Anatomy of a Video Codec

The inner workings of Ogg Theora

Dr. Timothy B. Terriberry

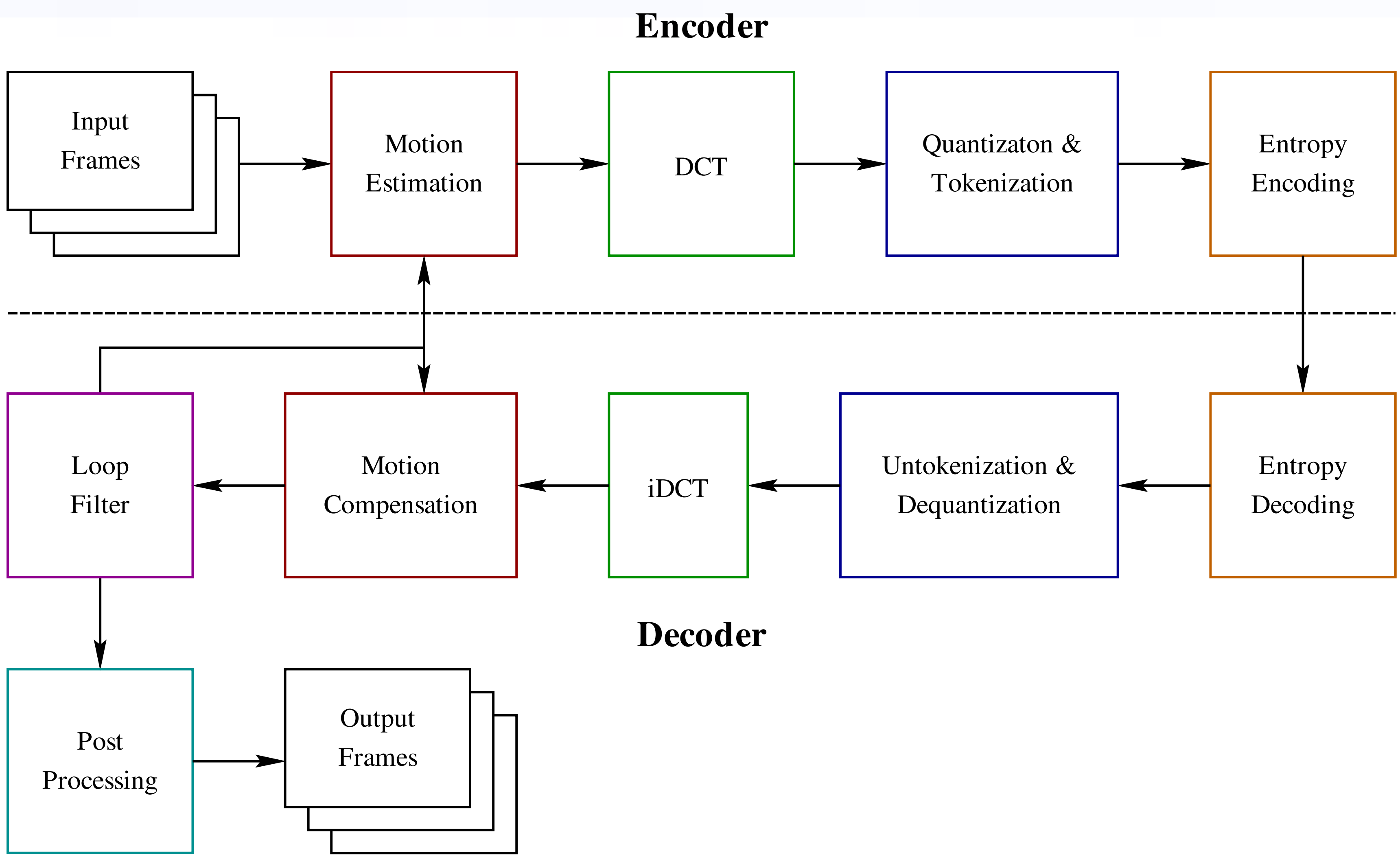
Outline

- Introduction
- Video Structure
- Motion Compensation
- The DCT Transform
- Quantization and Coding
- The Loop Filter
- Conclusion

Introduction

- What is Ogg Theora?
  - MC+2D DCT video codec, like MPEG, H.263, etc.
  - Based on VP3, donated by On2 Technologies
  - Patent unencumbered
    - On2 shipped VP3 for many years
    - Gave everyone a transferable, irrevocable patent license
  - Primary users: live streaming & web video
    - Wikipedia, Metavid, etc.
    - Cortado (Java), plug-ins (vlc, xine, Quicktime, etc.), mv_embed
    - Native Firefox and Opera support soon

Block Diagram



Outline

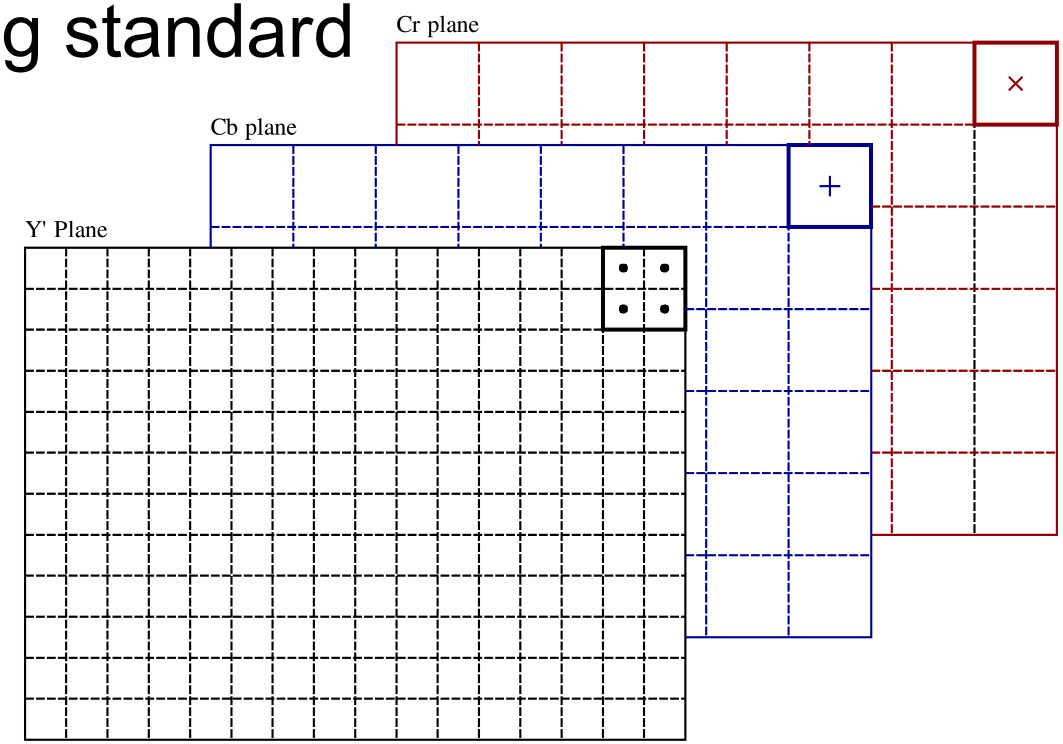
- Introduction
- **Video Structure**
- Motion Compensation
- The DCT Transform
- Quantization and Coding
- The Loop Filter
- Conclusion

Color Space

- $Y'C_bC_r$: Luma, Chroma blue, Chroma red
  - Luma corresponds to grayscale
  - Nonlinear (not gamma corrected)
    - Intensity levels near zero closer together than near 255
    - This is the way human perception works
    - Important for compression
  - Headroom:
    - Normal range of values is (16,16,16) to (219,240,240)
  - Conversion: Multiple standards
    - See Theora specification for details

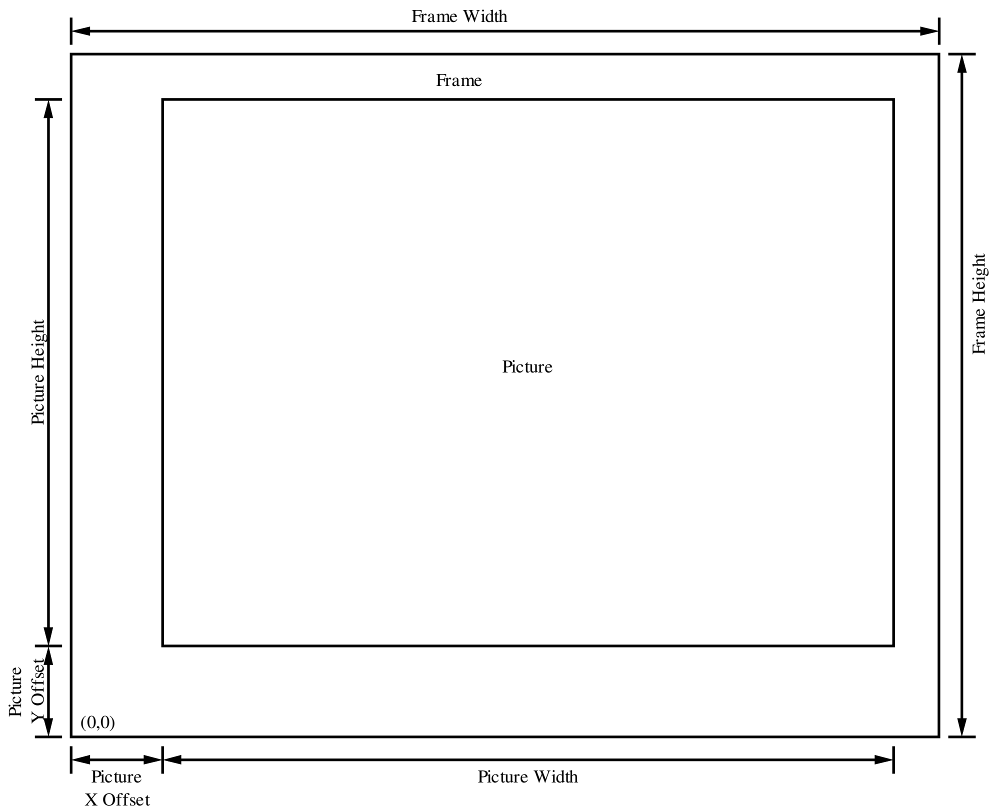
Pixel Format

- Most video is 4:2:0
  - Subsampled by a factor of two in each direction
  - Name comes from signal bandwidth ratios in the original analog standard

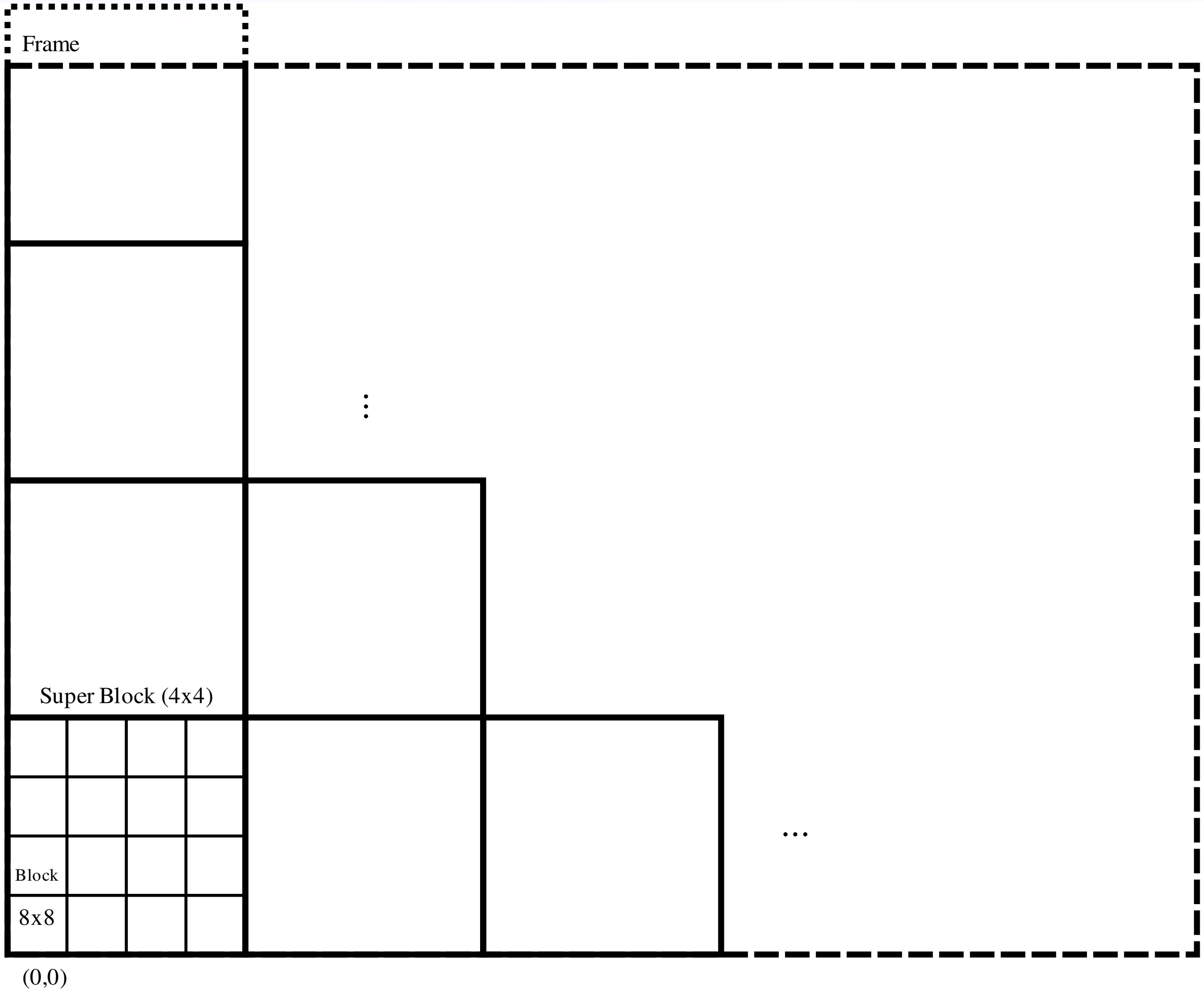


Picture Size

- Frame size must be a multiple of 16
- A smaller “picture region” is actually displayed

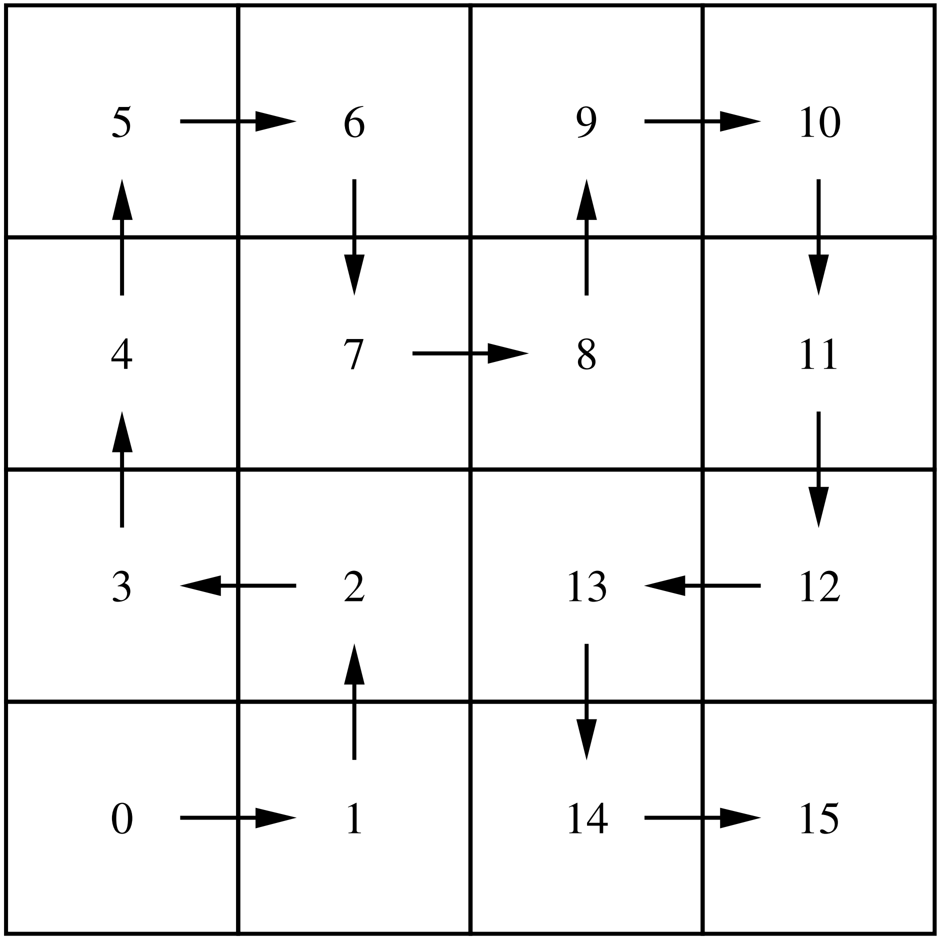


Blocks and Superblocks



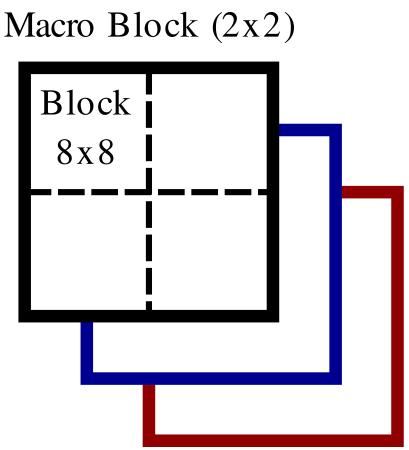
Coded Order

- Within a superblock, blocks are coded along a "Hilbert curve"
- This is a fractal space filling curve
  - Fills a 2D area
  - Each block is adjacent to the next block
- Adjacent blocks are highly correlated



Macro Blocks

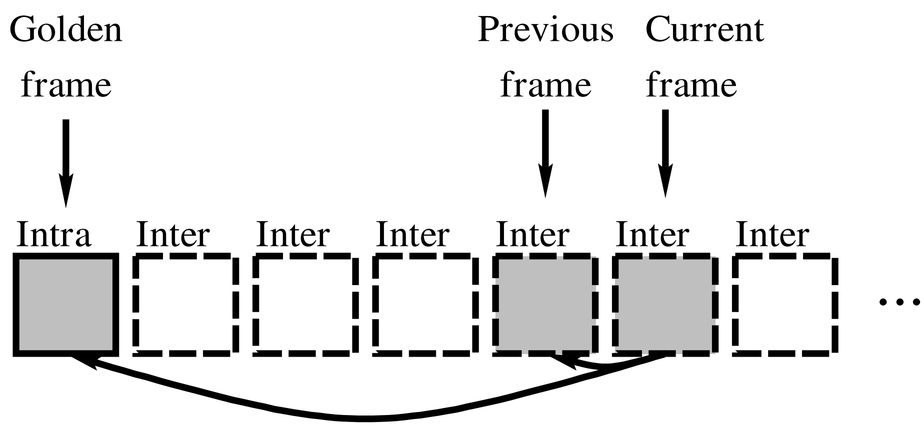
- A superblock is contained within a single plane
- Macro blocks cut across all three planes



- 2x2 group of blocks in the luma plane + corresponding blocks in the chroma planes

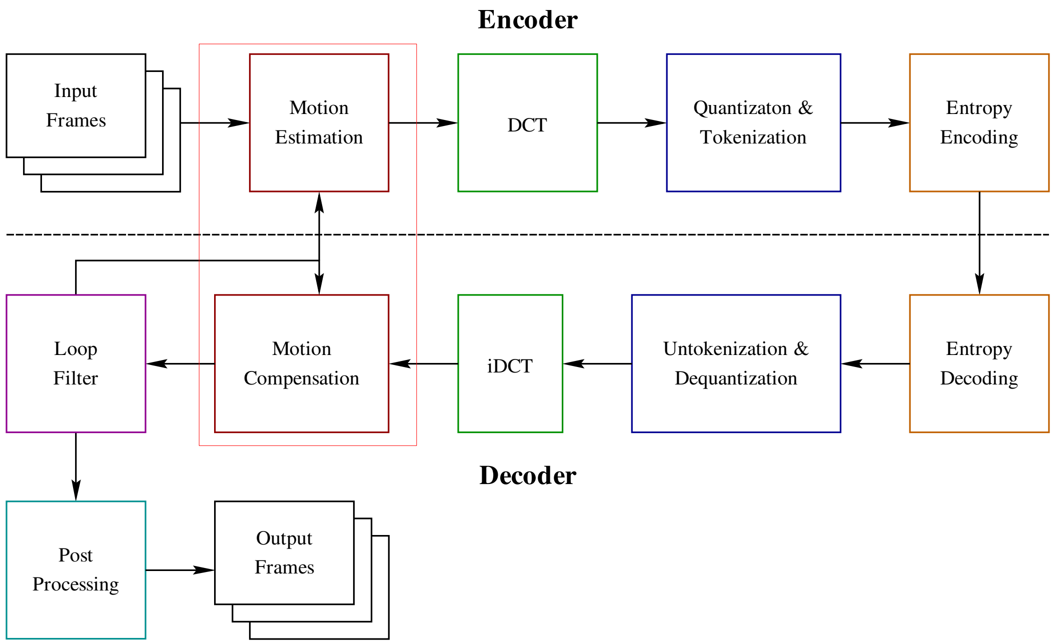
Frame Types

- INTRA frames do not use motion compensation
  - Can be decoded without reference to other frames
- INTER frames do use motion compensation
  - Reference data in the previous frame and the most recent intra frame (the “golden frame”)



Outline

- Introduction
- Video Structure
- **Motion Compensation**
- The DCT Transform
- Quantization and Coding
- The Loop Filter
- Conclusion



Motion Compensation

- Video changes slowly over time
- By subtracting out the previous frame, we remove much of the information
- A motion vector is stored with each macro block to point to the piece to copy

Input

⊖

Reference frame

=

Residual

To code or not to code?

- Not coding a block at all uses very few bits
  - The majority of compression in static scenes comes from skipping blocks entirely
- Frame data is copied directly from the previous frame, and no residual is sent
- If we can identify these early on, we can skip motion search and save processing time, too
  - Current encoder uses simple change thresholding
- How do we signal which blocks are coded?
  - RLE+VLC

Coded Block Flags

- Coded blocks are highly spatially correlated
  - Try to mark entire superblocks at a time
  - Inside a superblock, follow Hilbert curve
- Three-phase process
  - Partition superblocks into “partially coded” and “the rest”
  - Partition “the rest” of the superblocks into “fully coded” and “not coded”
  - Partition the blocks in partially coded superblocks into “coded” and “not coded”

Coded Block Flags

- Represent each partition as a bit string, and encode with RLE+VLC

Superblock Flags

VLC Code	*Run Lengths*	*Compression Ratio*
0	1	100%
10x	2...3	100-150%
110x	4...5	80-100%
1110xx	6...9	67-100%
11110xxx	10...17	47-80%
111110xxxx	18...33	30-56%
111111xxxxxxxxxxxx	34...4129	0.4%-52%

Block Flags

VLC Code	*Run Lengths*	*Compression Ratio*
0x	1...2	100-200%
10x	3...4	75-100%
110x	5...6	67-80%
1110xx	7...10	60-86%
11110xx	11...14	50-64%
11111xxxx	15...30	30-60%

- Code just the first bit value, and then the run lengths: each run of bits must alternate values
- For blocks, we *know* the longest run is 30

Motion Search

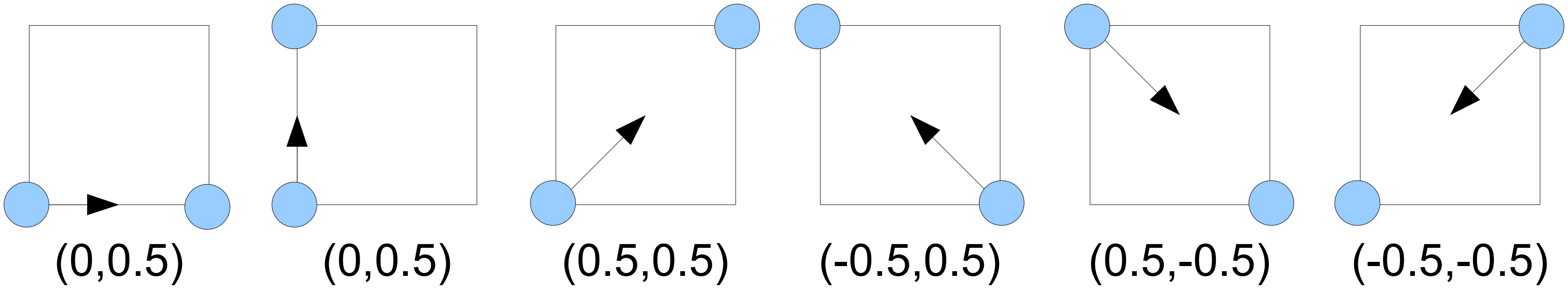
- Want to identify the "best" motion vector
  - Trade-off match quality against cost to code
  - Rate-distortion optimization: cost = $D + \lambda R$
  - λ is the number of bits you're willing to spend for a unit decrease in distortion
  - Current encoder uses just *D* in many places
    - We are fixing this
- How to measure *D*?
  - Sum of Absolute Differences: $\sum |x_i - y_i|$
  - Typically luma plane only (chroma ignored)

Motion Search

- 2 reference frames to check per macro block, plus 4MV
- MV range: (-15.5,-15.5)...(15.5,15.5)
- Find best full-pel vector, then refine to half-pel
- Full search
  - Very slow: 492032 pixel references per macro block
- Logarithmic search: 16384 pixel references
  - Look at (±8,±8), then (±4,±4) around that, etc.
  - Current encoder uses this, with fallback to full search
- Predictive search: ~1K pixel references on average
  - Predict MV from neighbors in space and time

Half-Pel Refinement

- Most codecs implement half-pel MV's by averaging 2 to 4 pixels
  - Linear interpolation suffers from aliasing near edges
  - Aliasing error is *worst* at the halfway point
- Theora: if you're going to do something bad, at least make it really fast
  - Only averages 2 values, even with a (0.5,0.5) MV



Chroma Subsampling

- Theora does not support MV resolution finer than half-pel
- Chroma planes are usually sub-sampled
  - A half-pel vector from the luma plane is quarter-pel
- Round MV's: ¼, ½, and ¾ all treated as ½
  - If a luma vector averages two values, then so will a chroma vector
- Averaging suppresses noise, and most of the benefit of half-pel comes from this effect
  - Real interpolation quality is secondary

Macro Block Modes

- 8 possible modes
- NOMV: use a MV of (0,0)
- LAST: copy the previous MV
  - LAST2 copies the 2nd to last
  - This is the *only* advantage Theora takes of MV correlation
- 4MV: Code a separate MV for each luma block

Macro Block Mode	*Reference Frame*
INTRA	None
INTER_NOMV	Previous
INTER_MV	Previous
INTER_MV_LAST	Previous
INTER_MV_LAST2	Previous
INTER_MV_4MV	Previous
INTER_GOLDEN_NOMV	Golden
INTER_GOLDEN_MV	Golden

Mode Decision

- How do we decide which mode to use?
  - Current code checks *D* for “cheaper” modes, then tries the more expensive ones (e.g., 4MV) if they fail
- R-D optimization is better (in development)
  - What are *R* and *D*?
  - The cost to code the mode *and* the residual
  - Could transform, quantize, encode for each choice
    - Too expensive, and even then computing exact *R* is hard
  - Instead, estimate them using the SAD after MC
    - Giant table lookup trained on lots of video

Coding Macro Block Modes

- Fixed code, dynamic alphabet
- Encoder chooses which mode corresponds to each code word
  - 6 standard lists, or explicitly send the list
  - Encode with a highly skewed VLC code

Mode Code
0
10
110
1110
11110
111110
1111110
1111111

- Fallback: encode each mode with 3 bits

Motion Vector Coding

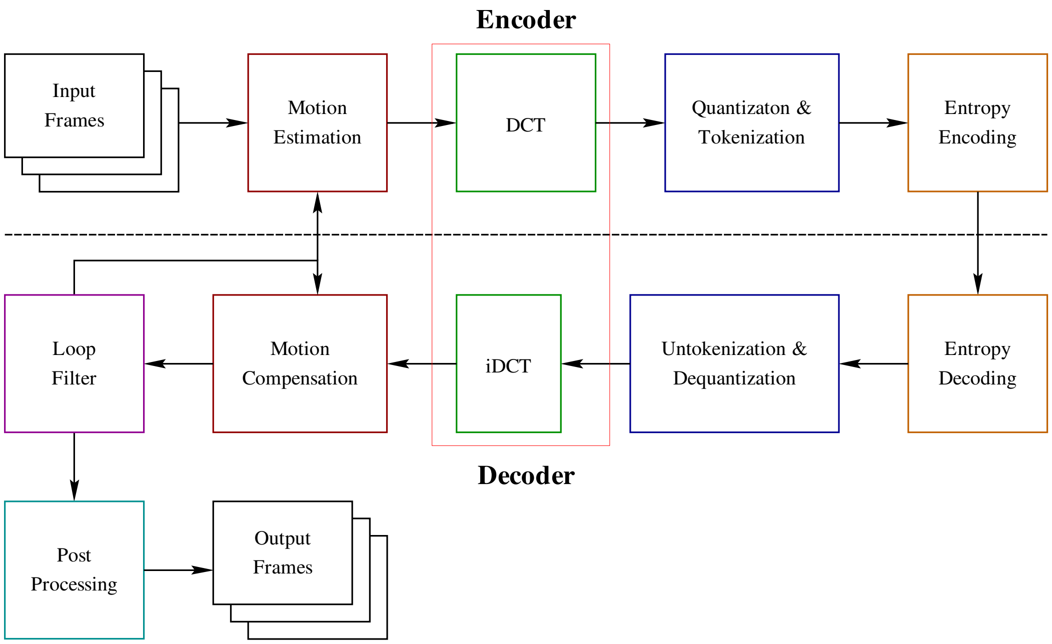
- Each macro block codes between 0 and 4 MV's (depending on mode and coded luma blocks)
- Coded with a fixed VLC code

MV Range	*Number of Bits*
±0...0.5	3
±1...1.5	4
±2...3.5	6
±4...7.5	7
±8...15.5	8

- Fallback: encode each component with 6 bits

Outline

- Introduction
- Video Structure
- Motion Compensation
- **The DCT Transform**
- Quantization and Coding
- The Loop Filter
- Conclusion



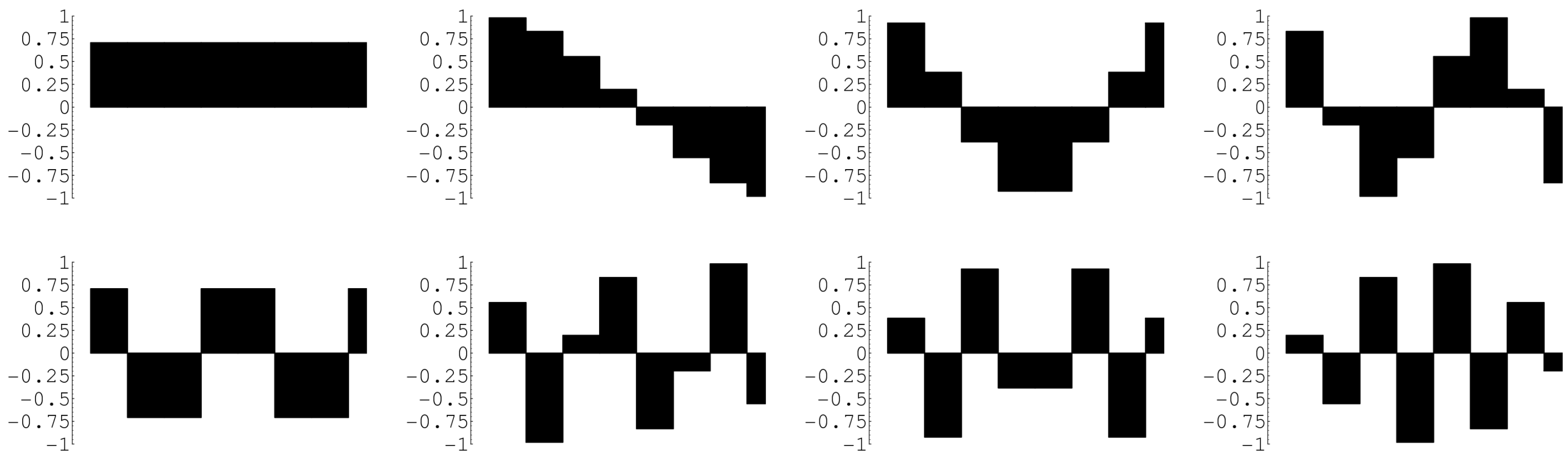
The DCT Transform

- MC has removed temporal correlation
- DCT removes spatial correlation from the residual
- Approx. of ideal Karhunen-Loève Transform
  - Compute the eigenvectors of the covariance matrix
  - Project data onto the eigenvectors (PCA)
  - But: need enough data to estimate covariance
  - But: need to send the eigenvectors
- DCT is close to K-L for natural images

The DCT Transform

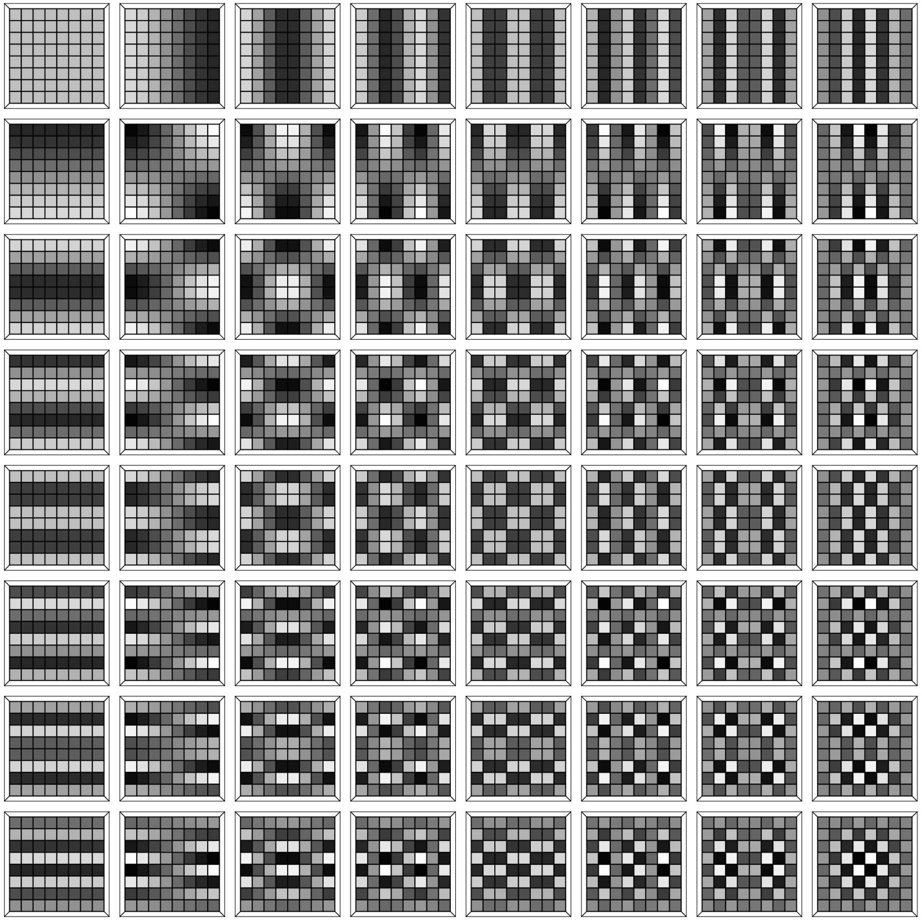
- Applied to each 8x8 block
- In 1-D essentially a matrix multiply: **y = G·x**
  - **G** is orthogonal: acts like an 8-dimensional rotation
  - Basis functions:

DC AC...



The DCT Transform

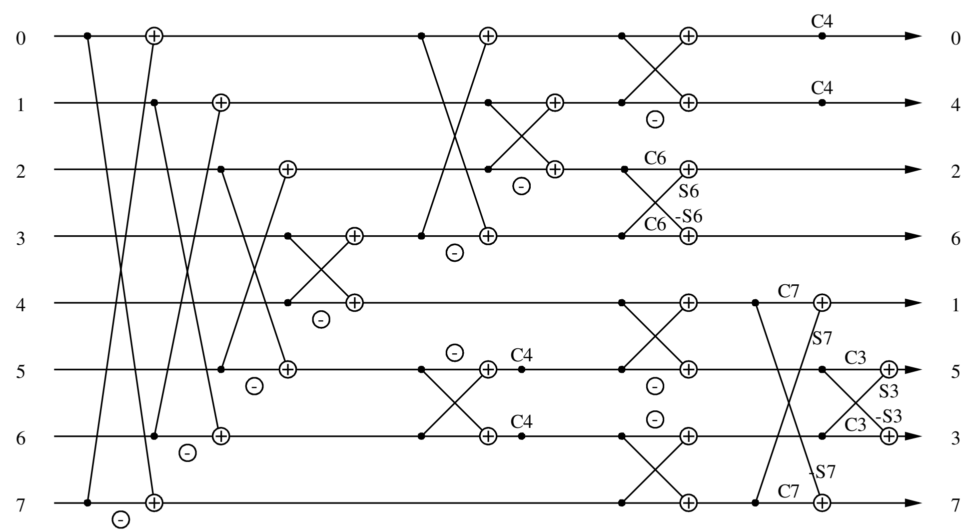
- In 2D, first transform rows, then columns
  - $Y = G \cdot X \cdot G^T$
- Basis functions:



- Two 8x8 matrix multiplies is 1024 mults, 896 adds
  - 16 mults/pixel

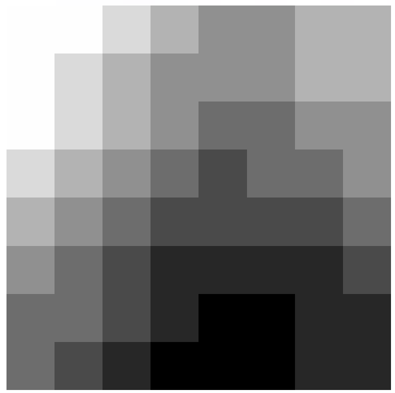
Fast DCT

- The DCT is closely related to the Fourier Transform, so there is also a fast decomposition
- 1-D: 16 mults, 26 adds



- 2-D: 256 mults, 416 adds (4 mults/pixel)

DCT Example



Input Data

156	144	125	109	102	106	114	121
151	138	120	104	97	100	109	116
141	129	110	94	87	91	99	106
128	116	97	82	75	78	86	93
114	102	84	68	61	64	73	80
102	89	71	55	48	51	60	67
92	80	61	45	38	42	50	57
86	74	56	40	33	36	45	52

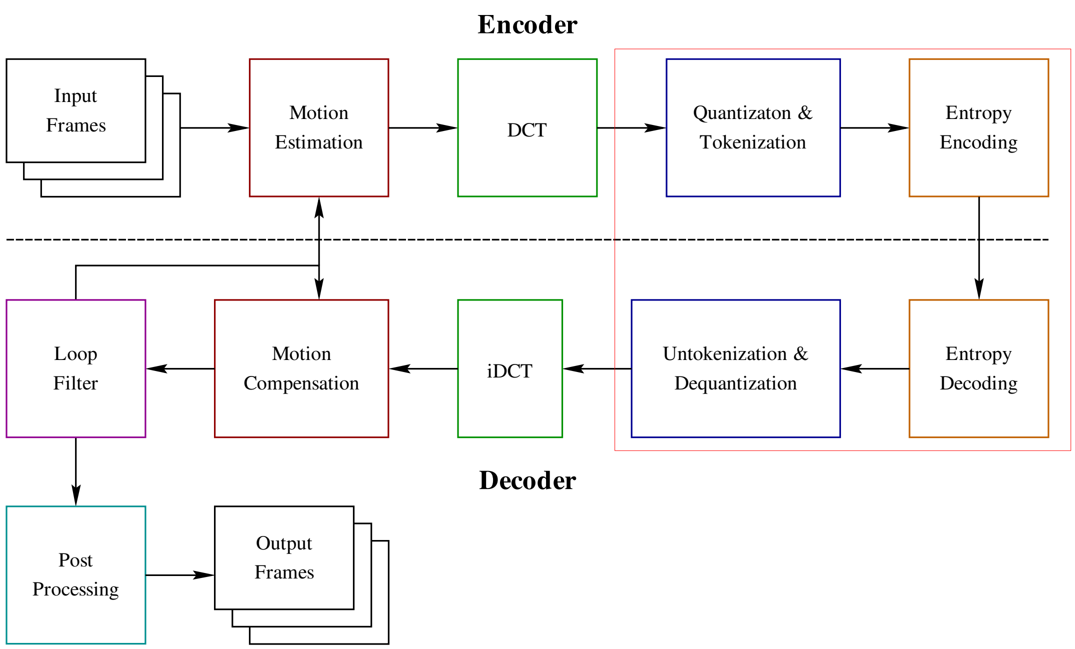
Transformed Data

700	100	100	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

Shamelessly stolen from the MIT 6.837 lecture notes:
http://groups.csail.mit.edu/graphics/classes/6.837/F01/Lecture03/Slide30.html

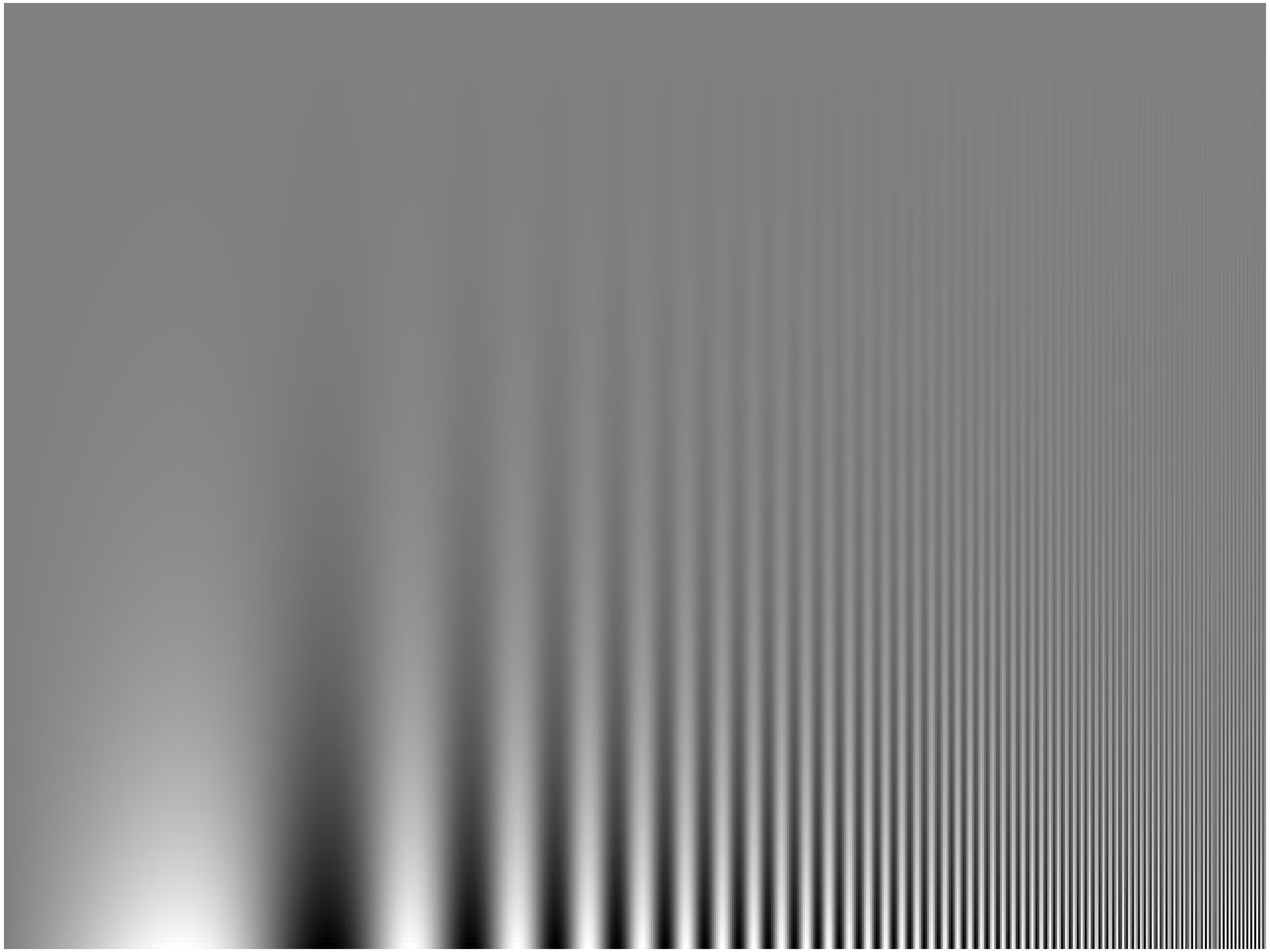
Outline

- Introduction
- Video Structure
- Motion Compensation
- The DCT Transform
- **Quantization and Coding**
- The Loop Filter
- Conclusion



The Contrast Sensitivity Function

- Contrast perception varies by spatial frequency



Quantization Matrices

- Only lossy step in the entire process
- Divide each coefficient by a number chosen to match the CSF
  - Example matrix:

Quantization Matrix

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	58	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

- But that's at the visibility threshold
  - Above the threshold distribution more even
- Most codecs vary quantization by scaling a single base matrix
- Theora allows interpolation between matrices

DC Prediction

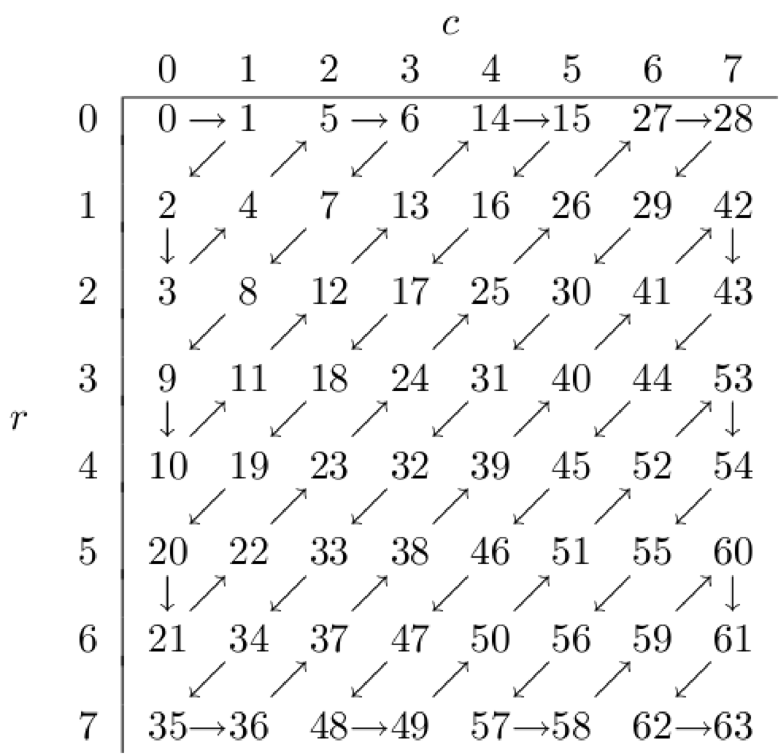
- DC coefficients look like a 1/8th resolution copy of the original image: still lots of correlation
- A simple filter is used to predict each coefficient from its neighbors
  - Preceding neighbors in *raster* order used (not coded)
  - Only those neighbors predicted from the same frame
  - Filter coefficients vary by available neighbors
  - As a last resort, just use the last value with the same prediction type
- Subtract off prediction on encode, add in decode

Per-block quantization

- Up to 3 quantizers can be specified per frame
  - Can be used to sharpen edges,
  - Reduce detail in smooth regions,
  - Foreground/background regions, etc.
- Pick one to use for the AC coefs. of each block
  - DC is predicted *after* quantization (unfortunate)
- Chosen quantizer signaled with same RLE+VLC scheme as coded blocks

Zig-Zag Scanning

- Coefficients in a block scanned in zig-zag order
  - Roughly low frequency → high
  - Creates long runs of zeros



Tokenization

- Coefficient values are translated into one of 32 tokens + a fixed number of “extra bits”
  - Fairly unique to Theora
- Tokens are entropy coded, extra bits are written verbatim to the stream

EOB Tokens

- Signals the "End Of Block"
  - All the remaining coefficients are zero
  - Follows Hilbert curve (spatial correlation)
- Multiple blocks combined into EOB runs

Token Value	*Extra Bits*	*EOB Run Length*
0	0	1
1	0	2
2	0	3
3	2	4...7
4	3	8...15
5	4	16...31
6	12	1...4095

Zero Run Tokens

- A run of zeros that doesn't end the block

Token Value	*Extra Bits*	*Number of Coefficients*	*Description*
7	3	1...8	Short zero run
8	6	1...64	Zero run
23	1	2	One zero followed by ±1
24	1	3	Two zeros followed by ±1
25	1	4	Three zeros followed by ±1
26	1	5	Four zeros followed by ±1
27	1	6	Five zeros followed by ±1
28	3	7...10	6...9 zeros followed by ±1
29	4	11...18	10...17 zeros followed by ±1
30	2	2	One zero followed by ±2...3
31	3	3...4	2...3 zeros followed by ±2...3

Coefficient Tokens

- Encode the value of a single non-zero coefficient

Token Value	*Extra Bits*	*Coefficient Value*
9	0	+1
10	0	-1
11	0	+2
12	0	-2
13	1	±3
14	1	±4
15	1	±5
16	1	±6
17	2	±7...8
18	3	±9...12
19	4	±13...20
20	5	±21...36
21	6	±37...68
22	10	±69...580

- Note: There's a maximum value
  - Implies a minimum quantizer

Token Coding

- *All* of the tokens for a single coefficient are coded before moving to the next (in zig-zag order)
  - Requires all blocks to be transformed+quantized before entropy coding
  - Poor cache locality when decoding
- Tokens which span multiple coefficients are coded when the first one would be
  - This block is skipped during token decode until the next coefficient is needed

Huffman Coding

- Shannon source coding theorem:
  - The best code for *independent*, *identically distributed* variables with probability distribution $\{p_i\}$ uses $-\log_2(p_i)$ bits per value
- Huffman gave an algorithm for translating probabilities p_i into a prefix-free code
  - Optimal when $-\log_2(p_i)$ is restricted to be an integer
- Main idea: code frequently occurring symbols with fewer bits, and only use more on rare ones

Huffman Tables

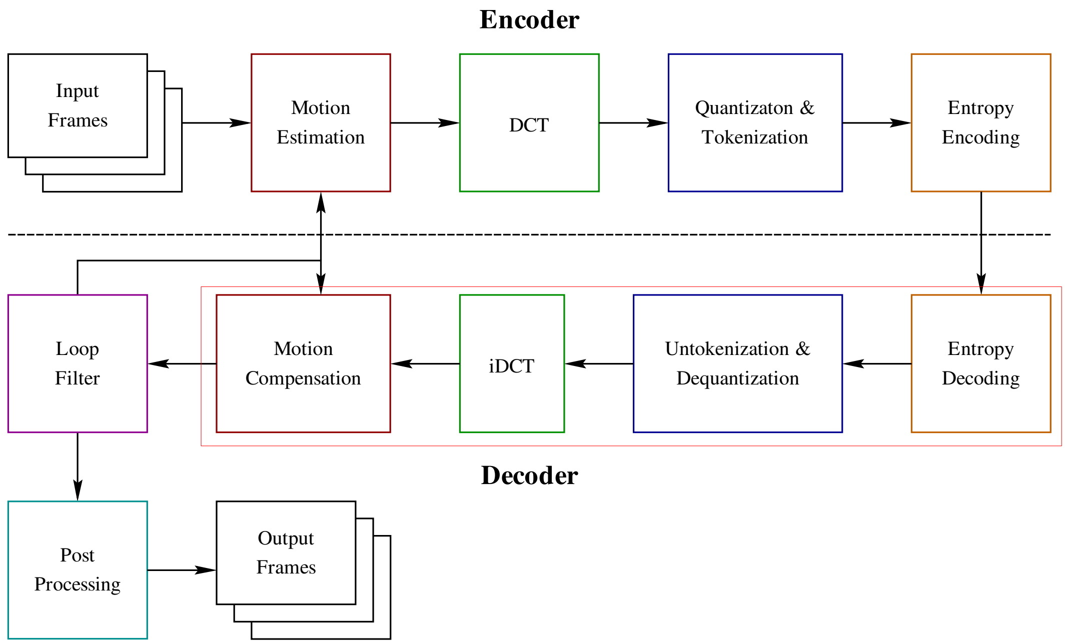
- VLC codes for tokens are stored in the header
  - 80 possible codes to choose from
  - 32 token possible token values in each code
- Divided into 5 groups of 16 by zig-zag index

Zig-Zag Index	*Huffman Group*
0	0
1...5	1
6...14	2
15...27	3
28...63	4

- Pick one table in group 0 for the DC coefficients
- Pick *one* table index (0...15) to use for *all* four AC groups

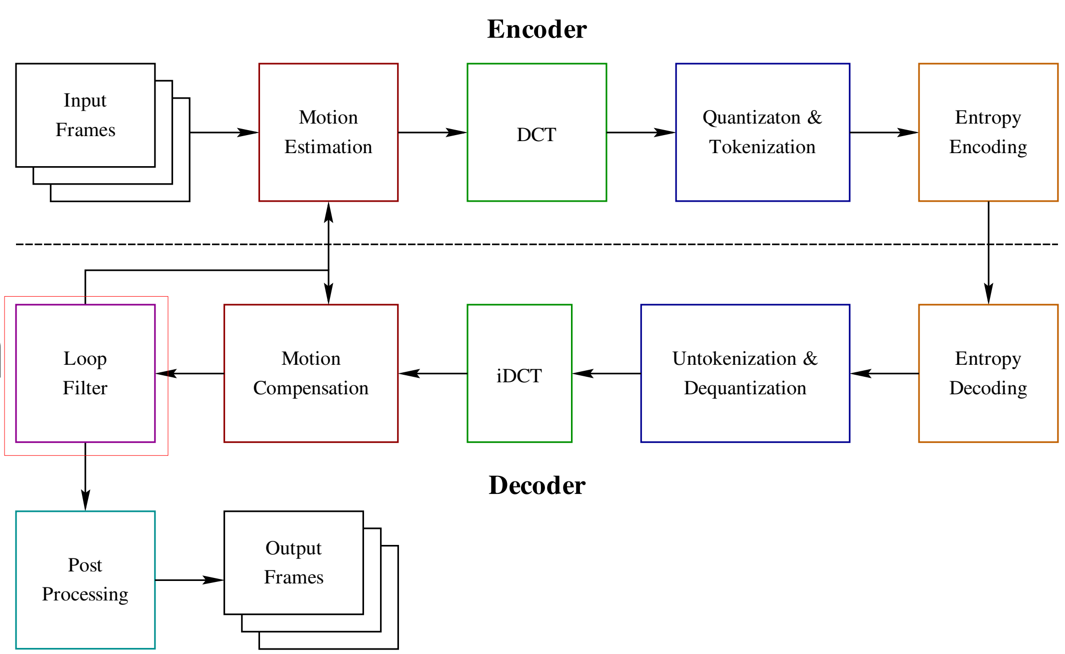
Encoding → Decoding

- We have all the tools: purely mechanical



Outline

- Introduction
- Video Structure
- Motion Compensation
- The DCT Transform
- Quantization and Coding
- **The Loop Filter**
- Conclusion

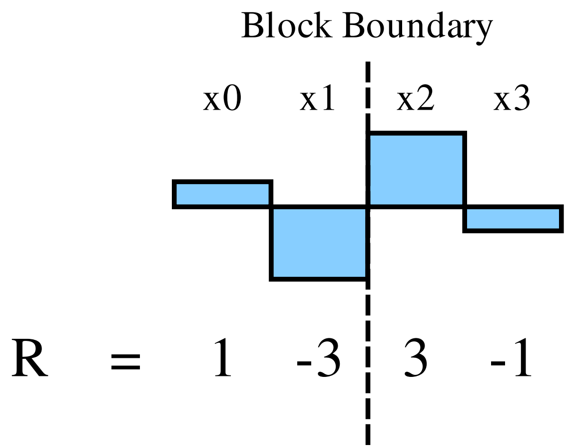


The Loop Filter

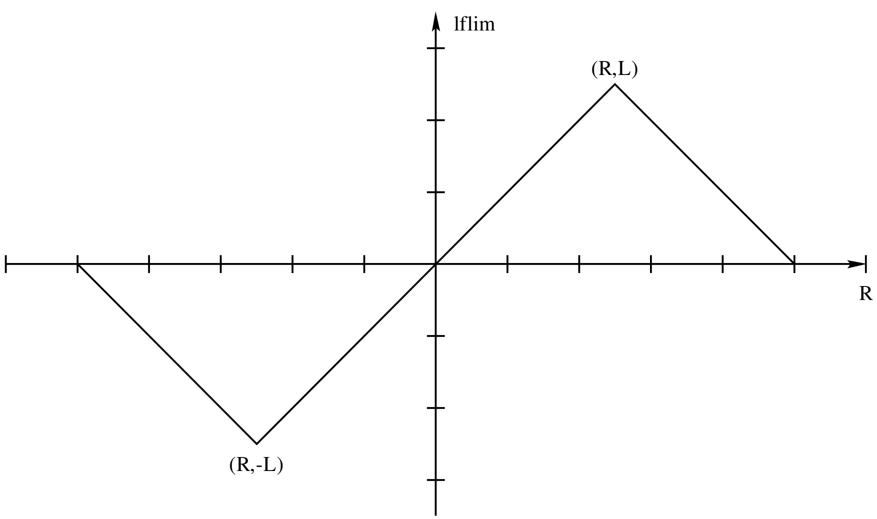
- Block-based codecs have blocking artifacts
  - MPEG4 Part 2 and earlier used post-processing
- But if post-processing improves the image, feeding it back into the prediction is better
  - But processing is no longer optional
- H.264 also added a loop filter (years after Theora)

The Loop Filter

- Run a small filter across the block edge



- Adjust the inner values base on its strength

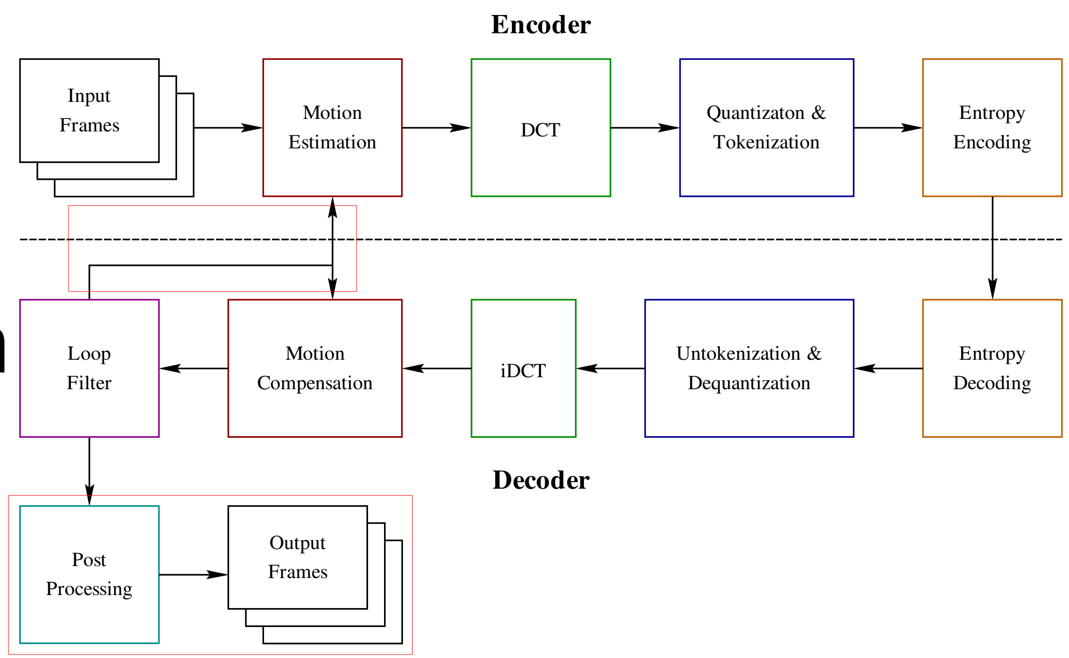


x1 = x1 + lflim(R,L)

x2 = x2 - lflim(R,L)

Outline

- Introduction
- Video Structure
- Motion Compensation
- The DCT Transform
- Quantization and Coding
- The Loop Filter
- **Conclusion**



The End

- After the loop filter, the frame is complete
- In both the encoder and decoder, it feeds back in and becomes a new reference frame
- In the decoder, it is ready for display
  - There’s more post-processing available
    - Stronger de-blocking, de-ringing
  - Much more CPU-intensive, and so optional
    - We even provide an API to enable it now

Future Directions

- Arithmetic/Range encoding
  - Allows a fractional number of bits: 6-12% savings for free
- Overlapped transforms
  - Similar to the MDCT used in Vorbis: no blocking artifacts
  - Better energy compaction than wavelets with less computation
- Blocking-free transforms require blocking-free motion compensation

Questions?