Informatik-Projekt 9: Progressive Movie File

Stefan Weber

Herbstsemester 2011/2012

Ein Projekt im Studiengang Master Of Science in Engineering

Fachhochschule Nordwestschweiz, Hochschule für Technik
Institut für Mobile und Verteilte Systeme
Steinackerstrasse 5, CH-5210 Windisch

Projektbetreuer: Prof. Dr. Christoph Stamm

Abstract

Dies ist die Dokumentation des Informatik-Projekts 9 von Stefan Weber, am Studiengang Master Of Science in Engineering (MSE) für Informatik der Fachhochschule Nordwestschweiz.

In den Projekten 7 und 8 hat der Autor das Video-Codec MPGF mit purer Einzelbildkompression erarbeitet und es für Live-Streaming optimiert (LPGF). Ein theoretisches Schema für das zu entwickelnde Codec PMF mit Bewegungsschätzung wurde entworfen.

In diesem Dokument werden die theoretischen und praktischen Überlegungen und Erkentnisse dargelegt, welche der Autor im Verlauf des Semesters erlangte. Im ersten Teil dieser Dokumentation werden Verbesserungen am bisher entwickelten Codec eingeführt: Die Aufteilung der übertragenen Bilder in mehrere Teile ist für das Streaming-Verhalten von Vorteil – jedoch nur bis zu einem bestimmten Grad. Dieses Optimum wird im Kapitel 2.3 bestimmt. In Kapitel 3 wird das Verfahren hierarchischer Bewegungsschätzung vorgestellt sowie den verbesserten Algorithmus *3DRS*, welcher die rechnerische Komplexität stark verringert, diskutiert. Zum Schluss betrachten wir die Herausforderungen bei der Implementation des Video-Codec *Progressive Movie File* (PMF Version 3), analysieren die erreichten Werte und schlagen weitere Optimierungen vor.

Der Autor

Stefan Weber hat nach seiner Lehre als Mediamatiker von 2007-2010 den Bachelor-Studiengang für Informatik mit Spezialisierung in *Information Processing and Visualization* an der Fachhochschule Nordwestschweiz absolviert. Im Anschluss ist er im Herbst 2010 in den Vollzeit *Master Of Science in Engineering* Studiengang für Informatik eingetreten, in dessen Rahmen diese Projekte stattfinden.

Inhaltsverzeichnis

1 Einführung **3**
1.1 Aufgabenstellung . . . 3

2 Verbesserungen am Codec **4**
2.1 Overhead im Advanced Streaming Format . . . 4
2.2 Frame-Aufteilung . . . 5
2.3 Optimum . . . 6

3 Bewegungsschätzung **8**
3.1 Hierarchische Bewegungsschätzung . . . 8
3.1.1 Der 3DRS Algorithmus . . . 8

4 Implementation PMF Version 3 **10**
4.1 Differenzbild Fehlervererbung . . . 10
4.2 Bewegungsschätzung . . . 12
4.2.1 Minimierung der Bitrate . . . 12
4.2.2 Kompression der Bewegungsvektoren . . . 13
4.3 Decoder Multi-Threading . . . 15
4.4 Layer-Dropping Adaptivitäts-Schema . . . 16
4.4.1 Qualitätsreduktion mit Differenzübertragung . . . 17

5 Anpassungen am Bildcodec **18**

6 Reflexion **20**

7 Ehrlichkeitserklärung **21**

Literaturverzeichnis **22**

A PMF Version 1 Schemata **23**

B Paper-Draft **25**

1 Einführung

Die vorliegende Master Thesis ist das dritte Projekt des Studierenden im Master-Studiengang. Da die beiden Vorprojekte eng mit dem vorliegenden Projekt verwandt sind, wird dem Leser die Lektüre der Projektdokumentationen *IP 7* und *IP 8* empfohlen (auf dem mitgelieferten Datenträger enthalten).

Video-Streaming über mobile Paketdienste erfordert effiziente Kompressionsalgorithmen und den Einsatz adaptiver Verfahren zum Ausgleich von Übertragungsqualitätsschwankungen. Skalierbare Codierungsverfahren sind eine Alternative zum vorgängigen Auswählen einer voreingestellten, sinnvollen mittleren Übertragungsqualität und auch zum adaptiven Umschalten verschiedener vorcodierter Streams. Solche Verfahren erzeugen nur einen Stream, welcher aber die Übertragung in Abhängigkeit des Durchsatzes und die Decodierung von Teil-Streams erlaubt.

Abbildung 1 illustriert den Kontext, in welchem diese MSE-Projektreihe stattfindet. Im aktuellen Projekt liegt der Fokus auf der Optimierung des Streaming-Verhaltens (Server - Client) des bestehenden Codec durch Aufteilung der Übertragung und Reduzierung der Bitrate.

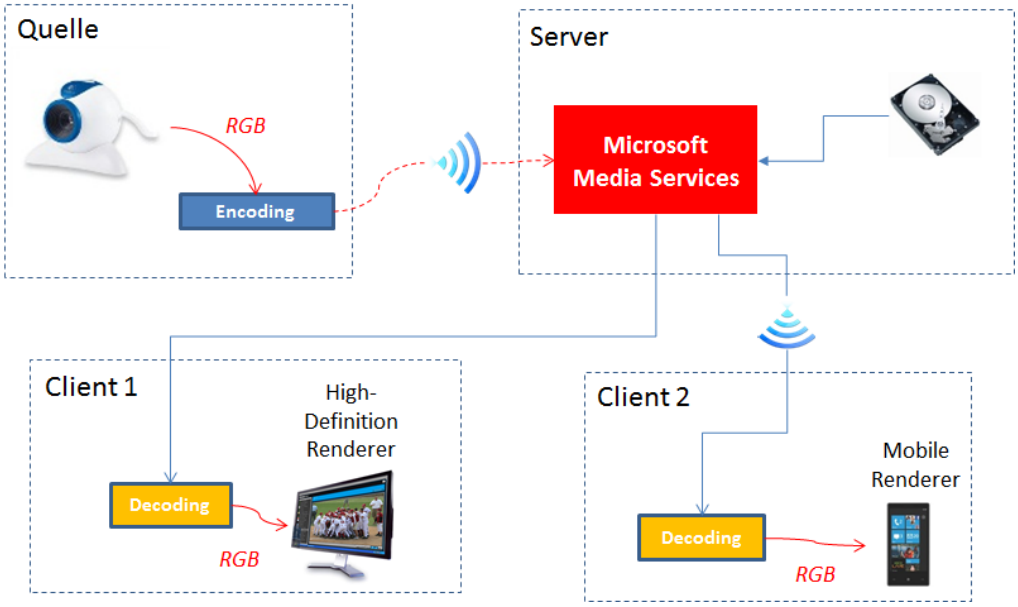

Abbildung 1: Eine Übersicht über den Kontext dieser Arbeit

1.1 Aufgabenstellung

Welche Video-Techniken sind heute im praktischen Einsatz für Live-Streaming vorhanden? Welche Voraussetzungen müssen erfüllt sein, damit sie genutzt werden können? Welche Anpassungen an unserem Videoformat PMF sind erforderlich, um ein praxistaugliches, adaptives Video Live-Streaming zu ermöglichen?

Der Studierende soll ein entsprechendes System in Absprache mit dem Betreuer entwickeln und erproben.

2 Verbesserungen am Codec

Der bereits entwickelte Video-Codec PMF verwendet das *Progressive Graphics File* (PGF) zur Kompression der Einzelbilder (siehe Stamm [1,2]). PGF basiert auf der *Diskreten Wavelet Transformation* (DWT) und implementiert eine *Multiskalen-Analyse* zur progressiven Erzeugung der *Wavelet-Pyramide*.

Die in PMF Version 1 verwendeten *Tiles* sind kleine Bildausschnitte des komprimierten Bildes, welche die Grösse des kleinsten Vorschaubilds der Wavelet-Pyramide haben. Im vorherigen Projekt wurde bereits gezeigt, dass durch Aufteilung des Bildes zur Übertragung ein optimiertes Streaming-Verhalten erreicht werden kann, wie in der Skizze in Abbildung 2 dargestellt.

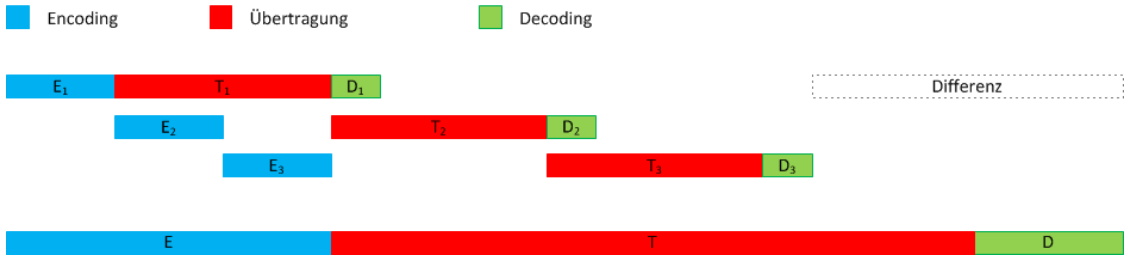

Abbildung 2: Übertragung des Bildes in 3 Teilen verglichen mit Übertragung als Einzelbild (unten)

2.1 Overhead im Advanced Streaming Format

PGF teilt das Bild in 2^{2*L} Tiles auf, wobei L der Anzahl Levels entspricht. Für ein Bild mit 5 Skalen erhalten wir somit $2^{2*5} = 1024$ Tiles. Durch diese Aufteilung werden die einzelnen übertragenen Pakete sehr klein, was zu einem überhöhten Overhead im verwendeten *Advanced Streaming Format* (ASF) Container führt, wenn jedes Tile separat übertragen wird: Beim genannten Beispiel mit 5 Skalen kann der Overhead 15% der Gesamtgrösse ausmachen. In Abbildung 3 wird die resultierende Dateigrösse an einem Beispiel in Abhängigkeit der Bild-Aufteilung illustriert.

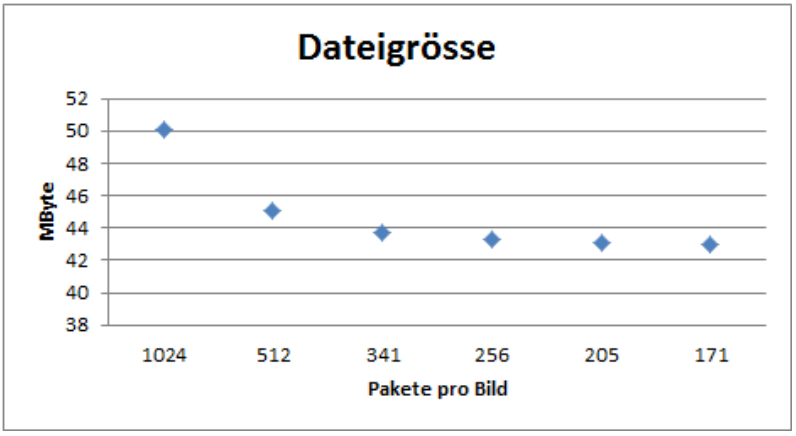

Abbildung 3: Der Verlauf der Dateigrösse bei Zusammenfassung von Tiles

Gemäss diesen Ergebnissen sollte die Anzahl Teile demnach minimiert werden, um die Dateigrösse so gering als möglich zu halten.

2.2 Frame-Aufteilung

Nebst der Dateigrösse spielt auch die rechnerische Komplexität des Codecs eine grosse Rolle. Abbildung 4 zeigt die gemessene Geschwindigkeits-Performance des Video-Codecs bei Aufteilung der Bilder in verschieden viele Teile.

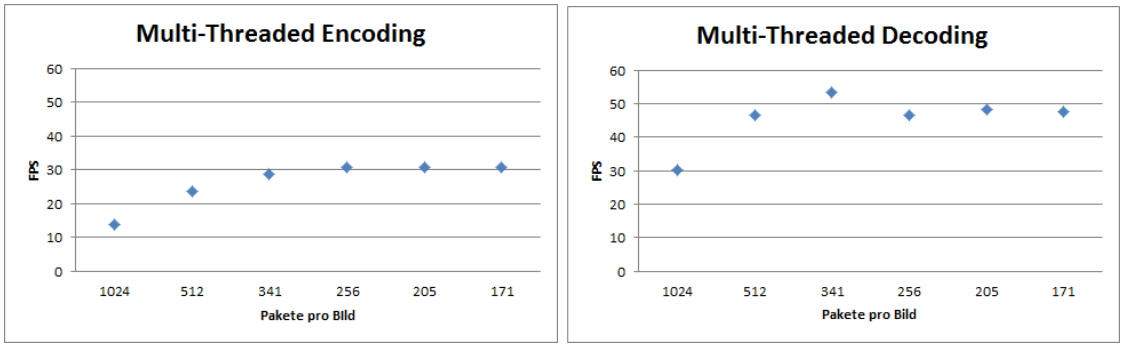

Abbildung 4: Die Codec Performance bei Zusammenfassung von Tiles

Wie die resultierende Bitrate nimmt also auch die benötigte Zeit für das Encoding und Decoding ab, je weniger ein Bild aufgeteilt wird.

Die Aufteilung des Bildes in einzelne Pakete wurde jedoch aus einem Grund gemacht: um das Streaming-Verhalten des Codecs zu verbessern (siehe Abbildung 2). Deshalb wollen wir im folgenden untersuchen, inwiefern der theoretische Streaming-Vorteil und die effektive Performance durch die Zusammenfassung von Paketen beeinflusst wird.

Die Zeit für das Fertigstellen eines Frames kann wie folgt theoretisch berechnet werden: Seien $E_1 = E_2 = ... = E_n$ die theoretisch gemittelten Encoding-, $D_1 = D_2 = ... = D_n$ die Decoding- und $T_1 = T_2 = ... = T_n$ die Übertragungs-Dauer der n Bildpakete ($E = \sum_{i=1}^{n} E_i$, $D = \sum_{i=1}^{n} D_i$ und $T = \sum_{i=1}^{n} T_i$), dann ergibt sich die Frame-Dauer F in Abhängigkeit der Anzahl Teile aus:

$$F(n) = E_1 + \sum_{i=1}^{n} T_i + D_n \mid \text{falls } T > E \wedge T > D$$

$$F(n) = \sum_{i=1}^{n} E_i + T_n + D_n \mid \text{falls } E > T \wedge E > D$$

$$F(n) = E_1 + T_1 + \sum_{i=1}^{n} D_i \mid \text{falls } D > E \wedge D > T$$

vergleiche Abbildung 2.

Wenn nun in einem realistischen Szenario (357 KB/Frame, 20 Mbit/s Netzwerkdurchsatz) die zu übertragenden Daten in $n = 3$ Teile zerlegt werden, ergibt sich eine theoretische Reduktion um 28% der verwendeten Zeit pro Frame (E und D sind empirisch ermittelte Werte, die Übertragungsdauer T sei ein theoretischer Wert):

$$F(1) = E + T + D = 72ms + 143ms + 33ms = \mathbf{248}ms$$

$$F(3) = E_1 + \sum_{i=1}^{3} T_i + D_3 = \frac{72ms}{3} + 3 * \frac{143}{3} + \frac{33}{3} = 24ms + 143ms + 11ms = \mathbf{178}ms$$

In Abbildung 5 wird ein Plot der oben genannte Funktion $F(n)$ dargestellt. Ab 10 Bildteilen nähert sich die Funktion immer mehr der Übertragungsdauer von 143 ms an, da die (theoretische)

Encoding- und Decoding-Zeit pro Bildteil gegen 0 konvergieren. $F(n)$ ist also limitiert durch den dominanten Faktor – in diesem Beispiel die Übertragungsdauer.

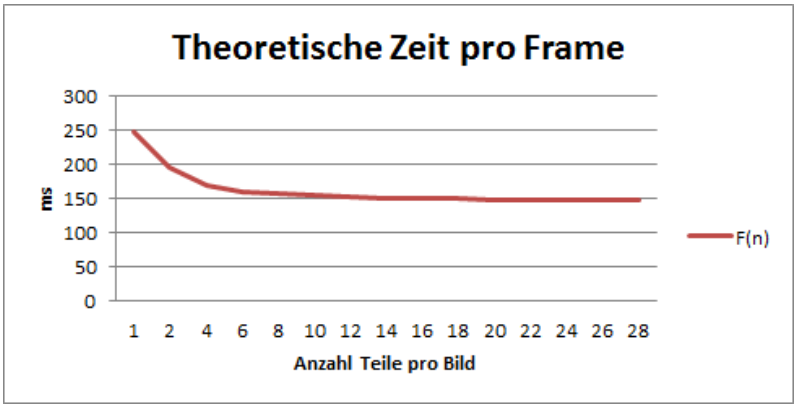

Abbildung 5: Theoretische Berechnung der benötigten Zeit pro Frame: Encoding, Decoding und Übermittlung

Offensichtlich bringt eine Aufteilung des Bildes in 1024 Teile, wie sie in PGF gemacht wird, deshalb kaum mehr theoretische Geschwindigkeitsvorteile, weil die Kurve rasch gegen den dominanten Zeitfaktor konvergiert. Das Zusammenfassen der PGF-Bildpakete, um den in Abbildung 4 gezeigten Speedup zu nutzen, ist deshalb zu empfehlen.

2.3 Optimum

Wir haben also gesehen, dass einerseits eine Aufteilung des Frames wünschenswert ist, aber sich andererseits bei zu grosser Aufteilung der Overhead des ASF Formats zu stark auswirkt. Diese beiden Kurven wollen wir im Folgenden gegeneinander auswerten, um ein Optimum zu finden.

Die totale Verarbeitungsdauer unterteilen wir zwei Teile:

- *Interne* Zeit: Die effektive Prozessierungsdauer des entsprechenden Transform, z.B. das Komprimieren der Videodaten.
- *Externe* Zeit: Dauer, welche zwischen den Aufrufen des Transforms verstreicht. Enthält unter anderem Input/Output.

Beide Zeiten können mit einer exponentiellen Funktion approximiert werden:

$D(x) = C * x^b + O.$

Wobei C eine Performance-Konstante darstellt, O der Offset der Funktion und b der Steigungs-Exponent. Die Abbildungen 6 und 7 stellen folgende Werte dar:

- Die *externe* Zeit beim En-/Decoding-Vorgang im Verlauf der Anzahl zusammengefasster Tiles pro Sample dar, wobei Messwerte und Approximation eingezeichnet sind.
- Der theoretische Zeitaufwand bei einem Streaming-Szenario (Vorteil durch Aufteilung der Frames): *Interne* Prozessierungs-Dauer plus theoretische Netzwerkübertragung.
- Die Summe ergibt eine Funktion, bei deren Minimum die optimale Anzahl zusammenzufassender Bildteile liegt.

Es zeigt sich, dass in diesem Szenario das (theoretische) Optimum bei 30 übertragenen Paketen pro Frame liegt, weil dort die Summenfunktion ein Minimum ergibt.

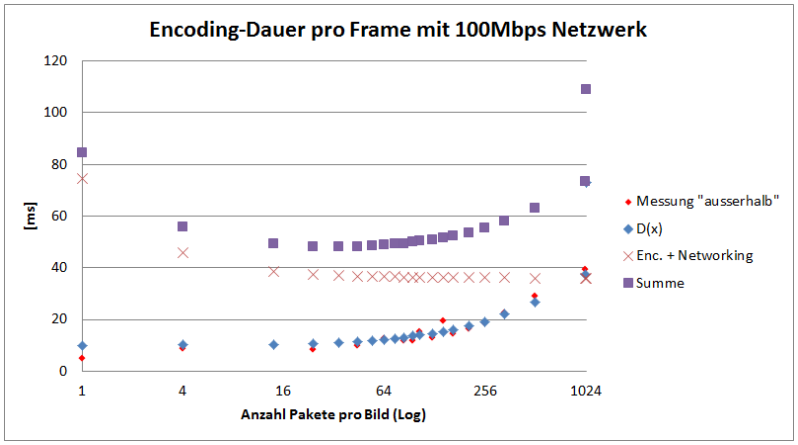

Abbildung 6: Encoding: Totalzeit aus "externer" Zeit plus theoretischer Prozessierungs- und Netzwerk Übertraungsdauer.

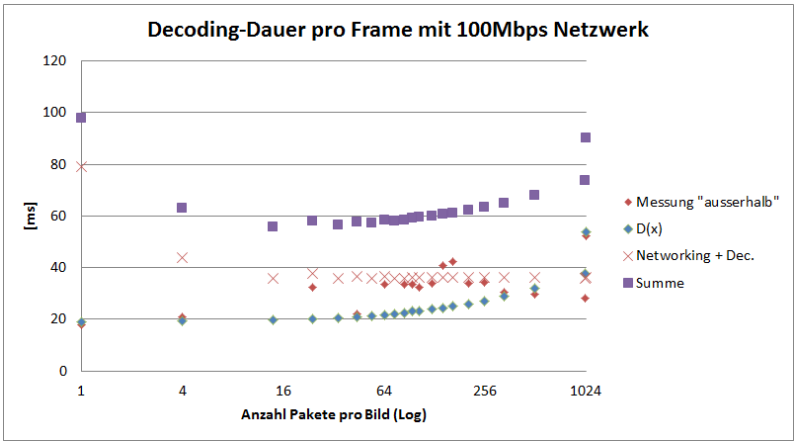

Abbildung 7: Decoding: Totalzeit aus "externer" Zeit plus theoretischer Prozessierungs- und Netzwerk Übertragungsdauer.

3 Bewegungsschätzung

Moderne Videokompressionsverfahren versuchen, die in natürlichen Videos oft vorkommende zeitliche *(temporale) Redundanz* zu reduzieren. Temporale Redundanz bedeutet, dass die in zwei aufeinanderfolgenden Frames enthaltenen Informationen oft ähnlich sind, beispielsweise bei einem Kameraschwenk nur etwas verschoben. Damit nicht in beiden Frames der selbe Inhalt komprimiert und übermittelt wird, werden Verfahren angewandt, um die temporale Redundanz beschreiben und somit entfernen zu können. Diese Bemühungen werden unter dem Begriff *Bewegungsschätzung* zusammengefasst (siehe Strutz [3]). Besonders für die Aufgabe des *Block Matchings* wurden für unterschiedliche Anwendungen bereits komplexe, stark optimierende Algorithmen gefunden. Block Matching ist das Problem, für Bildblöcke möglichst gut übereinstimmende Blöcke in einem Zielbild zu finden. Je besser dieses Problem gelöst wird, desto geringer ist die Differenz welche pro Frame übermittelt werden muss. Trivialerweise wird das Problem optimal gelöst, wenn für jeden Bildblock das komplette Zielbild durchsucht wird (*Full Search*). Für die meisten Anwendungen muss jedoch ein Kompromiss zwischen minimaler Differenz und geringer Rechenkomplexität (möglichst geringe Anzahl Blockvergleiche) gefunden werden. Für diese Entscheidung muss auch das Ziel-Szenario berücksichtigt werden: Ist es von zentraler Wichtigkeit, den Encoder auf schwacher Hardware betreiben zu können, oder spielt die Übertragungsleistung eine dominante Rolle?

3.1 Hierarchische Bewegungsschätzung

Als *hierarchische Bewegungsschätzung* bezeichnen wir eine Bewegungsschätzung, welche auf unterschiedlichen Grössenstufen des Videoframes ausgeführt wird. Beispielsweise wird mit jeder Stufenreduktion die Breite und Höhe des Bildes halbiert. Die hierarchische Bewegungsschätzung beginnt auf der kleinsten Stufe. Dort kann für das *Block-Matching* ein grosser Radius verwendet werden, ohne zu viele Berechnungen anstellen zu müssen. Danach wird das Verfahren auf der nächsthöheren Stufe des Bildes wiederholt, wobei die in der unteren Stufe gefundenen Bewegungsvektoren als mögliche Kandidaten gehandelt werden. Auf der höheren Stufe wird jedoch der Block-Matching Radius reduziert, um die Komplexität nicht zu stark zu erhöhen. Diese Schritte werden so oft wiederholt, bis die Stufe mit der originalen Bildgrösse erreicht wird. Das Verfahren ermöglicht ein gutes Block-Matching durch dank hierarchischem Vorgehen grossen Radius.

Ein Beispiel eines hierarchischen Algorithmus ist im Video-Codec Dirac implementiert. Ahtsham u.A. [4] analysieren in ihrer Arbeit diese Implementation und zeigen auf, dass beim dort eingesetzten Algorithmus die Suchpunkte des Block-Matching pro Block auf 121 (Radius 5), 81 (Radius 4), 49 (Radius 3), 25 (Radius 2) und 9 (Radius 1) reduziert werden, wobei aber mit jeder Skala die Anzahl zu vergleichenden Blöcke mit der Bildgrösse quadriert wird. In der höchsten Skala (Originalgrösse) wird also jeder Block nur noch um jeweils 1 Pixel verschoben, was zu 9 Blockvergleichen führt.

Problem: Der beschriebene Ansatz hat jedoch auch Schwächen, welche von Ahtsham u.A. [4] auch erkannt wurden. So wird die Qualität der Bewegungsschätzung zwar als sehr gut eingestuft, deren Berechnungskomplexität ist jedoch für Anwendungen wo ein performanter Encoder notwendig ist zu hoch. So müssen bei jedem Frame die n Bildstufen generiert werden und die Bewegungsvektoren sukzessive für alle Stufen ermittelt werden. Messungen am PMF Codec haben ergeben, dass die erste Implementation der Bewegungsschätzung mit hierarchischem Verfahren ca. 65% der benötigten Encoding-Zeit ausmachte.

3.1.1 Der 3DRS Algorithmus

Die Anzahl Berechnungen beim hierarchischen Algorithmus in der vorgestellten Version ist zu hoch für ein effizientes Encoding. Ahtsham u.A. [4] schlagen in ihrer Arbeit für Dirac einen auf dem *Three-Dimensional Recursive Search* (3DRS, eingeführt von De Haan u.A. [5]) basierten Algorithmus vor, bei welchem nicht mehr für jedes Inter-Frame der komplette hierarchische Vergleich

durchgeführt werden muss.

Der Begriff der *Group Of Pictures* (GOP) bezeichnet eine Folge von Frames zwischen zwei Keyframes. Das vorgeschlagene Schema sieht vor, dass für das erste Inter-Frame nach dem Keyframe jeder GOP für maximale Genauigkeit eine komplette hierarchische Bewegungsschätzung wie bisher durchgeführt wird. Die Bewegungsvektoren dieses Frames werden im Encoder gespeichert. Beim nächsten Frame wird dieses gespeicherte Array jeweils unter Verwendung des 3DRS Algorithmus aktualisiert. Dabei werden für jeden Block zuvor bereits berechnete Bewegungsvektoren (aus zeitlich vorhergehenden Frames bzw. innerhalb des aktuellen Frames bereits ermittelte Vektoren) als Referenz verwendet, um einen möglichst guten Vektor zu ermitteln. Abbildung 8 illustriert, welche Blöcke für den aktuellen Block X als Referenz gewählt werden können: A, B und C sind Blöcke aus dem aktuellen Frame, E bezeichnet den Block aus dem vorhergehenden Frame mit x- und y-Offset von 2.

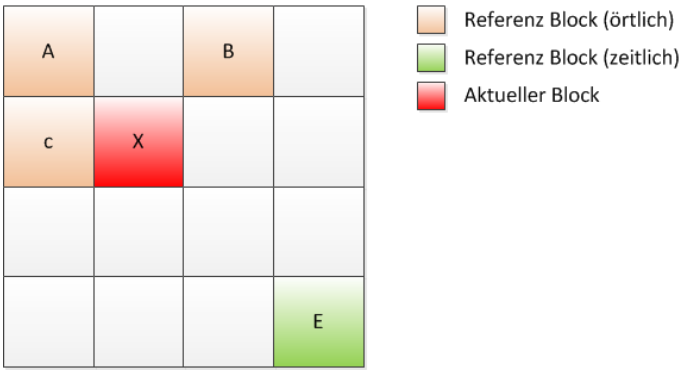

Abbildung 8: Die verwendeten örtlich und zeitlich versetzten Referenzblöcke des 3DRS

Aus den 5 erhaltenen Referenz-Bewegungsvektoren wird der beste gewählt. Von dessen Ziel aus wird dann einerseits ein einfacher *Diamond Search* mit nur 1 Pixel Radius ausgeführt, was zu 4 weiteren Differenzberechnungen führt (links, rechts, oben, unten). Zusätzlich kann noch eine örtlich geschätzte Vorhersage mit einbezogen werden (Bewegungen der umliegenden Blöcke berücksichtigen), was zu einer fünften Differenzbildung führt.

Ahtsham u.A. [4] haben in ihrer Arbeit eine Reduktion der nötigen Differenzbildungen um bis zu 60% erreicht, unter Beibehaltung der entstandenen Qualität.

4 Implementation PMF Version 3

In den folgenden Abschnitten gehe ich darauf ein, welche Probleme bei der Differenzbildung auftreten können, wie die Bewegungsschätzung und das Multi-Threading Konzept implementiert wurde.

4.1 Differenzbild Fehlervererbung

Bei der Kompression von Bilddaten wird meistens verlustbehaftete Kompression verwendet, da damit eine weit höhere Kompressionsrate erreicht werden kann als mit verlustfreier Kompression, ohne dass die Wahrnehmung der Qualität zu stark leidet. In dieser Arbeit verwenden wir das *Progressive Graphics File* (PGF) zur Kompression der Einzelbilder (siehe Stamm [1, 2]). PGF basiert auf der *Diskreten Wavelet Transformation* (DWT), gefolgt von einer mehr oder weniger aggressiven *Quantisierung*.

Die Verwendung von verlustbehafteten Kompressionsmethoden (durch Quantisierung) führt bei jedem Bild einen Fehler ein. In einem Video wird für die Reduktion von temporaler Redundanz die Differenz zwischen zwei Frames komprimiert und versandt. in Kapitel 3 haben wir gesehen, wie sich diese Differenz durch Bewegungsschätzung minimieren lässt. Der Fehler wird also durch die komprimierte Differenz zwischen Einzelbildern erzeugt:

$D_1 = F_1 - F_0$ wobei F_i die Input-Bilder bezeichnen. Dann ist $F_1 = F_0 + D_1$. Wenn die Differenz zudem verlustbehaftet komprimiert wird, gilt $F'_1 = F_0 + D'_1 \neq F_1$ (D'_1 sei die fehlerbehaftete Rekonstruktion der Differenz). Der in F'_1 enthaltene Fehler ist nicht weiter schlimm; die Parametrisierung des Codec wird so vorgenommen, dass dieser Fehler akzeptabel ist. In einem Video werden jedoch mehrere Bilder hintereinander versandt, welche durch ein Differenzbild aufeinander referenzieren. Eine Vererbung dieses eingeführten Fehlers von Frame zu Frame ist unbedingt zu vermeiden, da sich die Bildqualität ansonsten im Verlauf des Videos reduziert. Im Folgenden untersuchen wir wie es zu einer Vererbung kommen kann und wie sie vermieden wird.

Abbildung 9 zeigt schematisch die Verarbeitung der ersten drei Frames eines Videos. Das Blitz-Symbol markiert das Einführen eines Fehlers (durch Quantisieren, Q), in der Notation wird F'_0 aus F_0. Im Flowchart wird ersichtlich, dass sich ein Fehler weitervererbt, was die Qualität des empfangenen Bildes kontinuierlich reduziert: Wenn die Differenz mit dem oben erwähnten verlustbehafteten Referenzbild gebildet wird, entsteht daraus ein doppelter Fehler. Dieser Fehler wird durch die Differenzbildung nicht aufgehoben.

In Tabelle 1 wird die Fortplanzung des Fehlers für die ersten beiden Frames textuell dargestellt, wobei die Schritte DWT und Q als Funktion P zusammengefasst werden.

Tabelle 1: Fehlervererbung durch doppelte Quantisierung

Input	**Encoder**	**Decoder**
F_0	$f'_0 = P(F_0)$ Speichere $F'_0 = P^{-1}(f'_0)$	Speichere $F'_0 = P^{-1}(f'_0)$
F_1	$f'_1 - f''_0 = P(F_1 - F'_0)$ $F'_1 - F''_0 = P^{-1}(f'_1 - f''_0)$ Speichere $F''_1 = (F'_1 - F''_0) + F'_0 \neq F'_1$	$F'_1 - F''_0 = P^{-1}(f'_1 - f''_0)$ Speichere $F''_1 = (F'_1 - F''_0) + F'_0 \neq F'_1$

Lösung: Das Problem der sich fortpflanzenden Fehler kann behoben werden, in dem die Differenz im Frequenzbereich (das heisst in unserem Fall auf Basis der DWT-Koeffizienten) berechnet wird. Damit kann einer doppelten Quantisierung der Referenzbild-Koeffizienten vorgebeugt werden. Die Grafik in Abbildung 10 illustriert diesen vereinfachten Vorgang anhand der ersten drei Frames eines Videos.

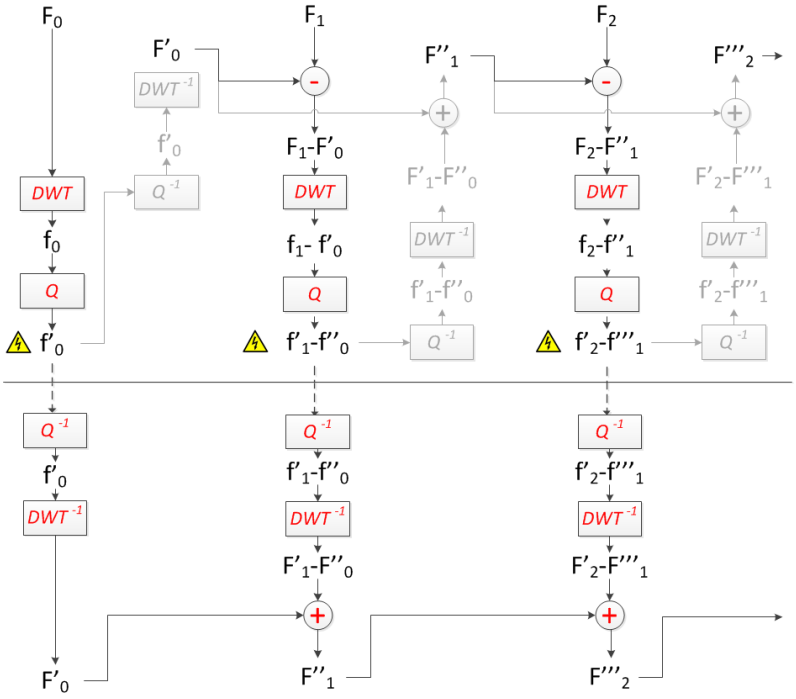

Abbildung 9: Fehlervererbung durch doppelte Quantisierung.

Die Vereinfachung in der Skizze weist bereits darauf hin, dass dieses Verfahren auch in der Tat eine rechnerisch einfachere Variante darstellt – durch die direkte Verwendung der Koeffizienten des Input-Bildes für die Differenz kann sich der Encoder das Bilden der Rekonstruktion (durch Rückwärtstransformation und Summe, in Abbildung 9 grau dargestellt) sparen. In Tabelle 2 wird analog zu oben aufgezeigt, dass die beschriebene Lösung der Koeffizienten-Differenz eine gehbare Variante ist, um die Vererbung des eingeführten Fehlers zu vermeiden: Ausschlaggebend ist, dass bei der Summe die Referenz-Teile identisch sind und sich somit wegkürzen lassen (im Beispiel f'_0).

Tabelle 2: Differenzbild-Erzeugung und Kompression korrigiert

Input	**Encoder**	**Decoder**
F_0	$f'_0 = P(F_0)$	$F'_0 = P^{-1}(f'_0)$
F_1	$f'_1 = P(F_1)$ $(f'_1 - f'_0)$	$f'_1 = (f'_1 - f'_0) + f'_0$ $F'_1 = P^{-1}(f'_1)$

Bemerkung: Während der Erarbeitung der oben gezeigter Lösung wurde ein Zwischenschritt implementiert, bei dem die Differenz zwar im Frequenzbereich gebildet wurde, die Rekonstruktion jedoch im Ortsraum. Dieses Vorgehen hat nebst rechnerisch höherer Komplexität auch das Problem, dass die Rekonstruktion nicht korrekt ist:

$$P^{-1}(f_1 - f_0) + F_0 \neq F_1$$

das heisst $P^{-1}(f_1 - f_0) \neq F_1 - F_0$, wobei $P^{-1}(f_0) = F_0$ und $P^{-1}(f_1) = F_1$.

Weitere Bemerkung: Wie erwähnt wird im erwähnten Verfahren die Differenz im Frequenzraum gebildet. Wenn die Differenz mit Hilfe von Bewegungsschätzung minimiert werden soll, muss dem Programmierer bewusst sein, dass die Koeffizienten im Frequenzraum nicht direkt mit den Bildpixeln des Ortsraums korrelieren – die Erkenntnisse aus klassischer Bewegungsschätzung müssen

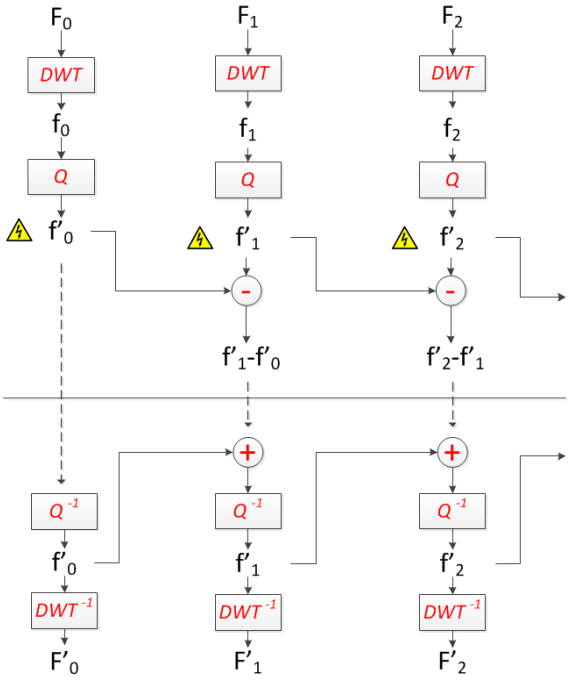

Abbildung 10: Konstanter Fehler durch Differenzbildung mit DWT-Koeffizienten

jeweils relativiert werden. Dieses Problem findet in der vorliegenden Arbeit leider keinen Platz mehr, bietet jedoch interessanten Stoff für weiterführende Projekte. Die aktuelle Implementation von PMF Version 3 berücksichtigt die Möglichkeit der Bewegungsschätzung, die Encoding-Komponente erstellt jedoch nur primitive Bewegungsvektoren, um die rechnerische Komplexität gering zu halten.

4.2 Bewegungsschätzung

Im letzten Semester wurde ein Vorschlag für die Implementierung von Bewegungsschätzung in unserem Video-Codec gemacht. Im Anhang A sind die damals vorgestellten Schemata für den PMF Video-Codec nochmals abgedruckt.

Wie wir im vorherigen Kapitel gesehen haben, ist es aus Performance- und Qualitätsgründen von Vorteil, wenn die Differenzbildung und somit auch die Bewegungsschätzung im Frequenzbereich durchgeführt werden. Die Abbildungen 11 und 12 zeigen die neuen Schemata für Encoder und Decoder. F bezeichnet jeweils ein im Ortsraum vorliegendes Bild, f' repräsentiert die (quantisierten) DWT-Koeffizienten und mit $m(f)$ notieren wir die bewegungskompensierte Version (d.h. ein gemäss Bewegungsvektor verschobener Bildblock) von f.

Die aktuelle PMF Version 3 implementiert genau die hier vorgestellten Schemata.

4.2.1 Minimierung der Bitrate

Wie in der Bemerkung im Abschnitt 4.1 erwähnt, ist die Minimierung der Bitrate noch nicht ausgeschöpft.

In Abbildung 13 wird die erreichte Bitrate des Vollbild-Codecs (MPGF) mit derjenigen von PMF v.3 verglichen. In der Abbildung ist zu sehen, dass PMF v.3 nur eine geringe Verbesserung gegenüber des Vollbild-Codecs aufweist. Durchschnittlich ist ein Vollbild-Frame in MPGF bei Qua-

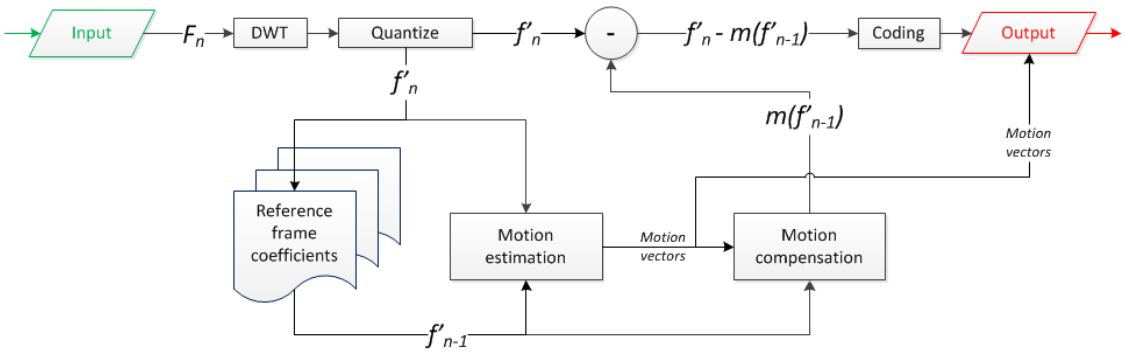

Abbildung 11: Aufbau des PMF v.3 Encoders

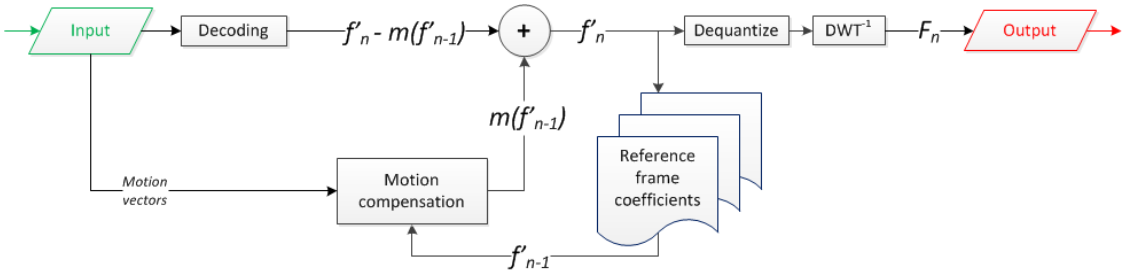

Abbildung 12: Aufbau des PMF v.3 Decoders

litätsstufe 5 rund 50 KByte gross, ein PMF Differenzframe ebenfalls noch 40 KByte plus einige Bytes für die komprimierten Bewegungsvektoren (siehe nächster Abschnitt).

Nachdem die Koeffizienten des Referenz- und des Eingabebildes voneinander subtrahiert wurden, sind die Werte zwar durchschnittlich nur noch ein Drittel so gross und das Histogramm der Daten zeigt wie erwartet eine Annäherung an den Nullpunkt, was für die Quantisierung der Werte von Vorteil ist.

Die Koeffizienten sind jedoch mit blosser Differenzbildung noch nicht ausreichend minimiert, was sich in der mangelhaften Kompressionsrate niederschlägt. Eine aufwändiger implementierte Bewegungsschätzung dürfte noch kleinere Werte erreichen, was zu einer weiteren Reduktion der Bitrate führt.

4.2.2 Kompression der Bewegungsvektoren

Um die im Differenzbild berücksichtigte Bildbewegung zu kompensieren, werden die im Encoder berechneten Bewegungsvektoren im Video-Datenstrom mit versandt. Jeder Bildblock besitzt einen Bewegungsvektor, welcher wiederum eine x- und eine y-Komponente enthält, welche den Pixel-Offset als Richtungsvektor darstellen. Die Anzahl Vektoren ist

$$\frac{W}{W_b} * \frac{H}{H_b}$$

wobei W und H die Bildbreite bzw. -Höhe bezeichnen und W_b bzw. H_b die Dimensionen der verwendeten Blöcke. Ein Beispiel illustriert den Aufwand für das Versenden der Vektoren ohne Kompression (S_d sei die Grösse des verwendeten Datentyps in Byte):

$$\frac{W}{W_b} * \frac{H}{H_b} * 2 * S_d = \frac{1280}{8} * \frac{720}{8} * 2 * 1\text{B} = 28\,800\text{B}$$

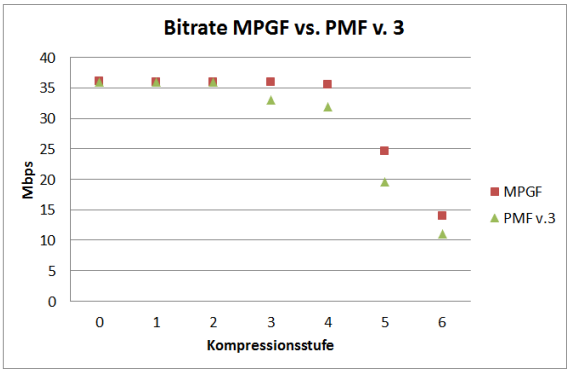

Abbildung 13: Vergleich der entstehenden Bitrate zwischen MPGF (Vollbild) und PMF v.3 (Differenzbild)

In diesem Beispiel wurde 1 Byte (*char*) pro Vektorkomponente verwendet. Ohne Zweifel sind 29 KByte noch zu viel – in der aktuellen Version wird die Bildgrösse durch die involvierten Vektoren nur um rund 10 KByte reduziert, was demnach einer Erhöhung der resultierenden Bitrate gleich käme.

Um die Vektoren so stark wie möglich zu komprimieren, wird einerseits die verwendeten Anzahl Bits pro Wert limitiert und andererseits ein verlustloser Kompressionsalgorithmus verwendet.

Reduktion der Anzahl verwendeter Bits

Wenn wie im oberen Beispiel ein ganzes Byte für einen Wert verwendet würde, entspräche dies einer Verschiebung von +/- 127 Pixel in beide Dimensionen. Eine Bewegung zwischen zwei Frames dürfte jedoch (je nach Framerate und Video-Inhalt) kaum so gross ausfallen. Der mögliche Wertebereich wurde deshalb in der aktuellen Version von PMF auf +/- 15 reduziert. Dieser vorzeichenbehaftete Wertebereich ist mit 5 Bits darstellbar. Da in C/C++ keine primitiven Datentypen mit weniger als 1 Byte existieren, muss das zu übertragende Byte-Array schrittweise auf Bit-Stufe abgefüllt werden. Dazu wird über das Array iteriert, wobei bei jeder Iteration 5 Werte abgefüllt werden können (das kleinste gemeinsame Vielfache von 5 und 8 Bits ist 40, daher stimmen die Grenzen der einzufüllenden 5-Bit Werte und diejenige der Ziel-Bytes nach 40 Bits wieder überein).

Die 5 Werte eines Schritts werden folgendermassen in das Byte-Array abgefüllt:

```
// 5 bits for x0, left 3 bits for y0.
MVs[index++] = (x[0] << 3) | ((y[0] >> 2) & 0x7);

// 2 bits from y0 left, followed by x1, this leaves 1 bit for y1
MVs[index++] = (y[0] << 6) | ((x[1] << 1) & 0x3E) | ((y[1] >> 7) & 0x1);

// 4 bits from y1 left, this leaves 4 bits for x2.
MVs[index++] = (y[1] << 4) | ((x[2] >> 1) & 0xF);

// 1 bit from x2 left, 5 bits from y2, this leaves 2 bits for x3
MVs[index++] = (x[2] << 7) | ((y[2] << 2) & 0x7C) | ((x[3] >> 3) & 0x3);

// 3 bits from x3 left, this leaves 5 bits for y3, and that's it!
MVs[index++] = (x[3] << 5) | (y[3] & 0x1F);
```

Beim Empfänger müssen die Werte nach der Dekompression dann entsprechend erneut "ausgepackt" werden:

```
Input[index].xDiff = (MVs[i*5+0] >> 3);
Input[index].yDiff = (MVs[i*5+0] << 5) | ((MVs[i*5+1] >> 3) & 0x18);
Input[index].yDiff = Input[index].yDiff >> 3;
index++;

Input[index].xDiff = (MVs[i*5+1] << 2);
Input[index].xDiff = Input[index].xDiff >> 3;
Input[index].yDiff = (MVs[i*5+1] << 7) | ((MVs[i*5+2] >> 1) & 0x78);
Input[index].yDiff = Input[index].yDiff >> 3;
```

```
index++;

Input[index].xDiff = (MVs[i*5+2] << 4) | ((MVs[i*5+3] >> 4) & 0x8);
Input[index].xDiff = Input[index].xDiff >> 3;
Input[index].yDiff = (MVs[i*5+3] << 1);
Input[index].yDiff = Input[index].yDiff >> 3;
index++;

Input[index].xDiff = (MVs[i*5+3] << 6)  | ((MVs[i*5+4] >> 2) & 0x38);
Input[index].xDiff = Input[index].xDiff >> 3;
Input[index].yDiff = (MVs[i*5+4] << 3);
Input[index].yDiff = Input[index].yDiff >> 3;
index++;
```

Mit der Reduktion von 8 auf 5 Bits pro Wert würde beim oben erwähnten Beispiel die genutzte Datenmenge von 28.8 auf 18 KByte schrumpfen. Dies soll mit Hilfe eines Kompressionsalgorithmus noch weiter reduziert werden.

Verlustlose Kompression der Werte

Für die Verwendung in einem Video-Codec muss der genutzte Kompressionsalgorithmus rechnerisch geringe Komplexität aufweisen. Wir möchten gute Resultate mit geringstmöglichem Aufwand erreichen. Der lz4 [1] Algorithmus ist auf hohe Geschwindigkeit bei verlustloser Kompression optimiert. Das lz4-Format wird in einem Blog-Post von Collet [6] beschrieben.

Messungen haben gezeigt, dass sich durch die Verwendung von lz4 die Grösse der Bewegungsvektoren beim erwähnten Beispiel im Schnitt auf 148 Bytes reduzieren lassen, was einer Reduktion um Faktor 122 entspricht. Diese Grösse ist nun akzeptabel, da sie die Grenze klar unterschreitet, ab welcher der Einsatz von Bewegungsschätzung eine Reduktion der Bitrate zur Folge hat.

4.3 Decoder Multi-Threading

Um die Leistung moderner Rechner ausnutzen zu können, ist die Nutzung mehrere CPU-Kerne unumgänglich. Auch in bisherigen Versionen war das PMF Videocodec bereits fähig für *Multi-Threading*. In der aktuellen Version wird jedoch eine zusätzliche Schwierigkeit eingeführt: die Abhängigkeit zwischen den Videoframes. Da zur Ausnutzung der temporalen Redundanz die Differenz zwischen den einzelnen Frames berechnet und übertragen wird, ist es notwendig dass dem Empfänger für die Rekonstruktion jeweils das vorhergehende Bild zur Verfügung steht. Wegen dieser Einschränkung können einzelne Frames nicht parallel verarbeitet werden.

Diese Abhängigkeit wird aber jeweils aufgelöst, sobald ein *Key Frame* in den Stream eingefügt wird, also eine neue GOP beginnt. Eine einzelne GOP enthält in sich Abhängigkeiten, welche aber nicht über seine Grenzen hinaus reichen. Wir können demnach eine einzelne GOP als kleinstmögliche Entität betrachten, welche sequentiell decodiert werden muss. Daraus folgt, dass mehrere komplette GOPs parallel verarbeitet werden können.

In Abbildung 14 wird an einem vereinfachten Beispiel das Prinzip des Konzepts aufgezeigt. In dieser Grafik werden GOPs von lediglich fünf Frames verwendet und nur die ersten 20 Frames dargestellt um die Lesbarkeit zu erhöhen. Input- und Output-Management müssen dafür sorgen, dass die Datenpakete die korrekte Decoder-Komponente erreichen bzw. in der richtigen Abspiel-Reihenfolge ausgegeben werden. Beispielsweise wird Frame 5 vor Frame 2 fertiggestellt, darf aber natürlich nicht in dieser Reihenfolge in den Ausgabe-Strom geschrieben werden.

Die Implementation dieses Konzepts hat gezeigt, dass sich damit die verfügbaren CPU-Kerne optimal ausnutzen lassen, Abbildung 15 zeigt ein Screenshot der Analyse des Multi-Threading Verhaltens durch Microsoft Visual Studio 2010 bei vier aktivierten Threads. Die grünen Bereiche stellen Ausführungzeit dar, rot ist Warte- und Synchronisationszeit. In den untersten vier Zeilen, welche die instanzierten Worker Threads darstellen, ist eine hohe Parallelität an Ausführungszeit erkennbar.

[1] http://code.google.com/p/lz4/

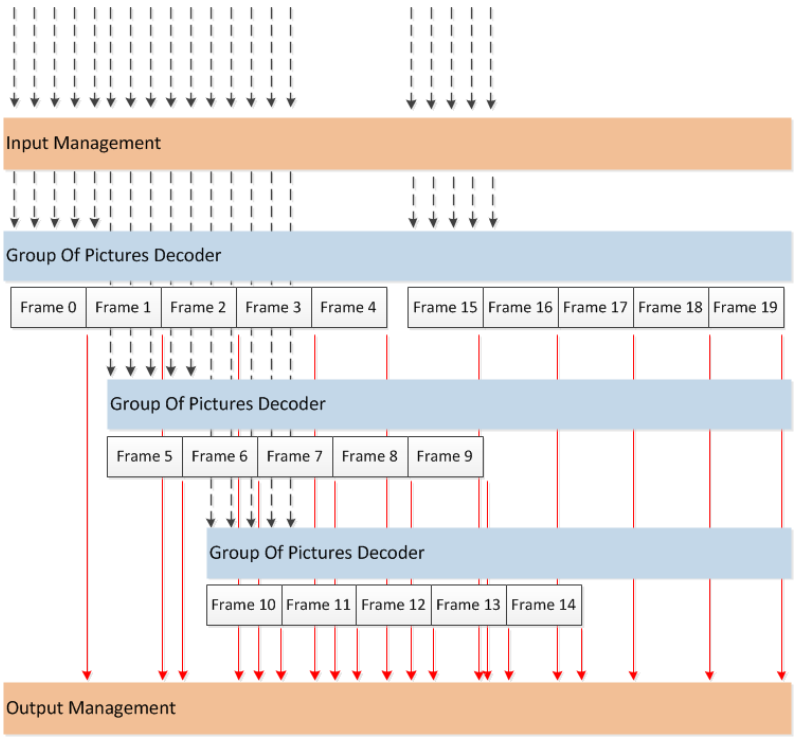

Abbildung 14: Eine vereinfachte Darstellung des Decoder Multi-Threading Konzepts

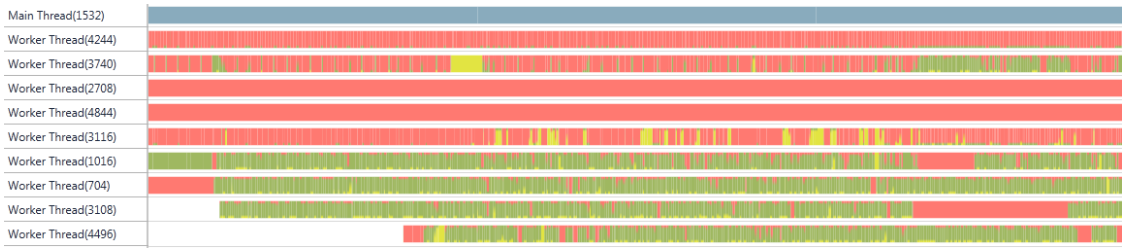

Abbildung 15: Screenshot aus dem Visual Studio 2010 Profiler

4.4 Layer-Dropping Adaptivitäts-Schema

Bei der Wiedergabe von digitalen Videodaten ist die Leistung der beteiligten Komponenten oft variabel. Beispielsweise kann die CPU des Empfängers je nach Nutzung des Rechners unterschiedlich stark ausgelastet sein, oder auf dem Empfangskanal können Störungen (*Packet loss*) oder Verzögerungen auftreten, insbesondere bei Übertragung über das Internet und/oder kabellose Verbindungen.

Um diesen Umständen entgegen zu wirken, ist es erforderlich, dass ein Video-Codec sich während der Wiedergabe an den geänderten Kontext anpassen kann (*Adaptivität*). Um dies zu bewerkstelligen gibt es im wesentlichen zwei verschiedene Ansätze, wie sie auch von Cuetos u.A. diskutiert werden [7]:

- Es können mehrere Streams des Videos in unterschiedlicher Qualität angeboten werden, zwischen welchen der Empfänger automatisch oder manuell umschalten kann. Diese Methode erfordert typischerweise ein relativ hoher Preprocessing Aufwand, da das Video in den unterschiedlichen Qualitätsstufen bereitgestellt werden muss.

- Alternativ kann ein einzelner Datenstrom genutzt werden, um das Video in unterschiedlicher Qualität darzustellen. Der Empfänger entscheidet hierbei, wie viel der empfangenen Daten er für die Wiederherstellung des Videos verwenden kann. Man spricht von *Layers*, deren verarbeitete Anzahl dynamisch reduziert oder erhöht werden kann. Sofern kein Rückkanal zum Absender besteht, muss bei dieser Methode jedoch das Video in der höchsten Qualität übertragen werden, auch wenn der Empfänger einige Layers ignoriert.

Durch die Verwendung von PGF für die Einzel-/Differenzbild-Kompression liegt die zweite Methode näher. PGF verarbeitet die Bilddaten in einer *Multiskalen-Analyse* schrittweise. Dieser Vorgang lässt eine effiziente Aufteilung in unterschiedliche Qualitätsstufen zu, da bei der DWT-Synthese das Bild stufenweise rekonstruiert werden kann. Der Empfänger kann bei der Verarbeitung eines Video-Frames so viele Skalen rekonstruieren wie Zeit bleibt – sobald die Zeit abgelaufen ist, wird die beste verfügbare Skala dargestellt.

4.4.1 Qualitätsreduktion mit Differenzübertragung

Bei falsch angewandter Qualitätsreduktion kann es bei Verwendung von Differenzbildern zu unerwünschten Bildfehlern kommen. In Abschnitt 4.1 haben wir gesehen, dass bei der ersten Methode die Differenzbildung im Ortsraum pixelweise stattfindet. Wie erwähnt führt dies dazu, dass sich ein Fehler von Frame zu Frame weitervererben kann. Wird der Vorgang der Rekonstruktion aber frühzeitig abgebrochen um Zeit zu sparen, so muss für die Summe eine noch stärker fehlerbehaftete Version einer niedrigeren Skala verwendet werden.

In Bildteilen mit starker Veränderung äussert sich dieser Fehler stärker als in Bereichen mit geringer Differenz. Für den Betrachter des Videos wirkt ein solcher Fehler stärker störend als eine homogene Qualitätsreduktion des gesamten Bildes.

In der aktuellen Version von PMF wird dieses Problem umgangen, da nicht die Rekonstruktion eines Bildes für die Differenzbildung verwendet wird, sondern dessen DWT-Koeffizienten.

5 Anpassungen am Bildcodec

Um das PMF v.3 Video-Codec in dieser Form realisieren zu können, mussten am zugrunde liegenden Bildkompressions-Codec einige Änderungen vorgenommen werden. Zu diesem Zweck wurde der Fork *PMFCodec* des etablierten PGFCodec (Progressive Graphics File) erstellt. Abbildung 16 listet die neu eingefügten Methoden und die Abhängigkeiten auf.

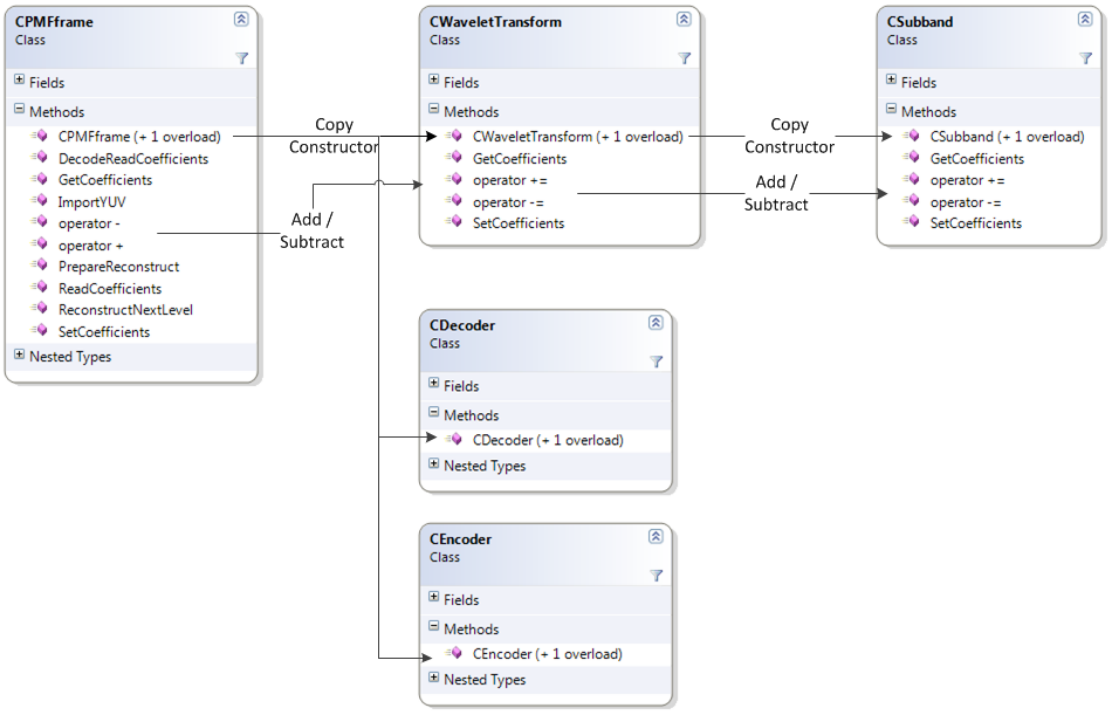

Abbildung 16: Die neu eingeführten Methoden im PMFCodec

- **Copy Konstruktoren:** Um CPMFframe Objekte kopieren zu können, so wie wir es im Video-Codec benötigen, wurden auf allen Hierarchien Copy Konstruktoren eingefügt, welche es zulassen, die Objekte als *Deep Copy* zu kopieren.
- **Addition/Subtraktion:** Für die Differenzbildung auf Koeffizienten-Basis ist das Verwenden von überladenen Additions- und Subtraktions-Operatoren eine elegante Methode. Die Operatoren wurden auf allen Stufen implementiert, bis am Schluss in der CSubband Klasse die effektive Operation auf dem primitiven Typ INT16 ausgeführt wird. In der aktuellen Version werden stattdessen jedoch die Set/GetCoefficients Methoden verwendet, um die Differenzbildung unter Berücksichtigung der Bewegungsschätzung durchführen zu können.
- **Set/GetCoefficients:** Die Implementation der Bewegungsschätzung soll nicht innerhalb des PMFCodec realisiert werden. Deshalb bieten diese Methoden die Möglichkeit, direkt auf die DWT-Koeffizienten der CPMFframe Objekte zugreifen zu können.
- **ReadCoefficients:** Füllt die Subbänder mit Koeffizienten aus dem Eingabestrom ohne sie zu dequantisieren.
- **Reset:** Wenn statt ImportYUV/Bitmap direkt auf die Koeffizienten zugegriffen wird, kann diese Methode verwendet werden, um die Dimensionen und das aktuelle Level zurückzusetzen.

- **PrepareReconstruct**, **ReconstructNextLevel:** Levelweise rekonstruieren des Bildes. Verhindert die Zerstörung der gelesenen Koeffizienten, da auf Kopien der CWaveletTransform Objekte rekonstruiert wird.

6 Reflexion

Ich durfte in diesem Semester eine Menge unterschiedliche Aufgaben in Angriff nehmen: der erste Teil war eher theoretischer Natur durch die Erarbeitung der optimalen Anzahl zu übertragender Bildteile und des Verfassens des Paper Drafts. Das Schreiben des Paper Drafts stellte eine grosse Herausforderung dar, weil es absolutes Neuland für mich war. Das Schreiben zwang mich aber dazu, mich nochmals intensiv mit dem Kontext meiner Arbeit zu beschäftigen und den aktuellen Markt zu untersuchen, wovon ich bestimmt profitieren konnte.

Später im Semester habe ich das Konzept für die Implementation der Bewegungsschätzung erarbeitet. Auch dort musste ich mir zuerst die theoretischen Grundlagen aneignen: Beim Studium der in Paper-Draft und vorliegender Dokumentation zitierten Papers erweiterte ich mein Wissen über die Thematik. Danach konnte ich in den Quellcode einiger quelloffener Video-Codecs eintauchen. Ich stellte fest, dass zwar auch dort nicht alles Gold ist, was glänzt, aber die Komplexität eines ausgereiften Systems wie Dirac oder h.264 hat dennoch bleibenden Eindruck hinterlassen.

Bei der ersten Programmierphase implementierte ich das bisher nur konzeptuell erarbeitete Schema zur hierarchischen Bewegungsschätzung. Wie üblich erwarten den Programmierer dann eine Menge Herausforderungen, welche bei der Planungsphase noch keine Beachtung fanden. Die in dieser Dokumentation erläuterten Probleme der Fehlervererbung traten auf, und nach intensiver Suche konnten wir zeigen, dass die Rekonstruktion einer Koeffizienten-Differenz nicht das gleiche Ergebnis ergibt wie die (Ortsraum-)Differenz der Rekonstruktionen. Leider ist mir bis jetzt nicht klar weshalb dies so ist. Ein studentisches Nachfolgeprojekt könnte diese Problemstellung unter Umständen beantworten.

In der letzten Phase des Projektes wurde die vorliegende PMF Version 3 implementiert, wie sie in dieser Dokumentation beschrieben wird. Zu Lasten von hierarchischer Bewegungsschätzung wurde die Differenzbildung mit einer primitiven Bewegungsschätzung auf DWT-Koeffizienten implementiert. Dadurch, dass die Koeffizienten nicht direkt mit dem Bild im Ortsraum korrelieren, können die klassischen Algorithmen der Bewegungsschätzung nicht ohne weiteres angewandt werden. Eine spannende Herausforderung, welche in diesem Projekt keinen Platz mehr fand, ist die Abschätzung des Potentials und die Optimierung dieser Bewegungsschätzung.

7 Ehrlichkeitserklärung

Der Autor des vorliegenden Dokuments, Stefan Weber, bestätigt hiermit, das Informatik-Projekt 9 am Institut für Mobile und Verteilte Systeme nach bestem Gewissen unter Einhaltung aller gebotenen Regeln durchgeführt zu haben.

Der Autor bestätigt weiter, dass keine Teile der Dokumentation oder des geschriebenen Quellcodes unrechtmässig kopiert wurden oder gegen schweizerisches Recht verstossen.

20. Februar 2012,

Stefan Weber

Literatur

[1] C. Stamm. PGF: Progressive Graphics File, version 6. http://www.libpgf.org. Zugegriffen am 4. November 2011.

[2] C. Stamm. PGF: A new progressive file format for lossy and lossless image compression. In *Journal of WSCG*, volume 10(2), pages 421–428, 2002.

[3] T. Strutz. *Bilddatenkompression: Grundlagen, Codierung, Wavelets, JPEG, MPEG, H.264, Band 3.* Vieweg Praxiswissen, 2005.

[4] A. Ali, S.F. Ali, N. Khan, and S. Masud. Performance improvement in motion estimation of dirac wavelet based video codec. In *Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on*, pages 764 –769, sept. 2009.

[5] G. de Haan, P.W.A.C. Biezen, H. Huijgen, and O.A. Ojo. True-motion estimation with 3-d recursive search block matching. *Circuits and Systems for Video Technology, IEEE Transactions on*, 3(5):368 –379, 388, oct 1993.

[6] Y. Collet. Real time data compression: Lz4 explained. http://fastcompression.blogspot.com/2011/05/lz4-explained.html. Zugegriffen am 18. Februar 2012.

[7] P. De Cuetos, D. Saparilla, and K.W. Ross. Adaptive streaming of stored video in a tcp-friendly context: Multiple versions or multiple layers? In *in Proc. Packet Video Workshop*, 2001.

A PMF Version 1 Schemata

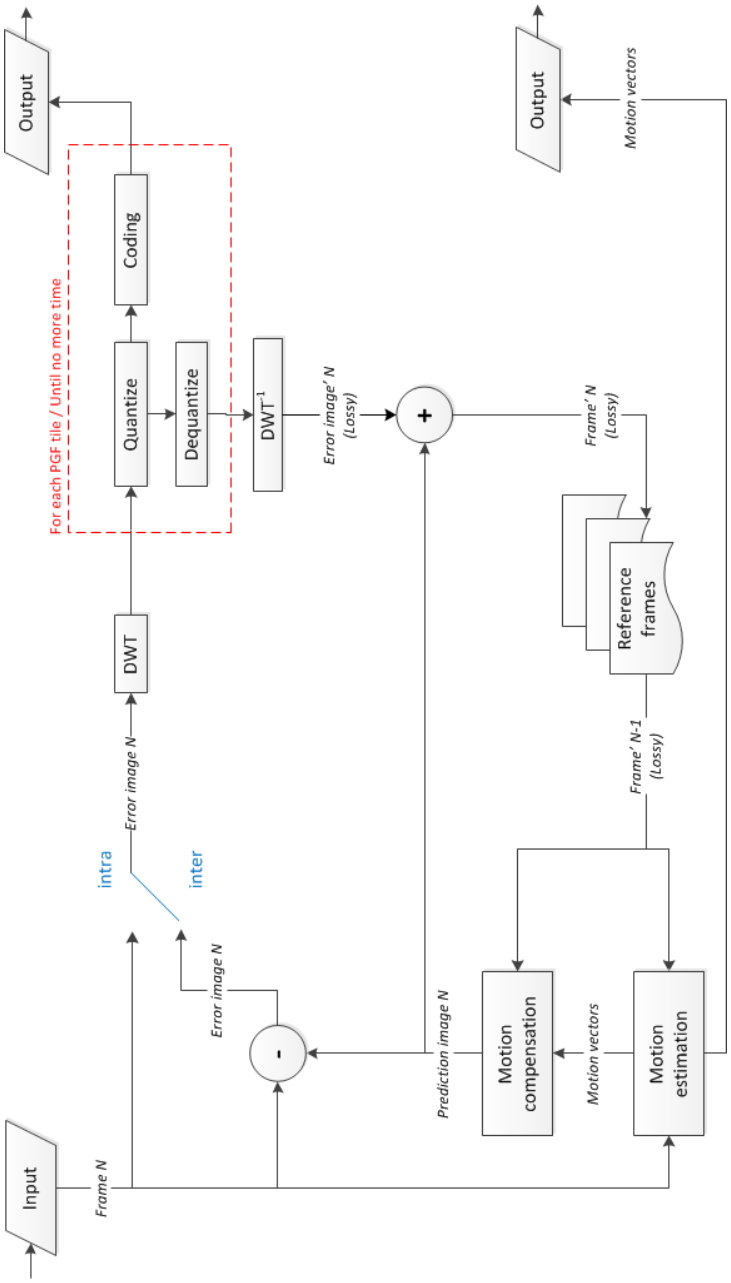

Abbildung 17: Flowchart Schema des PMF Encodings

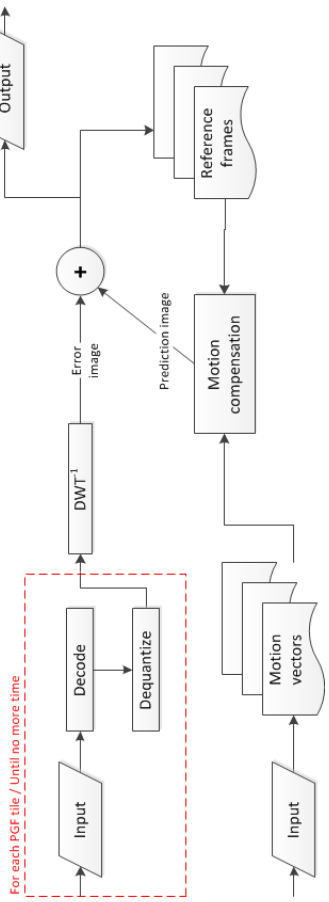

Abbildung 18: Flowchart Schema des PMF Decodings

B Paper-Draft