11-1-2011

An Analysis of VP8, a new video codec for the web

Sean Cassidy

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
An Analysis of VP8, a New Video Codec for the Web

by

Sean A. Cassidy

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science
in Computer Engineering

Supervised by

Professor Dr. Andres Kwasinski
Department of Computer Engineering
Rochester Institute of Technology
Rochester, NY
November, 2011

Approved by:

Dr. Andres Kwasinski
R.I.T. Dept. of Computer Engineering

Dr. Marcin Łukowiak
R.I.T. Dept. of Computer Engineering

Dr. Muhammad Shaaban
R.I.T. Dept. of Computer Engineering
Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

An Analysis of VP8, a New Video Codec for the Web

I, Sean A. Cassidy, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

______________________________________________________________
Sean A. Cassidy

______________________________________________________________
Date
Acknowledgements

I would like to thank my thesis advisors, Dr. Andres Kwasinski, Dr. Marcin Łukowiak, and Dr. Muhammad Shaaban for their suggestions, advice, and support. This thesis could not have been completed if it were not for their input.

I would like to thank my fiancée, Amanda Fishedick, for her advice, support, and encouragement. If it were not for her, there would be no box plots in this thesis, and would be worse overall.

Finally, I would like to thank my friends and family. Without the encouragement and advice of my family, I would not be at R.I.T. completing my thesis now. No amount of acknowledgement can express my gratitude at their ceaseless work and care to get me to where I am today. I would like to specifically thank my friends: Robert Ghilduta, Jonathon Szymaniak, and Frank Jenner. They are some of the smartest people I know, and offered numerous suggestions and support. This thesis would not have been completed in its current form without them.
Contents

Abstract                                                  1
Glossary                                                  3

1 Introduction                                               6
  1.1 Scope of this Thesis ........................................ 6
  1.2 Related Work                                          8
  1.3 Overview of Video                                      9
    1.3.1 Color ..................................................... 10
    1.3.2 Transforms ............................................... 11
    1.3.3 Entropy Coding .......................................... 13
  1.4 Objective Quality Measurements ................................ 15

2 VP8 and H.264 Compared                                    18
  2.1 VP8 Internals                                          18
    2.1.1 Quantization ............................................. 19
    2.1.2 Transforms ................................................ 21
    2.1.3 Boolean Entropy Encoder .................................. 24
    2.1.4 Intraframe Prediction ..................................... 26
    2.1.5 Interframe prediction ..................................... 30
2.1.6 Scan Order ............................................. 32
2.1.7 Loop Filter ........................................... 33
2.2 Features Unique to VP8 or H.264 ......................... 45
   2.2.1 B-Frames, Alternate Reference Frames, and Golden Frames .......................... 45
   2.2.2 Adaptive Behavior .................................. 48
   2.2.3 Streaming Support .................................. 51
2.3 Beyond H.264 and VP8: HEVC and the future .......... 55
2.4 Hardware Implementation of VP8 ......................... 56

3 Test Methodology .......................................... 58
   3.1 Encoders, Configuration, and Data Acquisition .......... 59
      3.1.1 Encoders ........................................ 59
      3.1.2 Data Acquisition ................................ 59
   3.2 Uncompressed Source Videos .......................... 61
   3.3 Statistical Significance ................................ 64
      3.3.1 Choosing the $\alpha$-level ..................... 65
   3.4 Initial Testing and Capabilities ....................... 66
      3.4.1 Feature Set Testing Video Results ............ 66
      3.4.2 Rate-Distortion Curves ......................... 67
   3.5 Use Case Scenarios .................................... 69
      3.5.1 Video Conferencing ........................... 69
      3.5.2 High Definition Television and Film Broadcast ... 71
   3.6 Intra-coding Tests .................................... 73
   3.7 Inter-coding Tests .................................... 75
   3.8 Loop Filter Efficiency Tests ......................... 78
List of Figures

2.1 A Flowchart of the Encoding Process .................................. 20
2.2 Subblock Mapping to the Token Probability Table ..................... 22
2.3 The area around a macroblock that will be intra-compressed .......... 27
2.4 An intra-predicted macroblock using vertical prediction ............... 27
2.5 An intra-predicted macroblock using horizontal prediction .......... 28
2.6 Luma Intra-prediction overview ....................................... 29
2.7 Scan Orders for Transformed Coefficients ............................. 33
2.8 The pixels for simple loop filtering on a vertical block border ........ 37
2.9 The pixels for simple loop filtering on a horizontal block border .... 37
2.10 Bidirectional Frames in H.264 ......................................... 46
2.11 Alternate Reference and Golden Frames in VP8 ...................... 47
2.12 Segments and Slices .................................................... 49

4.1 PSNR Summary of Videos at CIF, 150 Kbps, H.264 Baseline and VP8 Good Deadline ......................................................... 81
4.2 SSIM Summary of Videos at CIF, 150 Kbps, H.264 Baseline and VP8 Good Deadline .......................................................... 82
4.3 PSNR Summary of Videos at CIF, 150 Kbps, H.264 High Profile and VP8 Best Deadline ......................................................... 83
4.4 SSIM Summary of Videos at CIF, 150 Kbps, H.264 High Profile and VP8
   Best Deadline ............................................................................. 83
4.5 Summary of Four H.264 Profiles vs. VP8 on 4CIF Videos at 575 Kbps ..... 88
4.6 Summary of Four H.264 Profiles vs. VP8 on 720p Videos at 1000 Kbps ..... 89
4.7 Parkrun at 720p, encoded at 1000 Kbps ........................................... 90
4.8 Foreman CIF Video RD Curve for H.264 and VP8 ............................. 90
4.9 Soccer 4CIF Video RD Curve for H.264 and VP8 ............................ 91
4.10 Stockholm 720p Video RD Curve for H.264 and VP8 ....................... 91
4.11 Summary of Akiyo at CIF Resolution, 25 Kbps ............................... 93
4.12 Frames of Akiyo at CIF Resolution, 25 Kbps .................................. 93
4.13 Summary of Akiyo at CIF Resolution, 50 Kbps ............................... 94
4.14 Frames of Akiyo at CIF Resolution, 50 Kbps .................................. 95
4.15 Summary of Students at CIF Resolution, 25 Kbps, KF Interval 300 ... 97
4.16 Frames of Students at CIF Resolution, 25 Kbps, KF Interval 300 ...... 97
4.17 Summary of Students at CIF Resolution, 25 Kbps, KF Interval 30 .... 99
4.18 Frames of Students at CIF Resolution, 25 Kbps, KF Interval 30 ...... 99
4.19 SSIM Summary of the Three Videos at 720p, nominally 3000 kbps ..... 101
4.20 SSIM Frames of City Outdoors, 720p, nominally 3000 kbps ............... 101
4.21 SSIM Summary of the Three Videos at 1080p, nominally 5000 kbps .... 103
4.22 SSIM Frames of Outdoor Pedestrians, 1080p ................................. 105
4.23 SSIM Summary of In to Tree and Rush Hour ................................. 106
4.24 SSIM Frames of In to Tree, 720p ................................................. 106
4.25 Bitrate vs. SSIM for H.264 Baseline and VP8 Good Deadline .......... 110
4.26 Bitrate vs. SSIM for H.264 High Profile and VP8 Best Deadline ....... 111
4.27 PSNR Summary of Inter-coding Tests ........................................ 113
4.28 SSIM Summary of Inter-coding Tests ........................................ 114
4.29 Percent Difference for PSNR and SSIM among four resolutions for VP8 and H.264
# List of Tables

2.1 The Five Types of Motion Vectors ........................................... 31

3.1 Versions of the Software Used .............................................. 60
3.2 Source Videos at 352 × 288 Used and their Attributes ................. 61
3.3 Source Videos at 704 × 576 Used and their Attributes ................. 62
3.4 Source Videos at 1280 × 720 Used and their Attributes ............... 62
3.5 Source Videos at 1920 × 1080 Used and their Attributes .............. 63
3.6 Initial Testing Encoder Parameters ....................................... 67
3.7 Feature Set Testing Encoder Parameters .................................. 67
3.8 Video Conferencing Encoder Parameters, from Slowest Encoding Speed to Fastest .................................................. 70
3.9 Source Videos for 720p Composite Films .................................. 71
3.10 Source Videos for 1080p Composite Films ................................ 72
3.11 Intra-coding Test Encoder Profiles and Parameters .................... 74
3.12 Configurations for Inter-coding Tests ..................................... 76
3.13 Configurations for Two Pass Loop Filter Tests .......................... 79

4.1 p-values for RD Curve for Foreman ....................................... 86
4.2 p-values for RD Curve for Soccer .......................................... 86
4.3 p-values for RD Curve for Stockholm ..................................... 87
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>p-values for Frames for the Fastest and Slowest Profiles for Akiyo at 25 Kbps</td>
<td>95</td>
</tr>
<tr>
<td>4.5</td>
<td>p-values for Frames for the Fastest and Slowest Profiles for Akiyo at 50 Kbps</td>
<td>96</td>
</tr>
<tr>
<td>4.6</td>
<td>p-values for Frames for the Fastest and Slowest Profiles for Students at 25 Kbps, 300 KF Interval</td>
<td>98</td>
</tr>
<tr>
<td>4.7</td>
<td>p-values for Frames for the Fastest and Slowest Profiles for Students at 25 Kbps, 30 KF Interval</td>
<td>100</td>
</tr>
<tr>
<td>4.8</td>
<td>p-values for the Average PSNR and SSIM Values based on $\sigma$</td>
<td>109</td>
</tr>
<tr>
<td>4.9</td>
<td>Average Percent Difference for the four Resolutions and Overall</td>
<td>115</td>
</tr>
<tr>
<td>B.1</td>
<td>p-values for SSIM Frames of City Outdoors, 720p</td>
<td>135</td>
</tr>
<tr>
<td>B.2</td>
<td>p-values for SSIM Frames of Outdoor Pedestrians, 1080p</td>
<td>135</td>
</tr>
<tr>
<td>B.3</td>
<td>Raw PSNR Values for Ice for Inter-Coding Test</td>
<td>136</td>
</tr>
<tr>
<td>B.4</td>
<td>Raw SSIM Values for Ice for Inter-Coding Test</td>
<td>136</td>
</tr>
<tr>
<td>B.5</td>
<td>Raw PSNR Values for Parkrun for Inter-Coding Test</td>
<td>137</td>
</tr>
<tr>
<td>B.6</td>
<td>Raw SSIM Values for Parkrun for Inter-Coding Test</td>
<td>137</td>
</tr>
</tbody>
</table>
Abstract

Video is an increasingly ubiquitous part of our lives. Fast and efficient video codecs are necessary to satisfy the increasing demand for video on the web and mobile devices. However, open standards and patent grants are paramount to the adoption of video codecs across different platforms and browsers. Google On2 released VP8 in May 2010 to compete with H.264, the current standard of video codecs, complete with source code, specification and a perpetual patent grant.

As the amount of video being created every day is growing rapidly, the decision of which codec to encode this video with is paramount; if a low quality codec or a restrictively licensed codec is used, the video recorded might be of little to no use. We sought to study VP8 and its quality versus its resource consumption compared to H.264 – the most popular current video codec – so that reader may make an informed decision for themselves or for their organizations about whether to use H.264 or VP8, or something else entirely.

We examined VP8 in detail, compared its theoretical complexity to H.264 and measured the efficiency of its current implementation. VP8 shares many facets of its design with H.264 and other Discrete Cosine Transform (DCT) based video codecs. However, VP8 is both simpler and less feature rich than H.264, which may allow for rapid hardware and software implementations. As it was designed for the Internet and newer mobile devices, it contains fewer legacy features, such as interlacing, than H.264 supports.

To perform quality measurements, the open source VP8 implementation libvpx was used. This is the reference implementation. For H.264, the open source H.264 encoder x264 was used. This encoder has very high performance, and is often rated at the top of its field in efficiency. The JM reference encoder was used to establish a baseline quality for H.264.
Our findings indicate that VP8 performs very well at low bitrates, at resolutions at and below CIF. VP8 may be able to successfully displace H.264 Baseline in the mobile streaming video domain. It offers higher quality at a lower bitrate for low resolution images due to its high performing entropy coder and non-contiguous macroblock segmentation.

At higher resolutions, VP8 still outperforms H.264 Baseline, but H.264 High profile leads. At HD resolution (720p and above), H.264 is significantly better than VP8 due to its superior motion estimation and adaptive coding. There is little significant difference between the intra-coding performance between H.264 and VP8. VP8’s in-loop deblocking filter outperforms H.264’s version. H.264’s inter-coding, with full support for B frames and weighting outperforms VP8’s alternate reference scheme, although this may improve in the future. On average, VP8’s feature set is less complex than H.264’s equivalents, which, along with its open source implementation, may spur development in the future.

These findings indicate that VP8 has strong fundamentals when compared with H.264, but that it lacks optimization and maturity. It will likely improve as engineers optimize VP8’s reference implementation, or when a competing implementation is developed. We recommend several areas that the VP8 developers should focus on in the future.
Glossary

4CIF  Common Interchange Format, a format for broadcast video. Used in this thesis to refer to $704 \times 576$ resolution.

ASIC  Application Specific Integrated Circuit.

ASO  Arbitrary Slice Ordering, a method in the H.264 standard to order slices in any order.

AVC  Advanced Video Coding, part of the H.264 standard

BER  Bit error rate, the percentage of bits in a stream that are in error.

CABAC  Context-Adaptive Binary Arithmetic Coding, part of the H.264 standard to losslessly compress residue.

CAVLC  Context-Adaptive Variable Length Coding, part of the H.264 standard to losslessly compress residue. Simpler than CABAC, but lower performing.

CIF  Common Interchange Format, a format for broadcast video. Used in this thesis to refer to $352 \times 288$ resolution.

DC  Direct current, although used in this thesis and other works to refer to 0 Hz frequency data.

DCT  Discrete Cosine Transform, a common method in image and video coding to get 2-D frequency domain representation of data.
FMO  Flexible Macroblock Ordering, a method in the H.264 standard to allow macroblocks that are not next to each other to share the same slice.

HEVC  High Efficiency Video Coding, the successor to H.264, slated for release in 2013.

HVS  Human Visual System

Macroblock  A square group of pixels. In this thesis, this exclusively refers to an $8 \times 8$ macroblock of 16 pixels.

MP4  A file type that contains video and audio streams, also known as a container. Part of the MPEG-4 standard.

MPEGLA  The MPEG Licensing Authority

PSNR  Peak Signal-to-Noise Ratio, a measure of quality, measured in decibels (dB).

NAL  Network Abstraction Layer, part of the H.264 standard. This standardizes the network representation of H.264 and adds network-aware features.

SSIM  Structural Similarity, a measure of quality, measured from 0 to 1.0, where 1.0 is perfect lossless quality.

SVC  Scalable Video Coding, part of the H.264 standard, wherein the network video stream degrades gracefully with packet loss and packet error.

TCO  Total Cost of Ownership, a measure of how expensive a technology is, including all costs such as training, licensing fees, societal cost, etc.
**VHDL**  VHSIC Hardware Description Language.

**WHT**  Walsh-Hadamard Transform, another transform, similar to the DCT, to represent data in the frequency domain

**Y4M**  YUV for MPEG, a format for representing video in a raw format, with no loss of quality or compression.

**YCrCb**  Stands for the luminance, chroma red, and chroma blue color space. Widely used in image and video processing.

**YUV**  See **YCrCb**
Chapter 1

Introduction

Video coding is a complex topic, and seeks to appease the diverse group of video consumers: from people watching short clips on smartphones to real-time video conferencing, to professional film studios and television broadcasters.

This chapter introduces what this thesis attempts to address, related work, an overview of video coding including its various components, such as color and transformations.

1.1 Scope of this Thesis

This thesis covers VP8 and its implementation at the time of this writing. When considering which codec to use, many factors must be weighed, including the ease of use, performance, and total cost of ownership (TCO), which includes possible legal fees resulting from licensing or patent litigation. One particular codec may perform better than another, but if the TCO is outside of the available budget, it cannot be considered. Further, one must consider how easy it is for one’s organization to switch codecs; an online video service cannot easily switch millions of videos without millions of hours of effort and a guaranteed loss of visual quality, however, a video camcorder manufacturer can
switch the encoder they use in their products with relative ease.

VP8 was developed by On2 Technologies, later bought by Google Inc., to provide high quality video for the web and mobile devices. It has a perpetual patent grant – something which gives the implementor a no-charge, royalty-free, irrevocable patent license to use or sell VP8, and the official reference implementation is open source. This thesis will not cover estimated patent license cost, TCO, or legal implications of using VP8 over H.264 or vice versa. It will cover an objective analysis of the claims made by Google about VP8, specifically with respect to its computational complexity and performance.

To analyze VP8 effectively, tests were designed to measure how each vital feature of VP8 compares to a similar feature in H.264, as well as how each performs overall and in several use case scenarios. A recommendation is made for improving VP8’s implementation, and recommends which codec to use for which purposes if legal and TCO implications are not a factor. Not every use case or feature is possible to test, and there is room for further analysis and recommendations.

For the purpose of this thesis, H.264 indicates H.264/AVC. H.264/SVC was not analyzed in detail, although some discussion of some of its features and how they compare to VP8 are mentioned. Work on comparing VP8 to H.264/SVC has been studied by Seeling et al.[1], and is not duplicated here.

The general method for comparing VP8 and H.264 is to analyze the components that define the codecs, design methodology to compare these individual features, and run tests on a varied set of source videos to gain an accurate understanding of how each component compares to its counterpart in the other codec. By analyzing the codec by parts, rather than by the whole, emphasis can be placed on where optimization should take place. This also gives a more detailed view on the codecs, and can ignore implementation-specific bugs and shortcomings.
1.2 Related Work

A comparison between VP8 and H.264/SVC was performed by Seeling et. al, wherein rate-distortion performance was compared with H.264 when there is significant traffic variability on long video sequences[1]. Their conclusion was that VP8’s performance was slightly less than H.264 SVC’s performance in this domain. They also state the oft-repeated claim that On2, before it was sold to Google, claimed that VP8 offered twice the quality of H.264 at half its bandwidth.

Another study compared VP8 to H.264 and HEVC – the next evolution of H.264, which is planning to be standardized in 2013 – and found that, when 18 subjects viewed various videos, that VP8 was competitive with H.264, specifically, the open source encoder x264[2]. They used the 1080p source videos CrowdRun, DucksTakeOff, InToTree, Old-TownCross, and ParkJoy from the VQEG HDTV SVT Dataset, and reduced their resolution to $854 \times 480$ and reduced their frame rate to 25 FPS. VP8 was not able to encode at very low bitrates – instead it would use a higher bitrate than requested; these bitrates were 250 kbps and 500 kbps. Overall, VP8 fared worse than x264, JM – the reference implementation of H.264/AVC, and HM – the test implementation of HEVC, but the difference could be due to the standard error within the video.

A technical summary of VP8 was published by the principal authors of VP8[3]. It details the features that VP8 has in transformations, quantizations, its various reference frame types, filtering, and its boolean entropy coding. The summary also includes experimental results, such as VP8’s impressive decoding speed, where VP8 achieves an approximately 40% faster decoding speed when compared to H.264. When using PSNR, the RD curves presented in the summary for the videos HallMonitor, Highway, Pamphlet, and Deadline are nearly identical between VP8 and H.264. VP8 leads in the videos HallMonitor and Pamphlet, while it lags slightly in Deadline.
There is some literature related to VP7, the previous iteration of VP8[4]. Their findings, when VP7 was compared to H.264/SVC – the scalable video coding subset of H.264, was that VP7 outperformed H.264/SVC on ADSL2+, HSDPA, WiMAX, and Ethernet LAN networks for PSNR. However, H.264/SVC outperformed VP7 for the MOS, where five people viewed the videos. Their sample size of five for the subjective video analysis is small, and renders statistical analysis of their findings tenuous.

Surveys of H.264 have used PSNR [5, 6, 7] in comparing H.264 versus other codecs, such as XviD, DivX, and Windows Media Codec 9. Some have used SSIM to measure H.264 performance[8, 9, 7], while others have used lesser known metrics such as the JND (Just Noticeable Difference)[5] or VQM (Video Quality Metric) [9, 7]. While it is shown that PSNR and SSIM match the MOS values linearly[10], many studies also test MOS with a small group of people[4, 11, 7]. For this thesis, PSNR and SSIM were chosen, as they are popular, well tested metrics. MOS was also used for the subjective analysis portion.

H.264 was chosen as the sole codec to measure VP8 against, due to it outperforming all other modern video codecs[5, 7]. This alleviates the need to test a large number of codecs. x264 was chosen as the primary implementation of H.264 due to it being open source, freely available for many platforms, and for its performance. It is faster than the standard JM reference implementation by a factor of 50, and outputs a bitstream within 5% of the JM implementation for the same PSNR[12].

1.3 Overview of Video

Video is an significant part of modern life, from movies and television shows, to news, sports, home videos and events. The Internet has been expanding in bandwidth usage at an enormous rate, and Internet video tops the growth demographics. In late 2010, Cisco published a traffic analysis of the types of data that dominate the available bandwidth
on the Internet. Internet traffic increased 45.6% from 2008 to 2009, totaling approximately 176,200 petabytes (176.2 exabytes) for 2009. [13] The largest category of growing Internet traffic is video to both personal computers, and TVs. Cisco predicts that Internet video will occupy over 23,000 petabytes (23 exabytes) of the total Internet traffic in 2014, more than half of the total bandwidth.

Dominating the video realm is the de-facto standard H.264, which is used on an array of huge Internet video websites, from YouTube to Facebook. It offers a high peak signal-to-noise ratio (PSNR) at a low bit-rate, and has several modes related to error mitigation and error concealment. [14] A great deal of research has been created to improve H.264’s error control and response mechanisms [15].

However, H.264, despite being an open standard, is protected by a myriad of software and algorithmic patents in dozens of countries [16]. Some of these patents do not expire until 2028, and this stifles the complete adoption of H.264, especially in the flourishing open source community. In response, Google On2 released a competing video codec, named VP8, in May of 2010. It is designed to be used as a general purpose codec for streaming videos over the Internet, in a container known as WebM.

VP8 is designed to be simpler than H.264, and as such contains fewer color modes, less available transformation sizes, a simpler loop filter; all while delivering a similar bit-rate and visual quality to the end user. H.264’s complexity results in an efficient bitstream that is already well supported by both hardware and software platforms. It will need to be seen if VP8’s deviations will result in a quality video codec.

1.3.1 Color

Video formats that need to deal with color need to represent this information in a compact manner.

Chroma subsampling is a method by which to reduce the amount of memory required
– at the expense of increased algorithmic complexity required to subsample – to represent an image. Generally, the technique is using a smaller resolution mapped to by copying chroma pixels onto their respective luma counterparts, which are at a larger resolution. The format is generally as shown in Eq. 1.3.1. 

rk-Usage Changes Push Internet Traffic to the Edge

\[ J : a : b \]  \hspace{1cm} (1.3.1)

This uses the notion of a reference block of \( J \) pixels wide, and 2 pixels high. \( J \) is almost always 4. The value \( a \) represents the number of chroma blue, or Cb (which is a digitized chrominance sample of U) values over \( J \) and the value \( b \) represents the number of chroma red, or Cr (which is a digitized chrominance sample of V) values over \( J \). [17]

However, the very common value 4:2:0, which is indeed what VP8 uses as its only chroma subsampling technique, does not follow the convention shown in Eq. 1.3.1. For purposes of this document, 4:2:0 will signify a chrominance pixel, comprising of both Cb and Cr, that is at half the resolution of the luma pixel. Four luma pixels will map to a single chrominance value.

Representing one of several millions of colors for every single pixel, known as the 4:4:4 image format, requires a large amount of space, but represents the color information without loss of data.

1.3.2 Transforms

There are two main types of Fourier-class transforms used in modern video codecs. One is the discrete cosine transform (DCT) and the other is the Walsh-Hadamard transform (WHT).
The Discrete Cosine Transform

The DCT transforms a signal from the time domain to the frequency domain. As the human visual system (HVS) is less sensitive to high frequency information, the high frequency components are discarded as they are less important than the DC and lower frequency components. The formal definition of the $4 \times 4$ pixel DCT is shown in Eq. 1.3.2[18]. VP8 does not utilize this form directly; instead, it uses a one dimensional version with multiple passes. In addition, VP8 is dissimilar to many other video codecs in that it uses only a $4 \times 4$ transform. Other codecs also use a $8 \times 8$ transform or a $16 \times 16$ transform.

$$F(u, v) = \frac{1}{4} C_u C_v \sum_{i=0}^{3} \sum_{j=0}^{3} f(i, j) \cos \left[ \frac{\pi}{4} \left( i + \frac{1}{2} \right) u \right] \cos \left[ \frac{\pi}{4} \left( j + \frac{1}{2} \right) v \right]$$ (1.3.2)

The inverse DCT is shown in Eq. 1.3.3:

$$f(i, j) = C_u C_v \sum_{i=0}^{3} \sum_{j=0}^{3} F(u, v) \cos \left[ \frac{\pi}{4} \left( i + \frac{1}{2} \right) u \right] \cos \left[ \frac{\pi}{4} \left( j + \frac{1}{2} \right) v \right]$$ (1.3.3)

The values $C_u$ and $C_v$ are scaling coefficients defined in Eq. 1.3.4:

$$C_x = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}$$ (1.3.4)

The Walsh-Hadamard Transform

The WHT, also known as the Hadamard Transform, is recursively defined as seen in Eq. 1.3.5, where $H_m$ is a $2^m \times 2^m$ matrix, where $H_0 = 1$. 
\[ H_m = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix} \]  

(1.3.5)

To transform a matrix via the WHT, the matrix, \( A \), is multiplied by \( H_m \). \( A \) must also be a \( 2^m \times 2^m \) matrix. Eq. 1.3.6 shows the resulting transformed matrix, \( A_{WHT} \). This matrix transformation, however, has \( O(n^2) \) complexity. There exists a fast WHT algorithm that uses \( n \log n \) operations[19].

\[ A_{WHT} = H_mA \]  

(1.3.6)

1.3.3 Entropy Coding

The final step of video coding is entropy coding, which compresses data without removing entropy, or removing information. Messages, in the case of video, data, have a particular amount of unpredictability, known as entropy. The lower the entropy (unpredictability) the easier the information is to represent using less data, known as compression. The higher the entropy in the message, the lower the compression efficiency of the message is. In video coding, the lower the entropy of the residue – what is left over after transformation – the higher the efficiency of the entropy coder.

Huffman coding and arithmetic coding are two ways in which lossless coding – another term for entropy coding – is performed. Huffman coding is conceptually simpler, but arithmetic coding is more efficient.

Huffman Coding

Huffman coding is used throughout VP8 to compress information about the data, such as which prediction mode is being used, or which coefficient token is currently being used
for the sub-block in question.

Huffman coding ranks symbols in descending order or their likeliness to appear in the data stream. Higher probability symbols are given mappings to numbers with less bits, to achieve high compression performance.

**Arithmetic Coding**

Arithmetic coding is a type of entropy coding that can achieve high levels of compression, given certain information about the input domain. Arithmetic coding is similar to Huffman coding, which is another type of entropy coding, but instead of replacing each input symbol into its own code, arithmetic coding replaces the entire message into a number $n$, where $n \in [0.0, 1.0]$.

Arithmetic coding operates by defining a probability model, similar to Huffman coding, and ranks input symbols according to how likely they are. The optimal value is $-\log_2(P)$ bits for symbols of probability $P$.

After determining the probability of each input symbol, the intervals for each symbol can be defined. If there are three symbols, $A$ with probability 70%, $B$ with probability 20%, and $C$ with probability 10%, then the three intervals are, respectively, $[0, 0.7)$, $[0.7, 0.9)$, and $[0.9, 1.0)$.

To encode a message to send, calculate the number $n$ by subdividing the intervals to narrow the possibilities for messages. If the message is $CAB$, then the first interval is $[0.9, 1.0)$. Dividing this into three pieces of the probabilities specified in the previous paragraph results in three new intervals, $A \in [0.9, 0.97)$, $B \in [0.97, 0.99)$, and $C \in [0.99, 1.0)$. The next symbol is $A$, so that interval is chosen.

Dividing that interval into three pieces for the final symbol results in the three intervals, $A \in [0.9, 0.949)$, $B \in [0.949, 0.963)$, and $C \in [0.963, 0.97)$. The final symbol in the message is $B$, so that interval is chosen. Any number within that interval is acceptable,
but it is common practice to use the lower bound. Our message can then be represented as \( n = 0.949 \). Neither floating point numbers, nor infinite precision is used to represent 0.949; rather, a fixed precision is used to limit the number of bits necessary to represent this number.

To decode a message received, the intervals need to be subdivided over and over to reveal each symbol of the message, essentially the reverse of the encoding step. The example message has \( n = 0.949 \), which lies first in the interval for \( C \), therefore the first symbol is \( C \). The range for \( C \) would be then subdivided as before, until revealing that the next symbol must be \( A \), and then after subdividing that range, the final symbol, \( B \).

VP8 uses arithmetic coding as its lossless coding, to compress the residue after quantization, transformation, and prediction.

### 1.4 Objective Quality Measurements

There are many quantitative quality metrics for use with two-dimensional digital signals. PSNR is an important metric, and is used often. However, there are important limitations to using PSNR as a metric for measuring relative quality of encoded video. Comparing the results from one raw input video to various encodings from the same encoder produces a strongly correlated result with respect to standard subjective testing. However, across different videos, encoders, and even frame rates, PSNR is not an accurate measurement [10]. Due to this, PSNR will only be utilized to measure relative gain within one codec and within one source video only.

Recently, the structural similarity (SSIM) metric has become popular, and is claimed to better capture the psycho-visual perception of humans, based upon the HVS model [20]. These two popular metrics will be used in the evaluation of VP8 due to their support and the weight of evidence behind them. Other metrics, such as the Video Quality Metric
(VQM) and the HVS Similarity Measure (HSM) [9] were considered for the evaluation, but did not offer substantial benefits when compared to PSNR and SSIM.

The standard metric for analyzing the differences between any two digital data streams is the PSNR. The PSNR for video is defined as in Eq. 1.4.1b[18]. The value $x$ is the maximum value a pixel can be in image $A$ and $B$. For a 16-bit grayscale image, this value is 65535. The Mean Square Error (MSE) is used as shown in Eq. 1.4.1a, for grayscale images. $m$ refers to the number of rows in a matrix $A_{m,n}$, where $n$ is the number of columns.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (A(i,j) - B(i,j))^2$$  \hspace{1cm} (1.4.1a)

$$PSNR_{dB} = 10 \log_{10} \frac{x^2}{MSE}$$  \hspace{1cm} (1.4.1b)

SSIM is defined in terms of mean luminance intensity for an image $A$, Eq 1.4.2a, the signal contrast, using the standard deviation as an estimate, Eq 1.4.2b, to form the SSIM calculation as seen in Eq. 1.4.2c [20]. $C_1$ is a constant defined as $C_1 = (K_1 L)^2$, which is used to avoid situations where the square sum of the mean luminance intensities is near zero. $L$ is the highest value of a pixel, similar to $I_{max}$ for PSNR, and $K_1 \ll 1$. $C_2$ is similar to $C_1$ and is defined as $C_2 = (K_2 L)^2$, where $K_2 \ll 1$.

$$\mu_A = \frac{1}{N} \sum_{i=1}^{N} A_i$$  \hspace{1cm} (1.4.2a)

$$\sigma_A = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (A_i - \mu_A)^2}$$  \hspace{1cm} (1.4.2b)

$$SSIM(A,B) = \frac{(2\mu_A \mu_B + C_1)(2\sigma_{AB} + C_2)}{\mu_A^2 + \mu_B^2 + C_1}(\sigma_A^2 + \sigma_B^2 + C_2)$$  \hspace{1cm} (1.4.2c)
In general, both PSNR and SSIM are sensitive to similar noise and errors in images[21]. Using two similar metrics in this thesis gives a more complete picture of the codecs under study.

This chapter addressed the goals of and work related to this thesis. It examined much of the background material necessary to accurately follow the remainder of this document and to have a full understanding of the implications thereof.

The next chapter will introduce VP8 as a video codec, which will examine it at a high level of detail, and will compare it to H.264 and its feature set.
Chapter 2

VP8 and H.264 Compared

VP8, the video codec in the WebM media container released by Google Inc., in May 2010, is a modern video codec with many advanced features designed to have a high visual quality at a low bitrate. This chapter covers VP8 in detail, explaining its encoding process, quantization, transformation, entropy coding, prediction, and filtering.

A direct comparison with H.264 is made, discussing inter-prediction differences, adaptive behavior – including segmentation and slices, quantization, transformations, and entropy coding, and network streaming support. HEVC, the next video standard being developed by the Joint Collaborative Team on Video Coding (JCT-VC), is introduced and compared with VP8. The ease of implementation of VP8 in hardware is discussed, and the reference hardware implementation is compared with implementations of H.264 in hardware.

2.1 VP8 Internals

The encoding process of VP8 follows five steps in order. First, the macroblock to be encoded gets a predictor. The predictor, as discussed in 2.1.4, is subtracted from the mac-
roblock. Next, the macroblock is transformed, using either the DCT or the WHT, depending on the prediction mode. This is explained in Section 2.1.2. After transformation, the quantization step takes the outputs of the transformation process, and restricts them to a certain output range.

Figure 2.1 shows the encoding process. The output from the Prediction block is the predictor, and is subtracted from the macroblock by value. The Prediction block uses spatial information (pixel values surrounding the sub-block), and temporal information (motion vectors from other frames). The Transform block uses the prediction mode to decide whether or not to use the WHT or the DCT.

The predicted, transformed, and quantized macroblock is then run through the loop filter, discussed in Section 2.1.7, and finally entropy encoded, as discussed in Section 1.3.3.

2.1.1 Quantization

Quantization is a signal processing technique where a continuous or large set of input ranges are mapped to a discrete or smaller set of ranges. VP8, like many other video codecs, uses quantization as a method for discarding superfluous data to enhance the compression efficiency.

To quantize the residue, each coefficient is divided by one of six quantization factors, the selection of which depends upon the plane being encoded. In VP8, a plane is a set of two-dimensional data with metadata describing the type of that data. There are four types of planes in VP8: Y2, the virtual plane from the WHT, Y, the luminance plane, U, the chroma red plane, and V, the chroma blue plane. The quantization step also depends upon the coefficient position, either DC – coefficient 0, or AC – coefficients 1 through 15. These values are specified in one of two ways, via an index into a look-up table, or as an offset to an index.

The baseline quantization factor, $Y_{acr}$, is specified as a 7-bit lookup into the AC quan-
Figure 2.1: A Flowchart of the Encoding Process

tizer lookup table. \( Y_{ac} \) is added to each of the other quantization factors, which are specified as 4-bit positive or negative offsets from the index of \( Y_{ac} \).

Each other factor is specified as a four bit offset from the \( Y_{ac} \) index, and include a sign bit. This means that if \( Y_{ac} = 16 \), then a value of \( Y_1 = 3 \) would be 19, and a value of \( Y_2 = -10 \) would be 6. This allows an index range of \( \pm 15 \) from the index of \( Y_{ac} \). In the VP8
bitstream, the five factors other than $Y_{ac}$ are optional, and only included if a flag is true. If they are omitted, they are set to zero, which indicates that the same quantization factor as $Y_{ac}$ should be used for them.

To dequantize the residue before inverting the transform during decoding, each coefficient is multiplied by one of six dequantization factors.

### 2.1.2 Transforms

The quantized DCT coefficients are coded for each macroblock, and they make up what is known as the residue signal. The residue is added to the prediction blocks (either Intra-prediction as described in Section 2.1.4 or Inter-prediction as described in Section 2.1.5) to form the final reconstructed macroblock.

Unlike H.264, VP8 uses only $4 \times 4$ DCT and WHT for its transformations (see Sections 1.3.2 and 1.3.2 for more information). The DCT is used for the 16 Y sub-blocks of a macroblock, the 4 U sub-blocks, and the 4 V sub-blocks. The WHT is used to construct a $4 \times 4$ array of the average intensities of the 16 Y sub-blocks. This is a stand-in for the 0\textsuperscript{th} DCT coefficients of the Y sub-blocks. This is known as $Y_2$, and is considered a virtual subblock, contrasted to the 24 other, real, sub-blocks. The subblock $Y_2$ is conditional, based upon the prediction mode.

The $4 \times 4$ DCT in H.264 and VP8 can be computed with integer arithmetic only [22]. This enables it to operate quickly and with low complexity, as it avoids floating point operations. Further, only 16-bit arithmetic is required for both the H.264 DCT and the VP8 DCT, allowing it to function well on lower end microprocessors.
The Transformed Coefficients

The coefficients of the 16 sub-blocks of each macroblock are arithmetic coded, and are known as “tokens”. The probability table for decoding this is four-dimensional, and is dependent on the type of plane being decoded, the sub-block being decoded, the local complexity, and the token tree structure.

There are four possible values for the first dimension of the probability table, depending on what type of plane is being decoded, either Y after a Y2 plane, a Y2 plane, a chroma plane (U or V), or a Y plane without a Y2 plane, index, respectively from 0 to 3.

The next dimension depends upon the position of the current subblock within the current macroblock, and is indexed from 0 to 7, known as bands. The mapping of sub-blocks to the index is shown in Figure 2.2. The upper half of the macroblock and the last sub-block are treated specially, while the lower half shares index 6.

![Figure 2.2: Subblock Mapping to the Token Probability Table](image)

The local complexity dimension attempts to match the local area to the corresponding probability. If there are many zeros in the local area, it is more likely that index 0 is used. If there are some, but not a lot, 1 is used. If there is a large amount, index 2 is used.

For the first coefficient of the macroblock, the surrounding macroblocks are examined. The index is the number of surrounding macroblocks that contain at least one non-zero coefficient in their residue. This way, the first coefficient’s probability accuracy depends on how similar it is to the immediately surrounding macroblocks.

For the remainder of the coefficients, the local complexity index is described by Eq. 2.1.1, where $i_{lc}$ is the local complexity index, and $c_{last}$ is the previous coefficient decoded.
This can have suboptimal behavior when wrapping around the macroblock, for instance, from position 3 to position 4.

As the meaning between the first and remaining coefficients is slightly different for the local complexity dimension, it is important to note that this is acceptable, because each subblock position maps to a band, as shown in Figure 2.2. Therefore, the first coefficient has its own probabilities for the cases of surrounding macroblocks, and it doesn’t interfere with the other meaning of local complexity, which is the value of the previous coefficient, as described in Eq. 2.1.1.

\[
i_{lc} \leftarrow \begin{cases} 
0 & \text{if } c_{last} = 0 \\
1 & \text{if } |c_{last}| = 1 \\
2 & \text{if } |c_{last}| > 1 
\end{cases} \tag{2.1.1}
\]

**Inverse DCT and WHT**

Before dequantizing the DCT and WHT factors, the inverse transform must be performed. If the Y2 block exists (if a prediction mode other than B.PRED for intra- and SPLITMV for inter-prediction is used), then it is inverted via the WHT.

The WHT is defined in Eq. 1.3.5, and is the inverse WHT is implemented in Alg. 1 on page 24, where input and output are arrays of signed 16-bit integers. This is an \(O(n \log n)\) implementation of the Fast Walsh-Hadamard Transform, which uses an architecture similar to the Cooley-Tukey algorithm for Fast Fourier Transforms[19]. There is also an optimization if there is only one non-zero DC value in the input array.

For the inverse DCT, two passes of a 1D inverse DCT is used.

Alg. 2 on page 41 is used to compute the inverse DCT. The two constants \(f_1\) and \(f_2\) are typically expressed as 16-bit fixed-point fractions, and are given in the bitstream reference as \(f_1 \cdot 2^{16} = 20091\) and \(f_2 \cdot 2^{16} = 35468\).
Algorithm 1 4 × 4 Inverse WHT

\[
i \leftarrow 0
\]
\[
c \leftarrow 0
\]
\textbf{while } i < 4 \textbf{ do}
\[
a_1 \leftarrow \text{input}[c + 0] + \text{input}[c + 12]
\]
\[
b_1 \leftarrow \text{input}[c + 4] + \text{input}[c + 8]
\]
\[
c_1 \leftarrow \text{input}[c + 4] + \text{input}[c + 8]
\]
\[
d_1 \leftarrow \text{input}[c + 0] + \text{input}[c + 12]
\]
\[
\text{output}[c + 0] \leftarrow a_1 + b_1
\]
\[
\text{output}[c + 4] \leftarrow c_1 + d_1
\]
\[
\text{output}[c + 8] \leftarrow a_1 - b_1
\]
\[
\text{output}[c + 12] \leftarrow d_1 - c_1
\]
\[
i \leftarrow i + 1
\]
\[
c \leftarrow c + 1
\]
\textbf{end while}

\[
i \leftarrow 0
\]
\[
c \leftarrow 0
\]
\textbf{while } i < 4 \textbf{ do}
\[
a_1 \leftarrow \text{input}[c + 0] + \text{input}[c + 3]
\]
\[
b_1 \leftarrow \text{input}[c + 1] + \text{input}[c + 2]
\]
\[
c_1 \leftarrow \text{input}[c + 1] + \text{input}[c + 2]
\]
\[
d_1 \leftarrow \text{input}[c + 0] + \text{input}[c + 3]
\]
\[
\text{output}[c + 0] \leftarrow \frac{1}{8}(a_1 + b_1 + 3)
\]
\[
\text{output}[c + 1] \leftarrow \frac{1}{8}(c_1 + d_1 + 3)
\]
\[
\text{output}[c + 2] \leftarrow \frac{1}{8}(a_1 - b_1 + 3)
\]
\[
\text{output}[c + 3] \leftarrow \frac{1}{8}(d_1 - c_1 + 3)
\]
\[
i \leftarrow i + 1
\]
\[
c \leftarrow c + 4
\]
\textbf{end while}

2.1.3 Boolean Entropy Encoder

The VP8 data stream is compressed via a boolean entropy encoder, which is a type of arithmetic coder (see Section 1.3.3). For this type of arithmetic coding, there are only two symbols, true or false. The goal of VP8’s other steps, such as the DCT coefficient coding and prediction is to insert more false values than true values, so that the probability of a false value is higher, thus increasing the efficiency of the boolean entropy encoder.
The equation for determining the smallest datarate per value is in Eq. 2.1.2, where $R$ is measured in $\text{bits/value}$.

$$R = -p \log_2(p) - (1-p) \log_2(1-p)$$ \hspace{1cm} (2.1.2)

At the value $p = \frac{1}{2}$, $R = 1 \text{bits/value}$, which is the worst case of the boolean entropy encoder. However, at $p = \frac{1}{1063}$, $R = 0.01 \text{bits/value}$, which is substantially better than encoding every single boolean.

**Bit Representation of the Entropy Encoder**

The probabilities that the boolean entropy encoder works with in VP8 are unsigned 8-bit integers, $p'$. To get the actual probability $p \in [0, 1]$, the 8-bit integer is divided by 256.

The state of the encoder is maintained with five values, the current bit position $n$, the bit string already written, $w$, the bottom value, an 8-bit integer $i_{bot}$, and the range, another 8-bit integer, $i_{rng}$. The range is clamped to within a specified boundary, so that the probabilities remain accurate, $i_{rng} \in [128, 255]$.

The value $v$ is the next value of $w$, and the final value of $v$ is the end condition, where $v = x$. $v$ must satisfy the inequality in Eq. 2.1.3. The scale of the bit position 8-bits ahead is generated as, $s = 2^{-n-8}$. Another value, split, is calculated as follows, $\text{split} = 1 + \frac{p'(i_{rng}-1)}{256}$, and is constrained, $\text{split} \in [1, i_{rng} - 1]$.

$$w + (s i_{bot}) \leq v < w + (s (i_{bot} + i_{rng}))$$ \hspace{1cm} (2.1.3)

The boolean value to be encoded, $b$, has a zero probability of $\frac{p'}{256}$, where $p' \in [1, 255]$. The process for encoding one boolean value $b$ into the output $w$ is shown in Alg. 3. This algorithm is repeated for each boolean value that must be encoded. This is parallelized in the VP8 encoder to process 8-bits at one time.
To decode a boolean encoded by Alg 3, the decoding process described in Alg. 4 on page 42 is used.

2.1.4 Intraframe Prediction

VP8 uses two types of prediction vectors to achieve high compression performance. The simpler type is intra-prediction, where the frame is predicted from other components of the already constructed frame. As macroblock prediction is resolved in raster-scan order, intra-prediction generally works from the top left, to the bottom right.

Chroma Intraframe Prediction

Chroma intra-prediction works on the 8-by-8 blocks of both U and V chroma. The components of intra-prediction are $M$, which is the 8-by-8 matrix of either U or V, as shown in Figure 2.3.

$A$ is the bottom row of the macroblock above the current macroblock $M$, and is 1-by-8. If $M$ is currently in the topmost position, then the values of $A$ are all 127. $L$ is the rightmost right of the macroblock to the left of the current macroblock $M$, and is 8-by-1. If $M$ is currently in the leftmost position, then the values of $A$ are all 129. $P$ is a scalar, the bottom-rightmost chroma value from the macroblock above and to the left of the current macroblock $M$. If $M$ is currently in the topmost and leftmost position, then the value of $P$ is 129.

Vertical Prediction  Vertical prediction, known as $\text{v\_PRED}$ in the $\text{libvpx}$ source code, fills every 8 chroma column of $M$ with copies of $L$. See Figure 2.4 for a graphical representation of vertical prediction.
Horizontal Prediction  Horizontal prediction, known as H.PRED in the libvpx source code, fills every 8 chroma row of $M$ with $A$. See Figure 2.5 for a graphical representation of horizontal prediction.

DC Prediction  In DC prediction mode, known as DC.PRED in the libvpx source code, every chroma in $M$ is filled with the same value. The DC value is shown in Eq. 2.1.4.
∀i∀j \quad M_{i,j} = \frac{1}{16} \sum_{k=1}^{8} A_k + \frac{1}{16} \sum_{l=1}^{8} L_l \quad (2.1.4)

**True Motion Prediction**  The True Motion prediction mode, known as TM.PRED in the libvpx source code, uses row $A$, column $L$, and chroma pixel $P$ in its prediction. It assigns chroma values according to the algorithm specified in Eq. 2.1.5, with the definition of $\text{Clamp}_{j,k}$ as specified in Eq. 2.1.6.

\[ M_{i,j} = \text{Clamp}_{0,255} (L_j + A_i - P) \quad (2.1.5) \]

where

\[ \text{Clamp}_{j,k}(x) = \begin{cases} k & \text{if } x > k \\ x & \text{if } j > x \geq k \\ j & \text{if } j \geq x \end{cases} \quad (2.1.6) \]
Luma Intraframe Prediction

The prediction process for Luma intra-prediction is nearly identical to the chroma intra-prediction methods, with the primary difference being that luma has a 16-by-16 prediction matrix, while chroma is only 8-by-8. Luma intra-prediction has all the same prediction methods that chroma intra-prediction has (namely, vertical, horizontal, DC, and True Motion prediction) with one additional prediction mode.

The luma-specific intra-prediction mode is comprised of intra-prediction on the four $4 \times 4$ sub-blocks of the macroblock $M$. Each $4 \times 4$ sub-block can be independently intra-predicted via one of 10 sub-block intra-prediction modes. Figure 2.6 shows the many possible components of luma intra-prediction.

For subblock $B_1$, the prediction mode may use the 8 pixel row $A_1$, which spans from one pixel above the top left pixel of $B_1$ to one pixel above the top right pixel of $B_2$. It also may use $P_1$ the single pixel immediately to the left of $A_1$, or $L_1$, which is the 4 pixel column immediately below $P_1$.
Each one of the sub-blocks can use their own corresponding pixels to predict, as shown in Figure 2.6. $B_2$ may use $A_2$, which stretches into the neighboring macroblock $M_2$, or $P_2$, which is the fourth pixel from the left of $A_1$, or $L_2$, which is the rightmost 4 pixel column of $B_1$.

### 2.1.5 Interframe prediction

Interframe prediction is accomplished with a reference frame and offsets relative to the reference frame, known as motion vectors. Motion vectors are vectors through three dimensions: two-dimensions in space, to show movement from one position in a frame to another, and a third in time, to allow the motion to occur. Each macroblock for the current frame is predicted using the 16 luma sub-blocks and the 8 chroma red and 8 chroma blue sub-blocks.

Macroblocks in interframes can be intra-coded. This can be used to provide intra-refreshes between long key frames, where, at specific intervals, each interframe codes one column of intra-coded macroblocks. These columns sweep from left to right, refreshing the image as a fully coded key frame would.

**Motion Vectors**

Decoding the motion vectors will provide not only the sixteen Y sub-blocks with individual motion vectors, but it will also define the inter-prediction buffer. Like H.264, quarter pixel (qpel) precision – explored in “Interpolation and Filtering” on page 31 – is supported by VP8. This is accomplished by interpolating between two pixels to get a median value, and then interpolating between the median value and the two pixels to get quarter median values.

There are five types of motion vectors, described in Table 2.1. The probability table
used to decode the motion vectors is calculated via three neighboring macroblocks, top, left, and top-left. For macroblocks on the topmost or leftmost edge, there are implicit macroblocks with zero motion vectors. If a neighboring macroblock was intra-coded, it does not factor in to the calculation.

Table 2.1: The Five Types of Motion Vectors

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest</td>
<td>Use the nearest MV for this MB</td>
</tr>
<tr>
<td>Near</td>
<td>Use the next nearest MV for this MB</td>
</tr>
<tr>
<td>Zero</td>
<td>Use a zero MV for this MB</td>
</tr>
<tr>
<td>New</td>
<td>Use an explicit offset from implicit MV for this MB</td>
</tr>
<tr>
<td>Split</td>
<td>Use multiple MVs for this MB</td>
</tr>
</tbody>
</table>

To determine the best, nearest and near motion vectors, an algorithm determines the weight of the motion vector in question by examining surrounding macroblocks. Alg. 5 on page 43 shows the method by which this works. The function $frametype(M)$ takes a macroblock and returns the prediction mode used. If $F_{intra}$ is used, the algorithm will skip the macroblock in question. The function $MV(M)$ returns the motion vector for the corresponding macroblock.

**Interpolation and Filtering**

Interpolation is used to achieve quarter pixel accuracy in the motion vector estimation in VP8. Quarter pixel accuracy indicates that not only are individual pixels accessible by motion vectors, but the space between pixels (half pixel accuracy), and the space one quarter of the way between pixels. VP8 uses two types of FIR filters, bicubic filters for higher quality estimation, and bilinear filters for reduced complexity estimation. The bicubic filter has six filter coefficients (taps), and has eight preset filters, which are used depending on the circumstance.

The basic interpolation for either the bicubic or bilinear interpolation filter is by convo-
olution. The process uses a clamped convolution, as shown in Eq. 2.1.7, which defines the function $i\text{filter}$. This convolution is used in both passes, the horizontal initial pass, and the final vertical pass, which uses the $i\text{filter}_{\text{vert}}$ variant. The variant uses columns rather than rows, and takes advantage of the fact that the width of the $temp$ array mentioned in Alg. 6 on page 44 is of width 4.

$$i\text{filter} \left( \vec{x}, c \right) = \text{Clamp}_{0,255} \left( \sum_{c=-2}^{3} b_{c+2} \cdot x \left[ c \right] \right)$$

$$i\text{filter}_{\text{vert}} \left( \vec{x}, c \right) = \text{Clamp}_{0,255} \left( \sum_{c=-2}^{3} b_{c+2} \cdot x \left[ 4c \right] \right)$$

(2.1.7)

For the initial horizontal pass, nine rows of four interpolated values each are calculated, as shown in Alg. 6. The initial value, $p$, is used here not as a single scalar, but a scalar in a linear array of pixels. The function $i\text{filter}$ references pixels before and after the given value. After the $temp$ array of values is generated, the final vertical pass is performed, as shown in Alg. 7 on page 44.

The final array, is a two-dimensional array four pixels wide by four pixels tall, which contains the interpolated qpel values for the bottom right for the given pixel. The filter taps, given in Eq. 2.1.7 as $b$, are predefined. The choices for bicubic interpolation are whole pixel (copies the given pixel), $1/8$, $1/4$, $3/8$, symmetric $1/2$, $5/8$, $3/4$, and $7/8$. Bilinear pre-defined filters are also available for simpler encoding modes.

### 2.1.6 Scan Order

VP8 processes data in a predefined order, known as raster-scan order. This ordering begins at the top left, and progresses from left-to-right and top-to-bottom, in the same way that English is read. Figure 2.7b shows this scan order.

For storing the DCT or WHT transformed coefficients, VP8 stores them in the same
way, as in Figure 2.7b. As the transformed coefficients are most similar near the DC value, and grow in both the X and Y dimensions in the same way, H.264 uses a zig-zag packing method, as shown in Figure 2.7a.

This groups similar sized elements together for better lossless compression. This is due to the way the two-dimensional DCT organizes its data. High frequency data is in the bottom right, and lower frequencies radiate outwards. Snacking back and forth lumps together similar coefficients for a lower entropy, and thus a higher compression efficiency.

H.264 also has field scan order for interlaced videos, but VP8 does not support interlaced videos.

![Figure 2.7: Scan Orders for Transformed Coefficients](image)

2.1.7 Loop Filter

As VP8 operates on $4 \times 4$ macroblocks, there can be sharp contrast on the edge of two macroblocks. These aberrations can be unsightly, and need to be smoothed before the final image is ready. This process is known as loop filtering. This isn’t merely a cosmetic detail that the decoder does before producing a visible frame; its results are used for inter-prediction as well.
As loop filtering is very computationally intensive, there are several types of filtering, on the frame level and on the macroblock level. The frame header can select one of three types, “none”, “simple”, and “normal”. Each macroblock, assuming that “none” was not selected for this frame, can select its own loop filter that may be different than the frame default loop filter.

There are two control parameters, the loop filter level, denoted $F_l$, which can be modified on a per-macroblock basis, and the sharpness level, denoted $S_l$, which is constant for a frame. The parameter $F_l$ is a threshold, which can denote that differences below $F_l$ are artifacts arising from compression, and therefore needs to be smoothed. $F_l$ is generally related to quantization levels: the higher the quantization, the higher $F_l$ should be to smooth those anomalies out.

**Construction of Thresholds and Parameters**

The parameter $I_l$, the interior limit, is a parameter used to calculate various thresholds. Its derivation is seen in Eq. 2.1.8.

$$I_l' \left\{ \begin{array}{ll} \frac{F_l}{4} & \text{if } S_l > 4 \\ \frac{F_l}{2} & \text{if } S_l > 0 \\ F_l & \text{otherwise} \end{array} \right. \quad (2.1.8)$$

$$I_l \left\{ \begin{array}{ll} 9 - S_l & \text{if } F_l' > 9 - S_l \text{ and } F_l \neq I_l' \\ I_l' & \text{otherwise} \end{array} \right. \quad (2.1.9)$$

The loop edge thresholds, the macroblock edge threshold $E_{MB}$, and the subblock edge threshold $E_{SB}$, are calculated as seen in Eq. 2.1.9.
$$E_{MB} \leftarrow 2(F_I + 2) + I_l$$

$$E_{SB} \leftarrow 2F_I + I_l$$

For high edge variance thresholds, the value $E_{HV}$ is defined in Eq. 2.1.10.

$$E_{HV} \leftarrow \begin{cases} 
2 & \text{if decoding key frame and } F_I \geq 40 \\
1 & \text{if decoding key frame and } F_I \geq 15 \\
3 & \text{if not decoding key frame and } F_I \geq 40 \\
2 & \text{if not decoding key frame and } F_I \geq 20 \\
1 & \text{if not decoding key frame and } F_I \geq 15 
\end{cases}$$

### Pixel Adjustment

If either of the loop filters determine that a set of edge pixels needs to be adjusted, then a common adjustment routine is followed. This routine uses the $Clamp_{j,k}$ function, Eq. 2.1.6 extensively.

First, the parameters $A, A', \text{ and } B$ are set up in Eqs. 2.1.12 and 2.1.13. For convenience, Eq. 2.1.11 is used to shorten the notation to indicate the clamping of a signed byte.

$$C_B(x) = Clamp_{-128,127}(x)$$

$$A' \leftarrow \begin{cases} 
C_B (C_B (P_1 - Q_1) + 3 (Q_0 - P_0)) & \text{if using outer taps} \\
3 (Q_0 - P_0) & \text{if not using outer taps} 
\end{cases}$$

$$A \leftarrow \frac{A' + 4}{8}$$
\[ B \left\{ \begin{array}{ll} -1 & \text{if lower 3-bits of } A' = 4 \\ 0 & \text{otherwise} \end{array} \right. \]  

(2.1.13)

Then, the bordering pixels are adjusted using the parameters \( A \) and \( B \) in Eqs. 2.1.14 and 2.1.15.

\[
P'_0 \leftarrow P_0 + C_B (A + B) \]  

(2.1.14)

\[
Q'_0 \leftarrow Q_0 - A \]  

(2.1.15)

**Simple Loop Filter**

The primary focus of any macroblock loop filter is to reduce the difference in pixels along an edge. To achieve this, the simple loop filter adjusts only the luma values, leaving the chroma values unaffected. Differences in luma values above \( F''_l \) are assumed to be natural differences in the input video, while values below are assumed to be artifacts from the compression process.

The simple filter operates on four luma pixels in either a row, if the edge is vertical, or a column, if the edge is horizontal. The four pixels are termed \( P_0 \), the pixel before the edge, \( P_1 \), the pixel preceding \( P_0 \), \( Q_0 \), the pixel after the edge, and \( Q_1 \), the pixel after \( Q_0 \). See Figures 2.8 and 2.9 for a diagram of these border pixels. If the inequality in Eq. 2.1.16, where \( E \) is either \( E_{MB} \) or \( E_{SB} \) depending on the type of the current edge, is true, then the pixels are adjusted. If the inequality is false, the pixels do not change.

\[
E \geq 2 |P_0 - Q_0| + \frac{|P_1 - Q_1|}{2} \]  

(2.1.16)
Unlike the simple loop filter, the normal loop filter differs depending on whether the edge is a macroblock or subblock edge. There are strong similarities between the two algorithms, and they each use common algorithms for thresholding.

If every inequality of Eq. 2.1.17 is true, given the correct value of $E$ (either $E_{MB}$ for macroblocks or $E_{SB}$ for sub-blocks), then the filtering is enabled. Otherwise, the filtering is not used for this set of pixels.

Filtering depends on how severe the edge variance is, and Eq. 2.1.18 is used to determine if the current edge has high edge variance. If either inequalities of Eq. 2.1.18 are true, then the edge has high variance.
$E \geq 2|P_0 - Q_0| + \frac{|P_1 - Q_1|}{2}$ \hspace{1cm} (2.1.17)

$I_i \geq |P_3 - P_2|$
$I_i \geq |P_2 - P_1|$
$I_i \geq |P_1 - P_0|$
$I_i \geq |Q_3 - Q_2|$
$I_i \geq |Q_2 - Q_1|$
$I_i \geq |Q_1 - Q_0|$

$E_{HV} < |P_1 - P_0|$, \hspace{1cm} (2.1.18)

$E_{HV} < |Q_1 - Q_0|$

**Macroblock Edge Adjustment**  The process for determining whether the filter should be enabled for macroblocks is as follows. If the inequalities in Eq. 2.1.17 are true, then the inequalities in Eq. 2.1.18 are evaluated. If either is true, then the pixel adjustment procedure is performed as described in Section 2.1.7 is used, with outer taps enabled.

If both inequalities of Eq. 2.1.18 are false (no high edge variance), then the pixels are adjusted according to Eq. 2.1.19.

For convenience, Eq. 2.1.11 is used to shorten the notation to indicate the clamping of a signed byte.
\[
W \leftarrow C_B (C_B (P_1 - Q_1) + 3 (Q_0 - P_0)) \tag{2.1.19}
\]

\[
X'' \leftarrow C_B \left( \frac{1}{27} (27W + 63) \right)
\]

\[
Q_0 \leftarrow Q_0 - X''
\]

\[
P_0 \leftarrow P_0 + X''
\]

\[
X' \leftarrow \frac{1}{27} C_B (18W + 63)
\]

\[
Q_1 \leftarrow Q_1 - X'
\]

\[
P_1 \leftarrow P_1 + X'
\]

\[
X \leftarrow \frac{1}{27} C_B (9W + 63)
\]

\[
Q_2 \leftarrow Q_2 - X
\]

\[
P_2 \leftarrow P_2 + X
\]

**Subblock Edge Adjustment**  The process for determining whether the filter should be enabled for sub-blocks is as follows. If the inequalities in Eq. 2.1.17 are true, then the inequalities in Eq. 2.1.18 are evaluated. If either is true, then the pixel adjustment procedure is performed as described in Section 2.1.7 is used, with outer taps enabled.

If both inequalities of Eq. 2.1.18 are false (no high edge variance), then the pixel adjustment procedure described in Section 2.1.7 is used, with outer taps disabled. Then, only if there was no high edge variance, pixels \(P_1\) and \(Q_1\) are adjusted further. The pixels are adjusted according to Eq. 2.1.20, where \(A\) is the parameter from Eq. 2.1.12.
\[ P_1 \leftarrow C_B \left( P_1 + \frac{A}{2} \right), \]
\[ Q_1 \leftarrow C_B \left( Q_1 - \frac{A}{2} \right) \]
Algorithm 2 $4 \times 4$ Inverse DCT

\begin{align*}
  f_1 &\leftarrow \sqrt{2} \cos (\pi/8) - 1 \\
  f_2 &\leftarrow \sqrt{2} \sin (\pi/8) \\
  i &\leftarrow 0 \\
  c &\leftarrow 0 \\
  spitch &\leftarrow \frac{\text{pitch}}{2} \\
  \textbf{while} \ i < 4 \ \textbf{do} \\
  &a_1 \leftarrow \text{input}[c + 0] + \text{input}[c + 8] \\
  &b_1 \leftarrow \text{input}[c + 0] - \text{input}[c + 8] \\
  &t_1 \leftarrow f_2 \cdot \text{input}[c + 4] \\
  &t_2 \leftarrow \text{input}[c + 12] + (f_1 \cdot \text{input}[c + 12]) \\
  &c_1 \leftarrow t_1 - t_2 \\
  &t_1 \leftarrow \text{input}[c + 4] + (f_1 \cdot \text{input}[c + 4]) \\
  &t_2 \leftarrow f_2 \cdot \text{input}[c + 12] \\
  &d_1 \leftarrow t_1 + t_2 \\
  &\text{output}[c + 0 \cdot \text{spitch}] \leftarrow a_1 + d_1 \\
  &\text{output}[c + 3 \cdot \text{spitch}] \leftarrow a_1 - d_1 \\
  &\text{output}[c + 1 \cdot \text{spitch}] \leftarrow b_1 + c_1 \\
  &\text{output}[c + 2 \cdot \text{spitch}] \leftarrow b_1 - c_1 \\
  &i \leftarrow i + 1 \\
  &c \leftarrow c + 1 \\
  \textbf{end while} \\
  \text{end while} \\
  i &\leftarrow 0 \\
  c &\leftarrow 0 \\
  \textbf{while} \ i < 4 \ \textbf{do} \\
  &a_1 \leftarrow \text{input}[c + 0] + \text{input}[c + 2] \\
  &b_1 \leftarrow \text{input}[c + 0] - \text{input}[c + 2] \\
  &t_1 \leftarrow f_2 \cdot \text{input}[c + 1] \\
  &t_2 \leftarrow \text{input}[c + 3] + (f_1 \cdot \text{input}[c + 3]) \\
  &c_1 \leftarrow t_1 - t_2 \\
  &t_1 \leftarrow \text{input}[c + 1] + (f_1 \cdot \text{input}[c + 1]) \\
  &t_2 \leftarrow f_2 \cdot \text{input}[c + 3] \\
  &d_1 \leftarrow t_1 - t_2 \\
  &\text{output}[0] = \frac{1}{18} (a_1 + d_1 + 4) \\
  &\text{output}[3] = \frac{1}{18} (a_1 - d_1 + 4) \\
  &\text{output}[1] = \frac{1}{18} (b_1 + c_1 + 4) \\
  &\text{output}[2] = \frac{1}{18} (b_1 - c_1 + 4) \\
  &i \leftarrow i + 1 \\
  &c \leftarrow c + \text{spitch} \\
  \textbf{end while} \\
\end{align*}
**Algorithm 3** Encoding a Boolean Value

\[
\text{split} \leftarrow 1 + \frac{p'(i_{\text{rng}} - 1)}{256}
\]

\[
\text{if } b = \text{false} \text{ then }
\]

\[
i_{\text{rng}} \leftarrow \text{split}
\]

\[
\text{else}
\]

\[
i_{\text{rng}} \leftarrow i_{\text{rng}} - \text{split}
\]

\[
i_{\text{bot}} \leftarrow i_{\text{bot}} + \text{split}
\]

\[
\text{end if}
\]

**while** $i_{\text{rng}} < 128$ **do**

\[
i_{\text{rng}} \leftarrow i_{\text{rng}} \times 2
\]

\[
\text{if detect.carry}(i_{\text{bot}}) \text{ then}
\]

\[
w \leftarrow w + 1
\]

\[
\text{end if}
\]

\{May be implemented as a series of shift operations\}

\[
i_{\text{bot}} \leftarrow i_{\text{bot}} \times 2
\]

\[
w \leftarrow w \times 2 + i_{\text{bot}}/128
\]

**end while**

**Algorithm 4** Decoding a Boolean Value

\[
\text{split} \leftarrow 1 + \frac{p'(i_{\text{rng}} - 1)}{256}
\]

\[
\text{split}' \leftarrow \text{split} \times 256
\]

\[
\text{if } \text{value} < \text{split}' \text{ then}
\]

\[
i_{\text{rng}} \leftarrow \text{split}
\]

\[
b \leftarrow \text{false}
\]

\[
\text{else}
\]

\[
i_{\text{rng}} \leftarrow i_{\text{rng}} - \text{split}
\]

\[
\text{value} \leftarrow \text{value} - \text{split}'
\]

\[
b \leftarrow \text{true}
\]

**end if**

**while** $i_{\text{rng}} < 128$ **do**

\[
i_{\text{rng}} \leftarrow i_{\text{rng}} \times 2
\]

\[
\text{value} \leftarrow \text{value} \times 2
\]

**end while**
Algorithm 5 Determining Motion Vector Weight from Neighbors

\[ i \leftarrow 0 \]
\[ n \leftarrow 0 \]
\[ \text{if } \text{frametype}(MB_{top}) \neq F_{\text{intra}} \text{ then} \]
\[ \text{if } MV(MB_{top}) \neq 0 \text{ then} \]
\[ n \leftarrow 1 \]
\[ \text{near}[n] \leftarrow MV(MB_{top}) \]
\[ \text{Bias Calculation} \]
\[ \text{count}[1] \leftarrow 2 \]
\[ i \leftarrow 1 \]
\[ \text{end if} \]
\[ \text{count}[i] \leftarrow 2 \]
\[ \text{end if} \]
\[ \text{if } \text{frametype}(MB_{left}) \neq F_{\text{intra}} \text{ then} \]
\[ \text{if } MV(MB_{left}) \neq 0 \text{ then} \]
\[ \text{tmp} \leftarrow MV(MB_{left}) \]
\[ \text{Bias Calculation} \]
\[ \text{if } \text{tmp} \neq MV(MB_{left}) \text{ then} \]
\[ n \leftarrow n + 1 \]
\[ \text{near}[n] \leftarrow MV(MB_{top}) \]
\[ i \leftarrow i + 1 \]
\[ \text{end if} \]
\[ \text{count}[i] \leftarrow \text{count}[i] + 2 \]
\[ \text{else} \]
\[ \text{count}[0] \leftarrow \text{count}[0] + 2 \]
\[ \text{end if} \]
\[ \text{end if} \]
\[ \text{if } \text{frametype}(MB_{topleft}) \neq F_{\text{intra}} \text{ then} \]
\[ \text{if } MV(MB_{topleft}) \neq 0 \text{ then} \]
\[ \text{tmp} \leftarrow MV(MB_{topleft}) \]
\[ \text{Bias Calculation} \]
\[ \text{if } \text{tmp} \neq MV(MB_{left}) \text{ then} \]
\[ n \leftarrow n + 1 \]
\[ \text{near}[n] \leftarrow MV(MB_{top}) \]
\[ i \leftarrow i + 1 \]
\[ \text{end if} \]
\[ \text{count}[i] \leftarrow \text{count}[i] + 1 \]
\[ \text{else} \]
\[ \text{count}[0] \leftarrow \text{count}[0] + 1 \]
\[ \text{end if} \]
\[ \text{end if} \]
Algorithm 6 Initial Horizontal Pass for Interpolation

\[ \text{row} \leftarrow 0 \]
\[
\text{while } \text{row} < 9 \text{ do}
\]
\[
\text{col} \leftarrow 0
\]
\[
\text{while } \text{col} < 4 \text{ do}
\]
\[
\text{temp}[\text{row}][\text{col}] \leftarrow \text{ifilter}(p, \text{col})
\]
\[
\text{end while}
\]
\[
\text{end while}
\]

Algorithm 7 Final Vertical Pass for Interpolation

\[ \text{row} \leftarrow 0 \]
\[
\text{while } \text{row} < 4 \text{ do}
\]
\[
\text{col} \leftarrow 0
\]
\[
\text{while } \text{col} < 4 \text{ do}
\]
\[
\text{final}[\text{row}][\text{col}] \leftarrow \text{ifilter}_\text{vert}(p, \text{col})
\]
\[
\text{end while}
\]
\[
\text{end while}
\]
2.2 Features Unique to VP8 or H.264

As both VP8 and H.264 are detailed standards with many areas of application, there are features unique to either of them, and there are many features in common. By examining the features unique to each codec, we can analyze the intent of the designers and their limitations.

H.264 is a large standard, with many configurations, optional features, and recommendations. One of the most prominent features of H.264 is the concept of profiles. Originally, three profiles were supported: Baseline, Main, and Extended Profile[23]. The Baseline and Extended Profile include features that were suited for streaming video; features that allow for new slice types such as SP or SI slices, flexible macroblock ordering (FMO), arbitrary slice ordering (ASO), and redundant pictures. This, and VP8’s streaming support, is discussed in more detail in Section 2.2.3.

H.264 Main profile has largely be supplanted by the High profile introduced in the Fidelity Range Extensions of H.264/MPEG4-AVC[24]. It has B frames, interlaced coding, and CABAC that the Constrained Baseline profile lacks, and gains adaptive transform capability, quantization scaling matrices, monochrome support, and individual quantization factors for $C_r$ and $C_b$. High profile is the default of many modern H.264 encoders, such as x264 and MainConcept.

2.2.1 B-Frames, Alternate Reference Frames, and Golden Frames

In addition to intra frames and predictive frames, which both VP8 and H.264 have, H.264 contains bidirectional frames (B frames). Like P frames, B frames also reference the previous frame for inter-prediction, but they also reference frames that have not yet been decoded. While this increases the complexity of both the encoder and the decoder, it has a significant effect on performance. Figure 2.10 illustrates B frames in more detail: ar-
rows indicate what motion vectors that frame references. I frames do not reference other frames.

Further, H.264 brings the granularity of I, P, and B frames to the slice level. A frame in H.264 is divided up into a number of slices, which may contain a variable number of macroblocks. The slices need not be rectangular. With ASO, discussed in Section 2.2.3, the slices need not be in a particular order. VP8 lacks this detail, as the frame type determines the types of the macroblocks.

![Bidirectional Frames in H.264](image)

Figure 2.10: Bidirectional Frames in H.264

VP8, however, has only P frames. However, VP8’s P frames may be predicted from one of three sources: from the previous frame (as in H.264), from the most recent golden frame, or from the most recent alternate reference (altref) frame. Golden frames are frames that are blessed by the encoder as being suitable for prediction. I frames are automatically golden frames. This gives the encoder an extra option for P frames, either the previous frame or the most recent golden frame.

Another option is an alternate reference frame. These frames can be hidden, that is, they are optionally visible in the decoded video output. Instead, they are constructed from some number of previously encoded frames or from frames not yet encoded, and then can be referenced by any future frames. This power to construct the altref frames from any number of previous frames or from future is very powerful, and it can substantially improve decoding performance. An intelligent VP8 encoder could use altref frames as pseudo replacements for B frames[3].

Figure 2.11 shows altref and golden frames in VP8. Golden frames are marked with a
star, and altref frames are on a different track than the linear sequence of decoded videos. In Figure 2.11, the first P frame references the first I frame, which is automatically a golden frame. The second P frame is blessed as a golden frame by the encoder.

The next P frame references its previous frame, and the fourth P frame references the most recent golden frame, two frames back. An altref frame is constructed by referencing the first two P frames. This is referenced by the following P frame, which is then blessed as a golden frame. The second altref frame is constructed from an I frame and the P frame preceding it. This can help in situations where there is a rapid alternating between two scenes, without the need for scene cuts: one scene can be kept at the golden frame reference, and the other can be kept at the alternate reference.

B frames are an obvious advantage for H.264, but their complexity keeps them out of the Constrained Baseline profile, which is most common in hardware decoders on embedded devices. The golden and altref frame could be a suitable replacement, however, and it offers the possibility to be comparable to B frames in the absence of them. It is largely up to the encoder implementation to make intelligent use of B frames or VP8’s scheme.
2.2.2 Adaptive Behavior

The ability for a codec to adapt to different sections of a video frame – for instance, credits scrolling down the left side with action on the right side – and over time is essential to its compression efficiency across a range of videos and situations. Both VP8 and H.264 make extensive use of adaptive behavior, but each is different in critical ways.

Segments and Slices

Both H.264 and VP8 can group similar macroblocks together to specify parameters for the entire group, saving much needed bandwidth. In H.264 these groups are known as slices, and when flexible macroblock ordering (FMO, discussed in Section 2.2.3) is not used, processed in raster scan order. Macroblocks in slices do not use data in other slices, and they can be decoded in parallel. Figure 2.12a shows an example of how an H.264 encoder might allocate macroblocks into slices within a frame. The different patterns represent different slices. There can be many slices within an H.264 frame.

VP8 does not use the slice concept directly. Each macroblock in a VP8 frame can encode a segment identification number, 1 through 4, to indicate which quantization step size it uses. Figure 2.12b shows how VP8 could organize its macroblocks into segments. The segments need not be contiguous or have any predefined order. However, there is a strict maximum of four segments in a VP8 frame. Using segments is less flexible than H.264’s slice method, due to the low limit on the number of slices, and is of a similar complexity. The marginal benefit of segments being noncontiguous is likely of little importance.
Quantization

Quantization is perhaps the most significant component of video coding. By reducing the range that parameters can vary, compression is achieved at the cost of reconstruction: quantization is lossy and removes information from the encoded stream. However, a video frame might have a range of areas, each best encoded with a particular quantization step size, and adaptive quantization can help increase the amount of detail retained.

H.264 can use individually specified quantization step sizes for each macroblock, slice, or entire frame, enabling a flexible and powerful quantization method. VP8 can specify the quantization step size for either the entire frame or for a segment.

Transform

H.264 uses three transform sizes: 4 × 4 for high detail retention, 8 × 8 transform block size for chroma, and a 16 × 16 block size for very low frequency information [23]. This enables H.264 to adapt to spatial cohesion – the degree to which pixel values in a frame resemble their neighbors – in a frame by selecting a larger block size.

VP8, alternatively, uses only a 4 × 4 block transform. Similarly, the DC coefficients of each transform in a macroblock are transformed using a WHT, as described in Section 2.1.2. The H.264 DCT transform, Eq. 2.2.1 is less precise than VP8’s DCT transform, Alg. 2 on page 41.
VP8 uses fixed point arithmetic to encode the values $\sqrt{2} \cos(\pi/8) - 1$ and $\sqrt{2} \sin(\pi/8)$. H.264 uses fixed point arithmetic, but has no bits reserved for representing the fractional components of their numbers. This approximation loses compression efficiency, but at an acceptable level[22]. VP8’s WHT is seen in Eq. 2.2.2.

$$H = \begin{bmatrix}
1 & 1 & 1 & 1 \\
2 & 1 & -1 & -2 \\
1 & -1 & -1 & 1 \\
1 & -2 & 2 & -1
\end{bmatrix} \quad (2.2.1)$$

$$H_w = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & -1 & -1 \\
1 & -1 & 1 & -1 \\
1 & -1 & -1 & 1
\end{bmatrix} \quad (2.2.2)$$

Entrophy Coding

VP8’s boolean entropy encoder, Section 2.1.3, adapts to changes in the probabilities on a frame-to-frame basis. The probability table for encoding can be updated, via deltas, for each frame, and can be completely reset on I frames or, optionally, on golden frames. The variable-length coding (VLC) tables from H.264’s Context-Adaptive Variable-Length Coding (CAVLC) are the same: they can only be updated on a frame basis, lowering the possibility for optimization.

This is in contrast to Context-Adaptive Binary Arithmetic Coding (CABAC), which is used in H.264 Main and High profiles. After encoding each binarized value, the probability models are updated, giving CABAC a high degree of adaptability[25]. CABAC uses arithmetic coding, like VP8’s boolean entropy coder, and has a higher degree of complexity than CAVLC, which is the only entropy coder available in the Baseline profile.
Deblocking Filter

Contrasted to VP8, H.264 filters macroblock edges with less data. VP8, in Eqs. 2.1.17 and 2.1.18, shows the inequalities that, if any are true, the normal loop filter is enabled, which uses the threshold $I_t$, derived in Eq. 2.1.8. H.264, on the other hand, uses a simpler scheme, as shown in Eqs. 2.2.3 and 2.2.4[23].

Instead of generating the thresholds from sharpness and filter levels, as VP8 does, H.264 opts to use the quantization parameter ($QP$) and two scaling factors, $\alpha$ and $\beta$ which has a requirement of $\beta \ll \alpha$. If either inequality in Eq. 2.2.4 are true, then the pixels $P_1$ and $Q_1$ are adjusted, whereas Eq. 2.2.3 adjusts pixels $P_0$ and $P_1$.

H.264’s encoder can only pick two parameters to adjust its loop filter, unlike VP8, which can choose between two qualities of loop filters and adjust their sharpness and levels independently on a segment basis.

\[
|P_0 - Q_0| < \alpha \cdot QP \\
|P_1 - P_0| < \beta \cdot QP \tag{2.2.3} \\
|Q_1 - Q_0| < \beta \cdot QP \\
|P_2 - P_0| < \beta \cdot QP \\
|Q_2Q_0| < \beta \cdot QP \tag{2.2.4}
\]

We expect that VP8’s deblocking filter will outperform H.264’s in testing due to its high level of control and sophistication.

2.2.3 Streaming Support

Streaming support in modern video codecs is both an important component of codec design and relatively rare in real world implementations. Few transport layers will pass
malformed packets, instead opting to drop them in their entirety. Thus, codecs have
to deal less with bit errors, and more with missing frames, headers, context tables, and
metadata.

**H.264 Streaming Support**

Several features contribute to H.264’s streaming support. The most fundamental is H.264’s
network abstraction layer (NAL). It is flexible enough to allow the video coding layer
(VCL) to be applied to different transport layers – such as file containers, such as MP4,
or Matroska – or to RTP for lossy distribution of video, or to protocols such as H.32X[23].
To do this effectively, the VCL organizes data into NAL units, which represents a packet
of data. The entire scope of the NAL is outside the purview of this thesis, but it offers a
powerful system for organizing H.264 data in many different ways to fit many different
scenarios. Inherent in this, however, is its relative complexity to VP8’s bitstream. VP8’s
bitstream offers little of the same features, however, it is simpler to implement.

H.264, in its Baseline and Extended profiles, offers several features not present in the
Constrained Baseline, Main, or High profiles. One such feature is flexible macroblock
ordering (FMO). FMO partitions slices into slice groups which can be ordered specifically
to increase robustness of a network stream, or to increase coding efficiency. Similar in
spirit to FMO, is arbitrary slice ordering (ASO). Slices in H.264 represent individually
decodable frame data, and thus can be ordered differently than raster scan order. This can
improve performance with networks that have out-of-order network delivery issues[23].
Both of these features are in Baseline and Extended profiles of H.264.

To increase resilience to network disturbances, H.264 offers redundant slice support in
its Baseline and Extended profiles. The encoder can encode several versions of the same
slice, with possibly less bandwidth in redundant slices, at the expense of quality.

In the Extended profile, there are two features related to streaming which do not exist
in the Baseline profile. The first is data partitioning, wherein the encoder can partition the data into different priority levels, where some data is more important than other data. This, coupled with other features, such as FMO and ASO, can result in a decodable video stream even when the network error is high.

The second feature is SP and SI frames, which are additions to the I, P, and B frames for synchronization. When encoding the video, when P frames are normally encoded, occasionally an SP and SI frame are encoded. Under lossless network operation, the SP frame acts like a P frame. However, if a P frame preceding the SP frame is lost, the SI frame is sent instead of the SP frame, which allows the bitstream to resynchronize and not have propagating errors. Some results have shown that a 1.5 dB improvement for videos encoded between 100 Kbps and 600 Kbps[26] that use SP/SI frames on a lossy network.

While these features are impressive, and offer robust network performance, many implementations of H.264 do not implement these, choosing instead to implement the Constrained Baseline, Main, and High profiles instead. The JM reference implementation now supports many of these features, but in some versions, they are unreliable and error prone[27]. One reason for this might be that as of 2005, when many H.264 implementations were written or started, there was no reference implementation of many of these features[26].

**VP8 Streaming Support**

VP8 offers relatively few streaming features. Like H.264, many of VP8’s encoding features, such as the arithmetic coder, are context adaptive. If an incremental update to such an adaptive table is lost, all data until the table is fully defined again (usually at an I frame) will be corrupt. VP8 can optionally fully define all adaptive context tables at all golden frames, allowing the stream to recover quickly.
This does not seem to be that useful of a feature, but when coupled with the altref and golden frames described in Section 2.2.1, an intelligent VP8 encoder could limit the damage a dropped P frame could cause by encoding a P frame every so often that restarts from a previous golden or altref frame. This would end the cascade of errors, and at less bandwidth than encoding I frames more often.

**Analysis in this Thesis**

H.264 has significantly more streaming capabilities than VP8. However, few H.264 encoders and decoders support the more advanced features that the standard provides. VP8’s streaming support is lackluster, but is implemented in their encoder.

Due to the relative scarcity and relevance of streaming support, these features will not be examined in the testing section of this thesis. In the planning stages of this section, a BER error introducer was added to vpxdec, the VP8 decoding program. At even very low BER, such as $10^{-9}$ and lower, or dropping of some packetized bitstream data, the decoder would crash, or give a fatal error. These results significantly hampered any practical analysis of the capabilities of error laden streaming.
2.3 Beyond H.264 and VP8: HEVC and the future

It takes many years for codecs to mature from conception to implementation and public acceptance. H.264 was built upon decades of video coding history. Much of video coding can be seen all the way back to H.261, which had macroblocks, a 4:2:0 YCrCb colorspace, an approximate DCT, an inter-prediction.

High Efficiency Video Coding (HEVC), colloquially known as H.265, is slated to be finalized in 2013. It’s targeted for HD video, 720p, 1080p, and even Ultra HD (7680×4320). It seeks to increase coding efficiency by allowing adaptive transform sizes of up to 64×64, utilizing non-square transform blocks, and using tree prediction. The goal of the Joint Collaborative Team on Video Coding (JCT-VC) seeks to achieve 50% bitrate reductions for the same visual quality when compared to H.264. Initial results show that HM, the current test unoptimized implementation of HEVC, achieves these results on some test sequences[2].

VP8, compared to HEVC, lacks many of these new features that HEVC has. While HEVC does not make improvements to every component of video coding – such as entropy coding, where HEVC still uses CABAC or CAVLC – it offers many features that will make a big difference at very high resolutions, which is an area of video coding that VP8 does poorly in. VP8 would have been wise to optimize for high definition video, as that is where the future of video will be.
2.4 Hardware Implementation of VP8

The goal of many codec implementations in digital logic design is to increase the reusability of each component to reduce the amount of resources that are required to make an encoder and decoder pair. H.264 has numerous design architectures for H.264 Baseline and H.264 High profile.

VP8 shares many similarities with H.264, and the design architecture could likely be the same for components of it. VP8’s boolean entropy coder is an arithmetic coder, and could leverage an existing CABAC architecture, such as Johar et. al’s decoder[28]. VP8 lacks B frames, so there would be no need to have a buffer to store not-yet-played frames, and would look similar to an H.264 Baseline architecture in terms of an image buffer.

VP8 has a maximum of three reference frames, unlike H.264, giving a total of four frames that buffer space needs to be allocated for decoding: the previous key frame, the previous golden frame, the previous alternate reference frame, and the currently decoded frame. VP8’s segment limitation – also restricted to four – helps hardware implementations by being nominally easier to represent than slices. By storing the customization that each segment has in a four index table, each macroblock decode is simply a lookup in this table.

However, VP8’s DCT is fixed point arithmetic rather than the integer math of H.264’s DCT. This would likely require more attention and more bits and registers to represent each index in the macroblock matrix. VP8’s deblocking filter would also likely need more implementation detail, as there are two different filters, each customizable down to the segment level with different filtering strengths.

There exists VHDL and ASIC implementations of both the VP8 encoder and decoder, available from Google\(^1\). It is claimed that with less than a 100 MHz input clock, the

\(^1\)Despite multiple requests, we were not able to obtain the RTL for these designs
decoder is able to decode 1080p video at 30 to 60 fps. This implementation uses 384,000 logic gates and 52 KB of SRAM. The encoder claims usage of 650,000 logic gates and 172 KB of SRAM. These usage counts are higher than H.264 decoders and encoders by more than twice. It is expected, however, that competing implementations could drive down the usage substantially.

In this chapter, VP8 was examined and compared directly with H.264. VP8, while being a modern and powerful codec, lacks many of the features that makes H.264 perform well at high resolutions. VP8’s four segment limitation will likely mean that it will perform poorly at high resolution, but its inherent ASO will likely mean that it performs well at low resolution.

The next chapter will discuss the methodology developed to test VP8 and H.264. They will be compared as a whole for initial testing and use case testing, and tests are developed to test many of the features discussed in this chapter. By examining VP8 and H.264 as a whole and by components, a clear picture of the differences is developed.
Chapter 3

Test Methodology

Testing software as complex as video encoders necessitates a multifaceted approach. The approach taken in this chapter is to develop tests that compare VP8 and H.264 in the most equal manner: feature parity – matching equivalent features to test, such as altref frames in VP8 and B frames in H.264 – is attempted, and approximated where impossible. This means that, for example, in the inter-coding tests, that even though VP8 doesn’t have the ability to weight predicted frames, other variables – such as the ARNR filter strength – were adjusted instead.

The first tests are initial testing to establish a baseline for comparison. These baseline tests occur at several resolutions and bitrates, and mimic how most people would use these encoders. Most of the defaults for the encoders are chosen, except where something is changed for a specific need for a test. Two use cases are studied: low bitrate video conferencing at CIF resolution, and HD film coding with several scenes.

To examine VP8 and compare it to H.264 in detail, three tests were designed to test their capacity to compress data on inter-frames, intra-frames, and the effectiveness of their loop filter implementations. By examining these features independently, the basis that VP8 has – rather than its lack of optimization or maturity – will be shown. The goal
of these tests are to show the potential for VP8 to improve over time.

3.1 Encoders, Configuration, and Data Acquisition

To elucidate the decisions made in this thesis the details of encoding of uncompressed source videos, their configuration, and information acquired thereof are discussed in detail in this section.

3.1.1 Encoders

Choosing not only which codecs to judge VP8 against, but which implementations, is of paramount importance. As x264 is the premier freely available open source implementation of H.264, it was chosen to represent H.264.

The various software versions used in this thesis can be seen in Table 3.1. x264 was chosen as the newest available when testing began. VP8 was chosen as the most recent stable release, because, due to rapid development, the most recent development version produced artifacts in the output. The JM Reference Software 18.0 was chosen, as it was the newest at the time of this writing.

MediaInfo is a widely used tool that can report information about a video file, including the average bitrate of a video stream without container overhead. The open source qpsnr tool reports either the PSNR or SSIM of each frame of a video when compared with a reference.

3.1.2 Data Acquisition

There are several components to acquiring statistical information, and the correct selection of data acquisition methods ensures a fair comparison.
Table 3.1: Versions of the Software Used

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>x264</td>
<td>0.115.1995 c1e60b9 [29]</td>
<td>libswscale 0.11.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>libavformat 52.64.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gcc 4.4.5</td>
</tr>
<tr>
<td>vpxenc</td>
<td>0.9.6 [30]</td>
<td>gcc 4.4.5</td>
</tr>
<tr>
<td>JM Reference</td>
<td>18.0</td>
<td></td>
</tr>
<tr>
<td>MediaInfo</td>
<td>0.7.44</td>
<td></td>
</tr>
<tr>
<td>qpsnr</td>
<td>0.2.1</td>
<td>libswscale 0.11.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>libavformat 52.64.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gcc 4.4.5</td>
</tr>
</tbody>
</table>

After qpsnr is used, the frames are sorted. The minimum, 1st quartile, median, 3rd quartile, and maximum PSNR and SSIM are measured. The average PSNR and SSIM are calculated over all the frames reported by qpsnr. The variance, as discussed in Section 3.3 is computed.

MediaInfo is run to report the average bitrate used by a video file. This doesn’t convey the bandwidth required at all points, but is representative of the bandwidth requirement as a whole. For some tests, such as the high definition film and television tests (Section 3.5.2), the bitrate is kept constant between both encoders, using a two pass encoding. To do this, VP8 is encoded first, as its rate control method is still relatively unstable. Then x264 is used to encode at exactly that bitrate. MediaInfo is used to verify that their average bitrates are equivalent.
3.2 Uncompressed Source Videos

The source videos that were used in these tests are uncompressed video clips, ranging from a few seconds long to a few minutes, at various resolutions. All clips used the 4:2:0 color space at approximately 30 frames per second. These video clips were obtained from the Xiph Foundation\(^1\). The chosen videos are described in Tables 3.2, 3.3, 3.4 and 3.5.

The video clips were in Y4M format, which embeds the metadata – such as the resolution, frames per second, and color space – of the video within the file. A variety of videos were chosen at CIF, 4CIF, 720p, and 1080p resolutions, including a 3D animated trailer. The videos were chosen to match a wide range of possibilities for source videos.

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akiyo</td>
<td>0m10s</td>
<td>Newswoman reading the news, very low movement</td>
</tr>
<tr>
<td>Bridge Close</td>
<td>1m06s</td>
<td>Bridge over water with pedestrians at a far distance</td>
</tr>
<tr>
<td>Coastguard</td>
<td>0m10s</td>
<td>Boat sailing down river with slowly moving background</td>
</tr>
<tr>
<td>Foreman</td>
<td>0m10s</td>
<td>Closeup of worker talking, with camera pan to construction site</td>
</tr>
<tr>
<td>Mobile</td>
<td>0m10s</td>
<td>Complicated scene of calendar moving, train pushing ball, and gyro</td>
</tr>
<tr>
<td>Students</td>
<td>0m30s</td>
<td>Two people speaking at desk, gesturing, low movement</td>
</tr>
</tbody>
</table>

\(^1\)These clips can be obtained at [http://media.xiph.org/video/derf/y4m/](http://media.xiph.org/video/derf/y4m/)
Table 3.3: Source Videos at 704 × 576 Used and their Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>0m10s</td>
<td>Helicopter view of New York City, detail, camera movement</td>
</tr>
<tr>
<td>Crew</td>
<td>0m10s</td>
<td>Astronauts walking, camera flashes, camera movement</td>
</tr>
<tr>
<td>Harbour</td>
<td>0m10s</td>
<td>Sailboats viewed through masts and ropes, camera shakes</td>
</tr>
<tr>
<td>Ice</td>
<td>0m10s</td>
<td>People ice skating with cones and fast movement</td>
</tr>
<tr>
<td>Soccer</td>
<td>0m10s</td>
<td>Soccer game, fast camera movement and fast object movement</td>
</tr>
</tbody>
</table>

Table 3.4: Source Videos at 1280 × 720 Used and their Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobcal</td>
<td>0m10s</td>
<td>Zoom out from detailed picture, train moving along bottom</td>
</tr>
<tr>
<td>Parkrun</td>
<td>0m10s</td>
<td>Man runs and then stops in park, camera follows</td>
</tr>
<tr>
<td>Shields</td>
<td>0m10s</td>
<td>Man points out detailed shields, camera follows, then zoom in on one</td>
</tr>
<tr>
<td>Stockholm</td>
<td>0m10s</td>
<td>Pan of Stockholm, with vehicles moving, detail</td>
</tr>
<tr>
<td>Sintel</td>
<td>0m52s</td>
<td>Trailer for 3D animated film with fade to black, title cards</td>
</tr>
<tr>
<td>Old Town Cross</td>
<td>0m10s</td>
<td>Helicopter moving capture of a city, vehicles moving</td>
</tr>
<tr>
<td>Ducks Take Off</td>
<td>0m10s</td>
<td>Very blue pond, where sitting ducks fly away, leaving behind detailed ripples in the water</td>
</tr>
<tr>
<td>In to Tree</td>
<td>0m10s</td>
<td>Low flying shot of house, then a zoom into a tree</td>
</tr>
</tbody>
</table>
Table 3.5: Source Videos at 1920 × 1080 Used and their Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Sky</td>
<td>0m10s</td>
<td>Pan of a blue sky and trees, detail</td>
</tr>
<tr>
<td>Sintel</td>
<td>0m52s</td>
<td>Trailer for 3D animated film with fade to black, title cards</td>
</tr>
<tr>
<td>Touchdown</td>
<td>0m19s</td>
<td>American football, fast movement, fast and shaky camera movement</td>
</tr>
<tr>
<td>Tractor</td>
<td>0m27s</td>
<td>Tractor tilling field, closeup, then zooms out to entire field</td>
</tr>
<tr>
<td>Speed Bag</td>
<td>0m19s</td>
<td>Man using a speed bag, talks to the camera, cuts, high speed movement</td>
</tr>
<tr>
<td>Snow Mountain</td>
<td>0m19s</td>
<td>Static shot of mountains, fade to pan, fade to static shot, includes smoke off of white mountains</td>
</tr>
<tr>
<td>West Wind</td>
<td>0m19s</td>
<td>Scrolling text on black background on left, grain swaying on right side, simulates credits</td>
</tr>
<tr>
<td>Riverbed</td>
<td>0m10s</td>
<td>Closeup of rocks below clear river, water ripples</td>
</tr>
<tr>
<td>Pedestrian Area</td>
<td>0m15s</td>
<td>Slowly moving camera, capturing fast moving pedestrians and bicycles, moving in and out of frame</td>
</tr>
<tr>
<td>Rush Hour</td>
<td>0m20s</td>
<td>Closeup of cars at rush hour, pedestrians and fountain in background, slow camera tilt</td>
</tr>
</tbody>
</table>
3.3 Statistical Significance

In order to test whether or not there are statistically significant differences between two settings of an encoder or between two different encoders, $t$-tests are used to test for the significance of the data. This allows us to determine if the differences are within a reasonable error, and may be equivalent, or are very unlikely to have occurred by chance and are likely different by their nature.

For us to do so, we needed several important statistics from the data. For each frame in a source video, the PSNR and SSIM were calculated and logged. The values were sorted, and the minimum, 1st quartile, median, 3rd quartile, maximum, and mean values for both the PSNR and SSIM were found. The variance was found via Eq. 3.3.1. Taking the square root of the variance gives us the standard deviation, $\sigma$, in units of the metric being used. The standard deviation is used as a marker for the degree of error. In various plots in this document, the error bars correspond to the standard deviation.

$$\sigma^2 = E [(X - \mu)^2]$$  \hspace{1cm} (3.3.1)

After gathering our statistics, the test data was gathered together. Using either a Student’s $t$-test for equal variance data or a Welch’s $t$-test for unequal variance data, an independent two-sample two-tailed $t$-test. The test statistic, $t_s$ for a Student’s $t$-test is shown in Eq. 3.3.2, or $t_w$ for a Welch’s $t$-test is shown in Eq. 3.3.3. $S_{1,2}$ is the pooled standard deviation. The $s^2$ variables are the unbiased estimators of the variance in the sample. As our sample sizes will almost always be equal, $n_1 = n_2$.

To test for whether or not two variances are equal, the standard deviations are tested for statistical significance via Welch’s $t$-test. If the p-value was lower than the threshold for significance, then the variances are considered equal and Student’s $t$-test is used. Otherwise, Welch’s $t$-test is used.
\[ t_s = \frac{\bar{X}_1 - \bar{X}_2}{\sigma_{1,2} \cdot \sqrt{\frac{2}{n}}} \]  

(3.3.2)

\[ \sigma_{1,2} = \sqrt{\frac{1}{2} \left( \sigma_1^2 + \sigma_2^2 \right)} \]

\[ t_w = \frac{\bar{X}_1 - \bar{X}_2}{\sigma_{X_1-X_2}} \]  

(3.3.3)

\[ \sigma_{X_1-X_2} = \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \]

3.3.1 Choosing the \( \alpha \)-level

The choice of the statistical significance threshold, \( \alpha \) is an important decision when statistical testing. The chosen value for alpha is 0.1.

This decision was made to increase the statistical power of our \( t \)-tests, which will reject the null hypothesis with higher frequency. As the signal-to-noise ratio of the measurements – which themselves are signal-to-noise ratios – is the primary factor, a value above 0.1 would be too high. This ensures that there is less than a 10% chance that the result would have occurred by accident.

This value is mitigated by the high number of tests and the other statistical work, ensuring that our confidence is at a reasonable level while increasing our statistical power.
3.4 Initial Testing and Capabilities

To establish a baseline comparison between VP8 and H.264, some initial tests were performed that used the default configurations of JM, vpxenc and x264. As H.264 is not one encoding but rather a grouping of several different profiles, both H.264 Baseline and H.264 High Profile were compared with VP8 at different levels of effort.

Table 3.6 shows the settings for each of the encoders. x264 is shown as H.264, because it’s likely more representative of H.264 in public use than the JM Reference software is. It is important to use x264’s --no psy setting when testing quality metrics to ensure that x264 does not use psychovisual optimizations that decrease the resulting metric, but increase perceptive quality. For both VP8 and H.264, special parameters that tell the encoder to optimize for PSNR or SSIM were not used. This was done to ensure a fair comparison, rather than to see who can game the measurement algorithm. Most encoded videos would not tune for these parameters, and as such, that would render them inaccurate for real world analysis.

For the JM Reference encoder, the configurations in Table 3.6 reflect only differences from the default encoder.cfg configuration file, without necessary changes. When changing the defaults to work with Baseline, several options need to be changed to reflect restrictions that the Baseline profile imposes. These changes are not reflected in Table 3.6. In addition, for both JM Baseline and JM High configurations, rate control is enabled, and the bitrate is set to 150 kbps.

3.4.1 Feature Set Testing Video Results

From the results from the tests described in Section 3.4, it was obvious that VP8’s performance lies between H.264 Baseline and H.264 High profile. Four H.264 parameter sets were created, and are described in Table 3.7. They were compared against VP8’s Good
Table 3.6: Initial Testing Encoder Parameters

<table>
<thead>
<tr>
<th>Encoder Profile</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 Baseline</td>
<td>--profile baseline --bitrate 150 --ratetol 1.0 --no-psy</td>
</tr>
<tr>
<td>H.264 High</td>
<td>--profile high --bitrate 150 --ratetol 1.0 --no-psy</td>
</tr>
<tr>
<td>VP8 Good</td>
<td>--good --target-bitrate=150</td>
</tr>
<tr>
<td>VP8 Best</td>
<td>--best --target-bitrate=150</td>
</tr>
<tr>
<td>JM Baseline</td>
<td>ProfileIDC = 66, NumberBFrames 0</td>
</tr>
<tr>
<td>JM High</td>
<td>ProfileIDC = 100, NumberBFrames 7, WeightedPrediction = 1, WeightedBiprediction = 1</td>
</tr>
</tbody>
</table>

Deadline, as, not only is it easier to compare four variables against one rather than two fixed results, but VP8’s Best Deadline doesn’t have a significant improvement in terms of performance.

Table 3.7: Feature Set Testing Encoder Parameters

<table>
<thead>
<tr>
<th>Encoder Profile</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 Baseline</td>
<td>--profile baseline -B 575 --ratetol 1.0 --no-psy</td>
</tr>
<tr>
<td>H.264 BP+CABAC</td>
<td>--no-8x8dct --bfFrames 0 -B 575 --cqm flat --weightp 0 --no-psy</td>
</tr>
<tr>
<td>H.264 Main</td>
<td>--profile main -B 575 --ratetol 1.0 --no-psy</td>
</tr>
<tr>
<td>H.264 High</td>
<td>--profile high -B 575 --ratetol 1.0 --no-psy</td>
</tr>
<tr>
<td>VP8 Good</td>
<td>--good --target-bitrate=575</td>
</tr>
</tbody>
</table>

This was done for two resolutions, 4CIF, which represents medium quality videos, and 720p, which represents high definition video for films and television. Both results are described in Section 4.1.2.

3.4.2 Rate-Distortion Curves

Several rate-distortion (RD) curves were generated to analyze the bitrate versus quality tradeoff for both H.264 and VP8. As the objective is to find the curves most representative of H.264 and VP8, the minimum required bitrate for each was found iteratively. Both x264 and vpxenc will output a higher than nominal bitrate if it cannot find a way to
reach the nominal bitrate with the source video. After finding this minimum bitrate, the maximum bitrate was found by means of diminishing returns. If the increased bitrate failed to generate an improvement in the quality, then this was the maximum bitrate that would be necessary for demonstrating the RD curve.

The results for the RD curves are shown in Section 4.1.3.
3.5 Use Case Scenarios

There are a litany of ways in which video codecs are used, from professional broadcasting, to video conferencing, and for web videos. Several use cases were studied in detail to recommend a specific video encoder and settings for each. Each use case has different requirements, and their respective methodology section will elucidate their requirements and how best to measure the results.

3.5.1 Video Conferencing

Video conferencing is an important aspect of the modern world. Smartphones and personal computers are now being shipped with the ability to video chat with other people, and the challenges involved are unique. Due to the latency requirement, only a small or non-existent video buffer can be used, so inter-prediction is hampered at best. Instead, fast encoding and decoding is preferred. Small artifacts can be tolerated, as network conditions are dynamic, but the bitrate should remain nearly constant, and not vary too widely. Bitrate control was used to control the file size and to limit the maximum sporadic bitrate in addition to the average bitrate.

For the encoders, six speed levels were used. The “slowest” setting is the default for both x264 and vpxenc. Each step towards speed turns off expensive operations and optimizations. At level 4 and above for VP8 turns off RD optimization, which lowers the predictability and increases the size of the output bitrate. At values of “superfast” and higher for x264, subpixel motion estimation is significantly simpler and does not use RD optimization. Static thresholding for VP8 was turned on, as this can improve low movement scenarios for video conferencing.

The encoding times for all video conferencing tests were not analyzed due to the natural variation in encoding times and the lack of exclusive use of the encoding machine,
which could have distorted the timing results. In addition, VP8 is not expected to be as optimized as x264 due to its recent release. In future work, this will be an important metric to measure in a controlled setting.

Table 3.8: Video Conferencing Encoder Parameters, from Slowest Encoding Speed to Fastest

<table>
<thead>
<tr>
<th>Encoder Profile</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 Slowest</td>
<td>--preset medium --tune zerolatency --no-psy</td>
</tr>
<tr>
<td>H.264 Slower</td>
<td>--preset fast --tune zerolatency --no-psy</td>
</tr>
<tr>
<td>H.264 Slow</td>
<td>--preset faster --tune zerolatency --no-psy</td>
</tr>
<tr>
<td>H.264 Fast</td>
<td>--preset veryfast --tune zerolatency --no-psy</td>
</tr>
<tr>
<td>H.264 Faster</td>
<td>--preset superfast --tune zerolatency --no-psy</td>
</tr>
<tr>
<td>H.264 Fastest</td>
<td>--preset ultrafast --tune zerolatency --no-psy</td>
</tr>
<tr>
<td>VP8 Slowest</td>
<td>--good --cpu-used=0 --static-thresh=100 --auto-alt-ref=1 --lag-in-frames=0 --kf-max-dist=1 --drop-frame=0</td>
</tr>
<tr>
<td>VP8 Slower</td>
<td>--good --cpu-used=1 --static-thresh=100 --auto-alt-ref=1 --lag-in-frames=0 --kf-max-dist=1 --drop-frame=0</td>
</tr>
<tr>
<td>VP8 Slow</td>
<td>--good --cpu-used=2 --static-thresh=100 --auto-alt-ref=1 --lag-in-frames=0 --kf-max-dist=1 --drop-frame=0</td>
</tr>
<tr>
<td>VP8 Fast</td>
<td>--good --cpu-used=3 --static-thresh=100 --auto-alt-ref=1 --lag-in-frames=0 --kf-max-dist=1 --drop-frame=0</td>
</tr>
<tr>
<td>VP8 Faster</td>
<td>--good --cpu-used=4 --static-thresh=100 --auto-alt-ref=1 --lag-in-frames=0 --kf-max-dist=1 --drop-frame=0</td>
</tr>
<tr>
<td>VP8 Fastest</td>
<td>--good --cpu-used=5 --static-thresh=100 --auto-alt-ref=1 --lag-in-frames=0 --kf-max-dist=1 --drop-frame=0</td>
</tr>
</tbody>
</table>

As there are several different requirements for video conferencing, from ultra-low bitrate video conferencing at resolutions of CIF and less, all the way to real-time 1080p streaming, several tests were run. The first test is Akiyo at CIF resolution. As described in Table 3.2, a female newscaster reads a piece of paper with minimal movement. It mimics low movement video conferencing. It was encoded at two bitrates, 25 Kbps and 50 Kbps. The results of this test are discussed in Section 4.2.1.

For this test, no restrictions such as key frame placement were used. Another video was used to test the video conferencing behavior.

The results for this use case are discussed in Section 4.2.1.
3.5.2 High Definition Television and Film Broadcast

High definition video is an area that video codecs must perform well in to be accepted for mainstream use, due to the increasing use of HD resolutions in the home and on the web. Smartphones that can record video in 1080p are now becoming available.

Film and television shows have some important characteristics that tests for them must capture. They contain multiple scenes that may have completely different detail and movement traits. Usually, they have credits at the end, sometimes with additional video mixed with the credits. There are also animated movies, which have smoother gradients of color and less detail.

To create these scenes, several source videos were concatenated of different types. Table 3.9 was created for the 720p source videos. The first video, City Outdoors, is comprised of source videos that happen in the city and outdoors. The two city scenes are similar, but the cut to ducks taking off is abrupt and very different. The second video, Detail Outdoors, consists of three videos with high amounts of detail, and leads to another outdoor scene with much detail.

Table 3.9: Source Videos for 720p Composite Films

<table>
<thead>
<tr>
<th>City Outdoors</th>
<th>Detail Outdoors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Town Cross</td>
<td>Shields</td>
</tr>
<tr>
<td>Stockholm</td>
<td>MobCal</td>
</tr>
<tr>
<td>Ducks Take Off</td>
<td>Parkrun</td>
</tr>
<tr>
<td>In to tree</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.10 lists two videos created from 1080p source videos. The first video, Sports Film, has two sports clips, an American football clip, followed by a closeup of a boxer using a speed bag. An abrupt transition to an outdoor clip of a mountain, back to sports with the start of a race. Sports Film finishes with a credits simulation with grain blowing in the wind.

Outdoor Pedestrians is different in content and encoding difficulty. It starts outdoors
with a pan of the sky, followed by a farm and a riverbed. It then switches to a busy sidewalk and then a busy street. This transition will test the encoder’s ability to adapt to widely varying difficulty.

Table 3.10: Source Videos for 1080p Composite Films

<table>
<thead>
<tr>
<th>Sports Film</th>
<th>Outdoors Pedestrians</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touchdown</td>
<td>BlueSky</td>
</tr>
<tr>
<td>Speed Bag</td>
<td>Tractor</td>
</tr>
<tr>
<td>Snow Mountain</td>
<td>Riverbed</td>
</tr>
<tr>
<td>West Wind</td>
<td>Pedestrian Area</td>
</tr>
<tr>
<td></td>
<td>Rush Hour</td>
</tr>
</tbody>
</table>

The trailer for the animated film Sintel is also encoded to test how well a long animated clip fairs for encoding. No special tuning is given to animated clips, such as psychovisual optimizations that benefit animated film.

To compare the encoded videos, vpxenc was used in two-pass mode. It was given a target bitrate, and constraints on how to meet that bitrate. However, it rarely is accurate, and often encodes at approximately 75% of the given bitrate. After encoding this, x264 was used in two-pass mode to encode at the average bitrate that was reported by Mediainfo on the VP8 video. As x264’s rate control algorithm is very mature, it can very often meets this bitrate target exactly. This way, both videos have the same amount of resources available to them (bitrate), and the quality will be the only varying factor.

Two-pass was used to ensure accurate bitrate control. Often, when preparing encoded videos for distribution, there is sufficient time for two-pass encodings of television, films, and other long programs. High bitrates were used to maximize quality, as is desired for such programs. Lower bitrate tests of HD media was performed in Section 3.4.

After the results from these various encodings were collected, the PSNR and SSIM values were matched, and the differences in bitrates shown graphically. The results for this section are seen in Section 4.2.2.
3.6 Intra-coding Tests

Intra-coding is an important aspect of video coding. In order to examine the prowess of the intra-prediction of H.264 and VP8, the max number of predicted frames in each encoder was set to zero. This ensured that every encoded frame was not inter-predicted, and was a key frame. With this constraint, the quantizer was used as the control variable. In this way, the quality can be controlled where it is most relevant: the lossy intra-coding.

A rate-distortion curve was generated where the constraint was a constant quantizer value. In this way, the same amount of data is discarded by both encoders, rather than shooting for a particular bitrate. Currently, vpxenc takes this quantizer value as a best-effort suggestion, rather than a hard limit. By comparing video bitrates and quality metrics directly, this issue can be avoided.

Three sample videos at three different resolutions, CIF, 4CIF, and 720p, were chosen as they represent well many video coding challenges. Foreman was chosen as the CIF video due to its high use of color in an outdoor scene. Crew was chosen as the 4CIF video because of the sharp color contrasts and the use of camera flashes. Parkrun was chosen for its detail and potential for intra-prediction efficiency.

Table 3.11 lists the encoder parameters chosen for x264 and VP8. All encoders use only intraframes, specified as --I 1 for x264 and --kf-max-dist=1 for VP8, and all encoders are restricted to looking at the current frame only – not an issue for intra-coding in general, as intraframes only use the information in.

To use a constant quantizer for H.264, x264’s --qp option was used to force a constant quantizer parameter. As usual, x264’s psycho-visualization optimizations are turned off, as they can harm metrics.

For VP8, it has been recommended by the VP8 developers to use --drop-frame=0 to ensure that temporal resampling does not occur when using a constant quantizer. To
Table 3.11: Intra-coding Test Encoder Profiles and Parameters

<table>
<thead>
<tr>
<th>Encoder Profile</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 Baseline</td>
<td>--profile baseline --tune zerolatency -I 1 --no-psy</td>
</tr>
<tr>
<td>H.264 High</td>
<td>--profile high --tune zerolatency -I 1 --no-psy</td>
</tr>
<tr>
<td>VP8 Good</td>
<td>--good --cpu-used=0 --static-thresh=100 --auto-alt-ref=1 --lag-in-frames=0 --kf-max-dist=1 --drop-frame=0</td>
</tr>
<tr>
<td>VP8 Best</td>
<td>--best --cpu-used=0 --static-thresh=100 --auto-alt-ref=1 --lag-in-frames=0 --kf-max-dist=1 --drop-frame=0</td>
</tr>
</tbody>
</table>

use a constant quantizer in VP8, the options --min-q and --max-q are set to the same value.

The results for intra-coding are discussed in Section 4.3.
3.7 Inter-coding Tests

As discussed in Section 2.2.1, H.264 and VP8 have significant differences in their use of predictive frames. VP8 has no bidirectional prediction frames, and instead augments its basic intra and predictive frame scheme with golden frames and alternate reference (altref) frames.

By varying the strength of B frames in H.264 and how often altref frames are encoded, it is possible to compare the degree of effect that they have on the resulting video stream. VP8 is still in active development as of this writing, and the strength of their inter-coding varies as development continues. As such, the version of libvpx used is not the stable version described in Table 3.1, but rather the latest version as of this writing\(^2\).

To test the inter-coding capabilities of H.264 and VP8, the configurations in Table 3.12 were created. These vary from P-frame only (configurations 1 and 8) to the maximum amount of specialty frames (configurations 6, 7, 9 and 10). The configurations were designed to provide a wide range of

For x264, there are many options to control the frequency of placing B frames. The option \(-b\) controls the maximum number of B frames x264 can place in a row. The default value is 3, but many video encoding recommendations suggest 16 or higher. The option \(--b\text{-adapt}\) chooses the method by which x264 chooses B frames over P frames. A value of 0 tells x264 to place only B frames. A value of 1, the default, tells x264 to use a fast approximation algorithm to decide to use B frames only when it benefits the rate-distortion curve. The maximum value of 2 tells x264 to use an optimal, slow algorithm to choose when to use B frames.

The option \(--b\text{-bias}\) tells the algorithm chosen by \(--b\text{-adapt}\) how to favor B frames over P frames. A value of 0 weights them equally. Values below 0, to the minimum of

\(^2\)git version ff356497587cea9a2ce05c7714088662cdec6a70, Mon Aug 15 09:28:41 2011
Table 3.12: Configurations for Inter-coding Tests

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x264-default</td>
<td>--profile high -I 120</td>
<td>Defaults for every x264 configuration</td>
</tr>
<tr>
<td>vp8-default</td>
<td>--good --kf-max-dist=120 --passes=2</td>
<td>Defaults for every vpxenc configuration</td>
</tr>
<tr>
<td>x264 (1)</td>
<td>-b 0 -r 1 --b-adapt 1 --b-bias 0</td>
<td>VP8-like, no B frames, only one ref</td>
</tr>
<tr>
<td>x264 (2)</td>
<td>-b 3 -r 3 --b-adapt 1 --b-bias 0</td>
<td>x264 default</td>
</tr>
<tr>
<td>x264 (3)</td>
<td>-b 16 -r 3 --b-adapt 2 --b-bias -50</td>
<td>Many B frames possible, but weight B frames low</td>
</tr>
<tr>
<td>x264 (4)</td>
<td>-b 16 -r 3 --b-adapt 2 --b-bias 0</td>
<td>Many B frames possible, but same weights</td>
</tr>
<tr>
<td>x264 (5)</td>
<td>-b 16 -r 3 --b-adapt 2 --b-bias 100</td>
<td>Many B frames possible, heavily weight B frames</td>
</tr>
<tr>
<td>x264 (6)</td>
<td>-b 119 -r 3 --b-adapt 0 --b-bias 0</td>
<td>Use only B frames</td>
</tr>
<tr>
<td>x264 (7)</td>
<td>-b 119 -r 9 --b-adapt 0 --b-bias 0</td>
<td>Use only B frames, high number of refs</td>
</tr>
<tr>
<td>vp8 (8)</td>
<td>--auto-alt-ref=0 --arnr-maxframes=0 --arnr-strength=0 --arnr-type=0</td>
<td>VP8 default</td>
</tr>
<tr>
<td>vp8 (9)</td>
<td>--auto-alt-ref=1 --arnr-maxframes=4 --arnr-strength=2 --arnr-type=3</td>
<td>Alternate reference frames enabled</td>
</tr>
<tr>
<td>vp8 (10)</td>
<td>--auto-alt-ref=1 --arnr-maxframes=7 --arnr-strength=6 --arnr-type=3</td>
<td>Altref enable, ARNR filter high</td>
</tr>
</tbody>
</table>

−100, weight P frames more highly than B frames. Values above 0, to the maximum of 100, weight B frames over P frames.

The number of reference frames, that is, the number of frames that a B or P frame can reference, is controlled by −r. VP8 can only ever reference one frame from its P frames, but its altref frames can reference any number of frames. For test configuration 1, in Table 3.12, the number of reference frames is set to one. This is somewhat worse than VP8, as it cannot reference special P frames (golden frames in VP8), but only the most recent I or P frame.

For this test, two pass mode is used. VP8 configuration 8 is encoded first, at a nominal bitrate appropriate for the resolution. Then, all x264 configurations use the actual average
bitrate utilized by VP8, just as in Section 3.5.2. This way, each setting consumes the same amount of bandwidth.

These will be run on the 4CIF resolution video Ice and the 720p video Parkrun. These two videos are difficult to encode, and benefit from advanced inter-prediction such as B frames. If videos were chosen that would benefit little from B frames, then the differences would not be as pronounced.
3.8 Loop Filter Efficiency Tests

The loop, or deblocking filter is a vital component to macroblock-based orthogonal transformed video or still image codecs. As each image is subdivided, the boundaries that form between the divisions are enhanced, and can lead to blocky, unrealistic results.

The in-loop deblocking filter was an important addition to H.264, and immensely improved the visual performance of H.264 by not requiring the filter to be run outside of the encoding/decoding loop.

To test the relative acuity of VP8 and H.264’s loop filters, videos that were CIF, 4CIF, 720p, and 1080p resolution were encoded at four bitrates: 150 kbps, 600 kbps, 1000 kbps, and 2000 kbps, respectively. Each video was encoded twice. The first time the videos were encoded normally, and the second time the videos were encoded without the use of their loop filter.

For VP8, there is no explicit option to disable the loop filter. However, as VP8 has two loop filters, as described in Section 2.1.7, the simple loop filter at minimum power was used. This simulates a very mild loop filter. This has a side-effect, however, in that this forces the encoder to use only bi-linear sub-pixel filtering. Using bi-linear sub-pixel filtering will reduce the objective quality of the result, on approximately the same order that a minimal loop filter will improve the output over no loop filter. The simplified loop filter was also set to 7, which is the weakest setting.

Table 3.13 shows the configurations used for the loop filter efficiency test. Each encoder was used in two pass mode to facilitate maximum quality for the bitrate given. The absolute quality metrics for each result are immaterial for this test, as the largest differential in quality will be indicative of the most effective loop filter.

The values of PSNR were compared by the percent difference formula, Eq. 3.8.1. The variable \( m \) stands for measured value, such as PSNR or SSIM, and \( m_f \) indicates the metric
using the loop filter, while $m_n$ is the non-filtered version. The percent difference indicates the percentage by which PSNR or SSIM is improved by adding the loop filter.

$$ p_{diff} = 100 \frac{m_f - m_n}{\frac{1}{2} (m_f + m_n)} $$  \hspace{1cm} (3.8.1) 

Averaging of the percent differences, rather than using the median value, was used because of its sensitivity to outliers. Outliers can highlight a particular edge case where the loop filter made a particularly large difference, and hence its important to include it. In addition, due to the relative small sample size of videos, the median is likely not to be as accurate as the mean.

This chapter covered the tests and the reasoning as to how we configured VP8 and H.264, with their implementation details, and how we measured the results. In the next chapter, we introduce our results along with discussion as to what the graphs and tables indicate.

The results are organized in the same order that the tests were introduced in this chapter.
Chapter 4

Results

4.1 Initial Testing Results

To paint the picture of VP8’s competitiveness with H.264 with a broad stroke, videos were encoded at two different profiles for each of the three encoders: H.264 has Baseline profile and High profile, using both JM and x264, and VP8 has a Good quality deadline and a Best quality deadline.

4.1.1 CIF Resolution Video Results

The PSNR summary for the six CIF resolution videos, encoded with the initial testing parameters described in Table 3.6, H.264 Baseline and VP8 Good, is shown in Figure 4.1. This, and similar figures, are box-and-whisker plots of the associated frame data: the lowest part of the whisker is equivalent to the minimum PSNR value for that video, the bottom of the box is the 1st percentile PSNR, the horizontal line in the box is the median, the point (a plus sign or an asterisk) is the mean, the top of the box is the 3rd percentile, and the top of the whisker is the maximum PSNR value. The SSIM summary is shown in Figure 4.2.
VP8 seems to outperform H.264 Baseline (both x264 and JM) for these six videos in terms of PSNR. Videos with a large amount of detail, such as Mobile and Bridge Close, have the largest difference between VP8 and H.264 Baseline. Videos that have less temporal and spacial complexity, such as Akiyo and Students, have less of an obvious benefit. Videos that have an average amount of detail and camera movement, such as Coastguard and Foreman, VP8 outperforms H.264 Baseline, but not by such a large degree. In all cases, VP8 uses less bandwidth to achieve these benefits.

For SSIM, H.264 Baseline still does worse in every case except for Students, which H.264 has a slight edge. If you consider the minimum SSIM value for H.264, which is significantly lower than VP8’s minimum SSIM for Students, then the comparison is more evenly matched. VP8 still outperforms when a large amount of detail is used, as in Bridge Close and Mobile. This is likely due to high performing motion prediction and a high
compression entropy coder, rather than a benefit in the intra coder, a topic which is explored in more detail in Section 3.6 and 4.3.

Figure 4.2: SSIM Summary of Videos at CIF, 150 Kbps, H.264 Baseline and VP8 Good Deadline

For H.264 High profile with VP8 Best, Figure 4.3 was generated from PSNR, and Figure 4.4 was generated from SSIM results. H.264 High profile performs substantially better than H.264 Baseline, and outperforms VP8 on every video. VP8 almost does as well as H.264 High on Foreman, and the bitrate savings are significant. The SSIM results are even more dramatic than the PSNR results, and Foreman is still the most comparable of the videos, but H.264 outperforms VP8. The performance boost is likely to due to the addition of B-frames, the use of CABAC instead of CAVLC, and using weighted P-frames.
Figure 4.3: PSNR Summary of Videos at CIF, 150 Kbps, H.264 High Profile and VP8 Best Deadline

Figure 4.4: SSIM Summary of Videos at CIF, 150 Kbps, H.264 High Profile and VP8 Best Deadline
4.1.2 Feature Set Testing Video Results

To test which feature set of H.264 was most similar to VP8 in terms of quality and bitrate used, five 4CIF resolution videos were encoded at four different H.264 settings, as described in Table 3.7.

Figure 4.5, on page 88 shows the four settings of H.264 versus VP8. VP8’s box-and-whisker plot, at the rightmost of every section, can be compared with H.264 Baseline with CABAC to slightly above H.264 Main, which differs from H.264 High primarily in the fact that it does not use the option of using the $8 \times 8$ DCT in adaptive I-frames. VP8 does not use $8 \times 8$ DCT transforms at all.

The video that VP8 performed best on, Crew, originally had a 638 Kbps bitrate. This was high enough to harm comparison with the other videos, as it performed better, but also had a high bitrate. This video was re-encoded with a nominal bitrate of 490 Kbps, and output 583 Kbps. At 475 Kbps and below, VP8 produces output within 10 % of its nominal bitrate, unlike 490 Kbps nominal. While giving VP8 special attention is generally undesired, this is likely an encoder bug rather than an inherent flaw in VP8, and the results are still positive.

VP8, from this example, ranges in quality depending on the source video. It is worse at higher resolutions than the lower resolution tests in Section 4.1.1, but still maintains quality parity with H.264 up to H.264 Main.

For HD videos, Figure 4.6, on page 89 was generated from five HD videos. These videos, at $1280 \times 720$ resolution, were encoded at a nominal bitrate of 1000 kbps. This makes some videos, like Sintel and Parkrun encode at lower bitrates due to their relative ease of encoding when compared to more complicated videos such as MobCal or Stockholm. This represents different effort levels for the encoders, as the must lower the nominal bitrate when it is not necessary, and encode in high quality when the source is challenging.
From these results, it appears that VP8 is most similar to H.264 Baseline with CABAC, but it also has a substantially lower bitrate in most source videos for the same quality. This is a bias in the encoder, not something that is controllable by the VP8 bitstream guide. For Parkrun, VP8 does poorly, requiring a high bitrate to encode a video between BP+CABAC and Baseline H.264. It does not perform very well in any of the other tests, but does not perform as poorly as Parkrun, either.

For Parkrun, two captures are shown in Figure 4.7 on page 90. This shows the man with the patterned umbrella stopped after running. In Figure 4.7a, the pattern is shown, along with a close approximation of the original coloring. VP8’s Figure 4.7b, however, shows a grey blur instead of the patterned umbrella, capturing neither color nor detail. The remainder of the clip is similar in quality to these two clips. What should be noted is that H.264 High profile did not maintain the umbrella’s detail for the length of the clip; rather, it was blurry until the man stopped, where x264 noticed that increased detail was possible, and encoded it. VP8’s vpxenc made no such determination.

It’s obvious why H.264 High profile did so well in Figure 4.6’s SSIM metric: it has superior motion prediction (especially in chroma) and finer retained detail.

4.1.3 Rate Distortion Curve Results

Figure 4.8 shows the RD curve for the Foreman video. SSIM results, using the right Y-axis, is shown above the PSNR results, which uses the left Y-Axis. Table 4.1 shows the p-values for Foreman, computed using a Student’s $t$-test if the variances were statistically equal or a Welch’s $t$-test otherwise. No value met the threshold for significance of $\alpha = 0.1$, and thus the RD curves for Foreman are statistically significant.

However, it’s obvious that, on the average H.264 Baseline performs worse than VP8 and H.264 High profile. There doesn’t seem to be a significant difference between the two VP8 encodings and H.264.
Table 4.1: p-values for RD Curve for Foreman

<table>
<thead>
<tr>
<th>Encoder</th>
<th>PSNR p-value</th>
<th>SSIM p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline/Good</td>
<td>0.45291</td>
<td>0.56182</td>
</tr>
<tr>
<td>High/Best</td>
<td>0.90875</td>
<td>0.79656</td>
</tr>
</tbody>
</table>

Figure 4.9 shows the RD curve for the Soccer video. SSIM results, using the right Y-axis, is shown above the PSNR results, which uses the left Y-Axis. Table 4.2 shows the p-values for Soccer, computed using a Student’s \( t \)-test. No value met the threshold for significance of \( \alpha = 0.1 \), and thus the RD curves for Soccer are statistically significant. However, the p-value for Soccer for SSIM on the Baseline and Good comparison is very near the statistical significance level. This may become significant at higher resolutions.

However, it’s obvious that, on the average H.264 Baseline performs worse than VP8 and H.264 High profile. There doesn’t seem to be a significant difference between the two VP8 encodings and H.264. There is little to no difference between VP8 Good Deadline and VP8 Best Deadline.

Table 4.2: p-values for RD Curve for Soccer

<table>
<thead>
<tr>
<th>Encoder</th>
<th>PSNR p-value</th>
<th>SSIM p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline/Good</td>
<td>0.24214</td>
<td>0.11221</td>
</tr>
<tr>
<td>High/Best</td>
<td>0.29373</td>
<td>0.48727</td>
</tr>
</tbody>
</table>

Figure 4.10 shows the RD curve results for Stockholm at \( 1280 \times 720 \) pixel resolution. At this resolution and content, H.264 High profile performs substantially better than VP8 or H.264 Baseline. Table 4.3 lists the p-values for the \( t \)-tests performed on the RD curve. Values in square brackets indicate statistical significance, using \( \alpha = 0.1 \). This indicates that the benefit of H.264 high profile is statistically significant compared to VP8.

VP8, however isn’t significantly different than H.264 Baseline, even though the averages seen in Figure 4.10 seem to indicate that. This indicates that H.264 High is statistically different then H.264 Baseline. At this resolution, coding efficiency becomes the utmost of
importance due to the volume of data.

Table 4.3: p-values for RD Curve for Stockholm

<table>
<thead>
<tr>
<th>Encoder</th>
<th>PSNR p-value</th>
<th>SSIM p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline/Good</td>
<td>0.71311</td>
<td>0.45746</td>
</tr>
<tr>
<td>High/Best</td>
<td>$5.8705 \times 10^{-6}$</td>
<td>0.00039</td>
</tr>
</tbody>
</table>
Figure 4.5: Summary of Four H.264 Profiles vs. VP8 on 4CIF Videos at 575 Kbps
Figure 4.6: Summary of Four H.264 Profiles vs. VP8 on 720p Videos at 1000 Kbps
Figure 4.7: Parkrun at 720p, encoded at 1000 Kbps

Figure 4.8: Foreman CIF Video RD Curve for H.264 and VP8
Figure 4.9: Soccer 4CIF Video RD Curve for H.264 and VP8

Figure 4.10: Stockholm 720p Video RD Curve for H.264 and VP8
4.2 Use Case Scenario Results

4.2.1 Video Conferencing Result

Akiyo

The video conferencing encoding test described in Section 3.5.1 was used to encode the Akiyo video, at CIF resolution, at 25 Kbps and 50 Kbps. Figure 4.11 summarizes the six encoding profiles for H.264 and VP8.

It is a box and whisker plot, where the lowest part of the whisker indicates the minimum PSNR value for the entire sequence, the lower edge of the box indicates the 1st quartile, the line in the box represents the median, the point is the mean, the top edge of the box is the 3rd quartile, and the top of the whisker is the maximum PSNR value. The bitrate is also plotted as a line, as it is nominally a horizontal constant line of the requested bitrate.

For Figure 4.11, VP8 outperforms H.264 for every value of encoding complexity, often with VP8’s minimum value being nearly the 3rd quartile result for H.264. H.264 also has a higher bitrate requirement for a lower quality. It is also shown that H.264’s fastest setting significantly reduces the encoding quality, while there is only a small difference between VP8’s various encoding settings.

Figure 4.12 shows the PSNR values for each frame for the slowest and fastest encoders for H.264 and VP8. VP8 starts strong, while H.264’s quality increases as the length of the sequence improves. VP8’s fastest setting beats H.264’s until H.264 has a key frame at frame 251, represented by the vertical line in the plot. VP8 encoded no key frames for the length of the sequence, other than the first initial key frame. If this sequence was longer, as a video conference would likely be, H.264 would probably recover enough to offset the poor initial performance.
Figure 4.11: Summary of Akiyo at CIF Resolution, 25 Kbps

Figure 4.12: Frames of Akiyo at CIF Resolution, 25 Kbps
H.264 does better when 50 Kbps is used for Akiyo. Figure 4.13 shows the summary for Akiyo encoded at 50 Kbps using the six encoder profiles for H.264 and VP8. H.264’s bitrate is constant at 50 Kbps, while VP8’s varies, and is generally 10 to 15 Kbps lower, while providing a higher PNSR.

For the 50 Kbps frame analysis, shown in Figure 4.14, H.264’s slowest performs better, keeping up with VP8, and eventually catching up after its key frame at 251. VP8 again does not encode a key frame other than the initial one. As this is a configurable setting, it would depend on the requirements of the network as to how many key frames should be encoded.

![Figure 4.13: Summary of Akiyo at CIF Resolution, 50 Kbps](image)

The p-values for the PSNR values for the population of frames were collected in Table 4.4 and Table 4.5. The null hypothesis, that the values are the same, was rejected for each case, using a Welch’s t-test at a significance level of $\alpha = 0.1$. A two-tailed Welch’s t-test was used because the variances for the sample data were significantly different. Each
Figure 4.14: Frames of Akiyo at CIF Resolution, 50 Kbps

profile was tested against each other profile, so there is a symmetry along the diagonals of Tables 4.4 and 4.5.

This indicates that the trend noticed in Figures 4.12 and 4.14 is significant, and that VP8 outperforms H.264 in these video conferencing cases. This is a low bandwidth, low resolution, low movement case, however, and isn’t representative of every video conferencing scenario.

Table 4.4: p-values for Frames for the Fastest and Slowest Profiles for Akiyo at 25 Kbps

<table>
<thead>
<tr>
<th></th>
<th>H.264 Slowest</th>
<th>VP8 Slowest</th>
<th>H.264 Fastest</th>
<th>VP8 Fastest</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 Slowest</td>
<td></td>
<td>2.05390 × 10^{-62}</td>
<td>1.51689 × 10^{-29}</td>
<td>3.00968 × 10^{-21}</td>
</tr>
<tr>
<td>VP8 Slowest</td>
<td>2.05390 × 10^{-62}</td>
<td></td>
<td>1.03356 × 10^{-194}</td>
<td>1.50411 × 10^{-83}</td>
</tr>
<tr>
<td>H.264 Fastest</td>
<td>1.51689 × 10^{-29}</td>
<td>1.03356 × 10^{-194}</td>
<td></td>
<td>1.65574 × 10^{-128}</td>
</tr>
<tr>
<td>VP8 Fastest</td>
<td>3.00968 × 10^{-21}</td>
<td>1.50411 × 10^{-83}</td>
<td>1.65574 × 10^{-128}</td>
<td></td>
</tr>
</tbody>
</table>
Table 4.5: p-values for Frames for the Fastest and Slowest Profiles for Akiyo at 50 Kbps

<table>
<thead>
<tr>
<th></th>
<th>H.264 Slowest</th>
<th>VP8 Slowest</th>
<th>H.264 Fastest</th>
<th>VP8 Fastest</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 Slowest</td>
<td>1.47290 × 10^{-35}</td>
<td>5.15901 × 10^{-11}</td>
<td>1.63758 × 10^{-80}</td>
<td>6.74000 × 10^{-27}</td>
</tr>
<tr>
<td>VP8 Slowest</td>
<td>1.47290 × 10^{-35}</td>
<td>5.14471 × 10^{-230}</td>
<td>6.74000 × 10^{-27}</td>
<td>2.12953 × 10^{-194}</td>
</tr>
<tr>
<td>H.264 Fastest</td>
<td>1.63758 × 10^{-80}</td>
<td>5.14471 × 10^{-230}</td>
<td>2.12953 × 10^{-194}</td>
<td></td>
</tr>
<tr>
<td>VP8 Fastest</td>
<td>5.15901 × 10^{-11}</td>
<td>6.74000 × 10^{-27}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Students**

The test in Section 4.2.1 did not use any restrictions upon key frame placement. For the Students video, two different tests were run: one with key frames at most at every 300 frames, and at most every 30 frames. The Students video was used because it is a facsimile of a video conferencing session with two speakers, and is of a sufficient length to study the effect of key frame placement.

Figure 4.15 shows the summary of the Students video coded at a nominal bitrate of 25 Kbps. The key frame interval was set to 300. VP8 performs well for this case, and rivals H.264 at each encoder profile, but when the rate distortion optimizer is turned off (as it is for the Faster and Fastest settings), the required bitrate increases. However, the PSNR also increases with the utilized bitrate.

The frames for Students encoded at 25 Kbps and 300 key frame interval is shown in Figure 4.16. The vertical dotted lines represent the key frame placement. In general, VP8 does poorly when key frames are needed, where H.264 does well at key frame placement, and then the quality degrades as the distance from the previous key frame grows. This is likely to be characteristics of the encoders rather than the bitstream capabilities – the theoretical possibility available to decoder-defined standards such as H.264 and VP8 – themselves due to the inconsistent behavior at the start of the stream.

Table 4.6 lists the p-values for the Welch’s t-test that was performed on the population of frames in Students encoded at 25 Kbps with a 300 frame key frame interval. The null hypothesis, that there is no significant difference between the two samples of frames, was
Figure 4.15: Summary of Students at CIF Resolution, 25 Kbps, KF Interval 300

Figure 4.16: Frames of Students at CIF Resolution, 25 Kbps, KF Interval 300
rejected in every case ($\alpha = 0.1$) except for H.264 Slowest versus VP8 slowest. We failed to reject the null hypothesis in this case. A value of 0 indicates that the software used to compute the p-value could not represent such a small number, and it is approximately zero.

Table 4.6: p-values for Frames for the Fastest and Slowest Profiles for Students at 25 Kbps, 300 KF Interval

<table>
<thead>
<tr>
<th></th>
<th>H.264 Slowest</th>
<th>VP8 Slowest</th>
<th>H.264 Fastest</th>
<th>VP8 Fastest</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 Slowest</td>
<td>0.95957</td>
<td>1.86571 $\times 10^{-35}$</td>
<td>5.99763 $\times 10^{-168}$</td>
<td></td>
</tr>
<tr>
<td>VP8 Slowest</td>
<td>0.95957</td>
<td>4.07072 $\times 10^{-43}$</td>
<td>9.08420 $\times 10^{-11}$</td>
<td></td>
</tr>
<tr>
<td>H.264 Fastest</td>
<td>2.06915 $\times 10^{-35}$</td>
<td>5.23894 $\times 10^{-43}$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>VP8 Fastest</td>
<td>5.99763 $\times 10^{-168}$</td>
<td>9.08420 $\times 10^{-11}$</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Students was re-encoded at 25 Kbps, but with the restriction that a key frame must be placed at least every 30 frames, so that if the video source was filming at 30 FPS, a key frame would occur once per second. This is a common setup for video conferencing, which needs a quick recovery time if frames are lost.

Figure 4.17 shows the statistical summary of the six profiles for the two encoders. H.264 and VP8 seem to perform similarly for the Slowest profile, but turning off VP8’s rate distortion optimizer in the Faster and Fastest settings really increases the required bitrate from 25 Kbps to above 50 Kbps.

The PSNR of each frame is shown in Figure 4.18. The dotted vertical lines indicate where a key frame was placed, in this case, every 30 frames. VP8 Fastest is often above the other profiles, but it requires almost double the bandwidth, as shown in Figure 4.17. VP8 Slowest has a much more consistent PSNR value than of H.264 or VP8 Fastest.

Table 4.7 shows the p-values for the Welch’s $t$-test on the population of frames in Students encoded at 25 Kbps with a 30 frames key frame interval. In each case, the null hypothesis, that there is no significant difference between the two encoders or profiles selected, was rejected ($\alpha = 0.1$). This indicates that there is a significant difference between
Figure 4.17: Summary of Students at CIF Resolution, 25 Kbps, KF Interval 30

Figure 4.18: Frames of Students at CIF Resolution, 25 Kbps, KF Interval 30
each of the profiles examined.

Table 4.7: p-values for Frames for the Fastest and Slowest Profiles for Students at 25 Kbps, 30 KF Interval

<table>
<thead>
<tr>
<th></th>
<th>H.264 Slowest</th>
<th>VP8 Slowest</th>
<th>H.264 Fastest</th>
<th>VP8 Fastest</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264 Slowest</td>
<td>4.33874 $\times 10^{-09}$</td>
<td>1.05294 $\times 10^{-45}$</td>
<td>2.50456 $\times 10^{-133}$</td>
<td></td>
</tr>
<tr>
<td>VP8 Slowest</td>
<td>1.05294 $\times 10^{-45}$</td>
<td>1.18135 $\times 10^{-11}$</td>
<td>1.08694 $\times 10^{-322}$</td>
<td></td>
</tr>
<tr>
<td>H.264 Fastest</td>
<td>2.50456 $\times 10^{-133}$</td>
<td>1.18135 $\times 10^{-11}$</td>
<td>1.08694 $\times 10^{-322}$</td>
<td></td>
</tr>
<tr>
<td>VP8 Fastest</td>
<td>1.05294 $\times 10^{-45}$</td>
<td>1.18135 $\times 10^{-11}$</td>
<td>1.08694 $\times 10^{-322}$</td>
<td></td>
</tr>
</tbody>
</table>

For video conferencing where a large degree of key frames to ensure fast recovery from dropped or malformed frames, VP8 performs slightly worse than H.264. In addition, its fastest mode, --cpu-used=5, disables an important bitrate optimization, which dramatically increases the necessary bandwidth. This increase in bandwidth will increase the number of frames that are impacted by the channel’s BER or dropped frame rate. To compete with H.264, it has to work more slowly with its rate distortion optimizer enabled.

### 4.2.2 High Definition Television and Film Broadcast Results

Using the composite videos and the animated films discussed in Section 3.5.2, several graphs were generated.

The SSIM results for the 720p videos are shown in Figure 4.19. There are two main categories: low, meaning H.264 Baseline and VP8 Good deadline, and high, meaning H.264 High profile and VP8 Best deadline. The abbreviation “DO” indicates the source video Detail Outdoors, while the abbreviation “CO” indicates the source video City Outdoors. The abbreviation “Sin” stands for Sintel, the 3D animated film.

For Detail Outdoors, H.264 outperforms VP8 in both of its profiles. This indicates two primary things: VP8’s lack of adaptive quantization lowers VP8’s performance on detail oriented video.

The SSIM values for the frames are shown in Figure 4.20. The vertical lines represent
the scene breaks in between the source videos in City Outdoors, mentioned in 3.9. For Old Town Cross, H.264 High and H.264 Baseline substantially outperform VP8 Best deadline. Each encoder was able to maintain a somewhat constant quality as the camera pans.

For Stockholm, a similar video to Old Town Cross, VP8’s SSIM starts low, likely due to its lack of detail retention in the detailed Stockholm scene. VP8, however, steadily improves to track H.264 High profile, outperforming H.264 Baseline for the majority of Stockholm.

In the segment, Ducks Take Off, VP8 starts at a very high level due to the scene’s relative lack of detail compared to Stockholm. This quality drops off sharply as the ducks fly from the water, leaving circular wakes. VP8 tracks H.264 Baseline for the majority of the segment until the end, where it ends up between High and Baseline.

The final segment, In to Tree shows a steep decline in quality in H.264 High and Baseline profiles, while VP8’s quality steadily improves to be the top of all three, ending con-
siderably higher than H.264 High.

Figure 4.20: SSIM Frames of City Outdoors, 720p, nominally 3000 kbps

The SSIM results for the three 1080p videos are shown in Figure 4.21. This graph is similar to Figure 4.19 in structure. The abbreviation “SF” means Sports Film, “OP” means Outdoor Pedestrian, and “Sin” means Sintel, the 3D animated film. VP8 outperforms H.264 Baseline at 1080p on all videos, doing significantly better than H.264 Baseline on the Outdoor Pedestrian video. For H.264 High profile, VP8’s median and average seem to be near H.264, but VP8 has a wider variance.

For Sintel, VP8 outperforms H.264 Baseline by a slight amount, but loses to H.264 High profile by approximately the same amount. The difference in both cases is almost negligible. For the video Sports Film, VP8 outperforms H.264 Baseline by a significant margin except for its rather low minimum, which also moves the average to only slightly above H.264 Baseline. Verse H.264 High profile, the medians are almost indentical, but VP8’s average and 1\textsuperscript{st} and 3\textsuperscript{rd} quartiles are less than H.264.
Examining the frames of Outdoor Pedestrians gives more insight into the results seen in Figure 4.21. Figure 4.22 shows the SSIM values of H.264 Baseline, H.264 High profile, and VP8 Best deadline over time. The vertical lines indicate when one source video begins and another one starts, often triggering a large shift in quality as the encoders need to rapidly adapt.

For Blue Sky, the first of the source videos, as mentioned in Table 3.10, H.264 High profile is the highest in quality, but there is not a huge variation in between any of the three. For the most part, VP8 tracks H.264 Baseline. For Tractor, there is a large difference. VP8 has significant trouble with the zoomed in shot of the tractor, which likely indicates its problem with fast moving detail. As the shot zooms out, however, VP8 tracks H.264 high profile.

The Riverbed video introduces a reduction in performance in terms of SSIM. H.264 Baseline’s SSIM fluctuates, which is likely an artifact of x264’s rate distortion algorithm:
it is trying to encode at a high quality, but does not have the resources to do so. Instead, it quickly raises the quality to attempt to achieve this. The subjective quality of these bounces is likely to be very low. During Riverbed, VP8 steadily declines in quality after a high quality start, but remains significantly above H.264 Baseline’s average. H.264 High profile is able to maintain a high SSIM throughout Riverbed.

For Pedestrian Area, VP8 and H.264 High profile start out equal, but VP8 takes a small lead in SSIM for the entire segment. This is likely due to Pedestrian Area’s static background, which needs little adaptive behavior, and instead is very compressible via VP8’s high efficiency entropy coder. H.264 Baseline profile is easily the lowest of the three by a considerable margin.

Rush Hour is also led by VP8, by an even larger margin than it’s lead on Pedestrian Area. However, it has spikes in SSIM every 30 frames or so, which might be unseemly to a human observer, or, more likely, that the spikes are small enough to go unnoticed.

This frame view of Outdoor Pedestrians shows a much different story than Figure 4.21, and shows how VP8 might be able to compete in the HD realm. Improvements to VP8’s handling of highly dynamic content (Tractor and Riverbed) will improve VP8’s overall performance substantially.

The table of p-values that test for significance is shown in the appendix, Tables B.1 and B.2 in the Appendix on page 135. Each individual segment of City Outdoors and Outdoor Pedestrians and the overall result were each statistically significant, with \( \alpha = 0.1 \).

Both Figure 4.20 and Figure 4.22 had trends that indicated the last video segment performed better than the rest of the video. Another test on the last segment for the same settings was performed to find out if this was the case.

In to Tree and Rush Hour, the last videos of the conglomeration that had high performance were encoded alone, in the same way that the previous videos were. The SSIM summary of this test is shown in Figure 4.23. For In to Tree, H.264 Baseline and H.264
High profile outperform VP8 substantially, with H.264 High’s 1st quartile above VP8’s 3rd quartile.

Rush Hour is easier to encode, as noted by the tight distribution of the SSIM values. H.264 has a slight edge here, with H.264 High profile performing the best of the three.

The frames of In to Tree, as shown in Figure 4.24, show VP8’s tendency to allocate bandwidth to the end of videos in a two pass scenario, rather than in the beginning. It’s performance overall in this video is poor, as it performs worse than H.264 Baseline for much of the video. This graph renders the performance gain VP8 had over H.264 in Figures 4.20 and 4.22 somewhat less. VP8 cannot inherently encode these scenes more efficiently, but instead allocates more resources to them.
Figure 4.23: SSIM Summary of In to Tree and Rush Hour

Figure 4.24: SSIM Frames of In to Tree, 720p
4.3 Intra-coding Test Results

For intra-coding tests, which encompasses the ability of the encoder to use intra-prediction and entropy encoding, two tests were run, each on the same three source videos. Table 3.11 on page 74 summarizes the encoder options used for these two tests.

The first test was a comparison of H.264 Baseline profile and VP8’s good deadline encoder. Figure 4.25 shows the results of bitrate (controlled by the quantization factor parameter) versus quality, measured as SSIM.

The error bars are representative of the standard deviation of the frames, and the data points themselves are the average SSIM of all frames in the video. A high average SSIM with small error bars is desired due to the consistency the encoded video would have; large error bars indicate that the encoded video varies more widely in its quality.

VP8 has a higher SSIM for a given bitrate for each of the three videos encoded. The difference in SSIM between VP8 and H.264 for lower resolution videos is greater than higher resolution images. This is due to its intra-prediction efficiency, which performs more equally on higher resolution images, due to diminishing returns. Further, Parkrun has a high level of detail, which is challenging for intra-prediction to compress. These two factors are responsible for the higher SSIM difference in Foreman and Crew.

Also worthy of note is that H.264 Baseline does not use the $8 \times 8$ DCT that VP8 lacks, further equalizing the compression difference at HD resolutions. The $8 \times 8$ DCT is instrumental in retaining quality at higher resolutions due to the larger sample size it uses.

As the quantization factor increases, and the bitrate decreases, there are more zeros being encoded. Entropy encoders such as VP8’s boolean entropy encoder, compress strings of zeros very well. At the lower bitrates, the SSIM difference between H.264 and VP8 is greater than at higher bitrates, due to VP8’s superior entropy encoder. This shows that VP8’s boolean entropy encoder is significantly better than CAVLC, the entropy encoder
used in H.264’s Baseline profile.

For the second intra-coding test, H.264 High profile was used to compare against VP8’s best deadline encoder. This profile and deadline change were the only parameter changes between the two tests. Figure 4.26 shows the results for this test, and is formatted in the same way as 4.25.

For this test, the results are almost identical between VP8 and H.264. VP8 seems to lose some efficiency at the higher quantization factors, where H.264 has a slightly higher average efficiency. However, the results are so close, and the error is high compared to the difference to render the conclusion insignificant. At higher bitrates, there is no significant measurable difference.

To comment about the statistical significance of the PSNR and SSIM results, Student’s $t$-test and Welch’s $t$-test were used. First, a Welch’s $t$-test was used to measure whether or not the variances were equal. The standard deviation, $\sigma$ was tested. If the resulting p-value was less than the chosen value for $\alpha = 0.1$, the null hypothesis that the variances were equal is rejected. If the variances are unequal, Welch’s $t$-test was used for the average PSNR or SSIM values corresponding to those standard deviations. Otherwise, Student’s $t$-test was used, which assumes equal variances.

All tests used a two-tailed test, which is used when no assumption is made about the data, as in this case. The null hypothesis for the quality metrics is that there is no difference between H.264 and VP8. If the null hypothesis is rejected, it indicates that there is a statistically significant difference between the two.

In Table 4.8, the p-values for the samples are collected. Square brackets indicate statistical significance for $\alpha = 0.1$. For $\sigma$ the only statistically significant result indicating that the variances are different was for Foreman PSNR. Therefore the PSNR p-value for Foreman was calculated with Student’s $t$-test. The only statistically significant difference between H.264 and VP8 was for Crew under PSNR, where it is obvious that VP8 outper-
forms H.264. Otherwise, the significance of their difference is minuscule.

Table 4.8: p-values for the Average PSNR and SSIM Values based on $\sigma$

<table>
<thead>
<tr>
<th>Video</th>
<th>PSNR p-value</th>
<th>PSNR $\sigma$ p-value</th>
<th>SSIM p-value</th>
<th>SSIM $\sigma$ p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreman</td>
<td>0.37246</td>
<td>[0.00886]</td>
<td>0.22581</td>
<td>0.35943</td>
</tr>
<tr>
<td>Crew</td>
<td>[0.01505]</td>
<td>0.30333</td>
<td>0.40258</td>
<td>0.19586</td>
</tr>
<tr>
<td>Parkrun</td>
<td>0.34615</td>
<td>0.20152</td>
<td>0.16909</td>
<td>0.10522</td>
</tr>
</tbody>
</table>
Figure 4.25: Bitrate vs. SSIM for H.264 Baseline and VP8 Good Deadline
Figure 4.26: Bitrate vs. SSIM for H.264 High Profile and VP8 Best Deadline
4.4 Inter-coding Test Results

As discussed in Section 3.7, the tests in Table 3.12 on page 76 were performed and results gathered. Similar to the film and television tests (Section 4.2.2), the VP8 configuration 8 (Table 3.12) was first encoded at a nominal bitrate. For Ice, it was 500 Kbps, and for Parkrun, the nominal bitrate was 1300 Kbps. The output bitrate for Ice was 358 Kbps, and was 1426 Kbps for Parkrun.

Figure 4.27 shows the summary of Ice and Parkrun measured in PSNR. The abbreviation “BF” indicates bidirectional frames. There are three possibilities for each video: no B frames, many B frames (16 for H.264, and VP8 uses altref frames), or weighted B frames (weighted at 100 in x264, VP8 uses a stronger ARNR filter). Using no B frames, configuration 1, tries to achieve VP8 emulation in H.264. It does this by restricting the number of frames it can reference to one, and disabling B frames.

For Ice and Parkrun, in PSNR, it seems that H.264 outperforms VP8 for no B frames. The median is the same for Ice, but VP8 has a lower average and a larger variance. For Parkrun, VP8’s median is approximately H.264’s 1st quartile, but the averages are approximately equivalent. This indicates that VP8 has a higher PSNR skew, and the position of the 3rd quartile confirms that.

For the tests that use B frames, H.264’s advantage depends upon the video. For Ice, where VP8 was indicated to place altref frames, there is little improvement for VP8. There is a similar problem with H.264 with 16 B frames: there is little improvement, except for an increase in the minimum PSNR. When the B frames are weighted, however, the average and median improves and outperforms VP8. For Parkrun, VP8’s altref frames do not result in an improvement. For H.264, it’s not until that heavily weighted B frames are used is a large improvement noted. H.264’s average is near its 3rd quartile, and its median is well above VP8’s. In this video, large amounts of B frames make a distinct difference in
the quality.

![Figure 4.27: PSNR Summary of Inter-coding Tests](chart)

The SSIM summary, Figure 4.28, shows similar results. For Ice, using no B frames and equally weighted P and B frames actually results in VP8 having the slight edge. When heavily weighted B frames are used, H.264 outperforms VP8 on Ice. For Parkrun, H.264 outperforms VP8 for each test. Similarly to the PSNR summary, VP8 notices little to no improvement when altref frames are used.

When B frames are used, H.264 notices an increase in SSIM, and when those B frames are heavily weighted, the difference in SSIM is dramatic. H.264’s 1\textsuperscript{st} quartile is above VP8’s average and median, while H.264’s minimum is near VP8’s median.

The tables for the raw PSNR and SSIM data for Ice and Parkrun can be seen in the appendix, starting on page 136, Section B.2. These are included because the graphs do not display all the tests run. The most relevant tests were displayed graphically.

VP8 had no difference between configurations 9 and 10. This indicates that the ARNR
filter parameters have little to no effect on the output. There are three modes the ARNR can be in, left to right, top to bottom, and center, each numbered from 1 to 3, respectively. Additional tested showed that none of the three types had a deviation from the other.
4.5 Loop Filter Efficiency Test Results

Videos were run in the manner described in Section 3.8 to measure the effectiveness of the encoding’s loop filter. The test was not absolute, but rather differential, measured according to Eq. 3.8.1, for each PSNR and SSIM metrics.

After measuring the average PSNR and SSIM for each video encoded with vpxenc and x264, the percent difference was recorded. The percent differences for each resolution and metric were averaged, and then these were averaged. These averages can be seen in Table 4.9. VP8 has the highest percent difference in PSNR and SSIM, especially for HD resolutions.

More detail can be seen in Figure 4.29. The points for each resolution are the individual video’s average percent difference, and the horizontal line is the average of these points. The Y axis is a log graph to better show the detail at the 0.05 to 1.5 range in percent difference.

VP8 generally is at the top of each resolution, with SSIM being the largest difference. In contrast, H.264’s SSIM values are often near the bottom, indicating that when using SSIM to measure the quality, H.264’s loop filter is not as effective as VP8’s loop filter.

Table 4.9: Average Percent Difference for the four Resolutions and Overall

<table>
<thead>
<tr>
<th>Resolution</th>
<th>VP8 PSNR</th>
<th>VP8 SSIM</th>
<th>H.264 PSNR</th>
<th>H.264 SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIF</td>
<td>2.30459%</td>
<td>2.570615%</td>
<td>1.182509%</td>
<td>0.749946%</td>
</tr>
<tr>
<td>4CIF</td>
<td>1.92730%</td>
<td>0.740154%</td>
<td>1.024065%</td>
<td>0.642160%</td>
</tr>
<tr>
<td>720p</td>
<td>3.48362%</td>
<td>4.301508%</td>
<td>0.208637%</td>
<td>0.211006%</td>
</tr>
<tr>
<td>1080p</td>
<td>2.11403%</td>
<td>1.817493%</td>
<td>0.497533%</td>
<td>0.207108%</td>
</tr>
<tr>
<td>Overall</td>
<td>2.45738%</td>
<td>2.357443%</td>
<td>0.728186%</td>
<td>0.452555%</td>
</tr>
</tbody>
</table>

These results, while positive for VP8, are not without bias. Using bilinear subpixel filtering has an impact on the quality, which is also measured here, which should give VP8 a boost in relative percent difference. We believe that, due to VP8’s filter design, the difference is nominal and these results are still accurate.
Figure 4.29: Percent Difference for PSNR and SSIM among four resolutions for VP8 and H.264
Chapter 5

Conclusion

VP8 is a powerful modern video codec that is suitable for individuals and organizations that seek a patent-free alternative to H.264. Its quality on medium resolution web videos is comparable with H.264, and excels at low resolution and low bitrate videos. Compared to H.264 Baseline, VP8 outperforms it in quality for the same bitrate. Hardware implementations of VP8 are available and may come with some future mobile smartphones.

It is underperforming in higher resolution video, such as HD video, due to its simpler segmentation scheme, which reduces the effectiveness of its adaptive quantization and adaptive loop filter selection. VP8’s entropy coder is approximately as efficient as CABAC, but is somewhat simpler, partially due to the lack of needing to adapt after every bit. VP8’s intra prediction is sophisticated and performs as well as H.264 High profile on intra prediction tests.

The current implementation of VP8’s encoder, vpxenc, does not accurately meet target bitrate requests from the user, and as such, may be impractical for use where specific bandwidth requirements are paramount. The current implementation fails to take full advantage of alternate reference frames, and could benefit significantly from using altref frames as B frames. The RD optimization in vpxenc biases bandwidth towards the end
of the video, which gives inconsistent performance and distorts average PSNR or SSIM analysis.

However, for some videos, VP8’s RD curve is within standard error with even H.264 High profile under x264. Under PSNR and SSIM, for CIF videos, VP8 is commensurate with H.264 High profile, and outperforms H.264 Baseline substantially. For 4CIF and higher resolution, H.264 High profile has a nominally higher performing RD curve when compared with VP8, but the error is high enough to make the difference statistically non-significant. This result matches the subjective MOS scores recorded by Simone et. al[2].

5.1 Improvements to VP8

VP8 would benefit from several improvements. These reflect conceptual suggestions, rather than implementation detail suggestions. Unless otherwise noted in this section, VP8 refers to the current implementation, libvpx, rather than the VP8 bitstream specification.

Improved RD Optimizer

As mentioned in the conclusion, VP8 lacks a high quality rate distortion optimizer, leading to lower bitrates than requested. Further, it seems to spend bandwidth arbitrarily, rather than on difficult sections. It prefers to use bandwidth at the end of videos, likely because it has used too little to meet the average targeted bitrate.

Better Use of Alternate Reference Frames

It’s likely that improved use of alternate reference frames in VP8 will boost compression efficiency by 10% or more, as noted in the section on inter-coding tests. Further, the optional hiding of altref frames is an interesting idea, but all data that is not displayed is
overhead. More research will need to be done to see if this feature is useful, or if it’s better to just use them as bidirectional frames.

**Better Use of Segments**

VP8’s principal shortcoming for HD video is its four segment limit to adapt various parameters to different areas of the frame. H.264 can scale to HD resolutions due to its lack of a limit to the number of slices. At low resolutions, VP8’s FMO-like ability of its segments helps make it outperform H.264 at a low complexity.

We believe that it is possible to leverage altref frames to act like additional segments given a suitably intelligent encoder. The encoder would notice a string of frames with highly similar contents – something that would be handled easily by forward predictive frames – and use the altref frame to predict large portions of the frame. This is obvious and is currently done in the VP8 encoder as of this writing. However, it might be possible to use the segments in the altref frame to the encoder’s advantage, giving you not four segments to work with, but 16. For every segment in the frame encoded, there are four possible source segments from the altref frame, giving 16 possibilities for prediction.

**5.2 Recommendations**

As of yet, VP8’s support and implementations are tenuous, due to its immaturity. Its quality alone does not warrant a switch to VP8; in fact, the opposite is true. VP8’s encoded videos are, in general, lower quality than H.264’s except in low resolution, low bitrate scenarios.

Combined with the possibility for a low cost, high performance hardware implementation, VP8 seems well suited to the surveillance market. Because widespread adoption of a codec is not necessary for such video – only the encoder and decoder need match,
which is simple in proprietary video systems – widespread acceptance is not necessary. Due to its open source implementation, there are no upfront fees, and Google provides a perpetual patent grant. However, due to VP8’s similarity to H.264, and the relatively broad wording of many patents in the MPEGLA’s portfolio, legal action is still possible.

Due to Google’s development of the Android platform, which is the leading smartphone platform[31], VP8 is likely to be well supported through its container WebM in the smartphone market. VP8 outperforms H.264 Baseline profile at smartphone resolutions, which is the typical hardware implementation in smartphones, such as the iPhone. This may accelerate the growth of VP8 for web video and video conferencing.

VP8 has subpar performance for HD video when compared with H.264 High profile. This indicates that VP8 is likely to lag in the film and television markets, where H.264 is already the de facto standard.
Bibliography


Appendix A

Source Code Listings

A.1 The Video Run Script

```perl
#!/usr/bin/perl
use Cwd;
use File::Basename;
use File::Temp qw/tempfile/;
use Time::HiRes qw( gettimeofday tv_interval );

my $QPSNR = "qpsnr";
my $KEYFRAMES = "keyframes";
my $logfile = "$ENV{HOME}/thesis/results/raw";

my %info = (
    vp8 => {
        bin => "vpxenc",
        ext => "webm",
        oflag => "-o",
        iflag => "--threads=1",
        builddir => "$ENV{HOME}/build/libvx-v0.9.6/",
    }
);```
bitrate_factor => 1.0,
passes => "--fpf=/tmp/vpx fp --passes=2 --pass=",
}
}
h264 => {
  bin => "x264",
  ext => "mkv",
  oflag => "-0",
  iflag => "--threads,1 --quiet --no-progress",
  builddir => "$ENV{HOME}/build/x264/",
  bitrate_factor => 1000.0,
  passes => "--stats /tmp/264 fp --pass=",
}
);

my $encoding = shift @ARGV;
my $twopass = 0;
my $inputfile = shift @ARGV;
my $outfile;
my $basename;
my $index = 0;

if ($encoding =~ /jm$/) {
  $outfile = shift @ARGV;
  $basename = basename($outfile);
  $basename =~ s/^(.+)\.([^.]*)\+$1/; # truncate extension
goto QPSNR;
}
if ($inputfile =~ /twopass$/) {
  $twopass = 1;
$inputfile = shift @ARGV;

print "Two pass enabled: \n" "$inputfile \n";
}

@index = shift @ARGV;

$basename = basename($inputfile);
$basename =~ ~$1; # truncate extension
$basename .= $index;

my $enc = $info{l(}$encoding){};

$outfile = "$basename. " . $enc->{ext};

push @ARGV, $enc->{oFlag}{};
push @ARGV, $outfile;
push @ARGV, $enc->{iFlag}{};
push @ARGV, $inputfile;

my $tmp = join "", @ARGV;

my ($seconds, $microseconds) = gettimeofday;
my $enc_output;
if ($twopass)
{
    print "Invoking: $enc->{bin}{} $enc->{passes}{1} $tmp \n";
    system("$enc->{bin}{} $enc->{passes}{1} $tmp 2>&1");
    print "Invoking: $enc->{bin}{} $enc->{passes}{2} $tmp \n";
    $enc_output = ' $enc->{bin} $enc->{passes}{2} $tmp 2>&1';
}
else

{
    print "Invoking:$enc->bin.$tmp\n";
    $enc_output = '$enc->bin tmp 2>&1';
}

my ($osec, $omicro) = gettimeofday;
my $time = ($osec - $seconds) + ($omicro - $microseconds) / (1000.0 * 1000.0);

QPSNR:
my $psnr = 'QPSNR -apnr -r $inputfile $outfile 2>/dev/null';
my $ssim = 'QPSNR -assim -r $inputfile $outfile 2>/dev/null';
my $keyframes = 'KEYFRAMES $outfile 2>/dev/null';

my @blines = split /\n/, $psnr;
my @slines = split /\n/, $ssim;
my @klines = split /\n/, $keyframes;
my @psnr_stats;
my @ssim_stats;

open STATS, "">", "$basename.stats" or
die "Can't open $basename.stats: $1";

my ($s, $k);
my @iframes; # frames where there are key frames
# Average, Minimum, Maximum, Median, 1st Quartile, 3rd Quartile, Variance
my $avg_ssim = my $min_ssim = my $max_ssim = my $med_ssim = my $fqt_ssim = my $tqt_ssim = my $var_ssim = 0.0;
my $avg_psnr = my $min_psnr = my $max_psnr = my $med_psnr = my $fqt_psnr = my $tqt_psnr = my $var_psnr = 0.0;
my $avg_num = scalar(@slines) - 1;

foreach (@blines)
{  
    chomp;
    print STATS $.;
    $s = shift @slines;
    $k = shift @klines;
    chomp $s;
    chomp $k;

    $s =~ s/^.*,(.+),$/1/;  
    $s =~ s/^.*,(.+),$/1/;  

    if ($k =~ m/^(\d+),\s*(\d+)/)  
    {  
        push @iframes, $1 if ($2 > 0);
    }

    if ($s =~ m/^(\d+\.(\d+)?)/ & & $s > 0)  
    {  
        push @ssim_stats, ($s + 0.0);
        # print "ssim: " . ($ssim_stats[scalar(@ssim_stats) - 1]) . " psnr: ";
        $avg_ssim += $s;
    }
    if (/^((\d+\.(\d+)?)/i & & $_. > 0)  
    {  
        push @psnr_stats, ($_. + 0.0);
        # print " " . ($psnr_stats[scalar(@psnr_stats) - 1]) . "\n";
        $avg_psnr += $_.
    }

    print STATS $s . ",\n";
}
$keyframes = join ':', @iframes;  # join the list together

if (scalar(@ssim_stats) != scalar(@psnr_stats) || scalar(@ssim_stats) <= 0)
{
    print "SSIM: \n" . scalar(@ssim_stats) . "\n".
    "PSNR: \n" . scalar(@psnr_stats) . "\n"

    my $max = scalar(@ssim_stats);
    $max = scalar(@psnr_stats) if (scalar(@psnr_stats) > scalar(@ssim_stats));
    for(my $i = 0; $i < $max; $i++)
    {
        print "" . ($i + 1) . »,";

        if($i < scalar(@psnr_stats))
        {
            print $psnr_stats[$i] . »,";
        }
        else
        {
            print "XXX,";
        }
    }

    if($i < scalar(@ssim_stats))
    {
        print $ssim_stats[$i] . »,";
    }
    else
    {
        print "XXX,"
    }
}
print "\n";
}

die "Size\nnot\nequal";
}

@ssim_stats = sort {$a <=> $b} @ssim_stats;
@psnr_stats = sort {$a <=> $b} @psnr_stats;

$avg_ssim /= scalar(@ssim_stats);
$avg_psnr /= scalar(@psnr_stats);

$min_ssim = $ssim_stats[0];
$min_psnr = $psnr_stats[0];
$max_ssim = $ssim_stats[scalar(@ssim_stats) - 1];
$max_psnr = $psnr_stats[scalar(@psnr_stats) - 1];
$med_ssim = $ssim_stats[scalar(@ssim_stats) / 2];
$med_psnr = $psnr_stats[scalar(@psnr_stats) / 2];
$fqt_ssim = $ssim_stats[scalar(@psnr_stats) * 1 / 4];
$fqt_psnr = $psnr_stats[scalar(@psnr_stats) * 1 / 4];
$tqt_ssim = $ssim_stats[scalar(@psnr_stats) * 3 / 4];
$tqt_psnr = $psnr_stats[scalar(@psnr_stats) * 3 / 4];

# Compute the variance, unbiased
#

for(my $i = 0; $i < scalar(@ssim_stats); $i++)
{
    $var_ssim += ($ssim_stats[$i] - $avg_ssim) ** 2;
    $var_psnr += ($psnr_stats[$i] - $avg_psnr) ** 2;
}
$\text{ssim} /= (\text{scalar (@ssim\_stats)} - 1);
$\text{psnr} /= (\text{scalar (@psnr\_stats)} - 1);

# Log

open LOG, ">>", "$\text{logfile}/$\text{encoding}\_$.csv" or
die "Couldn't open $\text{logfile}/$\text{encoding}\_$.csv: $!");

my $\text{bitrate};

if ($\text{encoding} !~ /\text{jm}/i)
{
    $\text{bitrate} = '\text{mediainfo}\ \text{outfile}';
    if ($\text{bitrate} =~ m/Bit\srate/s+:s(\d+.\d+)?s(\d+)\s(Kbps|Mbps)/)
    {
        $\text{bitrate} = "$1$2";
        if ($3 eq "Mbps")
        {
            $\text{bitrate} *= 1000;
        }
    } else
    {
        print "\n\n<<<<<<\n"
        print $\text{bitrate};
        print "\n\n>>>>>>\n"
        die "Couldn't parse output for $\text{bitrate}";
    }

my $\text{currdir} = getcwd;
chdir $\text{enc}=>\text{builddir} or die "Couldn't cd to $\text{enc}=>\text{builddir}: $!");

my $\text{gitnum} = 'git log 2>/dev/null | head -n 1';
chomp $gitnum;

if (length($gitnum) < 5)
{
    # Error
    if ($enc->{builddir} =~ m/v(\d+/\d+/\d+)/)
    {
        $gitnum = $1;
    }
    else
    {
        die "Couldn’t get git commit or version string from path";
    }
}

my $cmdline = join "\", @ARGV;
$cmdline .= $enc->{passes}"X" if $twopass;
print LOG "$inputfile,$index,$gitnum,$time,"
    "$outfile,$cmdline,"
    "$min.psnr,$fqt.psnr,$med.psnr,$tqt.psnr,$max.psnr,$avg.psnr,"
    "$var.psnr,"
    "$min.ssim,$fqt.ssim,$med.ssim,$tqt.ssim,$max.ssim,$avg.ssim,"
    "$var.ssim,"
    "$bitrate,$keyframes\n";

close LOG;
close STATS;
Appendix B

Data Tables

B.1 \( t \)-test p-value Tables for HD Television and Film

Table B.1: p-values for SSIM Frames of City Outdoors, 720p

<table>
<thead>
<tr>
<th></th>
<th>H.264 BP v. VP8</th>
<th>H.264 HP v. VP8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td>( 1.284611 \times 10^{-55} )</td>
<td>( 1.270864 \times 10^{-56} )</td>
</tr>
<tr>
<td>Old Town Cross</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Stockholm</td>
<td>( 3.224369 \times 10^{-26} )</td>
<td>( 2.353832 \times 10^{-21} )</td>
</tr>
<tr>
<td>Ducks Take Off</td>
<td>0.000152</td>
<td>4.746303 \times 10^{-64}</td>
</tr>
<tr>
<td>In to tree</td>
<td>( 3.765971 \times 10^{-228} )</td>
<td>( 2.656876 \times 10^{-104} )</td>
</tr>
</tbody>
</table>

Table B.2: p-values for SSIM Frames of Outdoor Pedestrians, 1080p

<table>
<thead>
<tr>
<th></th>
<th>H.264 BP v. VP8</th>
<th>H.264 HP v. VP8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td>( 2.097634 \times 10^{-14} )</td>
<td>( 4.684079 \times 10^{-15} )</td>
</tr>
<tr>
<td>BlueSky</td>
<td>( 1.695898 \times 10^{-97} )</td>
<td>( 2.995189 \times 10^{-22} )</td>
</tr>
<tr>
<td>Tractor</td>
<td>( 1.141252 \times 10^{-66} )</td>
<td>( 6.713645 \times 10^{-192} )</td>
</tr>
<tr>
<td>Riverbed</td>
<td>( 4.277902 \times 10^{-111} )</td>
<td>( 1.773084 \times 10^{-23} )</td>
</tr>
<tr>
<td>PedArea</td>
<td>( 2.544987 \times 10^{-11} )</td>
<td>( 1.421869 \times 10^{-37} )</td>
</tr>
<tr>
<td>RushHour</td>
<td>( 6.734528 \times 10^{-89} )</td>
<td>( 3.419514 \times 10^{-93} )</td>
</tr>
</tbody>
</table>
# B.2 Inter-coding Test Result Tables

## Table B.3: Raw PSNR Values for Ice for Inter-Coding Test

<table>
<thead>
<tr>
<th></th>
<th>H.264</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Minimum</td>
<td>1st Q</td>
<td>Median</td>
<td>3rd Q</td>
<td>Maximum</td>
<td>Average</td>
</tr>
<tr>
<td>1</td>
<td>28.7258</td>
<td>30.8601</td>
<td>31.3264</td>
<td>31.7167</td>
<td>33.6821</td>
<td>31.28749</td>
</tr>
<tr>
<td>2</td>
<td>29.8239</td>
<td>31.5415</td>
<td>32.0396</td>
<td>32.3694</td>
<td>34.054</td>
<td>31.93619</td>
</tr>
<tr>
<td>3</td>
<td>29.5786</td>
<td>30.9275</td>
<td>31.407</td>
<td>31.7633</td>
<td>33.6621</td>
<td>31.37277</td>
</tr>
<tr>
<td>4</td>
<td>29.8586</td>
<td>31.5459</td>
<td>32.0266</td>
<td>32.3709</td>
<td>34.0823</td>
<td>31.93979</td>
</tr>
<tr>
<td>5</td>
<td>28.7798</td>
<td>31.1498</td>
<td>31.6141</td>
<td>32.1424</td>
<td>34.4165</td>
<td>31.62075</td>
</tr>
<tr>
<td>6</td>
<td>28.3958</td>
<td>30.5001</td>
<td>31.0901</td>
<td>31.7037</td>
<td>34.2573</td>
<td>31.09054</td>
</tr>
<tr>
<td>7</td>
<td>28.5776</td>
<td>30.5557</td>
<td>31.1203</td>
<td>31.7647</td>
<td>34.3585</td>
<td>31.12723</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>VP8</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Minimum</td>
<td>1st Q</td>
<td>Median</td>
<td>3rd Q</td>
<td>Maximum</td>
<td>Average</td>
</tr>
<tr>
<td>8</td>
<td>27.5783</td>
<td>30.3187</td>
<td>31.328</td>
<td>31.9161</td>
<td>34.8618</td>
<td>31.04945</td>
</tr>
<tr>
<td>9</td>
<td>27.6012</td>
<td>30.6868</td>
<td>31.5476</td>
<td>32.2418</td>
<td>34.5567</td>
<td>31.14962</td>
</tr>
<tr>
<td>10</td>
<td>27.6012</td>
<td>30.6868</td>
<td>31.5476</td>
<td>32.2418</td>
<td>34.5567</td>
<td>31.14962</td>
</tr>
</tbody>
</table>

## Table B.4: Raw SSIM Values for Ice for Inter-Coding Test

<table>
<thead>
<tr>
<th></th>
<th>H.264</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Minimum</td>
<td>1st Q</td>
<td>Median</td>
<td>3rd Q</td>
<td>Maximum</td>
<td>Average</td>
</tr>
<tr>
<td>1</td>
<td>0.882117</td>
<td>0.910921</td>
<td>0.917326</td>
<td>0.920761</td>
<td>0.939357</td>
<td>0.916052</td>
</tr>
<tr>
<td>2</td>
<td>0.897363</td>
<td>0.918529</td>
<td>0.924455</td>
<td>0.927809</td>
<td>0.942085</td>
<td>0.923074</td>
</tr>
<tr>
<td>3</td>
<td>0.893067</td>
<td>0.912303</td>
<td>0.918526</td>
<td>0.92147</td>
<td>0.939135</td>
<td>0.917354</td>
</tr>
<tr>
<td>4</td>
<td>0.898251</td>
<td>0.918576</td>
<td>0.924545</td>
<td>0.927632</td>
<td>0.941712</td>
<td>0.923111</td>
</tr>
<tr>
<td>5</td>
<td>0.887259</td>
<td>0.915987</td>
<td>0.922924</td>
<td>0.926344</td>
<td>0.94362</td>
<td>0.921198</td>
</tr>
<tr>
<td>6</td>
<td>0.879325</td>
<td>0.909889</td>
<td>0.917683</td>
<td>0.923086</td>
<td>0.942642</td>
<td>0.916316</td>
</tr>
<tr>
<td>7</td>
<td>0.881436</td>
<td>0.910443</td>
<td>0.918562</td>
<td>0.923625</td>
<td>0.943264</td>
<td>0.916963</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>VP8</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Minimum</td>
<td>1st Q</td>
<td>Median</td>
<td>3rd Q</td>
<td>Maximum</td>
<td>Average</td>
</tr>
<tr>
<td>8</td>
<td>0.893485</td>
<td>0.912797</td>
<td>0.920274</td>
<td>0.926994</td>
<td>0.941436</td>
<td>0.918692</td>
</tr>
<tr>
<td>9</td>
<td>0.893485</td>
<td>0.911105</td>
<td>0.920316</td>
<td>0.927189</td>
<td>0.943117</td>
<td>0.918608</td>
</tr>
<tr>
<td>10</td>
<td>0.893485</td>
<td>0.911105</td>
<td>0.920316</td>
<td>0.927189</td>
<td>0.943117</td>
<td>0.918608</td>
</tr>
</tbody>
</table>
Table B.5: Raw PSNR Values for Parkrun for Inter-Coding Test

<table>
<thead>
<tr>
<th>ID</th>
<th>Minimum</th>
<th>1st Q</th>
<th>Median</th>
<th>3rd Q</th>
<th>Maximum</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>19.9664</td>
<td>20.9349</td>
<td>21.7466</td>
<td>22.94</td>
<td>36.9139</td>
<td>22.2683</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Minimum</th>
<th>1st Q</th>
<th>Median</th>
<th>3rd Q</th>
<th>Maximum</th>
<th>Average</th>
</tr>
</thead>
</table>

Table B.6: Raw SSIM Values for Parkrun for Inter-Coding Test

<table>
<thead>
<tr>
<th>ID</th>
<th>Minimum</th>
<th>1st Q</th>
<th>Median</th>
<th>3rd Q</th>
<th>Maximum</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.640983</td>
<td>0.687208</td>
<td>0.710074</td>
<td>0.735774</td>
<td>0.930365</td>
<td>0.717275</td>
</tr>
<tr>
<td>2</td>
<td>0.669181</td>
<td>0.71266</td>
<td>0.733257</td>
<td>0.762811</td>
<td>0.930518</td>
<td>0.741368</td>
</tr>
<tr>
<td>3</td>
<td>0.6443</td>
<td>0.688104</td>
<td>0.710239</td>
<td>0.741263</td>
<td>0.930491</td>
<td>0.717976</td>
</tr>
<tr>
<td>4</td>
<td>0.670133</td>
<td>0.711482</td>
<td>0.732512</td>
<td>0.763143</td>
<td>0.930518</td>
<td>0.740947</td>
</tr>
<tr>
<td>5</td>
<td>0.668498</td>
<td>0.711252</td>
<td>0.733174</td>
<td>0.762593</td>
<td>0.930518</td>
<td>0.742839</td>
</tr>
<tr>
<td>6</td>
<td>0.65849</td>
<td>0.709746</td>
<td>0.743541</td>
<td>0.805057</td>
<td>0.930036</td>
<td>0.755018</td>
</tr>
<tr>
<td>7</td>
<td>0.659856</td>
<td>0.710904</td>
<td>0.744225</td>
<td>0.805577</td>
<td>0.930116</td>
<td>0.755811</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Minimum</th>
<th>1st Q</th>
<th>Median</th>
<th>3rd Q</th>
<th>Maximum</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.543608</td>
<td>0.619667</td>
<td>0.686507</td>
<td>0.803259</td>
<td>0.960793</td>
<td>0.705465</td>
</tr>
<tr>
<td>9</td>
<td>0.541691</td>
<td>0.619021</td>
<td>0.685261</td>
<td>0.803566</td>
<td>0.960793</td>
<td>0.705181</td>
</tr>
<tr>
<td>10</td>
<td>0.541691</td>
<td>0.619021</td>
<td>0.685261</td>
<td>0.803566</td>
<td>0.960793</td>
<td>0.705181</td>
</tr>
</tbody>
</table>