h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt
- Fachbereich Informatik -

Grundlagen der Videokompression

Seminararbeit im Kurs
Wissenschaftliches Arbeiten in der Inforamtik I

vorgelegt von
Justin Böhm und Matthias Greune

Referentin: <Name>

Ausgabedatum: <Datum>
Abgabedatum: <Datum>

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

<Name>
<Ort>, den 6. Dezember 2016

Abstrakt

Diese Arbeit gibt einen Überblick über die grundlegenden Methoden von Videokompressionsverfahren. Hierfür werden, sich am Encoding-Prozess von MPEG-1 orientierend, zunächst Arten der Irrelevanzreduktion, anschließend die wichtigsten Ansätze der Redundanzreduktion vorgestellt und anhand von Beispielcode erläutert. In eigenen Tests wurden unter Anwendung der vorgestellten Methoden zur verlustbehafteten und partieller Anwendung von verlustfreien Kompressionsalgorithmen Kompressionsraten mit einer Ratio bis zu 1:60 erreicht. Diese Arbeit zeigt somit, wie mittels weniger Grundlagen bereits vergleichsweise hohe Einsparungen im Speicherverbrauch von Videos erreicht werden können. Insbesondere mit Blick auf die stetig steigenden Forderungen nach höheren Framerates, und besserer Auflösung wird deutlich, welch hohe Relevanz das Thema Videokompression auch in Zukunft haben wird.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Videos sind seit der Entwicklung des Fernsehers zum Massenmedium kaum noch aus dem alltäglichen Leben wegzudenken. Seit dem Aufstieg des Internets als zentrales Kommunikationsmedium haben sich allerdings die Anforderungen an geeignete Speichertechniken von Videos drastisch verändert. Die heutigen Abspielgeräte haben noch immer begrenzten Speicherplatz und sind häufig nur mit schmalbandigen Internetanbindungen ausgestattet. Die Auflösung der Videos ist hingegen stark gestiegen. Um diese Ansprüche zu adressieren wurden Kompressionsalgorithmen entwickelt, die eine effiziente Speicherung speziell für bewegte Bilder ermöglichen. Die resultierenden Probleme aus dieser Art der Speicherung, wie Bildartefakte, sind heutigen Nutzern wohlbekannt. Die eigentliche Funktionsweise von Videokompression bleibt aber oft unbemerkt. TBC

2. Irrelevanzreduktion

Die rohe Aufnahme eines Bildes bietet eine Fülle an Informationen. Mit Blick auf die Eigenschaften des menschlichen Sehsinns lässt sich hierbei allerdings feststellen, dass einige Informationen relevanter für das Erkennen eines Bildes sind, als andere. Die Irrelevanzreduktion beschäftigt sich mit der Trennung und Reduzierung von weniger wichtigen Informationen und bietet damit Methoden zur verlustbehafteten Datenkompression an.

Bei der Videokompression werden im wesentlichen zwei Eigenschaften zur Reduktion von Daten ausgenutzt. Zum einen nimmt das Auge Varianzen in der Helligkeit (Luminanz) stärker wahr, als Änderungen im Farbton (Chrominanz). Zum Anderen ist das Auge besser in der Lage niedrige Ortsfrequenzen zu erkennen, als hohe - erkennt also grobe Strukturen eher als feinere. Diese Eigenschaften können nun ausgenutzt werden, um einen guten Kompromiss aus akzeptabler Bildqualität und guter Datenreduktion zu finden [**akramullah_digital_2014**].

2.1. Chroma Subsampling

Das Chroma Subsampling nutzt den Umstand aus, dass Helligkeitsvarianzen besser wahrgenommen werden, als Farbvarianzen. Zumeist liegen die Bildinformationen im Ausgangsformat jedoch im RGB Farbmodell vor, wobei hier die Helligkeitswerte in jeden Kanal eingehen. Um nun aber die Chrominanz bei gleichbleibender Auflösung der Luminanz zu reduzieren wird eine getrennte Darstellung dieser Informationen benötigt. Hierfür wird im MPEG-1 Standard die $\mathrm{YC}_B\mathrm{C}_R$ Darstellung verwendet, wobei das Y für die Luminanz steht und in C_B und C_R die Farbwerte codiert werden. Die Umrechnung lässt sich mittels folgender Formeln realisieren [**itu-t_recommendation_1995**]:

$$
\begin{aligned}
Y &= 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \\
U &= (B - Y) \cdot 0.493 \\
V &= (R - Y) \cdot 0.877
\end{aligned}
$$

Nun kann das eigentliche Subsampling stattfinden, welches bei MPEG-1 bei einer Auflösung von 4:2:0 realisiert wird. Die erste Zahl gibt hierbei die horizontale Abtastrate der Luminanz an. Die zweite Zahl steht für die horizontale Abtastrate der C_B und C_R Kanäle

in Relation zum ersten Wert. Die dritte Zahl gibt die vertikale Samplingrate an, wobei diese entweder 2 oder 0 betragen kann, also entweder kein vertikales Subsampling, oder vertikales Subsampling von 2:1 stattfindet. Für den Fall von 4:2:0 Subsampling bedeutet dies, dass jeweils 2x2 Bildpunkte des C_B und C_R Kanals auf einen Bildpunkt in der Ergebnismenge abgebildet werden. Hiermit wird also die Auflösung des C_B und C_R Kanals halbiert, was zu einer Datenreduktion von insgesamt 50% führt. [**poynton_chroma_????**]

Das Chroma Subsampling bietet somit eine gute Möglichkeit der Kompression, die allerdings nicht verlustfrei abläuft. Artefakte können, wie in Abbildung A.1 im Anhang dargestellt, bei Verwendung dieser Methode vor allem bei scharfen, farbigen Kanten entstehen, wenn diese durch einen gesubsampleten Block verlaufen.

2.2. Diskrete Kosinus Transformation

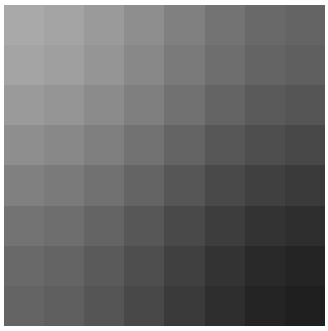
Wie bereits oben beschrieben neigt der menschliche Sehsinn dazu niedrige Ortsfrequenzen eher zu erkennen, als höhere. Eine Ortsfrequenz ist definiert als „Anzahl bestimmter periodischer Erscheinungen bezogen auf einen räumlichen Abstand" [**atmwiki_ortsfrequenz_????**]. Wir erkennen also gröbere Strukturen mit einer niedrigen Ortsfrequenz eher als feinere Strukturen mit einer höheren. Um diesen Umstand nun auszunutzen muss das Ausgangsbild von der räumlichen Ebene auf eine Frequenzebene transformiert werden, damit anschießend, in dem darauf folgenden Schritt der Quantisierung, die höheren Frequenzen reduziert werden können. Diese Transformation lässt sich mittels einer zweidimensionalen Diskreten Kosinus Transformation (DCT) bewerkstelligen.

Die DCT ist eine Sonderform der Foriertransformation, bei der eine Funktion mittels Sinusschwingungen approximiert wird. Die Fourriertransformation hat allerdings unter anderem den Nachteil, dass für jeden abgetasteten Punkt ein Tupel aus Amplitude und Phase bzw. Sinus und Kosinus Koeffizienten gespeichert werden muss. Die DCT nutzt nun den Umstand aus, dass das betrachtete Intervall begrenzt ist. Durch eine vertikale Spiegelung dieses Intervalls lassen sich die Sinus Anteile herauskürzen, wobei am Ende lediglich Kosinus Anteile übrig bleiben, also nur ein Koeffizient pro abgetasteten Punkt gespeichert werden muss. Des Weiteren bewirkt die Spiegelung, dass Start- und Endpunkt equivalent sind. Da die Fouriertransformation von einer unendlichen Folge ausgeht, muss der letzte Koeffizient den ggf. großen Unterschied zwischen Start- und Endpunkt ausgleichen. Sind diese Punkte aber equivalent, wird die Kraft des letzten Koeffizienten nicht verschwendet [**symes_peter_digital_2004**]. Verarbeitet werden mit der zweidimensionalen DCT immer 8x8 Blöcke eines jeden Kanals mit der Formel:

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

$$\text{wobei}\begin{cases} C_u=\frac{1}{\sqrt{2}} \text{ für } u=0, \text{ ansonsten } C_u=1 \\ C_v=\frac{1}{\sqrt{2}} \text{ für } v=0, \text{ ansonsten } C_v=1 \end{cases}$$

Die Abbildung 2.1 zeigt das Resultat einer angewandten DCT auf einen schwarz-weißen 8x8 Pixelblock, welcher aus jeweils einer horizontalen und einer vertikalen Kosinus Schwingung besteht. Der sogenannte DC Wert ist der erste Wert der Matrix und gibt die mittlere Helligkeit an. Alle anderen Komponenten beschreiben die relative Abweichung zu diesem Wert und werden gemeinhin als AC Werte betitelt, wobei diese zugleich die zum unteren rechten Rand hin höher werdenden Ortsfrequenzen repräsentieren. Wie bereits zu erkennen führt die DCT oftmals selbst schon durch Rundung auf ganzzahlige Ergebnisse zu einer Matrix mit einer erhöhten Anzahl gleicher Werte, die sich für die Anwendung weiterer, verlustfreier, Kompressionsmethoden eignet.



800	200	0	0	0	0	0	0
200	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Abbildung 2.1.: Mittels DCT gut komprimierbarer 8x8 Pixelblock
Links: Ausgangsbild, Rechts: Resultierende DCT-Matrix

2.3. Quantisierung

Im vorigen Schritt wurde durch Anwendung der Diskreten Kosinus Transformation eine Matrix mit den korrespondierenden Ortsfrequenzen eines 8x8 Pixelblocks gewonnen. Um nun tatsächlich eine Reduktion der höheren Ortsfrequenzen zu erreichen wird die Methode der Quantisierung angewandt. Hierbei wird eine ganzzahlige Division eines jeden DCT Koeffizienten mit einem Quantisierungswert vorgenommen. Das gerundete Ergebnis ist dann der quantisierte Wert. Durch diese Division und Rundung wird versucht die bisher noch hohen Koeffizienten zu verkleinern, sowie in den höheren Frequenzbereichen

möglichst auf Ergebnisse gleich Null zu kommen.

Im Fall von MPEG-1 wird hierfür ein Uniform Scalar Quantizer verwedet, bei dem die Eingangswerte durch Division der Schrittgröße auf Bereiche gleicher Größe abgebildet werden, wobei eine stufenähnliche Charakteristik entsteht. [**symes_peter_digital_2004**]. Um die errechneten Ortsfrequenzen in Relation zur Wahrnehmung des menschlichen Auges zu reduzieren wird hierfür eine Quantisierungsmatrix verwendet. Diese beinhaltet separate Werte für jeden DCT Koeffizienten. Die Schrittgröße setzt sich für AC-Werte zusammen aus dem korrespondierenden Quantisierungswert der Quantisierungsmatrix und einem Quantisierungsfaktor (MQuant). Der Quantisierungsfaktor dient der Steuerung der Bildqualität und kann einen Wert zwischen 1 und 31 annehmen, wobei ein Quantisierungsfaktor von 1 für eine hohe Bildqualität sorgt, ein Faktor von 31 hingegen für eine stark reduzierte. Da das Auge sensibel gegenüber großräumigen Luminanzfehlern ist, wird der DC durch eine feste Schrittgröße von 8 dividiert. [**ISO13586**] Eine Implementierung des vorgestellten Algorithmus ist in Listing 2.1 im Anhang zu sehen.

In Abbildung A.2 des Anhangs ist die angewandte Quantisierung exemplarisch an einem Beispielbild mit der im MPEG-1 Standard voreingestellten Quantisierungsmatrix (siehe Anhang, Tabelle A.1) sowie Quantisierungsfaktoren von eins, 16 und 31 dargestellt. Bei höhreren Quantisierungsfaktoren sind hier deutliche Qualitätsverluste zu erkennen, wobei die groben Strukturen des Bildes aber erhalten bleiben.

Durch die Anwendung der DCT wird ein eingehender 8x8 Pixelblock also in eine Darstellung transformiert, die es erlaubt mittels der Quantisierung vor allem enthaltene höhere Ortsfrequenzen zu reduzieren. Diese Prozesse führen zunächst jedoch nicht direkt zu einer Datenreduktion, da trotz des erhöhten Anteils gleicher Werte in der Matrix eben diese Werte auch gespeichert werden müssen. Allerdings wurde erreicht, dass die Entropiekodierung, welche im nachfolgenden Kapitel erläutert wird, bessere Kompressionsergebnisse erzielen kann.

3. Redundanzreduktion

Die Redundanzreduktion ist cool und sie reduziert Redundanz.

3.1. Entropiecodierung

3.2. Motion Compensation

Alle bis jetzt vorgestellten Ansätze der Videokompression beschäftigen sich mit der Kompression von Einzelbildern innerhalb eines Videos. Bei der Motion Compensation hingegen wird das Kompressionspotential ausgenutzt, dass innerhalb der Abhängigkeiten der Einzelbilder in einem Video steckt. Videos bestehen meist aus zusammenhängenden Szenen mit größtenteils unverändertem Inhalt innerhalb einer jeweils solchen Szene.

Man stelle sich zum Beispiel die folgende Szene vor: Eine Kamera filmt einen Mensch, unseren Protagonisten, der eine Straße entlang läuft und schließlich eine Bar betritt, eine typische Szene in Serien heutzutage.

Teilt man diese Szene in ihre Einzelbilder (Frames) auf, stellt man schnell fest, dass die Einzige Bewegung vom laufenden Protagonisten ausgeht und große Teile des Hintergrunds dabei in mehreren Bildern wiederholt werden. Motion Compensation nutzt die Redundanz dieser redundanten Hintergründe aus, indem es diese jeweils nur ein Mal speichert und in den folgenden Bildern darauf referenziert um ein für den Zuschauer unverändertes Bild anzuzeigen. Da Videos üblicherweise zum Großteil mit redundanten Bildteilen in einzelnen Szenen gefüllt sind, macht die von Motion Compensation erzielbare Kompression einen großen Teil des gesamt möglichen Kompressionpotentials innerhalb von Videos aus.

Damit Motion Compensation überhaupt funktionieren kann ist eine Aufteilung und Auswertung aller Video Einzelbilder in verschiedene Frames nötig.

3.2.1. Frames

Mit dem Kodieren teilt man alle Frames in eine bestimmte Bildart ein: Es gibt rein intracodierte Frames, die sogenannten intracoded Frames (kurz I-Frames), bei ihnen handelt es sich um einzelne Vollbilder, die von keinem anderen Bild des Videos abhängen. I-Frames sind also für sich stehende JPEG Bilder, welche mit den üblichen Methoden der Bildkompression verkleinert wurde. Außerdem gibt es intercodierte Frames, die nur eine vorhergesagte Differenz des Inhaltes in Abhängigkeit zu einem vorherigen I-Frame haben, die sogenannten predictive Frames (kurz P-Frames). Als letztes gibt es bipredictive Frames (kurz B-Frames), die sehr ähnlich zu P-Frames sind, jedoch in zwei Richtungen intercodiert wurden. Sie speichern nur die vorhergesagte Differenz des Inhaltes zum vorherigen I- oder P-Frame. Um die Vorhersagung zu erreichen zu können wird eine Reihenfolge der Codierung gewählt, die ungleich der Anzeigereihenfolge ist, wie auf der Abbildung X erkennbar ist. Dadurch wird der sowieso schon komplexe Prozess zusätzlich erschwert.

Ein kompletter Szenenwechsel, also das Ändern des kompletten Bildes, ohne statische Zusammenhänge, muss dem Encoder immer mitgeteilt werden. Dieser muss dann einen neuen I-Frame codieren, auf dem die folgenden P- und B-Frames basieren. Dadurch wird die potentielle Gefahr einer starken Artefaktbildung vorgebeugt.

Wenn man diese Aufteilung jedoch jeweils nur einmal pro Szene anwenden würde, würden mehrere Probleme bei wahllosem Zugriff entstehen. Wenn der I-Frame einer Szene fehlt oder übersprungen wird, würden die Änderungen, die in den folgenden P- und B-Frames festgehalten wurden, auf den falschen I-Frame angewendet, sodass im Video starke Artefakte entstehen, wie man in Abbildung X im Anhang erkennt. Beim Ausfall eines P-Frames einer Szene würde grundsätzlich das Gleiche gleiche passieren, jedoch nur bei den noch folgenden P-Frames der Szene.

Um diese unschönen Artefakte beim Vor- und Zurückspulen zu verhindern, dürfte nur zu einem I-Frame gesprungen werden, welches bei einer Aufteilung pro Szene jeweils nur der Anfang einer neuen Szene wäre.

Da bei einem Großteil der Anwendungsfälle von Videos jedoch eine fast vollständig wahlfreier Zugriff gewünscht ist, teilt man sie in viele kleine aufeinanderfolgende Bildergruppen (Group of pictures, kurz GOP) auf. Eine GOP wird meist mit 2 Parametern angegeben, in unserem Beispiel N und M. Dabei ist N eine bestimmte Anzahl von Frames aus denen die GOP besteht, also die Distanz von einem I-Frame zum nächsten I-Frame. M gibt die Distanz von einem I- oder P-Frame, bis zum jeweils Nächsten an, somit ist M-1 die Anzahl von B-Frames, die nach einem I- oder P-Frame folgen. Eine Bildergruppe fängt immer mit

einem I-Frame an und wiederholt sich bis zum Ende eines Videos mit einem konstanten Schema.

Mit den Parametern N=12 und M=4, würde die GOP dann aussehen wie auf der Abbildung X. (IBBBPBBBPBBB I...) TODO: ABBILDUNG

Bei MPEG ist eine Aufteilung mit den Parametern M=3 bis 4 und N= 11 bis 15 üblich.

Betrachtet man ein Video mit einer üblichen Framerate von 25, dann ist dadurch wahlfreier Zugriff mit einer Genauigkeit von bis auf die Hälfte einer Sekunde gegeben. Außerdem wird dadurch bei leichten Übertragungsfehlern einer Videodatei der Schaden minimiert, sodass das vom Endnutzer gesehene Bild nur maximal eine halbe Sekunde Artefakte anzeigt.

3.2.2. Makroblocks

Beim Komprimieren eines Videos wird zunächst ein Frame pro GOP mittels Irrelevanzreduktion komprimiert und dann als Referenz zwischengespeichert. Die folgenden Bilder werden, um die vom Encoder benötigte Arbeit einfach aufteilen zu können, in sogenannte Makroblöcke unterteilt. Diese Makroblöcke sind bei den meisten Standards auf eine feste Größe von 16x16 Pixel gesetzt.

3.2.3. Motion Compensation

:

Das in Makroblöcke aufgeteilte Bild wird vor der Durchführung der Irrelevanzreduktion mit der Referenz Block für Block verglichen um statische Bildinhalte zu erkennen. Ein Bildinhalt ist statisch, wenn sich ein Block von einem Bild zum nächsten nicht verändert hat. Alle statischen Bildinhalte werden dann entfernt, stattdessen wird auf den Inhalt der gespeicherten Referenz verwiesen. Damit sind zwar statische Bildinhalte kein Problem, allerdings kann, zum Beispiel bei einem Schwenken der Kamera, der trotzdem in beiden Bildern identisch vorhandene Hintergrund nicht kodiert werden, da sich sein Block in der Referenz zum Block des folgenden Bildes unterscheidet. Um diese immer noch redundanten Bildinformationen ebenfalls entfernen zu können, bedarf es einer komplexeren Vorgehensweise, der Motion Compensation. Die Motion Compensation sucht jene Blöcke in unserem neuem Frame mittels einem Block Matching Algorithmus heraus. Gefundene Blöcke werden mit einem Vektor, der von der Position des neuen Blocks, auf die Position

des Ursprungsblocks aus der Referenz zeigt, kodiert. Beim Dekodieren kann dann mittels dieser Vektoren auf die Position des alten Blocks referenziert werden, sodass nur dieser Vektor gespeichert werden muss.

TODO Abbildung undso wie bei Wikipedia EN, maybe eher bei Block matching

3.2.4. Block Matching

TODO: Block matching mathematik erklären?

4. Ausblick

ÄÖÜäöüß

5. Zusammenfassung

ÄÖÜäöüß

A. Weitere Abbildungen und Tabellen

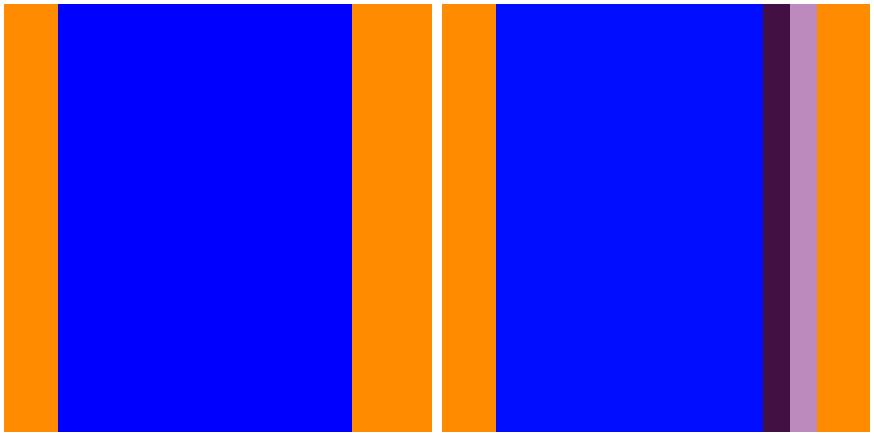


Abbildung A.1.: Artefakte durch Chroma Subsampling
Links: Original, Rechts: Subsampled. Die rechte Kante des blauen Farbblocks liegt in gesubsampleten 2x2 Blöcken, wodurch Artefakte entstehen. Die linke Kante liegt zwischen zwei 2x2 Blöcken, weshalb es zu keiner falschen Darstellung kommt.

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

Tabelle A.1.: Voreingestellte MPEG-1 Intracoding Quantisierungsmatrix. [**symes_peter_digital_2004**]

Abbildung A.2.: Ergebnis der Quantisierung mit verschiedenen Quantisierungsfaktoren
Oben links: Original, Oben rechts: Quantisiert mit Faktor 1, Unten links: Quantisiert mit Faktor 16, Unten rechts: Quantisiert mit Faktor 31.
Mit zunehmendem Quantisierungsfaktor ist ein ansteigender Verlust der Bildqualität zu beobachten, wobei grobe Strukturen weitestgehend erhalten bleiben. Original nach ***[brooke_cagle_2016]***

Genutzte Optionen			Größe in Kilobyte	Ratio
RLE	Chroma Subsampling	Quantisierung mit Faktor		
X		-	258.38	3.76
X	X	-	145.95	6.66
X	X	1	58.32	16.67
X	X	16	18.59	52.29
X	X	31	16.23	59.89

Tabelle A.2.: Testergebnisse der angewandten Kompressionsalgorithmen bei einer Ausgangsgröße von 970 Kilobyte des Originalbildes

B. Listings

```
def dct(f):
    # initialize resulting DCT array
    F = [8*[0], 8*[0], 8*[0], 8*[0], 8*[0], 8*[0], 8*[0], 8*[0]]
    # Go through f and calculate DC/AC for each value
    for u in range(0, 8):
        for v in range(0, 8):
            if u == 0:
                cu = 1/math.sqrt(2)
            else:  # u > 0:
                cu = 1
            if v == 0:
                cv = 1/math.sqrt(2)
            else:  # v > 0:
                cv = 1

            sum = 0
            for x in range(0, 8):
                for y in range(0, 8):
                    sum += f[x][y] * math.cos((((2 * x) + 1) * u
                        * math.pi ) / 16) * math.cos((((2 * y) +
                        1) * v * math.pi) / 16)
            # save result in F
            F[u][v] = round((cu * cv * sum) / 4)
    return F
```

Listing B.1: Implementierung der DCT für ein 8x8 Array

```
def quantize(dct, quantizer, MQuant=1):
    result = np.empty_like(dct)
    for x, row in enumerate(dct):
        for y, coefficient in enumerate(row):
            if x == 0 and y == 0:
                result[x][y] = int(coefficient / 8)
            else:
                result[x][y] = int( 8 * coefficient / (MQuant *
                   quantizer[x][y] ))
    return result
```

Listing B.2: Implementierung des Quantisierungsprozesses nach MPEG-1 Standard ohne Clipping