

## Inhaltsverzeichnis

<b>1</b>	<b>Logikgatter</b>	<b>2</b>
<b>2</b>	<b>Informationstheorie</b>	<b>2</b>
2.1	Datenquellen . . . . .	2
2.2	Formeln . . . . .	3
<b>3</b>	<b>Quellencodierung</b>	<b>3</b>
3.1	Huffman tree . . . . .	3
<b>4</b>	<b>Mediakompression</b>	<b>4</b>
4.1	JPEG . . . . .	4
4.1.1	Farbraumtransformation . . . . .	4
4.1.2	Chrominanz Downsampling . . . . .	4
4.1.3	Pixel-Gruppierung . . . . .	5
4.1.4	Diskrete Cosinustransformation . . . . .	6
4.1.5	Quantisierung . . . . .	7
4.1.6	Entropy-Coding . . . . .	8
4.1.7	In Datei verpacken . . . . .	8

# 1 Logikgatter




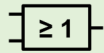


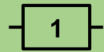


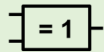

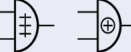
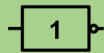


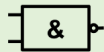


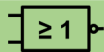


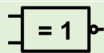

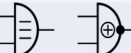
Function	Boolean Algebra <sup>(1)</sup>	IEC 60617-12 since 1997	US ANSI 91 1984	DIN 40700 until 1976
AND	$A \& B$			
OR	$A \# B$			
Buffer	$A$			
XOR	$A \$ B$			
NOT	$!A$			
NAND	$!(A \& B)$			
NOR	$!(A \# B)$			
XNOR	$!(A \$ B)$			

Abbildung 1: verschiedene Logikgatternotationen

## 2 Informationstheorie

Eine Information ist etwas, was vor seinem Eintreffen noch nicht bekannt war und kann viele Formen annehmen (Ton, Symbol, Text, Wert, ...). Information kann anhand der von ihr beseitigten Unsicherheit gemessen werden. Technisch gesehen ist die kleinste Masseinheit einer Information 1 Bit, sprich eine binäre Entscheidung.

### 2.1 Datenquellen

Eine Datenquelle wird als “diskret” bezeichnet, wenn sie abzählbare Messwerte liefert. Beispiele dafür sind digitale Sensoren, Ziehung der Lottozahlen, Wetterbericht im Radio. Datenquelle die “stetig” sind liefern nicht abzählbare Messwerte. Beispiele dafür sind jegliche analoge Messwerte (regeln eines Potentiometers, ablesen eines analogen Thermoeters).

Zudem kann eine Datenquelle “memoryless” sein, wenn die Messwerte statistisch unabhängig voneinander sind.

## 2.2 Formeln

**Die Warscheinlichkeit** einer Information wird errechnet indem man die Vorkommnis der Information durch die gesamten Vorkommnisse teilt.

$$P(x) = \frac{k(x)}{K}$$

**Der Informationsgehalt** einer Information wird in Bit angegeben und wird mit

$$I(x) = \log_2 \frac{1}{P(x)}$$

berechnet

**Die Entropie** einer Datenquelle bezeichnet den durchschnittlichen Informationsgehalt aller Informationen die diese liefert. Die Masseinheit der Entropie ist Bit/Symbol.

$$H(x) = \sum_{n=0}^{N-1} P(x_n) \log_2 \frac{1}{P(x_n)}$$

**Die Mittlere Codelänge** einer Datenquelle beschreibt die durchschnittliche Bitanzahl, die benötigt wird ein Zeichen der Quelle anzuzeigen.

$$L = \sum_{n=0}^{N-1} P(x_n) \cdot l_n$$

**Die Redundanz** einer Datenquelle bestimmt die Ineffizienz der Bitnutzung.

$$R = L(x) - H(x)$$

## 3 Quellencodierung

Das Ziel der Kompression ist, redundante und je nach Standard auch irrelevante Informationen zu minimieren um Ressourcen (Zeit, Energie, Bandbreite) zu sparen.

### 3.1 Huffman tree

Der Huffman tree bietet eine garantiert optimale Codierung. Das heisst eine Codierung mit der geringstmöglichen Redundanz. Dafür ordnet man alle Zeichen nach absteigender Probabilität und verbindet die kleinste Probabilität und die nächstgrössere zu einem Ast. Auf diesem addiert man die beiden Probabilitäten und ordnet den Ast neu ein. Diesen Vorgang wiederholt man, bis man an der Wurzel angekommen ist. Zur Überprüfung: die Summe aller Probabilitäten in der Wurzel muss 1 ergeben.

## 4 Mediakompression

### 4.1 JPEG

Das JPEG-Kompressionsverfahren ist ein **verlustbehafteter** Kompressionsalgorithmus, welcher sich zur komprimierung natürlicher Bilder eignet. Der JPEG-Algorithmus umfasst 7 Schritte.

#### 4.1.1 Farbraumtransformation

Unser Auge ist empfindlicher auf Helligkeitsunterschiede als auf Farbunterschiede. Das ermöglicht eine stärkere verlustbehaftete Kompression der Farben bevor wir markante Unterschiede feststellen können. Deshalb trennen wir die 3 Farbelemente aus dem RGB Farbraum auf in eine Helligkeitskomponente, einen Grün-Rot-Wert und einen Grün-Blau-Wert. Das das grüne Farbspektrum bleibt so stärker erhalten, da wir evolutionär bedingt empfindlicher auf Grünabstufungen sind als für andere Farben. Deshalb trennen wir die 3 Farbelemente aus dem RGB Farbraum auf in eine Helligkeitskomponente, einen Grün-Rot-Wert und einen Grün-Blau-Wert. Das das grüne Farbspektrum bleibt so stärker erhalten, da wir evolutionär bedingt empfindlicher auf Grünabstufungen sind als für andere Farben. Die Y-Komponente entspricht dem Graustufenbild des Ursprungbildes.

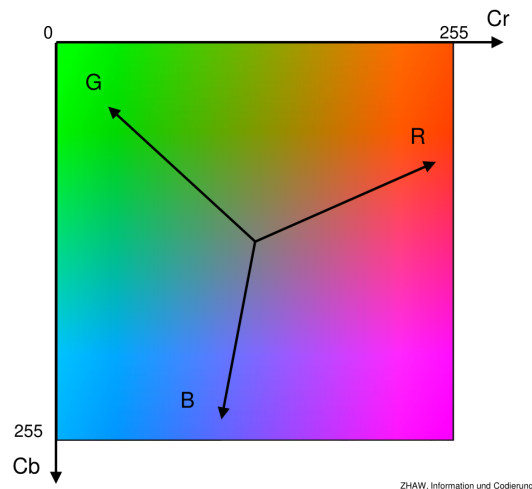


Abbildung 2: Verhältnis Cb zu Cr

#### 4.1.2 Chrominanz Downsampling

Unter dem Chrominanz Downsampling versteht man das Zusammenfassen mehrerer Pixel der Farbebenen. Hier werden das erste Mal irrelevante Informationen

“eliminiert”.

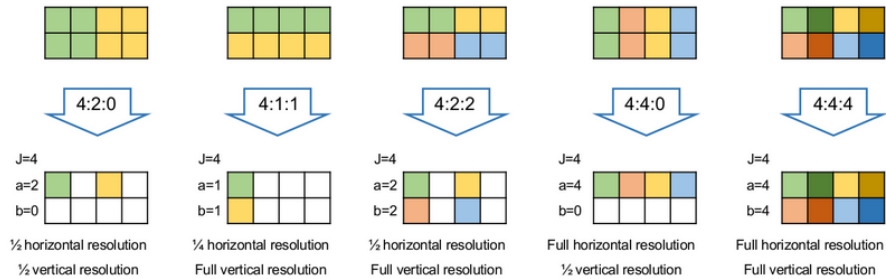


Abbildung 3: Chrominanz Downsampling

#### 4.1.3 Pixel-Gruppierung

Die Idee der nächsten Schritte ist, kleine Pixelgruppierungen möglichst effizient abzuspeichern. Dafür wird das jeder Informationskanal in 8x8 Pixelblöcke aufgeteilt, die dann jeweils gemeinsam komprimiert werden. Falls ein Kanal nicht sauber in 8x8 Pixelblöcke unterteilt werden kann, wird bei den inkompletten Blöcken am Rand entweder die letzte Spalte oder Zeile dupliziert, bis der Block voll ist oder den fehlenden Blöcken wird ein fixer Wert 0 zugewiesen. Beide Optionen führen zu kleinen, kaum spürbaren Artefakten an den Rändern.

#### 4.1.4 Diskrete Cosinustransformation

Dieser Schritt dient zur Übersetzung der Informationskanäle von Werten von 0-255 zu einer Kombination aus Cosinusfunktionen mit variierender Frequenz und ist komplett verlustfrei. Die Pixelblöcke werden einer nach dem anderen zu einer gewichteten Addition der Cosinusfunktionen übersetzt, da die Koeffizienten weniger Speicherplatz benötigen als die Werte der einzelnen Pixel selbst.

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

Abbildung 4: 8x8 Pixelarray

1260	-1	-12	-5	2	-2	-3	1
-23	-18	-6	-3	-3	0	0	-1
-11	-9	-2	2	0	-1	-1	0
-7	-2	0	2	1	0	0	0
-1	-1	2	2	0	-1	1	1
2	0	2	0	-1	2	1	-1
-1	0	0	-2	-1	2	1	-1
-3	2	-4	-2	2	1	-1	0

Abbildung 5: 8x8 Frequenzarray

Es findet also bereits eine verlustfreie Kompression statt. Für die Kompression verwendet man die Forward DCT über dem 2 dimensional 8x8-Pixelblock Array und erhält daraus für jeden Pixelblock ein 2 dimensionales Frequenzarray indem die jeweiligen Koeffizienten festgehalten werden. Die Forward DCT sieht wie folgt aus:

$$F_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^7 \sum_{y=0}^7 B_{yx} \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

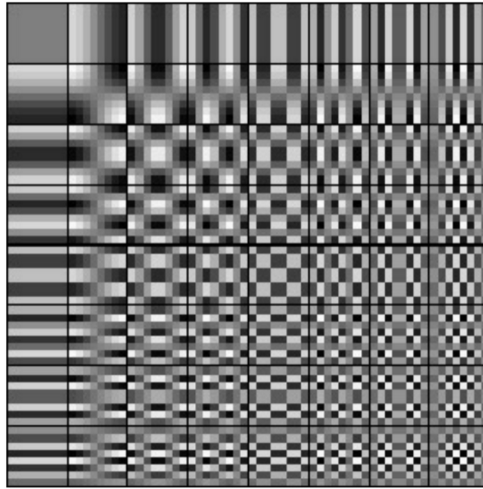


Abbildung 6: 8x8 Cosinusfunktionsmatrix

#### 4.1.5 Quantisierung

Die Quantisierung ist der letzte verlustbehaftete Schritt im ganzen Kompressionsverfahren. Hierfür verwendet man je nach gewünschter Kompression eine tolerantere oder eine aggressivere Quantisierungstabelle. Eine Quantisierungstabelle ist eine Tabelle die analog zu den Pixelblöcken 8x8 Werte beinhaltet, wobei die Werte oben links tiefer sind und gegen unten und gegen rechts höher werden. Indem man alle Werte des Frequenzarrays durch den entsprechenden Quantisierungskoeffizienten teilt und das Resultat auf die nächste ganze Zahl rundet erhält man die Quantisierte Koeffiziententabelle. Dabei gehen feine Unterschiede, die in der ursprünglichen Frequenztable unten rechts waren verloren, da sie durch höhere Werte geteilt werden, welche dann auf 0 abgerundet werden. Die massgebenden Frequenzen bleiben dabei besser erhalten.

##### Originale Koeffizienten ( $F_{vu}$ )

1260	-1	-12	-5	2	-2	-3	1
-23	-18	-6	-3	-3	0	0	-1
-11	-9	-2	2	0	-1	-1	0
-7	-2	0	2	1	0	0	0
-1	-1	2	2	0	-1	1	1
2	0	2	0	-1	2	1	-1
-1	0	0	-2	-1	2	1	-1
-3	2	-4	-2	2	1	-1	0

##### Quantisierungstabelle ( $Q_{vu}$ )

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

##### Quantisierte Koeffizienten

$$F'_{vu} = \text{round}(F_{vu}/Q_{vu})$$

79	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Abbildung 7: Quantisierung

#### 4.1.6 Entropy-Coding

Die Entropiecodierung dient dem effizienten speichern der quantifizierten Koeffiziententabelle. Da oben rechts tendenziell höhere Werte sind und unten links mehr 0 Werte, wird die Matrix in einem Zick-Zack-Muster durchlaufen und diese Tokens mithilfe eines RLE-Verfahrens gespeichert. Das RLE-Verfahren ist im JPEG-Algorithmus nicht genau vorgegeben.

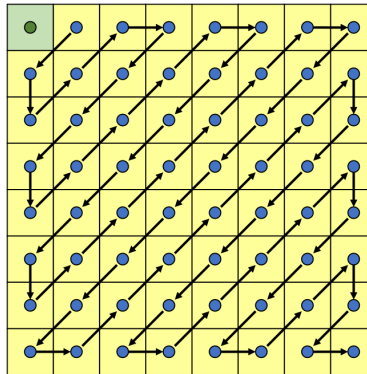


Abbildung 8: Zick-Zack durchlauf

#### 4.1.7 In Datei verpacken

Alle Werte die benötigt werden, um das Bild wieder herzustellen werden nun in eine .jpg Datei gespeichert. Das beinhaltet die komprimierte Koeffizientenmatrix sowie die genutzte Quantisierungstabelle und die entsprechenden Anhaltspunkte zum genutzten RLE-Verfahren.