

Implementierung von Kompressionsverfahren

Uwe Baier

Institut für theoretische Informatik
Universität Ulm

14.10.2015

- 1 Einführung
- 2 Themen
 - BWT – LZ77 – Kombination
 - JPEG
 - MPEG
- 3 Abschließende Anmerkungen

Einführung

Kompressionsverfahren werden eingesetzt bei...

- Komprimierung von Dateien (zip)
- Datenbanken
- bestimmten Datenträgern (z.B. CD, DVD, ...)
- Audio
- Video
- Bildern

Ohne Kompressionsverfahren könnten viele Dienste heute nicht angeboten werden, z.B. Streams, Filme (DVD/Blu-ray), Software, ...

In diesem Projekt

- betrachten älterer und neuerer Kompressionsverfahren, sowohl verlustfrei als auch verlustbehaftet
- Verfahren selber implementieren
- Erfahrungen sammeln
- Fine–Tuning der Verfahren durch Experimente
- Auch eigene Themenvorschläge sind möglich

Formales

- Projekt: 8 LP
- Treffen nach Vereinbarung
- Voraussichtliches Projektende: Ende WS15-16, Anfang bis Mitte SS16
- pro Thema 2 – 3 Personen

Aufgaben

- Lauffähige Implementierung erstellen
- Verfahren mit verschiedenen Parametern testen
- Kleinen Report erstellen, in dem Verfahren, eigene Strategien und experimentelle Resultate enthalten sind
- Präsentieren der Ergebnisse (Termin je nach Projektstand und Vereinbarung)

BWT – LZ77 – Kombination

- BWT: Vorstufe für verlustfreie Kompressionsverfahren
- LZ77: verlustfreies Kompressionsverfahren
- BWT – LZ – Kombination: Stärken beider Verfahren vereinigen

Burrows – Wheeler – Transformation

Gegeben: Text $S = \text{ctaataatg\$}$

Suffixe von S :

i	$S[i-1]$	S_i
0	\$	ctaataatg\$
1	c	taataatg\$
2	t	aataatg\$
3	a	ataatg\$
4	a	taatg\$
5	t	aatg\$
6	a	atg\$
7	a	tg\$
8	t	g\$
9	g	\$

Burrows – Wheeler – Transformation

Gegeben: Text $S = \text{ctaataatg\$}$

Suffixe von S :

i	$S[i-1]$	S_i
0	\$	ctaataatg\$
1	c	taataatg\$
2	t	aataatg\$
3	a	ataatg\$
4	a	taatg\$
5	t	aatg\$
6	a	atg\$
7	a	tg\$
8	t	g\$
9	g	\$

lexikographisch sortiert:

i	$SA[i]$	$S[SA[i]-1]$	$S_{SA[i]}$
0	9	g	\$
1	2	t	aataatg\$
2	5	t	aatg\$
3	3	a	ataatg\$
4	6	a	atg\$
5	0	\$	ctaataatg\$
6	8	t	g\$
7	1	c	taataatg\$
8	4	a	taatg\$
9	7	a	tg\$

Burrows – Wheeler – Transformation

Gegeben: Text $S = \text{ctaataatg\$}$

Suffixe von S :

i	$S[i-1]$	S_i
0	\$	ctaataatg\$
1	c	taataatg\$
2	t	aataatg\$
3	a	ataatg\$
4	a	taatg\$
5	t	aatg\$
6	a	atg\$
7	a	tg\$
8	t	g\$
9	g	\$

lexikographisch sortiert:

i	$SA[i]$	$S[SA[i]-1]$	$S_{SA[i]}$
0	9	g	\$
1	2	t	aataatg\$
2	5	t	aatg\$
3	3	a	ataatg\$
4	6	a	atg\$
5	0	\$	ctaataatg\$
6	8	t	g\$
7	1	c	taataatg\$
8	4	a	taatg\$
9	7	a	tg\$

- Suffixarray (SA): Startpositionen der Suffixe von S in lexikographisch sortierter Reihenfolge
- Burrows – Wheeler – Transformation (BWT): zyklisch vorheriger Buchstabe eines Suffixes in SA, $BWT[i] = S[SA[i] - 1 \text{ mod } n]$

BWT – Eigenschaften


- Originaltext kann aus BWT in linearer Zeit wiederhergestellt werden
- lange Runs gleicher Buchstaben:
 $BWT[i] = BWT[i + 1] = \dots = BWT[i + k]$ für große k
 \Rightarrow Run – Length Encoding / MTF – Transformation +
Huffmann Kodierung liefern komprimierte Darstellung
- Gute Kompression bei vielen Repeats
- In linearer Zeit berechenbar (per Suffixarray)
- Einsatz z.B. in bzip2

Lempel – Ziv 77

Gegeben: Text $S = \text{ctaataatg\$}$

Idee: Stelle Text als einzelne Zeichen oder als Wiederholung bereits betrachteter Textstücke dar

0	1	2	3	4	5	6	7	8	9
c	t	a	a	t	a	a	t	g	\$



Lempel – Ziv 77

Gegeben: Text $S = \text{ctaataatg}\$$

Idee: Stelle Text als einzelne Zeichen oder als Wiederholung bereits betrachteter Textstücke dar

0	1	2	3	4	5	6	7	8	9
c	t	a	a	t	a	a	t	g	\$

Zu den Teilstücken (*LZ – Faktoren*):

- *src*: vorherige Position der Wiederholung, oder Buchstabe bei $length = 0$
- *length*: Länge eines Faktors (bei 0 enthält *src* einen Buchstaben)

Lempel – Ziv 77

Gegeben: Text $S = \text{ctaataatg\$}$

Idee: Stelle Text als einzelne Zeichen oder als Wiederholung bereits betrachteter Textstücke dar

0	1	2	3	4	5	6	7	8	9
c	t	a	a	t	a	a	t	g	\$

Zu den Teilstücken (*LZ – Faktoren*):

- *src*: vorherige Position der Wiederholung, oder Buchstabe bei $length = 0$
- *length*: Länge eines Faktors (bei 0 enthält *src* einen Buchstaben)

LZ – Faktoren für $S = \text{ctaataatg\$}$ im Format (*src*, *length*):

('c', 0) ('t', 0) ('a', 0) (2, 1) (1, 4) ('g', 0) ('\$ ', 0)

LZ77 – Eigenschaften

- Originaltext kann aus LZ – Faktoren in linearer Zeit wiederhergestellt werden
- Gute Kompression bei langen Repeats
- In linearer Zeit berechenbar (per Suffixarray)
- Einsatz z.B. in `gzip`

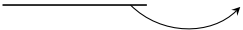

BWT – LZ77 – Kombination

Idee: Ersetze LZ – Faktoren aus Text durch Spezialzeichen,
speichere Faktoren, berechne BWT aus neuem Text \tilde{S}

0	1	2	3	4	5	6	7	8	9		0	1	2	3	4	5	6
c	t	a	a	t	a	a	t	g	\$	\Rightarrow	c	#	a	a	t	g	\$
										(4,4) \rightarrow							

BWT – LZ77 – Kombination

Idee: Ersetze LZ – Faktoren aus Text durch Spezialzeichen, speichere Faktoren, berechne BWT aus neuem Text \tilde{S}

0	1	2	3	4	5	6	7	8	9		0	1	2	3	4	5	6
c	t	a	a	t	a	a	t	g	\$	\Rightarrow	c	#	a	a	t	g	\$
																	

BWT - LZ - Kombination:

i	BWT[i]	\tilde{S}_i	
0	g	\$	
1	c	#aatg\$	$\rightarrow (4,4)$
2	#	aatg\$	
3	a	atg\$	
4	t	g\$	
5	a	tg\$	

BWT – LZ – Kombination — Ziele

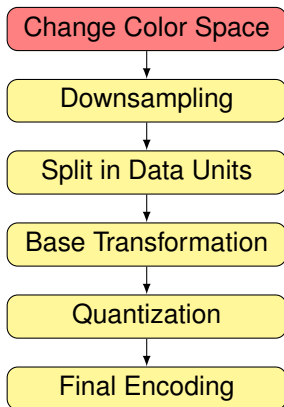
- Implementierung BWT – LZ + MTF + Huffman Kodierung und Dekodierung
- Vergleich mit gzip (LZ77 + Huffman), bzip2 (BWT + MTF + Huffman), ...
- Performance: asymptotisch lineare Laufzeit mit eventuellen Optimierungen, aber nicht primäres Ziel
- Kompression: primäres Ziel, Strategien zur Wahl der richtigen LZ – Faktoren entwickeln und testen

JPEG

Verlustbehaftetes Kompressionsverfahren für Bilder



JPEG – Kodierung — Change Color Space

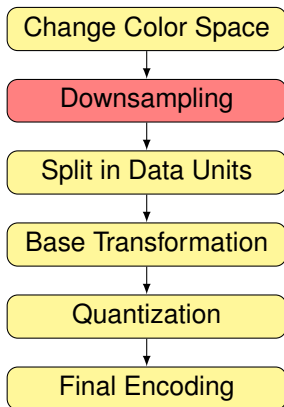


Farbraum des Bildes von RGB in YCBCR konvertieren:

Veränderungen der Chrominanz (CB und CR) werden vom menschlichen Auge weniger wahrgenommen

⇒ CB und CR – Komponenten können stärker komprimiert werden, ohne dass das menschliche Auge zu viel Qualitätsverlust bemerkt

JPEG – Kodierung — Downsampling

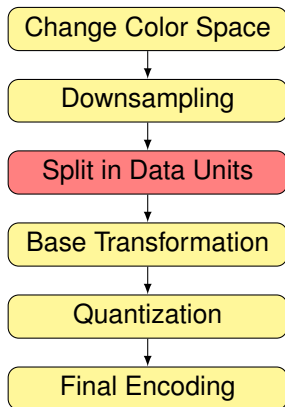


Pixelblöcke werden zu einem Pixel verschmolzen

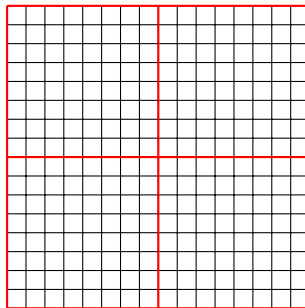
- Nur für Chrominanzfarben (CB und CR)
- Üblich: 2×2 Pixelblock \rightarrow 1 Pixel
- Neue Pixelwerte per arithmetischem Mittel

Effekt: Bildinformation der entsprechenden Farbkomponente wird reduziert

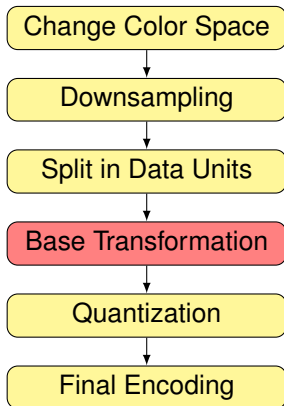
JPEG – Kodierung — Split in Data Units



Zusammenfassen von Pixelgruppen zu *Data Units*, um Performance von nachfolgenden Operationen zu verbessern (üblicherweise 8×8)



JPEG – Kodierung — Base Transformation



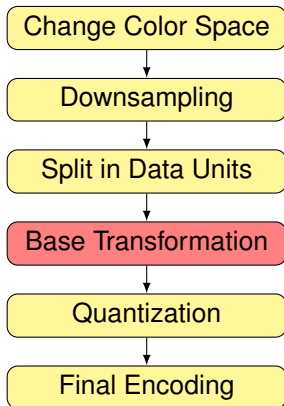
Übliche Darstellung eines Pixelblocks: per Koeffizienten von Einheitsmatrizen

$$\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \blacksquare & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} = \frac{1}{2} \cdot \begin{array}{|c|c|c|} \hline \blacksquare & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} + \frac{1}{2} \cdot \begin{array}{|c|c|c|} \hline \square & \blacksquare & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} + \frac{1}{2} \cdot \begin{array}{|c|c|c|} \hline \square & \square & \blacksquare \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} + \dots$$

⇒ Koeffizientenmatrix

1/2	1/2	1/2
1/2	1	1/2
1/2	1/2	1/2

JPEG – Kodierung — Base Transformation



Übliche Darstellung eines Pixelblocks: per Koeffizienten von Einheitsmatrizen

$$\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \blacksquare & \square \\ \hline \square & \square & \square \\ \hline \end{array} = \frac{1}{2} \cdot \begin{array}{|c|c|c|} \hline \blacksquare & \square & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} + \frac{1}{2} \cdot \begin{array}{|c|c|c|} \hline \square & \blacksquare & \square \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} + \frac{1}{2} \cdot \begin{array}{|c|c|c|} \hline \square & \square & \blacksquare \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} + \dots$$

⇒ Koeffizientenmatrix

1/2	1/2	1/2
1/2	1	1/2
1/2	1/2	1/2

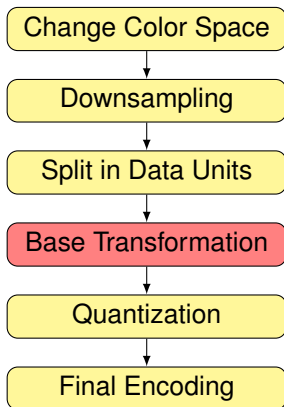
Oftmals Farbverläufe in Bildern ⇒ Stelle Bild per Koeffizienten von Erhebungen dar:

$$\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \blacksquare & \square \\ \hline \square & \square & \square \\ \hline \end{array} = \frac{1}{2} \cdot \begin{array}{|c|c|c|} \hline \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & \blacksquare & \blacksquare \\ \hline \end{array} + \frac{1}{2} \cdot \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \blacksquare & \square \\ \hline \square & \square & \square \\ \hline \end{array} + 0 \cdot \begin{array}{|c|c|c|} \hline \square & \square & \blacksquare \\ \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} + \dots$$

⇒ Koeffizientenmatrix

1/2	1/2	0
0	0	0
0	0	0

JPEG – Kodierung — Base Transformation

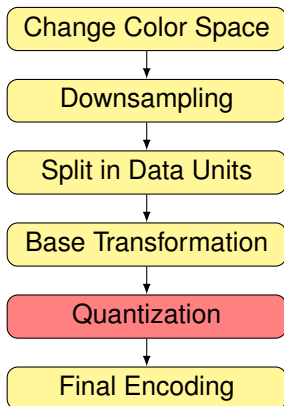


Vorteile der Basistransformation bei guter Wahl der Darstellungsmatrizen:

- Hauptbildinformation im oberen linken Eck der Koeffizientenmatrix
- kleine Werte in der unteren rechten Ecke der Koeffizientenmatrix
⇒ Werte vernachlässigbar, Kompressionsmöglichkeit

Basistransformation bei JPEG:
Diskrete Cosinus-Transformation (DCT)

JPEG – Kodierung — Quantization



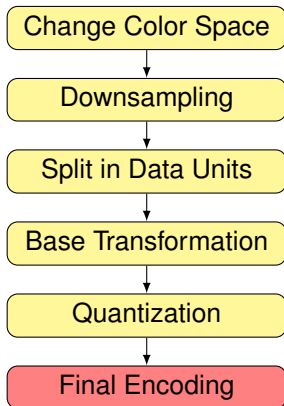
Quantisierung:

- Teile jeden Eintrag $D_{i,j}$ einer Data Unit durch einen Faktor $A_{i,j}$ und runde auf den nächsten Integer
- Quantisierungsmatrix A separat für Luminanz und Chrominanz
- Durch Basistransformation sind Werte im oberen linken Eck wichtiger als Werte im unteren rechten Eck

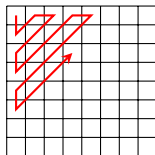
Empfohlene Quantisierungsmatrix (CB & CR):

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

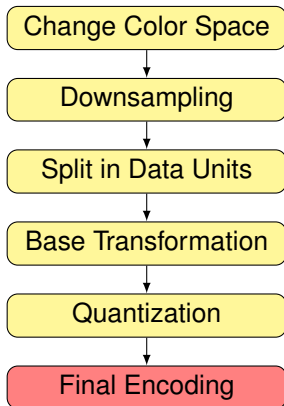
JPEG – Kodierung — Final Encoding



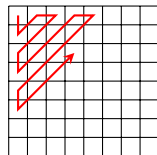
① Data units “flach-klopfen“ per Zig-Zag-Kodierung



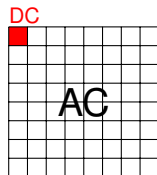
JPEG – Kodierung — Final Encoding



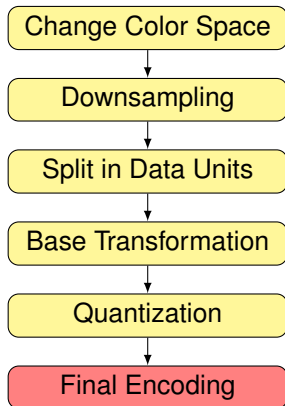
1 Data units “flach-klopfen“ per Zig-Zag-Kodierung



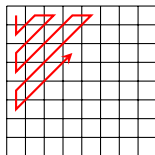
2 DC - Komponenten benachbarter Data Units korrelieren stark
 ⇒ Speichere statt Wert Differenz zum Wert des linken Nachbarn



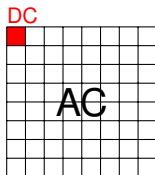
JPEG – Kodierung — Final Encoding



1 Data units “flach-klopfen” per Zig-Zag-Kodierung



2 DC - Komponenten benachbarter Data Units korrelieren stark
 ⇒ Speichere statt Wert Differenz zum Wert des linken Nachbarn



3 Kodiere Zahlenfolge mit Kombination aus Run-Length-Coding und Huffman

JPEG — Ziele

- eigene Implementierung eines JPEG – ähnlichen Formats
- Funktionsumfang: Bitmap in eigenes Format kodieren, Format in Bitmap dekodieren
- Variieren verschiedener Verfahren:
 - Data Unit Größe (4×4 , 8×8 , 16×16 , ...)
 - Basistransformation (DCT, DFT, ...)
 - Quantisierung (eigene Rechenvorschrift für Quantisierungsmatrix)
- Bewertung der Verfahrenskombination anhand experimenteller Daten:
 - Kompression
 - Performance
 - Bildqualität
- Empfohlen: C++ mit beliebiger Picture Library (Magick++, Simd, GD2, ...)

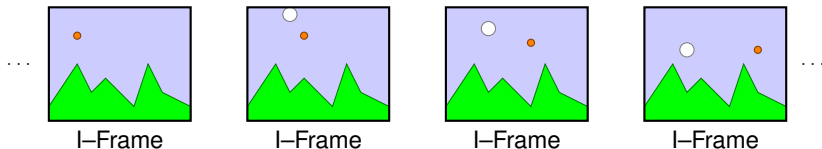
MPEG

Verlustbehaftetes Kompressionsverfahren für Videos,
benutzt JPEG um Einzelbilder (Frames) abzuspeichern



MPEG — Intra Frames

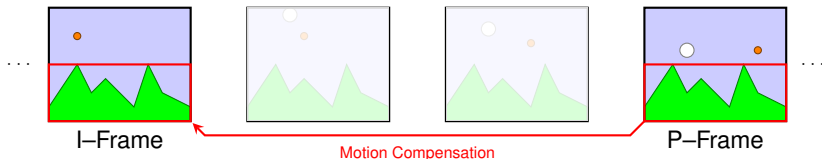
- Intra Frames (I-Frames): Ganzes Einzelbild wird als JPEG gespeichert
- Naiver Ansatz: Kodiere jeden Frame als I-Frame



Oftmals bei aufeinanderfolgenden Frames:
nur leichte Bewegung der Objekte, nahezu keine Änderung
⇒ Motion Compensation

MPEG — Predictive Coded Frames

- Predictive Coded Frames (P-Frames): Bestimmte Bildbereiche, die sehr ähnlich zu Bildbereichen des vorherigen I oder P Frames sind, ähnliche Bereiche vom Bild abziehen (Wiederauffinden mittels Vektor)

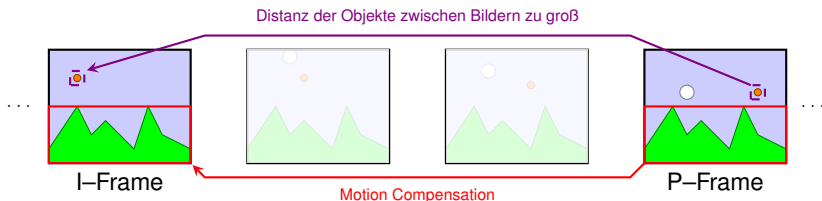


Effekt: zu speicherndes Restbild wird sehr “glatt“

⇒ bessere JPEG – Kompression

MPEG — Predictive Coded Frames

- Predictive Coded Frames (P-Frames): Bestimmte Bildbereiche, die sehr ähnlich zu Bildbereichen des vorherigen I oder P Frames sind, ähnliche Bereiche vom Bild abziehen (Wiederauffinden mittels Vektor)
- Bildbereiche des referenzierten Frames sollten nicht zu weit voneinander entfernt sein (Performance)

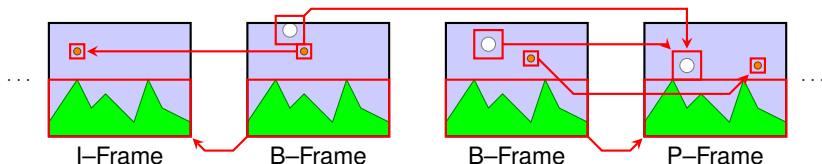


Effekt: zu speicherndes Restbild wird sehr “glatt“

⇒ bessere JPEG – Kompression

MPEG — Bidirectionally Predictive Coded Frames

- Bidirectionally Predictive Coded Frames (B-Frames): Wie P-Frames, nur Motion Compensation mit vorherigem und nächstem I- oder P-Frame
- Beispielweise nützlich, falls neue Objekte erscheinen



Effekt: Sehr gute Kompression für Restbild

MPEG — Motion Compensation

- Nutzung der Framearten:
 - I-Frames: Bei zu großen Veränderungen von Frames oder Qualitätsverlust
 - P-Frames: Bei “mittelmäßigen“ Veränderungen von Frames
 - B-Frames: Bei nur wenigen Veränderungen
- Werden nur I- und P-Frames genutzt, entsteht durch lange P-Frame-Folgen entweder schlechte Bildqualität (durch JPEG–Qualitätsverlust) oder schlechte Kompression
- B-Frames liefern gute Möglichkeit um sowohl Qualität als auch Kompression sicherzustellen
- Motion Compensation technisch: Teile Bild in Blöcke (Ideal: vielfaches von JPEG–Blockeinteilung), und speichere für jeden Block einen Vektor, der den Offset zum Bildbereich des referenzierten Frame beschreibt

MPEG – Display Order vs. Bitstream Order

Display Order: Ordnung, in der die Frames vom Benutzer angesehen werden

I	B	B	P	B	B	I	B	B	P
1	2	3	4	5	6	7	8	9	10

Bitstream Order: Ordnung um Frames zu dekodieren, je nach Abhängigkeiten der Frames

I	P	B	B	I	B	B	P	B	B
1	4	2	3	7	5	6	10	8	9

Vorteil der Bitstream Order: Frames liegen wie zur Dekodierung benötigt vor, Einsatz von Multithreading:

- 1 I-Frame dekodieren
- 2 abhängige P-Frames dekodieren
- 3 abhängige B-Frames dekodieren (parallel)

MPEG — Ziele

- Implementierung eines eigenen MPEG – Formats
- Fokus auf Motion Compensation (JPEG: Routine benutzen)
- Strategien zur Klassifizierung von Frames entwickeln
- Verschiedene Motion Compensation Algorithmen testen
- Messen von Kompression und Qualität mittels Testdaten
- Mediaplayer für eigenes MPEG – Format und Stream von Bitmap - Dateien bauen (mit Framesprungfunktion)
- Empfohlen: Java, nötige Funktionalität vorhanden

Abschließende Anmerkungen

- Hilfestellung wird wie benötigt gegeben
- Experimentelle Resultate in Eigenregie oder per vorhandener Benchmarks
- Falls Wunschthema gefunden: Mail an `uwe.baier@uni-ulm.de`, mit Teampartnern und deren E-mail – Adressen
- Auch eigene Themenvorschläge per Mail an `uwe.baier@uni-ulm.de`