Bitstromformate am Beispiel H.264 Annex B

Medientechnologie IL

Andreas Unterweger

Vertiefung Medieninformatik
Studiengang ITS
FH Salzburg

Sommersemester 2014

Begriffsdefinition I

- Bitstrom: (Kodierte) Sequenz von Bits
  - Im Encoder: Erzeugte Daten
  - Im Decoder: „Erwartete“ Daten
- Bitstromformat: Binärdatenformat (Aufbau und Struktur)
  - Im Encoder: Erzeugtes Datenformat
  - Im Decoder: Erwartetes Datenformat
- Bitstromformatspezifikation: (Standardisierte) Definition
  - Kodierung (Kompression, Endianness etc.)
  - Optionale Start- und/oder End-Kennungen
  - Optionale Header
  - ...

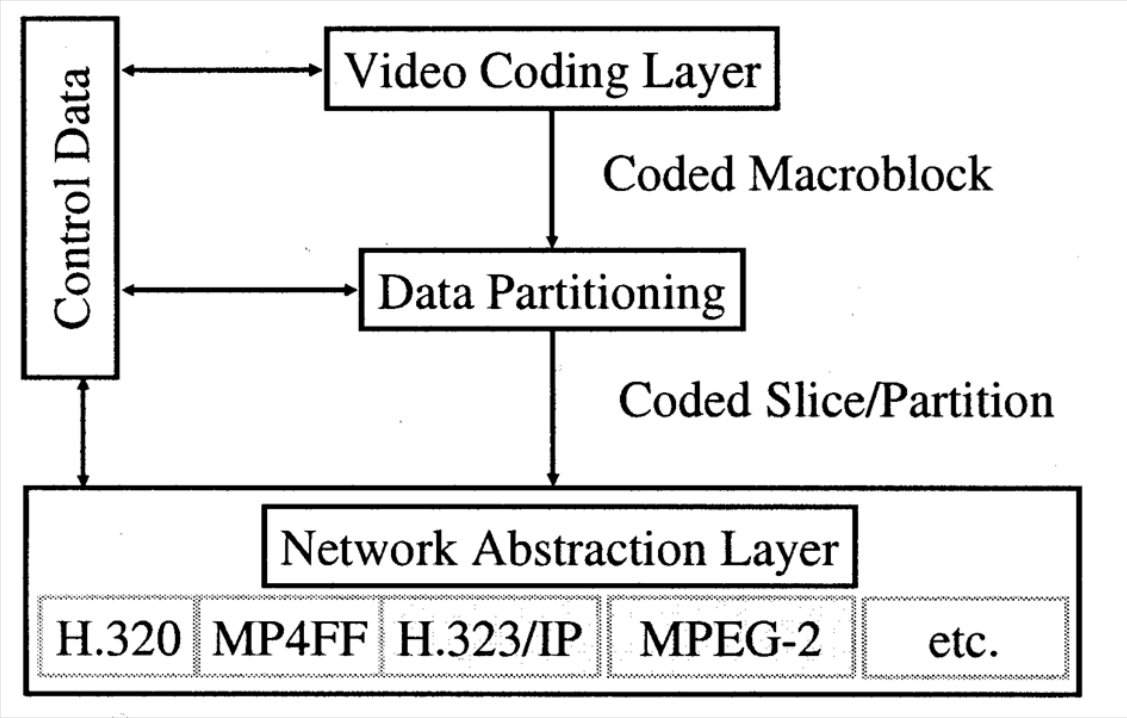
Begriffsdefinition II

- Syntaxelement: Sequenz von Bits, die als Einheit betrachtet wird
  - Länge kann konstant oder variabel sein
  - Länge kann von Kontext abhängen (z.B. anderen Syntaxelementen)
  - Muss nicht unbedingt an Bytegrenze beginnen
  - Kodierung ist vorgegeben
  - Beispiel: $\ldots \underbrace{0100\,0001}_{\text{'A'}} \ldots$
- Header: Sammlung von Syntaxelementen zur Beschreibung (des Formats) der Folgedaten
  - Anzahl der Syntaxelemente kann konstant oder variabel sein
  - Anzahl der Syntaxelemente kann von Kontext abhängen
- Startcode: Eindeutige Sequenz von Bits, die Bitstrom-, Header- und/oder Syntaxelementbeginn kennzeichnet
  - Länge und Bitfolge sind konstant
  - Beginnt meist an Bytegrenzen (zum einfacheren Auffinden)
  - Darf nicht im restlichen Bitstrom vorkommen $\rightarrow$ Escaping

Beispiele für Bitstromformate

- Für unkomprimierte Daten
  - WAVE (Waveform Audio File Format): (L)PCM-Audio
  - PBM (Portable Bitmap): Graustufenbilder
  - IYUV (Intel YUV): Videos im YCbCr-Farbraum ($\rightarrow$ Labor)
- Für komprimierte Daten
  - JFIF (JPEG File Interchange Format): JPEG-Bilder
  - ADIF (Audio Data Interchange Format): AAC-Frames
  - **H.264 Annex B: H.264-Videos**
- Für mehrere Datenströme $\rightarrow$ Containerformate

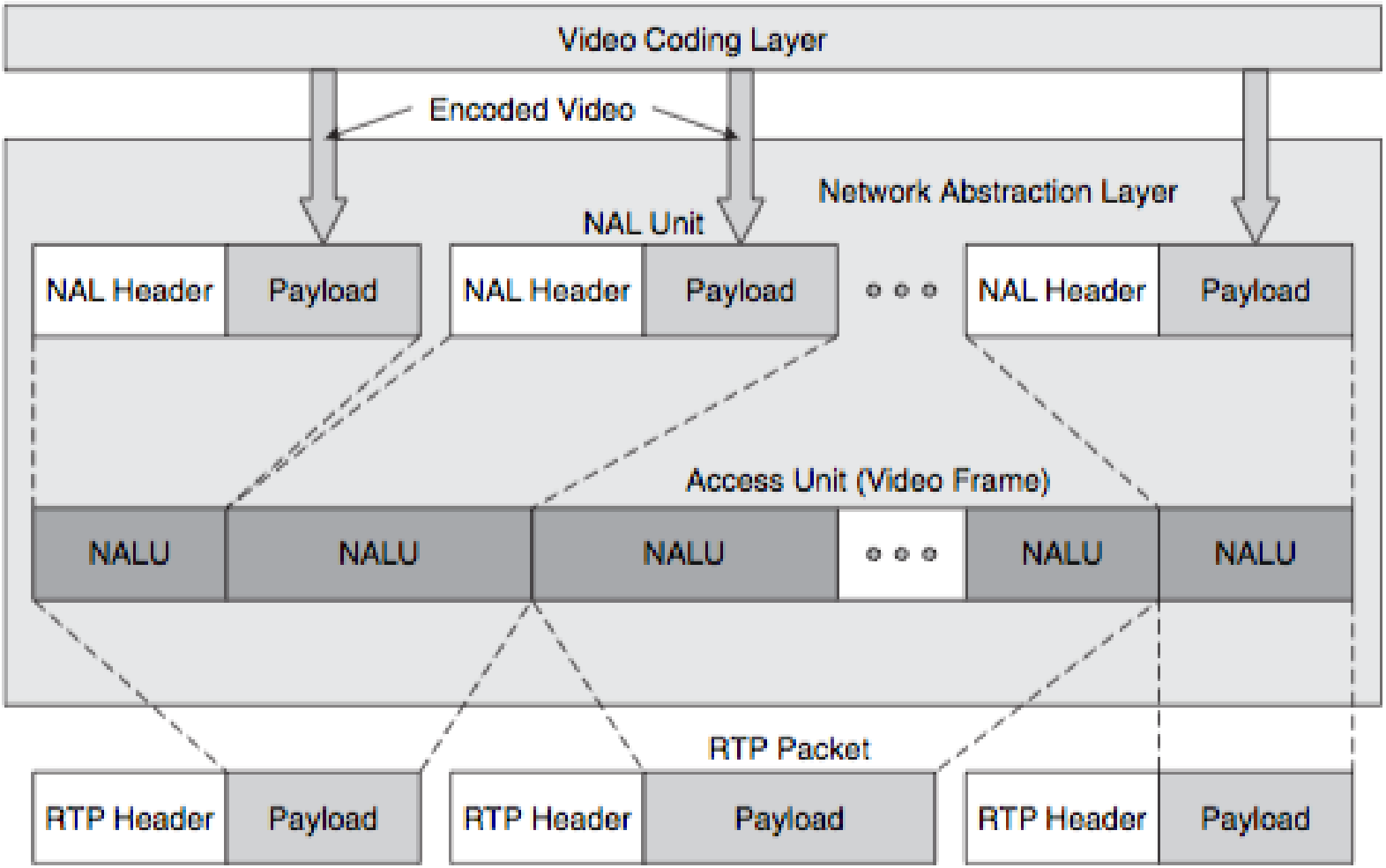
H.264-Architektur I: VCL/NAL



Quelle: `http://www2.engr.arizona.edu/~yangsong/h264.htm`

- Video Coding Layer (VCL): Videokodierung
- **Network Abstraction Layer (NAL): Datenübertragung**

H.264-Architektur II: NAL am Beispiel RTP



Quelle: `http://ciscovoiceguru.com/6306/cisco-tp-03/`

NALU-Struktur

- NAL kann auch Speicherung abstrahieren
- Annex B des H.264-Standards definiert Bitstromformat
- Kleinste Speichereinheiten: NAL Units (NALUs)
  - Startcode: `0x000001` (Vereinfachung)

  → Escaping notwendig (ohne Details)
  - Header: Prüfbit, NALU-Typ und Priorität
  - Daten: RBSP (Raw Byte Sequence Payload)

→ Annex-B-Bitstrom ist Sequenz von NALUs

Quelle: `http://codesequoia.wordpress.com/2009/10/18/h-264-stream-structure/`

NALU-Typen

- Verschiedene NALU-Typen (Auswahl):

NALU-Typ	**Beschreibung**
1	Slice (Teil eines Frames)
5	IDR-Slice (Instantaneous Decoding Refresh)
7	SPS (Sequence Parameter Set)
8	PPS (Picture Parameter Set)

- Slice: Enthält kodierte Makroblöcke (Details auf Folie 11)
- IDR-Slice: Enthält rein intrakodierte Makroblöcke
- SPS: Enthält Dekodierungsinformationen zur Videosequenz
- PPS: Enthält Dekodierungsinformationen zu mehreren Frames

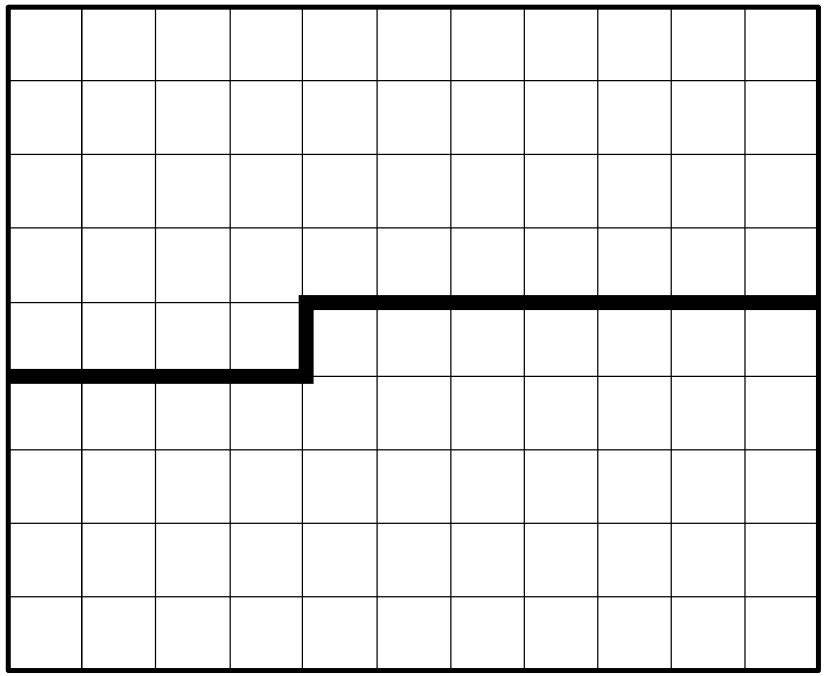
SPS- und PPS-Daten

- SPS-Daten (Auszug)
  - SPS-ID (mehrere Sequenzen pro Datei möglich)
  - Bildgröße und Randinformation (Cropping)
  - Profil und Level (Dekodierbarkeitsinformation)
  - Farbraum und Farbtiefe (nur in manchen Profilen)
  - Maximale Anzahl MC-Referenzbilder
  - (Optional) VUI (Video Usability Information): z.B. Framerate
- PPS-Daten (Auszug)
  - PPS-ID (Wechsel zwischen PPS möglich)
  - SPS-ID (ist einer Sequenz via SPS-ID zugeordnet)
  - Deblocking-Parameter
  - (Optional) Anordnung der Slices

Slices I

- Ein Frame besteht aus mehreren Slices
- Slices sind voneinander unabhängig

→ Parallele Dekodierung möglich



Quelle: ITU-T. Recommendation ITU-T H.264 – Advanced video coding for generic audiovisual services, April 2013

Slices II

- Eigenschaften
  - Ist einem PPS via PPS-ID zugeordnet
  - Ist unabhängig von anderen Slices desselben Frames
- Enthaltene Daten (Auszug)
  - Startflag: Enthält ersten Makroblock des Frames (ja/nein)?
  - Kodierreihenfolgezähler: Framenummer
  - Darstellungsreihenfolgezähler: POC (Picture Order Count)
  - Deblockingfiltereinstellungen
  - Kodierte Makroblöcke
- Kodierte Makroblockdaten (Auszug):
  - Anzahl übersprungener Makroblöcke
  - Pro Makroblock: Typ, Partitionierung, Koeffizienten, ...

Syntaxelementkodierung

- Verwendete Kodierungen (Auswahl):
  - Unkomprimiert (Länge konstant oder variabel)
  - **Exponential-Golomb-kodiert** (EG)
  - Entropiekodiert (CAVLC oder CABAC)
- Werte-Interpretation und -Semantik
  - Vorzeichenbehaftete Ganzzahl
  - Vorzeichenlose Ganzzahl
  - Index (vorzeichenlos) in vordefinierter Tabelle
- Headersyntaxelemente: Fast alle unkomprimiert oder EG-kodiert
  - Einfach zu lesen und zu verarbeiten
  - Overhead bei geringer Anzahl von Headern vernachlässigbar
- Datensyntaxelemente: Fast alle entropiekodiert
  - Aufwändig zu lesen (Großteil der Gesamtdaten)
  - Effizient $\rightarrow$ Reduktion der Datenrate

EG-Kodierung I

- Codewortlänge ist variabel
- Codewortlänge steigt mit Wert

→ Ideal bei geometrischer Verteilung

  - Kleine Werte sind sehr häufig
  - Große Werte sind sehr selten
  - Werte sind ganzzahlig(e Vielfache eines Basiswertes)

→ Anwendung bei Syntaxelementen, deren Werte (fast) geometrisch verteilt sind → Bitersparnis

- EG-Kodierung für vorzeichenlose (unsigned) Werte → UEG
- Abgeleitete Kodierung für vorzeichenbehaftete (signed) → SEG
  - Codewörter abwechselnd auf positive und negative Werte verteilen

  → SEG-Codewort länger als äquivalentes UEG-Codewort
  - $\forall x \in \mathbb{Z}^- : SEG(x) = UEG(2|x|), \forall x \in \mathbb{N}\backslash\{0\} : SEG(x) = UEG(2x-1)$

EG-Kodierung II

Wert	UEG-Codewort	SEG-Codewort
...	–	...
-4	–	0001001
-3	–	00111
-2	–	00101
-1	–	011
0	1	1
1	010	010
2	011	00100
3	00100	00110
4	00101	0001000
...	...	...

Effiziente Syntaxelementkodierung I: Prinzipien

- Effizienzmaximierung: Anzahl Bits pro Syntaxelements minimieren
  - Redundante Syntaxelemente vermeiden
  - Abhängigkeiten zwischen Syntaxelementen ausnutzen
  - Ungültige Werte aus Wertebereich ausschließen

  → Wertebereich skalieren und/oder verschieben
  - Gemeinsam genutzte Syntaxelemente in SPS/PPS auslagern
- Syntaxelementreduktion durch Fallunterscheidungen
  - Optionale Syntaxelemente (nur wenn benötigt)
  - Implizite Werte für Syntaxelemente (je nach Kontext)

  → Höherer Parsing- und Verifikationsaufwand
  - Häufige Fallunterscheidungen direkt in SPS/PPS auslagern

Effiziente Syntaxelementkodierung II: Beispiele

- SPS: `vui_parameters_present_flag`
  - Gibt an, ob VUI-Daten vorhanden sind

  → Zwei Möglichkeiten (ein Bit): `vui_parameters_present_flag`
- PPS: `pic_init_qp_minus26`
  - QP jedes Frames ist relativ zum PPS-QP kodiert
  - PPS-QP-Wertebereich: 1 bis 51: `pic_init_qp`
  - Mittlerer zu erwartender QP: $1 + \frac{51-1}{2} = 1 + \frac{50}{2} = 1 + 25 = 26$

  → Wertebereich um 26 verschieben: `pic_init_qp_minus26`
- SPS: `pic_width_in_mbs_minus1`
  - Bildbreite muss angegeben werden: `pic_width`
  - `pic_width` muss Vielfaches von 16 (Makroblockbreite) sein

  → Sechzehntel von `pic_width` speichern: `pic_width_in_mbs`
  - `pic_width_in_mbs` kann nicht null sein

  → Wertebereich um eins verschieben: `pic_width_in_mbs_minus1`

Effiziente Syntaxelementkodierung III: Weitere Beispiele

- SPS: `log2_max_frame_num_minus4`
  - Framenummer ist beschränkt (Wrap-Around): `max_frame_num`
  - `max_frame_num` muss eine Zweierpotenz sein

  → ld(`max_frame_num`) speichern: `log2_max_frame_num`
  - `max_frame_num` muss mindestens 16 sein

  → `log2_max_frame_num` muss mindestens ld(16)=4 sein

  → Wertebereich um vier verschieben: `log2_max_frame_num_minus4`
- Makroblock in Slice: `coeff_abs_level_minus1`
  - Speicherung von Transformationskoeffizienten notwendig: `coeff_level`
  - Vorzeichen werden getrennt gespeichert

  → abs(`coeff_level`) speichern: `coeff_abs_level`
  - Position der Nicht-Null-Koeffizienten wird separat signalisiert

  → `abs_coeff_level` kann nicht null sein

  → Wertebereich um eins verschieben: `coeff_abs_level_minus1`

Übersicht der Nachteile des Bitstromformats

- Kompakte Kodierung ist effizient, aber
  - Erschwert Parsing durch verschiedene Syntaxelementkodierungen
  - Erhöht Zeitbedarf für Parsing
  - Erfordert fehleranfällige Herleitung der Originalwerte
- Vielzahl optionaler Angaben macht Abspielen eines H.264-kodierten Videos (so gut wie) unmöglich
  - Ohne Framerate keine vernünftige Wiedergabe möglich
  - **Seeking/Random Access** schwierig bis unmöglich zu realisieren
  - **Zeitansteuerung** quasi unmöglich

  → Kaum ein H.264-Annex-B-Bitstrom kann per se wiedergegeben werden

  → H.264-Videos müssen in Container verpackt werden

  → Kaum ein Player spielt reine H.264-Annex-B-Bitströme ab

H.264-Annex-B-Abspielproblematik

- Definitionen
  - Random Access: Video ab einer gewählten Stelle wiedergeben
  - Seeking: Vor- bzw. Zurückspringen mittels Random Access
  - Zeitansteuerung: Sprung zu Framenummer und/oder Zeitpunkt
- Random Access in H.264 Annex B (allgemein)
  - P- und B-Frames hängen von anderen Frames ab
  - I-Frames können unabhängig dekodiert werden
  - IDR-Frames bilden Prädiktionsgrenze $\rightarrow$ Erlauben Random Access[1]
  - Problem: Nur erster Frame einer Sequenz muss IDR-Frame sein
- Zeitansteuerung in H.264 Annex B (allgemein)
  - Anzahl Frames wird nicht explizit gespeichert
  - Frames haben per se keine Zeitstempel
  - Kodierreihenfolge vs. Darstellungsreihenfolge
  - Bitstromposition vs. (theoretischer) Zeitstempel

$\rightarrow$ Random Access, Seeking und Zeitansteuerung nicht allgemein möglich

[1] Ausgenommen bei Open GOPs (ohne Details)

Danke für die Aufmerksamkeit!

Fragen?