist nicht zu verwechseln mit der RGB-Farbmischung aus der klassischen Farbenlehre nach Johannes Itten [11, S. 30ff.]. In Ittens



**Abbildung 2.1:** Farbkreis nach Johannes Itten (a) [11, S. 31] (Die Datei wurde aus [2] entnommen.), additive Farbmischung RGB (b) [3] (Das Originalbild wurde modifiziert.) und subtraktive Farbmischung CMYK (c) [4].



**Abbildung 2.2:** RGB-Farbwürfel [5].

Farbenlehre stehen die Buchstaben RGB für Rot, *Gelb* und Blau (Abbildung 2.1 (a)). Bei Letzterem handelt es sich um ein subtraktives System, wie es z. B. beim Malen mit Wasserfarben oder Ölfarben zum Einsatz kommt. Werden alle drei Farben (oder je zwei Komplementärfarben) im richtigen Verhältnis gemischt, entsteht Grau. Weiß, Schwarz und Grau werden nach Itten nicht als Farben bezeichnet [11, S. 37f.]. Im Gegensatz dazu mischen sich die additiven Komponenten Rot, Grün und Blau zu Weiß. Schwarz ist die Ausgangsfarbe, die auftritt wenn alle drei Farbkomponenten inaktiv sind.

Die mathematische Repräsentation des RGB-Raums kann als Einheitswürfel [15, S. 179f.] in einem dreidimensionalen Koordinatensystem dargestellt werden (Abbildung 2.2). Dabei beschreiben die Achsen des Systems die Komponenten Rot, Grün und Blau. Werden alle drei Komponenten zu gleichen Anteilen gemischt, entsteht immer ein Grauton. Daher liegen alle Graustufen entlang der Hauptdiagonale des Würfels.

### 2.1.2 YCbCr-Farbraum

Der YCbCr-Raum [15, S. 182ff.] ist eine Abwandlung des für die Fernsehtechnik entwickelten YUV-Raums und besteht wie auch der RGB-Raum aus drei Komponenten. Jedoch werden die Farben durch einen Helligkeitswert

Y (Luminanz) und zwei Farbdifferenzwerte Cb (Chrominanz Blau) und Cr (Chrominanz Rot) beschrieben. Der Vorteil des YCbCr-Raums gegenüber dem RGB-Raum besteht darin, dass die Chrominanzanteile getrennt von der Luminanz behandelt und verändert werden können. Dies bringt unter anderem Vorteile in der Datenreduktion, da das menschliche Auge für Helligkeitsinformation empfindlicher ist als für Farbinformation. Ohne sichtbaren Qualitätsverlust können die Farbdifferenzwerte mit geringerer Auflösung gespeichert werden.

Im RGB-Raum gespeicherte Farbwerte können mit wenig Aufwand in den YCbCr-Raum konvertiert werden. Dabei werden, vereinfacht dargestellt, folgende Gleichungen gebildet:

$$\begin{aligned} Y &= \text{Rot} + \text{Grün} + \text{Blau} \\ Cb &= \text{Blau} - \text{Rot} - \text{Grün} \\ Cr &= \text{Rot} - \text{Blau} - \text{Grün}. \end{aligned}$$

Tatsächlich werden die Anteile der einzelnen Farbkomponenten gewichtet. In [15, S. 183] wird folgende Notation für die Farbraumtransformation von RGB nach YCbCr vorgeschlagen:

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0,2990 & 0,5870 & 0,1140 \\ -0,1687 & -0,3313 & 0,5000 \\ 0,5000 & -0,4187 & -0,0813 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}. \quad (2.1)$$

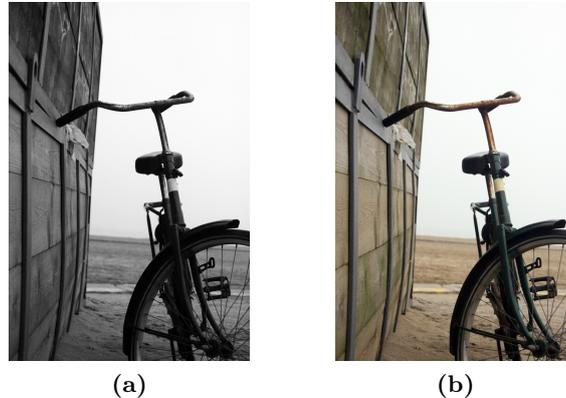
Die Rücktransformation von YCbCr nach RGB erfolgt durch die inverse Matrix:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1,0 & 0,0000 & 1,4020 \\ 1,0 & -0,3441 & -0,7141 \\ 1,0 & 1,7720 & -0,0001 \end{pmatrix} \cdot \begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix}.$$

## 2.2 Digitale Bilder

Unter einem digitalen Bild [9, S. 5ff.] ist nicht nur ein Foto zu verstehen – der Begriff umfasst auch optische Information wie z. B. Ultraschall- und Röntgenbilder oder Mikroskop- und Teleskopaufnahmen. Digitale Bilder können auf verschiedene Weise betrachtet werden. Sie können einerseits als Punktmenge in einem Koordinatensystem verstanden werden, wobei ein Punkt  $x$  Pixel genannt wird und die Koordinaten  $x = (n, m)$  sowie einen Grauwert oder mehrere Farbwerte besitzt, falls es sich um ein Farbbild handelt. Andererseits kann ein digitales Bild als zweidimensionale, diskrete Funktion aufgefasst werden. In der Informationstheorie wird auch allgemein von Signalen gesprochen.

Auf Speicherebene wird grundsätzlich jede Art von Information als binäre Zahlenfolge abgelegt. Unter einer Binärdarstellung der Zahl  $a$  verstehen



**Abbildung 2.3:** Grauwertbild (a) und Farbbild (b). (Foto: Hofstätter)

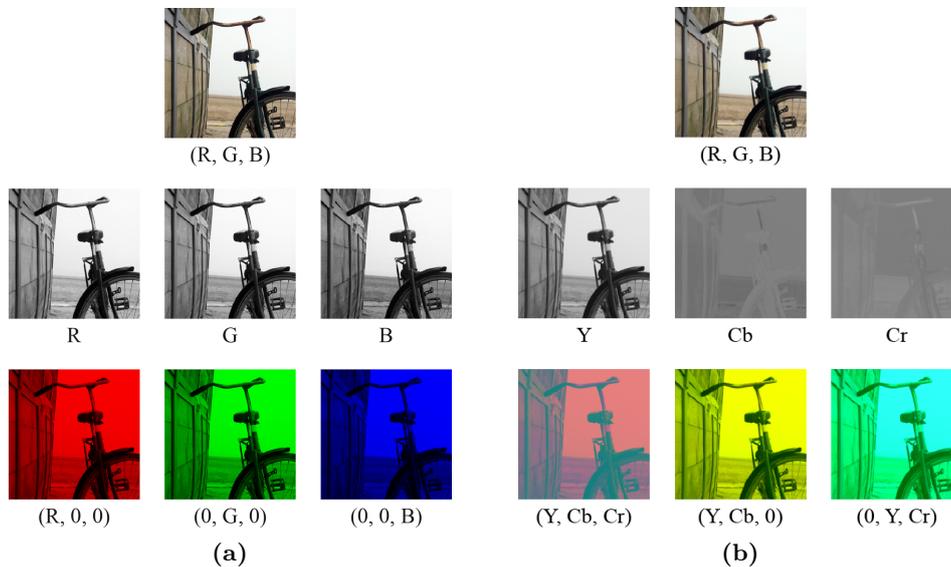
wir eine Folge der Form  $(a_n \dots a_k \dots a_1 a_0)$  mit Ziffern  $a_k \in \{0, 1\}$  und Stellenanzahl  $(n + 1)$ , für die  $a = \sum_{k=0}^n a_k \cdot 2^k$  gilt, wobei die Stellen auch Bits („Binary Digits“) genannt werden [7, S. 47].

Im Anwenderbereich stehen einem Grauwertbild [9, S. 13] meist 8 Bits pro Pixel und damit  $2^8 = 256$  verschiedene Graustufen zur Verfügung (Abbildung 2.3 (a)). Diese werden durch natürliche Zahlen im Intervall  $[0, 255]$  repräsentiert. Dabei steht 0 für Schwarz und 255 für Weiß. In Bereichen wie Medizin oder Astronomie haben Bilder eine höhere Auflösung. Dort sind Bildtiefen bis zu 16 Bits pro Pixel üblich. Mit 16 Bits können  $2^{16} = 65\,536$  verschiedene Graustufen dargestellt werden.

In einem typischen Farbbild [9, S. 13f.] sind jedem Pixel drei oder mehr Komponenten zugeordnet (Abbildung 2.3 (b)). Im Falle eines RGB-Bildes sind das Rot, Grün und Blau, die als Zahlentripel (R,G,B) gespeichert werden. Jeder Komponente sind 8 Bits Speicherplatz zugewiesen, wodurch für jede Komponente 256 Stufen zur Verfügung stehen. Daraus ergeben sich für das RGB-Bild  $256^3 = 16\,777\,216$  darstellbare Farben. Alle drei Komponenten werden jeweils durch natürliche Zahlen im Intervall  $[0, 255]$  dargestellt. Der Dezimalcode für Schwarz ist  $(0,0,0)$ , der Code für Weiß ist  $(255,255,255)$ . Farbbilder für professionelle Anwendungen können beispielsweise mit Bildtiefen von  $3 \times 14$  Bits = 42 Bits sogar  $2^{42} = 4\,398\,046\,511\,104$  Farben unterscheiden.

Einem YCbCr-Bild [15, S. 183] stehen ebenfalls 8 Bits pro Komponente zur Verfügung, wobei die Intervalle verschoben sind. Y wird durch natürliche Zahlen im Intervall  $[0, 255]$  dargestellt. Cb und Cr liegen im Intervall  $[-127,5; 127,5]$  und werden zur digitalen Weiterverarbeitung ganzzahlig gerundet und mit einem Offset in einen positiven Wertebereich gebracht.

Die Darstellung der einzelnen Komponenten ist je nach Farbraum unterschiedlich. Die Komponenten eines RGB-Bildes können als Einzelbilder



**Abbildung 2.4:** Darstellung der Komponenten Rot (R), Grün (G) und Blau (B) des RGB-Raums (a) und Grauwert- bzw. RGB-Darstellung der Komponenten Luminanz (Y), Chrominanz Blau (Cb) und Chrominanz Rot (Cr) des YCbCr-Raums (b).

interpretiert werden [15, S. 184] und sind üblicherweise als Grauwertbilder oder Farbbilder in der Farbe der jeweiligen Komponente dargestellt. Dies ist möglich, da sich Grauwerte, mathematisch gesehen, nicht von Farbwerten unterscheiden. Innerhalb der Rot-Komponente (R) beispielsweise steht die Zahl 0 für Schwarz und 255 für reines Rot. Abbildung 2.4(a) zeigt beide Varianten.

Die Darstellung der Komponenten des YCbCr-Raums ist schwieriger, da Farbdifferenzen nur schwer als Farbbilder interpretiert werden können. Abbildung 2.4(b) zeigt neben der gebräuchlichen Darstellung in Grauwertbildern [15, S. 185] auch verschiedene Möglichkeiten der färbigen Darstellung. Dabei werden die drei Komponenten (Y,Cb,Cr) als RGB-Komponenten interpretiert und in diesem Farbraum dargestellt.

### 2.2.1 Informationsgehalt

Der Informationsgehalt [15, S. 5f.] eines Symbols oder eines Ereignisses ist ein Begriff der Informationstheorie. Er beschreibt die Relevanz eines Ereignisses in Abhängigkeit von der Wahrscheinlichkeit seines Auftretens. Je seltener ein bestimmtes Ereignis auftritt, desto höher ist sein Informationsgehalt. Dieses Konzept ist auf viele Bereiche anwendbar. Im Falle eines Pixelbildes bedeutet dies, dass bei vielen gleichfarbigen Pixeln (Symbolen)

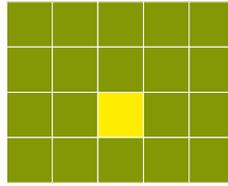


Abbildung 2.5: Bild mit 20 Pixeln.

der Informationsgehalt des einzelnen Pixels gering ist, wohingegen der Informationsgehalt eines einzelnen andersfarbigen Pixels sehr hoch ist. Der Informationsgehalt  $I$  des Symbols  $s_i$  in Abhängigkeit von dessen Auftretenswahrscheinlichkeit  $p_i = p(s_i)$  wird durch die Gleichung

$$I(s_i) = \log_2 \frac{1}{p_i} \quad [\text{bit}] \quad (2.2)$$

beschrieben. Folgendes Beispiel soll dies verdeutlichen: Soll das Pixelbild aus Abbildung 2.5 beschrieben werden, so ist die Relevanz des einen gelben Pixels wesentlich größer als die eines einzelnen grünen Pixels. Am Beispiel des Pixelbildes aus Abbildung 2.5 ergeben sich die Auftretenswahrscheinlichkeiten  $p(\text{grün}) = \frac{19}{20}$  und  $p(\text{gelb}) = \frac{1}{20}$ . Mit Gleichung 2.2 erhalten wir die Informationsgehalte  $I(\text{grün}) \approx 0,074$  bit und  $I(\text{gelb}) \approx 4,322$  bit. Der Informationsgehalt des gelben Pixels ist also um etwa einen Faktor 58 größer.

### 2.2.2 Entropie

Im Zusammenhang mit Informationstheorie wurde die Entropie erstmals von Claude E. Shannon [14, S. 59ff.] beschrieben. Die Entropie  $H$  beschreibt den mittleren Informationsgehalt eines Signals und wird laut [15, S. 7] in folgender Weise berechnet:

$$H = \sum_{i=1}^K p_i \log_2 \frac{1}{p_i} = - \sum_{i=1}^K p_i \log_2 p_i \quad [\text{bit/Symbol}]. \quad (2.3)$$

$K$  bezeichnet dabei die Anzahl verschiedener Symbole und  $p_i$  die Wahrscheinlichkeit, mit der das  $i$ -te Symbol auftritt. Das Pixelbild in Abbildung 2.5 beispielsweise enthält  $K = 2$  verschiedenfarbige Pixel und hat mit Gleichung 2.3 die Entropie  $H \approx 0,286$  bit/Symbol. Der minimale Entropiewert  $H = 0$  wird erreicht, wenn das gesamte Bild aus nur einer Farbe besteht. Für ein einheitlich grünes Bild ist die Auftretenswahrscheinlichkeit  $p(\text{grün}) = 1$ , der Informationsgehalt jedes Pixels  $I(\text{grün}) = 0$  und damit auch die Entropie  $H = 0$ . Ihr Maximum erreicht die Entropie laut [15, S. 7], wenn die vorkommenden Symbole, z. B. Farben, gleich verteilt sind. Es gilt

**Tabelle 2.1:** RGB-Farbwerte und zugehörige Binärcodes.

Farbe	R	G	B	Binärcode R	Binärcode G	Binärcode B
	132	151	5	10000100	10010111	00000101
	255	237	0	11111111	11101101	00000000
	191	146	0	10111111	10010010	00000000

also für jedes Symbol eine Auftretenswahrscheinlichkeit  $p_i = 1/K$  und ein Informationsgehalt  $I(p_i) = \log_2(K)$  bei  $K$  verschiedenen Symbolen. Die maximale Entropie ist demnach

$$H_{max} = \sum_{i=1}^K \frac{1}{K} \log_2 K = \log_2 K. \quad (2.4)$$

Die Berechnung der Entropie kann in weiterer Folge zur Bestimmung des mindestens benötigten Speicherplatzes herangezogen werden und kommt in der sogenannten Entropiecodierung zur Anwendung (siehe Abschnitt 2.3.2).

## 2.3 Codierung

Die Weise, in der Information codiert wird, kann stark variieren. Die einfachste Variante ist eine direkte Umrechnung von Dezimalzahlen in die Binärdarstellung. Da Farben in RGB-Bildern durch natürliche Zahlen im Intervall  $[0, 255]$  repräsentiert werden, können diese direkt in Binärzahlen umgerechnet werden. Tabelle 2.1 zeigt verschiedene Farben und die zugehörigen Dezimal- bzw. Binärcodes.

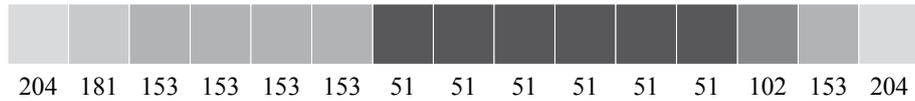
Dass diese Art der Codierung jedoch sehr ineffizient ist zeigt eine einfache Rechnung. Jeder Pixel eines RGB-Bildes wird durch  $3 \times 8 \text{ Bits} = 3 \text{ Byte}$ <sup>1</sup> repräsentiert. Für ein Bild mit 5 Mio. Pixeln<sup>2</sup> bedeutet das (ohne Kompression) einen Speicheraufwand von  $5 \cdot 10^6 \times 3 \text{ Byte} = 15 \cdot 10^6 \text{ Byte} \approx 14,3 \text{ Megabyte}$ . Mit Hilfe von geeigneten Codierungen und Kompressionen kann der Speicheraufwand für solche Bilder auf unter 1 Megabyte reduziert werden. In den folgenden Abschnitten werden zwei effiziente Codierungsarten betrachtet, die in der JPEG-Kompression eingesetzt werden.

### 2.3.1 Lauflängencodierung

In digitalen Bildern können viele gleiche Farb- oder Graustufen nebeneinander liegen. In solchen Fällen ist es naheliegend, diese in geeigneter Weise

<sup>1</sup>8 Bits = 1 Byte, 1024 Byte = 1 Kilobyte, 1024 Kilobyte = 1 Megabyte

<sup>2</sup>Typische Größe für Fotos von Digitalkameras, Smartphones, etc.



**Abbildung 2.6:** Ausschnitt aus einem Grauwertbild mit 15 Pixeln und zugehörige Grauwerte.

**Tabelle 2.2:** Lauflängencodierung.

Grauwerte	204 181 153 153 153 153 51 51 51 51 51 51 102 153 204
Differenzen	204 -23 -28 0 0 0 -102 0 0 0 0 0 51 51 51
Code	<b>0 204 0 -23 0 -28 3 -102 5 51 0 51 0 51</b>

**Tabelle 2.3:** Lauflängencodierung eines Vektors mit 63 Werten.

Vektor	0 -2 1 -1 0 0 -1 0 0 0 0 0 ... 0
Code	<b>1 -2 0 1 0 -1 2 -1 56</b>

zusammenzufassen. An Grauwertbildern wie Abbildung 2.3 (b) ist dies besonders gut zu erkennen. Abbildung 2.6 zeigt einen Ausschnitt aus einem Grauwertbild mit 15 Pixeln. In [15, S. 78ff.] sind folgende Varianten der Lauflängencodierung beschrieben:

Die zusammengefassten Grauwerte werden durch sogenannte Token ersetzt, die drei Elemente ( $ESC; r; s_i$ ) enthalten.  $ESC$  bezeichnet dabei ein Escape-Symbol, das den Beginn eines Token kennzeichnet. Als  $ESC$  dienen oft Symbole aus dem Alphabet aller möglichen Symbole, die im Signal nicht vorkommen. Die Anzahl von  $r$  gleichen Werten, die aufeinander folgen, wird Lauflänge genannt. Es folgt der Grauwert, der allgemein als Symbol  $s_i$  bezeichnet wird.

Wenn jeder Grauwert mit 8 Bits codiert wird ergibt sich für die 15 Pixel in Abbildung 2.6 ein Speicheraufwand von  $15 \times 8 \text{ Bits} = 120 \text{ Bits}$ .  $ESC$ ,  $r$  und  $s_i$  werden ebenfalls mit je 8 Bits codiert. Ein Token braucht also  $3 \times 8 \text{ Bits} = 24 \text{ Bits}$ . Es tritt somit erst ab vier zusammengefassten Grauwerten ein Kompressionseffekt auf. Mit 8 Bits kann eine maximale Lauflänge von  $r = 255$  dargestellt werden. Durch Speicherung der Lauflänge  $r - 4$  erhöht sich die maximale Lauflänge auf 259. Wird die Zahlenfolge aus Abbildung 2.6 mit dem Token ( $ESC; r - 4; s_i$ ) codiert, entsteht die neue Folge „204 181 ( $ESC; 0; 153$ ) ( $ESC; 2; 51$ ) 102 153 204“. Die 15 Farbwerte werden nun mit  $11 \times 8 \text{ Bits} = 88 \text{ Bits}$  codiert.

Sind für das Escape-Symbol keine freien Symbole verfügbar, besteht die Möglichkeit, den Token ( $s_i; r - 1$ ) zu verwenden. Dieser kommt zwar ohne

Escape-Symbol aus, codiert aber dafür jedes Element. Diese Variante ist nur sinnvoll, wenn selten einzelne Werte vorkommen, da diese von 8 auf 16 Bits expandieren. Für die Zahlenfolge aus Abbildung 2.6 entsteht die neue Folge „(204;0) (181;0) (153;3) (51;5) (102;0) (153;0) (204;0)“, für die  $14 \times 8 \text{ Bits} = 112 \text{ Bits}$  notwendig sind.

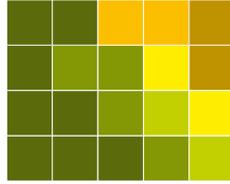
Folgende Variante der Lauflängencodierung ist besonders für die JPEG-Kompression von Bedeutung. Besteht eine Zahlenfolge vorwiegend aus Nullen, bietet es sich an, nur diese zusammenzufassen [15, S. 80f.]. Werden nur Nullen lauffängencodiert, sind im Token weder ein Escape-Symbol noch der Wert des Symbols  $s_i$  notwendig. In der codierten Folge alternieren Lauflängewerte und Symbole, wobei mit einer Lauflänge begonnen wird. Folgen zwei Werte ungleich Null aufeinander, so wird dazwischen eine Lauflänge 0 gesetzt. Im Fall des Grauwertbildes in Abbildung 2.6 kann beispielsweise für jeden Grauwert die Differenz zum jeweiligen Vorgänger berechnet werden. Dabei entstehen viele Nullen, die anschließend zusammengefasst werden können. Tabelle 2.2 zeigt, wie die Grauwerte aus Abbildung 2.6 auf diese Weise codiert werden. Die Lauflängen sind in der Tabelle hervorgehoben.

Die Mächtigkeit der Lauflängencodierung wird erst im Zusammenhang mit der JPEG-Kompression wirklich sichtbar. Wie später genauer erläutert wird, entstehen im Zuge der Kompression Vektoren mit 63 Werten, die fast alle Null sind (siehe Abschnitt 4.4). Die Werte ungleich Null befinden sich hauptsächlich am Anfang des Vektors. Eine Darstellung mit 8 Bits pro Wert würde einen Speicheraufwand von  $63 \times 8 \text{ Bits} = 504 \text{ Bits}$  pro Vektor bedeuten. Mit oben beschriebener Lauflängencodierung lässt sich dieser Aufwand um ein Vielfaches reduzieren, wie das Beispiel in Tabelle 2.3 verdeutlicht. Zur Darstellung dieses Vektors sind nur  $9 \times 8 \text{ Bits} = 72 \text{ Bits}$  notwendig.

### 2.3.2 Huffman-Codierung

Die Huffman-Codierung [10] ist eine sogenannte Entropiecodierung, die in vielen Anwendungen der Signalverarbeitung, z. B. in der JPEG-Kompression, zum Einsatz kommt. Ihr Ziel ist es, die mittlere Bitrate an die Entropie eines Signals anzunähern [15, S. 32]. Mittels des Informationsgehalts (Abschnitt 2.2.1) und der Entropie (Abschnitt 2.2.2) wird festgestellt, wieviele Bits mindestens notwendig sind, um ein Signal, z. B. ein Bild, verlustfrei zu speichern. Bei verlustfreier Codierung bleibt die ursprüngliche Information erhalten, während durch verlustbehaftete Codierung Information verloren geht, die nicht mehr rekonstruiert werden kann.

Die Werte  $s_i$  des Signals werden nicht direkt in Binärcodes übersetzt sondern durch binäre Codewörter  $c_i$  beschrieben, deren Länge  $l_i$  variabel ist [15, S. 32]. Anstatt jeden Farbwert eines Bildes mit 8 Bits zu codieren, können also beispielsweise manche durch 4 Bits und andere wiederum durch 8 oder 10 Bits dargestellt werden. Dabei werden häufig auftretende Werte, also solche mit niedrigem Informationsgehalt, durch kurze Codewörter und



**Abbildung 2.7:** Farbbild mit 20 Pixeln.

selten auftretende Werte durch lange Codewörter repräsentiert. Aus den Auftretenswahrscheinlichkeiten  $p_i$  und den Codewortlängen  $l_i$  ergibt sich die mittlere Codewortlänge

$$\bar{l} = \sum_{i=1}^K p_i \cdot l_i \quad [\text{Bits/Symbol}]. \quad (2.5)$$

Die minimale mittlere Codewortlänge  $\bar{l}$  ist nach unten durch die Entropie des Signals beschränkt. Das sogenannte *Noiseless Coding Theorem* wurde von Claude E. Shannon beschrieben und unter anderem in [6, S. 37f.] bewiesen. Das Theorem lautet allgemein für einen Code mit Basis  $D$  (in ähnlicher Form)

$$\frac{H}{\log_2 D} \leq \bar{l}.$$

Für einen Binärcode mit Basis  $D = 2$  ist die minimale mittlere Codewortlänge  $\bar{l}$  also größer oder gleich der Entropie  $H$ . In [6, S. 39] wird weiter gezeigt, dass immer eine optimale Codezuordnung existiert, die der Bedingung (in ähnlicher Form)

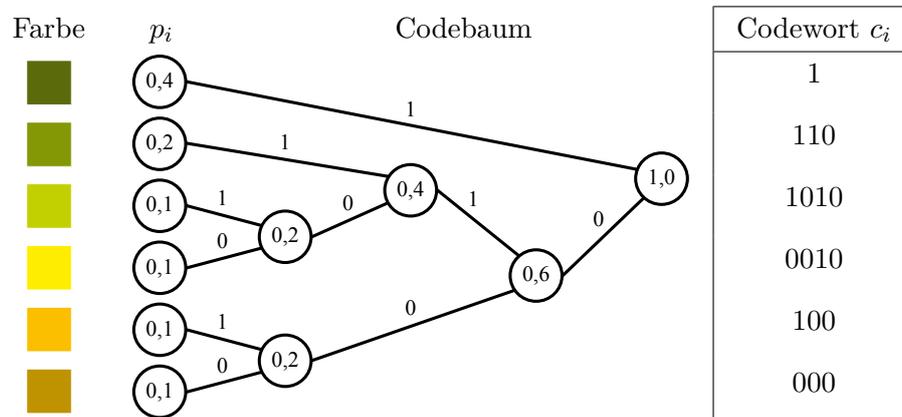
$$H \leq \bar{l} < H + 1 \quad (2.6)$$

genügt. Da die Entropie größer gleich Null und in der Einheit bit/Symbol gegeben ist, kann sie durch Aufrunden des Ergebnisses auf natürliche Zahlen direkt zur Berechnung der durchschnittlichen Codewortlänge  $\bar{l}$  eingesetzt werden. Selbes gilt für den Informationsgehalt eines Symbols und seine minimale Codewortlänge.

Angenommen in einem Farbbild mit 5 Mio. Pixeln hat jedes Pixel eine andere Farbe. Es kommen also  $K = 5 \cdot 10^6$  verschiedene Farben gleichverteilt vor, wodurch die Entropie maximal wird. Mit Gleichung 2.4 ist die maximale Entropie  $H_{max} = \log_2(5 \cdot 10^6) \approx 22,25$  bit/Symbol. Daraus ergibt sich für dieses Bild ein Speicheraufwand von  $5 \cdot 10^6 \times 23$  Bits  $\approx 13,7$  Megabyte. In der Realität sind weder die Farbwerte eines Bildes gleichverteilt, noch kommen alle  $256^3$  möglichen Farben tatsächlich vor. Daher ist die Entropie eines Bildes in der Regel deutlich kleiner.

Das Pixelbild aus Abbildung 2.5 ist ein anderer Extremfall. Es besitzt aufgrund der nur 2 verschiedenen Farben die Entropie  $H \approx 0,286$  bit/Symbol

**Tabelle 2.4:** Codebaum der Huffman-Codierung und resultierende Codewörter (Variante 1).



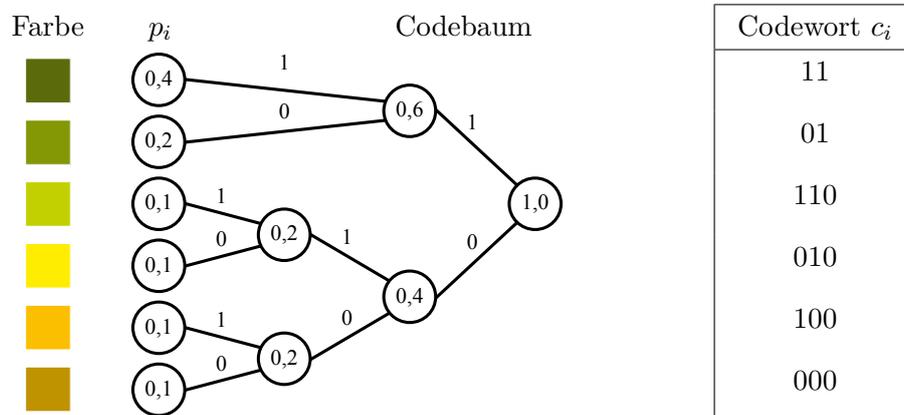
und kann somit durch eine mittlere Codewortlänge von 1 Bit dargestellt (z. B. Grün = 0 und Gelb = 1) und mit 20 Bits gespeichert werden. Ohne Kompression und als RGB-Bild gespeichert wären es  $20 \times 24$  Bits = 480 Bits.

Zur Realisierung einer solchen optimalen Zuordnung gibt es verschiedene Ansätze. David Huffman hat 1952 eine Entropiecodierung mit optimaler Codewortzuordnung vorgestellt [10]. Die Konstruktionsvorschrift wurde in [15, S. 36] zusammengefasst und ist hier wörtlich übernommen:

1. Betrachte alle Symbole als Blätter eines Codebaums und trage ihre Wahrscheinlichkeiten ein.
2. Fasse die beiden geringsten Wahrscheinlichkeiten zu einem Knoten zusammen und weise ihre Summe dem Knoten zu.
3. Beschrifte die neuen Zweige mit 0 bzw. 1.
4. Wenn die Wurzel des Baumes mit der Wahrscheinlichkeit  $p = 1.0$  erreicht ist, beende die Konstruktion.
5. Setze bei 2. fort.

Folgendes, an [15, S. 37] angelehnte, Beispiel verdeutlicht diese Codierung. Abbildung 2.7 zeigt ein Pixelbild mit sechs verschiedenen Farben, die unterschiedlich häufig auftreten. Tabelle 2.4 zeigt alle vorkommenden Farben, deren Auftretenswahrscheinlichkeiten  $p_i$ , den zugehörigen Huffman-Codebaum und die daraus resultierenden Codewörter  $c_i$ . Bei mehreren kleinsten Wahrscheinlichkeiten ist die Konstruktionsvorschrift nicht eindeutig. Dies stellt jedoch kein Problem dar, da verschiedene Varianten möglich sind und alle zu einer optimalen Codewortzuordnung führen [15, S. 36f.]. Tabelle 2.5 zeigt eine andere Variante des Codebaums. Hier ist die maximale Codewortlänge um 1 Bit kürzer als bei der ersten Variante. Was hier unterschiedlich erscheint ist tatsächlich äquivalent, da die mittlere Codewortlänge beider

**Tabelle 2.5:** Codebaum der Huffman-Codierung und resultierende Codewörter (Variante 2).



Varianten gleich ist, wie das Einsetzen in Gleichung 2.5 zeigt:

$$\bar{l}_1 = 0,4 \cdot 1 + 0,2 \cdot 3 + 0,1 \cdot 4 + 0,1 \cdot 4 + 0,1 \cdot 3 + 0,1 \cdot 3 = 2,4 \text{ Bits/Symbol},$$

$$\bar{l}_2 = 0,4 \cdot 2 + 0,2 \cdot 2 + 0,1 \cdot 3 + 0,1 \cdot 3 + 0,1 \cdot 3 + 0,1 \cdot 3 = 2,4 \text{ Bits/Symbol}.$$

Die Entropie des Bildes in Abbildung 2.7 ist nach Gleichung 2.3

$$H = -(0,4 \cdot \log_2 0,4 + 0,2 \cdot \log_2 0,2 + 4 \cdot 0,1 \cdot \log_2 0,1) \approx 2,32193 \text{ bit/Symbol}.$$

Die von Shannon gesetzten Grenzen in Ungleichung 2.6 werden also eingehalten.

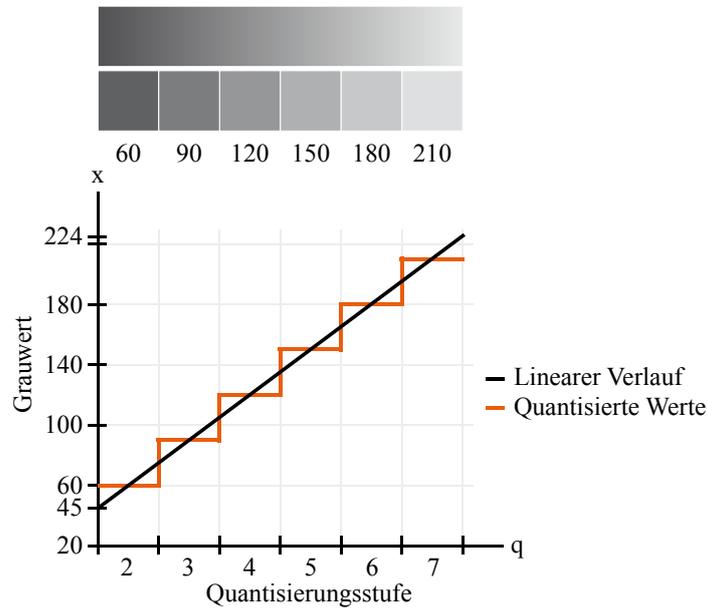
## 2.4 Quantisierung

Die stufenweise Zerlegung einer kontinuierlichen Funktion in eine diskrete Funktion wird als Quantisierung bezeichnet. Die Quantisierung kommt auch zum Einsatz, wenn der Wertebereich verkleinert werden soll. Es stehen dabei verschiedene Varianten zur Verfügung, die Zerlegungsstufen festzulegen. Bei der gleichmäßigen Quantisierung [15, S. 21f.], wie sie im Wesentlichen in der JPEG-Kompression zum Einsatz kommt, wird die Funktion in Intervalle mit fester Breite  $\Delta$  unterteilt. Die Quantisierung gilt als verlustbehaftet, da die Funktionswerte  $x$  intervallweise zu je einer Quantisierungsstufe  $q$  zusammengefasst werden und eine Rekonstruktion der ursprünglichen Werte nicht möglich ist. In [15, S. 22] ist die Vorschrift der gleichmäßigen Quantisierung (in ähnlicher Notation) festgelegt als

$$q = \left\lfloor \frac{|x|}{\Delta} + \frac{1}{2} \right\rfloor \cdot \text{sgn}(x) \quad (2.7)$$

und die zugehörigen Quantisierungswerte  $x_Q$  ergeben sich aus

$$x_Q = q \cdot \Delta. \quad (2.8)$$



**Abbildung 2.8:** Quantisierung eines linearen Grauwertverlaufs im Intervall  $[45, 224]$  mit Stufenbreite  $\Delta = 30$ . (Die Darstellung ist an [15, S. 22] angelehnt.)

**Tabelle 2.6:** Quantisierungsintervalle und -werte zu Abbildung 2.8.

$x$	$[45, 74]$	$[75, 104]$	$[105, 134]$	$[135, 164]$	$[165, 194]$	$[195, 224]$
$q$	2	3	4	5	6	7
$x_Q$	60	90	120	150	180	210

Abbildung 2.8 zeigt eine schematische Darstellung der gleichmäßigen Quantisierung eines Grauwertverlaufs. Im Verlauf befinden sich in aufsteigender Reihenfolge alle 180 Grauwerte  $x$  aus dem Intervall  $[45, 224]$ . Durch Quantisierung nach Vorschrift 2.7 mit fester Breite  $\Delta = 30$  entstehen sechs Quantisierungsstufen  $q$ , die jeweils 30 Grauwerte zusammenfassen. Letztere werden durch den Quantisierungswert  $x_Q$  ersetzt. Die genauen Quantisierungsintervalle zu Abbildung 2.8 sind in Tabelle 2.6 aufgelistet. In diesem Beispiel beginnen die aufsteigenden Quantisierungsstufen bei  $q = 2$ , da  $q = \lfloor \frac{45}{30} + \frac{1}{2} \rfloor = 2$  und  $x_Q = 2 \cdot 30 = 60$ . Auf diese Weise liegt der Quantisierungswert in der Mitte des Werteintervalls. Da nicht nur Verläufe quantisiert werden können, sind die Stufen nicht zwingend in aufsteigender Reihenfolge sondern ergeben sich aus den zu quantisierenden Werten. Für negative Werte kann auch die Quantisierungsstufe  $q$  negativ sein.

## Kapitel 3

# Diskrete Kosinustransformation

Die diskrete Kosinustransformation (DCT) gehört in der Bildverarbeitung neben einer Reihe anderer Transformationen, wie z. B. auch die Fouriertransformation, zu den Spektraltechniken [9, S. 309ff.]. Solche Transformationen werden unter anderem in der Kompression von Bild- und Videodaten eingesetzt, aber auch in Bereichen wie Nachrichtentechnik und Audioverarbeitung sind Spektraltechniken verbreitet. Sie werden im Allgemeinen dazu genutzt, ein Signal  $x(n)$  aus dem Zeit- bzw. Ortsbereich in einen Frequenzbereich  $X(k)$  zu transformieren. In [15, S. 108] ist eine Signaltransformation für ein Signal  $x(n)$  mit  $N$  Elementen (in ähnlicher Notation) allgemein formuliert als

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot a(k, n), \quad k = 0, 1, \dots, N-1. \quad (3.1)$$

Die Funktion  $a(k, n)$  wird als Transformationskern oder Basisfunktion bezeichnet [15, S. 108f.]. Die inverse oder Rücktransformation ist allgemein

$$x(n) = \sum_{k=0}^{N-1} X(k) \cdot \bar{a}(n, k), \quad n = 0, 1, \dots, N-1.$$

Spektraltechniken werden als Überlagerung von gewichteten Basisfunktionen interpretiert, wobei die Werte des zu transformierenden Signals als Gewichte eingesetzt werden. Da es sich bei digitalen Bildern um zweidimensionale Signale handelt, werden diese im Kontext der JPEG-Kompression mit einer zweidimensionalen Variante der DCT transformiert. Es ist sinnvoll, zuerst die eindimensionale DCT zu betrachten, da die DCT, wie die meisten mehrdimensionalen Signaltransformationen, durch Hintereinanderführung von eindimensionalen Transformationen umgesetzt werden kann. Diese Eigenschaft wird *separierbar* genannt [15, S. 108].

### 3.1 Eindimensionale DCT

Die Kosinustransformation arbeitet mit rein reellen Basisfunktionen und ist dadurch optimal für den Bild- und Videobereich geeignet. Die folgende Darstellung der DCT orientiert sich an den Notationen von [15, S. 112] und [9, S. 368]. Der Transformationskern  $a(k, n)$  der eindimensionalen DCT (1D-DCT) ist

$$a(k, n) = C_k \cdot \sqrt{\frac{2}{N}} \cdot \cos\left(\frac{(2n+1)k\pi}{2N}\right), \text{ mit } C_k = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } k = 0, \\ 1 & \text{für } k \neq 0. \end{cases}$$

Mit Gleichung 3.1 ist die 1D-DCT also definiert als

$$X(k) = C_k \cdot \sqrt{\frac{2}{N}} \cdot \sum_{n=0}^{N-1} x(n) \cdot \cos\left(\frac{(2n+1)k\pi}{2N}\right), \quad (3.2)$$

$$\text{mit } C_k = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } k = 0, \\ 1 & \text{für } k \neq 0 \end{cases}$$

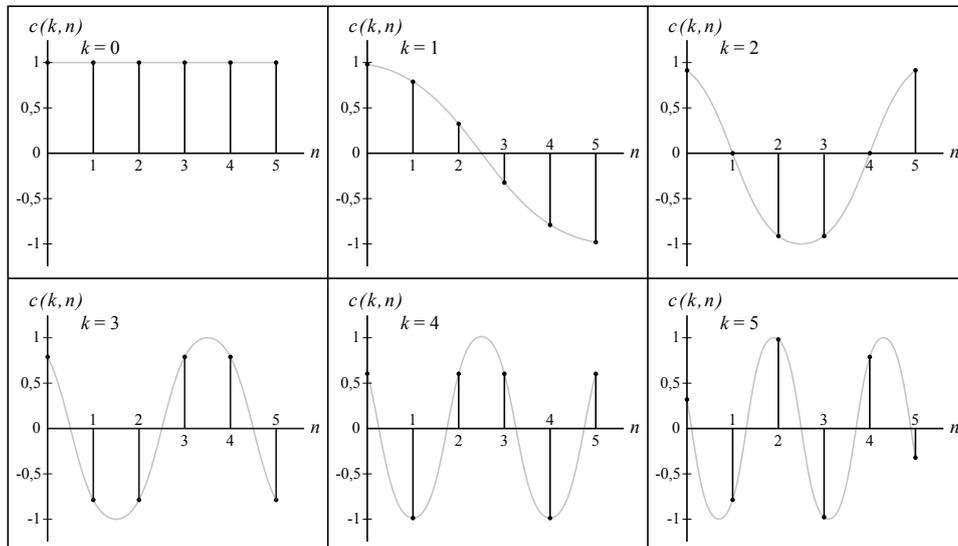
und die zugehörige inverse Transformation (1D-IDCT) lautet

$$x(n) = \sqrt{\frac{2}{N}} \cdot \sum_{k=0}^{N-1} C_k \cdot X(k) \cdot \cos\left(\frac{(2n+1)k\pi}{2N}\right), \quad (3.3)$$

$$\text{mit } C_k = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } k = 0, \\ 1 & \text{für } k \neq 0. \end{cases}$$

Auf den ersten Blick sehen die Kosinusanteile in Gleichungen 3.2 und 3.3 gleich aus, erzeugen jedoch unterschiedliche Frequenzen, da die Indizes unterschiedlich eingesetzt werden [9, S. 368]. Da die Faktoren  $\sqrt{\frac{2}{N}}$  und  $C_k$  bis auf den Fall  $k = 0$  konstant sind, können die Kosinusanteile der Basisfunktionen  $c(k, n) = \cos\left(\frac{(2n+1)k\pi}{2N}\right)$ , wie in [9, S. 368] beschrieben, gesondert betrachtet werden. Der Sonderfall  $k = 0$  erzeugt die lineare Funktion  $c(0, n) = 1$ , alle weiteren Fälle  $k \neq 0$  erzeugen Kosinusschwingungen der Periodenlänge  $\frac{2N}{k}$ . Abbildung 3.1 zeigt die Kosinusanteile der Basisfunktionen für  $N = 6$  Elemente. In Tabelle 3.1 sind die zugehörigen Kosinuswerte  $c(k, n)$  aufgelistet. In der Literatur werden die Kosinusanteile oft als kontinuierliche Funktionen dargestellt. Tatsächlich ist die DCT aber eine diskrete Funktion, weshalb diese Darstellung zu Verwirrungen führen kann.

Eine andere Möglichkeit, die Kosinusanteile darzustellen, sind sogenannte Basisbilder [13, S. 492ff.]. Im Kontext der JPEG-Kompression werden üblicherweise nur die Kosinusanteile des  $8 \times 8$  großen DCT-Blocks (siehe Abschnitt 4.2) als Basisbilder dargestellt. Die Darstellung in Bildern kann



**Abbildung 3.1:** Kosinusanteile  $c(k, n)$  der Basisfunktionen für  $N = 6$ . (Die Darstellung orientiert sich an [9, S. 369].)

**Tabelle 3.1:** Gerundete Kosinusanteile  $c(k, n)$  der Basisfunktionen für  $N = 6$ . (Siehe Abbildung 3.1)

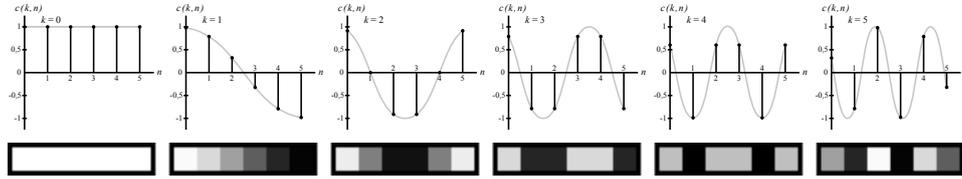
$k \backslash n$	0	1	2	3	4	5
0	1,00	1,00	1,00	1,00	1,00	1,00
1	0,97	0,71	0,26	-0,26	-0,71	-0,97
2	0,87	0,00	-0,87	-0,87	0,00	0,87
3	0,71	-0,71	-0,71	0,71	0,71	-0,71
4	0,50	-1,00	0,50	0,50	-1,00	0,50
5	0,26	-0,71	0,97	-0,97	0,71	-0,26

für  $N \times M$  große DCT-Blöcke verallgemeinert und für die gesamte Transformation fortgesetzt werden.

Zur Darstellung in Bildern werden die Kosinusanteile in natürliche Zahlen des Intervalls  $[0, 255]$  transformiert und als Grauwerte dargestellt. Die Werte von  $c(k, n)$  liegen im Intervall  $[-1, 1]$  und werden mit Hilfe der Vorschrift

$$gray = \left( \frac{c}{2} + \frac{1}{2} \right) \cdot 255 \quad (3.4)$$

in das Grauwertintervall  $[0, 255]$  übersetzt. Das obere Ende des Intervalls wird zu Weiß (255), das untere Ende zu Schwarz (0). Da die Intervalle über



**Abbildung 3.2:** Vergleich der Darstellungsarten der Kosinusanteile  $c(k, n)$  für  $N = 6$ . (Die Darstellung orientiert sich an [13, S. 494].)



**Abbildung 3.3:** Vergleich der Basisbilder der eindimensionalen Kosinusanteile  $c(k, n)$  und  $\bar{c}(n, k)$  für  $N = 4, 6, 8$ . (Die Darstellung orientiert sich an [13, S. 494].)

Null zentriert sind, werden Werte nahe Null als mittleres Grau dargestellt. Jedes der  $N$  Basisbilder enthält  $N$  Werte. Abbildung 3.2 zeigt einen direkten Vergleich der beiden Darstellungsarten. Für unterschiedliche Anzahlen von Elementen ergeben sich unterschiedliche Konstellationen von Basisbildern. Ebenso unterscheiden sich die Basisbilder der DCT sowie der inversen DCT deutlich voneinander. Ein Vergleich der Basisbilder der Kosinusanteile  $c(k, n)$  der 1D-DCT und  $\bar{c}(n, k)$  der 1D-IDCT für  $N = 4, 6, 8$  Elemente ist in Abbildung 3.3 dargestellt. Offensichtlich ergeben sich gewisse typische Grundmuster.

Zur genaueren Betrachtung und Darstellung der 1D-DCT in Bildern zerlegen wir nun Gleichung 3.2 in die Zwischenschritte

$$\begin{aligned}
 w(k) &= C_k \cdot \sqrt{\frac{2}{N}} \\
 c(k, n) &= \cos\left(\frac{(2n + 1)k\pi}{2N}\right) \\
 t(k, n) &= x(n) \cdot w(k) \cdot c(k, n) \\
 X(k) &= \sum_{n=0}^{N-1} t(k, n)
 \end{aligned}
 \tag{3.5}$$

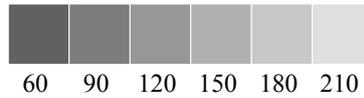


Abbildung 3.4: Sechs Grauwerte  $x(n)$ .

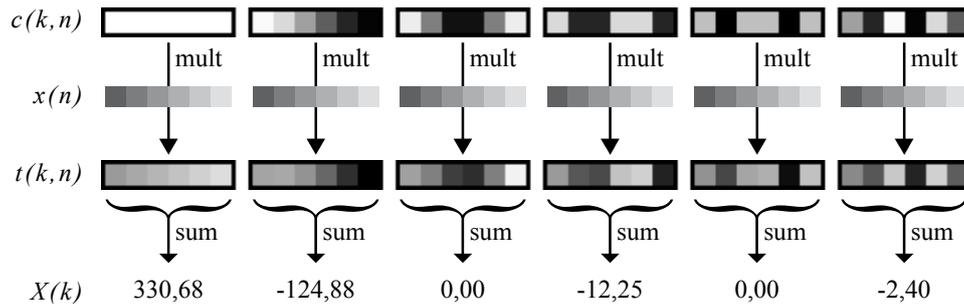


Abbildung 3.5: Zwischenschritte der 1D-DCT nach Zerlegung 3.5 und resultierende DCT-Koeffizienten.

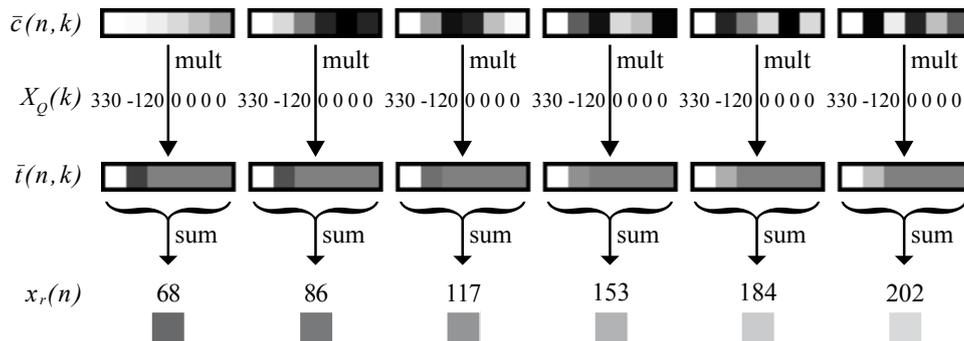


Abbildung 3.6: Zwischenschritte der 1D-IDCT nach Zerlegung 3.6 und resultierende Grauwerte.

und transformieren damit die Grauwerte  $x(n) = \{60, 90, 120, 150, 180, 210\}$  aus Abbildung 3.4. Die Kosinusanteile  $c(k, n)$  werden komponentenweise mit allen Werten des zu transformierenden Signals  $x(n)$  sowie den Konstanten  $w(k)$  gewichtet und werden im Folgenden als gewichtete Kosinusanteile  $t(k, n)$  bezeichnet. Durch die anschließende Summation ergeben sich die sogenannten DCT-Koeffizienten  $X(k)$  [15, S. 195]. Abbildung 3.5 zeigt eine schematische Darstellung dieser Zwischenschritte.

Während die einzelnen Signalwerte im Zeit- bzw. Ortsbereich zum Teil voneinander unabhängig sind, wird im Frequenzbereich jeder DCT-Koeffizient durch alle Signalwerte beeinflusst. Dadurch ist es möglich, im Frequenz-

**Tabelle 3.2:** Quantisierung der DCT-Koeffizienten  $X(k)$  von sechs Grauwerten  $x(n)$  nach Vorschrift 2.7 mit Stufenbreite  $\Delta = 30$ .

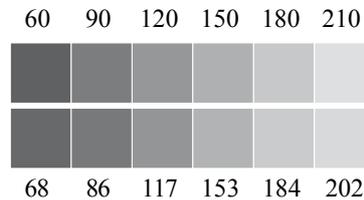
$X(k)$	330,68	-124,88	0	-12,25	0	-2,40
$q$	11	-4	0	0	0	0
$X_Q(k)$	<b>330</b>	<b>-120</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>

bereich verschiedene Operationen, z. B. eine Quantisierung (Abschnitt 2.4), durchzuführen, die sich bei der Rücktransformation auf alle Signalwerte auswirken. Die Rücktransformation wird von großen DCT-Koeffizienten stärker beeinflusst als von kleinen. Auch wenn kleine DCT-Koeffizienten durch die Quantisierung ausgelöscht werden, werden die ursprünglichen Werte durch die Rücktransformation gut approximiert [13, S. 481].

Um die Funktionsweise der inversen DCT näher zu betrachten ist es sinnvoll, zuerst eine Operation auf den sechs DCT-Koeffizienten  $X(k)$  aus Abbildung 3.5 durchzuführen und erst die veränderten DCT-Koeffizienten zurück zu transformieren [15, S. 198]. Dadurch wird sichtbar, dass sich kleine Abweichungen der DCT-Koeffizienten nur unwesentlich auf die Qualität der Rekonstruktion auswirken. Verwenden wir dazu die gleichmäßige Quantisierung nach Vorschrift 2.7. Die Schritte und Ergebnisse der Quantisierung der DCT-Koeffizienten mit Stufenbreite  $\Delta = 30$  sind in Tabelle 3.2 dargestellt. Anschließend werden die quantisierten DCT-Koeffizienten rücktransformiert. Dazu werden die Kosinusanteile  $\bar{c}(n, k)$  der 1D-IDCT komponentenweise mit allen rekonstruierten DCT-Koeffizienten  $X_Q(k)$  und den Konstanten  $w(k)$  gewichtet und anschließend summiert. Daraus entstehen die rekonstruierten Grauwerte  $x_r(n)$ . Die in Abbildung 3.6 gezeigte schematische Darstellung der 1D-IDCT folgt dem selben Prinzip wie die 1D-DCT und orientiert sich an folgender Zerlegung von Gleichung 3.3:

$$\begin{aligned}
 w(k) &= C_k \cdot \sqrt{\frac{2}{N}} \\
 \bar{c}(n, k) &= \cos\left(\frac{(2n+1)k\pi}{2N}\right) \\
 \bar{t}(n, k) &= X_Q(k) \cdot w(k) \cdot \bar{c}(n, k) \\
 x_r(n) &= \sum_{k=0}^{N-1} \bar{t}(n, k).
 \end{aligned} \tag{3.6}$$

Da Grauwerte durch natürliche Zahlen dargestellt werden, müssen die reellen Ergebnisse der IDCT gerundet werden. Werte, die über das Intervall hinausragen, werden auf 0 bzw. 255 gesetzt. Im direkten Vergleich der ursprüng-



**Abbildung 3.7:** Vergleich der ursprünglichen Grauwerte mit der Rekonstruktion aus Abbildung 3.6.

lichen Grauwerte mit den rekonstruierten Grauwerten (Abbildung 3.7) fällt auf, dass die Originalwerte zwar etwas verschoben wurden. Die Abweichungen sind jedoch zu gering, um optisch unterscheidbar zu sein.

Im Hinblick auf Kompression ist die DCT ein verlustfreies Verfahren. Da viele DCT-Koeffizienten nahe Null sind, bietet es sich allerdings an, diese zu quantisieren und einer Lauflängencodierung (Abschnitt 2.3.1) zu unterziehen. Erst durch die verlustbehaftete Quantisierung entsteht in obigem Beispiel ein Informationsverlust.

### 3.1.1 Grauwertberechnung

Die Transformation der Kosinusanteile in Grauwerte wurde bereits in Vorschrift 3.4 gezeigt. Die Darstellung der gewichteten Kosinusanteile ist aufwendiger, da die Intervalle sowohl kleiner als auch größer als  $[0, 255]$  sein können. Die gewichteten Kosinusanteile  $t(k, n)$  der 1D-DCT liegen noch überschaubar im Intervall  $[-255 \cdot \sqrt{\frac{2}{N}}, 255 \cdot \sqrt{\frac{2}{N}}]$ . Durch die große Intervallbreite der DCT-Koeffizienten  $X(k)$  liegen aber vor allem die gewichteten Kosinusanteile  $\tilde{t}(n, k)$  der 1D-IDCT in einem großen Intervall. Große Intervalle in nur 256 Graustufen darzustellen bedeutet, dass viele Werte zu einem zusammengefasst werden. Dieser Vorgang entspricht im wesentlichen einer Quantisierung und bedeutet Informationsverlust.

Verschiedene Versuche haben gezeigt, dass es für die Darstellung der gewichteten Kosinusanteile  $t(k, n)$  und  $\tilde{t}(n, k)$  in Grauwerten von Vorteil ist, das Intervall dynamisch an die tatsächlich vorhandenen Werte anzupassen und um Null zu zentrieren. Dazu werden der größte Wert  $t_{max}$  und der kleinste Wert  $t_{min}$  ermittelt. Der größere der Absolutbeträge von  $t_{max}$  und  $t_{min}$  bestimmt das Intervall  $[-t_{opt}, +t_{opt}]$ . Die Umrechnung in Grauwerte erfolgt nach der Vorschrift

$$\begin{aligned}
 t_{opt} &= \max(|t_{min}|, |t_{max}|) \\
 range &= 2 \cdot t_{opt} \\
 gray &= \left( \frac{t}{range} + \frac{1}{2} \right) \cdot 255.
 \end{aligned} \tag{3.7}$$

### 3.2 Zweidimensionale DCT

Die zweidimensionale DCT (2D-DCT) ergibt sich direkt aus der 1D-DCT und wird für zweidimensionale Signale, wie z. B. Bilder, eingesetzt. Durch die Separierbarkeit kann ein Bild erst zeilenweise und anschließend spaltenweise abgearbeitet werden [9, S. 371f.]. Bei einer zweifachen Hintereinanderausführung der 1D-DCT (Gleichung 3.2) entsteht die zweidimensionale Transformation

$$X(k, l) = C_k \cdot \sqrt{\frac{2}{N}} \cdot \sum_{n=0}^{N-1} \underbrace{\left[ C_l \cdot \sqrt{\frac{2}{M}} \cdot \sum_{m=0}^{M-1} x(n, m) \cdot \cos\left(\frac{(2m+1)l\pi}{2M}\right) \right]}_{\text{1D-DCT}} \cdot \cos\left(\frac{(2n+1)k\pi}{2N}\right).$$

In [9, S. 371] ist folgende gebräuchliche Notation der 2D-DCT (in ähnlicher Form) zu finden:

$$X(k, l) = \frac{2C_k C_l}{\sqrt{NM}} \cdot \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x(n, m) \cdot \cos\left(\frac{(2n+1)k\pi}{2N}\right) \cdot \cos\left(\frac{(2m+1)l\pi}{2M}\right),$$

$$\text{mit } C_k = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } k = 0, \\ 1 & \text{für } k \neq 0 \end{cases} \quad \text{und } C_l = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } l = 0, \\ 1 & \text{für } l \neq 0. \end{cases} \quad (3.8)$$

Die zweidimensionale inverse DCT (2D-IDCT) lautet

$$x(n, m) = \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} \frac{2C_k C_l}{\sqrt{NM}} \cdot X(k, l) \cdot \cos\left(\frac{(2n+1)k\pi}{2N}\right) \cdot \cos\left(\frac{(2m+1)l\pi}{2M}\right),$$

$$\text{mit } C_k = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } k = 0, \\ 1 & \text{für } k \neq 0 \end{cases} \quad \text{und } C_l = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } l = 0, \\ 1 & \text{für } l \neq 0. \end{cases} \quad (3.9)$$

In ähnlicher Weise wie die 1D-DCT kann auch die 2D-DCT in Zwischenschritten zerlegt und in Bildern dargestellt werden. Für einen Bildblock mit  $N \times M$  Pixeln entstehen  $N \times M$  Basisbilder mit je  $N \times M$  Werten. Wird ein Grauwertbild mit  $3 \times 3$  Pixeln (Abbildung 3.8) transformiert, so werden aus 9 Grauwertpixeln 9 DCT-Koeffizienten berechnet. Die Darstellung des zugehörigen Transformationsablaufs der 2D-DCT in Abbildung 3.9 entspricht



**Abbildung 3.8:** Grauwertbild mit 9 Pixeln und 3 Graustufen.

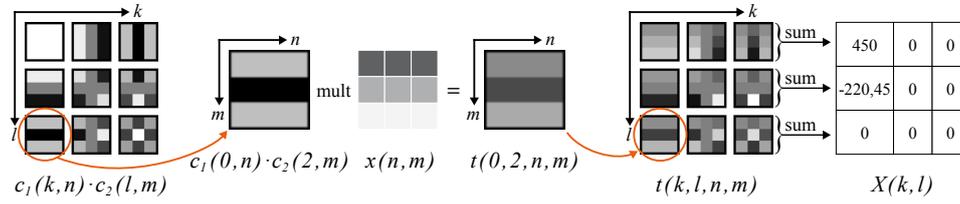
folgender Zerlegung von Gleichung 3.8:

$$\begin{aligned}
 w(k, l) &= \frac{2C_k C_l}{\sqrt{NM}} \\
 c_1(k, n) &= \cos\left(\frac{(2n+1)k\pi}{2N}\right) \\
 c_2(l, m) &= \cos\left(\frac{(2m+1)l\pi}{2M}\right) \\
 c(k, l, n, m) &= c_1(k, n) \cdot c_2(l, m) \\
 t(k, l, n, m) &= x(n, m) \cdot w(k, l) \cdot c(k, l, n, m) \\
 X(k, l) &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} t(k, l, n, m).
 \end{aligned} \tag{3.10}$$

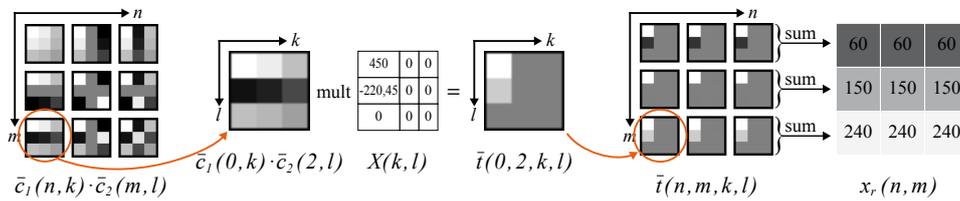
Äquivalent dazu kann auch die 2D-IDCT (Gleichung 3.9) in folgende Zwischenschritte zerlegt werden:

$$\begin{aligned}
 w(k, l) &= \frac{2C_k C_l}{\sqrt{NM}} \\
 \bar{c}_1(n, k) &= \cos\left(\frac{(2n+1)k\pi}{2N}\right) \\
 \bar{c}_2(m, l) &= \cos\left(\frac{(2m+1)l\pi}{2M}\right) \\
 \bar{c}(n, m, k, l) &= \bar{c}_1(n, k) \cdot \bar{c}_2(m, l) \\
 \bar{t}(n, m, k, l) &= X(k, l) \cdot w(k, l) \cdot \bar{c}(n, m, k, l) \\
 x_r(n, m) &= \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} \bar{t}(n, m, k, l).
 \end{aligned} \tag{3.11}$$

Der schematische Ablauf der 2D-IDCT ist in Abbildung 3.10 dargestellt. Zur Darstellung in Bildern werden die Zwischenergebnisse der Zerlegungen 3.10



**Abbildung 3.9:** Zwischenschritte der 2D-DCT nach Zerlegung 3.10. Zur einfacheren Darstellung wurde die Gewichtung  $w(k,l)$  in der Grafik ausgelassen.



**Abbildung 3.10:** Zwischenschritte der 2D-IDCT nach Zerlegung 3.11. Zur einfacheren Darstellung wurde die Gewichtung  $w(k,l)$  in der Grafik ausgelassen.

und 3.11, wie im eindimensionalen Fall, in Grauwerte des Intervalls  $[0, 255]$  übersetzt. Da die Kosinusanteile  $c(k,l,n,m)$  und  $\bar{c}(n,m,k,l)$  im Intervall  $[-1, 1]$  liegen, werden sie mit Vorschrift 3.4 in Grauwerte umgerechnet. Die gewichteten Kosinusanteile  $t(k,l,n,m)$  und  $\bar{t}(n,m,k,l)$  werden ebenfalls wie im eindimensionalen Fall mit Vorschrift 3.7 transformiert.

Für große Bilder, wie z. B. Fotos, wäre es zwar grundsätzlich möglich, alle DCT-Koeffizienten in einem Durchlauf zu berechnen. Jedoch ist es effizienter und in der Praxis üblich, große Bilder blockweise zu verarbeiten [13, S. 482]. Im Zuge der JPEG-Kompression werden Bilder in Blöcke mit  $8 \times 8$  Pixeln unterteilt.

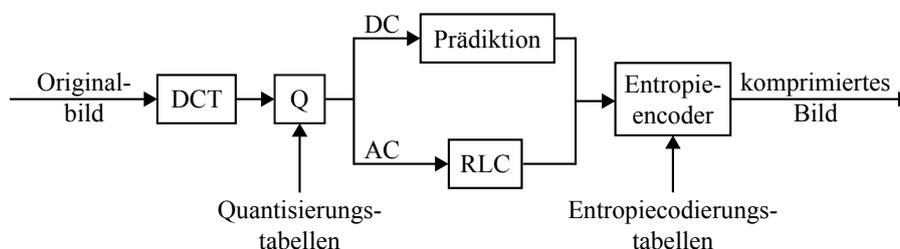
Neben Grauwertbildern können natürlich auch Farbbilder mit der DCT transformiert werden. Die Pixel eines RGB-Bildes (Abschnitt 2.2) beispielsweise werden durch je drei Farbwerte repräsentiert. Jede Farbkomponente wird wie ein Grauwertbild behandelt und gesondert transformiert.

## Kapitel 4

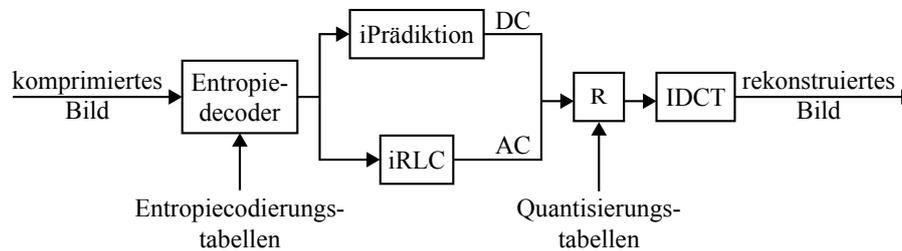
# Die JPEG-Kompression

Die JPEG-Kompression hat sich zum meistverwendeten Kompressionsverfahren für Grauwert- und Farbbilder etabliert. Dank der guten Kompressionsraten von durchschnittlich 1 : 16 findet JPEG in vielen Bereichen Anwendung [9, S. 18]. In handelsüblichen Digitalkameras und Smartphones beispielsweise ist diese Kompression bereits integriert und ermöglicht Anwendern eine effiziente Speicherung der Bilder. Die JPEG-Kompression kann bis zu 255 Komponenten mit einer Genauigkeit von 8 oder 12 Bits pro Pixel codieren [15, S. 194].

Das verhältnismäßig aufwendige Verfahren gliedert sich in mehrere Schritte, die in Abbildung 4.1 schematisch dargestellt sind. Das komprimierte Bild wird mit der Dateiendung *.jpg* oder *.jpeg* versehen und muss zur Betrachtung oder Weiterverarbeitung mit einem sogenannten Decoder dekomprimiert werden. Ein JPEG-Decoder ist in jeder gängigen Bildbetrachtungssoftware integriert und wird automatisch ausgeführt. Das Verfahren zur Dekompression ist in Abbildung 4.2 dargestellt. Die JPEG-Kompression ist ein verlustbehaftetes Verfahren, wodurch nur eine unvollständige Rekonstruktion des Originalbildes möglich ist. Die ursprünglichen Daten können nicht wieder hergestellt werden.



**Abbildung 4.1:** Ablauf der JPEG-Kompression [15, S. 195]. (Die Darstellung wurde modifiziert.)



**Abbildung 4.2:** Ablauf der JPEG-Dekompression [15, S. 195]. (Die Darstellung wurde modifiziert.)

## 4.1 Farbraumkonversion

Vor Beginn der Kompression werden, ausgehend von einem RGB-Bild, die drei Farbkomponenten mit Hilfe von Gleichung 2.1 in den YCbCr-Farbraum konvertiert (Abschnitt 2.1). Da das menschliche Auge für Helligkeitsunterschiede empfindlicher ist als für Farben, kann durch Unterabtastung [15, S. 18f.] der Farbdifferenzen eine Datenreduktion auf die Hälfte erzielt werden. Dabei wird für die Farbdifferenzsignale Cb und Cr die Anzahl der gespeicherten Werte reduziert. Gebräuchlich ist die 4:2:0-Abtastung [15, S. 188,194], die in horizontaler und vertikaler Richtung nur jeden zweiten Wert speichert. Für die Helligkeitskomponente Y stehen unverändert 8 Bits pro Pixel zur Verfügung. Für die Farbdifferenzsignale Cb und Cr wird pro vier Pixel nur je ein Wert mit 8 Bits gespeichert, wodurch der Speicheraufwand auf 2 Bits pro Pixel und Komponente reduziert wird.

## 4.2 Diskrete Kosinustransformation

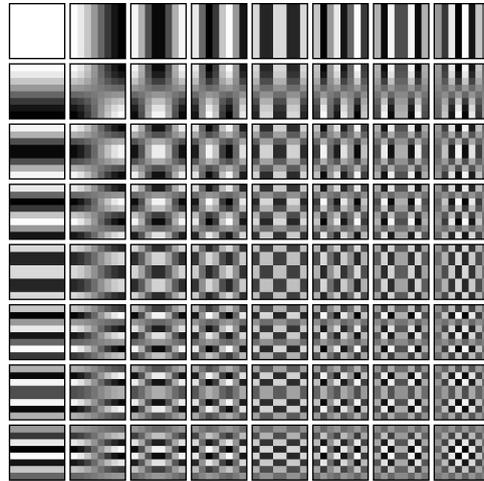
Zur Durchführung der 2D-DCT werden die drei Komponenten des YCbCr-Bildes gesondert behandelt. Jede Komponente wird in Blöcke zu je  $8 \times 8$  Pixeln unterteilt und blockweise transformiert [15, S. 194f.]. Die 2D-DCT (Gleichung 3.8) für  $N \times M = 8 \times 8$  Pixel ist

$$X(k, l) = \frac{C_k C_l}{4} \cdot \sum_{n=0}^7 \sum_{m=0}^7 x(n, m) \cdot \cos\left(\frac{(2n+1)k\pi}{16}\right) \cdot \cos\left(\frac{(2m+1)l\pi}{16}\right),$$

$$\text{mit } C_k = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } k = 0, \\ 1 & \text{für } k \neq 0 \end{cases} \quad \text{und } C_l = \begin{cases} \frac{1}{\sqrt{2}} & \text{für } l = 0, \\ 1 & \text{für } l \neq 0 \end{cases}$$

und erzeugt für jeden Block eine Koeffizientenmatrix mit 64 DCT-Koeffizienten. Der Koeffizient  $X(0, 0)$  wird Gleichanteil oder DC-Koeffizient<sup>1</sup> genannt,

<sup>1</sup>engl.: DC = direct current



**Abbildung 4.3:** Basisbilder der 2D-DCT für  $8 \times 8$  Werte [15, S. 196]. (Die konkrete Datei wurde generiert.)

alle weiteren Koeffizienten werden als Wechselanteile oder AC-Koeffizienten<sup>2</sup> bezeichnet [15, S. 195]. Diese Unterscheidung ist notwendig, da die DC-Koeffizienten nach der Quantisierung gesondert behandelt werden.

Abbildung 4.3 zeigt die 64 Basisbilder der 2D-DCT für einen  $8 \times 8$ -Block. In der linken oberen Ecke ist der Gleichanteil gut sichtbar [15, S. 197]. Durch Nachrechnen des DC-Koeffizienten

$$X(0,0) = \frac{1}{8} \cdot \sum_{n=0}^7 \sum_{m=0}^7 x(n,m) \cdot \cos(0) \cdot \cos(0) = \frac{1}{8} \cdot \sum_{n=0}^7 \sum_{m=0}^7 x(n,m)$$

wird offensichtlich, dass dieser das achtfache des Mittelwerts über alle Grauwerte des Blocks bildet. Es kann davon ausgegangen werden, dass die Mittelwerte benachbarter Blöcke nahe beieinander liegen, wodurch es sinnvoll ist, diese gesondert zu codieren [13, S. 530].

### 4.3 Quantisierung

Anschließend werden alle Koeffizientenmatrizen einer Quantisierung unterzogen, die der gleichmäßigen Quantisierung (Abschnitt 2.4) ähnlich ist. Zur Verbesserung der Resultate kann die Stufenbreite  $\Delta$  variiert werden [15, S. 197]. Dies geschieht mit Quantisierungstabellen  $Q(k,l)$ . Tabelle 4.1 zeigt die Quantisierungstabellen, die JPEG standardmäßig zur Verfügung stellt. Wenn andere Quantisierungstabellen eingesetzt werden, werden diese in die komprimierte Datei eingefügt.

<sup>2</sup>engl.: AC = alternating current

**Tabelle 4.1:** Quantisierungstabellen  $Q(k, l)$  für Helligkeit (a) und Farbdifferenzen (b) [15, S. 197].

(a)	(b)
16	17
11	18
10	24
16	47
24	99
40	99
51	99
61	99
12	18
12	21
14	26
19	66
26	99
58	99
60	99
55	99
14	24
13	26
16	56
24	99
40	99
57	99
69	99
56	99
14	47
17	66
22	99
29	99
51	99
87	99
80	99
62	99
18	99
22	99
37	99
56	99
68	99
109	99
103	99
77	99
24	99
35	99
55	99
64	99
81	99
104	99
113	99
92	99
49	99
64	99
78	99
87	99
103	99
121	99
120	99
101	99
72	99
92	99
95	99
98	99
112	99
100	99
103	99
99	99

**Tabelle 4.2:** Durch den Parameter  $R = 4$  erzeugte Quantisierungstabelle nach Vorschrift 4.1.

1	5	9	13	17	21	25	29
5	9	13	17	21	25	29	33
9	13	17	21	25	29	33	37
13	17	21	25	29	33	37	41
17	21	25	29	33	37	41	45
21	25	29	33	37	41	45	49
25	29	33	37	41	45	49	53
29	33	37	41	45	49	53	57

Eine andere Variante, die Stufenbreite zu variieren, ist in [13, S. 529] beschrieben. Mit der Vorschrift

$$Q(k, l) = 1 + (k + l) \cdot R \quad (4.1)$$

kann durch die Eingabe nur eines Parameters  $R$  eine Quantisierungstabelle mit wachsenden Werten generiert werden. Tabelle 4.2 zeigt die Quantisierungstabelle, die durch den Parameter  $R = 4$  erzeugt wird. Einerseits ist es von Vorteil, die hochfrequenten DCT-Koeffizienten im rechten unteren Teil der Koeffizientenmatrix stärker zu quantisieren, andererseits erspart diese Variante dem Benutzer der JPEG-Kompression den Aufwand, 64 Quantisierungswerte per Hand in eine Tabelle einzugeben.

Die Quantisierungsvorschrift der JPEG-Kompression ist in [15, S. 197] angegeben als

$$q(k, l) = \left\lfloor \frac{X(k, l)}{Q(k, l)} + \frac{1}{2} \cdot \text{sgn}(X(k, l)) \right\rfloor. \quad (4.2)$$

Eine genauere Analyse dieser Formel zeigt jedoch, dass dem Autor ein Feh-

ler unterlaufen ist. Um dies zu verdeutlichen betrachten wir das Beispiel in [15, S. 199] und rechnen die Quantisierung mit  $\Delta = 16$  für den DCT-Koeffizienten  $-5$  nach. Mit Vorschrift 4.2 ist der zugehörige quantisierte DCT-Koeffizient

$$q = \left\lfloor \frac{-5}{16} + \frac{1}{2} \cdot \text{sgn}(-5) \right\rfloor = \lfloor -0,8125 \rfloor = -1.$$

Laut [15, S. 199] sollte das Ergebnis jedoch 0 sein. Werden die Werte in Vorschrift 2.7 eingesetzt, so ist das Ergebnis richtig. Zur Analyse betrachten wir die Floor-Funktion genauer, die in [7, S. 787] folgendermaßen definiert ist:

$\lfloor x \rfloor$  ist die größte ganze Zahl kleiner gleich  $x$ .

Das bedeutet, dass zwar  $\lfloor 0,8125 \rfloor = 0$ , jedoch ist  $\lfloor -0,8125 \rfloor = -1$ . Das Ziel der Quantisierung ist es, so viele DCT-Koeffizienten wie möglich 0 zu setzen, um diese in weiterer Folge einer Lauflängencodierung zu unterziehen. Mit Vorschrift 4.2 werden jedoch viele DCT-Koeffizienten zu  $-1$ . Angelehnt an Vorschrift 2.7 muss die Quantisierungsvorschrift der JPEG-Kompression also lauten

$$q(k, l) = \left\lfloor \frac{|X(k, l)|}{Q(k, l)} + \frac{1}{2} \right\rfloor \cdot \text{sgn}(X(k, l)) \quad (4.3)$$

und Nachrechnen zeigt:

$$q = \left\lfloor \frac{|-5|}{16} + \frac{1}{2} \right\rfloor \cdot \text{sgn}(-5) = \lfloor 0,8125 \rfloor \cdot (-1) = 0.$$

## 4.4 Codierung

Nun werden die quantisierten DCT-Koeffizienten  $q(k, l)$  codiert, wobei die DC-Koeffizienten gesondert von den AC-Koeffizienten behandelt werden. In einem sogenannten prädiktiven Verfahren wird für alle DC-Koeffizienten die Differenz zum DC-Koeffizienten des vorhergehenden Blocks berechnet [15, S. 198,200]. Die Prädiktionsvorschrift lautet

$$DIFF = DC_i - PRED \quad \text{mit } PRED = \begin{cases} 0 & \text{für } i = 0, \\ DC_{i-1} & \text{für } i > 0. \end{cases}$$

Dabei bezeichnet  $i$  die Blocknummer,  $DC_i$  den DC-Koeffizienten des aktuellen Blocks und  $DIFF$  die Differenz zum Vorgänger. Anschließend werden die  $DIFF$ -Werte in Kategorien unterteilt und Huffman-codiert (Abschnitt 2.3.2). Die verwendeten Codes werden in die komprimierte Datei eingefügt. Der Binärcode wird dabei aus dem Huffman-Code der Kategorie sowie dem Index innerhalb der Kategorie zusammengesetzt. Tabelle 4.3 zeigt die Huffman-Codes, die vom JPEG-Standard vorgeschlagen werden.

**Tabelle 4.3:** Huffman-Codes zur Codierung der DC-Koeffizienten für Luminanz und Chrominanz [15, S. 200].

Kategorie	DIFF-Wert	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	100	110
4	-15, ..., -8, 8, ..., 15	101	1110
5	-31, ..., -16, 16, ..., 31	110	11110
6	-63, ..., -32, 32, ..., 63	1110	111110
7	-127, ..., -64, 64, ..., 127	11110	1111110
8	-255, ..., -128, 128, ..., 255	111110	11111110
9	-511, ..., -256, 256, ..., 511	1111110	111111110
10	-1023, ..., -512, 512, ..., 1023	11111110	1111111110
11	-2047, ..., -1024, 1024, ..., 2047	111111110	11111111110

Der *DIFF*-Wert  $-5$  beispielsweise liegt hier in Kategorie 3 mit Index 2 und wird folglich durch die Bitfolge 100 010 dargestellt. Der Index  $c$  innerhalb der Kategorie  $n$  des *DIFF*-Werts  $d$  kann durch die Vorschrift

$$\begin{aligned} \text{if } (d > 0) \quad c &= d \\ \text{else} \quad c &= 2^n - 1 + d \end{aligned}$$

berechnet werden [15, S. 200]. Die Kategorienummer gibt an, wieviele Bits zur Codierung der Indizes reserviert sind.

Die 63 AC-Koeffizienten jedes Blocks werden durch Zick-Zack-Abtastung in einen Vektor transformiert und dadurch in etwa der Größe nach sortiert. Da durch die Quantisierung viele AC-Koeffizienten zu Null werden, ist es sinnvoll, diese einer Lauflängencodierung (RLC) zu unterziehen. Diese wird zur Optimierung der Resultate mit einer Huffman-Codierung kombiniert [15, S. 200ff.]. Mit Tabelle 4.4 werden die AC-Koeffizienten in Kategorien eingeteilt, wobei nur AC-Koeffizienten ungleich Null betrachtet werden. Die jeweilige Lauflänge eines AC-Koeffizienten wird aus dem Abstand zum letzten AC-Koeffizienten ungleich Null berechnet. Aus der Kombination von Lauflänge und Kategorie werden die Huffman-Codes gebildet. Diese können für jedes Bild explizit berechnet oder aus der Tabelle entnommen werden, die der JPEG-Standard vorschlägt (siehe Tabellen 4.5, 4.6, 4.7). In jedem Fall wird die Tabelle mit der Codezuordnung in die komprimierte Datei eingefügt, wobei nur die tatsächlich verwendeten Codes notwendig sind.

**Tabelle 4.4:** Huffman-Kategorien zur Einteilung der AC-Koeffizienten [15, S. 201]. (Die Darstellung wurde modifiziert.)

Kategorie	AC-Wert
1	-1, 1
2	-3, -2, 2, 3
3	-7, ..., -4, 4, ..., 7
4	-15, ..., -8, 8, ..., 15
5	-31, ..., -16, 16, ..., 31
6	-63, ..., -32, 32, ..., 63
7	-127, ..., -64, 64, ..., 127
8	-255, ..., -128, 128, ..., 255
9	-511, ..., -256, 256, ..., 511
10	-1023, ..., -512, 512, ..., 1023

Analog zur Codierung der DC-Koeffizienten wird der Binärcode der AC-Koeffizienten aus dem Code für Lauflänge und Kategorie sowie dem Index innerhalb der Kategorie gebildet. Die Kategorie 0 ist für Sondersymbole wie z. B. ZRL („Zero Run Length“) oder EOB („End of Block“) reserviert [15, S. 201]. ZRL beschreibt die Lauflänge von 16 aneinander gereihten Nullen. Das Ende des Blocks ist erreicht, sobald keine AC-Koeffizienten ungleich Null mehr auftreten.

Beginnt der AC-Vektor z. B. mit dem Koeffizienten  $-18$ , so hat dieser die Lauflänge 0. Laut Tabelle 4.4 fällt  $-18$  in Kategorie 5 mit Index 13. Mit der Kombination von Lauflänge und Kategorie 0/5 wird dem AC-Koeffizienten  $-18$  der Huffman-Code 11010 aus Tabelle 4.5 zugewiesen. Der Code wird von der fünfstelligen Binärdarstellung des Indexes 13 gefolgt. Der AC-Koeffizient  $-18$  mit Lauflänge 0 wird somit durch die binäre Folge 11010 01101 codiert.

## 4.5 Decodierung und Rekonstruktion

Da die komprimierte Datei die quantisierten und codierten DCT-Koeffizienten enthält, kann sie nicht ohne Weiteres als Bild dargestellt, sondern muss von einem JPEG-Decoder dekomprimiert und rekonstruiert werden (siehe Abbildung 4.2). Anhand der, in der Datei gespeicherten, Huffman-Tabellen wird der Binärcode in DC-Koeffizienten und AC-Vektoren übersetzt und daraus anschließend die  $8 \times 8$ -Blöcke der quantisierten DCT-Koeffizienten  $q(k, l)$  gebildet. Die Rekonstruktion der DCT-Koeffizienten  $X_Q(k, l)$  erfolgt durch komponentenweise Multiplikation der quantisierten DCT-Koeffizienten

**Tabelle 4.5:** Huffman-Codes zur Codierung der AC-Koeffizienten (Teil 1)  
[15, S. 202f.]. (Die Darstellung wurde modifiziert und umsortiert.)

Lauflänge/ Kategorie	Codewort	Lauflänge/ Kategorie	Codewort
0/0	(EOB) 1010		
0/1	00	3/1	111010
0/2	01	3/2	111110111
0/3	100	3/3	111111110101
0/4	1011	3/4	1111111110001111
0/5	11010	3/5	11111111110010000
0/6	1111000	3/6	11111111110010001
0/7	11111000	3/7	11111111110010010
0/8	1111110110	3/8	11111111110010011
0/9	1111111110000010	3/9	11111111110010100
0/10	1111111110000011	3/10	11111111110010101
1/1	1100	4/1	111011
1/2	11011	4/2	1111111000
1/3	1111001	4/3	1111111110010110
1/4	111110110	4/4	11111111110010111
1/5	11111110110	4/5	11111111110011000
1/6	1111111110000100	4/6	11111111110011001
1/7	1111111110000101	4/7	11111111110011010
1/8	1111111110000110	4/8	11111111110011011
1/9	1111111110000111	4/9	11111111110011100
1/10	1111111110001000	4/10	11111111110011101
2/1	11100	5/1	1111010
2/2	11111001	5/2	11111110111
2/3	1111110111	5/3	1111111110011110
2/4	111111110100	5/4	11111111110011111
2/5	1111111110001001	5/5	11111111110100000
2/6	1111111110001010	5/6	11111111110100001
2/7	1111111110001011	5/7	11111111110100010
2/8	1111111110001100	5/8	11111111110100011
2/9	1111111110001101	5/9	11111111110100100
2/10	1111111110001110	5/10	11111111110100101

**Tabelle 4.6:** Huffman-Codes zur Codierung der AC-Koeffizienten (Teil 2)  
[15, S. 202f.]. (Die Darstellung wurde modifiziert und umsortiert.)

Lauflänge/ Kategorie	Codewort	Lauflänge/ Kategorie	Codewort
6/1	1111011	9/1	111111001
6/2	11111110110	9/2	111111110111110
6/3	111111110100110	9/3	111111110111111
6/4	111111110100111	9/4	111111111000000
6/5	111111110101000	9/5	111111111000001
6/6	111111110101001	9/6	111111111000010
6/7	111111110101010	9/7	111111111000011
6/8	111111110101011	9/8	111111111000100
6/9	111111110101100	9/9	111111111000101
6/10	111111110101101	9/10	111111111000110
7/1	11111010	10/1	111111010
7/2	11111110111	10/2	111111111000111
7/3	111111110101110	10/3	111111111001000
7/4	111111110101111	10/4	111111111001001
7/5	111111110110000	10/5	111111111001010
7/6	111111110110001	10/6	111111111001011
7/7	111111110110010	10/7	111111111001100
7/8	111111110110011	10/8	111111111001101
7/9	111111110110100	10/9	111111111001110
7/10	111111110110101	10/10	111111111001111
8/1	111111000	11/1	1111111001
8/2	11111111000000	11/2	111111111010000
8/3	111111110110110	11/3	111111111010001
8/4	111111110110111	11/4	111111111010010
8/5	111111110111000	11/5	111111111010011
8/6	111111110111001	11/6	111111111010100
8/7	111111110111010	11/7	111111111010101
8/8	111111110111011	11/8	111111111010110
8/9	111111110111100	11/9	111111111010111
8/10	111111110111101	11/10	111111111011000

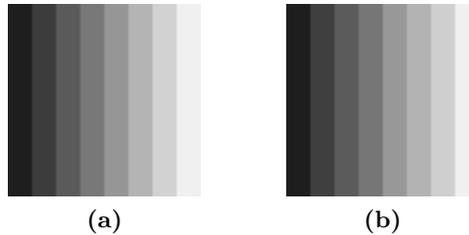
**Tabelle 4.7:** Huffman-Codes zur Codierung der AC-Koeffizienten (Teil 3) [15, S. 202f.]. (Die Darstellung wurde modifiziert und umsortiert.)

Lauflänge/ Kategorie	Codewort	Lauflänge/ Kategorie	Codewort
12/1	1111111010	14/1	1111111111101011
12/2	1111111111011001	14/2	1111111111101100
12/3	1111111111011010	14/3	1111111111101101
12/4	1111111111011011	14/4	1111111111101110
12/5	1111111111011100	14/5	1111111111101111
12/6	1111111111011101	14/6	1111111111100000
12/7	1111111111011110	14/7	1111111111100001
12/8	1111111111011111	14/8	1111111111100010
12/9	1111111111000000	14/9	1111111111100011
12/10	1111111111000001	14/10	1111111111100100
13/1	11111111000	15/1	111111111110101
13/2	1111111111000010	15/2	1111111111101011
13/3	1111111111000011	15/3	111111111110111
13/4	1111111111000100	15/4	11111111111000
13/5	1111111111000101	15/5	111111111110001
13/6	1111111111000110	15/6	111111111110010
13/7	1111111111000111	15/7	111111111110011
13/8	1111111111001000	15/8	1111111111100100
13/9	1111111111001001	15/9	1111111111100101
13/10	1111111111001010	15/10	1111111111100110
		15/0	(ZRL) 1111111001

mit den Werten der Quantisierungstabelle  $Q(k, l)$  nach der Vorschrift

$$X_Q(k, l) = q(k, l) \cdot Q(k, l) \quad (4.4)$$

bzw. durch Multiplikation aller quantisierten DCT-Koeffizienten mit der Stufenbreite  $\Delta$  [15, S. 198]. Die rekonstruierten DCT-Koeffizienten  $X_Q(k, l)$  werden mit der inversen DCT (2D-IDCT) in Grauwerte transformiert. Die rekonstruierten Grauwerte decken sich auf Grund der verlustbehafteten Kompression nicht mit den Grauwerten des Originalbildes.



**Abbildung 4.4:** Originalbild (a) und rekonstruiertes Bild (b) zu Abbildung 4.5.

## 4.6 Vom Bild zum Binärkode

Um den vollständigen Ablauf der JPEG-Kompression vom Originalbild bis zum Binärkode und zurück zum Bild zu verdeutlichen, ist dies in Abbildung 4.5 an konkreten Zahlen beispielhaft dargestellt. Das Beispiel orientiert sich an [15, S. 199,204]. Ein  $8 \times 8$  Pixel großer Block mit einem Grauwertverlauf wird mit der 2D-DCT (Abschnitt 4.2) transformiert und mit Stufenbreite  $\Delta = 30$  gleichmäßig quantisiert (Abschnitt 4.3). Wie in Abschnitt 4.4 beschrieben, werden im letzten Schritt die quantisierten DCT-Koeffizienten codiert. Die Rekonstruktion (Abschnitt 4.5) ist auf der rechten Seite der Abbildung zu sehen. Abbildung 4.4 zeigt das Originalbild und das zugehörige rekonstruierte Bild.

Anhand des Binärkodes in der letzten Zeile von Abbildung 4.5 lässt sich der Kompressionsfaktor für diesen Bildblock berechnen. Würden alle 64 Grauwerte durch je 1 Byte dargestellt, wären für diesen Block 64 Byte notwendig. Der komprimierte Binärkode beansprucht 46 Bits = 5,75 Byte. Daraus ergibt sich ein Kompressionsfaktor von  $5,75 : 64 \approx 1 : 11$ .

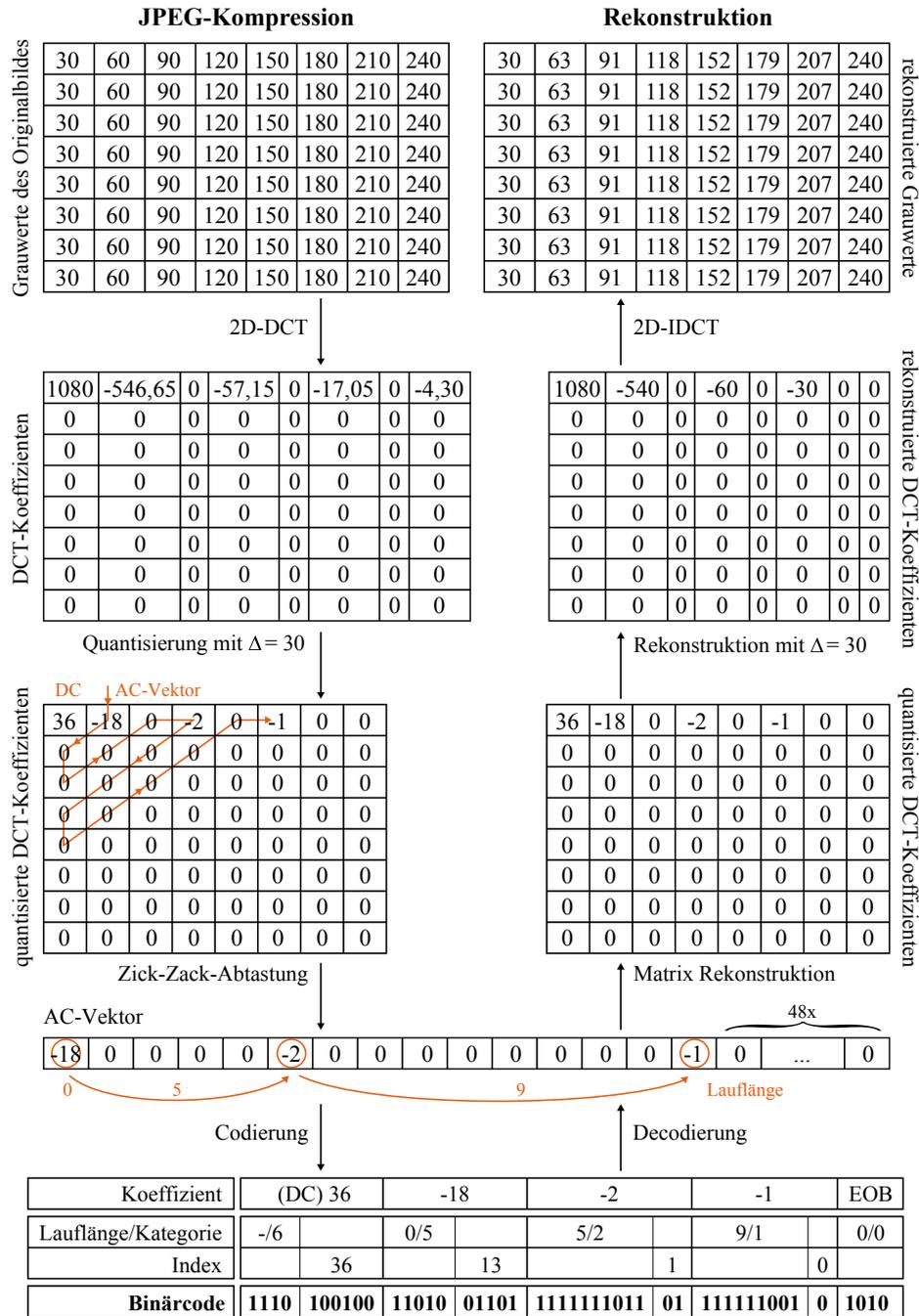


Abbildung 4.5: Kompression und Rekonstruktion eines 8x8-Blocks.

## Kapitel 5

# Der JPEG-Simulator

Im Zuge dieser Arbeit wurde ein Programm zur Simulation der JPEG-Kompression entwickelt, welches auf der beiliegenden CD zu finden ist. Ziel der Simulation ist es, die Auswirkung der DCT und der Quantisierung auf die Qualität des rekonstruierten Bildes sowie die Anzahl der zu codierenden Werte zu analysieren. Anhand dieser Beobachtungen können Zusammenhänge untersucht und die Eigenschaften der JPEG-Kompression veranschaulicht werden. Da die  $8 \times 8$ -Blöcke eines Bildes voneinander unabhängig transformiert werden, wird die Simulation für einen  $8 \times 8$  Pixel großen Grauwertblock durchgeführt, anstatt ganze Bilder zu komprimieren. Auf diese Weise sind die Ergebnisse überschaubar und die Auswirkungen auf die einzelnen Pixel gut sichtbar.

Die Simulation berechnet die wesentlichen Zwischenschritte der 2D-DCT und 2D-IDCT, wie sie in den Zerlegungen 3.10 und 3.11 beschrieben sind, sowie verschiedene Quantisierungsarten. Die Huffman- und Lauflängencodierung sind nicht Teil der Simulation, da sie verlustfreie Codierungen sind und sich daher nicht vorrangig auf das qualitative Ergebnis der Kompression auswirken. Die Darstellung der Zwischenschritte erfolgt wie in Kapitel 3 in Bildern und Tabellen. Die Algorithmen entsprechen ebenfalls den Formeln und Vorschriften dieses Kapitels und wurden allgemein gehalten. Dadurch ist es leicht möglich, den JPEG-Simulator für ähnliche Anwendungen zu modifizieren oder um Funktionen zu erweitern. Der vollständige Quellcode ist auf der CD sowie in Anhang A zu finden.

### 5.1 Beschreibung der Eingabe

Die Benutzeroberfläche bietet verschiedene Eingabeoptionen an und ist in Abbildung 5.1 dargestellt. Zur Auswahl stehen sechs verschiedene Grauwertbilder mit je  $8 \times 8$  Pixeln, die einem DCT-Block der JPEG-Kompression entsprechen. Farbbilder werden hier nicht explizit behandelt, da die drei Farbkomponenten unabhängig voneinander und einzeln komprimiert wer-

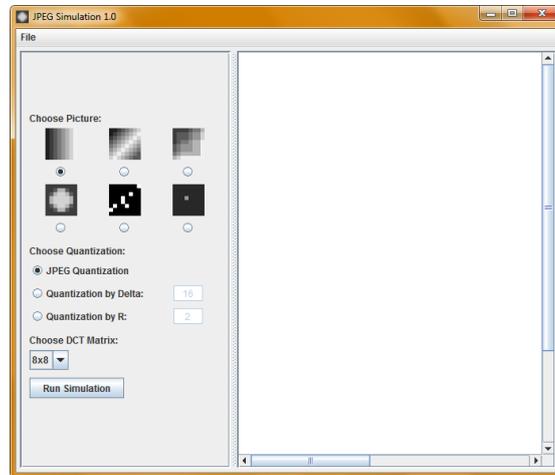


Abbildung 5.1: Eingabe des JPEG-Simulators.

den und sich somit mathematisch nicht von Grauwertbildern unterscheiden.

Zur Quantisierung stehen drei Varianten zur Verfügung. Die Auswahl „JPEG Quantization“ steht für die Quantisierung mit variabler Stufenbreite nach Vorschrift 4.3. Die implementierte Quantisierungstabelle entspricht Tabelle 4.1 (a), wie sie vom JPEG-Standard vorgeschlagen wird. Die Auswahl „Quantization by Delta“ aktiviert das Eingabefeld für die Stufenbreite  $\Delta$  der gleichmäßigen Quantisierung nach Vorschrift 2.7. Die Auswahl „Quantization by R“ aktiviert das Eingabefeld für den Parameter  $R$ , der eine Quantisierungstabelle nach Vorschrift 4.1 generiert. Zulässige Parameter für  $\Delta$  und  $R$  sind alle natürlichen Zahlen ohne Null. Wird jedoch  $\Delta$  bzw.  $R$  größer als der größte DCT-Koeffizient gesetzt, werden alle DCT-Koeffizienten zu Null quantisiert und die Rekonstruktion ist ein einheitliches Grau.

Der Menüpunkt „Choose DCT Matrix“ bietet drei verschiedene Blockgrößen an. Der JPEG-Standard schreibt zwar eine Blockgröße von  $8 \times 8$  Pixeln vor, jedoch ist es interessant zu beobachten, wie sich die Blockgröße auf die Qualität der Kompression auswirkt. Bei einer Blockgröße von  $2 \times 2$  wird das Grauwertbild in sechzehn Blöcke unterteilt, die einzeln transformiert werden.

Der Button „Run Simulation“ startet die Simulation, stellt die Zwischenschritte und Ergebnisse in einzelnen Ausgabefenstern dar und aktiviert die zu den Fenstern gehörende Liste, die ein Aus- und Einblenden der Fenster ermöglicht. Auch Fenster, die geschlossen wurden, können auf diese Weise wieder hergestellt werden. Fenster, die verschoben oder deren Größe verändert wurden, können durch Doppelklicken der Titelleiste auf ihre ursprüngliche Position und Größe zurückgesetzt werden. Nach Veränderung der Eingabeparameter muss der Button „Run Simulation“ erneut betätigt werden, um die Ergebnisse in den Fenstern zu aktualisieren.

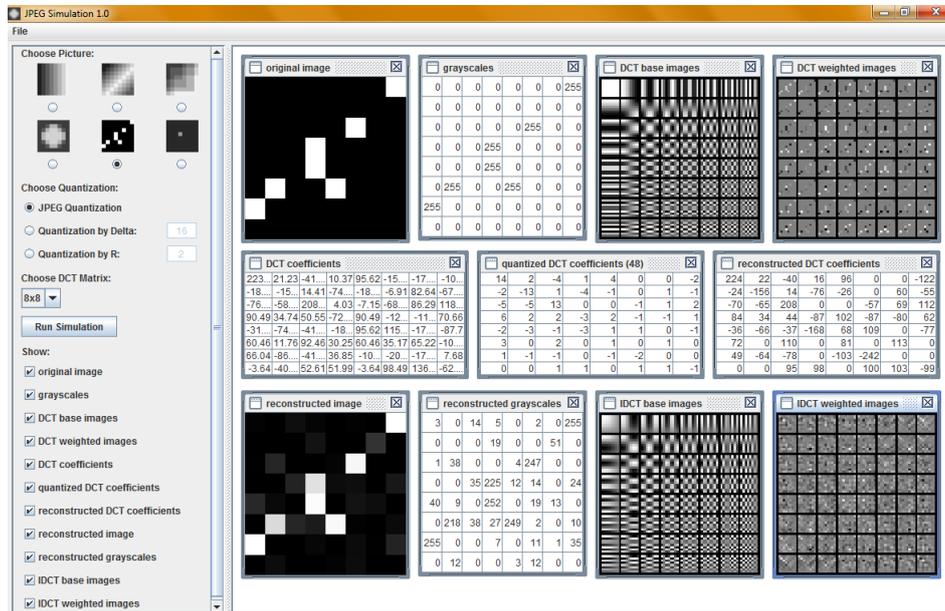


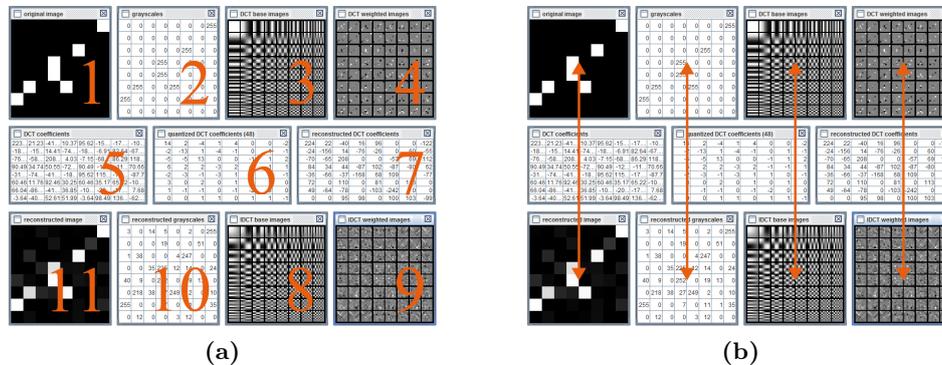
Abbildung 5.2: Ausgabe des JPEG-Simulators.

## 5.2 Beschreibung der Ausgabe

Die Ausgabe der Simulation ist in Abbildung 5.2 dargestellt. In elf Fenstern sind die Zwischenschritte der 2D-DCT nach Zerlegung 3.10, der Quantisierung, sowie der 2D-IDCT nach Zerlegung 3.11 grafisch und tabellarisch abgebildet. Die Reihenfolge der Fenster orientiert sich nicht an der mathematischen Abfolge der JPEG-Kompression, sondern ist für bessere Vergleichbarkeit optimiert. Da sowohl die 2D-DCT als auch die 2D-IDCT dargestellt sind, sind die vergleichbaren Schritte untereinander angeordnet. Die Nummerierung in Abbildung 5.3 (a) entspricht der mathematischen Reihenfolge. Abbildung 5.3 (b) zeigt, welche Schritte miteinander verglichen werden sollen. Das Originalbild beispielsweise ist direkt mit dem rekonstruierten Bild vergleichbar. Selbiges gilt für die Basisbilder der 2D-DCT und 2D-IDCT, da erst im direkten Vergleich ersichtlich ist, wie sich die Basisbilder unterscheiden.

Folgende Zwischenschritte und Ergebnisse sind in den elf Fenstern dargestellt (Die Nummerierung entspricht den Zahlen in Abbildung 5.3 (a)):

1. „original image“  
zeigt das Originalbild  $x(n, m)$ .
2. „grayscales“  
zeigt die Grauwerte des Originalbildes  $x(n, m)$  in tabellarischer Form.



**Abbildung 5.3:** Mathematische Reihenfolge der Zwischenschritte (a) und korrespondierende Zwischenschritte (b).

3. „DCT base images“  
zeigt die Basisbilder  $c(k, l, n, m)$  der 2D-DCT nach Zerlegung 3.10, wobei die Basisbilder unabhängig vom Originalbild sind.
4. „DCT weighted images“  
zeigt die gewichteten Basisbilder  $t(k, l, n, m)$  der 2D-DCT nach Zerlegung 3.10 und Gewichtung mit den Werten des Originalbildes  $x(n, m)$ .
5. „DCT coefficients“  
zeigt die DCT-Koeffizienten  $X(k, l)$  in tabellarischer Form nach vollständiger Transformation des Originalbildes mit der 2D-DCT nach Gleichung 3.8.
6. „quantized DCT coefficients (18)“  
zeigt die quantisierten DCT-Koeffizienten  $q(k, l)$  nach Vorschrift 4.3 in tabellarischer Form. Bei Wahl des Parameters „Quantization by Delta“ wird in der Vorschrift die Quantisierungstabelle  $Q(k, l)$  gegen  $\Delta$  getauscht. Der Parameter „Quantization by R“ generiert eine Quantisierungstabelle nach Vorschrift 4.1. Die Zahl in Klammern gibt Auskunft über die Anzahl der quantisierten Koeffizienten ungleich Null.
7. „reconstructed DCT coefficients“  
zeigt die rekonstruierten DCT-Koeffizienten  $X_Q(k, l)$  nach Vorschrift 4.4 in tabellarischer Form. Bei Wahl des Parameters „Quantization by Delta“ wird in der Vorschrift die Quantisierungstabelle  $Q(k, l)$  gegen  $\Delta$  getauscht. Der Parameter „Quantization by R“ generiert eine Quantisierungstabelle nach Vorschrift 4.1.
8. „IDCT base images“  
zeigt die Basisbilder  $\bar{c}(n, m, k, l)$  der 2D-IDCT nach Zerlegung 3.11, wobei die Basisbilder unabhängig von den DCT-Koeffizienten sind.

9. „IDCT weighted images“  
zeigt die gewichteten Basisbilder  $\bar{t}(n, m, k, l)$  der 2D-IDCT nach Zerlegung 3.11 und Gewichtung mit den rekonstruierten DCT-Koeffizienten  $X_Q(k, l)$ .
10. „reconstructed grayscales“  
zeigt die rekonstruierten Grauwerte  $x_r(n, m)$  in tabellarischer Form nach vollständiger Transformation der rekonstruierten DCT-Koeffizienten  $X_Q(k, l)$  mit der 2D-IDCT nach Gleichung 3.9.
11. „reconstructed image“  
zeigt das rekonstruierte Grauwertbild  $x_r(n, m)$ .

Die Werte der Basisbilder sowie der gewichteten Basisbilder werden nur optisch dargestellt, da es sich für einen  $8 \times 8$ -Block um  $8^4 = 4096$  Werte handelt. Diese tabellarisch aufzulisten wäre zu unübersichtlich. Es bietet sich aber an, diese Werte in Bildern darzustellen.

### 5.3 Eigenschaften der JPEG-Kompression

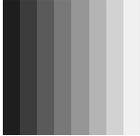
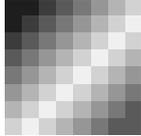
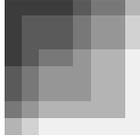
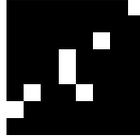
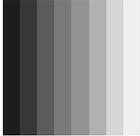
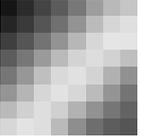
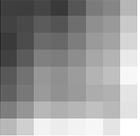
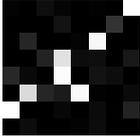
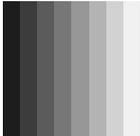
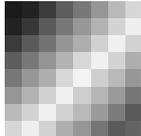
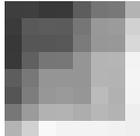
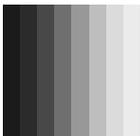
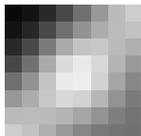
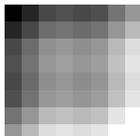
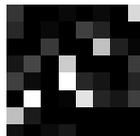
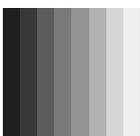
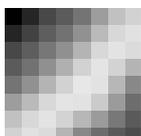
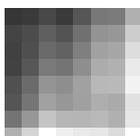
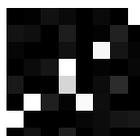
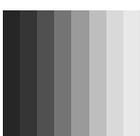
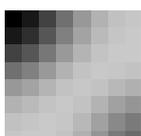
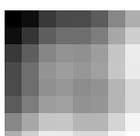
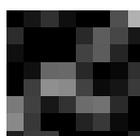
Die folgende Analyse beschreibt die grundlegenden Eigenschaften der JPEG-Kompression und gibt einen Einblick in die Einsatzmöglichkeiten des JPEG-Simulators. Die Ergebnisse decken sich im Wesentlichen mit den Erkenntnissen in [15, S. 198] und [13, S. 485f., 520ff.].

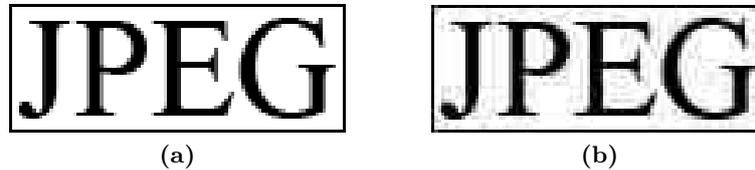
Da die 2D-DCT in der Regel nicht verändert werden kann, wird die Qualität der JPEG-Kompression in erster Linie von der Quantisierung beeinflusst. Qualität meint dabei einerseits die optische Qualität der Rekonstruktion und andererseits den Kompressionsfaktor. Wird die Quantisierung auf  $\Delta = 1$  gesetzt, wird diese praktisch deaktiviert, da nur die Nachkommastellen der DCT-Koeffizienten abgeschnitten bzw. gerundet werden. Eine Deaktivierung der Quantisierung ist im JPEG-Simulator nicht implementiert, da die 2D-DCT eine exakte Transformation ohne Datenverlust ist. Die auf zwei Nachkommastellen gerundeten DCT-Koeffizienten werden ohnehin im Fenster „DCT coefficients“ aufgelistet und eine Rekonstruktion würde (bis auf kleine Rundungsfehler) dem Originalbild entsprechen.

Eine optimal gewählte Quantisierung liefert eine qualitativ gute Rekonstruktion bei einem hohen Kompressionsfaktor. Die Wahl der Quantisierung ist vom jeweiligen Bild und dessen Eigenschaften abhängig. Die selbe Quantisierung kann für das eine Bild optimal und für ein anderes schlecht sein. Tabelle 5.1 zeigt vier verschiedene  $8 \times 8$ -Bildblöcke und deren Rekonstruktionen nach unterschiedlicher Quantisierung sowie die jeweilige Anzahl der zu codierenden DCT-Koeffizienten ungleich Null. Die Anzahl der DCT-Koeffizienten ungleich Null beeinflusst maßgeblich den Kompressionsfaktor – je weniger Koeffizienten, desto effizienter die Kompression.

Die JPEG-Kompression ist vor allem für Fotos optimiert, die in der Regel keine harten Kanten sondern hauptsächlich Verläufe enthalten [13, S. 520].

**Tabelle 5.1:** Originalbilder und rekonstruierte Bilder nach JPEG-Kompression mit unterschiedlicher Quantisierung. Die Zahl unterhalb jedes Bildes beschreibt die Anzahl der zu codierenden DCT-Koeffizienten ungleich Null.

Originalbild				
Quantisierung mit ...				
JPEG-Tabelle				
	3	18	17	48
$\Delta = 10$				
	4	44	43	61
$\Delta = 150$				
	2	7	5	23
$R = 10$				
	3	13	15	45
$R = 50$				
	2	6	6	6



**Abbildung 5.4:** Vergleich eines unkomprimierten Schriftzugs (a) mit dessen Rekonstruktion nach einer JPEG-Kompression (b).

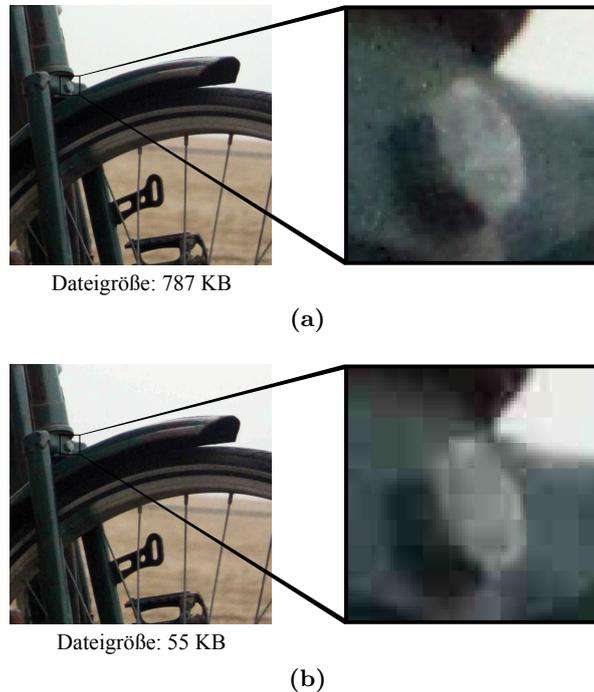
Wie an den ersten beiden Bildern in Tabelle 5.1 deutlich zu sehen ist, liefert die JPEG-Kompression für Verläufe auch bei starker Quantisierung sehr gute Ergebnisse. Ein vertikaler oder horizontaler Verlauf kann sogar aus nur zwei DCT-Koeffizienten annehmbar rekonstruiert werden. Auch für einen diagonalen Verlauf sind etwa sieben DCT-Koeffizienten bereits ausreichend. In einem  $8 \times 8$ -Block sind die Abweichungen vom Originalbild zwar deutlich sichtbar, jedoch fallen diese in einem Foto mit 5 Mio. Pixeln kaum auf.

Für Bilder mit harten Kanten, wie z. B. Grafiken oder Logos, fallen die Kompressionsergebnisse deutlich schlechter aus. Am dritten Bild in Tabelle 5.1 ist gut sichtbar, dass die Quantisierungstabelle, die vom JPEG-Standard vorgeschlagen wird (Tabelle 4.1 (a)), für harte Kanten ungeeignet ist. Eine gleichmäßige Quantisierung mit  $\Delta = 10$  liefert hier zwar ein deutlich besseres optisches Ergebnis, jedoch bei einem niedrigen Kompressionsfaktor.

Ähnlich schlecht funktioniert die JPEG-Kompression für reine Schwarz-Weiß-Bilder, wie z. B. Schriften. Eine gute Rekonstruktion ist nur bei schwacher Quantisierung und damit niedrigem Kompressionsfaktor möglich. Abbildung 5.4 zeigt den Vergleich eines unkomprimierten Schriftzuges mit dessen Rekonstruktion nach einer JPEG-Kompression. Durch die Kompression verliert das Bild an Schärfe und es entstehen Artefakte.

Werden die DCT-Koeffizienten zu stark quantisiert, entstehen sogenannte Blockartefakte [15, S. 198]. Da die  $8 \times 8$ -Blöcke unabhängig voneinander komprimiert werden, können Kanten an den Grenzen zwischen den Blöcken entstehen und die einzelnen Blöcke werden sichtbar. Im schlechtesten Fall werden alle 64 Pixel eines Blocks durch nur einen Wert dargestellt. Abbildung 5.5 (b) zeigt ein stark komprimiertes Foto. In der Vergrößerung eines Ausschnitts sind die Blockartefakte deutlich erkennbar. Im Vergleich mit dem schwächer komprimierten Foto in Abbildung 5.5 (a) fällt auf, dass bei einem Kompressionsfaktor von  $55 : 787 \approx 1 : 14$  der optische Qualitätsverlust, auf das ganze Bild gesehen, verhältnismäßig gering ausfällt und erst bei Betrachtung der vergrößerten Ausschnitte sichtbar wird.

Obwohl der JPEG-Standard eine fixe Blockgröße von  $8 \times 8$  vorschreibt, sind in der Simulation zusätzlich Blockgrößen von  $4 \times 4$  und  $2 \times 2$  implementiert. Auf diese Weise lässt sich die Auswirkung der Blockgröße auf das Kom-

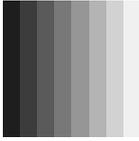
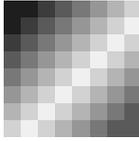
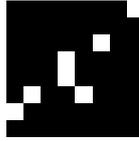
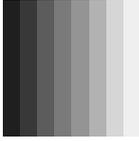
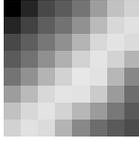
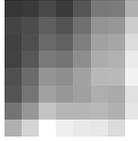
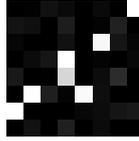
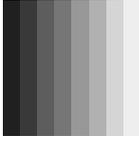
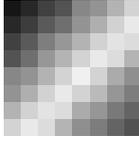
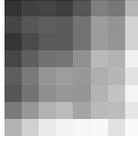
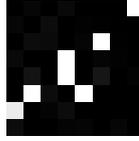
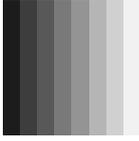
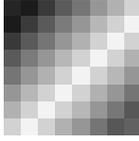
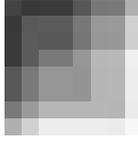
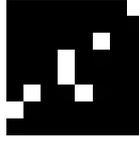


**Abbildung 5.5:** Vergleich eines schwach komprimierten Fotos (a) mit einer stark komprimierten Variante (b).

pressionsergebnis beobachten und es können ebenso Rückschlüsse auf größere Blöcke gezogen werden. Tabelle 5.2 zeigt verschiedene Rekonstruktionen nach JPEG-Kompression mit unterschiedlicher Blockgröße und Quantisierung mit  $R = 10$ . Eine kleinere Blockgröße wirkt sich zwar positiv auf die optische Qualität aus, jedoch steigt dabei in den meisten Fällen die Anzahl der zu codierenden DCT-Koeffizienten ungleich Null.

Ein anderer Aspekt, der mit der Blockgröße verbunden ist, ist die Komplexität der Berechnung. Wird ein Bild mit  $8 \times 8$  Pixeln als  $8 \times 8$ -Block transformiert, müssen  $8^4 = 4096$  gewichtete Kosinusanteile  $t(k, l, n, m)$  berechnet werden. Wird das selbe Bild in vier  $4 \times 4$ -Blöcke unterteilt, müssen nur  $4 \cdot 4^4 = 1024$  Werte berechnet werden. Bei sechzehn  $2 \times 2$ -Blöcken sind es sogar nur  $16 \cdot 2^4 = 256$  Berechnungen. Die Komplexität der Berechnung der DCT steigt also stark mit zunehmender Blockgröße [13, S. 528]. Nun treffen verschiedene Vor- und Nachteile aufeinander. Mit zunehmender Blockgröße steigt zwar die Komplexität der Berechnung, jedoch zu Gunsten eines besseren Kompressionsfaktors. Auf die optische Qualität eines komprimierten Bildes wirken sich größere Blöcke aber negativ aus, da die Unschärfe im Bild zunimmt und Blockartefakte eher sichtbar werden. Mit einer Blockgröße von  $8 \times 8$  Pixeln wurde also ein guter Kompromiss aus allen diesen Aspekten gefunden [13, S. 482].

**Tabelle 5.2:** Originalbilder und rekonstruierte Bilder nach JPEG-Kompression mit unterschiedlicher Blockgröße und Quantisierung mit  $R = 10$ . Die Zahl unterhalb jedes Bildes beschreibt die Anzahl der zu codierenden DCT-Koeffizienten ungleich Null.

Originalbild				
Blockgröße				
$8 \times 8$				
	3	13	15	45
$4 \times 4$				
	8	24	26	55
$2 \times 2$				
	32	50	40	28

## Kapitel 6

# Analyse der Vermittelbarkeit an Schulen und höheren Bildungsanstalten

Ungeachtet der späteren Berufswahl sollte in Zeiten zunehmender Digitalisierung allen Schülerinnen und Schülern ein gewisses Grundmaß an technischem Verständnis vermittelt werden. Bereiche wie Fotografie und neue Medien beispielsweise arbeiten viel mit JPEG-komprimierten Bildern und setzen zumindest ein Grundverständnis über die Funktionsweise der JPEG-Kompression voraus. In vielen technischen Bereichen, z. B. in der Nachrichtentechnik, ist ein umfassenderes Wissen über die Funktionsweise von Kompressionen notwendig. Schließlich dient die JPEG-Kompression auch als Beispiel für andere Kompressionsarten und sollte daher nicht nur im fachspezifischen, sondern auch im allgemeinschulischen Kontext Beachtung finden.

Der in dieser Arbeit vorgestellte JPEG-Simulator ist dazu gedacht, Lehrerinnen und Lehrern ein didaktisches Werkzeug in die Hand zu geben und sie dazu zu befähigen, die JPEG-Kompression in kürzerer Zeit genauer aufzubereiten bzw. zu behandeln. Der JPEG-Simulator kann der Lehrperson einerseits dazu dienen, die Eigenschaften zu erarbeiten und Aufgaben vorzubereiten und andererseits zum Generieren von Bildern und Tabellen für den Unterricht eingesetzt werden.

Schülerinnen und Schülern sowie Studentinnen und Studenten kann der JPEG-Simulator als Basis für experimentelles Lernen und Verstehen dienen. In vielen Anwendungsbereichen ist eine genaue Kenntnis der mathematischen Formeln nicht notwendig, ein detailliertes Verständnis der Wirkungsweise verschiedener Parameter jedoch wünschenswert und hilfreich. Dieses Verständnis kann durch analytisches Experimentieren mit dem JPEG-Simulator erarbeitet werden. Mit Vorbereitung und didaktischer Aufbereitung durch die Lehrperson kann der JPEG-Simulator bereits in allgemein

bildenden höheren Schulen (AHS) eingesetzt werden. Ein Einsatz wäre z. B. in Fächern wie Mathematik und Informatik denkbar. Im universitären Bereich kann der JPEG-Simulator zur intensiveren Auseinandersetzung mit dem Thema dienen und zu einem besseren Verständnis der zugrunde liegenden Mathematik beitragen.

In der AHS wird die Kosinusfunktion bereits in der 9. Schulstufe gelehrt [8]. Die DCT ist für diese Schulstufe zwar mathematisch zu komplex, jedoch kann die JPEG-Kompression mit Hilfe des JPEG-Simulators auch ohne genaue Kenntnis der Formeln in vier bis fünf Unterrichtseinheiten behandelt werden. Es ist sinnvoll, in den ersten zwei Unterrichtseinheiten die Grundbegriffe in vereinfachter Weise zu erarbeiten. Nach einer Einführung in die Funktionsweise des JPEG-Simulators durch die Lehrperson können die Schülerinnen und Schüler anschließend mit dem JPEG-Simulator und einem Arbeitsblatt die Eigenschaften der JPEG-Kompression herausarbeiten, indem sie die Eingabeparameter verändern und die Auswirkungen auf das rekonstruierte Bild beobachten und schriftlich festhalten.

Bis zur 12. Schulstufe der AHS werden alle nötigen mathematischen Grundlagen gelehrt, die für das Verständnis der JPEG-Kompression notwendig sind. Es ist also möglich, dieses Thema (z. B. im Rahmen eines Wahlfachs oder einer fächerübergreifenden Kooperation mit Informatik) ausführlich zu behandeln. Die Erarbeitung der Formeln kann mit Bildern unterstützt werden, die der JPEG-Simulator generiert. Die DCT kann für kleine Blöcke (z. B.  $2 \times 2$  Pixel) sogar händisch berechnet werden. Anschließend werden die Ergebnisse durch Eingabe der Parameter in den JPEG-Simulator verifiziert. Die notwendigen Codierungsarten können im Fach Informatik erarbeitet werden um anschließend die gesamte JPEG-Kompression bis zum Binär-code durchzuführen. Für eine solche vertiefende Einführung in die JPEG-Kompression sollte ein zeitlicher Rahmen von zumindest 10 Unterrichtseinheiten veranschlagt werden. Eine Bearbeitung dieses Themas ist auch in kürzerer Zeit möglich, indem nur Teile der JPEG-Kompression genau behandelt werden und z. B. die Berechnung der DCT mit dem JPEG-Simulator durchgeführt wird.

Der JPEG-Simulator ist ein Prototyp, der die grundlegenden Funktionen zur Verfügung stellt. Die Algorithmen und Methoden sind bewusst allgemein gehalten, um ein einfaches Erweitern des Funktionsumfangs zu ermöglichen. Einerseits kann das Programm von Lehrerinnen und Lehrern erweitert und verändert werden, andererseits kann der Quellcode als Grundgerüst für schulische oder studentische Programmierprojekte zu den Themen Kompression, Quantisierung und Kosinustransformation verwendet werden. Mögliche Erweiterungen wären z. B. das Hinzufügen eigener Testbilder und Quantisierungstabellen. Auch eine Umsetzung für Bilder beliebiger Größe oder Farbbilder würde sich anbieten.

## Kapitel 7

# Dokumentation des Quellcodes

### 7.1 Klasse ImageComponent

Die Klasse `ImageComponent` ist eine Hilfsklasse zur Darstellung von Bildern und ist an Listing 20.16 aus [16] angelehnt.

### 7.2 Klasse Gui

Die Klasse `Gui` steuert die Benutzeroberfläche des JPEG-Simulators, verwaltet die Eingabe und Ausgabe und ruft die Methoden der Klasse `jpegSimulator` auf. Der Quellcode entstand mit Hilfe der Bücher [12] und [16]. Teile wurden von der Programmierumgebung *Eclipse Luna*<sup>1</sup> generiert. Da diese Klasse nur zur Erzeugung der Benutzeroberfläche dient, wird sie hier nicht weiter dokumentiert.

### 7.3 Klasse jpegSimulator

Die Klasse `jpegSimulator` bildet den Kernteil des JPEG-Simulators und beinhaltet alle Algorithmen zum Berechnen der Quantisierung und der Zwischenschritte der 2D-DCT und 2D-IDCT. Des Weiteren stellt sie sechs verschiedene Testbilder sowie verschiedene Methoden zur Verfügung. Alle Werte werden in zweidimensionalen Arrays gespeichert. Im Sinne der besseren Lesbarkeit und Verständlichkeit wurde der Sourcecode nicht auf Effizienz hin optimiert. Die Variablennamen orientieren sich weitgehend an den Bezeichnungen in dieser Arbeit. Der Aufbau der Algorithmen ist an [9, S. 370] orientiert.

---

<sup>1</sup><https://www.eclipse.org/>

### 7.3.1 Globale Variablen

`int[] [] image0`

speichert ein Grauwertbild mit einem vertikalen Verlauf aus 8 Graustufen.

`int[] [] image1`

speichert ein Grauwertbild mit einem diagonalen Verlauf.

`int[] [] image2`

speichert ein Grauwertbild mit harten Kanten.

`int[] [] image3`

speichert ein Grauwertbild mit einem hellen Kreis auf dunklem Hintergrund.

`int[] [] image4`

speichert ein reines Schwarz-Weiß-Bild.

`int[] [] image5`

speichert ein Grauwertbild mit einem hellen Pixel auf dunklem Hintergrund.

`int[] [] quantTable0`

speichert die Quantisierungstabelle, die vom JPEG-Standard zur Verfügung gestellt wird (Tabelle 4.1 (a)).

### 7.3.2 Private Methoden

```
int[] [] mirrorTable(int[] [] array)
```

spiegelt die übergebene Tabelle und gibt sie zurück.

```
int[] [] toGrayscale(double[] [] array, int N, int M)
```

übersetzt reelle Zahlen in Grauwerte aus dem Intervall  $[0, 255]$ . Um die 256 verschiedenen Grauwerte optimal auszunützen, wird für jedes übergebene Array die tatsächliche Intervallbreite berechnet und um Null zentriert. Die Berechnung folgt der Vorschrift 3.7.

**Parameter:**

`array` – Tabelle mit Werten, die in Grauwerte übersetzt werden sollen

`N` – Breite des DCT-Blocks

`M` – Höhe des DCT-Blocks

**Rückgabe:**

`grayscales` – Tabelle mit Grauwerten

### 7.3.3 Öffentliche Methoden

```
int[] [] getImage0() ... getImage5()
```

gibt das jeweilige Testbild zurück.

```
int[] [] getQuantizationTable()
```

gibt die Quantisierungstabelle zurück.

```
int[] [] getQuantizationTableFromR(int R)
```

generiert eine Quantisierungstabelle nach Vorschrift 4.1 aus dem Parameter *R* und gibt diese zurück.

**Parameter:**

*R* – Quantisierungsparameter

**Rückgabe:**

*table* – Quantisierungstabelle

```
int[] [] quantization(double[] [] X, int[] [] Q, int N, int M)
```

berechnet die Quantisierung nach Vorschrift 4.3 und gibt die quantisierten DCT-Koeffizienten zurück.

**Parameter:**

*x* – Tabelle mit DCT-Koeffizienten, die quantisiert werden sollen

*Q* – Quantisierungstabelle

*N* – Breite des DCT-Blocks

*M* – Höhe des DCT-Blocks

**Rückgabe:**

*q* – Tabelle mit quantisierten DCT-Koeffizienten

```
int[] [] quantization(double[] [] X, int delta)
```

berechnet die gleichmäßige Quantisierung nach Vorschrift 2.7 und gibt die quantisierten DCT-Koeffizienten zurück. Bei der Division durch eine Quantisierungskonstante spielt die Größe des DCT-Blocks keine Rolle und muss daher nicht übergeben werden.

**Parameter:**

*x* – Tabelle mit DCT-Koeffizienten, die quantisiert werden sollen

*delta* – Quantisierungskonstante

**Rückgabe:**

*q* – Tabelle mit quantisierten DCT-Koeffizienten

```
int[] [] reverseQuantization(int[] [] q, int[] [] Q, int N, int M)
```

rekonstruiert die DCT-Koeffizienten nach Vorschrift 4.4 und gibt diese zurück.

**Parameter:**

*q* – Tabelle mit quantisierten DCT-Koeffizienten, die rekonstruiert werden sollen

*Q* – Quantisierungstabelle

*N* – Breite des DCT-Blocks

*M* – Höhe des DCT-Blocks

**Rückgabe:**

*x<sub>q</sub>* – Tabelle mit rekonstruierten DCT-Koeffizienten

```
int[] [] reverseQuantization(int[] [] q, int delta)
```

rekonstruiert die DCT-Koeffizienten nach Vorschrift 2.8 und gibt diese zurück. Bei der Multiplikation mit einer Quantisierungskonstante spielt die Größe des DCT-Blocks keine Rolle und muss daher nicht übergeben werden.

**Parameter:**

`q` – Tabelle mit quantisierten DCT-Koeffizienten, die rekonstruiert werden sollen

`delta` – Quantisierungskonstante

**Rückgabe:**

`xq` – Tabelle mit rekonstruierten DCT-Koeffizienten

```
int[] [] dctBaseImages(int N, int M)
```

berechnet die Kosinusanteile  $c(k, l, n, m)$  der 2D-DCT für die Blockgröße  $N \times M$  nach Zerlegung 3.10. Vor der Rückgabe werden die Kosinusanteile mit Vorschrift 3.4 in Grauwerte des Intervalls  $[0, 255]$  transformiert. Damit die Grenzen der einzelnen Bilder sichtbar sind, werden sie durch je einen schwarzen Pixel getrennt.

**Parameter:**

`N` – Breite des DCT-Blocks

`M` – Höhe des DCT-Blocks

**Rückgabe:**

`baseImages` – Basisbilder der 2D-DCT

```
int[] [] idctBaseImages(int N, int M)
```

berechnet die Kosinusanteile  $\bar{c}(k, l, n, m)$  der 2D-IDCT für die Blockgröße  $N \times M$  nach Zerlegung 3.11. Vor der Rückgabe werden die Kosinusanteile mit Vorschrift 3.4 in Grauwerte des Intervalls  $[0, 255]$  transformiert. Damit die Grenzen der einzelnen Bilder sichtbar sind, werden sie durch je einen schwarzen Pixel getrennt.

**Parameter:**

`N` – Breite des DCT-Blocks

`M` – Höhe des DCT-Blocks

**Rückgabe:**

`baseImages` – Basisbilder der 2D-IDCT

```
int[] [] dctWeightedImages(int[] [] x, int N, int M)
```

gewichtet die Kosinusanteile  $c(k, l, n, m)$  der 2D-DCT für die Blockgröße  $N \times M$  mit dem zu transformierenden Bild  $x(n, m)$  und dem Gewicht  $w(k, l)$  nach Zerlegung 3.10. Vor der Rückgabe werden die gewichteten Kosinusanteile mit der Methode `int[] [] toGrayscale(double[] [] array, int N, int M)` nach Vorschrift 3.7 in Grauwerte des Intervalls  $[0, 255]$  transformiert. Damit die Grenzen der einzelnen Bilder sichtbar sind, werden sie durch je einen

schwarzen Pixel getrennt. Die Blockgröße  $N \times M$  kann zwar beliebig gewählt werden, sollte jedoch kleiner sein als das zu transformierende Bild. Werden  $N$  und  $M$  zu groß gewählt, werden sie auf die Größe des Bildes gesetzt.

**Parameter:**

$x$  – Originalbild, das transformiert werden soll

$N$  – Breite des DCT-Blocks

$M$  – Höhe des DCT-Blocks

**Rückgabe:**

`grayscale` – Grauwerte der gewichteten Kosinusanteile der 2D-DCT

```
int[] [] idctWeightedImages(double[] [] X, int N, int M)
```

gewichtet die Kosinusanteile  $\bar{c}(k, l, n, m)$  der 2D-IDCT für die Blockgröße  $N \times M$  mit den Koeffizienten  $X(k, l)$  und dem Gewicht  $w(k, l)$  nach Zerlegung 3.11. Vor der Rückgabe werden die gewichteten Kosinusanteile mit der Methode `int[] [] toGrayscale(double[] [] array, int N, int M)` nach Vorschrift 3.7 in Grauwerte des Intervalls  $[0, 255]$  transformiert. Damit die Grenzen der einzelnen Bilder sichtbar sind, werden sie durch je einen schwarzen Pixel getrennt. Die Blockgröße  $N \times M$  kann zwar beliebig gewählt werden, sollte jedoch kleiner oder gleich der Koeffizientenmatrix sein. Werden  $N$  und  $M$  zu groß gewählt, werden sie auf die Größe der Koeffizientenmatrix gesetzt.

**Parameter:**

$x$  – DCT-Koeffizienten, die rücktransformiert werden sollen

$N$  – Breite des DCT-Blocks

$M$  – Höhe des DCT-Blocks

**Rückgabe:**

`grayscale` – Grauwerte der gewichteten Kosinusanteile der 2D-IDCT

```
double[] [] dct(int[] [] x, int N, int M)
```

transformiert das Bild  $x(n, m)$  mit der 2D-DCT mit Blockgröße  $N \times M$  nach Gleichung 3.8 und gibt die DCT-Koeffizienten  $X(k, l)$  zurück. Die Blockgröße  $N \times M$  kann zwar beliebig gewählt werden, sollte jedoch kleiner sein als das Bild. Werden  $N$  und  $M$  zu groß gewählt, werden sie auf die Größe des Bildes gesetzt.

**Parameter:**

$x$  – Originalbild, das transformiert werden soll

$N$  – Breite des DCT-Blocks

$M$  – Höhe des DCT-Blocks

**Rückgabe:**

$x$  – DCT-Koeffizienten des Originalbildes

```
int[] [] idct(int[] [] X, int N, int M)
```

rücktransformiert die DCT-Koeffizienten  $X(k, l)$  mit der 2D-IDCT mit Blockgröße  $N \times M$  nach Gleichung 3.9 und gibt eine Rekonstruktion des Originalbildes zurück. Die reellen Ergebnisse der Rücktransformation werden gerundet, da nur ganzzahlige Graustufen möglich sind. Die Blockgröße  $N \times M$  kann zwar beliebig gewählt werden, sollte jedoch der Blockgröße der Hintransformation entsprechen. Werden  $N$  und  $M$  größer als die Koeffizientenmatrix gewählt, werden sie auf die Größe der Koeffizientenmatrix gesetzt.

**Parameter:**

**x** – DCT-Koeffizienten, die rücktransformiert werden sollen

**N** – Breite des DCT-Blocks

**M** – Höhe des DCT-Blocks

**Rückgabe:**

**xr** – Rekonstruktion des Originalbildes

```
int countCoefficients(int[] [] array)
```

zählt die Anzahl der DCT-Koeffizienten, die nicht Null sind und gibt die Anzahl zurück.

**Parameter:**

**array** – Tabelle mit DCT-Koeffizienten

**Rückgabe:**

**count** – Anzahl der DCT-Koeffizienten ungleich Null

Anhang A

Sourcecode

# Abbildungsverzeichnis

2.1	Farbkreis nach Johannes Itten (a) [11, S. 31] (Die Datei wurde aus [2] entnommen.), additive Farbmischung RGB (b) [3] (Das Originalbild wurde modifiziert.) und subtraktive Farbmischung CMYK (c) [4]. . . . .	4
2.2	RGB-Farbwürfel [5]. . . . .	4
2.3	Grauwertbild (a) und Farbbild (b). (Foto: Hofstätter) . . . .	6
2.4	Darstellung der Komponenten Rot (R), Grün (G) und Blau (B) des RGB-Raums (a) und Grauwert- bzw. RGB-Darstellung der Komponenten Luminanz (Y), Chrominanz Blau (Cb) und Chrominanz Rot (Cr) des YCbCr-Raums (b). . . . .	7
2.5	Bild mit 20 Pixeln. . . . .	8
2.6	Ausschnitt aus einem Grauwertbild mit 15 Pixeln und zugehörige Grauwerte. . . . .	10
2.7	Farbbild mit 20 Pixeln. . . . .	12
2.8	Quantisierung eines linearen Grauwertverlaufs im Intervall [45, 224] mit Stufenbreite $\Delta = 30$ . (Die Darstellung ist an [15, S. 22] angelehnt.) . . . . .	15
3.1	Kosinusanteile $c(k, n)$ der Basisfunktionen für $N = 6$ . (Die Darstellung orientiert sich an [9, S. 369].) . . . . .	18
3.2	Vergleich der Darstellungsarten der Kosinusanteile $c(k, n)$ für $N = 6$ . (Die Darstellung orientiert sich an [13, S. 494].) . . . .	19
3.3	Vergleich der Basisbilder der eindimensionalen Kosinusanteile $c(k, n)$ und $\bar{c}(n, k)$ für $N = 4, 6, 8$ . (Die Darstellung orientiert sich an [13, S. 494].) . . . . .	19
3.4	Sechs Grauwerte $x(n)$ . . . . .	20
3.5	Zwischenschritte der 1D-DCT nach Zerlegung 3.5 und resultierende DCT-Koeffizienten. . . . .	20
3.6	Zwischenschritte der 1D-IDCT nach Zerlegung 3.6 und resultierende Grauwerte. . . . .	20
3.7	Vergleich der ursprünglichen Grauwerte mit der Rekonstruktion aus Abbildung 3.6. . . . .	22
3.8	Grauwertbild mit 9 Pixeln und 3 Graustufen. . . . .	24

3.9	Zwischenschritte der 2D-DCT nach Zerlegung 3.10. Zur einfacheren Darstellung wurde die Gewichtung $w(k, l)$ in der Grafik ausgelassen. . . . .	25
3.10	Zwischenschritte der 2D-IDCT nach Zerlegung 3.11. Zur einfacheren Darstellung wurde die Gewichtung $w(k, l)$ in der Grafik ausgelassen. . . . .	25
4.1	Ablauf der JPEG-Kompression [15, S. 195]. (Die Darstellung wurde modifiziert.) . . . . .	26
4.2	Ablauf der JPEG-Dekompression [15, S. 195]. (Die Darstellung wurde modifiziert.) . . . . .	27
4.3	Basisbilder der 2D-DCT für $8 \times 8$ Werte [15, S. 196]. (Die konkrete Datei wurde generiert.) . . . . .	28
4.4	Originalbild (a) und rekonstruiertes Bild (b) zu Abbildung 4.5.	36
4.5	Kompression und Rekonstruktion eines $8 \times 8$ -Blocks. . . . .	37
5.1	Eingabe des JPEG-Simulators. . . . .	39
5.2	Ausgabe des JPEG-Simulators. . . . .	40
5.3	Mathematische Reihenfolge der Zwischenschritte (a) und korrespondierende Zwischenschritte (b). . . . .	41
5.4	Vergleich eines unkomprimierten Schriftzugs (a) mit dessen Rekonstruktion nach einer JPEG-Kompression (b). . . . .	44
5.5	Vergleich eines schwach komprimierten Fotos (a) mit einer stark komprimierten Variante (b). . . . .	45

# Tabellenverzeichnis

2.1	RGB-Farbwerte und zugehörige Binärcodes. . . . .	9
2.2	Laufängencodierung. . . . .	10
2.3	Laufängencodierung eines Vektors mit 63 Werten. . . . .	10
2.4	Codebaum der Huffman-Codierung und resultierende Code- wörter (Variante 1). . . . .	13
2.5	Codebaum der Huffman-Codierung und resultierende Code- wörter (Variante 2). . . . .	14
2.6	Quantisierungsintervalle und -werte zu Abbildung 2.8. . . . .	15
3.1	Gerundete Kosinusanteile $c(k, n)$ der Basisfunktionen für $N =$ 6. (Siehe Abbildung 3.1) . . . . .	18
3.2	Quantisierung der DCT-Koeffizienten $X(k)$ von sechs Grau- werten $x(n)$ nach Vorschrift 2.7 mit Stufenbreite $\Delta = 30$ . . . . .	21
4.1	Quantisierungstabellen $Q(k, l)$ für Helligkeit (a) und Farbdif- ferenzen (b) [15, S. 197]. . . . .	29
4.2	Durch den Parameter $R = 4$ erzeugte Quantisierungstabelle nach Vorschrift 4.1. . . . .	29
4.3	Huffman-Codes zur Codierung der DC-Koeffizienten für Lu- minanz und Chrominanz [15, S. 200]. . . . .	31
4.4	Huffman-Kategorien zur Einteilung der AC-Koeffizienten [15, S. 201]. (Die Darstellung wurde modifiziert.) . . . . .	32
4.5	Huffman-Codes zur Codierung der AC-Koeffizienten (Teil 1) [15, S. 202f.]. (Die Darstellung wurde modifiziert und umsor- tiert.) . . . . .	33
4.6	Huffman-Codes zur Codierung der AC-Koeffizienten (Teil 2) [15, S. 202f.]. (Die Darstellung wurde modifiziert und umsor- tiert.) . . . . .	34
4.7	Huffman-Codes zur Codierung der AC-Koeffizienten (Teil 3) [15, S. 202f.]. (Die Darstellung wurde modifiziert und umsor- tiert.) . . . . .	35

- 5.1 Originalbilder und rekonstruierte Bilder nach JPEG-Kompression mit unterschiedlicher Quantisierung. Die Zahl unterhalb jedes Bildes beschreibt die Anzahl der zu codierenden DCT-Koeffizienten ungleich Null. . . . . 43
- 5.2 Originalbilder und rekonstruierte Bilder nach JPEG-Kompression mit unterschiedlicher Blockgröße und Quantisierung mit  $R = 10$ . Die Zahl unterhalb jedes Bildes beschreibt die Anzahl der zu codierenden DCT-Koeffizienten ungleich Null. . . . . 46

# Literaturverzeichnis

- [1] <http://www.jpeg.org>.
- [2] [http://commons.wikimedia.org/wiki/File:Farbkreis\\_Ippen\\_1961.png](http://commons.wikimedia.org/wiki/File:Farbkreis_Ippen_1961.png), (Stand 28.02.2007).
- [3] <http://commons.wikimedia.org/wiki/File:AdditiveColorMixing.png>, (Stand 16.08.2008).
- [4] [https://commons.wikimedia.org/wiki/File:CMY\\_ideal\\_version.svg](https://commons.wikimedia.org/wiki/File:CMY_ideal_version.svg), (Stand 19.05.2009).
- [5] [http://commons.wikimedia.org/wiki/File:RGB\\_color\\_cube.svg](http://commons.wikimedia.org/wiki/File:RGB_color_cube.svg), (Stand 06.01.2015).
- [6] R. B. Ash: *Information theory*. Interscience Publ., New York, 3. Aufl., 1967.
- [7] H. J. Bartsch: *Taschenbuch mathematischer Formeln*. Carl Hanser Verlag, München, 21. Aufl., 2007, ISBN 978-3-446-40895-1.
- [8] BMBF. [https://www.bmbf.gv.at/schulen/unterricht/lp/lp\\_neu\\_ahs\\_07\\_11859.pdf](https://www.bmbf.gv.at/schulen/unterricht/lp/lp_neu_ahs_07_11859.pdf), (Stand 2004).
- [9] W. Burger und M. J. Burge: *Digitale Bildverarbeitung: Eine Einführung mit Java und ImageJ*. Springer-Verlag, Berlin, Heidelberg, 2005, ISBN 978-3-540-21465-6.
- [10] D. A. Huffman: *A Method for the Construction of Minimum-Redundancy Codes*. Proceedings of the IRE, 40(9):1098–1101, Sept 1952, ISSN 0096-8390.
- [11] J. Itten: *Kunst der Farbe: Studienausgabe*. Urania Verlag, Stuttgart, 2003, ISBN 978-3-332-01470-9.
- [12] G. Krüger: *Handbuch der Java-Programmierung*. Addison Wesley Verlag, München, 3. Aufl., 2003, ISBN 3-8273-2120-4.
- [13] D. Salomon und G. Motta: *Handbook of Data Compression*. Springer-Verlag, London, 5. Aufl., 2010, ISBN 978-1-84882-903-9.

- [14] C.E. Shannon und W. Weaver: *Mathematische Grundlagen der Informationstheorie*. R. Oldenbourg Verlag, München, 1976, ISBN 3-486-39851-2.
- [15] T. Strutz: *Bilddatenkompression: Grundlagen, Codierung, Wavelets, JPEG, MPEG, H.264*. Vieweg+Teubner Verlag, Wiesbaden, 4. Aufl., 2009, ISBN 978-3-8348-0472-3.
- [16] C. Ullenboom: *Java ist auch eine Insel*. Galileo Press, Bonn, 9. Aufl., 2010, ISBN 978-3-8362-1506-0.