WebM / VP8

Daniel Wolf

31. August 2012

Zusammenfassung

Diese Ausarbeitung wurde im Rahmen des Seminars des Lehr- und Forschungsgebiet Theoretical Computer Sciences der RWTH Aachen über Kompressionsalgorithmen erstellt. Es enthält vor diesem Hintergrund einen kurzen Überblick über die Geschichte des Videocontainerformat WebM und einen detaillierteren Einblick in die Videokompression durch den VP8 Codec. Dabei steht vor allem die Prediction im Vordergrund. Den Abschluss dieser Arbeit bildet ein kurzer Vergleich von VP8 mit H.264 und einen Überblick über die aktuellen Entwicklungen rund um das WebM-Projekt.

1 Einleitung

Heutzutage verursachen Videos einen erheblichen Teil des gesamten Internet-Traffic und es ist davon auszugehen, dass diese Entwicklung sich in den kommenden Jahren durch die Verbreitung von mobilen Endgeräten weiter verstärken wird. Es gab jedoch bis 2010 keinen freien Video-Codec, der für die Nutzung im Internet geeignet war. Der damals am weitesten verbreite Codec H.264 ist nicht patentfrei und unterliegt bis heute bei kommerzielle Nutzung Lizenzgebühren.

Mit dem WebM-Projekt starte Google im Jahr 2010 einen Versuch einen freien Video-Codec für das Web zu etablieren, der von der Leistungsfähigkeit mit H.264 konkurrieren kann. Hinter dem WebM-Projekt steht das gleichnamige Videocontainerformat WebM. Es basiert auf dem freien Audio-Codec Vorbis und dem Video-Codec VP8, welchen Google im Januar 2010 zusammen mit dem amerikanischen Unternehmen On2 Technologies gekauft hat. VP8 war von On2 Technologies entwickelt und bereits 2008 als geschützter Video-Codec veröffentlicht worden.

VP8 eignete sich besonders gut als Grundlage für den Versuch einen neuen freien Video-Codec zu etablieren, da bereits On2 Technologies das Ziel verfolgt hatte VP8 für die Nutzung im Internet zu optimieren. VP8 sollte im Vergleich zu anderen bekannten Video-Codecs wie H.264 einerseits konkurrenzfähige Kompressionsraten liefern und andererseits besser für das

Abspielen auf mobilen Geräten geeignet sein. Diese beiden Leitgedanken haben die Entwicklung von VP8 geprägt und lassen sich an verschiedenen Punkten des Codecs immer wieder finden. Gleichzeitig mit dem Start von WebM verkündete Google auch den Start des Bildcontainerformates WebP, welches auf der Einzelbildkompression des VP8 basiert. Im Endeffekt ist die Einzelbildkompression dabei nur eine Beschränkung des Videos auf einen Frame.

Diese Arbeit betrachtet VP8 jedoch als Video-Codec und behandelt die Einzelbildkompression nur als Teil der Videokompression. Im Mittelpunkt dieses Papers steht das Kapitel 2, welches die einzelnen Techniken und Konzepte des Codecs betrachtet. Dabei orientiert sich der Aufbau des Kapitels an den Schritten, die ein Encoder durchläuft, wenn er ein Video kodiert. Kapitel 3 enthält einen kurzen Vergleich von VP8 mit H.264 und gibt einen Überblick über die Stärken und Schwächen von VP8. Darauf aufbauend beleuchtet Kapitel 4, warum es WebM bis heute nicht gelungen ist H.264 als führenden Videostandard des Internets abzulösen.

2 VP8

Der VP8 Codec ist nur für Videos definiert, denen ein 8-bit YUV 4:2:0 Bildformat zugrunde liegt. Der Ausganspunkt für jeden Encoder ist daher eine Serie von Frames, die sich jeweils aus der Luminanzkompnente Y und den Chrominazkomponenten U und V zusammensetzt. Die Auflösung der Chromaebene steht dabei im Verhältnis 4:2 zur Auflösung der Lumaebene. Es gibt also pro Pixel im Urbild, einen 8-bit großen Wert in der Luminanzkomponente und auf vier Werte in der Luminanzkomponente kommt jeweils ein 8-bit großer Wert in den beiden Chromakomponenten.

2.1 Grundlagen

Die Frames des Videos werden sowohl vom En- als auch vom Decoder schrittweise und in chronologischer Reihenfolge abgearbeitet. Dabei zerlegen sie einen Frame von links oben nach rechts unten in 16x16 Pixel große Makroblöcke, die wiederum von oben links nach unten rechts abgearbeitet werden. Auf der Ebene des Datenmodells verbergen sich hinter jedem Makroblock ein 16x16 Pixel großer Luminanz- und zwei 8x8 Pixel große Chrominanzkomponenten. Diese werden jeweils nochmal in 4x4 große Subblöcke zerlegt (siehe Abb. 1). Dadurch wird unteranderem die Implementierung eines En- oder Decoders vereinfacht, da bei der Transformation und der Quantisierung nicht zwischen der Luma- und Chromaebene unterschieden wird. Dadurch, dass die Subblöcke der Y, U und V Ebene in die gleiche Datenstruktur passen, muss die Implementierung bei diesen Algorithmen nicht unterscheiden zu welcher Ebene der Subblock gehört. Würde die De- und

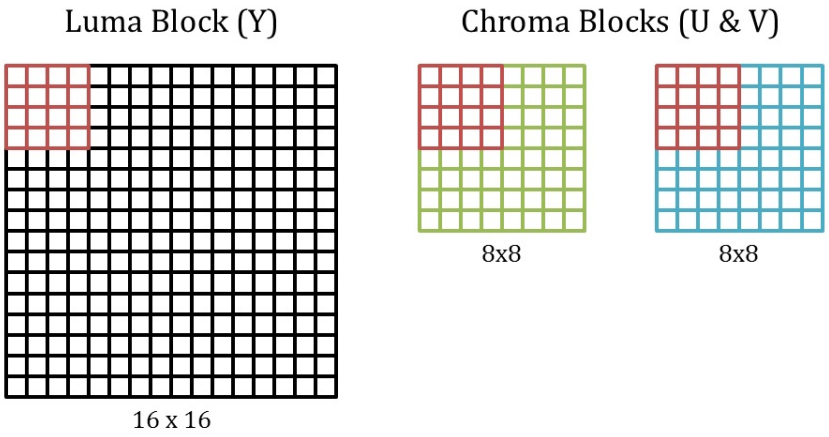

Abbildung 1: Luma- und Chromakomponenten eines Makroblocks und ihre Zerlegung in Subblöcke

Encoder auf der Ebene der Makroblöcke ansetzten, wäre aufgrund der unterschiedlichen Auflösungen der Luma- und Chrominanzblöcke diese Vereinfachung nicht möglich. Außerdem ermöglicht die Unterteilung in die kleineren Subblöcke bei Techniken wie beispielsweise dem Loop-Filter oder der Intraframe-Prediction eine genauere Justierung der Einstellungen, da man einzelne Subblöcke mit Unterschiedlichen Modi bearbeiten kann.

2.2 Prediction

Genau wie die Videocodecs der MPEG-Familie basiert auch VP8 im Kern auf der Idee, dass man anstelle der kompletten Werte für jedes Pixel nur die Differenzen zu einer aus vorhergehenden Daten berechneten Vorhersage (*prediction*) speichert. Der erste Schritt in der Encodierung eines jeden Subblocks ist daher die Berechnung der jeweiligen Prediction. VP8 bietet dazu zwei verschiedene Ansätze, die Intraframe-Prediction und die Interframe-Prediction. Erstere basiert auf der Idee, dass sich benachbarte Pixel eines Bildes bzw. Frames in der Regel ähnlich sind. Die Intraframe-Prediction berechnet daher die Vorhersagewerte für ein Pixel aus den Pixeln benachbarter Blöcke desselben Frames. Die Interframe-Prediction versucht auszunutzen, dass in Videos verschiedene Frames in der Regel sehr ähnlich Dinge zeigen. Zum Beispiel könnte es sein, dass ein Frame dieselbe Szene wie der unmittelbar vorangegangen Frame zeigt, nur um ein paar Pixel verschoben, weil die Kamera sich bewegt hat. Die Interframe-Prediction versucht daher Bereiche vorherzusagen, in dem sie auf Bereiche anderer Frames verweist.

Welche Art der Prediction für die Berechnung des Residums eines Subblocks eingesetzt wird, hängt von dem Makroblock ab, zu dem der Subblock gehört. VP8 kennt zwei Arten von Frames, die Intraframes, welche auch

Key-Frames genannt werden und die den I-Frames des MPEG Standards sehr ähnlich sind, und die Interframes, welche den P-Franes des MPEG Standards nahe kommen. Keyframes dürfen nur Intraframe-Prediction enthalten und sind daher vollkommen unabhängig von anderen Frames. Jeder Makroblock eines Keyframes wird also mit Hilfe der Intraframe-Prediction komprimiert. Im Gegensatz dazu können Interframes sowohl auf der Interals auch auf der Intraframe-Prediction basieren. Es ist also durchaus möglich einzelne Makroblöcke innerhalb eines Interframes mit Hilfe von Intraframe-Prediction zu berechnen. Dies bietet sich vor allem dann an, wenn bereits die Intraframe-Prediction sehr gute Ergebnisse erzielen kann. Die Interframe-Prediction benötigt nämlich Motion Vektoren, welche wiederum zusätzliche Daten sind, die später kodiert werden müssen.

Obwohl Keyframes in der Regel nicht so stark komprimiert werden können wie Interframes, werden sie von den Encoden normalerweise in regelmäßigen Abständen in das Video eingebaut. Dies geschieht zum einen, da sie problemlos als Einstiegspunkt in das Video genutzt werden können und zum anderen verhindern, dass sich Übertragungsfehler unbegrenzt im Video fortpflanzen. Vor allem vor dem Hintergrund, dass VP8 als Codec für das Internet konzipiert wurde, haben diese beiden Funktionen eine besondere Bedeutung, denn sie sind essentiell für das Streaming von Videos.

2.2.1 Intraframe-Prediction

Die Intraframe-Prediction kann auf zwei verschiedenen Ebenen ansetzen. Es gibt vier Basismodi, die sich jeweils sowohl auf die Luminanz- als auch auf die Chrominanzebene eines Makroblocks beziehen. Jedem Makroblock, der mit Hilfe der Intraframe-Prediction berechnet wird, wird einer dieser Basismodi zugewiesen. Für die Chromablöcke gibt es darüber hinaus keine weiteren Möglichkeiten die Intraframe-Prediction zu steuern. Für die Lumaebene gibt es zusätzlich zehn weitere Lumamodi, die auf der Ebene der Lumasubblöcke ansetzen. Sie sind optional und können über ein spezielles Flag aktiviert werden. In einem Makroblock mit aktivierten Lumamodi, wird die Chrominanzebene weiterhin gemäß dem Basismodus des Makroblocks berechnet. Die Lumaebene wird jedoch nicht gemäß dem Basismodus berechnet, sondern es wird jedem Lumasubblock individuell einer der Lumamodi zugewiesen.

Grundsätzlich gilt, dass alle Modi der Intraframe-Prediction nur auf angrenzenden Pixel unmittelbar benachbarte Blöcke basieren, die bereits encodiert wurden. Da sowohl die Makroblöcke als auch die Lumasubblöcke von oben links nach unten rechts abgearbeitet werden, stehen also nur die folgenden zur Verfügung (vgl. Abb. 2):

- Die unterste Zeile des Blocks, der unmittelbar oberhalb des aktuellen Blocks liegt (A)

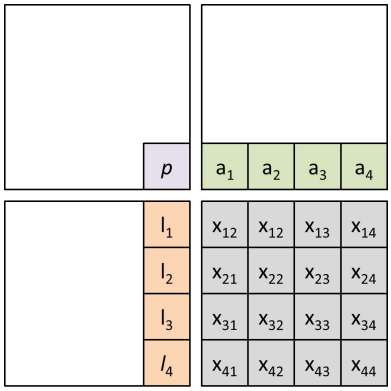

Abbildung 2: Illustration der Referenzwerte für die Intraframe-Prediction am Beispiel eines Subblocks

- Die Zeile am rechten Rand des Blocks, der unmittelbar links des aktuellen Blocks liegt (L)
- Der Wert der unteren rechten Ecke des Blocks, der unmittelbar oben links über dem aktuellen Block liegt (p)

Die vier Basismodi wurden von Bankoski, Wilkins und Xu in ihrem Paper [4] sehr treffend und kurz wie folgt beschrieben (vgl. jeweils Abb. 2):

- **H_PRED (horizontal prediction):** Füllt jede Spalte des Blocks mit einer der Kopie der Spalte L: $x_{ij} = l_i$
- **V_PRED (vertical prediction):** Füllt jede Zeile des Blocks mit einer Kopie der Zeile A: $x_{ij} = a_j$
- **DC_PRED (DC prediction):** Füllt den gesamten Block mit dem Durchschnitt aus allen Werten der Zeile A und der Spalte L:

$$x_{ij} = \frac{\sum_{k=1}^{n}(l_k + a_k)}{2n}, \text{ mit } n = \begin{cases} 16, \text{ für Lumablöcke} \\ 8, \text{ für Chromablöcke} \\ 4, \text{ für Lumasubblöcke} \end{cases}$$

- **TM_PRED (True Motion prediction):** Füllt jedes Feld des Blocks mit der Summe der Elemente mit der gleichen Position aus der Zeile A und der Spalte L. Davon wird dann noch der Wert von p subtrahiert: $x_{ij} = l_i + a_j - p$

Die ersten vier Lumamodi sind identisch mit diesen Basismodi. Zusätzlich gibt es aber noch sechs weitere Lumamodi, die darauf basieren, dass man die Anpassungen nicht nur vertikal oder horizontal, sondern auch diagonal durchführt. Eine Auflistung und Erläuterung dieser Modi findet sich in Kapitel 11.2 des *VP8 Data und Decoding Guide* [3].

Die Prediction basiert dabei nicht auf den Makroblöcken des originalen Videos, sondern auf den Makroblöcken des codierten Videos, welche durch die Quantisierung (vgl. Kapitel 2.4 mit einem Fehler behaftet sind. Dadurch wird sichergestellt, dass der Schritt der Intraframeprediction verlustfrei ist. Als Konsequenz muss jeder Block auch im Encoder nicht nur die Transformation und die Quantisierung, sondern anschließend jeweils auch den inversen Vorgang durchlaufen. Ansonsten kann er für die Intraframe-Prediction nicht genutzt werden.

2.2.2 Interframe-Prediction

Die Interframe-Prediction kann für Makroblöcke von Interframes eingesetzt werden. Dabei wird die Vorhersage für einen Block konstruiert, indem ein Block eines anderen Frames kopiert wird. Die Referenzen werden in Form von Motion-Vektoren gespeichert. Dabei gilt, dass sich alle Motion-Vektoren, die innerhalb eines Makroblocks verwendet werden, auf denselben Frame beziehen müssen. Pro Makroblock können also nur Referenzen auf genau einen anderen Frame benutzt werden.

Dabei kann der Encoder aber nicht beliebig aus allen anderen Frames des Videos wählen. Die VP8 Spezifikation schränkt die Frames, die referenziert werden dürfen, im Vergleich zu anderen Video-Codecs stark ein. Es stehen zu jedem Zeitpunkt für die Referenzen höchstens drei Frames als mögliches Ziel zur Verfügung. Eine Möglichkeit ist dabei immer der Frame der unmittelbar vor dem aktuellen Frame kodiert wurde. Dazu kommen der letzte *golden reference frame* und der letzte *alternate (constructed) reference frame*. Diese Beschränkung auf drei Frames hat den Vorteil, dass VP8 Decoder in der Regel weniger Speicher verbrauchen als die Decoder vergleichbarer anderer Formate, denn es müssen zu keinem Zeitpunkt mehr als vier Frames im Speicher vorgehalten werden. Dabei ist der vierte Platz bereits für den Frame vorgesehen, der zu diesem Zeitpunkt decodiert wird. Die Beschränkung auf nur drei referenzierbare Frames macht vor allem vor dem Hintergrund Sinn, dass Experimente der Gruppe um *Bankoski* [4] ergaben, dass die Qualität des Kompression bei mehr als drei referenzierbaren Frames nicht mehr signifikant ansteigt.

Genau wie die Intraframe-Prediction arbeitet auch die Interframe-Prediction verlustfrei. Der Encoder berechnet sie auf der Basis von Frames, die bereits Transformation, Quantisierung, die dazugehörigen inversen Schritte und auch den Loop-Filter durchlaufen haben. Der Loop-Filter ist dabei ganz bewusst vor die Interframe-Prediction angesiedelt, da sich dadurch der Decodierungsprozess für einige Frames beschleunigen lässt (vgl. Kapitel 2.6). Dieser Umstand kommt der Nutzung von VP8 auf mobilen Endgeräten mit weniger Rechenleistung zu Gute, da Loop-Filter einen beträchtlichen Teil des Rechenaufwandes, der bei der Decodierung eines Videos anfällt, ausmachen.

Golden-Reference-Fame: Jeder Frame des Videos kann vom Encoder bei Bedarf zu einem Golden-Reference-Fame gemacht werden. Er muss dazu nur ein Flag im Header des Framepakets setzen. Außerdem gilt, dass jeder Keyframe automatisch ein Golden-Reference-Fame ist. Empfängt ein Decoder einen Golden-Reference-Fame lädt er diesen nach der vollständigen Decodierung nicht nur auf den Speicherplatz des letzten Frames, sondern hinterlegt ihn zusätzlich an der für den Golden-Reference-Fame vorgesehen Speicherstelle. Dort bleibt er so lange bis der nächste Golden-Reference-Fame oder Keyframe vollständig decodiert wurde. Er kann also in allen Frames bis und einschließlich des nächsten Golden-Reference-Fames referenziert werden. Der Nutzen des Golden-Reference-Fames ist vielfältig und erstreckt sich von der Optimierung des Komprimierungsergebnisses bis hin zur Optimierung von Anwendungsfällen wie dem Streaming von Videos. *Bankoski* zeigt in [2] unter anderem die folgenden beiden Anwendungsbeispiele auf:

- Golden-Reference-Fames können genutzt werden, um einen Frame zu speichern der einen Hintergrund zeigt, vor dem sich im Laufe der folgenden Frames verschiedene Objekte bewegen. Die Rekonstruktion des Hintergrunds an Stellen, von denen sich die Objekte wegbewegen, kann dann auf dem Golden-Reference-Fame basieren.
- Decoder von Videostreams können Golden-Reference-Fames auch zur Wiederherstellung des Bildes nutzen, wenn einzelne Frames oder Datenpakte verloren gegangen sind.

Alternate-Reference-Frame: Das Konzept des Alternate-Reference-Frames ist dem Konzept des Golden-Reference-Fames sehr ähnlich. Auch für die Alternate-Reference-Frames gilt, dass jeder Interframe durch das Setzen eines Flags zu einem Alternate-Reference-Frame werden kann und außerdem jeder Keyframe automatisch ein Alternate-Reference-Frame ist. Der Unterschied zu den Golden-Reference-Fames ist, dass Alternate-Reference-Frames nicht zwingend Teil des eigentlichen Videos sind. Alternate-Reference-Frames bieten ein Flag, welches dem Decoder signalisiert, dass dieser Frame nicht Teil des Videos ist und daher nicht angezeigt werden darf. Der Encoder kann so Alternate-Reference-Frames nutzen, um dem Decoder Frames zu übermitteln, die zwar so nicht Teil des Videos sind, das Kompressionsergebnis aber verbessern. Diese Möglichkeit ist für den VP8 Codec sehr wichtig, denn im Gegensatz zu neueren Codecs aus der MPEG Familie kennt er keine B-Frames. Interframes dürfen keine Referenzen auf Frames enthalten, die in der chronologischen Abfolge des Videos nach ihnen liegen und somit auch erst nach ihnen decodiert werden. Diese Einschränkung reduziert zwar die Komplexität und den Speicherverbrauch des Decodierungsprozesses, ist aber für die Kompression eine relevante Einschränkung. Durch den Einsatz von Alternate-Reference-Frames werden die Auswirkungen dieser Einschränkung jedoch begrenzt.

Motion-Vektoren: VP8 speichert nur Motion-Vektoren, die sich auf Lumainanzkomponenten beziehen. Für die Vorhersage von Chrominanzkomponenten werden eigene Motion-Vektoren aus dem Vektor der zugehörigen Luminanzkomponente berechnet. Jeder Vektor setzt sich aus einer horizontalen und einer vertikalen Komponente zusammen, die die Position des zu kopierenden Blocks referenzieren. Sie geben jedoch nicht die absolute Position des zu kopierenden Blocks im Frame an, sondern beschreiben eine Position in Abweichung zur Position des aktuellen Blocks. Dadurch lässt sich derselbe Motion-Vektor für verschiedene Makroblöcke benutzen, die die gleiche Bewegung vollzogen haben. Da pro Makroblock die Motion-Vektoren nur auf einen anderen Frame verweisen dürfen, muss der Zielframe des Vektors nicht gesondert gespeichert werden. Er wird für alle Vektoren eines Makroblocks im Header des Makroblocks definiert.

Modi der Interframe-Prediction Für alle Videokompressionsverfahren, die mit Motion-Vektoren arbeiten, ist es eine Herausforderung, dass die Motion-Vektoren in der Regel einen beachtlichen Teil des Volumens der codierten Daten ausmachen. Daher versucht VP8 den Umstand auszunutzen, dass benachbarte Makroblöcke häufig die gleiche Bewegung vollziehen und daher den gleichen Motion-Vektor benutzen können. VP8 bietet drei Modi für die Interframe-Prediction, die darauf verzichten einen neuen Motion-Vektor zu definieren. Der einfachste davon ist *mv_zero*. Er verzichtet komplett auf die Angabe eines Motion-Vektors und wird benutzt, wenn sich der Makroblock im Vergleich zu dem ausgewählten Referenzframe überhaupt nicht verschoben hat. „Die Modi *mv_nearest* und *mv_near* benutzen automatisch den letzten bzw. vorletzten von Null verschiedenen Motion-Vektor benachbarter Makroblöcke“ (vgl. [4, S.3]) Eine detaillierte Erklärung wie VP8 bei Sonderfällen (z.B. benachbarte Makroblöcke sind intracoded) entscheidet welcher Motion-Vektor als letzter bzw. vorletzter Vektor gilt, findet man in Kapitel 16 von *VP8 Data Format and Decoding Guide* [3].

Zusätzlich zu den drei bereits genannten gibt es noch die Modi *mv_new* und *mv_split*. Ersterer wird genutzt, wenn ein neuer Motion-Vektor definiert werden muss, letzterer unterscheidet sich von den anderen Modi grundlegend, da er nicht auf der Ebene der Makro-, sondern auf der Ebene der Subblöcke ansetzt. Es wird also nicht mehr dem gesamten Makroblock ein Motion-Vektor zugeordnet, sondern es werden in dem Makroblock verschiedene Bereiche definiert, die sich gleich bewegen. Es gibt dabei vier Möglichkeiten den Makroblock zu partitionieren:

1. Jeder einzelne der 16 Subblöcke erhält einen separaten Motion-Vektor
2. Der Makroblock wird in vier 8x8 große Viertel geteilt
3. Der Makroblock wird in horizontal in zwei 8x16 große Hälfen geteilt

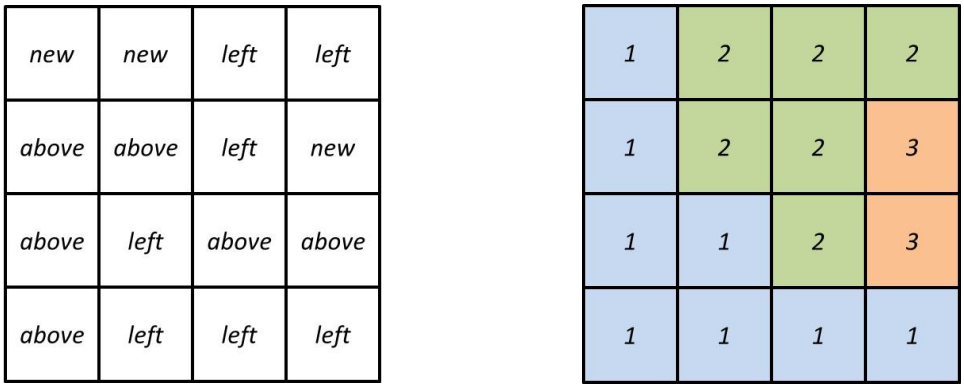

Abbildung 3: Motion-Vektoren eines Subblocks im Interprediction-Modus *mv_split*

4. Der Makroblock wird in vertikal in zwei 16x8 große Hälfen geteilt

Der Modus *mv_split* bietet sich an, wenn in einem Makrobock verschiedene Objekte enthalten sind, die sich unabhängig voneinander bewegen. Da in diesem Modus aber bis zu 16 mal so viele Motion-Vektoren benötigt werden wie in den anderen Modi der Interframe-Prediction, gilt nochmal verstärkt, dass nach Möglichkeit vermieden werden sollte jeden Motion-Vektor neu zu codieren. Daher kommt die Idee, die hinter den Modi *mv_nearest* und *mv_near* steht, wieder zur Anwendung. Anstelle der Definition eines neuen Motion-Vektors für einen Block, kann auch einfach der Vektor *above* oder *left* erneut genutzt werden (vergleiche Abb. 3). Zusätzlich gibt es auch hier die Möglichkeit durch *zero* auf die Angabe eines Motion-Vektors zu verzichten.

2.3 Transformation

Auf die Prediction folgt im Encodierungsprozess die Transformation. Das Ziel der Transformation ist es zusammen mit der Quantisierung die verbliebenen Residuen der Subblöcke für die arithmetischen Kodierung vorzubereiten. Es geht also darum möglichst viele betragsmäßig kleine Werte oder noch besser Nullen in die Subblöcke zu schreiben.

Wie die meisten anderen Codecs setzt auch VP8 dabei in erster Linie auf eine zweidimensionale (2-D) diskrete Kosinustransformation (DCT). Für die inverse Transformation spezifiziert VP8 eine Zwei-Wege-Implementierung der eindimensionalen inversen DCT *LLM*, die von C. Loeffler et al. entwickelt wurde (siehe [10]). Für den Encoder definiert der Codec keine festen Vorgaben. Es ist jedoch zu beachten, dass die (inverse) DCT nur theoretisch verlustfrei ist. In der Praxis sind die Implementierungen der DCT von kleinen Fehlern betroffen, da die Werte nur mit endlicher Genauigkeit abgespeichert werden können. Der Fehler der dadurch entsteht, kann zwar im Vergleich zu dem Fehler, der durch die Quantisierung entsteht, vernachlässigt

werden (siehe [3, S.81]), jedoch ergibt sich daraus, dass sich die Implementierung der DCT im Encoder an der Spezifikation der inversen DCT und im Besonderen ihrem Rundungsverhalten orientieren sollte.

Zusätzlich zur DCT benutzt VP8 noch eine Walsh-Hadamard-Transformation (WHT) als Transformation zweiter Ordnung (nähere Infos zur WHT findet man in [14] und [1]). Auch für WHT definiert der Standard nur die inverse Transformation: $Y = HXH^T$, wobei Y das Ergebnis, X der Input und

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \text{ ist.}$$

Die WHT wird eingesetzt, um die folgenden beiden Umstände auszunutzen: Das Ergebnis der DCT sind Subblöcke an deren erster Stelle (erste Zeile, erste Spalte) der Durchschnittswert (DC) des gesamten Blocks steht. Außerdem sind sich die durchschnittlichen Helligkeitswerte aller Lumasubblöcke, die zu einem Makroblock gehören, in der Regel sehr ähnlich, da die Helligkeit in so einem kleinen Bildausschnitt nicht besonders stark schwankt. Das gilt vor allem, wenn die Prediction für alle 16 Lumasubblöcke eines Makroblocks auf der gleichen Quelle basiert. Die WHT wird genau in diesen Situationen noch zusätzlich zur DCT eingesetzt.

Wenn der Makroblock auf Intraframe-Prediction ohne einen Lumamodus oder auf Interframe-Prediction, die nicht den Modus *mv_split* benutzt, beruht, erstellt der Encoder nach der DCT aus den 16 Lumasubblöcken den sogenannten Y2-Block. Der Y2-Block hat die gleiche Größe wie die Subblöcke (4x4) und enthält jeweils den ersten Wert aus den Lumasubblöcken nach der DCT. Da im Gegenzug dieser Wert jeweils aus den Lumasubblöcken gelöscht wird und damit ihr größter Wert durch eine Null ersetzt wird, lassen sich diese durch die folgenden Schritte der Quantisierung und arithmetischen Kodierung besser komprimieren. Der entstehende Y2-Block, der in der Regel neun sehr ähnliche Werte enthält, ist nach Anwendung der WHT viel besser für die weitere Codierung geeignet (siehe Abb. 4). Die WHT bereitet also eine bessere Codierung vor. Da die benutzte Hadamard-Matrix als Werte nur 1 und -1 enthält und sich somit sehr schnell ohne Multiplikationen berechnen lässt, macht sie den Schritt der Transformation nicht viel aufwendiger und überhaupt nicht fehleranfälliger.

2.4 Quantisierung

Der vorletzte Schritt des Encodierungsprozesses und der zweite Schritt im Decoder ist die Quantisierung. Sie besteht aus einer einfachen Division im Encoder und der dazu inversen Multiplikation im Decoder. Diese werden eingesetzt, um die Werte, die nach der Transformation in den Subblöcken

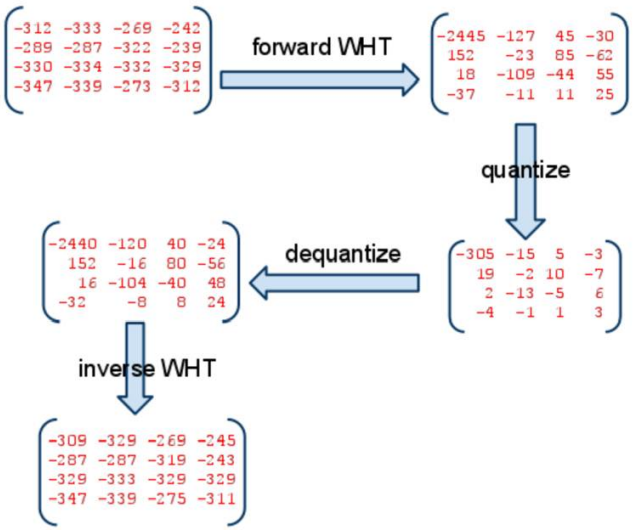

Abbildung 4: WHT und inverse WHT, Quelle: [17]

verbleiben, betragsmäßig weiter zu verkleinern. VP8 arbeitet mit acht verschiedenen Quantisierungslevels. Welches der Levels benutzt wird, richtet sich nach der Position des zu quantisierenden Wertes in seinem Subblock. Es gibt für alle vier Typen von Subblöcken (Y, Y2, U und V) jeweils zwei unterschiedliche Levels. Das erste wird nur für den DC-Wert (erste Zeile, erste Spalte) eingesetzt, das andere für alle anderen 15 Werte.

Mit Hilfe der segmentbasierten Featureanpassung (*segement-based feature adjusment*) können die Quantisierungslevel individuell angepasst werden (siehe [3, S.41]. Wenn die segmentbasierten Featureanpassung für einen Frame aktiviert ist, wird jedem Makroblock innerhalb des Frames eine *segement_id* zugewiesen. Dabei sind pro Frame bis zu vier verschiedene Segmente möglich. Für jedes dieser Segmente können unteranderem eigene Quantisierungslevels für die acht beschriebenen Situationen bestimmt werden. Die Definition dieser Levels erfolgt auf der Basis zweier Quantisierungstabellen (siehe [3, S. 77]).

Durch die Division ist der Schritt der Quantisierung nicht verlustfrei. Die Subblöcke stimmen also nach der inversen Quantisierung nicht mit den Subblöcken überein, die nach der Transformation berechnet wurden. Auf das gesamte Verfahren gesehen ist die Quantisierung der Hauptgrund dafür, dass VP8 Videos nicht verlustfrei komprimieren kann. Der Fehler, der durch die Quantisierung entsteht, macht den Einsatz eines Loop-Filters (siehe Kapitel 2.6) nötig.

2.5 Arithmetischen Kodierung

Die letzte Stufe des Encodierungsprozesses ist die arithmetische Kodierung, welche im Wesentlichen auf dem Prinzip der Huffmankodierung basiert und verlustfrei arbeitet. Der Kontext in dessen Rahmen der gleiche Huffman-Baum benutzt wird ist dabei ein kompletter Frame. Die Wahrscheinlichkeitsverteilung kann somit nur zu Beginn eines jeden Frames bearbeitet werden. Bis auf wenige Headerinformationen werden alle Daten der arithmetischen Kodierung unterzogen. So durchlaufen beispielsweise auch Headerinformationen der einzelnen Makro- und Subblöcke und die Motion-Vektoren diesen Schritt. Dabei versucht VP8 besondere Eigenschaften verschiedener Komponenten gezielt auszunutzen. Beispielsweise werden die Subblöcke nach einem Zickzack-Muster codiert oder in den Fällen, in denen ein Y2-Block existiert, wird die erste Position der Y-Blöcke komplett ausgelassen, da diese per Definition Null sein muss (siehe Kapitel 2.3). Die Feinheiten dieser Tricks lassen sich gut direkt an der Referenzimplementierung in [3] nachvollziehen.

Alle in den vorherigen Kapiteln aufgeführten Techniken arbeiten auf die arithmetische Kodierung hin. Sie setzt auf der Ebene einzelner Bits an und kann eine 0 viel besser kodieren als eine 1. Daher zielten alle vorher aufgeführten Schritte darauf ab, einen Datensatz mit möglichst vielen Nullen zu erzeugen.

2.6 Loop-Filter

Der von VP8 eingesetzte Loop-Filter hatte keine Auswirkungen auf die Kompressionsrate. Er wird aber benötigt um die Qualität des kodierten Videos zu erhöhen. Durch die Fehler der Quantisierung entstehen im Video sogenannte Artefakte. Das bedeutet, dass an den Grenzen zwischen Makroblöcken und an den Grenzen zwischen Subblöcken anstelle weicher Farbläufe oft harte Kanten entstanden sind. Es ist die Aufgabe des Loop-Filters die Farbverläufe durch ein Angleichen der Farbwerte auf beiden Seiten der Kanten möglichst wiederherzustellen.

VP8 bietet verschiedene Einstellungsmöglichkeiten für die Stärke des Loop-Filters. Dabei wird im Wesentlichen zwischen dem *simple filter* und dem *normal filter* unterschieden. Die wichtigsten Unterschiede sind: Während sich ersterer nur auf die Luminanzbene bezieht, behandelt letzterer Y, U und V Blöcke. Außerdem unterscheidet der *simple filter* nicht zwischen Kanten, die zwischen zwei Subbblöcken liegen, und Kanten zwischen zwei Makroblöcken. Er wendet immer denselben Algorithmus an, der nur wenige Einstellungsmöglichkeiten bietet. Der *normal filter* benutzt im Gegensatz dazu auf der Ebene der Subblöcke einen vergleichbaren Algorithmus wie der *simple filter*. Bei Kanten zwischen Makroblöcken greift er aber auf einen komplizierteres Verfahren zurück, das mehr Einstellungsmöglichkeiten bietet. So lässt sich zum Beispiel die Intensität des Filters bestimmen. Genauere Infor-

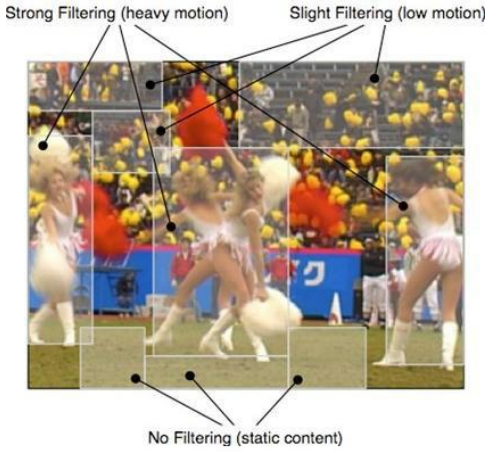

Abbildung 5: Beispiel für EInstellungsmöglichkeiten des Loop-Filter. Das Bild wurde entnommen aus [4]

mationen dazu findet man wiederum in [3]. Der *normal filter* liefert in der Regel bessere Ergebnisse. Er hat aber den Nachteil, dass rechenaufwendiger ist als der *simpel filter*.

Mit Hilfe der bereits in Kapitel 2.4 erläuterten segmentbasierten Featureanpassung lassen sich verschiedene Filterprofile für verschiedene Bereiche eines Frames definieren. Eine Besonderheit des Loop-Filters bei VP8 ist, dass er sich für einzelne dieser Segmente komplett abschalten lässt, was dem Decoder viel Rechenarbeit erspart. Ein dafür beispielhaftes Szenario ist, dass bei einer langen Folge von sehr ähnlichen Frames der Loop-Filter nur auf Keyframes eingesetzt wird. Alle folgenden Interframes greifen dann durch die Interframe-Prediction auf dieses Ergebnis zurück. Da die Veränderungen von Frame zu Frame in diesem Szenario aber minimal sind, besteht kein Bedarf den Loop-Filter noch auf andere Frames anzupassen, was den Decodierungsprozess der Interframes stark beschleunigt. Dieses Vorgehen ist nur möglich, weil die Interframe-Prediction auf den Frames basiert, die den Loop-Filter durchlaufen haben.

3 Vergleich zwischen VP8 und H.264

H.264 ist der momentan am weitesten verbreitete Video-Codec im Internet und es war das erklärte Ziel von Google diesen durch WebM zu verdrängen. Der Erfolg dieses Unterfangs hing und hängt dabei maßgeblich davon ab, ob die beiden Video-Codecs konkurrenzfähig sind.

Grundsätzlich sind sich die beiden Codecs sehr ähnlich und Unterschiede finden sich meistens nur im Detail. Alle in Kapitel 2 behandelt Techniken und Konzepte gibt es in ähnlicher Form auch bei H.264. Daher lassen sich

	MPEG 4	MPEG 4 AVC	WebM
96 kb/s	4,68 MB	0,62 MB	1,23 MB
1000 kb/s	6,18 MB	5,25 MB	5,54 MB

Abbildung 6: Vergleich der Größen des Codierten Videos; Ausgangsdatei: 1.61GB Video; 1280x720 mit einer Länge von 52 Sekunden von http://media.xiph.org/

kaum grundlegende Unterschiede feststellen. Die größten Unterschiede gibt es bei den Frames, auf die im Rahmen der Interframe-Prediction Bezug genommen werden darf. H.264 kennt neben den I-Frames, welche den Keyframes, und den P-Frames, welche den Interframes entsprechen, noch die sogenannten B-Frames. B-Frames dürfen genau wie P-Frames Referenzen auf andere Frames enthalten, nur sind dabei auch Referenzen auf zukünftige Frames möglich. H.264 hat also nicht die strikte Einschränkung, dass sich Frames nur vorher bereits codierte Frames beziehen dürfen. Im Gegenzug bietet VP8 die Alternate-Reference-Frames mit der Möglichkeit Frames einzubinden, die gar nicht Teil des Videos sind. Ob Alternate-Reference-Frames B-Frames gleichwertig ersetzen können, lässt sich nicht abschließend sagen. S.A.Cassidy hat in [6] jedoch herausgefunden, dass die aktuellen Implementierungen der Encoder dieses Konzept bisher kaum ausnutzen. Momentan scheint H.264 VP8 auf diesem Gebiet also noch überlegen zu sein.

Bemerkenswert ist in diesem Zusammenhang außerdem, dass H.264 bis zu 16 Frames vorhält, um sie mit Motion-Vektoren zu referenzieren. Dem gegenüber stehen bei VP8 wie bereits in Kapitel 2.2.2 erläutert ganz bewusst nur drei referenzierbare Frames, um den Speicherverbrauch beim Decodieren zu reduzieren. Ob und wie sich dieser Unterschied wirklich auswirkt lässt sich aus den Tests, die mit beiden Codecs von unabhängigen Dritten durchgeführt wurden, jedoch nicht ablesen. Der einzige Test dazu ist im Paper [4] der VP8 Entwickler erwähnt.

Grundsätzlich gilt, dass VP8 vor allem bei kleineren Videos, die Kompressionsraten von H.264 nicht erreicht (vgl. Abb. 6). Das bedeutet jedoch nicht, dass die Kompressionsraten von VP8 dadurch schlecht sind. Die Unterschiede sind in der Regel nicht besonders groß und daher in Zeiten, in denen Speicherplatz immer billiger wird kaum noch von Bedeutung. Viel wichtiger ist daher die Qualität der Videos. Die objektiven Messungen und Vergleiche von Indikatoren ergeben, dass VP8 in der Regel eine vergleichbare Qualität zu H.264 erzeugt (siehe z.B. [6]). Bei kleinen Bitraten ist VP8 H.264 sogar überlegen. Diese Erkenntnisse werden auch von den meisten Vergleichen bestätigt, die nicht auf objektiven Indikatoren, sondern auf subjektiven Vergleichen der Videos bestehen (z.B. [12] und [7]).

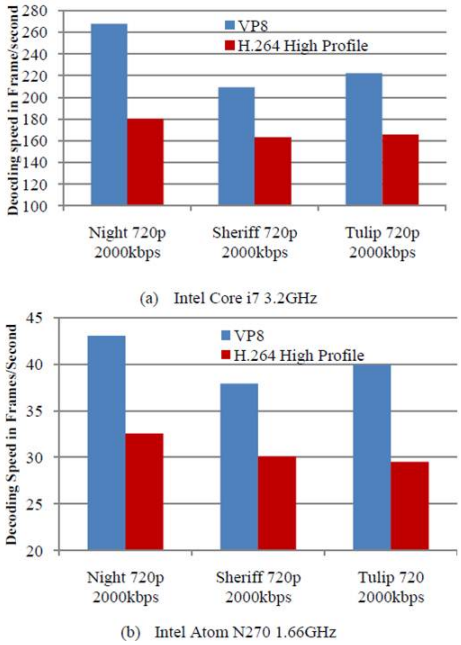

Abbildung 7: Geschwindigkeit des Decodierens, entnommen aus [4]

Eines der obersten Ziele von VP8 war es, einen ressourcenschonenden Decodierungsprozess zu ermöglichen. Laut Bankoski et al. ist dies vor allem bei der Geschwindigkeit des Decodierungsprozess gelungen (vgl. Abb. 7). Im Mittelpunkt der meisten unabhängigen Tests steht jedoch eher die Prozessorauslastung des Decodierers. Diese ist bei VP8 sehr viel höher als bei H.264 (siehe [12]). Die Gründe dafür liegen wahrscheinlich vor allem darin, dass die Decoder bei weitem noch nicht so ausgereift sind wie die Decoder für H.264. Außerdem gibt es zwar erste Grafikprozessoren, die eine Hardware-Unterstützung für WebM anbieten, diese sind aber vor allem in mobilen Geräten noch nicht sehr verbreitet.

Zusammenfassend ist zu sagen, dass VP8 konkurenzfähig zu H.264 ist, aber in der Summe momentan außer der Lizenzfreiheit keine wirklichen Vorteile bietet.

4 Aktuelle Entwicklungen um das WebM-Projekt

Kurz nach der *Google I/O 2010*, auf der Google das WebM-Projekt veröffentlich hatte, schienen die Vorrausetzungen für eine Etablierung des WebM-Standards hervorragend zu sein. Google hatte noch während der Konferenz angekündigt, dass Youtube zukünftig alle Videos als WebM-Datei vorhalten würde. Außerdem hatten viele namenhafte Organisationen wie beispielsweise die Mozilla Foundation und Unternehmen wie AMD oder Nvidia an-

gekündigt das Projekt zu unterstützen. Zusätzlich erklärte Google im Januar 2011, dass der hauseigene Browser Chrome zukünftig genau wie Mozialls Firefox H.264 nicht mehr standartmäßig unterstützen würde (siehe [8]).

Das Ergebnis all dieser Bemühungen ist, dass WebM heute ein etablierter Videostandard ist. Jedoch zeigen Untersuchungen von mefeedia.com, dass der Prozentsatz der im Internet verfügbaren Videos, die mit H.264 codiert sind, immer weiter steigt. Zum Zeitpunkt des Releases von WebM waren nach diesen Untersuchungen 26% aller im Internet verfügbaren Videos in H.264 codiert. Dieser Wert ist seitdem auf 80% gestiegen (s. [16]). Man muss daher konstatieren, dass es WebM nicht gelungen ist H.264 zu überflügeln und dieser Schritt scheint auch mittelfristig nicht mehr absehbar, da WebM momentan eben keine grundsätzlichen Vorteile bietet (vgl. Kapitel 3).

Dieser Trend spiegelt sich auch im Verhalten der Unterstützer und Förderer des WebM-Projekt wieder. So hat Google zwar die Ankündigung, alle Videos auf Youtube als WebM-Datei vorzuhalten, umgesetzt. Sie liegen aber auch gleichzeitig noch als MPEG4/AVC Datei bereit. Auch hat sich Google bis heute nicht getraut H.264 aus Chrome zu entfernen. Der Grund dafür lässt sich erahnen, wenn man einen Blick auf Mozilla wirft. Mozilla hat vor kurzem angekündigt mit seinen alten Prinzipien zu brechen und H.264 als Standard in Firefox einzufügen. Begründet wird dieser Schritt damit, dass ein Browser ohne H.264 auf dem Markt der mobilen Geräte wie Handys kaum eine Chance hat (siehe [9]). Es scheint also so als hätten selbst die einstige Unterstützer und Förderer den Glauben daran verloren, dass WebM H.264 jemals vollständig verdrängen könnte.

Ein weiteres Problem für die Etablierung von WebM war, dass seit dem Start des WebM-Projekts umstritten ist, ob VP8 wirklich keine Patente anderer Firmen verletzt. Das amerikanische Unternehmen MPEG LA, an dem unter anderem Apple und Microsoft beteiligt sind, bezweifelte die Patentfreiheit von Beginn an (siehe [11]). Jedoch ist zu beachten, dass MPEG LA die Rechte am heute weitverbreiten und lizenzgebührenpflichtigem (siehe [13]) H.264 hält und daher ein gesteigertes Interesse daran hat, dass sich kein Konkurrenzformat etablieren kann. Die rechtliche Lage ist bis heute nicht abschließend geklärt. MPEG LA hat noch im Jahre 2010 öffentlich Interesse bekundet einen Patent-Pool für durch VP8 verletzte Patente anzulegen, um die Interessen gesammelt zu vertreten. Auf einen offiziellen Aufruf im Februar 2011 haben sich laut Angaben von MPEG LA zwölf Firmen gemeldet (siehe [5]). Bis August 2012 hat aber keine der beteiligten Parteien noch weitere Schritte unternommen, um den vermeintlichen Patentkonflikt zu klären. Google geht offiziell weiter davon aus, dass VP8 keine Patene verletzt, während MPEG LA das Gegenteil behauptet. Eine abschließende Klärung der Patentsituation ist daher bis heute nicht erfolgt und wird wohl mittelfristig von Gerichten getroffen werden müssen.

Literatur

[1] A. Abdelazima, M. Varleya, and D. Ait-Boudaoudb. Eeffect of the hadamard rransdorm on motion estimation of different layers in video coding. In *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXVIII, Part 5*, Newcastle upon Tyne, UK, 2010.

[2] J. Bankoski. On2's Truemotion VP7 video codec and golden frames. *EE Times*, 2008.

[3] J. Bankoski, J. Koleszar, L. Quillio, J. Salonen, P. Wilkins, and Y. Xu. VP8 Data Format and Decoding Guide, 2011. http://tools.ietf.org/html/rfc6386.

[4] J. Bankoski, P. Wilkins, and Y. Xu. Technical Overview of VP8, an Open Source Video Codec for the Web, 2011. http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/de//pubs/archive/37073.pdf.

[5] Peter Bright. MPEG LA: 12 companies own patents essential to Google's VP8 codec. am 15.03.2012 erschienen auf *http://arstechnica.com*: http://arstechnica.com/business/2011/07/mpeg-la-12-companies-own-patents-essential-to-googles-vp8-codec/, besucht am 25.08.2012.

[6] S.A. Cassidy. An Analysis of VP8, a New Video Codec for the Web. Master's thesis, Department of Computer Engineering Rochester Institute of Technology, 2011. https://ritdml.rit.edu/bitstream/handle/1850/14525/SCassidyThesis11-2011.pdf?sequence=1.

[7] J. Garrett-Glaser. The first in-depth technical analysis of VP8. am 19.05.2010 erschienen auf *http://x264dev.multimedia.cx*: http://x264dev.multimedia.cx/archives/377,besuchtam25.08.2012.

[8] Mike Jazayeri. More about the Chrome HTML Video Codec Change. am 14.01.2011 erschienen auf *http://blog.chromium.org*: http://blog.chromium.org/2011/01/more-about-chrome-html-video-codec.html, besucht am 27.8.2012.

[9] Anita Klingler. Mozilla kapituliert gegenüber Web-Videoformat H.264. am 15.03.2012 erschienen auf *www.zdnet.de*: http://www.zdnet.de/41560918/mozilla-kapituliert-gegenueber-web-videoformat-h-264/, besucht am 25.08.2012.

[10] C. Loeffler, A. Ligtenberg, and G. S. Moschytz. Practical fast 1-D DCT algorithms with 11 multiplications. In *Proceedings of the IEEE*

International Conference on Acoustics, Speech, and Signal Processing (ICASSP '89), volume 2, pages 988 – 991, Glasgow, UK, 1989.

[11] Cade Metz. Google open video codec may face patent clash. am 15.03.2012 erschienen auf *http://www.theregister.co.uk*: http://www.theregister.co.uk/2010/05/21/mpegla_mulls_patent_license_for_webm/, besucht am 25.08.2012.

[12] J. Ozer. WebM vs. H.264: A Closer Look. am 30.06.2010 erschienen auf *http://www.streamingmedia.com*: http://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=68594,besuchtam25.08.2012.

[13] Florian Plag. MPEG-4: Alles zum Thema Lizenzrechte. am 15.12.2008 erschienen auf *http://www.video-flash.de*: http://www.video-flash.de/index/mpeg-4-alles-zum-thema-lizenzrechte/, besucht am 25.08.2012.

[14] W.K. Pratt, J. Kane, and H.C. Andrews. Hadamard Transform Image Coding. In *Proceedings of the IEEE, VOL. 57, NO. 1*, Newcastle upon Tyne, UK, 1969.

[15] F. D. Simone, L. Goldmann, J.s. Lee, and T. Ebrahimi. Performance analysis of VP8 image and video compression based on subjective evaluations. In *SPIE Optics and Photonics, Applications of Digital Image Processing XXXIV,*, San Diego, 2011.

[16] *Frank*. HTML5 Based Video Availability. am 19.12.2011 erschienen auf *http://blog.mefeedia.com*: http://blog.mefeedia.com/html5-dec-2011, besucht am 25.08.2012.

[17] *Multimedia Mike*. The Big VP8 Debug. am 19.10.2010 erschienen auf *http://static.googleusercontent.com*: http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/de//pubs/archive/37073.pdf, besucht am 25.8.2012.

[18] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. In *IEEE Transactions on Circuits and Systems for Video Technology, vol. 13*, pages 560 – 576, 2003.