The transcoder challenge: What is so difficult about building a transcoder for watermarking?

Andreas Unterweger

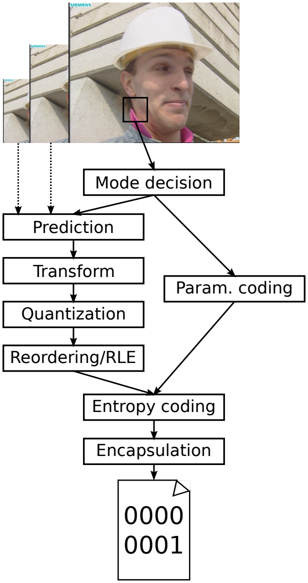
Department of Computer Sciences
University of Salzburg

March 9, 2012

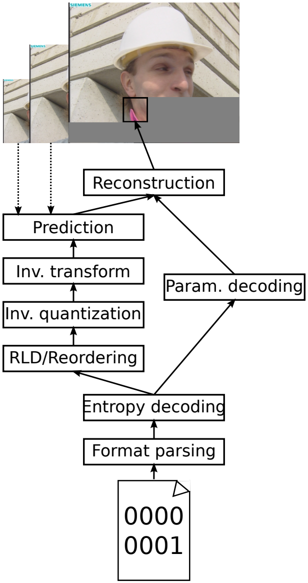
What do we want?

- A transcoder which
  - Replaces the values of selected syntax elements
  - Adapts the rest of the bit stream so it remains format compliant
  - Does not change anything else (structure/length preserving)
- Big picture: An application which
  - Gets an H.264 bit stream and a watermark as input
  - Embeds the watermark using the transcoder
  - Provides an interface for quality measurement

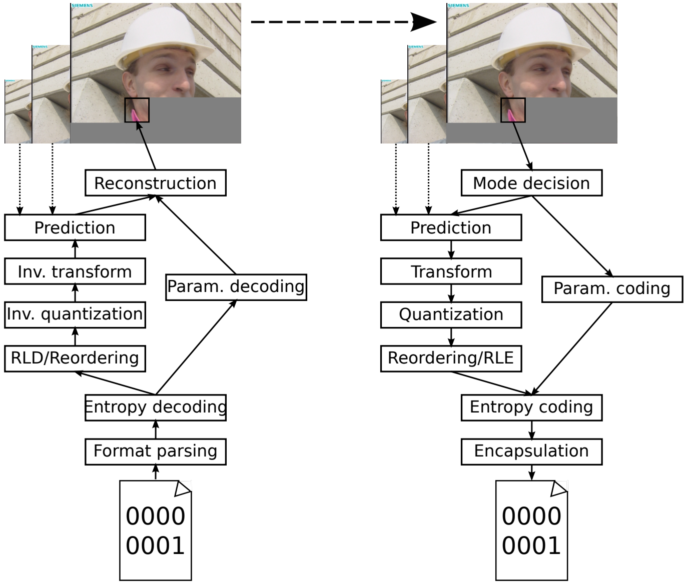
Term: Encoder



Term: Decoder

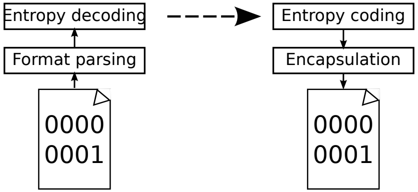


Term: Transcoder (classic)



Term: Bit stream transcoder

- Value modification and entropy re-encoding
- The actual minimum of what is required for our purposes
- Without entropy code adaptivity, it would be simple bit replacement



What is wrong with classic transcoders?

- They perform operations that we do not want/need
    - All pictures are completely decoded and re-encoded
    - The encoder expects to be configured
    - The encoder makes decisions of his own
- They do not perform operations that we want/need
    - Original encoder decisions are not preserved (sometimes considered)
    - Encoder decisions cannot be influenced at the required level of detail
- Changing them is hard
    - Few transcoders are open source
    - Those which are, bridge existing or modified decoders and encoders
    - Transcoders are built for full transcoding, not watermarking

Is there no ready-to-use solution?

- Standard transcoders are not built for (just) entropy re-encoding
- Watermarking is not at all a common transcoding application
- Very few people/companies need bit stream transcoders
- Very few people/companies build bit stream transcoders
- One known transcoder from the University of Ghent (being evaluated)

(Ab)using existing encoders and decoders

- Encoders and decoders are combined to (classic) transcoders
- Idea: combine only the parts we need to get a bit stream transcoder
- Issues:
    - Small number of open source encoders and decoders to choose from
    - Decoders are not built to decode only up to a certain level
    - Encoders are not built to encode only down from a certain level
    - Different implementations are very hard to bridge

Overview of selected open source H.264 encoders/decoders

Implementation	Encoder	Decoder	Speed	State
JM (JVT)	✓	✓	Slow	Mature
x264	✓	–	Fast	Mature
libavcodec	–*	✓	Fast	Mature
IPP** (Intel)	✓	✓	Fast	Mature
t264	✓	–***	Slow	Alpha

* Can use x264's library version when built with it
** From IPP code samples; relies on IPP libraries
*** Not fully implemented

JM based implementations: JSVM, JMVM, KTA
libavcodec based implementations: ffmpeg and others (ffmpeg based)

H.264 encoder/decoder selection

- Short summary
  - t264 does not decode properly and is not mature enough
  - x264 does not decode
  - libavcodec decodes and can encode using x264's library
  - JM and IPP encode and decode
- Possible selection
  - Parts of x264, JM or IPP for the encoder side
  - Parts of libavcodec, JM or IPP for the decoder side
- Side note: libavcodec based transcoder not feasible as x264 library bridging cannot be reused

Combining different encoder and decoder parts

- Different implementations use
  - Different data structures
  - Different functions
  - Different intermediate steps to combine
- Consequences
  - Bridging slows down transcoding due to extensive copying/converting
  - Combining different implementations is hard and time consuming
  - Using only one may be a better idea

Using code from one implementation

- Reduced number of possibilities to choose from
  - JM (slow)
  - IPP (fast, but relies on IPP libraries – costs!)
- Issues
  - Decoder design differs from encoder design
  - Encoders/decoders are not designed for intermediate data access
  - Copying parsed bits to the output is not enough

Example: Difference between encoder and decoder I

- Example from IPP: encode/decode macroblock type with CABAC
- Encoder side:
  - One function to encode all macroblock types for all slice types
  - Function decides what to encode based on macroblock
- Decoder side:
  - Multiple functions (one for each macroblock and slice type)
  - Caller has to choose appropriate function and set up environment

Example: Difference between encoder and decoder II

- Encoder side (adopted from *umc_h264_bs_tmpl.h*):

```
Status MBTypeInfo_CABAC(void* state,
                        EnumSliceType SliceType,
                        Ipp32s mb_type_cur,
                        MB_Type type_cur,
                        MB_Type type_left,
                        MB_Type type_above);
```

- Decoder side (*umc_h264_segment_decoder.h*):

```
void DecodeMBTypePSlice_CABAC(void);
```

- Different parameters
- Different data structures
- Different environments

What about implementing a transcoder from scratch?

- Advantages
  - Coding effort is limited to the parts we need
  - Bridging is not necessary as built in by design
  - Licensing costs are not an issue
- Disadvantages
  - Existing implementations are not reused
  - Very hard and time consuming
- Hardness estimation
  - Typical scale: several thousand lines of code (H.264)
  - Hundreds of video sequences to test to assure stability
  - Requires a deep understanding of the H.264 standard

Conclusion

- Building a transcoder for watermarking is hard
- Writing it from scratch is too time consuming
- Classic transcoders cannot be used
- Solution 1: Use an existing (rare) bit stream transcoder
  - Evaluation pending (can it do what we want it to do?)
  - Final costs and licensing unclear
- Solution 2: Build a bit stream transcoder
  - Parts of existing encoders and decoders have to be (re)used
  - Connecting these parts is not trivial

Thank you for your attention!

Questions?