# MultiMediaPraktikum

Universität Tübingen Wilhelm-Schickard-Institut für Informatik

**Wintersemester 2002/ 2003**
**Aufgabe 2 - MPEG-Video**

Ulrike Schaal
David Eißler
Marc-Oliver Pahl

# Inhaltsverzeichnis

# MPEG-1
# Farbformate

## RGB

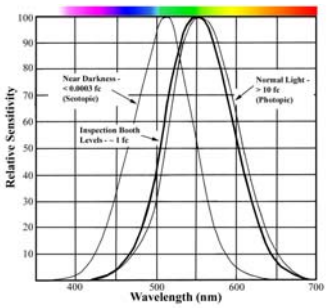
Beim RGB-Farbformat werden die Farben als drei-Komponenten-Vektor gespeichert; und zwar entsprechend ihren Anteilen an den Grundfarben [R]ot, [G]rün, [B]lau (additive Farbmischung).

Das Format wird vor allem bei Bildern auf dem Computer verwendet, da dort das Bild eben gerade aus den drei Grundfarben zusammengesetzt wird.

## YUV

Beim YUV-Format wird die Farbe als Luminanz (=Helligkeit) [Y] und Chrominanz (=Farbdifferenzsignal) [UV] gespeichert. Bei der Gewichtung der Farbanteile für das Luminanzsignal hat man die Empfindlichkeitsfunktion des Auges zugrunde gelegt:

$$Y = 0.299*R + 0.587*G + 0.114*B$$




Grün wird vom Auge am intensivsten wahrgenommen, weshalb es auch am Meisten zur Luminanz (auch für Schwarzweißdarstellung) beiträgt. Im U- und V-Teil des Vektors wird der Unterschied von Blau bzw. Rot zur Luminanz gespeichert.

Die Norm wurde im Zusammenhang mit der Fernsehnorm PAL (auch bei SECAM) entwickelt und hat den Vorteil, dass auftretende Störsignale dem Betrachter nur wenig auffallen, weil man sie eben gar nicht so stark wahrnehmen kann. Aus demselben Grund sind sich U- und V-Kanal auch sehr gut für das Subsampling geeignet.

Ein großer Vorteil dieses Formats für das Fernsehen ist auch, dass es abwärtskompatibel zu Schwarzweißfernsehern ist. Diese stellen nämlich einfach das Y-Signal (die Luminanz) dar.

Aus RGB errechnet es sich folgendermaßen:

$$Y = 0.299*R + 0.587*G + 0.114*B$$

$$U = B-Y$$

$$V = R-Y$$

## YIQ

Dieses Format wurde mit der NTSC-Fernsehnorm (USA/ Japan) entwikkelt und hat eigentlich dieselben Zielsetzungen, wie das YUV-Format.

Entsprechend ist das Y-Signal auch gleich. Nur I und Q entsprechen nicht U und V sondern sind zu den U- und V-Vektoren im RGB-System um 33° gedreht:

$$I = 0.877(R-Y)\cos(33°) - 0.492(B-Y)\sin(33°)$$

$$Q = 0.877(R-Y)\sin(33°) - 0.492(B-Y)\cos(33°)$$

Dadurch sind die Farben dann in den I- und Q-Kanälen allerdings „vermischt“, was dazu führt, dass Übertragungsfehler (durch Subsampling verstärkt) zu Farbverschiebungen (also nicht nur Farbe stärker oder schwächer sondern andere Farbe!) führen.

Aus RGB errechnet es sich folgendermaßen:

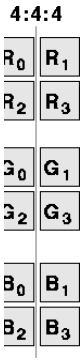
$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

## Subsampling

Wie zuvor erwähnt, nimmt das Auge Helligkeitänderungen stärker war, als Farbschwankungen. Das macht man sich beim Subsampling zunutze:

Farbinformation wird ungenauer gespeichert als Helligkeitsinformation.

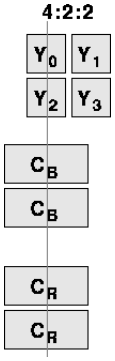
Subsampling bedeutet „Unterabtastung“. D.h. von unseren drei Komponenten des Farbvektors (s.o.) werden nicht pro Pixel jeweils alle drei sondern zum Beispiel pro 2x2-nur ein U- und V-Wert übertragen. Dieser ist dann für alle vier Pixel gleich.




### 4:4:4

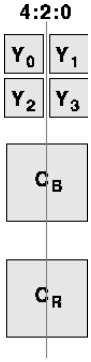
„kein Subsampling“

Bei diesem Subsamplingformat werden für vier Pixel vier Y-, vier U- und vier V-Werte gespeichert.




### 4:2:2

- horizontales Subsampling um Faktor 2
- je zwei aufeinanderfolgende Farbinformationen werden zusammengefasst
- jede Zeile enthält doppelt so viele Y-Samples wie Cb oder Cr Samples
- Einsparung von 33% Bandbreite

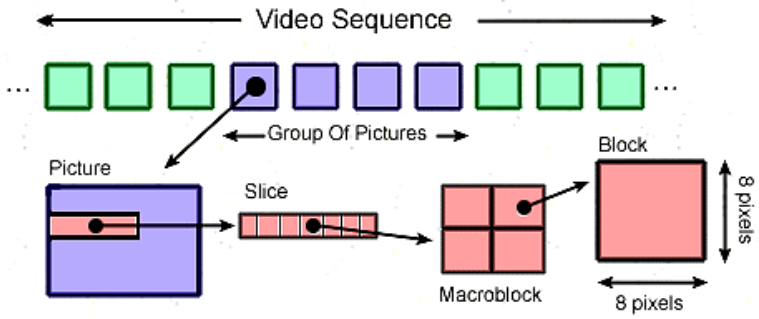



### 4:2:0

- horizontale und vertikale Zusammenfassung von Farbinformationen, jeweils um Faktor 2
- je vier Farbinformationen werden zusammengefasst
- auf 4 Pixel Luminanzinformation kommt je ein Pixel Farbinformation
- Einsparung von 50% Bandbreite

## Dekodierung des MPEG-1-Datenstroms

Der MPEG-1 Datenstrom setzt sich aus Layern zusammen. Diese werden beim Entpacken auseinander genommen und dann dekodiert.




Sequence-Layer...

dient zur Steuerung des Zwischenspeichers bei der Dekodierung.

Group of Pictures-Layer...

beinhalten die einzelnen kodierten Bilder und beginnt mit einem I-Frame (damit die Dekodierung starten kann, muss sie bei einem Vollbild beginnen). Die einzelnen Bilder innerhalb der GOP (ca. 0,5 sec) lassen sich nur aus der gesamten GOP rekonstruieren (B-/ P-Frames).

Picture-Layer...

enthält ein Einzelbild (P-/ B-/ I-Frame).

Slice-Layer...

enthält mehrere Makroblöcke des kodierten Bildes (die Größe ist variabel). Im Slice-Layer wird die DCT-Qualität skaliert.

Macroblock-Layer...

beinhaltet einen Makroblock.

Block-Layer...

entspricht einem Block

---

Frametypen:

Im MPEG-Datenstrom gibt es drei verschiedene Arten, wie Frames kodiert werden: I-, P- und B-Frames.

I-Frames

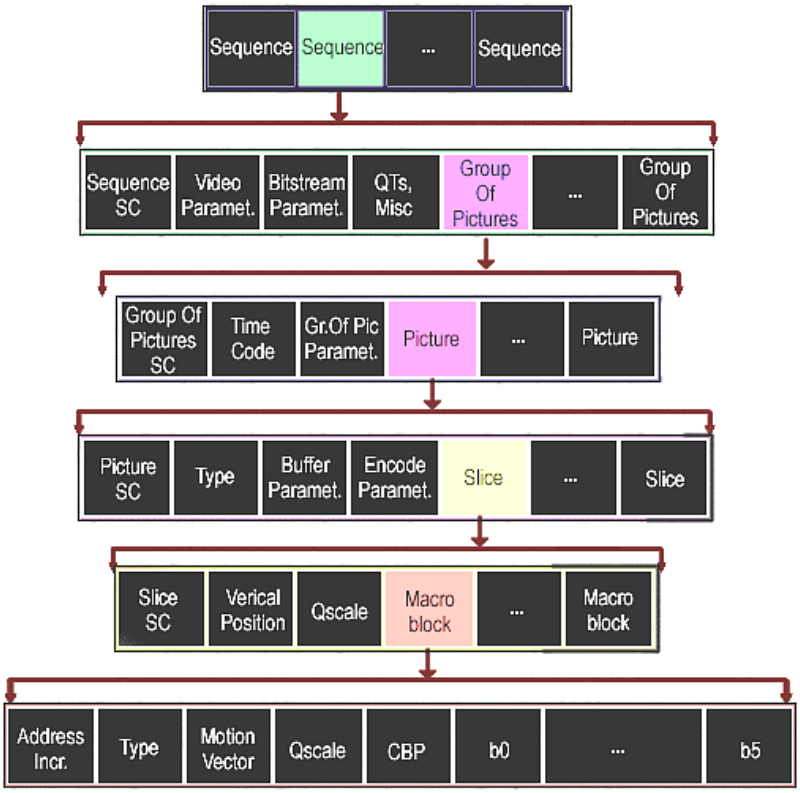
Die I-(Intra coded)-Frames enthalten am meisten Bildinformationen. In ihnen ist das gesamte Bild kodiert.

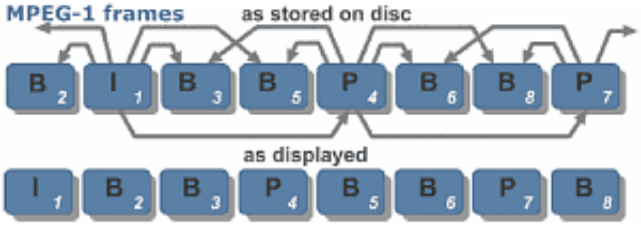
P-Frames

P-(Predicted)-Frames enthalten nur Differenzinformationen zu einem vorhergesagten Wert und sind daher ohne vorigen I-Frame (bzw. in der Folge dann auch P-Frame) nutzlos.

B-Frames

B-(Bidirectionally predicted)-Frames enthalten sogar nur Differenzinformationen zu dem aus dem vorherigen und dem nachfolgenden I-Frame (bzw. P-Frame) vorhergesagten Werten und brauchen daher beide.




Beim Dekodieren werden die Bilder zuerst in der Reihenfolge, in der sie im Stream ankommen (oben) dekodiert und anschließend in die richtige Reihenfolge zum Anzeigen (unten) gebracht.

## MPEG2

### Unterschiede zwischen MPEG-1 und MPEG-2?

Fest definierte Sequenzen von Bildarten, die eine minimale Ende-zu-Ende-Verzögerung bei gegebener Datenrate ermöglichen.

MPEG-2 unterstützt niedrigere Audio-Abtastraten.

MPEG-2 unterstützt interlaced video.

MPEG-2 unterstützt höhere Bandbreite.

MPEG-2 unterstützt höhere Bildauflösung.

MPEG-2 unterstützt Skalierung.

## MPEG2

**Welche Vor- und Nachteile besitzt MPEG-2 im Vergleich zu anderen Formaten wie MJPEG („Motion JPEG“) und H.261?**

| | **MPEG-2** | **MJPEG** (kein Standard!!!) | **h.261** |
|---|---|---|---|
| **fps** | variabel (8 Stück) | nicht standardisiert | feste fps (30/1001) |
| **Farbformat** | | nicht standardisiert | CCIR 601 (YUV 2:2:1) |
| **Frame-Typen** | P-/ B-/ I-Frames | I-Frames | InterFrames (~P-) |
| **Dec/ Enc** | Asymmetrie mgl. | asymmetrisch | asymmetrisch |
| **Audio** | Mehrkanal | nicht standardisiert | - |
| **Echtzeitkompression** | nein | ja (mit Chips -> DV) | ja |
| **Bandbreite** | 2-80 Mbit/ s | nicht standardisiert | n*64KBit (ISDN-Knl) |

**MPEG-2 unterstützt mehrere Ebenen („Levels“) und Profile. Welche sind das und wo werden sie eingesetzt?**

Profile:

- simple profile: nicht skalierbar, keine B-Frames, 4:2:0
- main profile: nicht skalierbar, B-Frames, 4:2:0
- SNR-skalierbar: SNR-skalierbar, B-Frames, 4:2:0
- räumlich skalierbar: SNR- und räumlich skalierbar, B-Frames, 4:2:0
- high profile: SNR- und räumlich skalierbar, B-Frames, 4:2:0, 4:2:2

Skalierung:

Die Bilder werden in verschiedenen Qualitätsstufen kodiert. Dabei werden unterschieden:

a) räumliche Skalierung (unterschiedliche Bildgrößen)

b) Skalierung der (Bildwiederhol-)Rate

c) Amlitudenskalierung (Auflösung der Quantisierung der DCT-Koeffizienten)

Levels:

- high level
- high-1440-level
- main-level
- low level

Vor allem das main-Level findet seine Anwendung in der Übertragung digitaler Videos über Kabel, Satellit und andere Broadcast-Kanäle 2-80MB/ s, digitale Speicherung und andere Kommunikationsanwendungen.

main und high profile werden auch für hochqualitatives Fernsehen verwendet.

| | Namen der Profile | Simple Profile | Main Profile | SNR-skalierbares Profil | Räumlich skalierbares Profil | High Profile |
|---|---|---|---|---|---|---|
| Profile | Eigenschaften der Profile | keine B-Frames | B-Frames | | | |
| | | 4:2:0 | | | | 4:2:0 oder 4:2:2 |
| | | Nicht skalierbar | | SNR-skalierbar | SNR- oder räumlich skalierbar | |
| Level - Ebenen | High Level 1920 Pixel/Zeile 1152 Zeilen | | ≤ 80 MBit/s | | | ≤ 100 MBit/s |
| | High-1440 Level 1440Pixel/Zeile 1152 Zeilen | | ≤ 60 MBit/s | | ≤ 60 MBit/s | ≤ 80 MBit/s |
| | Main Level 720 Pixel/Zeile 572 Zeilen | ≤ 15 MBit/s | ≤ 15 MBit/s | ≤ 15 MBit/s | | ≤ 20 MBit/s |
| | Low Level 352 Pixel/Zeile 288 Zeilen | | ≤ 4 MBit/s | ≤ 4 MBit/s | | |

## CodeComments

### main (avi2mpg1.c):

- Parameter auf ihre Default-Werte

- Kommandozeilenargumente parsen, MMX-Verfügbarkeit checken

- .avi-Quelldatei öffnen und deren Header-Information für das Bild (Bildgröße, Framerate, Farbformat etc.) auswerten.

- dasselbe für Audio

- avi2m1v: die Video-Kodierung

- avi2mp2: die Audio-Kodierung.

Am Ende werden beiden Streams mit mplex gemultiplext.

### avi2m1v.c:

In dieser Datei wird alles für den Encoding-Vorgang nötige intialisiert:

- Die eingegebenen Parameter auf Gültigkeit geprüft (setparam())

- Eventuell die Motion-Detection-Matrix initialisiert (sonst auch als Parameter in einem File anzugeben) (setparm())

- Die Quantisierungsmatix gefüllt (initquantmat())

- Die Videogröße an die Blockgröße angepasst (Höhe und Breite = 16*x; 4x4 = ein Block; init()/ setparm())

- Speicher alloziert für die Frames (init())

- Frame- und Bitrate gesetzt (init())

### putseq.c:

Hier werden die Daten encodiert.

- die GOPs werden aufgebaut und in den Sequence-Layer geschrieben

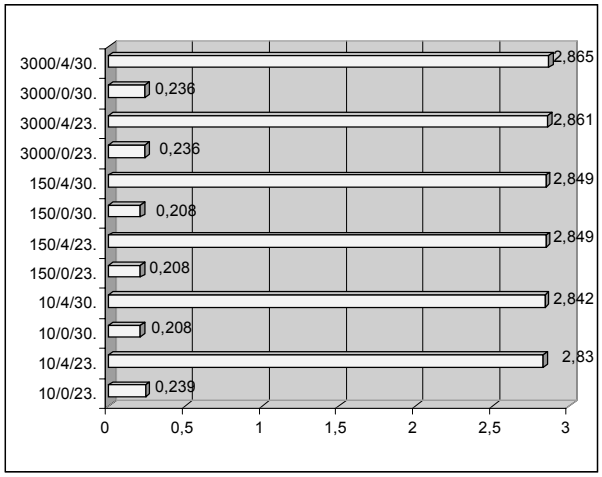
buchstaben.avi und tiger.avi waren nach mpeg-1 zu kodieren.

## Praxis

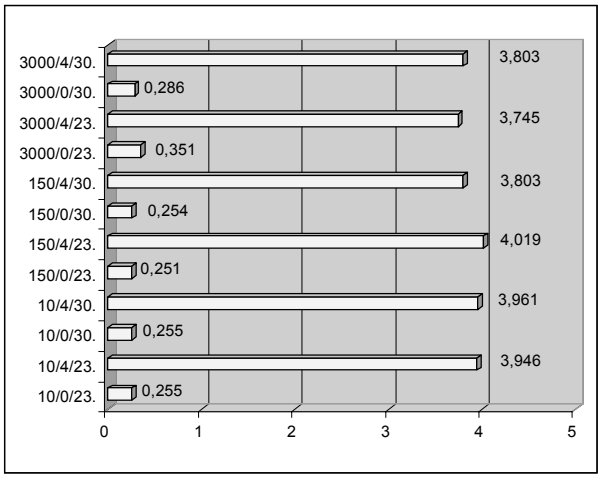
Enkodierzeit:

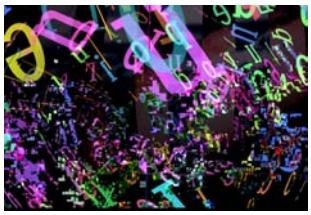
average frames/ second

tiger.avi:




buchstaben.avi:



Wie man hier sieht, hat der Encoder manchmal leichte Pobleme mit der Kodierung gehabt...

Wie zu erwarten liegt die Kodierzeit bei Kodierung mit maximaler (Stufe 4) Bewegungs-Detektion deutlich höher (ca. 11mal) als ohne.

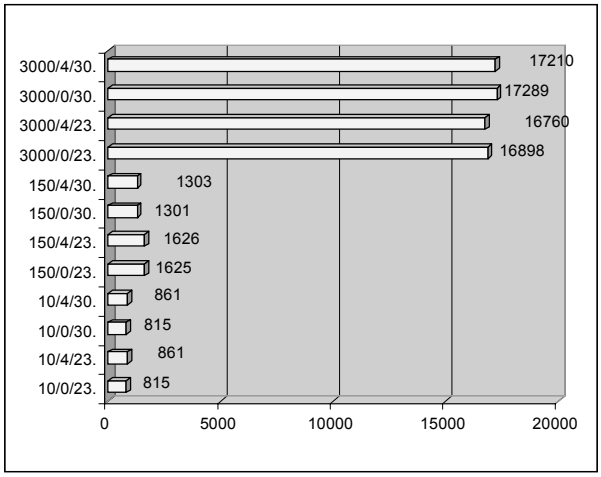
Etwas überraschend ist auf den ersten Blick vielleicht, dass es bei der Kodierzeit kaum auf die ZielByteRate ankommt. Das liegt aber wohl daran, dass der Parameter wohl hauptsächlich bei der Quantisierung einfließt. (?)

## Praxis

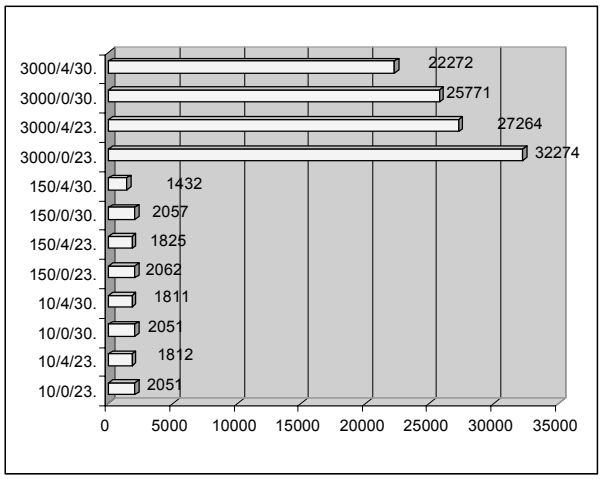
Dateigrößen:

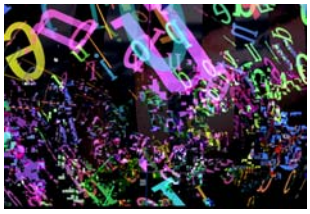
in kB

tiger.avi:




buchstaben.avi:



Hier zeigt sich, dass die Motion-Detection bei ähnlichen Bildinhalten (buchstaben.avi) tatsächlich nochmal eine deutliche Dateigrößenverringerung (10%) bringt. Da lohnt sich die Wartezeit beim Encodieren.

Bei stark unterschiedlichen Inhalten, wie beim tiger.avi, dagegen bringt sie fast Nichts sondern im Gegenteil sogar manchmal größere Dateien.

## Praxis

subjektive Beurteilung

Einordnung der Qualität der erzeugten Files auf einer Skala von 0(schlecht) bis 5(Referenzmaterial).

Die Framerate wirkt sich nicht sichtbar auf die Qualität aus, das Video wird bei 30fps einfach schneller gespielt, als bei 23.976 fps.

buchstaben.avi

Auch die eingeschaltete Motion-Detection wirkt sich bei den Buchstaben trotz geringerer Dateigröße nicht sichtbar negativ aus, weshalb wir für jede ByteRate nur eine Bewertung haben.

Das buchstaben.avi zeigt bunte Buchstaben mit scharfen Kanten, die durch die Gegend fliegen. Scharfe Kanten sind generell ein Problem bei der JPG-Komprimierung und das zeigt sich auch. Bei ByteRate 10 sind deutliche Artefakte zu erkennen sowohl an den Rändern, als auch innerhalb der Buchstaben. Deshalb auch hier unsere Bewertung 0.

Bei 150 Byte/ s ist das Ganze nur unmerklich besser, deshalb auch hier nur eine 1, weil die Artefakte noch extrem stören.

Bei 3000 Byte/ s ist das Bild dagegen gut quantisiert. In der Bewegung ist es vom Original kaum noch zu unterscheiden, daher hier eine 4.

tiger..avi

Anders sieht es beim tiger.avi aus. Hier bringt die Motion-Detection keinen Gewinn bei der Dateigröße und liefert sogar schlechtere Bilder, weil durch die Blockverschiebung zusätzliche Artefakte entstehen.

Bei 10 Byte/ s ohne Motion-Detection sind vor allem auf hellen Flächen Artefakte zu erkennen. Unsere Bewertung 1.

Mit Motion-Detection sind sogar noch mehr Artefakte da, daher hier 0.

Bei 150 Byte/ s ohne Motion-Detection ist das Bild schon deutlich besser, aber es sind immer noch sichtbare Artefakte da, daher 3.

Hier ist das Bild auch mit Motion-Detection ungefähr gleichgut, also auch 3.

Bei 3000 Byte/ s ohne Motion-Detection ist das Bild fast so gut wie das Original, nur ab und zu gibt es noch ein paar Artefakte (besonders bewegungsreiche Stellen -> Bandbreite) => 4.

Mit Motion-Detection entstehen ein paar mehr Fehler, aber nicht sehr störend, daher nochmal 4.

Es ist anhand der Bilddaten des tiger.avizuerst verwunderlich, dass die Motion-Detection zu keiner Datenreduktion führt. Da aber viele Teile des Bildes über einen längeren Zeitraum statisch sind, muss da fast nichts codiert werden innerhalb einer GroupOfPictures und daher gibt es einfach zu wenige Blöcke, die per Bewegungsvektor irgendwo hin könnten (<-> buchstaben.avi).