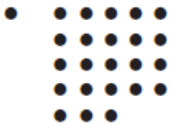


Fachhochschule Köln
Cologne University of Applied Sciences

Masterarbeit Medientechnologie

Deinterlacing und zeitliche Skalierung im Umfeld der UHD Konvertierung

vorgelegt von

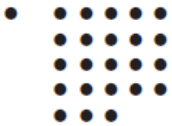
Daniel Schwingenheuer

Mat.-Nr. 11068139

Erstgutachter: Prof. Dr. Klaus-Dieter Ruelberg (Fachhochschule Köln)

Zweitgutachter: Prof. Dr. Michael Silverberg (Fachhochschule Köln)

Dezember 2014



Fachhochschule Köln
Cologne University of Applied Sciences

Masterthesis Media Technology

Deinterlacing and temporal Upscaling regarding UHD Conversion

submitted by

Daniel Schwingenheuer

Mat.-Nr. 11068139

First Reviewer: Prof. Dr. Klaus-Dieter Ruelberg (Cologne University of Applied Sciences)

Second Reviewer: Prof. Dr. Michael Silverberg (Cologne University of Applied Sciences)

December 2014

Kurzfassung

Masterarbeit

Titel: Deinterlacing und zeitliche Skalierung im Umfeld der UHD Konvertierung

Gutachter:

- Prof. Dr. Klaus-Dieter Ruelberg (Fachhochschule Köln)
- Prof. Dr. Michael Silverberg (Fachhochschule Köln)

Zusammenfassung:

Diese Arbeit untersucht die zeitliche Interpolation von Videosignalen. Durch die Einführung des Ultra High Definition Television Standard wird eine höhere Bildwiederholrate benötigt um Judder-Artefakte zu vermeiden. In diesem Zusammenhang kommt der zeitlichen Interpolation von nicht nativem UHD Material eine besondere Bedeutung zu. Neben der räumlichen Interpolation ist diese essenziell. In dieser Arbeit wird eine Übersicht über die Anforderungen an UHD Video gegeben. Bewegungskompensierende Verfahren für das Deinterlacing und Upscaling in Verbindung mit einer Phasenkorrelation werden behandelt. Ein MATLAB-Programm wird entwickelt mit dem ein Upscaling und Deinterlacing von HD Material durchgeführt werden kann.

Stichwörter: UHD, HD, interlaced, progressive, Deinterlacing, Upscaling, Phasenkorrelation, Bewegungsschätzung

Datum: 06. Dezember 2014

Abstract

Master Thesis

Title: Deinterlacing and temporal Upscaling regarding UHD Conversion

Reviewers:

- Prof. Dr. Klaus-Dieter Ruelberg (Cologne University of Applied Sciences)
- Prof. Dr. Michael Silverberg (Cologne University of Applied Sciences)

Abstract:

This thesis investigates the temporal interpolation of video. Due to the introduction of Ultra High Definition Television Standard, a higher framerate is needed to prevent sequences from judder. In this context, the temporal interpolation of non-native UHD video deserves a deep analysis. Besides the spatial upconversion, temporal upscaling is essential. This thesis gives an overview over the current situation and the challenging introduction of UHD. Moreover it describes motion compensating deinterlacing and upscaling methods consindering phase correlation for motion estimation. Finally, a MATLAB programm is developed to realise a deinterlacing and upscaling of HD video.

Keywords: UHD, HD, interlaced, progressive, deinterlacing, upscaling, phase correlation, motion estimation

Date: 6th December 2014

Inhaltsverzeichnis

1 Einleitung

In der Hoffnung auf viele friedlich farbige aber auch spannend farbige Ereignisse. Über die zu berichten und die darzustellen sich lohnt. [...] Gebe ich jetzt gewissermaßen den Startschuss für das deutsche Farbfernsehen.

Willy Brandt, IFA 25.08.1967 [25]

Mit diesen Worten leitete der damalige Bundesaußenminister Willy Brandt die vielleicht größte Revolution im deutschen Fernsehen ein. Zum ersten Mal in der Geschichte wurde in der Bundesrepublik Deutschland ein TV-Programm in Farbe ausgestrahlt.
Heute, 47 Jahre nach diesem fernsehgeschichtlich herausragenden Ereignis, blickt dieses Medium auf eine Erfolgsgeschichte zurück, begleitet von kleineren und größeren technischen Revolutionen. Die Umstellung auf digitales Fernsehen und die Einführung des High Definition Standards sind beispielhaft für die Suche nach immer besserer technischer Qualität und einem realitätsnahen Seherlebnis.
Doch diese Suche ist mit dem bisher Erreichten lange nicht abgeschlossen. Längst stehen neue Entwicklungen vor der Vollendung und sind bereit für den Wettbewerb in dem Maßstäbe ständig neu definiert werden.
Auch aktuell findet einmal mehr eine Revolution auf dem Gebiet der Fernsehtechnik statt. In nicht allzu ferner Zukunft soll der Ultra-High Definition (Ultra-HD) Standard für den Konsumenten verfügbar werden. Die Firma Sony testete bereits während des Fußball-Confederations Cup 2013 ihre Systeme für eine Übertragung [21].
Die Einführung dieser neuen TV-Generation birgt an vielen Stellen Probleme, die es zu lösen gilt. So muss zum Beispiel für eine reibungslose Einführung sichergestellt werden, dass auch Material, das älteren Standards entspricht, in den neuen Standard konvertiert werden kann.

Im Bezug auf die Einführung von Ultra HD und die Kompatibilität zu HD ergeben sich somit zwei große Problemfelder. Möchte man HD Material auf einem Ultra HD System wiedergeben, so müssen zum einen die fehlenden örtlichen Pixel eines Bildes berechnet werden, um die entsprechenden Bildergrößen zu erlagen. Zum anderen spielt bei größeren Pixelzahlen eine weitere Komponente eine immer bedeutendere Rolle: Die Bildwiederholrate. Konnte man bei SD und HD mit 25 Bildern pro Sekunde im *progressive*, bzw. 50 Bildern pro Sekunde im *interlaced* Verfahren ruckelfreie Bilder erzeugen, so wird in Zukunft diese zeitliche Auflösung deutlich erhöht werden müssen. Andernfalls geht die

Qualitätssteigerung durch die Pixelsteigerung auf Grund von Bewegungsunschärfe verloren.
Mit dieser Arbeit wird in die erwähnte Problematik der zeitlichen Skalierung von Videosignalen eingeführt. In Kapitel 2 wird zunächst eine Übersicht über die historische Entwicklung, insbesondere des *Interlacing*, gegeben, bevor in Kapitel 3 die dadurch resultierende Ausgangssituation und die Problemstellung bezüglich der Kompatibilität herkömmlicher TV Standards mit dem neu definierten Ultra-HD Standard beschrieben werden. Daraus folgt eine Definition der durch diese Arbeit behandelten Themengebiete.
Kapitel 4 vollzieht eine Frequenzanalyse des *Interlacings* und entwickelt somit die Voraussetzungen für ein *Deinterlacing* und *Upscaling* im weiteren Verlauf der Arbeit. Bevor diese Grundlagen in Kapitel 8 konkret angewendet werden behandeln die Kapitel 5, *Motion Estimation*, Kapitel 6, *Deinterlacing*, und Kapitel 7, *Temporal Upscaling*, die Herangehensweise für die einzelnen Schritte der Konvertierung. Kapitel 9 beschreibt die Rahmenbedingungen für das in dieser Arbeit entwickelte Programm, indem es Hard- und Softwarekonfiguration und -benutzung erläutert. In Kapitel 10 ist der für diese Arbeit in MATLAB programmierte *Media Converter* dokumentiert. Einige der in den theoretischen Kapiteln behandelten Verfahren sind an dieser Stelle praktisch umgesetzt. Eine Konvertierung im Zeitbereich lässt sich somit exemplarisch für Eingangsmaterial im Zeilensprungverfahren von $25Hz$ auf bis zu $120Hz$ im Vollbildmodus skalieren.
Kapitel 11 wertet die mit dem *Media Conveter* erzielten Versuchsergebnisse aus und beurteilt die Qualität der benutzen Verfahren. Abschließend erfolgt in Kapitel 12 eine Zusammenfassung. Außerdem werden mögliche Themen für die Vertiefung angesprochen.

Köln, 06.12.2014

2 Historische Entwicklung

Zu Beginn der Entwicklungsarbeiten für ein Fernsehsystem waren die Anforderungen für eine hinreichende visuelle Qualität, als auch systembedingte technische Einschränkungen die limitierenden Parameter. So wurde beispielsweise in Deutschland eine *Frame Rate* von $25Hz$ gewählt, die der halben Stromnetz-Frequenz entspricht. Dadurch wurden störenden Interferenzen im TV-Gerät verhindert und Problemen mit der Beleuchtungsfrequenz bei der Aufnahme vorbeugt[20]. Das menschliche Auge nimmt Bewegungen ab einer *Frame Rate* von ungefähr 15 Bildern pro Sekunde als fließend wahr. Mit den gewählten $25Hz$ wird dieser Wert deutlich überschritten.
E.W. Engstrom [8] zeigte bereits 1935, dass das menschliche Auge sehr empfindlich gegenüber großflächigem Flimmern ist. Erst ab einer Bildwiederholrate von $40Hz$ wird Großflächenflimmern nicht mehr wahrgenommen. Diese Grenze darf bei einem TV-Standard keinesfalls unterschritten werden. Außerdem wurde vorgeschlagen, die *Frame Rate* an die Netzfrequenz zu koppeln und diese somit in Deutschland auf $50Hz$ festzulegen. Da zu diesem Zeitpunkt der Entwicklung die Bandbreite der übertragenden Systeme nicht ausreichte um 50 Vollbilder zu übertragen, musste ein Kompromiss gefunden werden. Da Engstrom außerdem beschrieb, dass das Großflächenflimmern sehr viel negativer wahrgenommen wird als das Flackern einzelner Zeilen, wurde das *Interlacing* entwickelt.
Nachdem sie zum Leuchten gebracht wurden, benötigten die Pixel der zu diesem Zeitpunkt benutzten CRT-Displays (*Cathode Ray Tube*) eine Abklingzeit, bis sie wieder in ihrem Ausgangszustand waren. Diese Eigenschaft führte zusammen mit den Ergebnissen von Engstrom dazu, dass in Folge nur noch Halbbilder übertragen wurden. Das erste Halbbild enthielt die ungeraden Zeilen (*odd Lines*) und das zweite die geraden Zeilen (*even Lines*). Auf dem CRT-Display wurden diese Bilder nacheinander angezeigt, wobei für das erste Halbbild die ungeraden Zeilen benutzt wurden und für das zweite Halbbild nur die geraden Zeilen. Durch das Abkling-Verhalten der einzelnen Bildpunkte sah der Zuschauer in statischen Bildpassagen sogar Vollbilder.
Durch diesen Trick war es möglich, trotz der eigentlich nur für 25 Bilder pro Sekunde ausreichenden Bandbreite, eine (Halbbild-)*Frame Rate* von $50Hz$ zu realisieren. Das Großflächenflimmern wurde durch diesen Schritt nicht mehr vom Zuschauer wahrgenommen.
Das *Interlacing* war zu Beginn des Fernsehens dafür verantwortlich, dass überhaupt mit einer ausreichenden Bildqualität Bewegtbilder übermittelt werden konnten. Allerdings brachte es auch einige Artefakte mit sich. So verringerte sich die vertikale Auflösung in bewegten Bildteilen, außerdem flackerten Zeilen, die so dünn waren, dass sie nur in einer einzelnen Zeile im Gesamtbild und nicht in beiden Halbbildern vorhanden waren. Darüber hinaus entstand *Moiré* an horizontalen Bildstrukturen bei bewegten Bildvorlagen [20].
Zunächst unabhängig von der Fernsehtechnik wurden für den Computermarkt, LCD- und Plasma-Displays entwickelt, die - um die oben genannten Nachteile zu vermeiden -

lediglich mit Vollbildern arbeiteten. Da Fernseh- und Computermarkt allerdings immer enger zusammen wuchsen, wurden Lösungen gesucht, auch *Interlaced* Signale darzustellen. Werden diese Sequenzen auf LCD- und Plasma-Displays wiedergegeben, so müssen beide Halbbilder wieder zusammengeführt werden. Diese geben jedoch einen unterschiedlichen Zeitpunkt einer Bewegung wieder. Das führt zu dem inakzeptablen Sägezahneffekt (*jigsaw*), bei dem die Kanten eines sich bewegenden Objektes ausfransen [16].
Da sich die flachen Displays im Computer- wie im TV-Markt wegen ihrer höheren Auflösung und ihres im Vergleich zu CRT-Displays geringen Gewichtes und Ausmaßes durchgesetzt haben, werden Videosequenzen heutzutage ausschließlich *progressive* wiedergegeben. Jedes Display führt, so es *interlaced* Signale bekommt, die zwei Halbbildern wieder zu einem Vollbild zusammen. Für dieses sogenannte *Deinterlacing* gibt es inzwischen unzählige Methoden und Varianten.

3 Ausgangssituation und Problemstellung

In diesem Kapitel werden zunächst die Parameter heutiger und zukünftiger TV-Standards erläutert. Durch den Vergleich von *High Definition* und *Ultra-High Definition* wird die in 3.3 beschriebene Problemstellung verdeutlicht. In dieser wiederum ist die Aufgabe und das Ziel dieser Arbeit definiert.

3.1 Systemparameter

Um ein Fernsehsystem zu charakterisieren, ist die Angabe einiger Parameter notwendig. Hierzu gehören die Anzahl der vertikalen Zeilen eines Vollbildes, die Art der Abtastung (*interlaced*, *progressive*) und die Anzahl der Vollbilder. Bei allen aktuellen TV-Systemen wird ein Seitenverhältnis von 16:9 vorausgesetzt (ausgenommen SDTV - Standard Definition Television). Tabelle 3.1 gibt eine Übersicht über die zurzeit in der TV Produktion vorhandenen Standard- und High Definition-Formate.

TV-System	**Parameter**
SDTV	576i/25
HDTV System 1	720p/50
HDTV System 2	1080i/25
HDTV System 3	1080p/25
HDTV System 4	1080p/50

Tabelle 3.1
SDTV- und HDTV-Formate [6][28]

Darüber hinaus hat die ITU in ihrer Recommendation ITU-R BT.2020 [12] bereits im August 2012 einige weitere TV-Formate standardisiert, die mit dem Namen *Ultra-HD* bezeichnet werden. Die möglichen Parameter sind in Tabelle 3.2 wiedergegeben.

Ultra-HD		
Bildgrößen	7 680 x 4 320	3 840 x 2 160
Frame Rate	120, 60, 60/1.001, 50, 30, 30/1.001, 25, 24, 24/1.001	
Scan Mode	*progressive*	

Tabelle 3.2
Ultra-HD [12]

Der Vergleich der beiden Tabellen zeigt deutliche Unterschiede der Systeme auf. Es wird mit der Einführung von Ultra-HD nicht nur die Pixelzahl massiv erhöht, sondern auch die maximale Bildwiederholrate wird bis zu $120Hz$ betragen. Darüber hinaus gibt es nur noch *progressive* abgetastetes Material.

3.2 Abtastung bei Ultra-HD

Die Gründe für die Wahl der Bildwiederholfrequenz von $25Hz$, bzw. $50Hz$ im *interlaced* Modus wurden bereits in Kapitel 2 beschrieben. Um zu verstehen, warum bei Ultra-HD viel höhere Taktraten eingeführt werden, muss der Blick auf einen weiteren Aspekt gerichtet werden: die Bewegungsauflösung.
Bewegt man eine Kamera während der Aufnahme oder bewegt sich ein Objekt in einer aufgenommenen Szene, so verringert sich in der Regel die Auflösung. Abbildung 3.1 zeigt ein Beispiel. Auf Grund der hohen Geschwindigkeit, werden die Flügel der Ente nicht mehr scharf dargestellt. Die Auflösung im Vergleich zum Rumpf der Ente verringert sich deutlich.

Abbildung 3.1
Bewegungsunschärfe

Dieses Phänomen ist durch die sogenannte Bewegungsunschärfe zu erklären und wird in Abbildung 3.2 verdeutlicht. Eine Kamera filmt einen Ball, der sich während der Aufnahme um einen Winkel α bewegt. Ist eine *Frame Rate* von $25Hz$ eingestellt, kann die Kamera jedes Einzelbild $1/25Hz = 40ms$ lang aufzeichnen. Durch die Bewegung des Balls wird er von mehr Pixeln als im Ruhezustand aufgezeichnet. Das Bild 'verschmiert' durch die lange Belichtungszeit auf dem Sensor - der Ball wirkt im Ergebnis unscharf.
Wird nun, wie bei Ultra-HD, die Pixeldichte auf dem Sensor extrem erhöht, so werden bei dem selben Experiment noch viel mehr Pixel den Ball aufzeichnen. Die ursprünglich viel höhere Pixelzahl auf dem Sensor und die dadurch mögliche Qualitätssteigerung wird durch die zunehmende Bewegungsunschärfe wieder verringert.
Abhilfe schafft in diesem Zusammenhang eine erhöhte Bildfrequenz. Bei $120Hz$ beispielsweise ist die Aufnahmezeit nur noch $8ms$ pro Bild. Bei gleicher Bewegung des Balls wird er auf deutlich weniger Pixel 'verschmiert' [11].

3.3 Problemstellung

Der Vergleich der in Absatz 3.1 dargestellten technischen Rahmenbedingungen der aktuell ausgestrahlten TV-Systeme SDTV und HDTV mit dem zukünftigen TV-Systems Ultra-HDTV lässt auf Probleme schließen. Eine wichtige Überlegung bei der Einführung eines neuen Standards muss es sein, die Kompatibilität mit aktuellen Standards zu wahren. Analog zu den drei verschiedenen Systemparametern, die oben erwähnt werden, müssen die Signale auch unter drei verschiedenen Aspekten angepasst werden:

- *Frame Size*
- *Frame Rate*
- *Scan Mode*

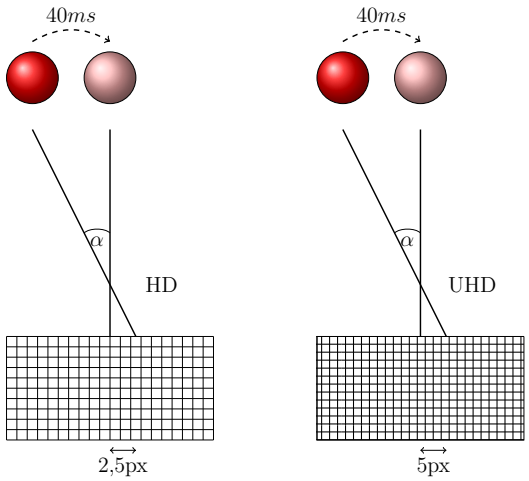


Abbildung 3.2
Bewegungsunschärfe im Vergleich - HD und Ultra-HD

Input	Output	Fokus	Input	Output	Fokus
25i	25p	x	25p		
	50p	x	50p	50p	x
	60p	x		60p	x
	100p	x		100p	x
	120p	x		120p	x
29,97i	25p		30p		
	50p			50p	
	60p			60p	
	100p			100p	
	120p			120p	
30i	25p		30p		
	50p			50p	
	60p			60p	
	100p			100p	
	120p			120p	

Tabelle 3.3
Konvertierungsanforderungen für UHD und Fokus dieser Arbeit

In dieser Arbeit werden die beiden letzten Aspekte untersucht. Tabelle 3.3 enthält eine Übersicht über mögliche Konvertierungsrichtungen im Zusammenhang mit UHD. In der jeweils letzten Spalte ist angegeben, auf welche Richtungen diese Arbeit besonders eingeht. Das in Kapitel 10 entwickelte Programm kann diese umsetzen. Es wird, unabhängig vom Eingangssignal, eine Konversion auf das jeweilige Format durchgeführt. Ein wesentlicher Aspekt während dieses Vorgangs ist das *Deinterlacing* der im Zeilensprung aufgezeichneten Eingangssignale. In einem weiteren Schritt wird zudem auf das *Upscaling* eingegangen.

Zurzeit wird eine Erweiterung des Ultra-HD Standards diskutiert. Um in europäischen Ländern die Einführung zu erleichtern, soll eine *Frame Rate* von $100Hz$ hinzugefügt werden [26]. Aus diesem Grund ist diese Frequenz ebenfalls im *Media Converter* (Kapitel 10) implementiert.

Die *Frame Rate* ist aus historischen Gründen (siehe Kapitel 2) an die Stromnetzfrequenz gekoppelt. Daher gibt es im europäischen Raum überwiegend $25Hz$-Material. Aus diesem Grund legt diese Arbeit den Fokus auf die Konvertierung von $25Hz$-basiertem Video in den Ultra-HD Standard. Zum großen Teil lässt sich $30Hz$ basiertes Material allerdings analog bearbeiten.

4 Frequenzbetrachtung

Das Kapitel 2 verdeutlicht, dass der Hauptgrund für die Einführung des *Interlacing* eine zu geringe Bandbreite des Übertragungssystems war. Um mit den vorhandenen Kapazitäten auszukommen wurde das Signal in vertikaler Richtung unterabgetastet. Statt 576 Zeilen pro Bild hatte jedes Bild lediglich 288 Zeilen. Die horizontale Pixelzahl blieb in jedem Bild bei 720 Pixeln. Bei der Einführung von HDTV wurde in einigen Versionen des Standards das *Interlacing* übernommen.
Durch die angesprochene Unterabtastung ergeben sich einige Konsequenzen für das Signal und die dargestellten Inhalte. Das Kapitel 4 beschreibt die Konsequenzen für den Frequenzbereich des Signals.

4.1 Alias

Im Gegensatz zu anderen Bereich der digitalen Signalverarbeitung, wie beispielsweise der A/D Wandlung von Audiosignalen, ist Alias im Bereich der Videotechnik nicht zu vermeiden und tritt andauernd auf. Das wohl bekannteste Beispiel ist der sogenannte Postkutschen-Effekt, bei dem sich die Räder einer Postkutsche bei einer bestimmten Geschwindigkeit rückwärts zu drehen scheinen, obwohl die Kutsche vorwärts fährt.
Der Effekt lässt sich durch das nyquist-shannonsche Abtasttheorem beschreiben. Demnach muss die Abtastfrequenz eines Signals mindestens doppelt so groß sein wie die maximale Frequenz innerhalb des Signals.

$$f_{abtast} >= 2 \cdot f_{max} \tag{4.1}$$

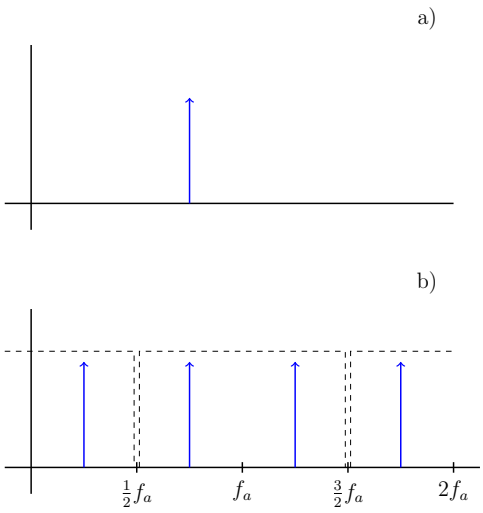


Abbildung 4.1
Entstehung von Alias -
a) Frequenz vor der Unterabtastung
b) Frequenz nach der Unterabtastung

Nur wenn das Abtasttheorem erfüllt ist, wird das Signal Alias-frei wiedergegeben. Ist dieses Theorem nicht erfüllt, erscheinen innerhalb des Signals ungewollte Artefakte. Abbildung 4.1 beschreibt, was durch die Spiegeleigenschaft des Spektrums in Kombination mit einer Unterabtastung passiert. Das Spektrum wiederholt sich an den Vielfachen der Abtastfrequenz. So entstehen auch im Nutzsignal Frequenzen, die ursprünglich im Signal nicht vorhanden

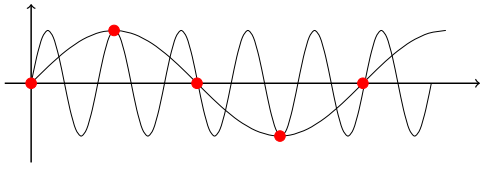


Abbildung 4.2
Sinussignal - Frequenzänderung durch Unterabtastung

waren. In Abbildung 4.2 wird deutlich, wie durch die Abtastung aus einem Sinus-Signal der Frequenz f_1 ein Signal der deutlich niedrigeren und offensichtlich falschen Frequenz f_2 wird.

Alias lässt sich im Nachhinein nicht wieder aus Signalen entfernen. Auch durch eine nachträgliche Überabtastung lassen sich die durch die Unterabtastung entstandenen Alias-Effekte nicht herausrechnen.

4.2 Räumlich-zeitliches Abtasten

Bei der Aufzeichnung eines Videosignals wird ein zeitlich und räumlich kontinuierliches Signal $f_C(\tilde{x}, \tilde{y}, \tilde{t})$ in ein diskretes Signal $f_D(x, y, t)$ gewandelt. Mathematisch lässt sich dieser Vorgang als die Multiplikation jeder der drei Komponenten von f_C mit einem Dirac Kamm III beschreiben:

$$f_D(x, y, t) = f_C(\tilde{x}, \tilde{y}, \tilde{t}) \cdot III_a(\tilde{x}) \cdot III_d(\tilde{y}) \cdot III_T(\tilde{t}) \tag{4.2}$$

Der Dirac-Kamm aus Formel 4.2 ist definiert als:

$$III_k(\tilde{m}) = \sum_{l=\infty}^{\infty} \delta(\tilde{m} - l \cdot k) \tag{4.3}$$

Der Dirac-Kamm entspricht demnach einer Folge von Dirac-Impulsen mit dem Abstand k, die diskrete Stellen aus dem kontinuierlichen Signal $\tilde{m}$ herauslösen. In Formel 4.2 geben somit a,d und T die Abstände der Dirac-Impulse an, die auf die kontinuierlichen Bildsignale $\tilde{x}$, $\tilde{y}$ und $\tilde{t}$ angewendet werden. Bezogen auf ein kontinuierliches Bildsignal entspricht a somit dem horizontalen Pixelabstand, d dem vertikalen Pixelabstand und T dem zeitlichen Abstand der Bilder.
Um den Frequenzbereich der Funktion f_D betrachten zu können werden die beiden Formeln 4.2 und 4.3 zunächst kombiniert:

$$\begin{aligned} f_D(x, y, t) &= f_C(\tilde{x}, \tilde{y}, \tilde{t}) \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \delta(\tilde{x} - k \cdot a)\delta(\tilde{y} - l \cdot d)\delta(\tilde{t} - m \cdot T) \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f_C(k \cdot a, l \cdot d, m \cdot n)\delta(\tilde{x} - k \cdot a)\delta(\tilde{y} - l \cdot d)\delta(\tilde{t} - m \cdot T) \end{aligned} \tag{4.4}$$

Formel 4.4 kann nun mithilfe der Fourier-Transformation in den Frequenzbereich überführt werden.

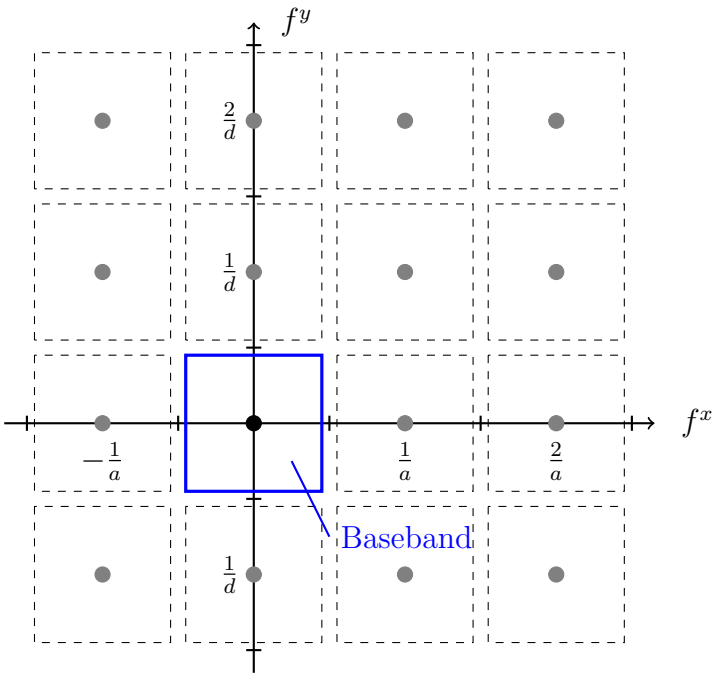


Abbildung 4.3
Frequenzen im *progressive* Signal

$$F_D(f^x, f^y, f^t) = \frac{1}{a \cdot d \cdot T} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} F_C(f^x - \frac{k}{a}, f^y - \frac{l}{d}, f^t - \frac{m}{T}) \tag{4.5}$$

Bezüglich des Spektrums kann aus der Formel 4.5 der Schluss gezogen werden, dass sich das Spektrum in x-Richtung an den Punkten $\frac{1}{a}$, in y-Richtung an den Punkten $\frac{1}{d}$ und in t-Richtung an den Punkten $\frac{1}{T}$ wiederholt.
Das in Abbildung 4.3 dargestellte Muster ist mit dem nyquist-shannonschen Abtasttheorem zu erklären (siehe 4.1). In Abbildung 4.3 ist das *Baseband* mit blau markiert. Man sieht deutlich, dass dieses nur bis an die Grenzen der benachbarten Bänder heranreicht, diese sich aber nicht überschneiden.

Beispiel Zur Verdeutlichung werden die oben beschriebenen Verhältnisse, auf ein progressives SDTV-Signal angewendet. Das abgetastete Bild hat eine horizontale Pixelzahl von $x = 720$ und eine vertikale Pixelzahl von $y = 576$. Die Variable a ist der horizontale Abstand der Pixel. Dieser kann somit definiert werden als $a = \frac{1}{x} \cdot pw$ (*Picture Width*). In diesem Fall ergibt sich $a = \frac{1}{720} \cdot pw$. Die Frequenzbänder haben in der f^x Richtung einen Abstand von $f_{x-abtast} = \frac{1}{a} = \frac{720}{pw}$. Entsprechend des Abtasttheorems (Formel 4.1) darf das Signal eine Maximale Frequenz von $f_{x-max} = \frac{1}{2} \cdot f_{x-abtast} = \frac{720}{2pw} = \frac{360}{pw}$ haben. Örtliche Frequenzen werden als *Cicles* pro *Picture Width* oder *Picture Height* angegeben ($[c/pw]$, bzw. $[c/ph]$). Somit ergibt sich für die maximale Frequenz in horizontaler Richtung $f_{x-max} = 360[c/pw]$.

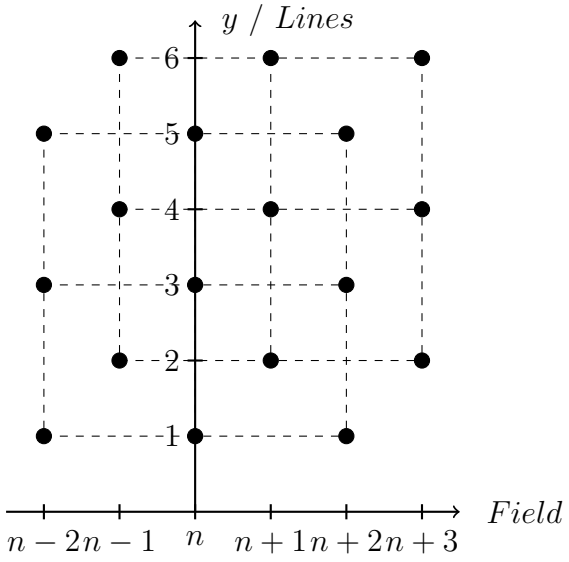


Abbildung 4.4
Beim *Interlacing* übertragene Zeilen (y/t-Ebene)

Eine Periode in einem Bild besteht aus einem schwarzen und einem weißen Pixel. Dementsprechend braucht man zwei Pixel für die Darstellung einer Periode. Hat man 720 Pixel zur Verfügung, so kann man maximal 360 Perioden darstellen. Das Ergebnis für f_{x-max} ist dementsprechend logisch.
Für f_{y-max} ergibt sich durch analoge Rechnung eine Frequenz von $f_{y-max} = 288[c/ph]$.

Das in diesem Abschnitt beschriebene Verfahren ist die ideale Abtastung eines idealen Bildes. In der Realität lassen sich weder Bandgrenzen so klar ziehen, wie es theoretisch gemacht wurde, noch kann garantiert werden, dass das Ausgangsbild lediglich aus Frequenzen unterhalb der Maximalfrequenzen f_{x-max} und f_{y-max} besteht. So entstehen beispielsweise durch Bewegung der Objekte im Bild oder der Kamera höhere Frequenzen. Die Auswirkungen dieser Unterabtastung wurden bereits in 4.1 behandelt.
Im Folgenden werden die Auswirkungen des *Interlacings* auf das Spektrum betrachtet.

4.3 Auswirkungen von Interlacing auf den Frequenzbereich

Beim *Interlacing* werden Halbbilder übertragen. Es wird in vertikaler Richtung unterabgetastet, sodass ein Halbbild nicht mehr aus der ursprünglichen Zeilenzahl y besteht, sondern lediglich aus $\frac{y}{2}$ Zeilen. In einer Bildfolge werden in einem Halbbild die geraden (*even Lines*) und im nächsten Halbbild die ungeraden (*odd Lines*) Zeilen des Ursprungsbildes übertragen. In x- und t-Richtung wird keine Unterabtastung vorgenommen. Insgesamt ergibt sich somit eine Reduzierung der Bandbreite um den Faktor zwei. Abbildung 4.4 zeigt die übertragenen Zeilen in der y/t-Ebene beim *Interlacing*.

Zudem sind die Bilder der gleichen Phase durch gestrichelte Linien miteinander verbunden. Man erkennt zwei Raster der selben Struktur, die in der y/t-Ebene verschoben sind. Dieses Phänomen wird bei der Frequenzbetrachtung ausgenutzt. Zu beachten ist, dass in dieser Abbildung das räumlich-zeitliche Verhältnis dargestellt ist. In Abbildung 4.3 waren es räumlich-räumliche Verhältnisse.
Das in Abbildung 4.4 dargestellte Raster $r_{interlaced}$ lässt sich wie in Formel 4.6 beschreiben. Es wird deutlich, dass sich in der y- und in der t-Ebene die Parameter durch das *Interlacing* geändert haben.

$$r_{interlaced}(x,y,t) = III_a(\tilde{x})III_{2d}(\tilde{y})III_{2T}(\tilde{t}) + III_a(\tilde{x})III_{2d}(\tilde{y}-d)III_{2T}(\tilde{t}-T) \tag{4.6}$$

Das diskrete Signal $f_D(x,y,t)$ ergibt sich durch die Multiplikation vom kontinuierlichen Eingangssignal $f_C(\tilde{x},\tilde{y},\tilde{t})$ mit dem Raster $r_{interlaced}$:

$$f_D(x,y,t) = f_C(\tilde{x},\tilde{y},\tilde{t}) \ \cdot \ r_{interlaced}(\tilde{x},\tilde{y},\tilde{t}) \tag{4.7}$$

$$\downarrow$$

$$F_D(f^x,f^y,f^t) = F_C(f^{\tilde{x}},f^{\tilde{y}},f^{\tilde{t}}) \ \overset{3}{*} \ R_{interlaced}(f^{\tilde{x}},f^{\tilde{y}},f^{\tilde{t}}) \tag{4.8}$$

Aus der Multiplikation wird im Frequenzbereich eine Faltung mit dem Spektrum $R_{interlaced}$ des Gitters $r_{interlaced}$. Um beurteilen zu können, welche Auswirkungen die Operation auf den Frequenzbereich des Signals hat genügt es, $R_{interlaced}$ genauer zu betrachen. Dieses ergibt sich aus 4.6 zu:

$$\begin{aligned} R_{interlaced} &= \frac{1}{4adt} \cdot III_{\frac{1}{a}}(f^x) \cdot III_{\frac{1}{2d}}(f^y) \cdot III_{\frac{1}{2T}}(f^t) \\ &+ \frac{1}{4adt} \cdot III_{\frac{1}{a}}(f^x) \cdot III_{\frac{1}{2d}}(f^y)e^{-j2\pi f^y d} \cdot III_{\frac{1}{2T}}(f^t)e^{-j2\pi f^t T} \\ &= \frac{1}{4adt} \cdot III_{\frac{1}{a}}(f^x) \cdot III_{\frac{1}{2d}}(f^y) \cdot III_{\frac{1}{2T}}(f^t) \cdot (1 + e^{-j2\pi f^y d}e^{-j2\pi f^t T}) \end{aligned} \tag{4.9}$$

Das Ergebnis besteht aus zwei Teilen. Der erste Teil $(\frac{1}{4adt} \cdot III_{\frac{1}{a}}(f^x) \cdot III_{\frac{1}{2d}}(f^y) \cdot III_{\frac{1}{2T}}(f^t))$ beschreibt die Frequenzanteile im Spektrum, der zweite Teil $(1 + e^{-j2\pi f^y d}e^{-j2\pi f^t T})$ gibt die Positionen an, an denen diese auftreten.

Es ergibt sich exemplarisch für $e^{-j2\pi f^y d}$:

$$\begin{aligned} für f^y = 0: &\qquad e^{-j2\pi f^y d} = 1 \\ für f^y = \frac{1}{2d}: &\qquad e^{-j2\pi f^y d} = -1 \\ für f^y = \frac{1}{d}: &\qquad e^{-j2\pi f^y d} = 1 \end{aligned} \tag{4.10}$$

Die Betrachtung kann analog für den Term $e^{-j2\pi f^t T}$ durchgeführt werden. Das Produkt aus $e^{-j2\pi f^t d}$ und $e^{-j2\pi f^t T}$ schwankt demnach zwischen -1 und 1. Der Term $1+e^{-j2\pi f^y d}e^{-j2\pi f^t T}$ nimmt Werte zwischen 0 und 2 an.
Im Endeffekt ergibt sich ein Raster, bei dem die Hälfte der Frequenzen ausgelöscht und die andere Hälfte verdoppelt wird. Abbildung 4.5 zeigt das Spektrum eines *Interlaced*-Signal in der y/t-Ebene.

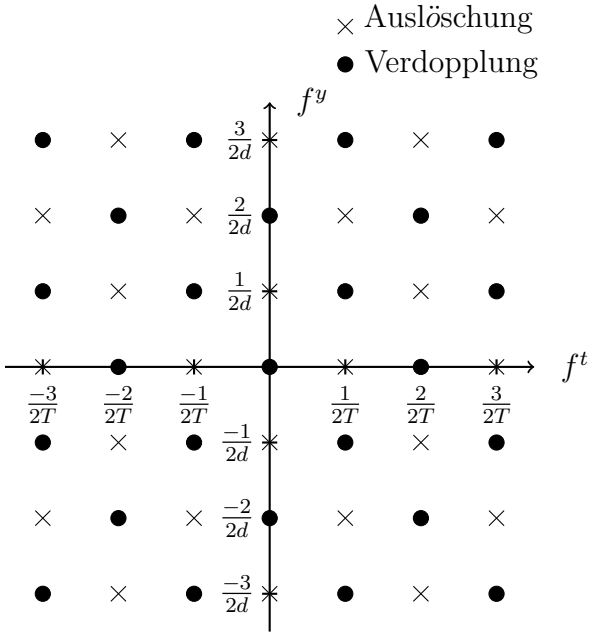


Abbildung 4.5
Spektrum eines *Interlaced*-Signals

Es lässt sich ableiten, wie die Beschaffenheit des Basisbandes in diesem Spektrum ist. Abbildung 4.6 verdeutlicht die Bandgrenzen und somit die Problematik des *Interlacing* Vorgangs. Durch die Rautenform werden stationäre Bilder fehlerfrei wiedergegeben; sobald allerdings Bewegung im Bild vorhanden ist, reduziert sich die vertikale Auflösung drastisch [2].

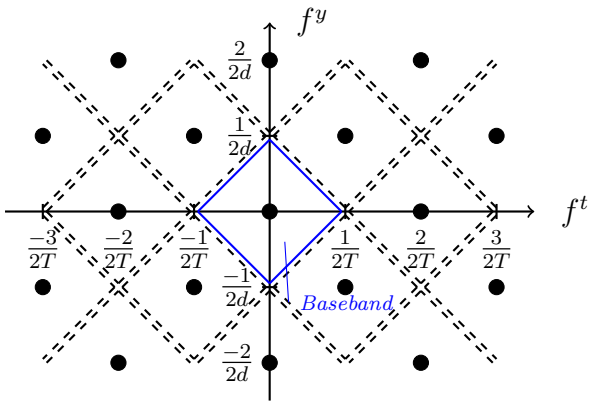


Abbildung 4.6
Basisband des *Interlaced*-Signals

Wie oben bereits beschrieben, werden die Bandgrenzen bei einem Videosignal regelmäßig überschritten. Bewegungen von Kamera und Objekten in der Szene sind schwer zu beschränken, die Oberflächenbeschaffenheit und somit die Horizontal- und Vertikalauflösung der Objekte sind nicht in jeder Szene kontrollierbar. Alias ist also ein allgegenwärtiger Effekt bei Videosignalen. Beim *Deinterlacing* ist die Strategie, möglichst gut mit diesem Effekt umzugehen und ihn durch die weitere Bearbeitung nicht noch zu verstärken.

5 Motion Estimation

Ein zentraler Bereich der Videobearbeitung ist die *Motion Estimation*. Bewegungsvektoren werden insbesondere für die drei wichtigen Aufgaben der Kodierung, Formatkonvertierung und des *Deinterlacing* benötigt. Zur Bestimmung der Bewegung innerhalb eines Bildes werden viele unterschiedliche Ansätze verfolgt, alle lassen sich jedoch in drei Kategorien unterteilen:

- pixelbasierend
- blockbasierend
- objektbasierend

Die pixelbasierenden Algorithmen wurden historisch gesehen als erste entwickelt. Wie der Name bereits vermuten lässt, analysieren diese Algorithmen jedes einzelne Pixel. Durch die Erkenntnis, dass diese Suchstrategie sehr ineffizient ist, ging man zu den blockbasierenden Algorithmen über. Diese werden heutzutage am meisten eingesetzt. Das zu analysierende Bild wird in Blöcke unterteilt. Der Algorithmus prüft, an welcher Stelle dieser Block im darauffolgenden Bild wiederzufinden ist. Ein weiterer Ansatz ist die objektbasierende Herangehensweise. Anstatt das Bild in Blöcke eines festen Rasters zu unterteilen wird eine Objekterkennung durchgeführt und es wird die Bewegung dieser festen Objekte analysiert. Obwohl für viele Aufgaben in der Videoverarbeitung die blockbasierenden Verfahren genutzt werden, liefern die objektbasierenden laut [2] ebenso vielversprechende Ergebnisse.

In diesem Kapitel wird nur die blockbasierende Phasenkorrelation vorgestellt. Da für die anstehende Aufgabe eine möglichst gute Bildqualität benötigt wird (siehe 8.2), beschränkt sich Kapitel 5 auf dieses Verfahren, das bemerkenswert genaue Ergebnisse liefert [29]. H. Foroosh [9] benennt einige sehr nützliche Eigenschaften der Phasenkorrelation. So ist diese Methode beispielsweise sehr robust gegenüber Luminanzschwankungen und Rauschen, auch können große und verschiedene Bewegungen im Bild erkannt werden. Des Weiteren lässt sich der Algorithmus auf Subpixelgenauigkeit erweitern. Im Folgenden werden alle Bestandteile des Algorithmus beschrieben. Eine Übersicht über weitere Algorithmen und die historischen Zusammenhänge sowie eine qualitative Beurteilung findet sich in [2].

5.1 Bewegungsarten

In einer Videosequenz können sich Bildelemente von einem zum anderen *Frame* auf verschiedene Art und Weise verändern. Stone et a. [23] fasst folgende Transformationsfamilien zusammen:

- starre Transformationen
  (Translation, Rotation, Zoom/Rescaling)

- lineare und affine Transformationen
  (Scherung und perspektivische Transformationen)
- nichtlineare Verzerrungen

Im Idealfall würde eine *Motion Estimation* für jede Transformationsart für jedes Pixel die richtige Bewegung erkennen. In vielen Fällen ist eine Bewegungsschätzung einfacher, bei der mehrere Pixel die selbe Bewegung aufweisen.

5.2 Match Kriterium

Immer wieder muss während der Bewegungsschätzung geprüft werden, wie gut ein verschobenes und somit angenährtes Pixel (oder auch ein kleiner Block) aus einem Bild mit dem Pixel (Block) des zeitlich nachfolgenden Bildes übereinstimmt. Zu diesem Zweck werden in [2] verschiedene Kriterien vorgestellt. Die Vorschriften sind an dieser Stelle für das Prüfen eines Blocks $B(\vec{X})$ mit dem Zentrum $\vec{X} = (X, Y)$ formuliert. $\vec{V}$ ist die Summe aller errechneten Bewegungsvektoren.

5.2.1 SAD (Summed Absolute Difference)

Die *Summed Absolute Difference* ist die meist benutzte Vorschrift.

$$error_{SAD} = \min_{\vec{v} \in \vec{V}} \left(\sum_{\vec{x} \in B(\vec{x})} |F(\vec{x}, n) - F(\vec{x} - \vec{v}, n+1)| \right) \tag{5.1}$$

5.2.2 MSE (Mean Square Error)

Eine verbreitete Alternative ist der *Mean Square Error*. Jedoch ist diese Methode auch deutlich rechenaufwändiger.

$$error_{MSE} = \min_{\vec{v} \in \vec{V}} \left(\sum_{\vec{x} \in B(\vec{x})} (F(\vec{x}, n) - F(\vec{x} - \vec{v}, n+1))^2 \right) \tag{5.2}$$

5.3 Probleme und Grenzen

Bei dem Versuch, Bewegungen innerhalb eines Bildes zu errechnen, ergeben sich kritische Momente. Diese Situationen, in denen Informationen nicht eindeutig bestimmt werden können bzw. fehlen, können grob in drei Kategorien unterteilt werden. Algorithmen, die dazu dienen Bewegungen zu errechnen und mit Hilfe dieser Berechnung Zusatzinformationen innerhalb einer Sequenz zu erzeugen, müssen mit diesen problematischen Situationen umgehen können und so fehlertolerant sein, dass sie dennoch die richtige Bewegung annehmen. Im Folgenden sind die drei möglichen Problemquellen kategorisiert [5].

Aperture

Das sogenannte *Aperture* Problem tritt an geraden Kanten oder auch bei monochromen Flächen auf. Verschiebt man die Blöcke/Pixel um verschiedene Bewegungsvektoren, entstehen dennoch gleiche *Error*-Werte. Bei Kanten tritt das Problem bei den Vektoren auf, die entlang der Kante zeigen. Bei monochromen Flächen tritt das Phänomen für alle Vektoren auf. Es ist somit nicht zu unterscheiden, welcher Vektor der tatsächlichen Bewegung entspricht. Abhilfe schafft in diesem Fall die Verwendung von großen Suchradien im Vergleich zur Objektgröße. Auf diese Art lassen sich oft anders geartete Strukturen in die Berechnung mit einbeziehen.

Noise

Je mehr Rauschen im Bild vorhanden ist, desto schwieriger wird es der wahren Bewegung entsprechende Vektoren zu bestimmen. Insbesondere bei kleinen Suchbereichen kann es vorkommen, dass falsche Bewegungsvektoren bei der Berechnung des *Match* Kriteriums zu besseren Ergebnissen führen als die tatsächlichen Vektoren. In diesem Fall ist das lokale Minimum ungleich dem globalen Minimum. Je weniger Struktur im Bild vorhanden ist, desto problematischer wird das Rauschen.

Occlusion

Objekten, Blöcken oder Pixeln können nur solange sinnvolle Bewegungsvektoren zugewiesen werden, wie in beiden Bildern, zwischen denen die Transformation stattfindet, auch die selben Objekte, Blöcke und Pixel vorhanden sind. Wird durch eine Bewegung eines Objektes beispielsweise der Hintergrund verdeckt, der zuvor sichtbar war, oder sind Objekte durch einen Kameraschwenk nicht mehr im Bild zu sehen, sind für diese Pixel keine sinnvollen Bewegungsvektoren zu ermitteln. In diesen Fällen spricht man von *obscured Background*. Auch das Gegenteil kann passieren, nämlich dass Elemente im Bild erscheinen (*uncovered Background*), die zuvor noch nicht zu sehen waren. Auch für diese Teile gibt es keine sinnvollen Vektoren.

5.4 Motion Estimation Techniken

5.4.1 Phase Correlation

Das Prinzip der Phasenkorrelation wurde zuerst im Jahr 1975 von *C.D. Kuglin* und *D.C. Hines* vorgestellt [13]. Es bedient sich einer der wesentlichen Eigenschaften der Fourier-Transformation:

Eine räumliche Verschiebung eines Bildes resultiert in einer linearen Phasendiffernz im Freqzenbereich [29].

G_1 und G_2 sind die 2D-Transformierten der Bilder g_1 und g_2. G_1 und G_2 können dargestellt werden als:

$$G_1(f^x, f^y) = |G_1(f^x, f^y)|e^{j\varphi_1(f^x, f^y)} \tag{5.3}$$

$$G_2(f^x, f^y) = |G_2(f^x, f^y)|e^{j\varphi_2(f^x, f^y)} \tag{5.4}$$

Zwischen diesen beiden Funktionen kann eine Phasendifferenz gebildet werden:

$$e^{i\Phi} = e^{i\varphi_1} - e^{i\varphi_2} \tag{5.5}$$

Diese Phasendifferenz ist die Darstellung der räumlichen Verschiebung der Bilder im Frequenzbereich. Wird diese Phasenverschiebung durch die inverse Fouriertransformation zurück in den räumlichen Bereich transformiert, so erhält man die tatsächliche Verschiebung:

$$d = F^{-1}(e^{j\varphi}) \tag{5.6}$$

Um die Phasendifferenz zu errechnen, benutzt man das normierte Kreuzleistungsdichtespektrum $\frac{G_1 G_2^*}{|G_1 {G_2}^*|}$:

$$\frac{G_1 G_2^*}{|G_1 {G_2}^*|} = \frac{|G_1||G_2|e^{j\varphi_1}e^{-j\varphi_2}}{|G_1 {G_2}^*|} = e^{j\varphi_1}e^{-j\varphi_2} = e^{j(\varphi_1 - \varphi_2)} \tag{5.7}$$

Im konkreten Bereich der Videosignalanalyse ist folgender Zusammenhang zwischen g_1 und g_2 anzunehmen:

$$g_2(x, y) = g_2(\vec{r}) = g_1(x + a, y + b) = g_1(\vec{r} + \vec{l}) \tag{5.8}$$

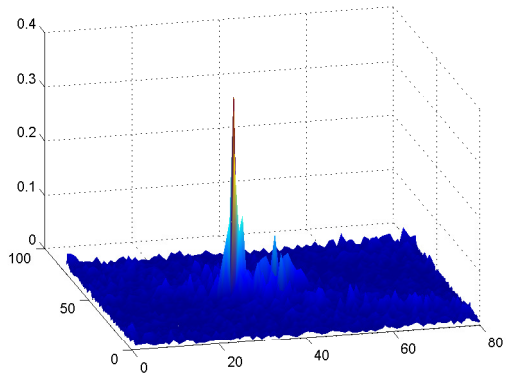


Abbildung 5.1
Peaks in der Phasenkorrelationsebene

g_2 ist eine um $\vec{l} = (a, b)$ verschobene Kopie von g_1. Für die Fouriertransformierte von G_2 ergibt sich:

$$G_2(\vec{f}) = G_1(\vec{f})e^{-2\pi\vec{f}\vec{l}} \tag{5.9}$$

Die Phasendifferenz zwischen G_1 und G_2 beträgt:

$$\varphi = -2\pi\vec{f}\vec{l} \tag{5.10}$$

Wenn man diese Phasendifferenz zurück in den Ortsbereich transformiert, so ergibt sich ein Delta-Funktional an der um $\vec{l}$ verschobenen Stelle $\delta(\vec{r} - \vec{l})$. In Abbildung 5.1 ist eine solches Deltafunktional grafisch dargestellt. Über einen *Peakfinding*-Algorithmus lassen sich die Bewegungsvektoren aus der Ebene extrahieren.

5.4.2 Motion Estimation mit Phase Correlation

Der unter 5.4.1 beschriebene Algorithmus der Phasenkorrelation bestimmt die räumliche Verschiebung der Pixel zweier $n \times n$ Pixelblöcke der selben Szene. Durch das eingesetzte Verfahren erhält man eine Ebene, in der durch verschobene Dirac-Impulse mögliche Bewegungsvektoren dargestellt werden.
Da die Phasenkorrelation jedoch lediglich die im $n \times n$ Block vorhandenen Bewegungsvektoren identifizieren, nicht aber diese den einzelnen Subblöcken oder Pixeln zuordnen kann, bedarf es eines weiteren Schritts innerhalb des Algorithmus. Thomas et al. [27] entwickelt ein Verfahren, das in seinen Grundzügen auch heutzutage noch verwendet wird [5]. Im Folgenden wird zunächst das Funktionsprinzip erläutert. In 5.6.1 werden inzwischen entwickelte Erweiterungen des Verfahrens zur Verbesserung beschrieben.

Algorithmus

Eine Stärke der Phasenkorrelation ist die Genauigkeit, mit der Bewegungen innerhalb eines Bildes bestimmt werden können. Zudem liefert der Algorithmus auch bei starken Helligkeitsschwankungen oder Rauschen im Bild gute, brauchbare Ergebnisse. Auch können vergleichsweise große Verschiebungen innerhalb des Blocks problemlos identifiziert werden.
Problematisch im Hinblick auf die weitere Verarbeitung (siehe Kapitel 6) ist die Tatsache, dass die Phasenkorrelation lediglich *global Motion* messen kann, also Bewegungsvektoren, die ganzen Blöcken und nicht einzelnen Pixeln zugeordnet werden können. Für viele Aufgaben werden allerdings diese pixelgenauen Bewegungsvektoren benötigt. Um diese zu erlangen, schlägt Thomas et al. [27] einen zweistufigen Algorithmus vor.

1. Im ersten Schritt wird das Eingangsbild in große Blöcke (zum Beispiel 64×64 Pixel) unterteilt. Auf diese Blöcke und deren Pendant im nachfolgenden Bild wird jeweils eine Phasenkorrelation ausgeführt. Es können nun in jeder Phasenkorrelationsebene die Bewegungspeaks extrahiert werden. Am Ende dieses Schrittes ergibt sich also eine Liste möglicher Bewegungsvektoren, die den jeweiligen Blöcken zugeordnet sind.

2. Die im ersten Schritt erzeugten Vektoren, werden im zweiten Schritt weiterverarbeitet. Innerhalb eines Blocks werden kleinere Blöcke (zum Beispiel 2×2 oder 8×8) oder auch nur Pixel um jeden der für diesen Block errechneten Bewegungsvektoren verschoben. Für die Abweichung zwischen dem verschobenen Block/Pixel und dessen zeitlichen Nachfolger wird pro Verschiebungsvektor ein *Match* Kriterium (siehe 5.2) errechnet, das den gemachten Fehler wiedergibt. Im Endeffekt wird der Bewegungsvektor dem Block/Pixel zugewiesen, dessen Fehler am kleinsten ist.
   In dem Fall, dass kein Bewegungsvektor sinnvolle Ergebnisse liefert, weist das Bild vermutlich unberechenbare Bewegungen oder einen bisher vom Objekt verdeckten Hintergrund auf. Dieser Fall bedarf einer besonderen Behandlung, die in 5.6 näher erläutert wird.

Das Verfahren liefert gute Ergebnisse für Translationen innerhalb eines Bildes. Es ist wichtig, dass die Blockgröße bei der Phasenkorrelation doppelt so groß ist wie die größte im Bild zu erwartende Bewegung, da die *Peaks* in der Korrelationsebene sonst nicht eindeutig einer Verschiebungsrichtung zugeordnet werden können [27].
Da die Peaks in der Korrelationsebene am stärksten sind wenn eine Bewegung innerhalb eines Blocks häufig vorhanden ist, sind Zoom und Rotationsbewegungen nur schwer zu erfassen. Um dennoch auch für diese Veränderungen im Bild brauchbare Ergebnisse zu bekommen, werden in jedem Block für Schritt *zwei* des Verfahrens zu den eigenen k größten Verschiebungen jeweils die k größten Verschiebungen der angrenzenden Blöcke

mit aufgenommen. Insgesamt ergeben sich also pro Block $9 \times k$ Verschiebungsvektoren. In [27] wird vorgeschlagen, pro Block drei Vektoren zu extrahieren. Somit ergeben sich 27 mögliche Kandidaten pro Block.

5.5 True Motion

Ein Verfahren zur Bewegungsschätzung liefert zunächst lediglich die Wahrscheinlichkeit für eine bestimmte Bewegung [2]. Würde man mit diesen Ergebnissen bereits ein *Deinterlacing* oder eine Codierung ausführen, so ergäben diese eine schlechte Bildqualität. Unter dem Begriff *True Motion* versteht man ein Vektorfeld, in dem die geschätzten Bewegungsvektoren der *Motion Estimation* einem weiteren Bearbeitungsschritt unterzogen wurden. Insbesondere für Bilder, die große zusammengehörende Flächen beinhalten, ist ein konsistentes Vektorfeld wichtig.
Zudem wird in vielen Bewegungsschätzungsverfahren blockweise je ein Vektor berechnet. Um das Bewegungsfeld für Aufgaben wie *Deinterlacing* oder *Upscaling* nutzen zu können ist allerdings ein Vektor pro Pixel notwendig, da sonst Blockartefakte im Bild entstehen. Für diese Nachbereitung gibt es zwei verschiedene Herangehensweisen. Zum einen die sogenannten *expliziten* Varianten, die ein bereits existierendes Vektorfeld glätten und fehlende Vektoren auf Pixelebene interpolieren, zum anderen die *impliziten* Varianten, die in die Algorithmen zur Bewegungsschätzung eingebaut werden und dafür sorgen, dass bereits bei der Berechnung des Vektorfeldes eine Homogenisierung stattfindet. Die Vektoren auf Pixelebene werden bei diesem Verfahren mit Hilfe der Bildinformationen und aus einem Pool von möglichen Vektoren der umliegenden Blöcken bestimmt. Bei Vektorfeldern, die dieser Bearbeitung unterzogen wurden, spricht man von *True Motion*. Um Bewegungsvektoren zu erlangen, die möglichst der tatsächlichen Bewegung entsprechen, sind die impliziten Verfahren zu bevorzugen. In einigen Fällen, insbesondere an Kanten, ist keine Homogenität der Bewegungsvektoren gewünscht. Die expliziten Verfahren glätten jedoch das gesamte Vektorfeld ohne auf solche Sonderfälle Rücksicht zu nehmen[5][2].

5.5.1 Blockmatching

Da die Phasenkorrelation die Bewegungsvektoren lediglich für einen Block von $m \times m$ Pixeln berechnen kann und diese keinem der Pixel zugeordnet sind, ist in einem zweiten Schritt ein *Blockmatching* notwendig. Da in diesem Fall jedes Pixel lediglich bezüglich einer begrenzten Anzahl $\vec{D}$ an Vektoren getestet werden muss, ist ein *Brute Force* Algorithmus angebracht. Für jeden Kandidatenvektor wird ein Fehler errechnet (siehe 5.2), wobei der Vektor, der den geringsten Fehler erzeugt, am wahrscheinlichsten die wahre Bewegung beschreibt. Im Kapitel 10 dieser Arbeit werden außerdem zwei Methoden entwickelt, die die Umgebung des jeweiligen Pixels mit in die Entscheidung einbeziehen.

5.6 Obscured and Uncovered Background

Sich bewegende Objekte innerhalb eines Bildes erzeugen Unregelmäßigkeiten im Vektorfeld. Werden alle Bewegungsvektoren angewandt und die jeweiligen Pixel dementsprechend verschoben, so passiert es zwangsläufig, dass einige Bereiche des Bildes verschoben und durch keine neuen Pixel ersetzt werden. Es entstehen schwarze Löcher innerhalb des Bildes (vgl. Abbildung 5.2). Gleichermaßen gibt es aber auch Pixel im neu berechneten Bild, auf die zwei Bewegungsvektoren von unterschiedlichen Pixeln aus dem Ursprungsbild zeigen. Die Pixel werden doppelt belegt. Letzteres ist kein Problem, da das Pixel auch bei unterschiedlichen Vektoren eine ähnliche Farbe bekommt. Die Vektoren wurden anhand des *Match* Kriteriums ausgesucht. Der erste Fall, dass Pixel nicht neu belegt werden, ist inakzeptabel.
In diesem Fall wird das neue Bild folgendermaßen berechnet:

Abbildung 5.2
Uncovered Background

$$F_{MCshift}(\vec{x} + \alpha\vec{D}(\vec{x}, n), n + \alpha) = F(\vec{x}, n), (0 \leq \alpha \leq 1) \tag{5.11}$$

Vijay et al. [5] schlägt vor, anstatt vom Bild n auszugehen und die Vektoren als Adresse im Bild $n + \alpha$ zu betrachten an die geschrieben werden soll, vom Bild $n + 1$ ausgehen und für jeden Pixel die Leseadresse ändern:

$$F_{MCfetch}(\vec{x}, n + \alpha) = F(\vec{x} - \alpha\vec{D}(\vec{x}, n + \alpha), n), (0 \leq \alpha \leq 1) \tag{5.12}$$

Auf diese Art und Weise gibt es keine schwarzen Löcher, da die Pixel gewissermaßen vorinitialisiert werden. Für den Fall, dass keine Vektoren für bestimmte Pixel vorhanden sind (in dem Fall entstanden bisher schwarze Löcher), nehmen die Pixel die Werte aus dem Bild $n + 1$ an. Da die *uncovered* Pixel durch die Bewegung eines Objektes freigelegt werden, ist die Wahrscheinlichkeit sehr hoch, dass sie zum Zeitpunkt $n + \alpha$ den selben Wert wie zum Zeitpunkt $n + 1$ haben [5].

5.6.1 Erweiterungen des Phase Correlation Algorithmus

Der oben beschriebene Algorithmus zur *Motion Estimation* mit der Phasenkorrelation stammt aus dem Jahr 1987. Mittlerweile sind viele Erweiterungen entwickelt worden, die das Verfahren noch genauer, robuster oder schneller machen sollen. Das Funktionsprinzip ist dabei allerdings erhalten geblieben und wird bis heute eingesetzt. Einige interessante Verbesserungsmöglichkeiten werden in diesem Abschnitt behandelt.

Kontrasterhöhungen

Die verschiedenen *Peaks*, die durch das Verfahren der Phasenkorrelation entstehen, sind in der Korrelationsebene unterschiedlich groß ausgeprägt. Die tatsächliche Höhe hängt mit dem Verhältnis von sich überschneidenden und sich nicht überschneidenden Bildteilen zwischen den beiden aufeinanderfolgenden Bildern zusammen. Je größer die Überschneidung der Bildanteile ist, desto ausgeprägter ist der *Peak* in den Korrelationsebene [9]. Dieses bildinhaltsbedingte Rauschen lässt sich nicht herausfiltern. Jedoch entsteht bei dem oben beschriebenen Algorithmus auch ein verfahrensbedingtes Rauschen. Um dieses Rauschen zu mindern werden in diesem Abschnitt zwei Verfahren geschildert.
Der erste hier erwähnte Vorschlag zur Verbesserung stammt aus [27]. Die Blöcke, zwischen denen die Verschiebung gemessen wird, werden vorher mit einer *Windowing Function* belegt. Mit dieser Bearbeitung werden die Blöcke zum Rand hin schwarz. Es wird eine *Raised Cosine Function* für diese Bearbeitung vorgeschlagen, die den Vorteil einer steilen Flanke bietet und die Pixel in der Blockmitte unbehandelt lässt.
Mit dieser Anwendung wird ein deutlich höherer Kontrast zwischen den Bewegungspeaks erzielt und das Rauschen in der Korrelationsebene gemindert.
Bei Bewegungen der Kamera sind insbesondere an den Kanten der Blöcke die Pixel zwischen zwei zeitlich aufeinander folgenden Blöcken verschieden. Dieser *obscured* und *uncovered Background* fließt durch die Bearbeitung in geringerem Maße als Rauschen ein.
Der zweite wichtige Aspekt ist entscheidender. Durch die periodischen Eigenschaften der Fouriertransformation wird der Block bis ins Unendliche wiederholt aneinander gehängt. An den Blockgrenzen können starke Helligkeitsunterschiede auftreten. Diese erscheinen positionsmäßig in beiden verglichenen Blöcken an der selben Stelle und ergeben somit in der Korrelationsebene einen *Peak* an der Stelle $x = (0, 0)$. Dieser ist vergleichsweise groß und kann unter Umständen kleine, tatsächlich im Bild vorhandene Bewegungen verdecken.
Diese Maßnahme zur Kontrasterhöhung fördert also die deutlichere Darstellung der Ergebnisse der Phasenkorrelation. Problematisch ist der größere Rechenaufwand. Blöcke müssen sich überlappen, da die vom *Fade-Out* betroffenen Pixel nicht gewertet werden können und die *Windowing Function* angewendet werden muss.

Stone et al. [22] benutzt für das Windowing ein *Blackman-Window* und erzielt mit dieser Bearbeitung bereits wesentlich bessere Ergebnisse. Ahmed et al. [1] kritisiert an dieser Methode, dass sie relevantes Bildmaterial bearbeitet und beeinträchtigt. Dadurch seien lediglich Bewegungen in der Bildmitte erfassbar. Große Verschiebungen sind somit nicht zu bestimmen. Um dieses Problem zu lösen, wird eine Erweiterung des *Windowing* vorgeschlagen. Das Originalbild wird in alle Richtungen um δ Pixel erweitert. Diese bekommen jeweils den Wert des Originalbildes zugewiesen, der ihrer Position am nähesten liegt. In

einem zweiten Schritt wird lediglich auf diese zusätzlichen Pixel eine Gaußfunktion angewendet, wobei je nach Position nur die steigende, bzw. fallende Flanke genommen wird, sodass das Bild an den Kanten von Schwarz bis zur eigentlichen Helligkeit sanft steigt. Für Ecken wird eine zweidimensionale Gauß-Funktion benutzt.
[10] wiederum nutzt ein *Hamming-Window* für die Vorfilterung.

Sub-Pixel

Bilder bestehen naturgemäß aus diskreten Pixelwerten an diskreten Punkten. Dementsprechend wird bei der Transformation und der Rücktransformation in den Frequenzbereich die DFT (*Discrete Fourier Transformation*) und die IDFT (*Inverse Discrete Fourier Transformation*) oder die FFT (*Fast Fourier Transformation*) und die IFFT (*Inverse Fast Fourier Transformation*) angewendet. Es entsteht ein diskretes, periodisches Frequenzspektrum und schließlich ein diskretes Raster mit einem *Unit-Impulse* (Stoßfunktion). Wären alle Signale kontinuierlich, gäbe es anstatt eines *Unit-Impuls* einen Dirac-Stoß.
Durch die Tatsache, dass es sich bei Bildern um diskrete Signale handelt, kann die Verschiebung durch den *Unit-Impulse* nur pixelgenau ermittelt werden. Möchte man Sub-Pixel Genauigkeit erlangen, so muss der oben beschriebene Algorithmus erweitert werden. Foroosh et al. [9] beschreibt einen möglichen Weg zur Erweiterung auf Subpixelgenauigkeit. Er verfolgt den Ansatz, dass jede Subpixelbewegung einmal eine Integerbewegung war, die durch *Downsampling* der Originalbilder zu einer Subpixelbewegung geworden ist. Um den Rahmen dieser Arbeit nicht zu überschreiten, wird an dieser Stelle auf das Originaldokument verwiesen.

Frequenzbasierende Methoden

Yan et al. [29] beschreibt einen frequenzbasierenden Ansatz, bei dem die Rücktransformation aus dem Frequenzbereich nicht mehr notwendig ist und die Bewegung direkt aus der Frequenzdarstellung ermittelt wird. Foroosh et al. [9] erklärt jedoch, dass die frequenzbasierenden Ansätze allesamt das Problem der mit Rauschen versehenden Daten haben. Aus diesem Grund wird an dieser Stelle nur erwähnt, dass es Ansätze in diese Richtung gibt und nicht näher darauf eingegangen.

5.7 Motion Estimaton bei interlaced Signalen

Durch die Gegebenheiten bei *interlaced* Signalen müssen bei der Bewegungsschätzung zusätzliche Faktoren bedacht werden. Die verschiedenen Halbbilder unterscheiden sich zum einen in der Bewegungsphase, zum anderen durch einen vertikalen Versatz. Eine direkte Bewegunsschätzung zwischen einem geraden und einem ungeraden Halbbild liefert keine zufriedenstellenden Ergebnisse.

Renxiang et al. [16] beschreibt ein Verfahren, das vor der Bewegungsschätzung einen Korrekturfilter auf eines der beiden Halbbilder anwendet, wodurch der Phasenfehler korrigiert werden soll. Eine andere Variante um dieses Problem zu lösen ist die Bewegungsschätzung jeweils zwischen gleichen Typen von Halbbildern durchzuführen (n und $n+2$, $n+1$ und $n+3$). Bei dieser Variante wird von einer konstanten Geschwindigkeit eines Objektes von einem zum nächsten Halbbild gleichen Typs ausgegangen. Für die weitere Bearbeitung müssen die Bewegungsvektoren halbiert werden.

6 Deinterlacingverfahren

Ziel des sogenannten *Deinterlacing* ist es, die nicht im Bild vorhandenen Zeilen zu rekonstruieren. Dabei kann das *Deinterlacing* die originalen Zeilen eines Bildes nicht fehlerfrei wiederherstellen. Die Aufgabe eines jeden Algorithmus ist ein möglichst kleinen Fehler zwischen der rekonstruierten Sequenz und dem Original vor dem *Deinterlacing* zu erzeugen. Um die Qualität eines Algorithmus zu testen, lässt sich für Sequenzen, die erst nachträglich vom *progressive* ins *interlaced* Format überführt wurden und somit ebenfalls noch im *progressive* Format vorliegen, der gemachte Fehler ausrechnen.

$$minerror = seq_{original} - seq_{deinterlaced} \tag{6.1}$$

Je nach Anwendung spielt allerdings auch der Rechenaufwand für einen Algorithmus eine große Rolle. So muss in einigen Anwendungen eine Ausführung in *Real-Time* möglich sein oder ein *Deinterlacing* mit möglichst wenig Ressourcen durchgeführt werden. Demzufolge gibt es verschiedene Algorithmen, die jeweils ihrem Ziel entsprechend den optimalen *Trade-Off* zwischen Qualität und Rechenintensität suchen.

Obwohl die Herangehensweisen deutlich unterschieden werden müssen, lässt sich die Berechnung des Ausgangsbildes $F_{out}(\vec{x}, n)$ bei allen Verfahren folgendermaßen beschreiben:

$$F_{out}(\vec{x}, n) = \begin{cases} F_o(\vec{x}, n), & (y \bmod 2 = n \bmod 2) \\ F_i(\vec{x}, n), & (\text{otherwise}) \end{cases} \tag{6.2}$$

In Formel 6.2 beschreibt $F_{out}(\vec{x}, n)$ das Ergebnisbild nach dem Deinterlacing, $F_o(\vec{x}, n)$ das Originalbild, in dem entweder die geraden (*even Lines*) oder die ungeraden (*odd Lines*) Zeilen vorhanden sind, und $F_i(\vec{x}, n)$ das zu interpolierende Bild. $\vec{x} = \begin{pmatrix} x & y \end{pmatrix}^{-1}$ ist die Spalte/Zeile Kombination der Pixel des Bildes und n die Bildnummer.
Der Ausdruck '$y \bmod 2 = n \bmod 2$' wird für gerade Zeilen in geraden Bildern und für ungerade Zeilen in ungeraden Bildern *wahr*. Der Kern des *Interlacing*-Verfahrens wird durch diesen Ausdruck abgebildet [2].

In diesem Kapitel werden verschiedene *Deinterlacing*-Verfahren vorgestellt. Diese werden in zwei große Gruppen unterteilt, die *non-Motion Compensating* (6.1) und die *Motion Compensating* (6.2) Verfahren (analog zu [2]).
Die verschiedenen Algorithmen sind zu unterschiedlichen Zeiten entstanden und vor allem mit unterschiedlichen Zielen. So wurden insbesondere die *linearen Verfahren* in den Anfängen der Computertechnik entwickelt. Ziel dieser Verfahren war keine herausragende Bildqualität, sondern die Ausführbarkeit des Algorithmus in Echtzeit auf den damals noch vergleichsweise rechenschwachen Computern. Im Gegensatz dazu gibt es heutzutage

andere Verfahren, die ein möglichst perfektes Ausgangsbild als Ziel haben. Zunächst unabhängig von Rechenzeit werden Bewegungsschätzungen bestimmter Objekte im Bild vorgenommen und mit in die Berechnung einbezogen.
Diese Arbeit wird sich im weiteren Verlauf auf die qualitätsorientierten Verfahren stützen. Die *Motion Compensating* Verfahren produzieren die besten Ergebnisse [19]. Aus diesem Grund werden die *non-Motion Compensated* Verfahren in diesem Kapitel der Vollständigkeit halber zwar erwähnt und das jeweilige Prinzip kurz erläutert, eine umfassende Diskussion der Verfahren findet jedoch nicht statt. Weitere Informationen zu diesen Verfahren finden sich in [2].

6.1 Non-Motion Compensating Verfahren

Die in diesem Abschnitt vorgestellten Verfahren vernachlässigen allesamt die Bewegung innerhalb des Bildes. Die Verfahren wurden geschichtlich gesehen als erste entwickelt. Die Herangehensweise bringt den Vorteil der vergleichsweise einfachen Implementation mit sich. Zudem wird wenig Rechenleistung benötigt. Analog zu [2] werden die Verfahren in *lineare* (6.1.1) und *non-lineare* (6.1.2) Verfahren unterschieden.

6.1.1 Lineare Verfahren

Es gibt drei Varianten von linearen Verfahren. Zum einen kann *Deinterlacing inter-Field*, also innerhalb eines Bildes, stattfinden. Diese Gruppe wird als räumliches (*spatial*) *Interlacing* bezeichnet. Wird zum *Deinterlacing* die Information aus einem benachbarten Bild extrahiert (*intra-Field*) so spricht man von *temporal Deinterlacing*. Auch eine Kombination beider Verfahren (*spatio-temporal*) ist möglich.

Line Doubling (spatial Deinterlacing)

Beim *Line Doubling* wird jede nicht vorhandene Zeile eines Bildes durch die Kopie der darüber oder darunter liegenden Zeile gefüllt (siehe Abbildung 6.1).

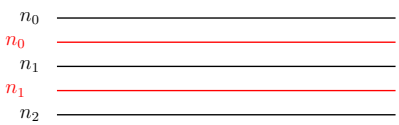


Abbildung 6.1
Line Doubling Verfahren

Line Average (spatial Deinterlacing)

Beim *Line Average* Verfahren wird ein Durchschnittswert aus der oberen und der unteren Zeile gebildet. Die Berechnung erfolgt für jeden Pixel separat. Das Prinzip ist in Abbildung 6.2 dargestellt.

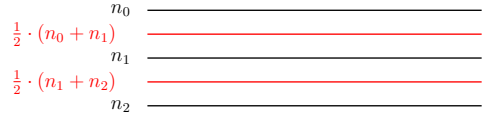


Abbildung 6.2
Line Average Verfahren

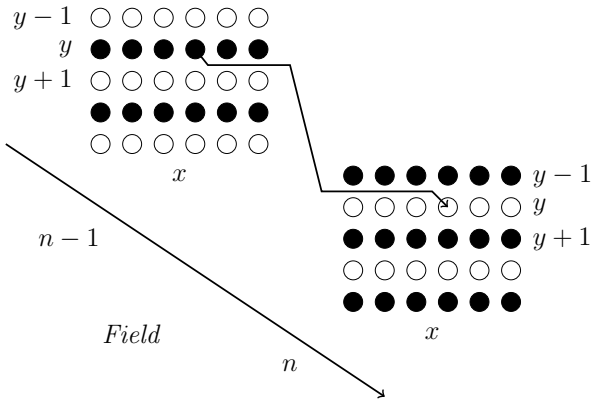


Abbildung 6.3
Field Repetition Verfahren (angelehnt an [2])

Field Repetition (Temporal Deinterlacing)

Analog zum *Line Doubling* Ansatz wird in diesem *inter-Frame* Ansatz eine Bildzeile wiederholt. Bei der *Field Repetition* Methode wird allerdings nicht die räumlich benachbarte Zeile genommen, sondern die zeitlich benachbarte aus dem vorangegangenen Halbbild. Abbildung 6.3 verdeutlicht dieses Verfahren.

Field Average (Temporal Deinterlacing)

Ebenfalls kann analog zum *Line Average* Verfahren ein *temporal* Verfahren angewandt werden. In diesem Fall wird für die zu interpolierende Zeile ein Mittelwert zwischen der Zeile an identischer Position im vorangegangenen und nachfolgenden Halbbild gebildet (siehe Abbildung 6.4).

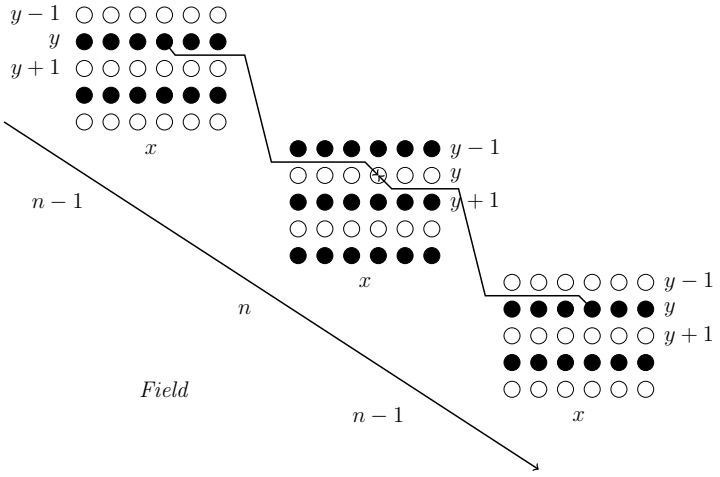


Abbildung 6.4
Field Averaging Verfahren (angelehnt an [2])

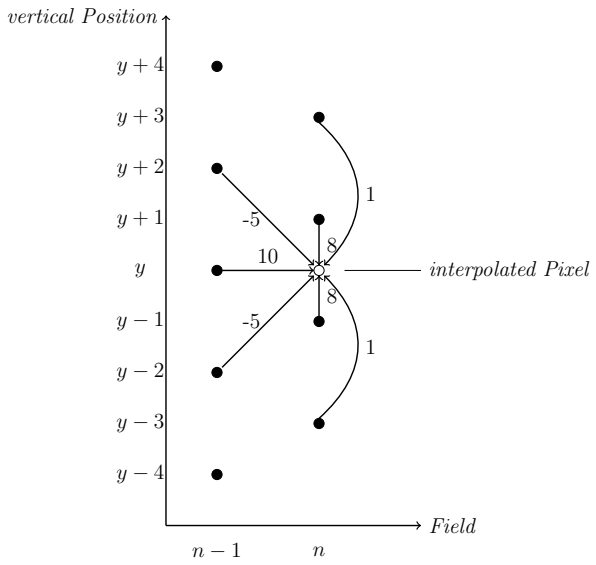


Abbildung 6.5
Spatio-Temporal Verfahren (angelehnt an [2])

Vertical-Temporal (Spatio-Temporal Deinterlacing)

Sowohl die *intra-Field* als auch die *inter-Field* Verfahren haben große Nachteile. Diese können durch eine Kombination beider Ansätze minimiert werden. Der *vertical-temporal* Filter berücksichtigt sowohl Pixel des vorangegangen Bildes, als auch des aktuellen Bildes. Abbildung 6.5 verdeutlicht das Verfahren.

6.1.2 Non-lineare Verfahren

Es zeigt sich, dass die linearen Verfahren eine Reihe von Artefakten aufweisen. Beim *Line Doubling* und *Line Average* werden die Bilder unschärfer, wohingegen *Field Repetition* und *Field Average* Probleme bei hohen vertikalen Details haben. Aus diesem Grund wurden die nicht-linearen Verfahren entwickelt, die sich strategisch auf die Vermeidung von Unschärfe und Detailverlust beim *Deinterlacing* stützen.

Motion-Adaptive Verfahren

Um ein *Deinterlacing*-Verfahren an die Bewegung in einem Bild anzupassen, ist es essenziell zu wissen, wann Bewegung auftritt. Die Bewegungserkennung an sich ist nicht trivial, da das ursprüngliche Signal durch Rauschen, Übersprechen bei analogen TV-Systemen, nachträglicher Filterung oder Alias, das durch das *Interlacing* entstanden ist, sogar ohne Bewegungen Artefakte aufweist.

Aus diesem Grund können mögliche Bewegungen lediglich geschätzt werden unter der Prämisse, dass

- das Rauschen und Alias im Vergleich zum unbeeinträchtigen Videosignal klein sind,
- die sich bewegenden Objekte im Vergleich zu den Pixeln groß sind.

[2] beschreibt viele einfache Verfahren mit Hilfe derer Bewegung in einer Bildfolge geschätzt werden kann. In der Regel beruhen diese Verfahren auf einer Analyse der Frequenzen in zwei Halbbildern, bzw. des zusammengesetzten Vollbildes. Je nachdem, ob Bewegung vorhanden ist oder nicht, wird in der Folge ein räumliches oder zeitliches *Deinterlacing* auf die jeweilige Bildstelle angewendet.

Edge-Oriented Dependent Verfahren

Alle bisher beschriebenen Verfahren berücksichtigen lediglich die y/t-Ebene. Es ist aber insbesondere für diagonale Kanten von großem Vorteil auch die x-Ebene in die Berechnungen mit einzubeziehen um eine zusätzliche Anpassungsmöglichkeit zu schaffen. Der *Edge-Oriented Dependent* Algorithmus wurde ursprünglich für Bildsequenzen mit Bewegung entwickelt. Er beschränkt sich auf *intra-Frame* Bearbeitung. Somit werden Bewegungsartefakte durch Objektverschiebung zwischen den Bildern vermieden. Sind diagonale Kanten in einem Bild vorhanden, so können diese durch eine andere Pixelkombination interpoliert werden.

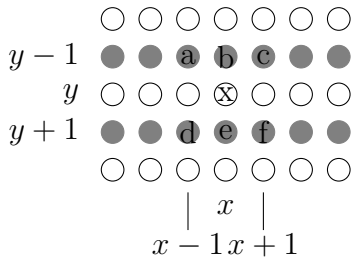


Abbildung 6.6
Edge-Oriented Dependent Verfahren
(angelehnt an [2])

In Abbildung 6.6 wird das Prinzip verdeutlicht. Durch das x ist das zu interpolierende Pixel gekennzeichnet. Die Buchstaben zeigen die zur Interpolation herangezogenen benachbarten Pixel auf. Das Endergebnis X ist definiert als:

$$X = \begin{cases} X_a, & ((|a-f| < |c-d|) \wedge (|a-f| < |b-e|)) \\ X_c, & ((|c-d| < |a-f|) \wedge (|c-d| < |b-e|)) \\ X_b, & (\text{otherwise}) \end{cases} \tag{6.3}$$

In Formel 6.3 sind X_a, X_b und X_c definiert als:

$$X_a = \frac{a+f}{2}, X_b = \frac{b+e}{2}, X_c = \frac{c+d}{2} \tag{6.4}$$

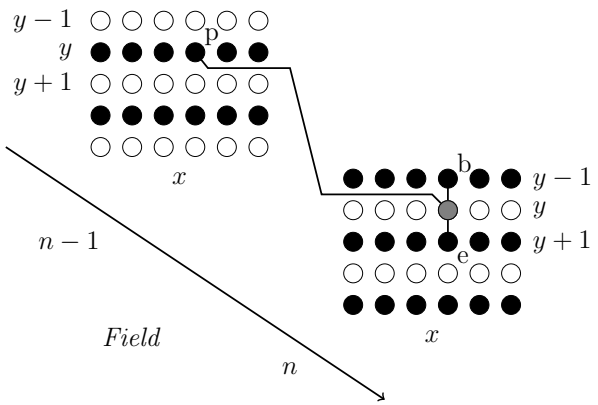


Abbildung 6.7
Median Verfahren (angelehnt an [2])

Median Verfahren

Das *Median* Verfahren (auch *Implicitly Adapting Algorithm*) wurde designt um automatisch zwischen einer *intra-Frame* und einer *inter-Frame* Interpolation umzuschalten. Die Entscheidung welche Methode anzuwenden ist, fällt auf Pixelbasis. Formel 6.5 beschreibt die Kriterien, nach denen die Interpolation stattfindet. In Abbildung 6.7 ist das Verfahren dargestellt. Zur Bestimmung des Pixelwertes werden sowohl die beiden vertikal benachbarten Pixel in Betracht gezogen, als auch das Pixel an identischer Position des vorangegangenen Halbbildes. Es wird der Wert eines dieser Pixel genommen. Das Verfahren entscheidet lediglich welcher Wert zu nehmen ist. Durchschnittswerte oder Ähnliches werden nicht kalkuliert.

$$MED\{b,e,p\} = \begin{cases} b, & ((e < b < p) \vee (p < b < e)) \\ e, & ((b \leq e \leq p) \vee (p \leq e \leq b)) \\ p, & (\text{otherwise}) \end{cases} \tag{6.5}$$

Hybride Algorithmen

Die oben beschriebenen Verfahren können auf vielfältige Art und Weise miteinander kombiniert werden. Es gibt verschiedene Filterkerne, die größere und kleinere Bereiche einbeziehen und diese anders gewichten. Außerdem gibt es Ansätze, zunächst bestimmte Vorberechnungen durchzuführen und mit den gefunden Werten das Median Verfahren auszuführen. Eine Übersicht ist in [2] zu finden.

6.2 Motion Compensating Verfahren

Die höchste Bildqualität beim *Deinterlacing* wird mit den *Motion Compensating* Verfahren erzielt. Diese Verfahren berücksichtigen bei ihrem Vorgehen die Bewegungen der einzelnen Objekte innerhalb des Bildes. Zu diesem Zweck werden zunächst Bewegungsvektoren (siehe Kapitel 5) bestimmt. Mit Hilfe dieser Bewegungsvektoren wird versucht, die Position des Objektes zum Zeitpunkt der Interpolation zu bestimmen und somit eine möglichst realistische Bildberechnung zu erlagen.
Im Vergleich zu den oben beschriebenen *non-Motion Compensating* Verfahren wird bei diesem Ansatz auf einem physikalischen Hintergrund aufgebaut. Es wird versucht, den tatsächlichen Zustand der Szene und der Objekte zu bestimmen. Bisher wurde lediglich ein statistischer Ansatz gewählt, bei dem Pixel und deren Nachbarzustände betrachtet wurden.
Auch wenn die *Motion Compensating* Verfahren in der Vorhersage und der Rekonstruktion vieler Szenen deutlich genauer sind, stoßen allerdings auch diese Verfahren in bestimmten Situationen an ihre Grenzen. Szenenwechsel oder *Fades* sind beispielsweise über Bewegungsvektoren nicht darzustellen [2].

6.2.1 Direkte Methoden

Der Übergang von einem *non-Motion Compensating* zu einem *Motion Compensating* Verfahren wird durch die Änderung der Pixelreferenzierung von $F(\vec{x}, n+m)$ zu $F(\vec{x} + m\vec{d}(\vec{x}, n), n+m)$ beschrieben. In diesem Zusammenhang bezeichnet $\vec{d}(\vec{x}, n)$ den Richtungsvektor, der über die Bewegungsschätzung auf Pixel oder sogar Subpixel-Basis bestimmt wurde. *Deinterlacing*-Verfahren, die über einfaches Austauschen der beiden Referenzierungen in *Motion Compensating* Verfahren umgewandelt werden können, werden „direkte Methoden “genannt und im Folgenden beschrieben [2].

MC Field Insertion

Die einfachste *Motion Compensating* Methode ist das *MC Field Insertion* Verfahren. Wie beim *Field Repetition* Verfahren wird auch hier die Zeile des vorangegangen Halbbildes, die an der selben Position ist wie die zu Interpolierende im aktuellen Bild, als Ausgangspunkt genommen. Um dem aktuellen zu interpolierenden Pixel einen Wert zuweisen zu können, wird die Position des Pixels im vorhergehenden Halbbild genommen und diese um den Bewegungsvektor korrigiert.

$$F_i(\vec{x}, n) = F(\vec{x} - \vec{d}(\vec{x}, n), n-1) \tag{6.6}$$

Das Verfahren wird in Abbildung 6.8 verdeutlicht.

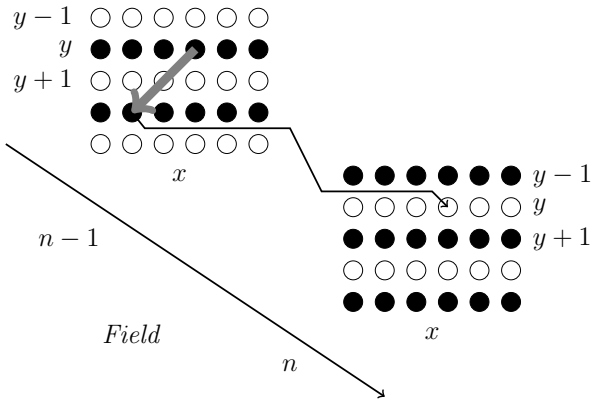


Abbildung 6.8
MC Field Insertion Verfahren (angelehnt an [2])

MC Field Averaging Ebenfalls gibt es analog zum *Field Averaging* ein *MC Field Averaging*. Formel 6.7 beschreibt die Berechnungsvorschrift für diese Berechnungsart, die in Abbildung 6.9 verdeutlicht wird.

$$F_i(\vec{x}, n) = \frac{F(\vec{x} - \vec{d}(\vec{x}, n), n-1) + F(\vec{x} + \vec{d}(\vec{x}, n), n+1)}{2} \tag{6.7}$$

Diese zwei Berechnungsarten bringen jedoch einige Probleme mit sich. Beide Verfahren sind anfällig gegenüber falschen Bewegungsvektoren. Die Wahrscheinlichkeit für einen falschen Bewegungsvektor kann durch größere Filter mit einem erweiterten zeitlichen Abstand minimiert werden. Jedoch wird der Vektor durch dieses Vorgehen ungenauer gegenüber der tatsächlichen Bewegung und der Aufwand der Berechnung steigt.
Ein weiteres Problem, das bei der *MC Field Averaging* Methode durch die Mittlung von zwei Pixeln auftritt, ist die Rauschreduktion. Diese kann sich insofern störend auswirken als das Endbild eine Mischung aus rauschreduzierten und nicht rauschreduzierten Zeilen ist [2].

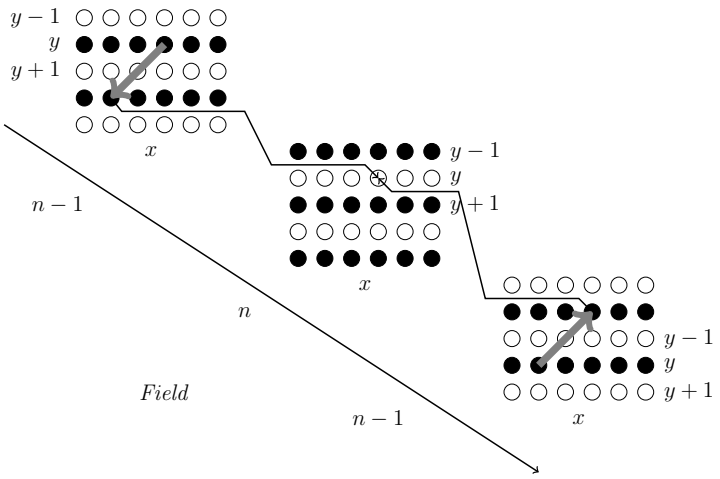


Abbildung 6.9
MC Field Averaging Verfahren (angelehnt an [2])

MC Median

Das *MC Median* Verfahren ist das Pendant zum *Median* Ansatz. Es funktioniert auf dieselbe Art und Weise, nur der Bildpunkt des vorhergehenden Bildes wird auf eine *Motion Compensating* Art und Weise errechnet (siehe Abbildung 6.10). Folgende Formel kann mit Formel 6.5 aufgelöst werden, $\vec{u}_y$ bezeichnet eine Verschiebung um eine Zeile:

$$F_i(\vec{x}, n) = MED\{F(\vec{x} - \vec{u}_y, n), F(\vec{x} + \vec{u}_y, n), F(\vec{x} - \vec{d}(\vec{x}, n), n - 1)\} \tag{6.8}$$

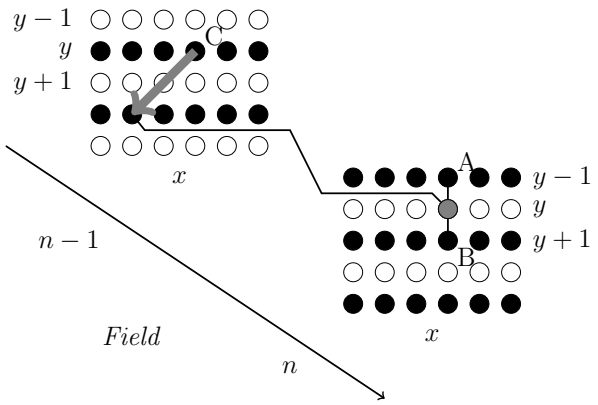


Abbildung 6.10
MC Median Verfahren (angelehnt an [2])

6.2.2 Hybride Methoden

Beliebige Kombinationen, auch zwischen den beiden Bereichen *non-Motion Compensating* und *Motion Compensating* wurden entwickelt. In [2] wird ein Algorithmus beschrieben, bei dem folgende vier Verfahren kombiniert wurden:

- *Line Average* ($F_1(\vec{x}, n)$),
- *Edge-Oriented Dependent Interpolation* ($F_2(\vec{x}, n)$),
- *Field Average* ($F_3(\vec{x}, n)$),
- *MC Field Average* ($F_4(\vec{x}, n)$)

Die vier Verfahren können über die Gewichtungsfaktoren k_i kombiniert werden:

$$F_i(\vec{x}, n) = \sum_{j=1}^{4} k_j F_j(\vec{x}, n) \tag{6.9}$$

6.2.3 Temporal Backward

Wie man in Abbildung 6.11 erkennen kann, bezieht dieses Verfahren nicht nur das direkt vorangehende Halbbild $n-1$, sondern ebenfalls das davor stehende Halbbild $n-2$ mit ein. Das Verfahren analysiert die Bewegungsvektoren und entscheidet anhand eines Schwellenwertes auf welchen Pixel ein Bewegungsvektor zeigt. Zeigt dieser auf das vorangehende Bild $n-1$ wird der Pixel aus diesem Bild als Interpolationsgrundlage genommen. Zeigt der Bewegunsvektor allerdings genau zwischen zwei der Pixel aus dem $n-1$ Bild, so wird analysiert ob der Vektor eventuell - innerhalb einer Fehlertoleranz - auf ein Pixel aus dem $n-2$ Bild zeigt. In diesem Fall wird entsprechend dieses Pixel verwendet [2].

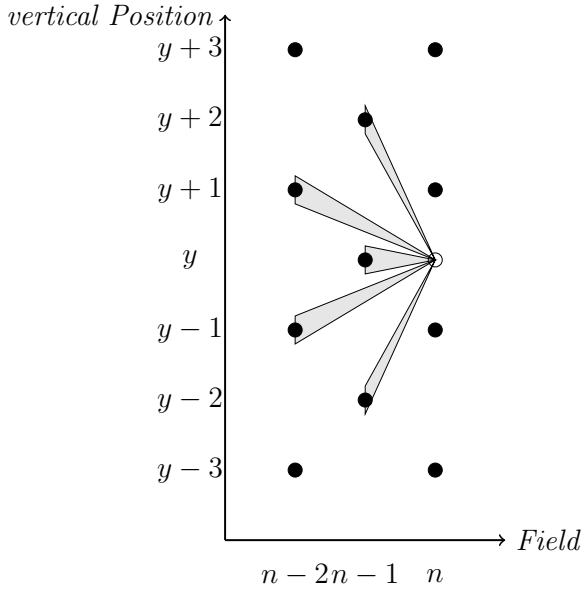


Abbildung 6.11
Temporal Backward Verfahren (angelehnt an [2])

6.2.4 Time-Recursive

Dieser Ansatz benutzt eine rekursive Methode für das *Deinterlacing*. Anstatt lediglich mit den vorhandenen Halbbildern zu arbeiten, wird das vorherige, bereits rekonstruierte Vollbild als Ausgangspunkt für die Interpolation genommen:

$$F_i(\vec{x}, n) = F_{out}(\vec{x} - \vec{d}(\vec{x}, n), n-1) \tag{6.10}$$

Solange das vorherige Bild fehlerfrei rekonstruiert wurde und die Bewegungsvektoren richtig berechnet wurden funktioniert dieses Verfahren. Gibt es allerdings Rechenfehler im $n-1$ Bild werden diese weiter fortgepflanzt. Aus diesem Grund gibt es eine Erweiterung dieses Verfahrens, das den *MC Median* Algorithmus mit einbindet:

$$F_i(\vec{x}, n) = MED\{F_{out}(\vec{x} - \vec{d}(\vec{x}, n), n-1), F(\vec{x} - \vec{u}_y, n), F(\vec{x} + \vec{u}_y, n)\} \tag{6.11}$$

6.2.5 Adaptive-Recursive

Der *adaptive-recursive* Ansatz ist eine Erweiterung eines beliebigen Verfahrens um eine nachträgliche Filterung, durch die alias-bedingte Ungleichheiten entlang der Bewegungslinie von Objekten herausgefiltert werden sollen. Für diese Filterung werden originale wie interpolierte Pixel gleichermaßen angepasst. Die Anwendung dieses Filters erster Ordnung lässt sich folgendermaßen beschreiben:

$$F_{out}(\vec{x},n) = \begin{cases} k(\vec{x},n)F(\vec{x},n) + (1-k(\vec{x},n))F_{out}(\vec{x}-\vec{d}(\vec{x},n),n-1), & (y \bmod 2 = n \bmod 2) \\ p(\vec{x},n)F_{init}(\vec{x},n) + (1-p(\vec{x},n))F_{out}(\vec{x}-\vec{d}(\vec{x},n),n-1), & (\text{otherwise}) \end{cases} \tag{6.12}$$

Der Ausdruck $y \; mod \; 2 = n \; mod \; 2$ ist bereits aus Formel 6.2 bekannt. Dementsprechend wird der obere Teil dieser Formel genau dann benutzt, wenn es sich um eine Zeile handelt, in der originale Informationen vorhanden sind. Wie oben beschrieben, werden bei diesem Verfahren allerdings auch diese Pixel behandelt und angeglichen. Dementsprechend sieht man in der Formel 6.12 eine Summe aus den aktuellen Pixeln und denen des vorhergehenden Bildes, die durch den Bewegungsvektor gefunden wurden. Die beiden Pixelinformationen fließen anteilig, bestimmt durch den Faktor $k(\vec{x},n)$ in das Endergebnis ein. Bei der Definition von $k(\vec{x},n)$, wird sich die Annahme, dass die Pixelinformationen aus dem n und $n-1$ Bild identisch sind und lediglich durch den Bewegungsvektor getrennt sind, zunutze gemacht. Die Differenz zwischen dem aktuellen und dem vorhergehenden Pixel ist somit eine Art Glaubwürdigkeitswert des Bewegungsvektors:

$$k(\vec{x},n) = CLIP\left(0,1,\alpha\sqrt{|F_{out}(\vec{x}-\vec{d}(\vec{x},n),n-1) - F(\vec{x},n)|}\right) \tag{6.13}$$

α ist lediglich ein Scaling-Faktor und die Funktion $CLIP(0,1,\eta)$ ist definiert als:

$$CLIP(0,1,\eta) = \begin{cases} 0, & (\eta < 0) \\ 1, & (\eta > 1) \\ \eta & (otherwise) \end{cases} \tag{6.14}$$

Der untere Teil der Formel 6.12 wird in den Fällen in Anspruch genommen, in denen keine originalen Informationen vorhanden sind und diese interpoliert werden müssen. In diesen Zeilen findet dementsprechend das eigentliche *Deinterlacing* statt. $F_{init}(\vec{x},n)$ ist in diesem Fall ein erstes Deinterlacing, das mit einem anderen Verfahren durchgeführt wurde. Das Gesamtergebnis setzt sich aus der Summe dieses ersten *Deinterlacing* und dem um den Bewegungsvektor verschobenen Inhalt des vorherigen Bildes zusammen.

Wie in [2] wird $F_{init}(\vec{x}, n)$ aus den beiden Verfahren *Median* und *Field Average* zusammengesetzt:

$$F_{init}(\vec{x}, n) = \begin{cases} MED\{F(\vec{x} - \vec{u}_y, n), F(\vec{x}, n-1), F(\vec{x} + \vec{u}_y, n)\}, & (D_{MED} + C_p \leq D_{av}) \\ \frac{F(\vec{x}-\vec{u}_y)+F(\vec{x}+\vec{u}_y,n)}{2}, & (\text{otherwise}) \end{cases} \tag{6.15}$$

An dieser Stelle wird durch den Term $(D_{MED} + C_p \leq D_{av})$ bereits eine Vorauswahl getroffen, welches der beiden Verfahren das bessere in diesem Fall ist. Die Elemente D_{MED} und D_{av} sind dementsprechend wieder als Glaubwürdigkeitswert für den Bewegungsvektor zu sehen und als Differenz zwischen dem interpolierten Wert und dem um den Bewegungsvektor bereinigten Wert des vorhergehenden Bildes definiert:

$$D_{MED} = |F_{out}(\vec{x} - \vec{d}(\vec{x}, n), n-1) - MED\{F(\vec{x} - \vec{u}_y), F(\vec{x}, n-1), F(\vec{x} + \vec{u}_y, n)\}| \tag{6.16}$$

$$D_{av} = |F_{out}(\vec{x} - \vec{d}(\vec{x}, n), n-1) - \left(\frac{F(\vec{x} - \vec{u}_y) + F(\vec{x} + \vec{u}_y, n)}{2}\right)| \tag{6.17}$$

Der Wert C_p ist in diesem Fall ein festgelegter Faktor. Dieser stellt die Grenze zwischen den beiden zur Verfügung stehenden Verfahren her.

Der Faktor $p(\vec{x}, n)$ wird in dieser Formel anders bestimmt als der Faktor $k(\vec{x}, n)$. Grund dafür ist, dass das Ergebnis zu stark von der Qualität des *Deinterlacing* abhängt. Aus diesem Grund ist diese Gewichtung an die Ungleichheit der benachbarten originalen Pixel gekoppelt[2]:

$$|F_{out}(\vec{x} - \vec{d}(\vec{x}, n), n-1) - F_{out}(\vec{x}, n)| = \frac{A + B}{2} \tag{6.18}$$

mit

$$A = |F(\vec{x} - \vec{u}_y, n) - F_{out}(\vec{x} - \vec{d}(\vec{x}, n) - \vec{u}_y, n-1)| \tag{6.19}$$

$$B = |F(\vec{x} + \vec{u}_y, n) - F_{out}(\vec{x} - \vec{d}(\vec{x}, n) + \vec{u}_y, n-1)| \tag{6.20}$$

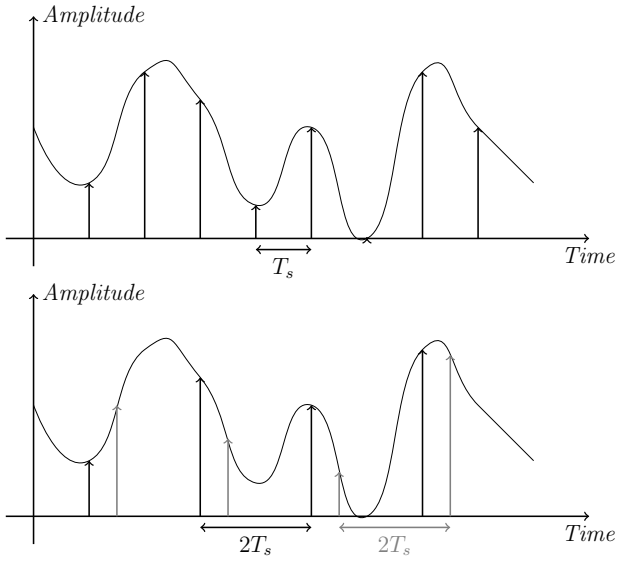


Abbildung 6.12
Zur Rekonstruktion nötige
Sets of Samples ($N = 2$)

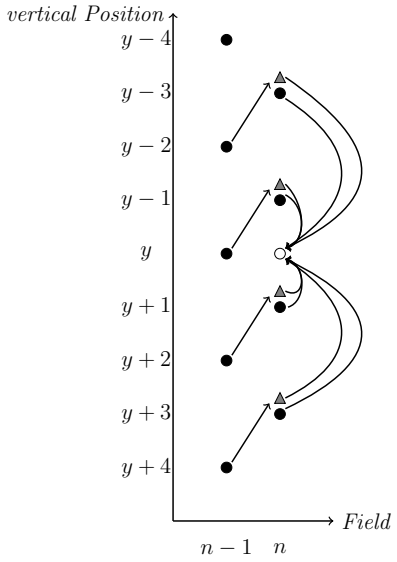


Abbildung 6.13
Transversal Generalized Ansatz -
zwei *Sets of Samples* (angelehnt an [2])

Der Faktor $p(\vec{x}, n)$ ist schließlich definiert als:

$$p(\vec{x}, n) = CLIP\left(0, 1, \frac{|A + B| + \delta}{2|F_{init}(\vec{x}, n) - F_{out}(\vec{x} - \vec{d}(\vec{x}, n), n - 1)| + \delta}\right) \tag{6.21}$$

Der zusätzliche Operand δ ist lediglich zur Vermeidung von Null-Divisionen und der identischen Filterung von benachbarten Pixeln bei kleinem Zähler und Nenner zuständig.

6.2.6 „Transversal“Generalized Sampling Theorem

Der *Transversal Generalized* Algorithmus macht sich eine Generalisierung des Abtasttheorems von Yen aus dem Jahr 1956 (siehe auch [30]) zunutze. Demnach kann ein Signal mit einer maximalen Frequenz von $0.5f_s$ durch N von einander unabhängigen Datensätzen wiederhergestellt werden, solange jeder Datensatz eine maximale Frequenz von $\frac{f_s}{N}$ besitzt. Dabei ist es unerheblich an welcher Stelle sich die *Samples* der Datensätze befinden. Abbildung 6.12 verdeutlicht diese Ausgangslage.
Diese theoretischen Ansätze können auf das *Deinterlacing*-Problem übertragen werden. In Abbildung 6.13 wird verdeutlicht, dass durch den Bewegungsvektor und das vorangegangene Halbbild ein zweites Set an Samples für das aktuelle Halbbild berechnet werden kann. Innerhalb der obigen Überlegungen nimmt in diesem Fall $N = 2$ an.

Der große Vorteil dieses Verfahrens ist die nicht vorhandene Fehlerfortpflanzung. Im Vergleich zu rekursiven Verfahren wird an dieser Stelle zu jeder Zeit mit neu kalkulierten Werten gerechnet und nicht auf bereits vorher errechnete Ergebnisse zurückgegriffen. Eine

mögliche Fehlerquelle ist die Fehlkalkulation der Bewegungsvektoren, die den zweiten Datensatz verfälschen.

Da dieses Verfahren das *Generalized Sampling Theorem* ausnutzt um in vertikaler Richtung zu interpolieren, wird es *„Transversal"Generalized Sampling Theorem (TGST)* genannt. Mit Hilfe dieses Theorems wird das ursprüngliche Signal im Folgenden rekonstruiert.

Wie aus Formel 6.2 zu sehen ist, setzt sich das Ausgangsbild F_{out} aus den Zeilen des Originalbildes F_o und den interpolierten Zeilen F_i zusammen. Letztere ergeben sich in diesem Verfahren durch die oben angesprochene Rekonstruktion aus aktuellen Samples und denen des vorangegangenen Bildes, die durch den Bewegungsvektor angepasst wurden [2]:

$$F_i(\vec{x},n) = \sum_k F(\vec{x} - (2k+1)\vec{u}_y, n)h_1(k,\delta_y) + \sum_m F(\vec{x} - \vec{e}(\vec{x},n) - 2m\vec{u}_y, n-1)h_2(m,\delta_y) \tag{6.22}$$

Innerhalb dieser Formel 6.22 sind h_1 und h_2 die GST Filter, die im weiteren Verlauf noch näher erläutert werden. Zudem wird in diesem Fall ein bearbeiteter Bewegungsvektor $\vec{e}(\vec{x},n)$ eingesetzt. Dieser wird folgendermaßen gebildet:

$$\vec{e}(\vec{x},n) = \begin{pmatrix} d_x(\vec{x},n) \\ 2ROUND\left(\frac{d_y(\vec{x},n)}{2}\right) \end{pmatrix} \tag{6.23}$$

$\vec{e}(\vec{x},n)$ setzt sich also zusammen aus dem x-Anteil des Bewegungsvektors und dem Doppelten der auf einen Integer gerundeten Hälfte des y-Anteils. Letzteres dient zur Sicherstellung, dass die Samples im zweiten Datensatz alle denselben Abstand zueinander haben und somit für das *Generalized Sampling Theorem* zu verwenden sind.
Die vertikale Bewegung δ_y, der durch die Filterfunktionen h_1 und h_2 berücksichtigt wird, ergibt sich über:

$$\delta_y(\vec{x},n) = |d_y(\vec{x},n) - 2ROUND\left(\frac{d_y(\vec{x},n)}{2}\right)| \tag{6.24}$$

Es wird an dieser Stelle angenommen, dass es sich bei dem aktuellen Bild um eines mit den ungeraden Zeilen handelt F^o. Das zu berechnende Bild entspricht somit dem aktuellen Halbbild mit den geraden Zeilen $F_i = F^e$. Zudem wird die zweidimensionale Operation aus Formel 6.22 in eine eindimensionale Operation umgewandelt. Es wird zunächst nur die y-Richtung betrachtet in der Annahme, dass Berechnungen in der x-Domäne in einem folgenden Schritt gemacht würden. Somit vereinfacht sich die Formel zu:

$$F^e(y,n) = \sum_k F(y-(2k+1),n)h_1(k) + \sum_m F(y-e_y-2m,n-1)h_2(m) \tag{6.25}$$

Nimmt man an dieser Stelle theoretisch an es gäbe ein progressives Bild F^p, so könnte F^e auch bestimmt werden als:

$$F^e(y,n) = \sum_k F(y-k,n-1)h(k) \tag{6.26}$$

Da Filter und insbesondere Filterkoeffizienten besonders gut in der z-Darstellung bestimmt werden können, kann Formel 6.26 auch folgendermaßen dargestellt werden:

$$F^e(z,n) = (F^p(z,n-1)H(z))_e = F^o(z,n-1)H^o(z) + F^e(z,n-1)H^e(z) \tag{6.27}$$

Analog zu diesem *even Field* kann auch das *odd Field* folgendermaßen bestimmt werden:

$$F^o(z,n) = (F^p(z,n-1)H(z))_o = F^o(z,n-1)H^e(z) + F^e(z,n-1)H^o(z) \tag{6.28}$$

Formel 6.28 kann umgeformt werden zu:

$$F^o(z,n-1) = \frac{F^o(z,n) - F^e(z,n-1)H^o(z)}{H^e(z)} \tag{6.29}$$

Durch Substitution von 6.29 in 6.27 erhält man:

$$F^e(z,n) = \frac{H^o(z)}{H^e(z)}F^o(z,n) + \left(H^e(z) - \frac{(H^o(z))^2}{H^e(z)}\right)F^e(z,n-1) \tag{6.30}$$

Schlussendlich ergeben sich $H_1(z)$ und $H_2(z)$ also zu:

$$H_1(z) = \frac{H^o(z)}{H^e(z)} \tag{6.31}$$

$$H_2(z) = H^e(z) - \frac{(H^o(z))^2}{H^e(z)} \tag{6.32}$$

TGST mit Interpolation erster Ordnung

Bei der *TGST*-Interpolation erster Ordnung wird die oben noch allgemein ausgedrückte Interpolation mit einem Interpolations-Filter erster Ordnung $H(z) = (1-\delta_y) + \delta_y z^{-1}$ mit $0 \leq \delta_y \leq 1$ realisiert [2].

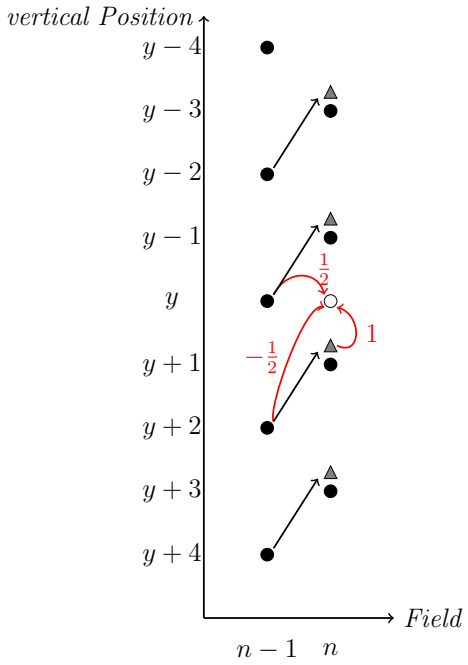


Abbildung 6.14
TGST mit *First Order Interpolation* (angelehnt an [2])

Somit ergibt sich:

$$H_1(z) = \frac{\delta_y}{1-\delta_y} z^{-1} \tag{6.33}$$

$$H_2(z) = (1-\delta_y) - \frac{(\delta_y)^2}{1-\delta_y} z^{-2} \tag{6.34}$$

Der Faktor δ_y ein Indikator für Bewegung. Angenommen es gäbe eine Bewegung von 0.5 Pixel pro Halbbild, also $\delta_y = 0.5$, so ergäbe sich mit Formel 6.30:

$$F^e(z,n) = z^{-1}F^o(z,n) + \frac{1}{2}(1-z^{-2})F^e(z,n-1) \tag{6.35}$$

Benutzt man an dieser Stelle die inverse Z-Transformation, um zurück in die räumliche y-Domäne zu gelangen, so ergibt sich:

$$F^e(y,n) = F^o(y+1,n) + \frac{1}{2}F^e(y,n-1) - \frac{1}{2}F^e(y+2,n-1) \tag{6.36}$$

Das gesuchte Pixel setzt sich also aus einem Pixel im aktuellen Bild und zwei Pixeln aus dem vorangegangenen Halbbild zusammen. In Abbildung 6.14 wird verdeutlicht, welche Pixel in die Berechnung mit einfließen [2].

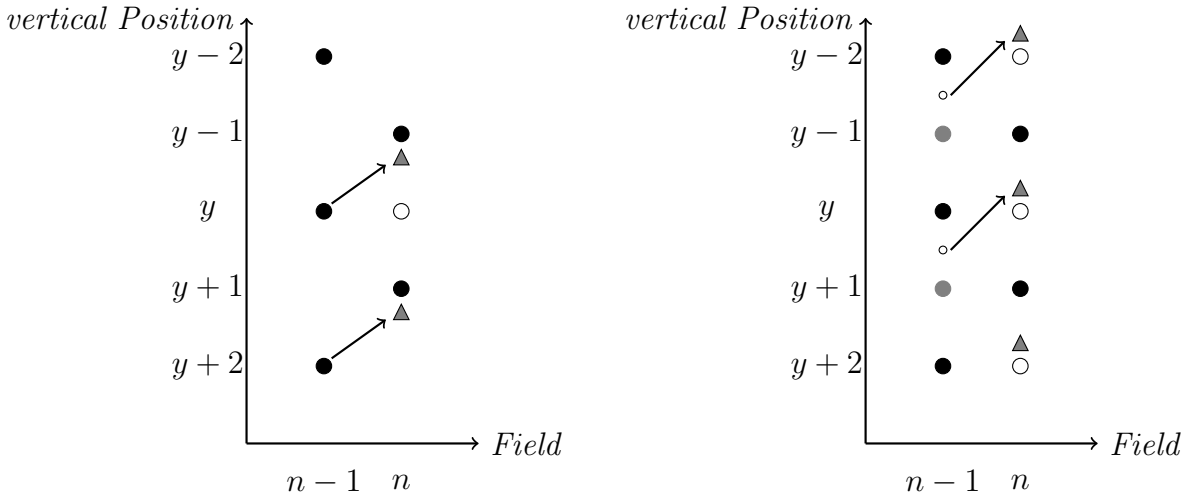


Abbildung 6.15
RGST (angelehnt an [2])

6.2.7 „Recursive“Generalized Sampling Theorem

Der *TGST* Algorithmus basiert auf einer Bewegungsschätzung des vorangegangenen Bildes in Kombination mit den Pixeldaten des aktuellen Bildes. Das Verfahren wird problematischer, je enger die Pixelabstände sind. Der Bewegungsvektor ist immer fehlerbehaftet durch Rauschen und Kalkulationsprobleme. Die relative Genauigkeit sinkt also mit zunehmendem Pixelabstand.
Um diesem Problem entgegen zu wirken, wurde das *„Recursive“Generalized Sampling Theorem* Verfahren entwickelt. Dieses funktioniert im Wesentlichen genauso wie der *„Transversal “Generalized Sampling Theorem* Algorithmus. Jedoch wird für die Berechnung nicht das vorausgehende Halbbild benutzt, sondern das bereits interpolierte vorausgehende Vollbild. Das Verfahren benutzt also wiederum den rekursiven Ansatz. Bedingt dadurch ist eine gewisse Fehlerfortpflanzung innerhalb des Verfahrens vorhanden, diese ist laut Tests in [2] allerdings weniger einflussreich, als der durch falsche Bewegungsvektoren entstandene Fehler.

Abbildung 6.15 verdeutlich das Verfahren. Im Gegensatz zum *TGST* werden im vorhergehenden Bild Sub-Pixel berechnet. Durch die Berechnung werden geringe Pixelunterschiede vermieden. In diesem Beispiel wird das Sub-Pixel anhand des Bewegungspfades verschoben. Die Formel 6.22 ändert sich somit zu:

$$F_i(\vec{x},n) = \sum_k F(\vec{x}-(2k+1)\vec{u}_y,n)h_1(k,\delta_y) + \sum_m F_{out}(\vec{x}-\begin{pmatrix} d_x(\vec{x},n) \\ \rho_y \end{pmatrix} - 2m\vec{u}_y, n-1)h_2(m,\rho_y) \tag{6.37}$$

In [2] wird davon ausgegangen, dass der Bewegungsvektor eine Auflösungsgenauigkeit von einem viertel Pixel besitzt. Demnach ist ρ_y definiert als:

$$\rho_y = \begin{cases} e_y, & (\delta_y = 0 \vee \delta_y = \pm\frac{1}{4}) \\ e_y + \frac{1}{4}Sign(\delta_y), & (\delta_y = \pm\frac{1}{2}) \\ e_y + \frac{1}{2}Sign(\delta_y), & (otherwise) \end{cases} \tag{6.38}$$

Innerhalb dieser Definition ist e_y die ganzzahlige vertikale Bewegung $e_y = 2Round(\frac{d_y}{2})$, δ_y der dabei entstehende Fehler $\delta_y = e_y - d_y$ und

$$Sign(s) = \begin{cases} -1, & (s < 0) \\ 1, & (otherwise) \end{cases} \tag{6.39}$$

RGST und Adaptive-Recursive Proteciton In [2] wird außerdem eine Erweiterung durch Kombination mit dem in Abschnitt 6.2.5 beschriebenen *Adaptive-Recursive* Verfahren vorgeschlagen. Der sogenannte $ARGST$-Algorithmus wird analog nach folgender Formel gebildet:

$$\begin{aligned} F_i(\vec{x}, n) \quad = \quad & (1 - p(\vec{x}, n)) \cdot \\ & \left(\sum_k F(\vec{x} - (2k+1)\vec{u}_y, n) h_1(k, \delta_y) + \sum_m F_{out}(\vec{x} - \begin{pmatrix} d_x(\vec{x}, n) \\ \rho_y \end{pmatrix} - 2m\vec{u}_y, n-1) h_2(m, \rho_y) \right) \\ & + p(\vec{x}, n) F_{init}(\vec{x}, n) \end{aligned} \tag{6.40}$$

6.3 Qualität der Algorithmen

Bellers et al. [2] führt eine komplexe Analyse der einzelnen Verfahren durch. Diese zu wiederholen soll nicht Teil dieser Arbeit sein. Aus diesem Grund wird an dieser Stelle auf [2] verwiesen und das Ergebnis übernommen, wonach der *Adaptive-Recursive* und der $ARGST$ Algorithmus die besten Ergebnisse liefern. Als Gründe für diese hervorragenden Ergebnisse werden die Rekursivität, die zu einer Stabilität über die Zeit führt und die gute Fehlerkorrektur genannt.

7 Temporal Upscaling

Temporal Upscaling (oder auch *Scan Rate Conversion* oder *Temporal Super-Resolution*) wird im Zusammenhang mit dem *Broadcasting* digitaler Bildsignale häufig verwendet. Insbesondere zur Vermeidung von Flimmern werden in vielen TV-Geräten die Signale intern auf Bildwiederholraten größer $100Hz$ interpoliert [5]. Wie in Kapitel 3 beschrieben, ist auch im UHD-Umfeld eine deutlich höhere *Frame Rate* notwendig.
Dieses Kapitel beschäftigt sich mit Methoden des *Temporal Upscaling*. Diese lassen sich, ähnlich den *Deinterlacing*-Verfahren in Kapitel 6, in zwei Gruppen unterteilen. Auf die *linearen Verfahren* wird an dieser Stelle nicht detailliert eingegangen, da diese deutliche Qualitätsmängel aufweisen. Die *Motion-Compensated* Methoden basieren ebenfalls auf der Auswertung von Bewegungsvektoren und liefern deutlich bessere Ergebnisse [5].

7.1 Lineare Methoden

Die linearen Methoden werden durch die Tatsache charakterisiert, dass sie jedes Pixel auf exakt gleiche Art und Weise behandeln.

7.1.1 First-Order

Der *First-Order* oder *Frame Insertion* Filter liest die Bilder des Originalsignals in einen Speicher ein. Um auf die gewünschte Zielframerate zu kommen, werden einzelne Bilder mehrmals hintereinander ausgelesen. Artefakte und Auswirkungen auf die Wahrnehmung werden in [5] beschrieben.

7.1.2 Higher-Order

Im Gegensatz zum *First-Order* Verfahren bedient sich das *Higher-Order* Verfahren (oder auch *Frame Average*) einer (bi-)linearen Interpolation. Anstatt lediglich ein *Frame* zu wiederholen, wird ein Mittel zwischen dem aktuellen und dem zeitlich folgenden *Frame* gebildet [5]. $\vec{x} = (x, y)$ ist die Pixelposition, n das *Frame* und α ist die zeitliche Distanz zwischen dem vorhandenen *Frame* und den zu Interpolierenden.

$$F_{avg}(\vec{x}, n + \alpha) = (1 - \alpha)F(\vec{x}, n) + \alpha F(\vec{x}, n + 1) \ , \qquad 0 \leq \alpha \leq 1 \tag{7.1}$$

7.2 Motion-Compensated Methoden

Um die Qualität des *Temporal Upscaling* zu verbessern, ist es notwendig mit Bewegungsvektoren zu arbeiten. Die Qualität der Konversion ist auf diesem Gebiet in hohem Maße von der Genauigkeit der Vektoren abhängig. Außerdem müssen die Vektoren die sogenannte *True Motion* (siehe Kapitel 5.5) repräsentieren, da ansonsten Bewegungen ruckeln oder sonstige Interpolationsfehler auftreten können [18].

7.2.1 Motion-Compensated First Order

Analog zu den *Deinterlacing*-Verfahren können auch die linaren *Upscaling* Algorithmen auf einfache Weise die *Motion-Compensation* mit einbeziehen. Der *First-Order* Algorithmus ergibt sich zu:

$$F_{MC}(\vec{x} + \alpha\vec{d}(\vec{x}, n), n + \alpha) = F(\vec{x}, n), \qquad 0 \leq \alpha \leq 1 \tag{7.2}$$

7.2.2 Motion-Compensated Higher Order

Eine mögliche *Higher-Order* Methode ist das *Motion-Compensated Average*, bei der zwei aufeinander folgende Bilder (bi-)linear miteinander verknüpft werden, wobei jedes Pixel entsprechend der Bewegungsvektoren in die Richtung des Pixels des anderen Bildes verschoben wird. Die Notation ist ähnlich der aus Abschnitt 7.1.2. Es wird das Zwischenbild $F_{mca}(\vec{x}, n + \alpha : 0 \leq \alpha \leq 1)$ zwischen den Bildern $F(\vec{x}, n)$ und $F(\vec{x}, n + 1)$ gebildet. Dabei repräsentiert $\vec{d}(\vec{x}, n)$ die Menge der Bewegungsvektoren zwischen den zwei Bildern.

$$F_{mca}(\vec{x}, n + \alpha) = (1 - \alpha)F(\vec{x} + (1 - \alpha)\vec{d}, n) + \alpha F(\vec{x} - \alpha\vec{d}, n + 1) \ , \qquad 0 \leq \alpha \leq 1 \tag{7.3}$$

Den Unterschied zwischen Formel 7.1 und 7.3 beschreibt Ojo et al. [18] als „signifikant“. Insbesondere sich bewegende Bereiche eines Bildes werden durch den Einsatz der Bewegungsvektoren deutlich schärfer wiedergegeben. Allerdings ergeben sich in statischen Bildteilen, wie zum Beispiel Untertiteln, Artefakte, die bei den Verfahren ohne *Motion-Compensation* nicht entstehen. Verursacht werden diese Artefakte durch fehlerhafte Bewegungsvektoren.
Die Verbesserung ist global, die der Einsatz von Bewegunsvektoren gebracht hat. Die Artefakte, die mit diesem Verfahren noch auftreten, besitzen einen lokalen Charakter. Aus diesem Grund schlägt Ojo et al. [18] vor, diese lokalen Probleme gesondert zu behandeln, basierend auf der Zuverlässigkeit der Bewegungsvektoren. Das Prinzip der *graceful Degradation* (zierliche Zerlegung) wird im Folgenden behandelt.

7.2.3 Graceful Degradation

MC Static Median Filtering

Falsche Bewegungsvektoren sind insbesondere bei statischem Text oder anderen feinen statischen Strukturen auffällig. Die *Higher-Order* Algorithmen kombinieren verschiedene Vorhersagen miteinander. Dadurch können sie die Auswirkungen falscher Vektoren mindern, sie aber niemals ganz eliminieren. Die Bildung eines Medians soll dies ermöglichen. Formel 7.4 beschreibt das Vorgehen, wobei F_{mca} die Formel 7.3 beschreibt.

$$F_{sta}(\vec{x}, n+\alpha) = MED(F(\vec{x}, n), F(\vec{x}, n+1), F_{mca}(\vec{x}, n+\alpha)) \tag{7.4}$$

Testergebnisse in [5] zeigen eine deutliche Verbesserung im Vergleich zum ursprünglichen Algorithmus. Dennoch hat auch das *MC Static Median Filtering* Probleme. Insbesondere detailreiche Strukturen werden durch das Vorgehen unscharf [18].

MC Dynamic Median Filtering

Vijay et al. [5] zeigt, dass die optimale Mischung zwischen dem *Motion-Compensated Median Filtering* und dem *Motion-Compensated Average* der *Motion-Compensated Dynamic Median* Filter ist. Dieser ist eine Abänderung des statischen Filters. Anstatt mit F_{mca} aus Formel 7.3 zu arbeiten, benutzt er das *non Motion-Compensated* F_{avg} aus Formel 7.1. Die anderen zwei Bestandteile des Filters bestehen aus *Motion-Compensated* Elementen.

$$F_{dyn}(\vec{x}, n+\alpha) = MED(F(\vec{x}-\alpha\vec{d}(\vec{x}, n+\alpha), n), F(\vec{x}+(1-\alpha)\vec{d}(\vec{x}, n+\alpha), n+1), F_{avg}(\vec{x}, n+\alpha)) \tag{7.5}$$

Jedoch gibt es auch Situationen in denen der dynamische Median Filter nicht optimal funktioniert. Das Problem ist die begrenzte Anzahl an Auswahlmöglichkeiten. Sind die Bewegungsvektoren falsch, so wird auf jeden Fall *Frame Average* genommen. Für den Fall jedoch, dass sich zwei ungleiche Bewegungen überlagern, können sowohl die Bewegungsvektoren, als auch das *Frame Average* die falsche Wahl sein.

Reciprocal Mixing

Um eine weitere Alternative zu schaffen, beschreibt Vijay et al. [5] das *Reciprocal Mixing*. Bei diesem Verfahren wird eine Mischung zwischen dem *MC Frame Average* und dem *Frame Average* gebildet, wie in Formel 7.6 beschrieben.

$$F_{mix}(\vec{x}, n+\alpha) = (1-k)F_{avg} + kF_{mca}, \qquad 0 \leq k \leq 1 \tag{7.6}$$

Der Faktor k in der Formel beschreibt die Zuverlässigkeit der Bewegunsvektoren. Nimmt k große Werte an, so sind die Vektoren zu einem großen Prozentsatz richtig und anders herum. Den Faktor k kann man beispielsweise analog zu [5] definieren. Zunächst werden Hilfsvariablen gesetzt:

$$\begin{aligned} A &= F(\vec{x}, n) \\ B &= F(\vec{x}, n+1) \\ C &= F(\vec{x} - \alpha\vec{d}(\vec{x}, n+\alpha), n) \\ D &= F(\vec{x} + \alpha\vec{d}(\vec{x}, n+\alpha), n) \end{aligned}$$

mit denen k folgendermaßen definiert wird:

$$k = \frac{|B - C|}{\beta|B * C - A - D| + \delta} \tag{7.7}$$

In dieser Formel ist δ lediglich eine sehr kleine Zahl um die Division durch Null zu verhindern und β ein Verstärkungsfaktor.

Cascaded Median Filter

Mit dem *Cascaded Median Filtering* wird versucht die Vorteile des *MC Static Median*, des *MC Dynamic Median* und des *Cascaded Median Filteris* zu kombinieren. Zu diesem Zweck wird ein weiterer Median Filter angewendet.

$$F_{out} = MED(F_{dyn}, F_{sta}, F_{mix}) \tag{7.8}$$

In [18] wird jedoch erwähnt, dass auch dieser Filter in den Regionen zum *Blur* neigt, in denen die Bewegungsvektoren nicht zuverlässig sind.

Ojo et al. [18] zeigt, dass nach einer objektiven Messung der *Reciprocal Mixing* Filter das beste Ergebnis hat. Dieses Ergebnis wird jedoch in Frage gestellt und die Theorie entwickelt, dass die guten Werte rein global gesehen sind. Der These nach ergibt ein kleiner, kaum wahrnehmbarer, globaler Fehler gepaart mit einem kleinen lokalen Fehler eine bessere visuelle Qualität, als ein perfekter globaler Wert mit großen lokalen Fehlern. Im Allgemeinen zeigt sich jedoch in der Analyse, dass jeder Filter in den Bereichen, für die er designt wurde (statische Details, Bewegung), die besten Ergebnisse liefert.
Generell hängt auch die Qualität der *Upconversion* von der Genauigkeit und der Zuverlässigkeit der Bewegungsvektoren ab.

8 Theoretisches Upscaling

Es ist nicht möglich HD Material durch einfache *Frame*- und Pixeldoppelung an die Ultra-HD Gegebenheiten anzupassen. Dies würde zu einer schlechteren Seherfahrung führen als bei einem herkömmlichen HD-Fernsehsystem. Die Gründe für dieses zunächst paradox erscheinende Problem werden in diesem Abschnitt behandelt.
In Kapitel 3 wurden bereits einige Grundüberlegungen bezüglich Ultra-HD im Bezug auf die *Frame Rate* angesprochen. Diese werden im aktuellen Kapitel weitergeführt und auf das konkrete Problem der Konvertierung von herkömmlichem HD-Material in den Ultra-HD Standard angewendet.

In den folgenden Abschnitten wird entwickelt, welche Parameter beim *Upscaling* einzuhalten und zu beachten sind. Das Kapitel dient als Grundlage für die in Kapitel 10 umgesetzten Programmelemente.
Bei den Überlegungen wird stets ein Betrachtungsabstand von $2,7m$, der ideale Betrachtungsabstand, vorausgesetzt. Außerdem werden die zu diesem Betrachtungsabstand idealen Bildschirmdiagonalen angenommen. Bei HDTV hat diese eine Länge von $68inch$, bei UHD $135inch$ (die Berechnung dieser Werte kann in dem Artikel [11] nachgelesen werden). Außerdem werden die Überlegungen für ein Ausgangsmaterial mit 25 Bildern pro Sekunde im *Interlaced*-Verfahren und ein Upscaling auf 120 Bilder pro Sekunde angestellt.

8.1 Judder

Nimmt man ein Standbild einer Szene auf, so wird ein Gegenstand auf einer bestimmten Anzahl a von Pixeln aufgenommen. Wird bei einer Videosequenz ein Kameraschwenk durchgeführt, oder bewegt sich der Gegenstand, so wird er auf der Anzahl $a + x$ von Pixeln abgebildet. x ist die Anzahl der Pixel, die der Gegenstand durch die vorhandene Bewegung innerhalb der Belichtungszeit wandern kann. Der Gegenstand wird unscharf. Die theoretisch maximale Auflösung wird nicht mehr erreicht.
Aus diesem Grund ist es möglich die Belichtungszeit der einzelnen *Frames* zu verringern. In diesem Fall wird der Sensor nur einen Bruchteil der durch die *Frame Rate* vorgegebenen maximalen Belichtungszeit belichtet. Den Rest der Zeit wird er verdunkelt. Somit wird dem Verschmieren der Gegenstände vorgebeugt.
Jedoch bringt auch diese Art der Aufnahme Probleme mit sich. Abgesehen davon, dass der Sensor lichtempfindlicher sein muss um bei kürzerer Belichtungszeit auch die selbe Helligkeit zu liefern, tritt für zu kurze Belichtungszeiten der sogenannte *Judder*-Effekt auf. Bei der Wiedergabe nimmt das menschliche Auge keine flüssige Bewegung mehr war, sondern der Gegenstand ruckelt und springt von einer Position zur nächsten.
Das menschliche Auge erwartet eine gewisse Bewegungsunschärfe bei bewegten Objekten. Je schärfer ein Objekt ist, desto weniger weit darf es vom einem zum nächsten Bild

wandern, ohne dass ein *Judder*-Effekt wahrgenommen wird. Ein scharfes Objekt kann daher zwischen zwei Bildern nur kleine Sprünge machen, ein durch Bewegung unscharfes Objekt größere.
An dieser Stelle lässt sich auch erklären, warum es nicht möglich ist, HD-*Frames* bei der Wiedergabe mit $120Hz$ mehrmals hintereinander zu zeigen bis die *Frame Rate* erreicht ist. Durch dieses Verfahren tritt der oben beschriebenen *Judder*-Effekt auf. Das Bild ruckelt, da der bewegte Gegenstand zu lange auf einer Position bleibt, bevor er zur nächsten springt.
Grund für diesen Effekt ist insbesondere die größere Bildschirmdiagonale. Der Sprung, den der Gegenstand auf einem UHD-Display macht ist wesentlich größer als auf einem HD-Display. Auf einem UHD Display mit der selben Diagonale, wie bei einem HD Display, tritt der Effekt nicht auf.

Prinzipiell gibt es zwei Ansätze um dieses Problem zu lösen. Der erste Ansatz, eine einfache *Frame*-Dopplung und eine Bewegungsunschärfe für die sich bewegenden Objekte zu berechnen, hat lediglich Nachteile. Das Material würde maximal mit der selben Auflösung wahrgenommen wie auf einem herkömmlichen HD Display. Der zweite Ansatz interpoliert die Zwischenbilder. Durch eine *Motion Estimation* wird die Bewegung der einzelnen Gegenstände nachempfunden und in die Berechnung der Zwischenbilder mit einbezogen. Es wird versucht, nachträglich eine Sequenz zu berechnen, die möglichst so ist, als wäre sie direkt mit einer höheren *Frame Rate* aufgezeichnet worden.
Konkret muss bei der Umsetzung des Versuchs in MATLAB (siehe Abschnitt 10) also ein *Upscaling* mit *Motion Estimation* durchgeführt werden.

8.2 Deinterlacing

Interlacing ist ein Prinzip, das aus der Not heraus eingeführt wurde, bei begrenzter Bandbreite ein flimmerfreies Signal zu kreieren (siehe Kapitel 2 und4). Um diese Ziele zu erreichen hat man andere Artefakte, wie beispielsweise das Zeilenflackern, in Kauf genommen. Da heutzutage jedoch höhere Bandbreiten realisiert werden können als zu Beginn der Fernsehübertragung und Displays ausschließlich progressive arbeiten, wird zunehmend Abstand vom *Interlacing* genommen. In der Definition des Ultra-HD Standards (siehe Kapitel 3) ist kein *Interlaced-Mode* mehr vorgesehen.
Aus oben genannten Gründen muss während der Konvertierung für *interlaced* Material ein *Deinterlacing* durchgeführt werden. Abschnitt 6 gibt eine Übersicht über vorhandene Verfahren und erläutert warum die *Motion Compensating* Verfahren deutliche Qualitätsvorteile haben. Da nach dem *Deinterlacing* auch die *Upconversion* durchgeführt werden muss, ist es sinnvoll ein Verfahren, das auf gute Qualität ausgelegt ist, zu benutzen.

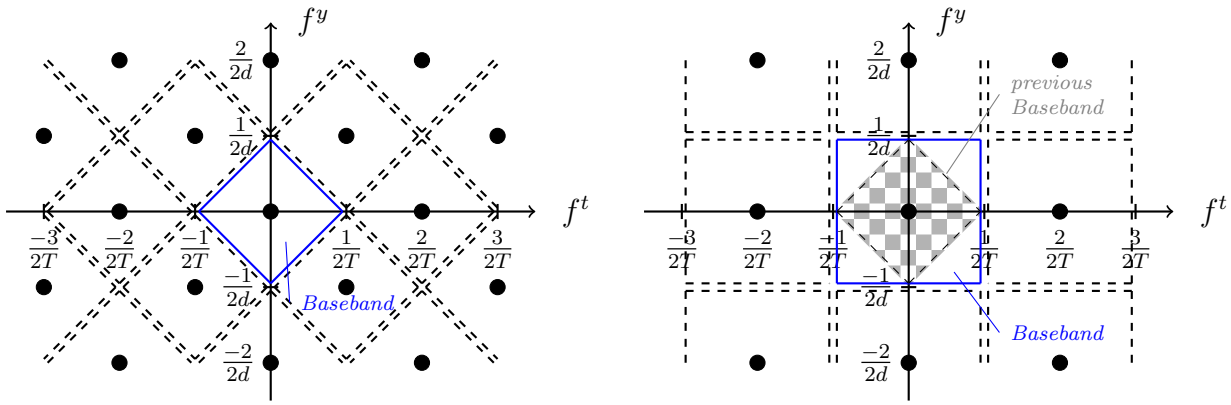


Abbildung 8.1
Vergleich des *Baseband* vor (links) und nach (rechts) dem *Deinterlacing*.

8.3 Frequenzen

Im Kapitel 4 wurde bereits beschrieben, wie das *Baseband* durch *Interlacing* verändert wurde. Für die weitere Vorgehensweise ist es von Bedeutung abzuschätzen, wie die angedachten Bearbeitungen das *Baseband* weiter beeinträchtigen. Da die hohen Frequenzen, also die feineren Bilddetails für die wahrnehmbare Bildqualität von entscheidender Bedeutung sind, darf die Bearbeitung das ursprüngliche *Baseband* an keiner Stelle weiter beschneiden.

8.3.1 Deinterlacing

Bevor ein *Upscaling* durchgeführt werden kann muss zunächst das *Deinterlacing* erfolgen. Abbildung 8.1 zieht einen Vergleich zwischen dem *Baseband* vor und nach dem *Deinterlacing*. Man sieht deutlich, dass das *Baseband* nach dem *Deinterlacing* größer geworden ist und das ursprüngliche *Baseband* komplett in das neue hineinpasst. Allerdings ist dies nur der Fall, wenn nach dem *Deinterlacing* ein $50Hz$ Signal vorliegt.
Die Bereiche, die zudem durch die höhere Auflösung des Signals freigeworden sind, werden zunächst nicht in Anspruch genommen. Die Bildinformationen, die in diesen Bereichen noch vor dem *Interlacing* waren, sind verloren gegangen. Für die weitere Bearbeitung ist wichtig, dass das Signal bei diesem Schritt nicht noch weiter eingeschränkt wird.

8.3.2 Upscaling

Für das zeitliche *Upscaling* erwartet man ein ähnliches Ergebnis wie für das *Deinterlacing*. Tatsächlich gibt Abbildung 8.2 die Verhältnisse im selben Maßstab wie in Abbildung 8.1 wieder. Man sieht, dass das neue *Baseband* wesentlich größer ist, als noch vor dem *Upscaling*. Das alte *Baseband* passt komplett in dieses neue *Baseband* hinein. Es ergeben sich demnach keinerlei neuen Probleme durch das *Upscaling*. Die rechteckige Form ergibt sich, da bei dem in dieser Arbeit durchgespielten Versuche keinerlei räumliches

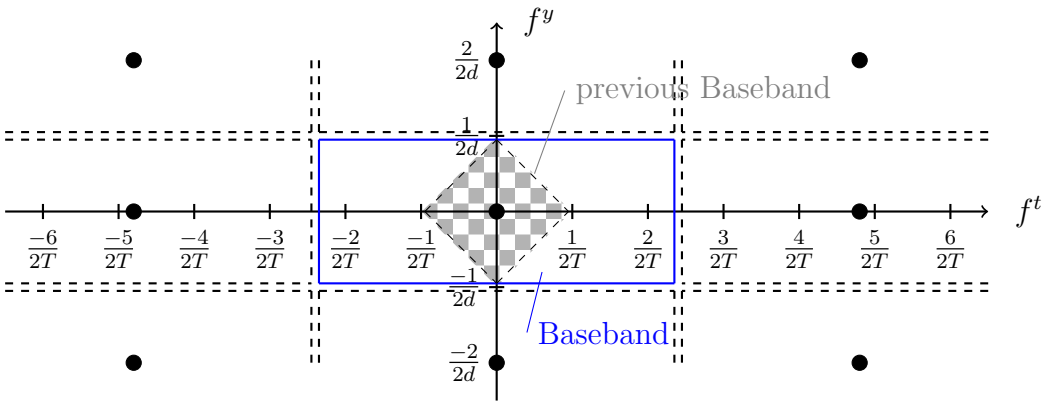


Abbildung 8.2
Baseband nach dem *Upscaling*

Upscaling durchgeführt wird. Demnach bleiben die Verhältnisse in der $x - y$ Ebene unverändert. Für die anderen Zielframeraten ($60Hz$ und $100Hz$) sind die Ergebnisse analog.

8.4 Alias

Während bei anderen Gebieten der digitalen Signalverarbeitung die Vermeidung von Aliasstörungen eine sehr hohe Priorität hat, wird die Verletzung des Abtasttheorems bei der Videoaufzeichnung regelmäßig in Kauf genommen. Zu häufig gibt es Elemente im Bild, die nicht so kontrollierbar sind, dass sie innerhalb dieser Grenzen bleiben (Postkutschen-Effekt). Der Grund für temporale Alias-Effekte wurde in Abschnitt 4.1 erklärt. Beim *Deinterlacing* bzw. beim *Upscaling* entstehen nun freie Frequenzbereiche im Basisband, die die Aliasgrenzen zu höheren Frequenzen verschieben. Bewegungen, die innerhalb der Parameter eines $25i$ Signals zu Alias führen können in einem $120p$ Signal problemlos aliasfrei wiedergegeben werden. Es ist jedoch nicht möglich Aliasstörungen wieder herauszurechnen, die bereits im Signal sind. Demnach wird das Videosingal auch nach dem *Deinterlacing* und der Konvertierung auf $120Hz$ die $25i$ Aliasfehler beinhalten. Die freien Frequenzen innerhalb des neu berechneten Signals werden weiterhin frei bleiben.

9 Hard- und Software

In diesem Kapitel wird die in dem Versuch verwendete Hard- und Software dargestellt. Zudem werden Einstellungen erläutert, die für die Rekonstruktion der Bedingungen notwendig sind, sowie die Ausführung des Media Converters erläutert.

9.1 MATLAB

Das in dieser Arbeit entwickelte Programm wird in *MATLAB* geschrieben. Es liegt folgendes System zu Grunde:

MATLAB R2011b
Windows 7 Home Premium 64-bit
Intel(R) Core(TM) i5-3317U CPU @ 1.70 GHz
8GB RAM
Intel(R) HD Graphics 4000

9.2 Toolboxes

Neben den Werkzeugen, die in der Basisversion von MATLAB vorhanden sind, werden zudem die *Image Processing Toolbox* und die *Signal Processing Toolbox* benutzt.

9.2.1 Image Processing Toolbox

Die *Image Processing Toolbox* bietet eine Reihe an Basisfunktionen bei der Verarbeitung von Bildern an. Für das entwickelte Programm sind insbesondere Filter- und Einlesefunktionen von großer Bedeutung. Alle gängigen Bilddateiformate werden unterstützt [17].

9.2.2 Signal Processing Toolbox

Die *Signal Processing Toolbox* stellt dem Entwickler sämtliche Werkzeuge zur Verfügung, die bei der Bearbeitung von analogen und digitalen Signalen benötigt werden. Für diese Arbeit sind beispielsweise die Transformation vom Ortsbereich in den Frequenzbereich und zurück essentiell [17].

9.3 MATLAB Coder

Der *MATLAB Coder* generiert automatisch aus MATLAB Code C- oder C++-Code. Programme können somit aus MATLAB exportiert werden und MATLAB-unabhängig weiterverwendet werden. Darüber hinaus bietet der *Coder* die Möglichkeit rechenintensive Programmteile in das sogenannte *MEX*-Format (*MATLAB Exchange*) zu übersetzen.

MEX definiert eine Schnittstelle zwischen MATLAB und C-/C++- Code, sodass innerhalb eines MATLAB-Programms Klassen beider Sprachen verwendet werden können.

Innerhalb des in dieser Arbeit entwickelten Programms werden häufig verschachtelte Schleifen benutzt. Da MATLAB diese Art des Programmierens deutlich langsamer als ein C-/C++-Programm ausführt, werden diese Programmteile über den *MATLAB Coder* in C-Code übersetzt und über *MEX* mit eingebunden[17].
Da C-/C++- Code systemabhängig unterschiedliche Dateien benötigt, müssen die *MEX*-Dateien für jedes System einmal neu erstellt werden. Die Datei *Media_Converter.prj* im Programm definiert sämtliche Austauschvariablen. Um die *MEX* Funktionen nutzen zu können müssen sie über den Button *Build* im *Media Coder* compiliert werden (siehe Abbildung 9.1).

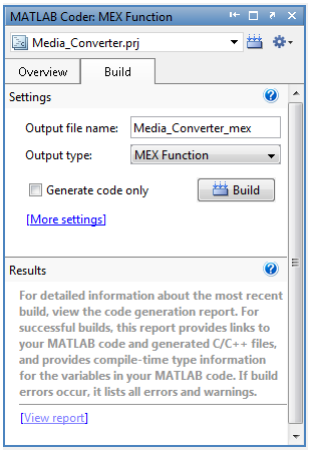


Abbildung 9.1
MATLAB Coder

Wichtig ist, dass im Feld *Output file name:* die Zieldatei ***Media_Converter_mex*** eingetragen wird. Außerdem muss der *MEX*-Code im MATLAB-Path des Projektes liegen.

9.3.1 Compiler

Für die Kompilierung des *MEX*-Codes wurde der C-Kompiler des *Microsoft-Software Development Kit (SDK) 7.1* benutzt. Die Kompiler lassen sich in MATLAB über den Befehl `mex -setup` konfigurieren. Außerdem muss vor der Kompilierung des *MEX*-Codes unter *More Settings* (siehe Abbildung 9.1) im Reiter *General* als *Language* die Zielsprache C gewählt werden.
Hinweis: Unter Umständen kommt es bei der Installation des *Microsoft-Software Development Kit (SDK)* zu einem Konflikt mit bereits installierten *Visual C++* Komponenten. Für diesen Fall findet sich unter [3] ein Lösung.

a) *Crowd Run*
b) *Ducks Take Off*
c) *In To Tree*
d) *Old Town Cross*
e) *Park Joy*

Abbildung 9.2
SVT High Definition Multi Format Test Set [7]

9.4 Videovorlagen

Für die Beurteilung des entwickelten Programms wird das von der EBU (European Broadcasting Union) empfohlenen und von SVT (Sveriges Television AB - öffentlich-rechtliche Fernsehgesellschaft Schwedens) bereitgestellte *SVT High Definition Multi Format Test Set* benutzt, sowie selbst entwickelte Testsequenzen benutzt.
Das *SVT High Definition Multi Format Test Set* besteht aus fünf verschiedenen Testsequenzen, die in verschiedenen Standards vorhanden sind [7]. Interessant für das entwickelte Programm sind die 1080i25- und die 1080p50-Versionen. Bei den *interlaced* Sequenzen werden beide Halbbilder in einem Vollbild transportiert. In diesem Fall gehören die ungeraden Zeilen um Bild n, die geraden Zeilen zum Bild $n+1$ (*Top-Field-First*). Informationen zu den einzelnen Testsequenzen und den Konversionen in die verschiedene Formate finden sich in [24]. Die fünf Testsequenzen werden in Abbildung 9.2 dargestellt.

Darüber hinaus wurden weitere Testsequenzen erstellt. Da diese unnatürliche Objekte und Hintergründe beinhalten werden diese innerhalb dieser Arbeit als *Artificial Test Sequences (ATS)* bezeichnet. Sie dienen zum initialen Testen des Programms. Außerdem wurde versucht Grenzfälle des Verfahrens zu rekonstruieren. In allen Fällen wandert ein runder Ball mit einer konstanten Geschwindigkeit über eine Oberfläche. Die verschiedenen Versionen sind in Abbildung 9.3 aufgeführt. Diese Testsequenzen liegen in SD und HD vor. Die Auswirkungen verschiedener Einstellung innerhalb des Programms lassen sich in SD ebenso gut beurteilen wie in HD, die Rechenzeit ist jedoch deutlich geringer.
Hinweis: Bei allen *interlaced* Testsequenzen wird von einer *Top-Field-First* Reihenfolge ausgegangen.

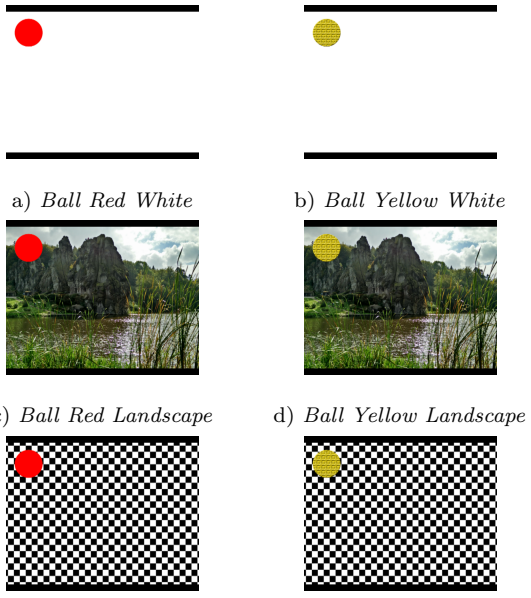


Abbildung 9.3
Ball-Testsequenzen -
a) niederfrequentes Objekt vor niederfrequentem Hintergrund
b) hochfrequentes Objekt vor niederfrequentem Hintergrund
c) niederfrequentes Objekt vor hochfrequentem Hintergrund
d) hochfrequentes Objekt vor hochfrequentem Hintergrund
e) niederfrequentes Objekt vor Schachbrett
f) hochfrequentes Objekt vor Schachbrett

9.5 Ausführen des Media Converters

Der Media Converter lässt sich in MATLAB ausführen, indem das Verzeichnis in dem alle Klassen (und unter Umständen die MEX-Dateien) als *Current Folder* gewählt ist (also zum *Path* gehört) und der Befehl `Media_Converter` in das *Command Window* eingetippt wird (siehe Abbildung 9.4). Vorher müssen alle gewünschten Einstellungen in der `config.m` getroffen werden (Kapitel 10).

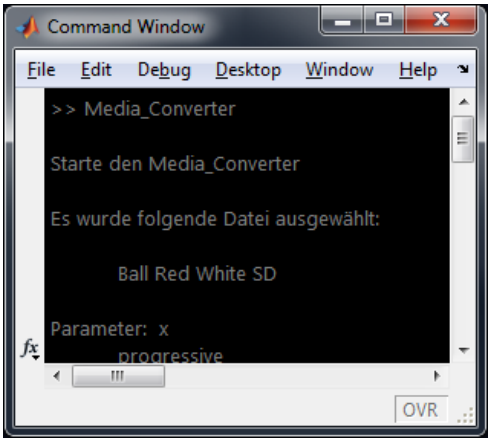


Abbildung 9.4
Starten des *Media Converter*

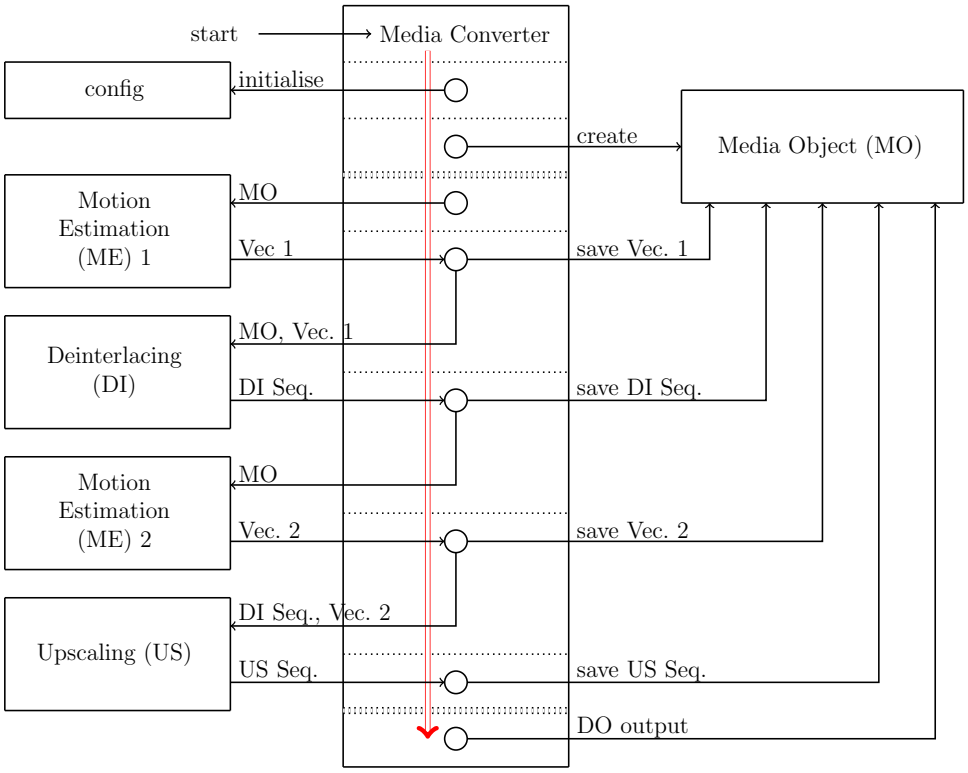


Abbildung 10.2
Programmstruktur

10 Media Converter

Die in dieser Arbeit beschriebene Konvertierung von Videosequenzen wird in MATLAB umgesetzt. Es wird erläutert, auf welche Art und Weise die verschiedenen Programmelemente arbeiten. Informationen zu den technischen Rahmenbedingungen und den Testsequenzen finden sich in Abschnitt 9.

Abbildung 10.1 zeigt alle Klassen, Funktionen und sonstige Projektdatein des *Media Converters*. Die Aufteilung hat teils den Hintergrund der Modularisierung und Aufgabenteilung, teils der schnelleren Verarbeitung durch die Verwendung von MEX (siehe 9.3). Im Folgenden werden die zentralen Aufgaben innerhalb der Programmstruktur erläutert, die die Arbeitsweise deutlicher werden lässt. Für manche Aufgaben werden mehrere Funktionen aus Abbildung 10.1 verwendet.

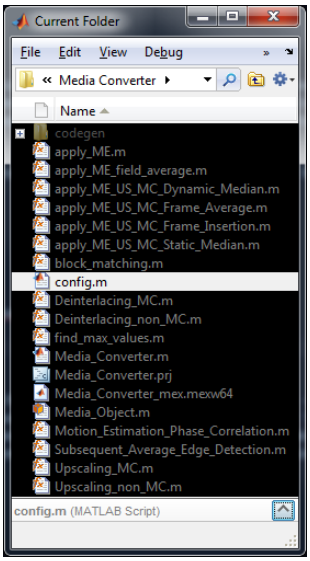


Abbildung 10.1
Alle Datein des *Media Converter*

10.1 Programmstruktur

Abbildung 10.2 gibt eine Übersicht über das Programm *Media Converter*. Jedes Rechteck beinhaltet einen Programmteil, wobei die Klasse *Media Converter* die Kontrolle über die restlichen Klassen besitzt und diese der Reihe nach aufruft.

10.1.1 Media Converter

Soll das Programm gestartet werden, so muss die Klasse *Media Converter* aufgerufen werden. Sie kontrolliert den Ablauf des gesamten Programms. Zunächst werden die Konfigurationsparameter geladen bevor ein *Media Object* erzeugt wird. Dieses repräsentiert im Laufe der Bearbeitung die tatsächlichen Videosequenzen. Die Klasse *Media Converter* prüft vor jedem Schritt, ob eventuell bei einer vorherigen Ausführung der als nächstes angedachte Schritt schon einmal ausgeführt wurde und ob von dieser Bearbeitung noch Informationen vorliegen. In dem Fall, dass es bereits Informationen (beispielsweise Bewegungsvektoren) gibt, lädt sie diese Informationen und überspringt den Bearbeitungsschritt. Um die durchgeführten Bearbeitungsschritte nachvollziehen zu können, werden die Informationen als Dateien in einem extra für die Bearbeitung angelegten Ordner `output_directory/Temp/` gespeichert. Dieser wird zu Beginn des Programms im definierten *Output*-Verzeichnis angelegt. Jede dort abgelegte Datei erhält einen Dateinamen, der durch eine *ID* definiert wird. Diese setzt sich aus den in der *config* definierten Parametern zusammen und gibt eindeutig die gewählte Programmkonfiguration wieder.
Nach dem Start des *Media Converters* wird in jedem Fall die Konfiguration vorgenommen und das *Media Object* erzeugt. Der weitere Verlauf hängt von den in der `config.m` gesetzten Parametern und den bereits vorhandenen Ergebnissen ab. Beispielsweise ist es nicht notwendig ein *Deinterlacing* und eine *Motion Estimation 1* durchzuführen, wenn das Eingangssignal *progressive* ist. Wenn in der *config* eine nicht bewegungskompensierende *Deinterlacing*-Methode ausgewählt wurde, muss ebenfalls keine *Motion Estimation 1* durchgeführt werden. Auf der anderen Seite ist kein *Upscaling* notwendig, wenn die *Frame Rate* der Ausgangssequenz gleich derjenigen der Eingangssequenz sein soll.
Dementsprechend ist der Ablauf des *Media Converters* abhängig von den Konfigurationsdaten. Jedoch wird für jeden durchgeführten Schritt (selbst wenn die Informationen zuvor bereits errechnet wurden und nur noch geladen werden müssen) das Ergebnis im *Media Object* gespeichert. Durch dieses Verfahren liegen bei der Ausgabe auch sämtliche Zwischeninformationen vor. In jedem Fall wird zum Schluss das *Media Object* vom *Media Converter* dazu aufgefordert, seine Informationen auszugeben. Dazu gehört zum einen das Speichern der erzeugten Sequenz (in verschiedenen Formen, siehe Abschnitt 10.1.2), sowie die Ausgabe eines *Previews*, das Zwischenergebnisse darstellen kann.

10.1.2 Konfiguration

In der `config.m` werden alle Elemente definiert, die für die Ausführung des Programms *Media Converter* von Bedeutung sind. Konkret handelt es sich um folgende Elemente:

- Globale Variablen
- Pfade
- Input Sequenz
- Processing Settings
- Preview Settings
- Motion Estimation
- Deinterlacing
- Upscaling
- Output Settings
- IDs

In diesem Abschnitt werden lediglich die Elemente angesprochen, die für den Ablauf des Programms zuständig sind (z.B. *Processing Settings*). Sämtliche Parameter, die die Berechnungen der einzelnen Bearbeitungsschritte beeinflussen, werden an den jeweiligen Stellen bei der Beschreibung der Klassen im weiteren Verlauf beschrieben.
Das Programm wird vom Nutzer über die Bearbeitung der `config.m` gesteuert. Dennoch gibt es in dieser Datei Elemente, die bei einer normalen Nutzung nicht bearbeitet werden sollten. Zur Verdeutlichung, an welchen Stellen eine Bearbeitung der Werte sinnvoll ist, sind diese über *start edit* und *end edit* Zeilen gekennzeichnet.

```
1 %config.m
2 %////////////////////////////start edit////////////////////////////
3
4 %               !!!Hier eigene Parameter waehlen!!!
5
6 %////////////////////////////end edit////////////////////////////
```

Globale Variablen

In der `config.m` werden viele Parameter definiert, auf die in den anderen Klassen zugegriffen wird. Aus diesem Grund werden diese als `global` definiert.

Pfade

Im Abschnitt Pfade werden die Ordner definiert in denen sich die Videodateien befinden. Über die Variable `choosepath` kann ein vordefinierter Pfad gewählt werden. Des weiteren können zusätzliche Pfade hinzugefügt werden:

```
1 %config.m
2 case x
3     input\_directory\_SVT = '';
4     input\_directory\_ATS = '';
5     output\_directory = '';
```

Die Variable `choosepath` kann im Folgenden auf den Wert von x gesetzt werden. Damit sich ein Dialogfenster beim Start des Programms öffnet, muss `choosepath=0` gesetzt werden.
Bei den SVT-Sequenzen ist zu beachten, dass lediglich der Oberordner für die Testsequenzen ausgewählt wird. Jede Testsequenz muss als Bitmap-Datei in dem vorgegebenen Ordner liegen (beispielsweise `1_CrowdRun_1080i25_CgrLevels_SINC_FILTER_SVTdec05_`). Die einzelnen Sequenzen werden im Folgenden in der `config.m` ausgewählt und zugewiesen.

Input Sequenz

Unter diesem Abschnitt sind sämtliche Sequenzen aus den Test-Paketen aufgelistet. Um eine dieser Sequenzen als *Input Sequzenz* auszuwählen, muss sie aktiviert werden. Dadurch wird der Name, der Modus, der Pfad, die Eingangs-*Frame Rate* und der Typ der Sequenz gesetzt.

```
1 %config.m
2 test_sequence = 'Ball Red Checkerboard';    input_mode = 'interlaced';
3     input_path = path_BRC_1080i25;  input_framerate = 25;
4     input_type = 'image_sequence';
```

Generell kann der Modus *interlaced* oder *progressive* sein, die *Frame Rate* $25Hz$ oder $50Hz$ und der Typ der Sequenz *image_sequenz* oder *video* je nachdem ob die Bildfolge als einzelne Bitmap-Bildatein oder als AVI-Video vorliegt. Sollten andere Sequenzen mit dem Programm genutzt werden, so müssen diese analog eingetragen werden.

Processing Settings

```
1 %config.m
2 input_start_frame = 1;
3 input_processing_frames = 3;
4 use_mex = true;
5 %use_mex = false;
```

Diese Parameter steuern den Ablauf des *Media Converter*. `input_start_frame` beschreibt das erste zu bearbeitende *Frame* der Sequenz. `input_processing_frames` hingegen wie viele *Frames* bearbeitet werden sollen. Somit kann aus einer Sequenz ein beliebiger Abschnitt für die Konvertierung ausgewählt werden. Es ist zu beachten, dass der Wert `input_processing_frames` lediglich die Anzahl der *Frames* der Eingangssequenz darstellt. Durch die *Motion Estimation* (für das letzte *Frame* lässt sich keine Bewegung vorhersagen), das *Deinterlacing* und die *Upconversion* wird dieser Wert während des Programmablaufs geändert und angepasst.

Preview Settings

```
1  %config.m
2  preview = 'on';
3  %preview = 'off';
4
5  %preview_frame = 1;
6  preview_frame = input_start_frame;
7
8  preview_ME_vector_subsampling=2;
9
10 preview_xstart = 1;
11 preview_xend = 700;
12 preview_ystart = 1;
13 preview_yend = 280;
```

Während der Berechnung kann an verschiedenen Schlüsselstellen im Programm ein *Preview* ausgegeben werden. Damit kann untersucht werden, ob die Ergebnisse des *Media Converters* an diesen Stellen wie gewünscht vorliegen. Der *Preview* kann über die Variable `preview` an- oder ausgeschaltet werden. Bei den bewegungskompensierenden Verfahren ist diese Funktion hilfreich um zu erkennen, ob die Vektoren ordnungsgemäß berechnet werden konnten. Damit die Darstellung nicht zu unübersichtlich wird, können die angezeigten Vektoren über die Variable `preview_ME_vector_subsampling` begrenzt werden. Im obigen Fall wird beispielsweise nur jeder zweite Vektor angezeigt.
Außerdem kann über die Variable `preview_frame` bestimmt werden, welches *Frame* für das *Preview* genutzt werden soll. Möchte man dieses manuell setzen, so kann man ein *Frame* angeben. Es ist zu beachten, dass dieses innerhalb der bearbeiteten *Frames* liegt. Um immer das erste *Frame* einer Sequenz für das *Preview* zu nutzen, kann der Wert

des input_start_frame benutzt werden. Desweiteren kann ein Bildteil für den *Preview* ausgewählt werden. Über die Variablen preview_xstart, preview_xend, preview_ystart, preview_yend wird der Bildausschnitt gewählt. Alle Parameter müssen innerhalb der Bildgrenzen liegen. Bei den *interlaced* Sequenzen wird der *Preview* von den Halbbildern gemacht. Somit ist der Maximalwert in y-Richtung die halbe Vollbildgröße.

Output Settings

```
1  %config.m
2  output_image_sequence = 'yes';
3  %output_image_sequence = 'no';
4
5          %Bewegungsschaetzung Deinterlacing
6  output_motion_vector_sequence_DI = 'yes';
7  %output_motion_vector_sequence_DI = 'no';
8
9          %Bewegungsschaetzung Upscaling
10 output_motion_vector_sequence_US = 'yes';
11 %output_motion_vector_sequence_US = 'no';
12
13 output_motion_vector_sequence_subsampling=4;
```

Es gibt verschiedene Formen der Ausgabe. Es wird in jedem Fall eine AVI-Videodatei im *Output*-Ordner erstellt. Zudem gibt es weitere Ausgabeoptionen, die in diesem Abschnitt eingestellt werden können. Mit output_image_sequence kann augewählt werden, ob die Sequenz als Einzelbild Bitmap-Dateien ausgegeben werden soll. Die Optionen output_motion_vector_sequence_DI und output_motion_vector_sequence_US bieten die Möglichkeit, die Videosequenz als Bitmap-Folge jeweils mit Bewegungsschätzung für das *Deinterlacing* und das *Upscaling* mit eingezeichneten Vektoren auszugeben. Analog zu 10.1.2 ist auch an dieser Stelle wieder ein Subsampling der Übersicht halber sinnvoll und über output_motion_vector_sequence_subsampling einzustellen.

IDs

Die Berechnungen des *Media Converters* dauern vergleichsweise lange. Um bei einer Änderung einzelner Parameter nicht auch die unveränderten Programmteile neu berechnen zu müssen, werden *IDs* erzeugt. Diese beinhalten sämtliche in der config.m ausgewählten Parameter für verschiedene Stadien der Berechnung. Alle Dateien im *Output* Ordner werden nach diesen IDs benannt. So können Zwischenergebnisse zweifelsfrei identifiziert und - so sie den aktuellen Einstellungen der config.m entsprechen - direkt für eine weitere Berechnung benutzt werden.

An diesen IDs muss nichts verändert werden.

10.1.3 Media Object

Nachdem der *Media Converter* sämtliche Parameter der `config.m` initialisiert hat, erzeugt dieser ein *Media Object*. In diesem werden der Sequenz zugehörigen Parameter und Zustände gespeichert. Es hat Zugriff auf die Sequenzen nach den verschiedenen Bearbeitungsschritten. Außerdem kontrolliert es die Parameter `processing_frames` und `start_frame`, sowie die Eigenschaften *Frame Rate*, *Height* und *Width*. Diese verändern sich je nach Bearbeitungsschritt. Das *Media Object* speichert sie jeweils separat ab, sodass auch im Nachhinein jeder der Zustand rekonstruiert werden kann. Es stellt außerdem *Getter-* und *Setter-* Methoden für sämtliche Variablen bereit.
Da das *Media Object* alle Informationen über die Sequenz enthält, ist dieses auch für die finale Ausgabe und den *Preview* zuständig. Dafür enthält es folgende Funktionen:

```
1 %Media_Object.m
2 function preview_ME(obj,frame,preview_ME_vector_subsampling)
3         %function body
4 end;
5
6 function output(obj,output_image_sequence,output_motion_vector_sequence,
7    output_directory,output_motion_vector_sequence_subsampling,US_ID)
8         %function body
9 end;
```

Import

Eine essentielle Aufgabe des *Media Objects* ist das Importieren der Bilderfolge, bzw. der Videodatei. Der Importiervorgang besteht aus drei Funktionen, die abhängig vom Dateityp und von vorherigen Programmdurchläufen, von der Klasse `Media_Converter` nacheinander aufgerufen werden.

```
1 %Media_Object.m
2 function create_input_video_sequence(obj, Input_ID,output_directory)
3         %function body
4 end;
5
6 function setup_processing(obj)
7         %function body
8 end;
9
10 function import_image_sequence(obj,PI_ID, output_directory)
11         %function body
12 end;
```

Die Funktion `create_input_video_sequence()` sorgt dafür, dass eine Bildersequenz in eine Videodatei überführt wird. Es werden sämtliche Bitmap-Dateien des Quellordners nacheinander eingelesen und in eine Videodatei geschrieben, die mit dem Namen `Input_ID`

im Ordner `output_directory/Import/` gespeichert wird. Dieser Schritt muss nur bei der ersten Verwendung der Datei ausgeführt werden und auch nur, wenn es sich um eine Bilderfolge handelt.

`setup_processing()` steuert die Parameter für die Verarbeitung entsprechend der Sequenzeigenschaften. Es wird überprüft, ob `input_start_frame` und `input_processing_frames` mit der Eingangsequenz realisierbar sind, oder ob sie über die Sequenzgrenzen hinaus zeigen. Außderdem werden diese beiden Parameter an den Modus *interlaced* oder *progressive* angepasst:

```
1 %Media_Object.m
2 if (strcmp(obj.input_mode,'interlaced'))
3     obj.processing_frames = 2 * obj.processing_frames;
4     obj.start_frame = 2 * obj.start_frame-1;
5 end;
```

Die jeweilige Verdoppelung der Werte ist notwendig, da es sich bei den Vorlagen, wie in 9.4 beschrieben, um geschachtelte Halbbilder handelt. Da für die weitere Verarbeitung der *interlaced* Sequenzen die Halbbilder aus der Eingangssequenz auseinander gezogen werden, verdoppeln sich somit die Werte.
Um die Halbbildtypen identifizieren zu können, wird bei ungeraden Halbbildern unten eine schwarze Zeile angefügt, bei geraden Halbbildern oben. Dies trifft lediglich auf *interlaced* Eingangssignale zu.

Für diese Trennung der Halbbilder ist die dritte Funktion `import_image_sequence()` zuständig. Sie extrahiert die zwei Halbbildtypen aus dem Eingangsbild und schreibt sie nacheinander in eine neue Videosequenz. Diese besitzt dementsprechend die doppelte *Frame Rate* und die einzelnen Halbbilder haben nur noch die halbe Zeilenanzahl. Programmintern wird diese Seqzenz als `post_import_video_sequenz` bezeichnet und ebenfalls im Ordner `output_director/Import/` mit dem Namen der `PI_ID` (Post Import ID) gespeichert. Um die Halbbilder weiterhin eindeutig zu identifizieren, werden ungerade Halbbilder mit einer schwarzen Zeile am Ende versehen, gerade Halbbilder beginnen mit einer schwarzen Zeile.
Sequenzen, die bereits *progressive* sind, werden nicht bearbeitet. Da im Folgenden allerdings mit der `post_import_video_sequenz` weitergearbeitet wird, kopiert die Funktion `import_image_sequence()` diese Sequenztypen mit dem neuen Namen der `PI_ID` an den selben Speicherort.

10.1.4 Motion Estimation

```
%config.m
ME_method = 'Phase Correlation';

ME_PC_block_size = 64;
```

In der Klasse `Motion_Estimation_Phase_Correlation.m` wird die Bewegungsschätzung vorgenommen. Für jeden Parameter gibt es das Präfix `ME_DI_` und das Präfix `ME_US_`. Da unter Umständen im Verlauf des Programms die *Motion Estimation* zweimal ausgeführt wird, gibt es die Möglichkeit für die Bewegungsschätzung beim *Deinterlacing* andere Werte zu setzen als für das *Upscaling*. Da die Wirkungsweise aller Variablen exakt gleich ist, werden sie in diesem Kapitel nur einmal verwendet und mit dem Präfix `ME_` dargestellt.

Innerhalb dieser Arbeit wird die `'Phase Correlation'` als mögliche Bewegungsschätzung implementiert. Wie in 5.4.2 beschrieben, wird das Eingangsbild für die *Motion Estimation* in Subblöcke unterteilt. Die Größe dieser Blöcke wird über den Parameter `ME_PC_block_size` geregelt. Innerhalb dieser Arbeit werden diese Blöcke immer quadratisch angenommen. Es werden die Werte $\{8, 16, 32, 64, 128\}$ unterstützt.

Windowing

```
%config.m
%ME_window = 'No';
%ME_window = 'Hamming';
ME_window = 'Decaying Extension';
```

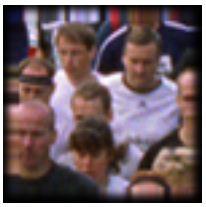
In Kapitel 5.6.1 wurden verschiedene Fensterfunktionen angesprochen, durch die der Kontrast innerhalb der Phasenkorrelationsebene erhöht werden soll. Der *Media Converter* bietet die Optionen `'No'`, `'Hamming'` und `'Decaying Extension'` für ein *Windowing* an.

No Window

Die Option `'No'` bedeutet, dass keine Fensterfunktion auf die Subblöcke angewendet wird (siehe 10.3).

No Window

Hamming Window - Hamming Window Size = 32



Decaying Extension Window - Delta = 5

Abbildung 10.3
Übersicht über die implementierten *Window* Funktionen

Hamming Window

```
1 %config.m
2 ME_hamming_window_size = 32;
```

Die Funktion `Hamming Window` wendet ein Hamming Fenster auf die Subframes an wie es auch in [10] genutzt wurde. Über die Funktion `ME_hamming_window_size` kann die Größe des Fensters bestimmt werden. Die einzelnen Subblöcke werden an den Grenzen um die Hälfte des Fensters mit angrenzenden Pixeln des originalen *Frames* erweitert. Auf diese erweiterten *Frames* wird das Fenster angewendet. Abbildung 10.3 zeigt die Filterung des Subblocks. Die Vergrößerung und der Übergang zu schwarz an den Kanten ist deutlich sichtbar.

Decaying Extension Window

```
1 %config.m
2 ME_Decaying_Extension_delta = 5;
```

Die Funktion `'Decaying Extension'` setzt die dritte in 5.6.1 beschriebene Fenstermöglichkeit um. Über den Parameter `ME_Decaying_Extension_delta` kann die von Ahmed et al. [1] eingeführte Erweiterung δ eingestellt werden. In Abbildung 10.3 wird deutlich, dass die zusätzlichen Pixel am Rand allesamt den Wert des ihnen am nähesten liegenden Pixel des Subblocks besitzen. Außerdem wurde eine Gaußfunktion auf die hinzugekommenen Randpixel angewendet.

Phasenkorrelation

```
1  %Motion_Estimation_Phase_Correlation.m
2  ME_subframe_n_fft = fft2(ME_subframe_n);
3  ME_subframe_n1_fft = fft2(ME_subframe_n1);
4
5  %Phase Correlation
6  temp=ME_subframe_n_fft.*conj(ME_subframe_n1_fft);
7  temp(temp==0)=1e-32; %Avoid division by zero
8  q = (temp) ./ (abs(temp));
9
10 q(isnan(q))=0;
11 verschiebung = ifft2(q);
12 verschiebung = abs(fftshift(verschiebung));
```

Nachdem die Fensterfunktion auf die Blöcke angewendet wurde, wird die Phasenkorrelation wie in 5.4.1 beschrieben durchgeführt. Zunächst werden die *Subframes* über die Funktion `fft2` in den Frequenzbereich transformiert. Im nächsten Schritt wird eine temporäre Variable erzeugt, die das Produkt der Einzelelemente aus dem Spektrum von `ME_subframe_n` und dem konjugiert Komplexen des Spektrums vom nachfolgenden Subblock `ME_subframe_n1` beinhaltet. Um einer Division durch Null vorzubeugen, werden alle Elemente in `temp`, die gleich Null sind auf einen sehr kleinen Wert ungleich 0 gesetzt. Wie in Formel 5.7 wird die Division der Einzelelemente von `temp` mit dem Absoluten von `temp` durchgeführt.
Durch die Rechengenauigkeit kann es unter Umständen zu Elementen im Ergebnis `q` kommen, die keine Zahl (`NaN`) sind. Um diesen Fehler zu vermeiden, werden die Elemente auf den Wert Null gesetzt. Dieser ist unproblematisch, da im Folgenden die *Peaks* aus dieser Ergebnisebene gesucht werden, wobei die größten Werte entscheidend sind.
Dem folgen die Rücktransformation in den Ortsbereich `ifft2(q)` und die Verschiebung der Ergebnisse in einen besser zu interpretierenden Bereich. Der letzte Schritt wird lediglich für eine bessere Handhabung im Folgenden durchgeführt. Am Ergebnis ändert dieser nichts.

Peakfinding

```
1  %config.m
2  ME_PC_calculated_vectors = 9;
3
4  ME_threshold_peak_finding = 0.05;
```

Nach der Phasenkorrelation müssen die *Peaks*, die den Verschiebungen im Bild entsprechen, aus den berechneten Ebenen extrahiert werden. Für diesen Schritt wird in der `config.m` festgelegt, wie viele *Peaks* pro Block extrahiert werden sollen. Die Variable `ME_PC_calculated_vectors` kann dabei Werte im Bereich $\{3, ..., 9\}$ annehmen. Außerdem

kann ein Schwellenwert `ME_threshold_peak_finding` eingestellt werden. *Peaks*, die unter diesen Schwellenwert fallen werden als Rauschen interpretiert und nicht als *Peaks* identifiziert. In diesem Fall gibt es weniger Vektoren als in `ME_PC_calculated_vectors` definiert. Die Variable mit der Phasenkorrelationsebene (**verschiebung**) wird an einen *Peakfinding*-Algorithmus übergeben:

```
1 %Motion_Estimation_Phase_Correlation.m
2 ME_motion_vector_block = find_max_values(verschiebung);
```

Die Arbeit des *Peakfinding*-Algorithmus ist in Abbildung 10.4 verdeutlicht. Es werden die höchsten *Peaks* gesucht und deren Position nacheinander in ein *Array* geschrieben. In der Abbildung handelt es sich um die Phasenkorrelationsebene des *Decaying Extension* Fensters. Die Blockgröße betrug 64 Pixel. Für das Fenster war ein δ-Wert von 5 Pixeln eingestellt. Durch die Addition auf beiden Seiten ergibt sich somit eine Blockgröße von 74×74 Pixeln. Der Mittelpunkt liegt also bei (37/37). Da MATLAB allerdings systembedingt den Mittelpunkt bei der Fouriertransformation und -rücktransformation um eine Einheit je Richtung verschiebt, liegt der Mittelpunkt in der Ebene bei (38/38). In Abbildung 10.4 sieht man deutlich den *Peak* im Mittelpunkt (0/0) bzw. (38/38). Von dort aus gesehen liegen andere *Peaks* bei (2/0), (3/0) oder $(1/-1)$. Der Inhalt der Variablen `ME_motion_vector_block` für den Block `m=18`, `n=10` ist in Tabelle 10.1 aufgelistet. Die zweite Spalte beinhaltet die Höhe der *Peaks*.

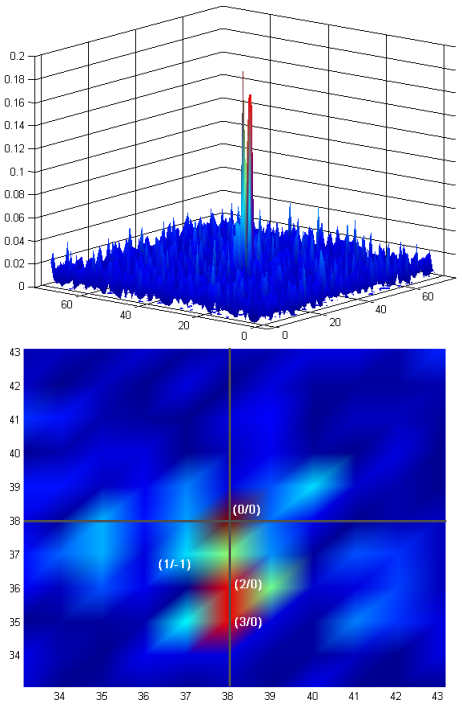


Abbildung 10.4
Peakfinding - Phasenkorrelationsebene *Decaying Extension Window*

Hinweis: Programmintern ist der Koordinatenursprung oben links. Außerdem wird in der Reihenfolge (Y/X) gearbeitet.

Nach der Durchführung der Phasenkorrelation für alle Blocks eines *Frames*, beinhaltet die Variable `ME_motion_vectors` der Klasse `Motion_Estimation_Phase_Correlation.m` für jeden Block eines *Frames* alle berechneten Bewegungsvektoren. Diese müssen im Folgenden entsprechend des zweiten Teils von Algorithmus 5.4.2 den einzelnen Pixeln zugewiesen werden.

Blockmatching

```
1 %config.m
2 ME_blockmatching_noise_tolerance = 0.020;
```

Ziel des *Blockmatchings* ist jedem einzelnen Pixel einen Bewegungsvektor zuzuweisen, der möglichst genau die Bewegung des Pixels zum nächsten *Frame* beschreibt (siehe 5.5). In der Klasse `Motion_Estimation_Phase_Correlation.m` wird für jeden Block eine Auswahl möglicher Vektoren zusammengestellt. `ME_motion_vector_test_set` speichert die Vektoren des aktuellen und der umliegenden Blöcke. Außerdem werden doppelte Vektoren aussortiert. Danach wird in `block_matching.m` jeder Pixelwert $F(\vec{x}, n)$ mit jedem Wert $F(\vec{x} + \vec{d_i}, n + 1)$ verglichen und ein *Error* gebildet ($\vec{d_i}$ sind alle Vektoren des Test Sets). Nach diesem Schritt wird überprüft für welchen Bewegungsvektor der *Error* am geringsten ist. Dieser Vektor wird dem jeweiligen Pixel zugeordnet.

Nr.	Peakhöhe	Y	X
1	0,1855	0	0
2	0,1628	2	0
3	0,1373	3	0
4	0,0962	1	-1
5	0,0936	1	0
6	0,0890	2	-1
7	0,0798	1	1
8	0,0685	3	1
9	0,0680	0	1

Tabelle 10.1
Inhalt der Variablen `ME_motion_vector_block`, 9 Vektoren, *Decaying Extension Window*

Es werden zunächst kürzere Bewegungen ausprobiert, bevor die längeren getestet werden. Um zu vermeiden, dass große Verschiebungen durch zufälliges Rauschen bessere Werte erzielen und deshalb angewendet werden, kann über die Variable `ME_blockmatching_noise_tolerance` eine Schranke eingestellt werden. Große Vektoren müssen somit deutlich besser sein als kurze damit deren Wert genommen wird.

Um die Genauigkeit der Vektorzuordnung zu verbessern, werden in dieser Arbeit zwei Erweiterungen entwickelt. Als Ergänzung können das `Blockmatching Surrounding` und das `Blockmatching Prescreening` dazu- oder abgeschaltet werden. Auch eine Kombination beider Verfahren ist möglich.

Blockmatching Surrounding

```
1 %config.m
2 %ME_block_matching_surrounding = 'On';
3 ME_block_matching_surrounding = 'Off';
4
5     ME_blockmatching_surrounding_factor = 0.5;
```

Die Funktion `ME_block_matching_surrounding` bezieht die umliegenden acht Pixel mit in die Entscheidung ein, welcher Vektor am besten passt. Somit soll verhindert werden, dass

falsche Vektoren lediglich auf Grund von Rauschen eines einzelnen Pixels angewendet werden. Für jedes Pixel wird einzeln ein *Error* berechnet. Danach wird die Summe der *Errors* der umliegenden Pixel mit dem Faktor `ME_blockmatching_surrounding_factor` gewichtet und zu dem nicht gewichteten *Error* des aktuellen Pixels addiert. Auch dieser Schritt wird für jeden Vektor aus dem Test Set durchgeführt und der Vektor zu dem der kleinste *Error* gehört wird ausgewählt

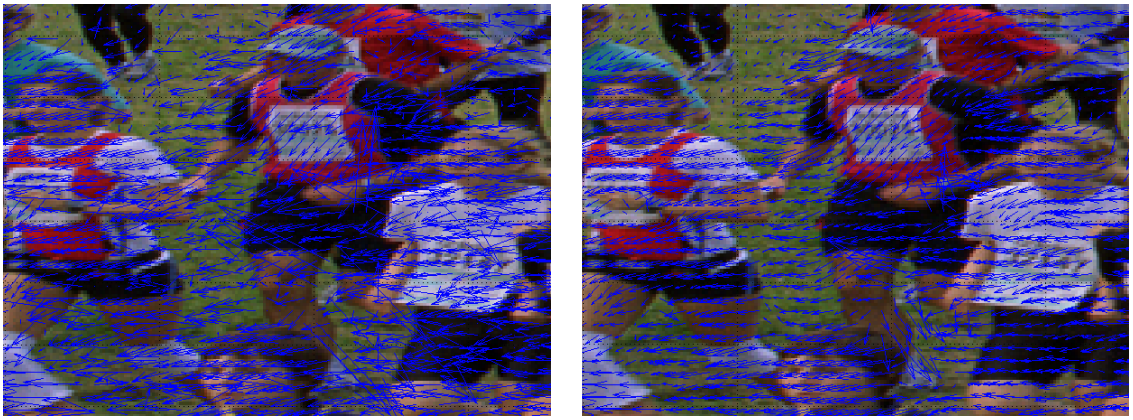
Blockmatching Prescreening

```
%config.m

%ME_block_matching_prescreening = 'On';
ME_block_matching_prescreening = 'Off';

            %Schwellen fuer die moeglichen Vektoren
    ME_blockmatching_prescreening_y_window = 5;
    ME_blockmatching_prescreening_x_window = 5;
            %Fenstergroesse
    ME_blockmatching_prescreening_block_x_start = -3;
    ME_blockmatching_prescreening_block_y_start = -3;
    ME_blockmatching_prescreening_block_x_end = 4;
    ME_blockmatching_prescreening_block_y_end = 4;
```

Dieser Ansatz soll eine Robustheit gegen das *Aperture* Problem entwickeln. Zu diesem Zweck wird der Pixel, für den ein Vektor berechnet werden soll, als Mittelpunkt eines Blocks $x \times x$ angenommen. Der Block wird über die Parameter unter `%Fenstergroesse` (siehe Code-Beispiel) eingestellt. Die Werte sind relativ zur Position des aktuellen Pixels. Für den gesamten Block wird ähnlich dem *Blockmatching Surrounding* der beste Vektor ermittelt. Danach werden alle Vektorkandidaten mit diesem Vektor verglichen und diejenigen, bei denen die Abweichung größer einem Schwellenwert ist werden aus der Liste gestrichen. Die Schwellenwerte können über `ME_blockmatching_true_motion_y_window` und `ME_blockmatching_true_motion_x_window` gesondert für jede Achse angegeben werden. Aus den übrig gebliebenen Vektoren wird einer für das Pixel ausgewählt. Dieses Verfahren verfolgt einen impliziten Ansatz um *True Motion* zu berechnen.

Smoothing

```
%config.m
%ME_smoothing_filter = 'on';
ME_smoothing_filter = 'off';

ME_smoothing_filter_kernel = [7 7 7];
```



Ohne *Smoothing* *Smoothing - Kernel* [7 7 7]

Abbildung 10.5
Vektoren ohne und mit *Smoothing* Funktion

Obwohl die *Blockmatching* Verfahren darauf abzielen, möglichst die wahre Bewegung eines Pixels zu erfassen, gibt es dennoch immer wieder Unregelmäßigkeiten im Vektorfeld (siehe Abbildung 10.5). Insbesondere einzelne falsche Vektoren können zu ungenügenden Ergebnissen führen. Um das Vektorfeld zu glätten kann ein *Smoothing*-Filter ausgewählt werden. Dieser dreidimensionale Filter wird über den Filterkernel eingestellt. Empfohlen werden die Werte $\{3, 5, 7\}$ (jeweils für alle drei Elemente des Kernels), wobei ein ungerader Wert eine Voraussetzung für diesen Filter ist.
Auch wenn das Vektorfeld wie in Abbildung 10.5 deutlich gleichmäßiger ist und auf den ersten Blick der wahren Bewegung der Objekte entspricht, muss mit dem *Smooting*-Filter vorsichtig umgegangen werden. Insbesondere an den Kanten sich bewegender Objekte tritt eine Unregelmäßigkeit im Vektorfeld auf. Diese sorgt später für einen scharfen Eindruck im Gesamtergebnis. Der *Smoothing*-Filter kann jedoch nicht unterscheiden, ob eine Unregelmäßigkeit im Vektorfeld durch eine Ecke im Originalbild auftritt und somit gewünscht ist, oder ob sie durch falsche Vektoren entsteht. Somit sorgt der Filter nicht nur für die Korrektur falscher Vektoren, sondern unter Umständen auch für eine Ungenauigkeit richtiger Vektoren. Dieses Verfahren entspricht der expliziten Umsetzung der *True Motion* (siehe 5.5).

10.1.5 Deinterlacing

```
%config.m
%non-motion compensated (Es wird keine Bewegungssch"atzung vorgenommen)
    %DI_method = 'Line Doubling';
    %DI_method = 'Line Average';
    %DI_method = 'Field Repetition';
    %DI_method = 'Vertical Temporal Median';

%motion compensated (Eine Beweungssch"atzung wird berechnet)
    DI_method = 'MC Field Insertion';
    %DI_method = 'MC Field Average';
    %DI_method = 'MC Median Insertion';
    %DI_method = 'MC Median Average';
```

Für das *Deinterlacing* stellt der *Media Converter* insgesamt acht verschiedene Methoden zur Verfügung. Es handelt sich um die vier nicht bewegungskompensierenden Verfahren *Line Doubling*, *Line Average*, *Field Repetition* und *Vertical Temporal Median*, sowie die vier bewegungskompensierenden Verfahren *MC Field Insertion*, *MC Field Average*, *MC Median Insertion* und *MC Median Average*. Über die Variable `DI_method` kann ein Verfahren ausgewählt werden.
Im Folgenden wird lediglich auf die Umsetzung der bewegungskompensierenden Verfahren eingegangen. Die nicht bewegungskompensierenden Verfahren sind nur zu Vergleichszwecken mit aufgenommen. Ausgenommen das `'Field Repetition'`-Verfahren wurden für die nicht bewegungskompensierenden Verfahren die in MATLAB bereitgestellten Funktionen genutzt.

Alle bewegungskompensierenden Methoden werden in einem zweistufigen Verfahren durchgeführt. Zunächst wird auf jedes Pixel der **`Post Import Sequenz`**, die aus aufeinanderfolgenden Halbbildern bei doppelter *Frame Rate* (siehe 10.1.3) besteht, der für diesen Pixel errechnete Bewegungsvektor angewendet. Somit entsteht eine neue Sequenz (`output_directory/Temp/DI_applied_Motion...avi`). In dieser Sequenz sind alle Pixel bereits bewegungskompensiert. Da die Bewegungsvektoren zwischen zwei Halbbildern gleichen Typs errechnet wurden ist zu beachten, dass diese vor der Anwendung halbiert werden müssen. Im zweiten Schritt des Verfahrens wird die neue, bewegungskompensierte Sequenz mit der **`Post Import Sequenz`** geschachtelt. Die zwei Halbbilder verschiedenen Typs werden zu einem Vollbild zusammengeführt.

Im Folgenden wird beschrieben, wie sich die Verfahren im Detail unterscheiden.

MC Field Insertion

Für das Verfahren `MC Field Insertion` wurde die Funktion `apply_ME()` entwickelt. Diese wendet die Bewegungsvektoren auf die einzelnen Pixel an. Es wird zunächst der Bewegungsvektor aus oben beschriebenen Gründen halbiert:

```
1 %apply_ME.m
2 DI_x_motion=round(MO_post_import_video_motion(j,k,2)/2);
3 DI_y_motion=round(MO_post_import_video_motion(j,k,1)/2);
```

In einem zweiten Schritt werden die Positionen bestimmt an denen das Pixel sich nach der Anwendung der Vektoren befindet. j und k sind die Position des Pixels im Subblock. Es wird nicht das zu füllende Bild durchgegangen und für jeden Pixel ein Wert errechnet, sondern das zu füllende `DI_frame` ist vorinitialisiert (siehe Abschnitt *Background* unten) und es wird das Ursprungsbild Pixel für Pixel anhand des berechneten Vektors verschoben.

```
1 %apply_ME.m
2 DI_x_position=k+DI_x_motion;
3 DI_y_position=j+DI_y_motion;
```

In der Variablen `DI_frame` wird das neue, bewegungskompensierte Bild gespeichert. An der bewegungskompensierten Position (`DI_y_position`/`DI_x_position`,:) wird dieser der Wert des Ursprungsbildes `DI_current_field(j,k,:)` zugewiesen.

```
1 %apply_ME.m
2 DI_frame(DI_y_position,DI_x_position,:)=DI_current_field(j,k,:);
```

Nachdem dieses Verfahren für jedes *Frame* abgeschlossen ist, wird der zweite Schritt des *Deinterlacing* durchgeführt.

Es werden zunächst zwei Bilder definiert:

```
1 %Deinterlacing_MC.m
2 DI_field_n1 = mat2gray(read(DI_MC_sequence,i-1));
3 DI_field_n = mat2gray(read(DI_input_video_object,i));
```

Das erste Bild `DI_field_n1` ist das in Schritt eins errechnete bewegungskompensierte Halbbild. Da innerhalb dieser Sequenz jedes *Field* bewegungskompensiert und an der selben Position wie zuvor wieder gespeichert wurde, gehört das Vorgängerbild `i-1` zum Bild `i` des zweiten definierten Halbbildes. In der Variablen `DI_field_n` wird das Halbbild aus der `Post Import Sequenz` (=`DI_input_video_object`) gespeichert.

Die Bilder werden folgendermaßen geschachtelt:

```
1  %Deinterlacing_MC.m
2  for j=1:MO.input_image_height
3      if (mod(i,2)==1)
4          if (mod(j,2)==1)
5              DI_frame(j,:,:)=DI_field_n(fix(j/2)+1,:,:);
6          else
7              DI_frame(j,:,:)=DI_field_n1(fix(j/2)+1,:,:);
8          end
9      else
10         if (mod(j,2)==1)
11             DI_frame(j,:,:)=DI_field_n1(fix(j/2)+1,:,:);
12         else
13             DI_frame(j,:,:)=DI_field_n(fix(j/2)+1,:,:);
14         end
15     end
16 end
```

Es wird jede Zeile des Bildes einzeln betrachtet. In dem Code steht `i` für das aktuell zu bearbeitende *Frame*. Über die erste Modulo-Abfrage `(mod(i,2)==1)` wird geprüft, ob es sich um ein grades (*even*) oder um ein ungerades (*odd*) Halbbild handelt in das das bewegungskompensierte Halbbild geschachtelt werden soll. Handelt es sich um ein ungerades ($mod(i,2) == 1$), so wird die obere IF-Abfrage ausgeführt, handelt es sich um ein gerades ($mod(i,2) == 0$), so wird die untere IF-Abfrage durchgeführt. Innerhalb beider Abfragen wird je nachdem, ob es sich um eine gerade oder ungerade Zeile `(mod(j,2)==1)` handelt entweder das bewegungskompensierte Field `DI_field_n1` oder das unbearbeitete `DI_field_n` genommen. Programmintern werden *Odd*-Halbbilder mit einer schwarzen Zeile unten, *Even*-Halbbilder mit einer schwarzen Zeile oben gespeichert. Aus diesem Grund ergibt sich für die Zeilenauswahl in jedem Fall `fix(j/2)+1`.

MC Field Average

Die Funktion `'MC Field Average'` setzt das Verfahren aus 6.2.1 um. Es ähnelt dem Verfahren `'MC Field Insertion'` bis auf die Pixelberechnung bei der Bewegunskompensation. Anstatt lediglich den Wert des Pixels aus dem vorhergehenden Bild für die Berechnung zu benutzen, wird nun auch der Wert des Pixels aus dem nachfolgenden Bild genommen. Da die Bewegungsvektoren für *Interlaced* Sequenzen mit einem Bildabstand von zwei Bildern durchgeführt wird, muss der jeweilige Vektor halbiert werden. Wie bei der `'MC Field Insertion'` benutzt man diesen um das Pixel aus dem vorhergehenden Bild zu finden, allerdings kann ein Vektor selber Länge auch als ein Zeiger auf das folgende Bild interpretiert werden. So wird auch dafür ein Pixelwert bestimmt, der mit umgekehrtem Vektor in das gesuchte Bild zurückgeführt wird. Im *Media Converter* übernimmt die Klasse `apply_ME_field_average.m` diese Bearbeitung.

```
%apply_ME_field_average.m
DI_x_position_following=round(k+MO_post_import_video_motion(j,k,2)-DI_x_motion);
DI_y_position_following=round(j+MO_post_import_video_motion(j,k,1)-DI_y_motion);

DI_x_grid_following=round(k+MO_post_import_video_motion(j,k,2));
DI_y_grid_following=round(j+MO_post_import_video_motion(j,k,1));
```

Da der erste Teil der Bearbeitung analog zum `'MC Field Insertion'` funktioniert und somit die Werte für die vorherigen Bildpixel schon im `DI_frame` eingetragen sind, wird im zweiten Schritt der Mittelwert genommen, wenn ein Vektor des folgenden Bildes auf eine Stelle im `DI_frame` zeigt.

```
%apply_ME_field_average.m
DI_frame(DI_y_position_following,DI_x_position_following,1) =
        0.5*DI_frame(DI_y_position_following,DI_x_position_following,1)
        + 0.5*DI_following_field(DI_y_grid_following,DI_x_grid_following,1);

DI_frame(DI_y_position_following,DI_x_position_following,2) =
        0.5*DI_frame(DI_y_position_following,DI_x_position_following,2)
        + 0.5*DI_following_field(DI_y_grid_following,DI_x_grid_following,2);

DI_frame(DI_y_position_following,DI_x_position_following,3) =
        0.5*DI_frame(DI_y_position_following,DI_x_position_following,3)
        + 0.5*DI_following_field(DI_y_grid_following,DI_x_grid_following,3);
```

Die Verschachtelung der zwei Halbbildtypen funktioniert genau wie oben.

MC Median Insertion/Average

Aus Abschnitt 6.2.1 wird deutlich, dass das Median Verfahren lediglich die bei der finalen Entscheidung, welchen Wert ein Pixel letztendlich bekommen soll, die Menge der möglichen Pixel erweitert und je nach Fall eine andere Entscheidung trifft. So wird bei diesem Verfahren nicht mehr zwingend der durch die `'MC Insertion'` oder `'MC Average'` bestimmte Wert genommen, sondern es wird ein Median aus diesem Wert und den Pixeln an der selben horizontalen Stelle der benachbarten Zeilen gebildet. Durch dieses Verfahren werden falsche Bewegungsvektoren herausgefiltert. Da für die Berechnung des Medians dennoch das Ergebnis für die `'MC Field Insertion'` bzw. die `'MC Field Average` benötigt wird, ist der erste Teil dieses Verfahrens analog zum *MC Field Insertion*-, bzw. zum *MC Field Average*-Verfahren oben. Es ändert sich jedoch der Teil der Schachtelung der beiden Halbbilder.

```
%Deinterlacing_MC.m
for j=2:MO.input_image_height-1
  if (mod(i,2)==1)
    if (mod(j,2)==1)
      DI_frame(j,:,:)=DI_field_n(fix(j/2)+1,:,:);
    else
      for k=1:MO.input_image_width
        DI_frame(j,k,1) = median([DI_field_n(fix(j/2),k,1),
                          DI_field_n(fix(j/2)+1,k,1), DI_field_n1(fix(j/2)+1,k,1)]);
        DI_frame(j,k,2)=median([DI_field_n(fix(j/2),k,2),
                          DI_field_n(fix(j/2)+1,k,2), DI_field_n1(fix(j/2)+1,k,2)]);
        DI_frame(j,k,3)=median([DI_field_n(fix(j/2),k,3),
                          DI_field_n(fix(j/2)+1,k,3), DI_field_n1(fix(j/2)+1,k,3)]);
      end;
    end;
  else
    if (mod(j,2)==1)
      for k=1:MO.input_image_width
        DI_frame(j,k,1)=median([DI_field_n(fix(j/2)+1,k,1),
                          DI_field_n(fix(j/2)+2,k,1), DI_field_n1(fix(j/2)+1,k,1)]);
        DI_frame(j,k,2)=median([DI_field_n(fix(j/2)+1,k,2),
                          DI_field_n(fix(j/2)+2,k,2), DI_field_n1(fix(j/2)+1,k,2)]);
        DI_frame(j,k,3)=median([DI_field_n(fix(j/2)+1,k,3),
                          DI_field_n(fix(j/2)+2,k,3), DI_field_n1(fix(j/2)+1,k,3)]);
      end;
    else
      DI_frame(j,:,:)=DI_field_n(fix(j/2)+1,:,:)
    end;
  end;
end;
```

Die Schachtelung hat die selbe Struktur wie die beim `'MC Field Insertion'`. Im Wesentlichen unterscheidet sie sich darin, dass für die Zeilen, die neu in das Halbbild eingefügt werden, nicht mehr nur der Wert des bewegungskompensierten Bildes `DI_field_n1` genommen wird, sondern der Median aus dem Pixel des Bildes `DI_field_n1` und den räumlich nächsten Pixeln der oberen und unteren Zeile des aktuellen Halbbildes (`DI_field_n`). Da für die erste und letzte Zeile jeweils ein Wert für die Median Bildung fehlt, werden diese gesondert behandelt (siehe `Deinterlacing_MC.m`) und die `for`-Schleife läuft nur von der zweiten bis zur vorletzten Zeile.

Obscured und Uncovered Background

Einer besonderen Bedeutung kommt beim *Motion Compensating Deinterlacing* dem *obscured* und *uncovered Background* zu. Das Problem der Doppelbelegung (*obscured Background*) fällt im Gesamtbild nicht auf, da durch die Herangehensweise bei der Bewegungskompensation aus mehreren möglichen Pixelwerten die wahrscheinlichste genommen wird. Durch die Bewegung der Objekte in einer Szene werden aber zwangsläufig Bildbereiche frei, die zuvor nicht zu sehen waren und auf die keine Bewegungsvektoren verweisen (*uncovered Background*). Wie in Abschnitt 5.6 beschreiben, kann durch eine Vorinitialisierung

des bewegungskompensierten Frames der freigelegte Hintergrund mit Farbwerten versehen werden. Zu diesem Zweck sind im *Media Converter* drei Funktionen implementiert.

```
%config.m
%DI_background = 'Zero';
DI_background = 'Following Field';
%DI_background = 'Subsequent Average';
```

Zero Background

Die Funktion `'Zero Background'` dient der Verdeutlichung, wie ein Bild aussähe, würde keinerlei Maßnahme gegen den *Uncovered Background* durchgeführt. Das `DI_frame` (beinhaltet das bewegungskompensierte Bild) wird lediglich mit schwarzen Pixeln initialisiert. Somit bleiben alle Pixel, für die keine Bewegung vorhanden ist, schwarz.

```
%Deinterlacing_MC.m
DI_frame(:,:,:)=0;
```

Following Field Background

Mit dieser Funktion wird die Vorinitialisierung der Variablen `DI_frame` mit dem nächsten folgenden Halbbild des selben Typs umgesetzt (siehe Abschnitt 5.6).

```
%Deinterlacing_MC.m
DI_frame=mat2gray(read(DI_input_video_object,i+2));
```

Subsequent Average Background

Der *Subsequent Average Background* Algorithmus arbeitet im ersten Teil wie die *Zero* Methode. Alle Pixel für die bewegungskompensierte Informationen vorhanden sind, werden gesetzt und alle anderen werden schwarz gelassen. In einem weiteren Schritt werden jedoch alle schwarzen Pixel durch den Mittelwert aus den Werten der Pixel mit selber horizontaler Position aus der vorhergehenden und der nachfolgenden Zeile ersetzt. Diese Referenzpixel sind vorhanden, da zu diesem Zeitpunkt die Schachtelung mit dem vorhandenen Halbbild bereits vollzogen ist.

10.1.6 Upscaling

Das *Upscaling* ist der letzte Bearbeitungsschritt des Programms *Media Converter*. Damit dieser durchgeführt werden kann, muss eine *progressive* Sequenz vorliegen. Dementsprechend wird das *Upscaling* je nach Eingangsmaterial direkt, bzw. erst nach dem *Deinterlacing* durchgeführt. Als Ausgabe-*Frame Rate* bietet der *Media Converter* folgende Einstellmöglichkeiten an:

```
1 %config.m
2 US_output_framerate = 50;
3 %US_output_framerate = 60;
4 %US_output_framerate = 100;
5 %US_output_framerate = 120;
```

Es ist zu beachten, dass über die Einstellung der Ausgabe-*Frame Rate* ebenfalls gesteuert werden kann, ob überhaupt ein *Upscaling* durchgeführt werden soll. Stimmt die *Frame Rate* der Sequenz vor dem *Upscaling* mit der Einstellung in der `config.m` überein, so wird keine Berechnung gestartet.
Darüber hinaus wird darauf hingewiesen, dass bei den Einstellungen $100Hz$ und $120Hz$ die Abspielgeschwindigkeiten halbiert werden und somit die *Frame Rate* der Ausgabesequenzen ebenfalls $50Hz$ bzw. $60Hz$ beträgt. Dieser Schritt wird vollzogen, da aktuelle Hardware größtenteils nur Bildwiederholraten bis $60Hz$ unterstützt. Durch die Halbierung der Abspielgeschwindigkeit können die Sequenzen dennoch begutachtet werden.

```
1  %config.m
2  %non-motion compensated (Es wird keine Bewegungsschaetzung vorgenommen)
3      %US_method = 'Frame Repetition';
4      %US_method = 'Frame Average';
5
6  %motion compensated (Eine Beweungsschaetzung wird berechnet)
7      %US_method = 'MC Frame Insertion';
8      %US_method = 'MC Frame Average';
9      %US_method = 'MC static Median';
10     US_method = 'MC dynamic Median';
```

Auch im *Upscaling* Bereich werden einzelne nicht bewegungskompensierende Verfahren zu Vergleichszwecken angeboten. Es handelt sich um das Verfahren `'Frame Insertion'` bei dem einzelne *Frames* wiederholt werden solange bis die gewünschte *Frame Rate* erreicht ist (siehe Kapitel 7.1.1) und das Verfahren `'Frame Average'`, bei dem das vorherige und das nachfolgende *Frame* anteilig kombiniert werden (siehe Kapitel 7.1.2).
Das *Upscaling* funktioniert in Ansätzen ähnlich dem *Deinterlacing*. Es werden das vorherige *Frame* `US_image_n1` und das nachfolgende *Frame* `US_image_n` definiert. Diese beiden *Frames* werden zusammen mit einer Variablen `m`, die den zeitlichen Abstand des zu berechnenden *Frames* vom vorherigen und nachfolgenden *Frame* definiert ($0 <= m <= 1$), an die jeweilige Funktion für das *Upscaling* weitergegeben.

MC Frame Insertion

Das Verfahren 'MC Frame Insertion' wird in apply_ME_US_MC_Frame_Insertion.m durchgeführt. Es funktioniert analog zu Schritt eins der *Deinterlacing* Algorithmen. Die Variablen j und k definieren die Pixelposition.

```
%apply_ME_US_MC_Frame_Insertion.m
US_x_motion = round(US_motion(j,k,2)*(m-floor(m)));
US_y_motion = round(US_motion(j,k,1)*(m-floor(m)));

US_x_position = k+US_x_motion;
US_y_position = j+US_y_motion;

US_image_n(US_y_position,US_x_position,:) = US_image_n1(j,k,:);
```

Im Gegensatz zum *Deinterlacing* müssen für das *Upscaling* die Vektoren nicht halbiert werden. Bei einer Vollbildsequenz gibt es keine Bilder verschiedener Phase. An dieser Stelle müssen die Vektoren aber entsprechend des zeitlichen Abstandes zum vorherigen Bild gekürzt werden, sodass sie auf die richtige Stelle im zeitlichen Verlauf zeigen.

MC Frame Average

Auch beim *Upscaling* erweitert die 'MC Frame Average' Methode die Herangehensweise der 'MC Frame Insertion'. In der apply_ME_US_MC_Frame_Average.m wird ebenfalls zunächst das US_image mit den Werten der *Frame Insertion* belegt. Im zweiten Schritt wird jedoch eine Kombination aus diesen und denen des nachfolgenden *Frames* mit rückwärtsorientieren Bewegungsvektoren gebildet. Die Gewichtung erfolgt anhand der Variablen m ((1-(m-floor(m))) bzw. (m-floor(m))).

```
%apply_ME_US_MC_Frame_Average.m
US_x_motion=round(US_motion(j,k,2)*(1-(m-floor(m))));
US_y_motion=round(US_motion(j,k,1)*(1-(m-floor(m))));

US_x_position_following=round(k+US_motion(j,k,2)-US_x_motion);
US_y_position_following=round(j+US_motion(j,k,1)-US_y_motion);

US_x_grid_following=round(k+US_motion(j,k,2));
US_y_grid_following=round(j+US_motion(j,k,1));

US_image(US_y_position_following,US_x_position_following,1) =
    (1-(m-floor(m)))*US_image(US_y_position_following,US_x_position_following,1) +
    (m-floor(m))*US_image_n(US_y_grid_following,US_x_grid_following,1);
US_image(US_y_position_following,US_x_position_following,2) =
    (1-(m-floor(m)))*US_image(US_y_position_following,US_x_position_following,2) +
    (m-floor(m))*US_image_n(US_y_grid_following,US_x_grid_following,2);
US_image(US_y_position_following,US_x_position_following,3) =
    (1-(m-floor(m)))*US_image(US_y_position_following,US_x_position_following,3) +
    (m-floor(m))*US_image_n(US_y_grid_following,US_x_grid_following,3);
```

MC Static Median

Um ungenauen Vorhersagen zu entgehen, wird bei diesem Verfahren die Median Bildung mit eingeführt. Wie der Formel 7.4 zu entnehmen ist, wird dieser mit dem bewegungskompensierten Pixel, und den zeitlich benachbarten Pixeln der ursprünglich vorhandenen Bildern gebildet. Um dieses zu realisieren, wird die Funktion `apply_ME_US_MC_Frame_Average.m` um folgende Zeilen erweitert:

```
1 %apply_ME_US_MC_Static_Median.m
2 for j=1:MO_post_deinterlacing_image_height
3     for k=1:MO_post_deinterlacing_image_width
4         US_image(j,k,1) = median([US_image(j,k,1), US_image_n(j,k,1), US_image_n1(j,k,1)]);
5         US_image(j,k,2) = median([US_image(j,k,2), US_image_n(j,k,2), US_image_n1(j,k,2)]);
6         US_image(j,k,3) = median([US_image(j,k,3), US_image_n(j,k,3), US_image_n1(j,k,3)]);
7     end;
8 end;
```

Nachdem im `US_image` das bereits bewegungskompensierte Bild gespeichert ist, wird dieses in der `apply_ME_US_MC_Static_Median.m` im Vergleich zur `apply_ME_US_MC_Frame_Average.m` noch einmal mit dem Median gefiltert.

MC Dynamic Median

Die Berechnung des `'MC Dynamic Median'` wird über das bewegungskompensierte vorhergehende *Frame* `US_image_1`, das rückwärts bewegungskompensierte nachfolgende *Frame* `US_image_2` und das nicht bewegungskompensierte *Frame* aus `'Frame Average'` (`imageadd`) gebildet:

```
1 %apply_ME_US_MC_Dynamic_Median.m
2 for j=1:MO_post_deinterlacing_image_height
3     for k=1:MO_post_deinterlacing_image_width
4         US_image(j,k,1)=median([US_image_1(j,k,1), US_image_2(j,k,1), imageadd(j,k,1)]);
5         US_image(j,k,2)=median([US_image_1(j,k,2), US_image_2(j,k,2), imageadd(j,k,2)]);
6         US_image(j,k,3)=median([US_image_1(j,k,3), US_image_2(j,k,3), imageadd(j,k,3)]);
7     end;
8 end;
```

Die Bildung von `US_image_1` und `US_image_2` passiert analog zu der Anwendung der Bewegungkompensation beim *MC Frame Average*. Es wird lediglich kein Mittelwert gebildet und sie werden in zwei unterschiedlichen Variablen gespeichert. Der `'MC Dynamic Median'` wird in der Klasse `apply_ME_US_MC_Dynamic_Median.m` ausgeführt.

Obscured and Uncovered Background

```
1 %config.m
2 US_background = 'Zero';
3 %US_background = 'Following Field';
4 %US_background = 'Subsequent Average Edge Detection';
```

Beim Upscaling liegen fast identische Verhältnisse bezüglich des *uncovered Background* wie beim *Deinterlacing* vor. Dementsprechend sind die Optionen ähnlich, die unter diesem Punkt in der `config.m` auswählbar sind. Die Optionen `'Zero'` und `'Following Field'` arbeiten auf gleiche Art und Weise wie beim *Deinterlacing*. Der einzige Unterschied liegt in der Option `Subsequent Average Edge Detection'`.

Subsequent Average Edge Detection

Die Pixel des *uncovered Background* können beim *Upscaling* nicht durch die Pixel an der selben Zeilenposition der angrenzenden Zeilen rekonstruiert werden, da diese nicht zwingend mit Werten gefüllt sein müssen. Beim *Deinterlacing* war durch die Schachtelung der Zeilen gegeben, dass diese Pixel auf jeden Fall die richtigen Werte enthielten. Beim *Upscaling* tritt es hingegen häufig auf, dass sich um die bewegten Objekte schwarze Ränder bilden.
Die Funktion `'Subsequent Average Edge Detection'` weist diesen Pixeln Werte zu indem sie für jedes Pixel die Werte aller umliegenden Pixel addiert (ausgenommen diejenigen, die auch schwarz sind) und aus diesen Pixeln einen Mittelwert bildet.

11 Auswertung der Versuchsergebnisse

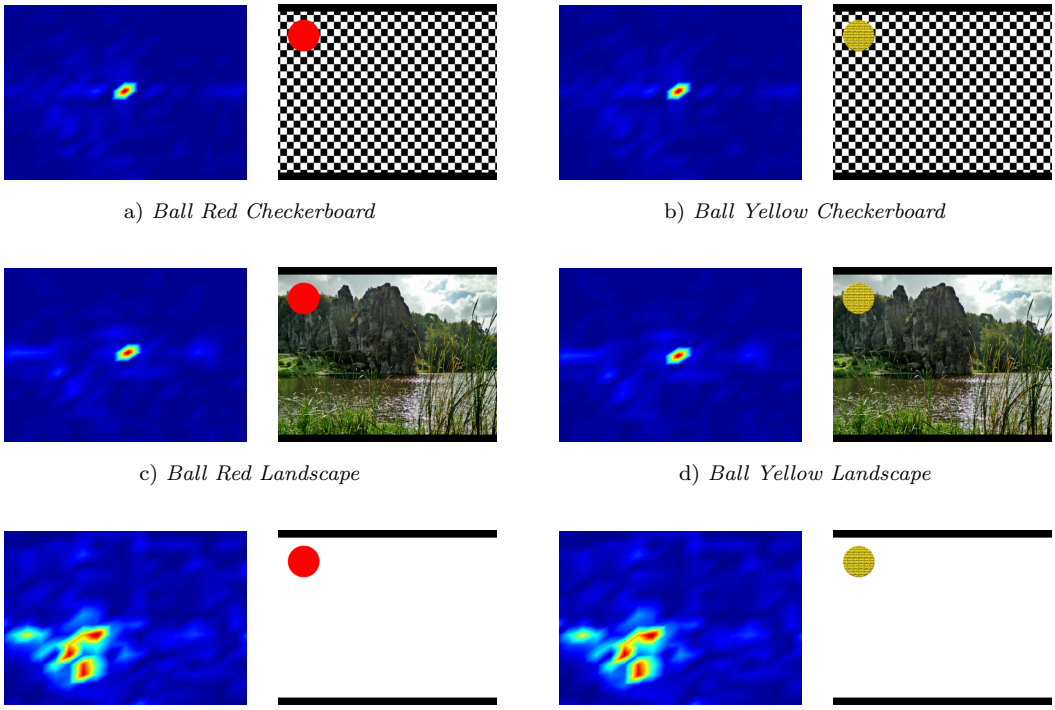
Dieses Kapitel wertet die Versuchsergebnisse aus. Auf Grund der hohen Anzahl an einstellbaren Parametern wird keine objektive Fehlerberechnung aller Sequenzen mit allen Einstellmöglichkeiten im Vergleich zu den Originalbildern durchgeführt, sondern für die verschiedenen Einstellmöglichkeiten eine Detailanalyse erarbeitet. Hierbei werden Szenarien aufgezeigt, in denen die Algorithmen wie beabsichtigt arbeiten. Darüber hinaus geht es aber insbesondere um die Analyse der Fälle, in denen die Berechnung nicht wie gewünscht funktioniert.

11.1 Phasenkorrelation

Eine herausragende Eigenschaft der Phasenkorrelation ist die Präzision, mit der Bewegungen erkannt werden können (vgl. Kapitel 5). Auch bei dem hier entwickelten Programm bestätigen sich diese Erfahrungen. Da es beim *Deinterlacing* enorm wichtig ist, dass die Objekte in beiden Halbbildern bei der Verschachtelung an der selben Stelle sind, soll Abbildung 11.1 an dieser Stelle als Beispiel angeführt werden. Die Kante weist keine Sägezahneffekte auf, obwohl sie aus zwei verschiedenphasigen Halbbildern besteht. Die Szene wurde mit dem *MC Field Insertion* Verfahren berechnet. Um keinen falschen Eindruck durch die Hintergrundbearbeitung zu bekommen, wurde die Einstellung *Zero* gewählt. Die Präzision der Vektoren wird deutlich.

Abbildung 11.1
Präzision der Bewegungsvektoren

Ein Problem des Verfahrens zeigt sich allerdings bei der Verwendung der ATS Sequenzen. Es ist für jede der Testsequenzen die Phasenkorrelationsebene dargestellt. Für die Sequenzen 11.3e und 11.3f wird Bewegungen erkannt. Es handelt sich um den niederfrequenten und den hochfrequenten Ball vor weißem niederfrequenten Hintergrund.
Das Problem wird bei der Betrachtung der anderen Sequenzen deutlich. Sobald der Hintergrund einer Szene detailliert ist und einen großen Anteil der Frequenzen des Bildes enthält, erkennt die Phasenkorrelation eine vordergründige Bewegung nur noch schwer. Es zeigen sich in den Bildern 11.3a-11.3d hellblaue Schattierungen in den Bereichen, in denen Bewegung stattfindet, allerdings auch in Bereichen, in denen keine Bewegung stattfindet. Der Signal-Rausch-Abstand sinkt in diesen Fällen extrem.
Dieses Phänomen lässt sich durch den hohen Anteil an Frequenzen erklären, deren Phase sich zwischen den Bildern nicht ändert. Die Höhe der *Peaks* in der Phasenkorrelationsebene entspricht dem Verhältnis von unverändertem zu verändertem Bildinhalt (siehe Kapitel 5).



a) *Ball Red Checkerboard* b) *Ball Yellow Checkerboard*

c) *Ball Red Landscape* d) *Ball Yellow Landscape*

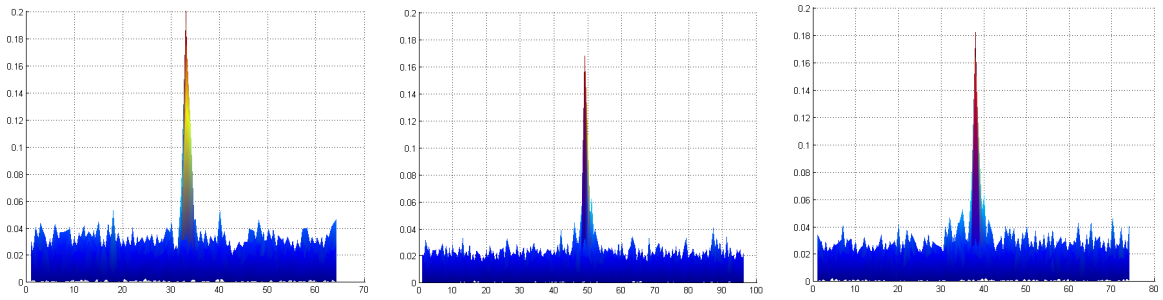
e) *Ball Red White* f) *Ball Yellow White*

Abbildung 11.3
Bewegungserkennung durch Phasenkorrelation

Abbildung 11.2
Park Joy

Diese Extremsituationen zeigen die Grenzen der Phasenkorrelation auf. Um dennoch Ergebnisse zu bekommen ist eine Erhöhung der `ME_PC_calculated_vectors` möglich. Jedoch werden dadurch zufällige Vektoren (Rauschen oder tatsächliche Verschiebung) aus der Phasenkorrelationsebene extrahiert. Der passende Vektor ist nur zufällig dabei. Die ursprüngliche Stärke der Phasenkorrelation, nämlich die Rechenzeit durch eine begrenzte Anzahl infrage kommender Vektoren zu begrenzen, verfällt mit dieser Lösung und das Verfahren käme einem *Brute Force Blockmatching* gleich. Auch auf Grund der Rechenzeit ist diese Alternative nicht sinnvoll.

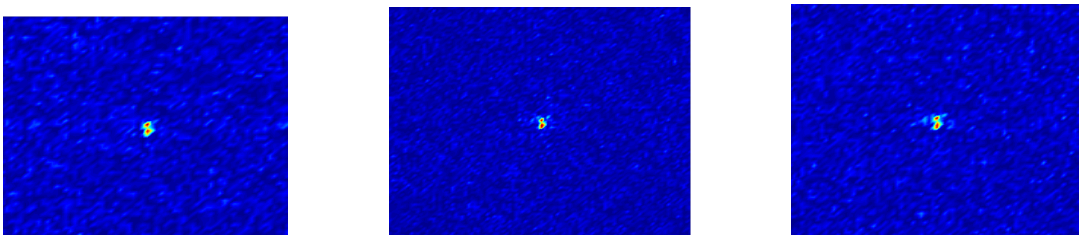
Diese Probleme der Phasenkorrelation treten in den beschriebenen Extremsituationen auf. Insbesondere die Szene *Park Joy* in Abbildung 11.2 ist problematisch. Die Bewegungen der Personen im Hintergrund werden einwandfrei erkannt. Die Bewegungen der Bäume im Vordergrund ergeben jedoch keinen scharfen *Peak* in der Korrelationsebene.



Ohne Fensterfunktion
XZ View

Hamming Window
XZ View

Decaying Extension Window
XZ View



Ohne Fensterfunktion
XY View

Hamming Window
XY View

Decaying Extension Window
XY View

Abbildung 11.4
Übersicht über die Phasenkorrelationsebenen ohne und mit Filterung der Subblöcke

11.1.1 Windowing

Das *Windowing* soll den Signal-Rausch Abstand bei der Phasenkorrelation erhöhen. Abbildung 11.4 zeigt die Phasenkorrelationsebenen für die Subblöcke aus 10.3 jeweils in der XZ- und XY-Ebene. Zunächst wird bei einem Vergleich der XZ-Ebenen der Einfluss der Fensterfunktionen deutlich. Die ungefilterte Ebene weist den höchsten *Peak* auf, jedoch auch den größten Rauschanteil. Das *Hamming*-Fenster vermindert das Rauschen deutlich, schwächt jedoch dabei auch die Höhe des *Peaks* ab. Ein Vergleich mit der XY-Ebene zeigt allerdings, dass es sich nicht um eine einfache Senkung der jeweiligen Werte handelt, sondern dass durch die Anwendung des *Hamming*-Fensters *Peaks* deutlicher werden. In der XZ-Ebene hat die *Decaying Extension* Fensterung einen kleineren Rauschanteil als die Ebene ohne Fensterung. Der *Peak* ist jedoch immer noch deutlich höher als bei dem *Hamming-Window*. Bei der Betrachtung der XY-Ebene fällt auf, dass noch weitere *Peaks* deutlicher zu differenzieren sind.
Die Fensterung bewirkt demnach den gewünschten Effekt. Rauschen wird durch die Vorverarbeitung abgeschwächt und das Nutzsignal ist besser zu identifizieren.

11.1.2 Blockmatching

Das *Blockmatching* ist eine zentrale Aufgabe bei der Bewegungsschätzung. In Grafik 11.5 werden die Auswirkungen der verschiedenen Verfahren deutlich. Einfaches *Blockmatching* hat ein inkonsistentes Vektorfeld zur Folge, das in vielen Bereichen nicht der wahren Bewegung im Bild entspricht. Die Pixel werden an die Positionen verschoben, die von den Farbwerten am besten zu ihnen passen. Diese Entscheidungen werden erheblich von dem im Bild vorhandenen Rauschen beeinflusst.

Um den Einfluss des Rauschens zu minimieren betrachtet das Verfahren *Blockmatching Surrounding* den *Error* nicht nur für jedes einzelne Pixel, sondern auch für die umliegenden Pixel. Somit wird der Einfluss des Rausches minimiert, da der gesamte Block im neuen Bild gleich sein muss. Den Einfluss sieht man deutlich in den Rasenanteilen im Bild 11.5b. Einige Elemente besitzen keine Bewegung, die im Bild 11.5a noch verschoben wurden.

Die Vorfilterung des *Prescreenings* wurde in 10.1.4 erklärt. Da zunächst anhand der Umgebung falsche Vektoren vor dem *Blockmatching* eliminiert werden, entspricht das Ergebnis deutlich mehr der *True Motion* als beim *Simple Blockmatching*. Es wird jedoch auch deutlich, dass die Rasenanteile in der Mitte wieder mit einem Bewegungsvektor versehen werden (11.5c).

Im Bild 11.5d wird die Kombination aus *Blockmatching Surrounding* und *Blockmatching Prescreening* gezeigt. In diesem Bild haben lediglich die sich bewegenden Personen Bewegungsvektoren. Der statische Rasen im Hintergrund wird nicht verschoben.

a) *Simple Blockmatching*

b) *Blockmatching Surrounding*

c) *Blockmatching Prescreening*

d) *Blockmatching Surrounding and Prescreening*

e) *Blockmatching Surrounding, Prescreening and Smoothing*

Abbildung 11.5
Vergleich der *Blockmatching* Verfahren

Es fallen einzelne Bewegungsvektoren mit großen Beträgen auf. Um diese zu eliminieren kann auf das entstandene Vektorfeld ein *Smoothing* angewendet werden. Dieses wird im nächsten Abschnitt behandelt.
11.6a zeigt einen Bildausschnitt, der mit dem einfachen *Blockmatching* auf Pixelbasis berechnet wurde. Bei 11.6b wurden die Zusatzfunktionen *Blockmatching Surrounding* und *Blockmatching Prescreening* benutzt. Das Ergebnis ist deutlich realistischer. Insbesondere die Zahl des Läufers ist auf Bild 11.6a nicht zu lesen, im Bild 11.6b jedoch deutlich erkennbar.
Innerhalb des *Media Converters* wird keine Sonderbehandlung von Randvektoren vorgenommen. Um die Qualität der äußeren Bildbereiche zu verbessern wäre dieses notwendig. Für den Vergleich der verschiedenen Verfahren ist eine Qualitätsminderung am Rand jedoch unerheblich. Aus diesem Grund wurde darauf verzichtet.

a) *Simple Blockmatching*

b) *Blockmatching Surrounding and Prescreening*

c) *Blockmatching Surrounding, Prescreening and Smoothing*

Abbildung 11.6
Auswirkung der *Blockmatching* Verfahren

11.1.3 Smoothing

Die Ergebnisse des *Smoothing* sind in Abbildung 11.5e zu sehen. Das Vektorfeld wirkt deutlich homogener als noch in 11.5d. In Abbildung 11.6c ist außerdem deutlich sichtbar, dass das Smoothing weitere falsche Vektoren eliminiert hat und somit die Qualität weiter gestiegen ist. Die Zahl ist sehr gut lesbar.
Bei den Versuchen hat sich gezeigt, dass das nachträgliche *Smoothing* beim *Upscaling* eine deutliche Qualitätsverbesserung bewirkt. Beim *Deinterlacing* ist das Gegenteil der Fall. Abbildung 11.7a zeigt einen Ausschnitt eines Bildes, bei dem die *Smoothing* Funktion während der *Motion Estimation* angewendet wurde, Abbildung 11.7b zeigt das selbe Bild ohne die *Smoothing* Funktion. In Bild 11.7a ist ein störender Sägezahneffekt sichtbar.

Das *Smoothing* legt einen Filterkernel auf einen Vektor, durch den dieser von seinen Nachbarpixeln beeinflusst wird. Einzelne falsche Vektoren werden somit gegen richtige ausgetauscht. An den Kanten bewegter Objekte ist das Vektorfeld allerdings ebenfalls inhomogen. Es ändert sich innerhalb weniger Pixel drastisch. Diese Inhomogenität entspricht einer Beschreibung der Realität. Durch den *Smoothing*-Filter werden jedoch auch diese Vektoren mit ihren Nachbarn verrechnet. Somit werden Kantenvektoren in der Regel kürzer, da ihre benachbarten Hintergrundvektoren oftmals wenig Bewegung aufweisen. Für das *Deinterlacing* müssen die Objekte genau an die selbe Position wie im anderen Halbbild verschoben werden. Durch die nun kürzeren Vektoren ist diese Bedingung nicht mehr erfüllt. Es tritt der Sägezahneffekt auf. Das Phänomen der kürzeren, bzw. längeren Vektoren äußert sich beim *Deinterlacing* in störenden Artefakten. Beim *Upscaling* tritt der Effekt in selber Weise auf, bewirkt jedoch keine visuellen Störungen wie Abbildung 11.6c beweist.

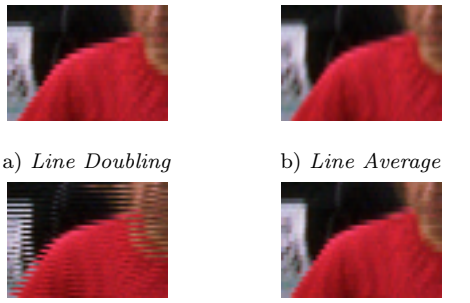
a) *Deinterlacing* mit *Smoothing*

b) *Deinterlacing* ohne *Smoothing*

Abbildung 11.7
Smoothing beim *Deinterlacing*

11.2 Deinterlacing

Abbildung 11.8 zeigt zu Vergleichszwecken die Ergebnisse des *Deinterlacings* mit den *non-Motion Compensating* Algorithmen. Die Artefakte zeigen, warum ein *Motion-Compensating* Ansatz notwendig ist. Das *Line Doubling* zeigt ein deutliches Treppenmuster an diagonalen Kanten. *Line Average* lässt das Bild unscharf wirken. *Field Repetition* führt insbesondere bei viel Bewegung zu einem Sägezahnmuster an Kanten und *Vertical Temporal Median* lässt die Zeilenstruktur ähnlich dem *Line Doubling* deutlicher werden.

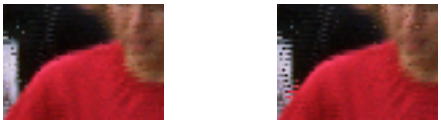


a) *Line Doubling* b) *Line Average*

c) *Field Repetition* d) *Vertical Temporal Median*

Abbildung 11.8
Ergebnisse der *non-Motion Compensating Deinterlacing*-Verfahren

Dem stehen in Abbildung 11.9 die Ergebnisse der *Motion-Compensated* Verfahren gegenüber. In allen vier bewegungskompensierten Verfahren ist die Qualität der Kante besser. Die Bewegungskompensation steigert den globalen Schärfeeindruck.
Insbesondere bei den Verfahren *MC Field Insertion* und *MC Field Average* können lokale, sehr störende Artefakte entstehen. Abbildung 11.10 zeigt die Artefakte für das *MC Field Insertion* Verfahren. Es wird deutlich, dass einige Pixel falsche Werte annehmen. In diesen Fällen sind die zu Grunde liegenden Bewegungsvektoren falsch. Wie bereits im Abschnitt 11.1.3 beschrieben, ist eine weitere Glättung des Vektorfeldes durch eine *Smoothing*-Funktion nicht möglich.



a) *MC Field Insertion* b) *MC Field Average*

c) *MC Median Insertion* d) *MC Median Average*

Abbildung 11.9
Ergebnisse der *Motion Compensating Deinterlacing*-Verfahren

Obwohl das Verfahren *MC Field Average* für jedes Pixel einen Mittelwert bildet, sind die visuellen Ergebnisse bei dieser Umsetzung noch fehlerhafter (vgl. Abbildung 11.11). Zusätzlich zu den falschen Vektoren ist als Grund anzuführen, dass die Phasenkorrelation auf Integerbasis arbeitet. Die zwei Vektoren beim *MC Average Verfahren* zeigen aus den zwei Richtungen nicht immer auf das selbe Pixel, sondern adressieren in einigen Fällen durch Rundung und Integerlängen nur zwei benachbarte Pixel.
Das *MC Median* Verfahren wurde entwickelt, damit die falschen Bewegungsvektoren keine störenden Artefakte mehr im Bild hinterlassen (vgl. Abschnitt 6.2.1). Es wird eine Nachverarbeitung angehängt, die die Pixel mit einem falschen Vektor gegen das Ergebnis einer Bearbeitung ohne *Motion Estimation* ersetzt. Die Ergebnisse in 11.9c und 11.9d zeigen eine deutliche Verbesserung. Auch die in Abbildung 11.12 dargestellen Artefakte stören den visuellen Gesamteindruck nur gering. An Kanten kann es vorkommen, dass der falsche Pixel aus

Abbildung 11.10
Artefakte beim *MC Field Insertion*

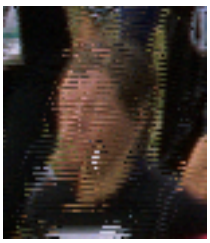


Abbildung 11.11
Artefakte beim *MC Field Average*

a) *MC Median Insertion* b) *MC Median Average*

Abbildung 11.12
Artefakte bei den *MC Median* Verfahren

a) Original b) Upscaled

Abbildung 11.14
Crowd Run Upscaling

der Auswahl von Hintergrund, Vordergrund und Bewegungskompensation gewählt wird. Da die beiden Verfahren auf den Ergebnissen des *MC Field Insertion* bzw. *MC Field Average* aufbauen, ist auch in diesem Fall das *MC Median Average* etwas schlechter als das *MC Median Insertion*. Da die Bewegungsvektoren beim *MC Field Average* an mehreren Stellen falsch sind, müssen mehr Pixel durch den Mittelwert aus den Pixeln mit der selben horizontalen Positon der benachbarten Zeilen ersetzt werden. Das Bild ist somit etwas unschärfer als das mit dem *MC Median Insertion* berechnete Bild.

11.2.1 Deinterlacing Background

Bei der Bearbeitung kann es immer wieder vorkommen, dass Pixeln kein Vektor zugewiesen wird. Entweder kann es sich um *uncovered* oder *obscured Background* handeln oder aber um Fehler bei der *Motion Estimation* oder beim *Blockmatching*. In Grafik 11.13 beispielsweise haben einige Pixel der unteren Pixelreihe der schwarzen Hose keinen Bewegungsvektor zugewiesen bekommen. Bei der Funktion *Following Field* wird diesen Pixeln der Wert des nächsten Halbbildes selben Typs zugeordnet. Auf Grund der Bewegung nach unten sind diese Pixel schwarz. Bei der Option *Subsequent Average* wird ein Mittelwert aus den Pixeln an der selben horizontalen Position der oberen und der unteren Zeile gebildet. Der Unterschied ist in Grafik 11.13 sichtbar, jedoch sind beide Optionen visuell qualitativ gleich.

a) Following Field b) Subsequent Average

Abbildung 11.13
Behandlung von Pixeln ohne zugewiesenen Vektor

11.3 Upscaling

In Abbildung 11.14 wird ein interpoliertes Bild mit dem Original verglichen. Für die Bearbeitung wurde der *MC Field Insertion* Algorithmus angewendet. Es wird deutlich sichtbar, dass die Bewegung der einzelnen Objekte erkannt und angewendet wurde, ohne dass visuelle Qualität verloren gegangen ist. Global arbeitet das Verfahren wie beabsichtigt. Kleinere lokale Probleme treten an den Kanten im Bild auf. Für die visuelle Qualität bedeutet dieses, dass Bewegungen flüssig dargestellt werden. Lediglich die Detailtiefe bzw. die Schärfe ist nicht optimal. Dieses und auch weitere Probleme werden im Folgenden analysiert.

Im Abschnitt 11.1 dieser Auswertung wurde auf ein großes Problem der Phasenkorrelation hingewiesen. In Bildausschnitten, bei denen sehr detailreicher Hintergrund vorhanden ist, wird eine vordergründige Bewegung eines anderen Objektes nicht zufriedenstellend erkannt. Dieses Problem ergibt sich insbesondere bei der Testsequenz *Park Joy*. Abbildung 11.15 zeigt zwei Bildausschnitte der *Park Joy* Szene, die mit dem *MC Frame Insertion* Verfahren interpoliert wurden. Als Hintergrund wurde *Zero* verwendet, damit allein die Ergebnisse der Phasenkorrelation deutlich werden. In 11.15a sieht man den qualitativ zufriedenstellend berechneten Hintergrund. Die Bewegungen der Frau sind erfasst worden, sodass keine Kanteneffekte entstehen. Die freien Stellen können gut über die Hintergrundbehandlung gefüllt werden. Im Gegensatz dazu steht die in 11.15b dargestellte Berechnung der unscharfen Baumstämme im Vordergrund. Das Verfahren hat die Bewegung nicht sauber erkannt. Die schwarzen Stellen, die durch die Bewegung der Baumstämme entstehen und richtig sind, werden von vielen nicht schwarzen Pixeln durchzogen. Da diese durch die Hintergrundbearbeitung nicht weiter verändert werden, ergeben sich Artefakte.

a) detailreicher Hintergrund b) unscharfer Vordergrund

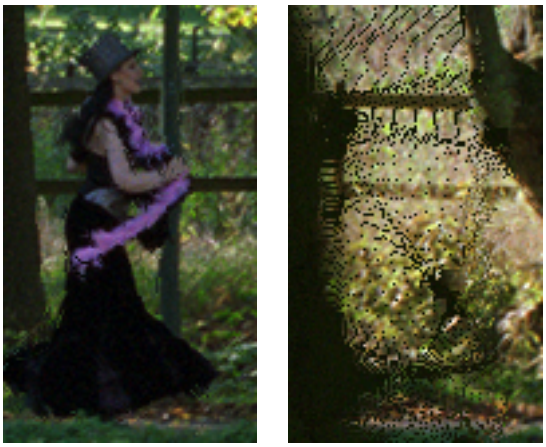
Abbildung 11.15
Probleme der Phasenkorrelation

11.3.1 Upscaling Background

Abbildung 11.16 und 11.17 zeigen die zwei möglichen implementierten Hintergrundbehandlungen. Es wird deutlich, dass das *Following Field* Verfahren Schwächen bei der Behandlung der Personenvorlagen (11.16a) aufweist. Es treten häufig Pixel mit falschen Werten auf. Dieses Phänomen lässt sich durch die Tatsache erklären, dass alle nicht bewegungskompensierten Pixel durch den Wert des Pixels an selber Stelle aus dem nächsten

vorhandenen Vollbild ersetzt werden. Da in diesem die Bewegung jedoch weiter fortgeschritten ist, passen einige Pixelwerte nicht.

Für das *Subsequent Average Edge Detection* Verfahren stellt sich die Situation anders dar. Nicht bewegungskompensierte Pixel werden aus den umliegenden Werten errechnet. Es treten keine störenden falschen Pixel in 11.17a auf.
Doch der Vorteil, den das *Subsequent Average Edge Detection* Verfahren bei den vergleichsweise kleinen Bewegungen des Hintergrundes besitzt, kann bei den großen Bewegungen des Vordergrundes nicht genutzt werden. 11.17b wirkt verwischt, unscharf und die Kante ist nicht mehr als solche zu erkennen. Dieses Phänomen lässt sich dadurch erklären, dass große schwarze Flächen vorhanden sind. Bei der Bearbeitung wird das Bild von oben links nach unten rechts zeilenweise durchlaufen. Gelangt der Algorithmus an eine Stelle, an der große schwarze Flächen vorhanden sind haben die Pixel teilweise nur die Möglichkeit, aus den Werten links und oben einen Durchschnitt zu berechnen. Der Einflussbereich ist zu gering und enthält unter Umständen bereits falsche Werte.



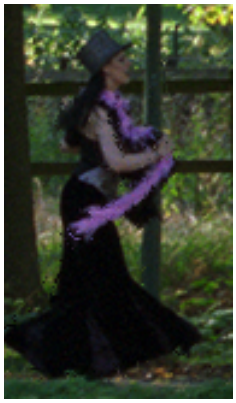
a) Hintergrund b) Vordergrund

Abbildung 11.16
Hintergrundbehandlung
Upscaling Following Field

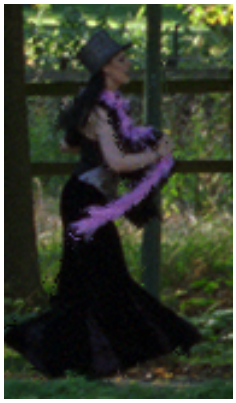
Für diese Bildbereiche ist der *Following Field* Algorithmus besser geeignet. Bei den großen Flächen fällt eine kleine Verschiebungsungenauigkeit nicht auf. Da die Pixel des nächsten Bildes genommen werden, wirkt der ersetzte Hintergrund scharf. Die braunen Pixel entstehen durch die oben bereits beschriebenen Probleme bei der Bewegungsschätzung. Sie sind kein Artefakt aus den Hintergrundbehandlungen.
Je nach Situation liefert eine andere Hintergrundbearbeitung die besseren Ergebnisse. Allgemein lässt sich feststellen, dass die *Subsequent Average Edge Detection* bei vielen kleinen Bewegungen sehr gute Ergebnisse liefert, die *Following Field* Herangehensweise bei großen Verschiebungen im Bild die bessere Wahl ist.

a) Hintergrund b) Vordergrund

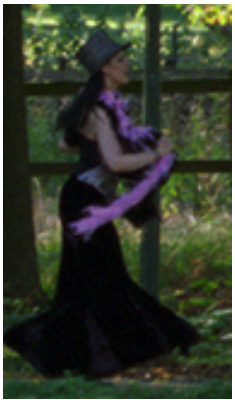
Abbildung 11.17
Hintergrundbehandlung *Upscaling*
Subsequent Average Edge Detection



a) *Frame Average* Hintergrund

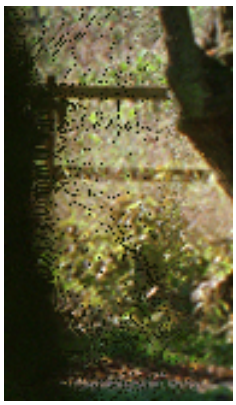


c) *Static Median* Hintergrund

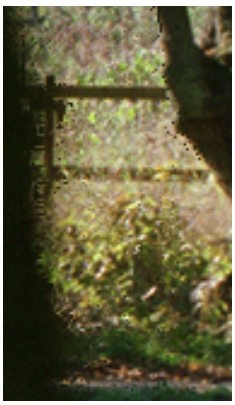


e) *Dynamic Median* Hintergrund

b) *Frame Average* Vordergrund



d) *Static Median* Vordergrund



f) *Dynamic Median* Vordergrund

Abbildung 11.19
Weitere *Upscaling* Verfahren (*Following Field*)

Auf Grund der Arbeitsweise der *Subsequent Average Field Detection* kann es bei der *MC Field Average* Methode zu Problemen kommen. Es werden lediglich Mittelwerte von Pixeln gebildet, die schwarz sind. Da der Algorithmus nicht auf Subpixelbasis arbeitet, können abgedunkelte Pixel im Bild entstehen, die jedoch nicht ganz schwarz sind und aus diesem Grund nicht weiterverarbeitet werden. Abbildung 11.18 zeigt das Problem.

Abbildung 11.18
Artefakte beim *Subsequent Average Edge Detection*

11.4 Weitere Verfahren

Durch die weiteren Verfahren kann die Qualität des Ausgangsbildes verbessert werden (vgl. Abbildung 11.19). *MC Dynamic Median* verringert die Anzahl der falschen Pixel in 11.19b gegenüber 11.16b drastisch. Zwischen *MC Frame Average* und *MC Static Median* liegt visuell kein Unterschied. Es ist darüber hinaus auffällig, dass auch die feinen Hintergrundbewegungen in 11.19e keine störenden Artefakte mehr aufweisen.

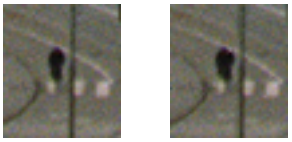
Für die Szene *Park Joy* ist die *Upscaling* Methode *MC Dynamic Median* in Kombination mit einer *Following Field* Hintergrundbearbeitung die beste Wahl. Wendet man diese Kombination jedoch auf die Szene *Crowd Run* an, so ist das visuelle Ergebnis im Vergleich zu einem *MC Field Insertion* Ansatz mit *Subsequent Average Edge Detection* Hintergrundbearbeitung schlechter (siehe Grafik 11.20). Die Algorithmen funktionieren demnach je nach Bildinhalt unterschiedlich gut.

a) *Dynamic Median, Following Field* b) *Field Insertion, Subsequent Average Edge Detection*

Abbildung 11.20
Crowd Run Upscaling

11.5 Integer Verschiebungen

Alle angewendeten Bewegungsvektoren basieren im *Media Converter* auf Integerlängen. Beim Upscaling wird aus diesem Grund an einigen Stellen gerundet. Bei sehr langsamen Bewegungen in einer Szene kann der in Abbildung 11.21 dargestellte Effekt auftreten. Es werden zwei Ausschnitte aufeinanderfolgender Bilder einer interpolierten Sequenz gezeigt. Die Person besitzt eine Bewegung, die so klein ist, dass sie für einige Pixel aufgerundet, für andere Pixel abgerundet wird. Der linke Teil der Person bleibt im berechneten Bild an selber Stelle, der rechte Teil verschiebt sich um einen Pixel. Aus diesem Grund wirkt die Person in 11.21b auseinandergezogen.



a) *Frame n* b) *Frame n + 1*

Abbildung 11.21
Integer Verschiebungen

12 Zusammenfassung und Ausblick

In dieser Arbeit wurden Verfahren zum *Deinterlacing* und *Upscaling* von Videosequenzen vorgestellt. Nach einer theoretischen Einführung wurden schließlich einige der Verfahren in einem MATLAB Programm umgesetzt mit dem Ziel, eine HD-Videosequenz unter speziellen Gesichtspunkten (*Scan Modus*, *Frame Rate*) an den im August 2012 vorgestellten Standard Ultra-HD [12] anzupassen.
In der Auswertung der in dieser Arbeit vorgestellen Algorithmen (Kapitel 11) wird deutlich, dass die erreichte Qualität an einigen Stellen durchaus zufriedenstellend ist, an anderen Stellen aber auch Verbesserungspotential aufweist. Um den Rahmen dieser Arbeit nicht zu überschreiten, wurden nicht alle im theoretischen Teil der Arbeit vorgestellten Verfahren praktisch umgesetzt. Insbesondere wurden *Deinterlacing*-Methoden nicht implementiert, deren Ergebnisse im Vergleich interessant wären.
Eine wesentliche Erkenntnis lässt sich dennoch aus den umgesetzten Algorithmen schließen. Die Qualität der gesamten Bearbeitung hängt zu einem sehr großen Anteil von der Genauigkeit der Bewegungsvektoren ab. Die Bewegungsschätzung ist essentiell für die Arbeitsweise der *Deinterlacing* und *Upscaling* Algorithmen, wobei auch die einfacheren dieser Algorithmen gute Ergebnisse liefern können, wenn die Bewegungsvektoren tatsächlich die *True Motion* beschreiben. Umgekehrt lässt sich der Einfluss falscher Vektoren durch das Design einiger *Deinterlacing* und *Upscaling* Algorithmen zwar minimieren, das Resultat ist an diesen Stellen aber nicht besser als das einer Berechnung ohne Bewegungsschätzung.
In der Auswertung (Kapitel 11) wird deutlich, dass die Anpassung der Parameter für die verschiedenen Algorithmen einen großen Effekt auf deren Ergebnisse haben. Je nach Situation führen andere Werte zu besseren Ergebnissen. Durch eine Untersuchung, in welcher Szene welche Variable die besten Ergebnisse liefert ließe sich, verbunden mit einer automatischen Wertzuweisung, die Robustheit und die Genauigkeit des *Media Converters* deutlich steigern. Insbesondere durch eine szenenangepasste Zuweisung der unterschiedlichen *Background*-Operationen lässt sich mit Blick auf Kapitel 11 eine Verbesserung erzielen.
Die Phasenkorrelation liefert bemerkenswert genaue Ergebnisse - wenn sie eine Bewegung erkannt hat. Es gab einige Szenarien in denen globale Bewegungen die kleineren lokalen Bewegungen im Ergebnis überdeckt haben. Um diesem Phänomen vorzubeugen, ist eine hierarchische Herangehensweise eine Lösung. Nachdem die Phasenkorrelation für einen großen Makroblock (128×128 oder 64×64 Pixel) durchgeführt wurde, wird dieser wiederum in kleiner Blöcke unterteilt für die eine weitere Phasenkorrelation durchgeführt wird um lokale Bewegungen zu erfassen. Außerdem hat eine Erweiterung des Verfahrens auf Subpixelgenauigkeit der Bewegungsvektoren vermutlich eine enorme Qualitätssteigerung zur Folge, da viele Rundungsfehler keine falschen Pixelwerte produzieren.

Aus diesem Grund sind bei dem in dieser Arbeit entwickelten *Media Converter* die Ergebnisse der *Insertion*-Verfahren besser als die der *Average*-Verfahren. Durch die Umstellung auf Subpixelgenauigkeit ändert sich diese Tatsache. [9] zeigt ein Herangehensweise für diese Erweiterung auf.
Gonzalez [10] schlägt eine noch umfangreichere Vorverarbeitung vor. Das Spektrum wird in Polarkoordinaten transformiert, nachdem ein *Hamming-Window* auf die Blöcke angewendet und die Fouriertransformation durchgeführt wurde. Durch diesen Schritt können Rotationsbewegungen im Bild besser erfasst werden. Den Untersuchungen zu Folge wird die Genauigkeit durch ein *Spectral Whitening* und einen *High Pass* weiter erhöht. Ein völlig anderer Ansatz beschäftigt sich mit der Berechnung eines *Markov-Random Fields* (MRF) [14, 4]. Er basiert auf der Annahme, dass der Zustand eines jeden Pixels lediglich auf dem Zustand der umliegenden Pixel basiert. Diese Verfahren ergeben ebenfalls qualitativ hochwertige Ergebnisse und werden deshalb an dieser Stelle erwähnt. Darüber hinaus kann ein MRF Ansatz auch für das *Deinterlacing* genutzt werden [15].

Das in Kapitel 3 gesetzte Ziel ein *Deinterlacing* und anschließendes *Upscaling* für *High Definition* Video durchzuführen, wurde erreicht. Mit dem entwickelten Programm *Media Converter* ist es möglich, ein Eingangsvideosignal so zu bearbeiten, dass es im Hinblick auf den *Scan Mode* und die *Frame Rate* den Anforderungen eines Ultra-HD Signals entspricht. Für die Phasenkorrelation wurden verschiedene Erweiterungen implementiert, die sowohl eine Verbesserung bei der Bewegungserkennung als auch beim *Blockmatching* zur Folge haben. Es können acht *Deinterlacing*- und sechs *Upscaling*-Verfahren eingestellt werden. Alle Parameter sind veränderbar, sodass verschiedene Kombinationen untersucht werden können.
Die angewendeten Verfahren funktionieren in vielen Situationen. In einigen Szenarien gibt es Verbesserungspotential, wobei die Fehlerquelle innerhalb der benutzten Verfahren, nicht in der Umsetzung liegt. In Kapitel 11 wurden die Schwachstellen beschrieben, die das Seherlebnis stören. Mögliche Weiterentwicklungen wurden daraufhin aufgezeigt.
Letztendlich wird jedes Verfahren designt um in einer bestimmten Situation die bestmöglichen Ergebnisse zu liefern. Das im Endeffekt beste Verfahren ist vermutlich eine Kombination aus vielen verschiedenen Algorithmen, in der jeder einzelne seine Stärken ausspielen kann und seine Schwächen verdeckt werden.

Literatur

[1] J. Ahmed and M.-N. Jafri. *Improved Phase Correlation Matching.* pages 128–135, Image and Signal Processing - 3rd International Conference, 2008.

[2] E.B. Bellers and G. de Haan. *De-interlacing: A Key Technology for Scan Rate Conversion.* Elsevier, 2000.

[3] Matlab Central. *Why does the SDK 7.1 installation fail with an „Installation Failed“ message on my Windows system?* http://www.mathworks.com/matlabcentral/answers/95039-why-does-the-sdk-7-1-installation-fail-with-an-installation-failed-message-on-my-windows-system, 06.10.2014.

[4] S. Dai, S. Baker, and S.-B. Kang. *An MRF-Based DeInterlacing Algorithm With Exemplar-Based Refinement.* IEEE, 2009.

[5] G. de Haan and R. Braspenning. *The Digital Signal Processing Handbook - Second Edition, Video, Speech and Audio Signal Processing and Associated Standards, Chapter 16.* CRC Press, 2010.

[6] EBU. *High Definition (HD) Image Formats for Television Production.* Technical report, EBU-Tech 3299, 2010.

[7] EBU/SVT. *SVT High Definition Multi Format Test Set.* https://tech.ebu.ch/hdtv_test-sequences, 2014.

[8] E.W. Engstrom. *A Study of Television Image Characteristics: Part Two: Determination of Frame Frequency for Television in Terms of Flicker Characteristics.* pages 295–310, April Proceedings of the Institute of Radio Engineers, 1935.

[9] H. Foroosh, J.-B. Zerubia, and M. Berthod. *Extension of phase correlation to subpixel registration.* pages 188–200, IEEE Trans. on Image Processing, 2002.

[10] R. Gonzalez. *Improving Phase Correlation for Image Registration.* University of Auckland, 2011.

[11] Prof. Dr.-Ing. R. Hedtke. *HDTV...und was kommt danach?* FKT 4/2013.

[12] ITU. *Parameter values for Ultra-High Definition Television Systems for Production and international Programme Exchange.* Technical report, ITU-R BT.2020, 2012.

[13] C.D. Kuglin and D.C. Hines. *The Phase Correlation Image Alignment Method.* Proceedings of the IEEE on Cybernetics and Society, 1975.

[14] G.-G. Lee, M.-J. Lai R.-L. Lin, H.-Y. Wang, and C.-W. Jhu. *A high-quality spatial-temporal content-adaptive deinterlacing algorithm*. IEEE, 2008.

[15] M. Li and T. Nguyen. *A De-Interlacing Algorithm Using Markov Random Field Model*. IEEE, 2007.

[16] R Li, B Zheng, and M.-L. Liou. *Reliable motion detection/compensation for interlaced sequences and its applications to deinterlacing*. pages 23–29, IEEE Transactions on Circuits and Systems for Video Technology, 2000.

[17] Mathworks. *MATLAB and Simulink for Technical Computing - MathWorks Deutschland*. http://www.mathworks.de/, 06.10.2014.

[18] O.A Ojo and G. de Haan. *Robust motion-compensated video upconversion*. pages 1045–1056, IEEE Transactions on Consumer Electronics, 1997.

[19] Z. Pang, D. Lei, D. Chen, and H. Tan. *A motion adaptive deinterlacing method based on human visual system*. pages 340–344, 4th International Congress on Image and Signal Processing (CISP), 2011.

[20] Prof. Dr. U. Schmidt. *Professionelle Videotechnik*. Springer Vieweg, 2013.

[21] Sony. *4K Live Production System at FIFA Confederations Cup 2013*, http://www.sony.co.uk/res/attachment/file/95/1237491578795.pdf, 06.10.2014.

[22] H.-S. Stone, B. Tao, and M. McGuire. *Analysis of image registration noise due to rotationally dependent aliasing*. Journal of Visual Communication and Image Representation, 2003.

[23] H.S. Stone, M.T. Orchard, E.-C. Chang, and S.A Martucci. *A fast direct Fourier-based algorithm for subpixel registration of images*. pages 2235–2243, IEEE Transactions on Geoscience and Remote Sensing, 2001.

[24] Sveriges Television AB (SVT). *The SVT High Definition Multi Format Test Set - Version 1.0*. https://tech.ebu.ch/docs/hdtv/svt-multiformat-conditions-v10.pdf, Februar 2006.

[25] SWR2. *Zeitwort*. http://www.swr.de/swr2/programm/sendungen/zeitwort/-/id=10045946/property=download/nid=660694/fuzehl/swr2-zeitwort-20120825.pdf, 06.10.2014.

[26] Dipl.-Ing. Y. Thomas. *„Higher Frame Rates“(HFR) - ein neuer Standard*. FKT 11/2014.

[27] G.A. Thomas. *Television Motion Measurement For DATV and other Applications.* Technical report, BBC Research Department, 1987.

[28] A. Vogel and P. Effenberg. *Handbuch HD-Produktion.* IRT, 2013.

[29] H. Yan and J.-G. Liu. *Robust Phase Correlation based Motion Estimation and Its Applications.* pages 104.1–104.10, 2008.

[30] J. Yen. *On Nonuniform Sampling of Bandwidth-Limited Signals.* Technical report, IRE Transactions on Circuit Theory, 1956.

Eidesstattliche Erklärung

Ich versichere hiermit, die vorgelegte Arbeit in dem gemeldeten Zeitraum ohne fremde Hilfe verfasst und mich keiner anderen als der angegebenen Hilfsmittel und Quellen bedient zu haben.

Köln, den 6. Dezember 2014

(Daniel Schwingenheuer)