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

Encoder:
- Input Frames
- Motion Estimation
- DCT
- Quantization & Tokenization
- Entropy Encoding

Decoder:
- Loop Filter
- Motion Compensation
- iDCT
- Untokenization & Dequantization
- Entropy Decoding

Post Processing
- Output Frames

The Xiph.Org Foundation
Outline

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

- $Y' C_b C_r$: 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”)

![Diagram showing frame types]
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

\[
\text{Input} \ominus \text{Reference frame} = \text{Residual}
\]
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

<table>
<thead>
<tr>
<th>VLC Code</th>
<th>Run Lengths</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>10x</td>
<td>2...3</td>
<td>100-150%</td>
</tr>
<tr>
<td>110x</td>
<td>4...5</td>
<td>80-100%</td>
</tr>
<tr>
<td>1110xx</td>
<td>6...9</td>
<td>67-100%</td>
</tr>
<tr>
<td>11110xxx</td>
<td>10...17</td>
<td>47-80%</td>
</tr>
<tr>
<td>111110xxxx</td>
<td>18...33</td>
<td>30-56%</td>
</tr>
<tr>
<td>111111xxxxxxxxxx</td>
<td>34...4129</td>
<td>0.4%-52%</td>
</tr>
</tbody>
</table>

Block Flags

<table>
<thead>
<tr>
<th>VLC Code</th>
<th>Run Lengths</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x</td>
<td>1...2</td>
<td>100-200%</td>
</tr>
<tr>
<td>10x</td>
<td>3...4</td>
<td>75-100%</td>
</tr>
<tr>
<td>110x</td>
<td>5...6</td>
<td>67-80%</td>
</tr>
<tr>
<td>1110xx</td>
<td>7...10</td>
<td>60-86%</td>
</tr>
<tr>
<td>11110xx</td>
<td>11...14</td>
<td>50-64%</td>
</tr>
<tr>
<td>11111xxxx</td>
<td>15...30</td>
<td>30-60%</td>
</tr>
</tbody>
</table>

- 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 = $D + \lambda R$
  – $\lambda$ 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 \textit{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

\begin{itemize}
  \item (0,0.5)
  \item (0,0.5)
  \item (0.5,0.5)
  \item (-0.5,0.5)
  \item (0.5,-0.5)
  \item (-0.5,-0.5)
\end{itemize}
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

<table>
<thead>
<tr>
<th>Macro Block Mode</th>
<th>Reference Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRA</td>
<td>None</td>
</tr>
<tr>
<td>INTER_NOMV</td>
<td>Previous</td>
</tr>
<tr>
<td>INTER_MV</td>
<td>Previous</td>
</tr>
<tr>
<td>INTER_MV_LAST</td>
<td>Previous</td>
</tr>
<tr>
<td>INTER_MV_LAST2</td>
<td>Previous</td>
</tr>
<tr>
<td>INTER_MV_4MV</td>
<td>Previous</td>
</tr>
<tr>
<td>INTER_GOLDEN_NOMV</td>
<td>Golden</td>
</tr>
<tr>
<td>INTER_GOLDEN_MV</td>
<td>Golden</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Mode Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>110</td>
</tr>
<tr>
<td>1110</td>
</tr>
<tr>
<td>11110</td>
</tr>
<tr>
<td>111110</td>
</tr>
<tr>
<td>1111110</td>
</tr>
<tr>
<td>1111111</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>MV Range</th>
<th>Number of Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>±0...0.5</td>
<td>3</td>
</tr>
<tr>
<td>±1...1.5</td>
<td>4</td>
</tr>
<tr>
<td>±2...3.5</td>
<td>6</td>
</tr>
<tr>
<td>±4...7.5</td>
<td>7</td>
</tr>
<tr>
<td>±8...15.5</td>
<td>8</td>
</tr>
</tbody>
</table>

- 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: \( y = G \cdot x \)
  - \( G \) is orthogonal: acts like an 8-dimensional rotation
  - Basis functions:
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

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:
- 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 $1/8^{th}$ 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

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0→1</td>
<td>5→6</td>
<td>14→15</td>
<td>27→28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2→4</td>
<td>7→13</td>
<td>16→26</td>
<td>29→42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3→8</td>
<td>12→17</td>
<td>25→30</td>
<td>41→43</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9→11</td>
<td>18→24</td>
<td>31→40</td>
<td>44→53</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10→19</td>
<td>23→32</td>
<td>39→45</td>
<td>52→54</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20→22</td>
<td>33→38</td>
<td>46→51</td>
<td>55→60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>21→34</td>
<td>37→47</td>
<td>50→56</td>
<td>59→61</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>35→36</td>
<td>48→49</td>
<td>57→58</td>
<td>62→63</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
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

<table>
<thead>
<tr>
<th>Token Value</th>
<th>Extra Bits</th>
<th>EOB Run Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4...7</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>8...15</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>16...31</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>1...4095</td>
</tr>
</tbody>
</table>
Zero Run Tokens

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

<table>
<thead>
<tr>
<th>Token Value</th>
<th>Extra Bits</th>
<th>Number of Coefficients</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
<td>1...8</td>
<td>Short zero run</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>1...64</td>
<td>Zero run</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>2</td>
<td>One zero followed by ±1</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>3</td>
<td>Two zeros followed by ±1</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>4</td>
<td>Three zeros followed by ±1</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>5</td>
<td>Four zeros followed by ±1</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>6</td>
<td>Five zeros followed by ±1</td>
</tr>
<tr>
<td>28</td>
<td>3</td>
<td>7...10</td>
<td>6...9 zeros followed by ±1</td>
</tr>
<tr>
<td>29</td>
<td>4</td>
<td>11...18</td>
<td>10...17 zeros followed by ±1</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>2</td>
<td>One zero followed by ±2...3</td>
</tr>
<tr>
<td>31</td>
<td>3</td>
<td>3...4</td>
<td>2...3 zeros followed by ±2...3</td>
</tr>
</tbody>
</table>
Coefficient Tokens

- Encode the value of a single non-zero coefficient

<table>
<thead>
<tr>
<th>Token Value</th>
<th>Extra Bits</th>
<th>Coefficient Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>+2</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>±3</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>±4</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>±5</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>±6</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>±7...8</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>±9...12</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>±13...20</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>±21...36</td>
</tr>
<tr>
<td>21</td>
<td>6</td>
<td>±37...68</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>±69...580</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Zig-Zag Huffman Index</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1...5</td>
<td>1</td>
</tr>
<tr>
<td>6...14</td>
<td>2</td>
</tr>
<tr>
<td>15...27</td>
<td>3</td>
</tr>
<tr>
<td>28...63</td>
<td>4</td>
</tr>
</tbody>
</table>

- 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

\[ R = 1 \quad -3 \quad 3 \quad -1 \]

- Adjust the inner values base on its strength

\[ x_1 = x_1 + l\text{lim}(R,L) \]
\[ x_2 = x_2 - l\text{lim}(R,L) \]
Outline

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