An improved algorithm for deinterlacing video streams

Christopher Weiss
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
AN IMPROVED ALGORITHM FOR DEINTERLACING VIDEO STREAMS

by

Christopher Weiss

Bachelor of Science
University of Nevada, Las Vegas
1997

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2006
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
The Thesis prepared by

Christopher Allen Weiss

Entitled
An Improved Algorithm for Deinterlacing Video Streams

is approved in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative
# TABLE OF CONTENTS

ABSTRACT ........................................................................................................................................... iv

LIST OF FIGURES .............................................................................................................................. v

ACKNOWLEDGEMENTS ....................................................................................................................... vi

CHAPTER 1  INTRODUCTION ........................................................................................................ 1
  Problems Arise With Interlacing ............................................................................................... 3

CHAPTER 2  METHODS OF MEASURE ..................................................................................... 5

CHAPTER 3  REVIEW OF PREVIOUS METHODS ......................................................................... 8
  Line Averaging ............................................................................................................................. 10
  Field Insertion ............................................................................................................................. 11
  Linear Vertical-Temporal Filtering ........................................................................................ 13
  Vertical-Temporal Median Filtering ...................................................................................... 14
  Edge-Based Line Averaging ...................................................................................................... 16
  Motion-Compensated Vertical-Temporal Median Filtering .............................................. 19

CHAPTER 4  NEW METHOD ........................................................................................................... 22
  Video Object Planes ...................................................................................................................... 22
  Difference Image Creation ......................................................................................................... 24
  Canny Edge Detection ................................................................................................................. 26
  Video Object Plane Bounding Box Extraction .................................................................... 31
  Video Object Plane Based Deinterlacing ................................................................................ 33

CHAPTER 5  COMPARATIVE RESULTS .......................................................................................... 39

CHAPTER 6  SUMMARY AND CONCLUSIONS ............................................................................. 42

APPENDIX  SOURCE CODE ........................................................................................................ on CD-ROM

REFERENCES ...................................................................................................................................... 44

VITA ...................................................................................................................................................... 46
ABSTRACT

An Improved Algorithm For Deinterlacing Video Streams

by

Christopher Weiss

Dr. Evangelos Yfantis, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

The MPEG-4 standard for computerized video incorporates the concept of a video object plane. While in the simplest case this can be the full rectangular frame, the standard supports a hierarchical set of arbitrary shaped planes, one for each content-sensitive video object. Herein is proposed a method for extracting arbitrary planes from video that does not already contain video object plane information.

Deinterlacing is the process of taking two video fields, each at half the height of the finalized image frame, and combining them into that finalized frame. As the fields are not captured simultaneously, temporal artifacts may result. Herein is proposed a method to use the above mentioned video object planes to calculate the intra-field motion of objects in the video stream and correct for such motion leading to a higher quality deinterlaced output.
| Figure 1 | General deinterlacing function ................................................................. | 8 |
| Figure 2 | Linear (non-motion compensation) deinterlacing ........................................ | 9 |
| Figure 3 | Line-averaged deinterlaced frame ............................................................... | 10 |
| Figure 4 | Artifacts introduced by field insertion ....................................................... | 12 |
| Figure 5 | Vertical-temporal filtering weights ............................................................. | 13 |
| Figure 6 | Vertical-temporal filtered image ................................................................ | 14 |
| Figure 7 | Vertical-temporal median filter .................................................................. | 15 |
| Figure 8 | Vertical-temporal 3-tap median filtering ..................................................... | 16 |
| Figure 9 | Edge line averaging edge detection ............................................................. | 17 |
| Figure 10 | Edge line averaging pixel interpolation ..................................................... | 17 |
| Figure 11 | Edge line averaged image ............................................................................. | 18 |
| Figure 12 | Block-based sum of absolute error .............................................................. | 19 |
| Figure 13 | Motion compensating vertical-temporal median filter ..................................... | 20 |
| Figure 14 | Motion compensated vertical-temporal median filtered frame ....................... | 21 |
| Figure 15 | Difference image function ........................................................................... | 24 |
| Figure 16 | Even field (n) .............................................................................................. | 25 |
| Figure 17 | The next even field (n+2) ............................................................................ | 25 |
| Figure 18 | Difference image of fields n and n+2 (border added) .................................... | 25 |
| Figure 19 | Horizontal derivative function ..................................................................... | 26 |
| Figure 20 | Vertical derivative function ......................................................................... | 26 |
| Figure 21 | Gradient magnitude function ....................................................................... | 26 |
| Figure 22 | Gradient direction function ......................................................................... | 27 |
| Figure 23 | Gradients of the difference image (gamma adjusted and border added) .......... | 27 |
| Figure 24 | Gradient direction estimation ....................................................................... | 28 |
| Figure 25 | Gradient magnitude estimation ..................................................................... | 28 |
| Figure 26 | Calculation of Pa and Pb ............................................................................ | 29 |
| Figure 27 | Non-maximal suppressed gradients (gamma adjusted and border added) ........ | 29 |
| Figure 28 | Gradient image, post-hysteresis (border added) .......................................... | 30 |
| Figure 29 | Bounding box capture distance .................................................................... | 32 |
| Figure 30 | Bounding box capture area .......................................................................... | 32 |
| Figure 31 | Video object plane bounding boxes (border added) ....................................... | 33 |
| Figure 32 | Motion compensated deinterlacing function ................................................. | 34 |
| Figure 33 | Even field edge image (border added) ........................................................... | 35 |
| Figure 34 | Odd field edge image (border added) ............................................................ | 36 |
| Figure 35 | Video object plane motion vector ............................................................... | 36 |
| Figure 36 | Moving-area video object plane deinterlace function .................................... | 37 |
| Figure 37 | Video object plane deinterlaced frame ....................................................... | 37 |
| Figure 38 | High-motion deinterlacing comparison ......................................................... | 39 |
| Figure 39 | Low-motion deinterlacing comparison .......................................................... | 40 |
| Figure 40 | Still frame test ............................................................................................. | 41 |
ACKNOWLEDGEMENTS

With special thanks to Dr. Evangelos Yfantis for his continued guidance and support throughout my academic career, Doctors Datta, Minor, and DuBose for their consideration of this thesis, and to my family for their support.
CHAPTER 1

INTRODUCTION

The earliest motion pictures had been exactly that: pictures. A series of still images displayed in rapid progression is treated by the human eyes and brain as one continuous, moving image. Even children have simulated this effect using a set of cards with a series of slightly changed images on each succeeding card. In the terminology of the field, such a video stream is called *progressive*, that is, each frame of video contains a complete picture. Such a system was completely preserved as the leap was made from paper flip books to motion picture photography, since each succeeding cell on a reel of film contains the next progressive image.

With the introduction of the television, however, a problem arose. In the physical world of the film reel, the amount of space occupied by the video is fixed by the size of the reel regardless of quality; once converted to a digital signal, however, the size of the transmission is directly related to both the size of the video and its quality.

Since the amount of radio frequency (RF) bandwidth was limited, and was shared with other competitors such as radio, various systems were proposed to achieve an acceptable level of signal quality while remaining within bandwidth constraints. Such systems were numerous and incompatible, so in 1940 the Federal Communications Commission established the National Television System Committee (NTSC) with the
goal of resolving such conflicts. In 1941 the NTSC standard for black and white television was released.

This standard required a resolution of 525 scan lines at 30 frames per second\(^1\). However, such data was not sent progressively, but rather interlaced; 262½ lines per field at 60 fields per second. Here a field is one half of the original image, created by taking either the even or odd numbered scan lines and extracting them.

The decision to interlace was not made at random. Each field consumes only one-half the bandwidth of a complete frame, however since twice as many of them have to be sent this would not seem to be an improvement. The improvement comes from the fields not being “extracted” from a series of progressive frames, but rather actually shot at 60 fields per second. That is, were a progressive series of images to be split into fields, then frame \(F\) shot at time \(T\) would be divided into fields \(f_1\) and \(f_2\), but both \(f_1\) and \(f_2\) would have been shot at time \(T\). Fields \(f_3\) and \(f_4\), taken from the next frame, would both reflect the camera’s capture \(1/30^{th}\) of a second later. When the original video is shot interlaced, however, frame \(F\) never truly exists. Rather, \(f_1\) and \(f_2\) are both shot as fields, and therefore \(f_2\) reflects the camera’s capture \(1/60^{th}\) of a second after \(f_1\). This can be seen as sacrificing spatial resolution for temporal resolution.

This change is subtle but important. For any two fields, the likelihood of a dramatic change in scene is small since they are shot only a small time apart; motion may occur but is likely to be localized. Since the human eye is more sensitive to large area flicker than small detail changes\(^2\), it is less likely to be distracted by the slight time-shifts between succeeding fields.
In addition, the construction of the cathode ray tube (CRT) based television made it a natural fit for interlaced video. In such units, a gun fires electrons at a layer of phosphorus, causing it to emit light briefly. By design, such emission is quite brief in common televisions, requiring the gun to repeatedly "redraw" the image on the phosphors over time. When given interlaced video, the television can draw the even field, followed by the odd field, and repeat. Thus, with correct phosphor timing the even field is fading as the odd field is drawn, and vice-versa, further minimizing the flicker introduced by interlacing. This system remained relatively unchallenged until the introduction of the computer, especially the personal computer, in the early 1980's.

Problems Arise With Interlacing

Until quite recently, the majority of video editing and manipulation tasks were performed using manual, non-digital methods. Once the computer became widely available and sufficiently powerful, however, the desire to use such processing power to edit images and later video streams became wide-spread. It is in this digital processing that the first problems arise with using interlaced video streams.

In a progressive video stream, any given frame can be extracted intact and manipulated. For an interlaced stream, however, any field contains only half the information necessary for processing. The process of combining two or more fields to create a single frame is called deinterlacing, and performing it introduces a host of additional complications into almost every video editing task.
To deinterlace a video stream that was created by the extracting of fields from a previously progressive stream is a trivial matter of simply interleaving the scan lines (this method, called "field insertion" is discussed in more depth below). However, the majority of video is recorded interlaced. Any given frame does not exist per se, and any method of expanding a given field or combining multiple fields can at best give only an approximation of what the frame between them might have held.

There is currently debate in both the television and PC arenas concerning whether, given the above state of affairs, interlacing should continue to be the standard for both broadcast television and computer video. The introduction of new technologies such as DVD video and high-definition television, along with higher compression rates for video streams, will continue to fuel such debates into the near future.

For already existing interlaced video, however, there are many reasons to perform deinterlacing. In addition to the above mentioned video editing, deinterlacing often comes to the forefront when a single, high quality still image is required to be extracted from a video stream. One common reason for such a still would be to produce a higher quality still image from a surveillance video camera which captured its video stream interlaced.

This paper is organized as follows: Chapter 2 details how the various historical deinterlacing algorithms will be objectively measured for comparison, and describes the test sets. Chapter 3 reviews six existing deinterlacing algorithms. Chapter 4 describes the new method proposed, and Chapter 5 provides a comparative analysis between the new method and the methods outlined in Chapter 3. Finally, Chapter 6 draws conclusions from the data in Chapter 5.
CHAPTER 2

METHODS OF MEASURE

In a standard video stream, there are generally five distinct types of motion, which can be present either separately or together. Each type of motion introduces its own problems for a deinterlacing algorithm. They are:

1) No motion. This occurs when the camera is itself stationary, and is observing purely stationary objects in fixed lighting.

2) Camera motion. Caused by the position of the camera moving, even as the objects being observed remain relatively static.

3) Object motion. The objects that are the target of the camera are in motion. The camera itself is held static.

4) Light motion. Not truly a type of motion, but rather a pseudo-motion introduced when the light source for the scene is altered (for example, by a cloud passing in front of the sun). This causes a relatively constant variance of pixel intensity across the field in the case of a single ambient light source change, but can cause localized variance in the case of a pinpoint light being used in the frame.

5) Scene Change. This occurs in a stream when two independent video streams are attached end to end. In this case the field at time T has no connection with the field at time T+1, and any valid combination of the two is impossible.
By 2006, there were more than 85 methods of performing deinterlacing. In addition to the standard analysis that would be applied to any computational algorithm (e.g., time complexity, space complexity, etc.), any deinterlacing algorithm would be tested against the above types of motion, and its performance measured against each.

There exist a wide variety of algorithms for comparing two images for similarity, and at first glance it would appear the best method for evaluating deinterlacing algorithms would be to apply the algorithm to the fields, and then compare the results to the original frame. The problem with this approach is that, in the case of video originally shot interlaced, the original frame does not exist. It is common, therefore, to create a simulated series of fields by first capturing a progressive video stream, and then rendering it down into a series of fields. While this method has the advantage of being able to create an objective measurement for a deinterlacing algorithm, it should be noted that such streams are not perfectly analogous to interlaced streams.

Once generated, the most commonly accepted means of comparing the output of deinterlacing algorithms is the mean squared error method (MSE). Marmolin suggested the use of a “Subjective MSE” algorithm for comparison, but De Haan and Bellers did not see significant differences between this approach and the standard MSE. De Haan suggested a different measure, motion trajectory inconsistency (MTI). MTI is a measurement based on the assumption that a perfect deinterlacing algorithm would generate the same frame regardless of whether the even or the odd pixels were interpolated. This compares favorably with television deinterlacing, in which an arbitrary frame may have been either even or odd first generated. While this measure has the advantage of being useable on streams that were originally shot interlaced, it still is of
limited value except as a general measure. De Haan noted that “Video quality still is a subjective matter, as it proves difficult to design a reliable objective measure reflecting the subjective impression. Although many attempts have been reported, none of these appears to be widely accepted.”

Herein we will use a simple measurement of the root mean squared error (RMSE) between the even and odd interpolated frames for a given pair of fields. This approach is severely limited, as will be discussed in chapter six, and therefore we will use it only as a preliminary ranking, and will concentrate more on photographs of generated frames to point out specific issues of subjective image quality.

Given the above, for each deinterlacing algorithm the following test cases are generated:

1) Still image testing. The even and odd fields from the same progressive image are given to the deinterlacing algorithm. This simulates the case where there is no motion of objects or camera. Field insertion will trivially have a 0 MSE for this case.

2) Even interpolation testing. The odd scan lines are taken as given, and the deinterlacer interpolates the even scan lines.

3) Odd interpolation testing. The even scan lines are taken as given, and the deinterlacer interpolates the odd scan lines.
CHAPTER 3

REVIEW OF PREVIOUS METHODS

The problem of a deinterlacer is: given a series of \( N \) input fields of height \( H \) as input, produces a series of \( N/2 \) frames of height \( 2H \) as output. This frame \( F_0 \) is usually formally defined as follows:

\[
F_x(x, y, n) = \begin{cases} 
F \left( \frac{x}{y}, n \right), (y \mod 2 = n \mod 2) \\
F_i \left( \frac{x}{y}, n \right), (otherwise)
\end{cases}
\]

Figure 1 - General deinterlacing function

Where \( x \) and \( y \) designate the spatial position of a given pixel, \( n \) is the field number, \( F \left( \frac{x}{y}, n \right) \) gives the input field where \( y \mod 2 = n \mod 2 \), and \( F_i \left( \frac{x}{y}, n \right) \) represents pixels generated by the deinterlacing interpolation. Such interpolation is performed using either spatial interpolation (using data within the same field), temporal interpolation (using data found in succeeding or preceding fields), or a combination of the two. All techniques currently popular fall into one of two categories, motion-compensated (MC) and non motion compensated.
De Haan and Bellers proposed that all linear, non motion compensated filters can be represented by the same function:

\[
F_n(\begin{bmatrix} x \\ y \end{bmatrix}, n) = \begin{cases} 
F(\begin{bmatrix} x \\ y \end{bmatrix}, n), (y \mod 2 = n \mod 2) \\
\sum_{k \in \{-1,0,1\}, (y + m \mod 2 = 1)} F(\begin{bmatrix} x \\ y \end{bmatrix}, k, n + m)h(k, m), \text{(otherwise)}
\end{cases}
\]

Figure 2 - Linear (non-motion compensation) deinterlacing

As this is the simplest function describing the linear, non-mc filters, it will be used herein as well.

The following methods were considered for comparison purposes (it should be noted that each of these methods except the last two were originally described in some detail by the De Haan and Bellers work referenced above):

1) Line Averaging (LA)
2) Field Insertion (FI)
3) Linear Vertical-Temporal Filtering (VT)
4) Vertical-Temporal Median Filtering (VT Median)
5) Edge-Based Line Averaging (ELA)
6) Motion-Compensated Vertical-Temporal Median Filtering (MC VT Median)
Line Averaging

Line averaging (here LA, often referred to as “Bob” in computer literature), is a purely spatial interpolation, using only the data present in one of the two fields in order to generate the interpolated pixels. In the simplest version of spatial interpolation, each line is simply repeated (equivalent to setting $h(k,0)=1$ for $k=1$, and $h(k,m)=0$ otherwise).

LA makes use of the high correlation between vertically related pixels to interpolate the intermediate pixels. Since the source data for all pixels is taken at the same time, any temporal artifacts are avoided.

![Figure 3 - Line-averaged deinterlaced frame](image-url)
Generally LA is implemented quite simply, with each interpolated pixel being the simple average of the pixels above and below it, that is, \( h(k,0) = 0.5 \) for \( k=-1,1 \) and \( h(k,m)=0 \) otherwise.

LA’s strength is its very low processing cost and ease of implementation. Its immunity to temporal artifacts makes it a strong performer as well in scenes where there is a very high degree of motion, as the blur effect introduced by the line average process actually appears normal to the human eye. LA fails worst on sequences with little or no motion: the loss of intrafield information leads to reduced resolution, but with no compensating reduction in motion artifacts.

In figure 3 motion artifacts are not present as line averaging is immune to temporal artifacting, but significant detail is lost around the windows, and vertical artifacts are introduced around diagonal lines present in the image (Image sequence taken from promotional clip of Pavilion of Women, 2001, Beijing Film Studio/Universal)

Field Insertion

While LA exploits the correlation between pixels vertically neighboring, field insertion (herein FI, commonly called “Weave” in computer literature) makes use of temporal correlation between pixels. Each field is staggered and “woven” together, that is, \( h(0,-1) = 1 \), and \( h(k,m)=0 \) otherwise.

FI, like LA, is very computationally simple and easy to implement. Since every pixel in a FI generated frame is actual data and not interpolated, it generates the highest quality results for stationary images (indeed, in the simulated data set for stationary images FI
generates a perfect image). However, in images with high degrees of motion, field insertion can result in a large degree of “tearing” temporal artifacts.

![Figure 4 - Artifacts introduced by field insertion](image)

While the static portions of the figure 4 are quite clear and sharp, the high motion plane objects are severely torn.
Linear Vertical-Temporal Filtering

Vertical-Temporal filtering (VT) uses data taken from both fields in a weighted average to interpolate the missing pixels in a frame. As the name suggests, only pixels that are neighbors either vertically (above or below), or temporally (in the other field) are considered in the average. While the best weights are subject to some debate, De Haan and Bellers recommended the following choices:

\[ h(k, m) = \begin{cases} 
1, 8, 8, 1, (k = -3, -1, 1, 3) \land (m = 0) \\
-5, 10, 5, (k = -2, 0, 2) \land (m = -1) \\
0 & \text{(otherwise)} 
\end{cases} \]

Figure 5 – Vertical-temporal filtering weights

VT filtering is only slightly more computationally complex than LA or FI, and has the advantage that it suffers less image degradation on still images than LA, and less temporal artifacts on high speed images than FI. Since data is only considered along the vertical band, objects moving horizontally with respect to the picture do not have artifacts introduced.

While it is true that information is lost by the VT filter if an object has vertical details or moves vertically, the VT filter is still considered a good choice when processing power is limited. Indeed, Seth-Smith and Walker claimed that a correctly defined VT filter can perform as well as a motion compensating filter, and with lower processing power.
Figure 6 – Vertical-temporal filtered image

While the static portions of figure 6 are left sharper than with the previous forms of filtering, artifacts are still visible around vertical details, and as shadows around the high-motion plane objects.

Vertical-Temporal Median Filtering

Median filtering, while non-linear, shares the performance characteristics of the linear filters due to its simple implementation. While a variety of median filters exist\textsuperscript{10,11}, here we will consider the simple three-tap median filter proposed by Annegarn et al\textsuperscript{12}. In it, the interpolated sample is the median value taken from its upper neighbor, lower neighbor, and temporal neighbor. Formally,
$F_o\begin{bmatrix}x \\ y \end{bmatrix}, n) = \begin{cases} F(\begin{bmatrix}x \\ y \end{bmatrix}, n), (y \mod 2 = n \mod 2) \\ \text{med}(F(\begin{bmatrix}x \\ y - 1 \end{bmatrix}, n), F(\begin{bmatrix}x \\ y + 1 \end{bmatrix}, n), F(\begin{bmatrix}x \\ y \end{bmatrix}, n - 1)), (\text{otherwise}) \end{cases}$

Figure 7 - Vertical-temporal median filter

VT median filters rely on spatial redundancy to dynamically switch between spatial and temporal interpolation depending on whether there is a high or low amount of motion present in the fields. In the case of no motion, the temporal neighbor is likely to have a high correlation and be picked by the median. However, with high degrees of motion this should no longer hold true, and the vertical neighbors should be chosen, resulting in spatial interpolation.

VT median filters are susceptible to both noise in the input signal and the introduction of vertical artifacts in high motion images. Various methods have been proposed to reduce these drawbacks, including smoothing the image prior to filtering and using a wider range of values for the median\textsuperscript{13,14}.

In figure 8 we find the same scene as before, this time deinterlaced using the 3-tap VT Median filter. The edges of the buildings are made somewhat jagged, as are the edges of the high-motion planes. However all artifacts are minor, part of reason for VT median filtering's enduring popularity.
Edge-Based Line Averaging

Edge based line averaging, proposed by Lee et al in 1994\textsuperscript{15}, makes use of directional correlation to interpolate missing pixels. It is similar to VT filtering in function and computational complexity, but does not constrain itself to only the vertical neighbors of a pixel. Rather, all neighbors are considered to find the most likely direction of an edge, and then those neighbors are used for the interpolation.

Formally, if we wish to interpolate the pixel $F_o\left[\begin{array}{c} x \\ y \end{array} \right], n$, then we will consider its six neighbors, and find the edge direction with the strongest correlation:
\[ E_1 = f\left(\frac{x-1}{y-1}, n-1\right) - f\left(\frac{x+1}{y+1}, n-1\right) \]
\[ E_2 = f\left(\frac{x}{y-1}, n-1\right) - f\left(\frac{x}{y+1}, n-1\right) \]
\[ E_3 = f\left(\frac{x+1}{y-1}, n-1\right) - f\left(\frac{x-1}{y+1}, n-1\right) \]

Figure 9 – Edge line averaging edge detection

ELA then finds the minimum of these three directions, and interpolates \( F_\theta\left(\frac{x}{y}, n\right) \) as the average of the pixels in that direction. So, \( E_{\min} = \min(E_1, E_2, E_3) \), and

\[
F_\theta\left(\frac{x}{y}, n\right) = \begin{cases} 
  f\left(\frac{x}{y}, n\right), (y \mod 2 = n \mod 2) \\
  1/2(f\left(\frac{x-1}{y-1}, n-1\right) + f\left(\frac{x+1}{y+1}, n-1\right), (E_{\min} = E_1) \\
  1/2(f\left(\frac{x}{y-1}, n-1\right) + f\left(\frac{x}{y+1}, n-1\right), (E_{\min} = E_2) \\
  1/2(f\left(\frac{x+1}{y-1}, n-1\right) + f\left(\frac{x-1}{y+1}, n-1\right), (E_{\min} = E_3) 
\end{cases}
\]

Figure 10 – Edge line averaging pixel interpolation

Although ELA in general will perform better than pure VT filtering in reducing the generation of temporal artifacts along horizontal and diagonal edges, it still is susceptible to picking the wrong direction for interpolation, therefore generating a noise pattern. Also, its limited range of a single pixel reflects an intrinsic assumption that objects in the image are large relative to the size of a pixel. In order to reduce these artifacts, Tai et al
proposed\textsuperscript{16} using Sobel filters to more accurately detect edges prior to performing edge line averaging. However, the standard ELA algorithm is still widely used due to simple computations and easy implementation in hardware, and has been modified and extended in a variety of algorithms\textsuperscript{17,18}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure11.png}
\caption{Edge line averaged image}
\end{figure}

The edge line average image can be seen in figure 11. ELA has detected the diagonal vertical edges and performed appropriate interpolation, cleanly interpolated the lines on the planes and buildings, but introduced artifacts due to bad detection in the flames, and at the edges of the reflections in the glass. In addition, all data from the second field is discarded, leading to a smoothed, less sharp image.
Motion-Compensated Vertical-Temporal Median Filtering

Most modern deinterlacing algorithms make use of some form of motion compensation in performing the deinterlace. Performing a motion-compensated deinterlacing step requires two algorithms, the first for creating the motion vectors $d \left( \begin{bmatrix} x \\ y \end{bmatrix} , n \right)$, and the second using those vectors to perform the deinterlace step.

There are a wide variety of motion estimation tools available, generally divisible into “block-based” and “mesh-based” methods. Early examples of block based methods include those of Kalevo and Haavisto\textsuperscript{19} and Kwon et al\textsuperscript{20}.

For the purposes of comparison here, we use an $n \times n$ block search algorithm for calculating the motion vectors, here choosing $n=4$. The first field is divided into $4 \times 4$ blocks, and this block is compared to neighboring blocks in the second field, up to some maximum search distance $m$, to find the neighboring block with the lowest sum of absolute error, where this neighborhood is given as $-m \leq X_{offset} \leq m + n$ and $-m \leq Y_{offset} \leq m + n$. The formula for SAE is given in figure 12.

\begin{equation*}
SAE(d \left( \begin{bmatrix} X_{offset} \\ Y_{offset} \end{bmatrix} , n \right)) = \sum_{i,j \in \{0,1,2,3\}} \left| F(\begin{bmatrix} x + i \\ y + j \end{bmatrix} , n) - F(\begin{bmatrix} x + i + X_{offset} \\ y + j + Y_{offset} \end{bmatrix} , n - 1) \right|
\end{equation*}

Figure 12 - Block-based sum of absolute error

Note that this search can be quite expensive computationally, as each block requires $O(n^2)$ operations to calculate the sum of absolute error, and there are $O(m^2)$ such block comparisons to make. In the above, we let $n=4$ and $m=3n$, with the search space centered...
on the original coordinates. Once the motion vector is calculated for each block, each pixel within that block is treated as having that motion vector as its motion vector.

Once the motion vectors are calculated, the interpolated pixel is treated as the input to the VT 3-tap median filter above, instead of the temporal neighbor, as follows:

\[ F(x, n) = \begin{cases} F\left(\left[\begin{array}{c} x \\ y \end{array}\right], n\right), (y \mod 2 = n \mod 2) \\
\text{med} \left\{ F\left(\left[\begin{array}{c} x \\ y-1 \end{array}\right], n\right), F\left(\left[\begin{array}{c} x \\ y+1 \end{array}\right], n\right), F\left(\left[\begin{array}{c} x \\ y \end{array}\right] + d\left(\left[\begin{array}{c} x \\ y \end{array}\right], n - 1\right)\right), \right\}, \text{(otherwise)} \end{cases} \]

Figure 13 – Motion compensating vertical-temporal median filter

Although the motion compensated pixel could be used directly, by treating it at an input to the vertical-temporal filter instead errors introduced by faulty motion vectors are minimized. Such a combination of linear and motion-compensating methods was recommended by Nguyen\(^{21}\) and Kovacevik\(^{22}\).

While block-based motion compensation can be quite effective at correcting motion-based errors in successive frames, it is inherently limited by the size of the search range \(m\) above. Any motion of more than \(m/2\) pixels will be undetectable by the block-based detection method. However, as the search space grows polynomially with \(m\), large values of \(m\) are not practical computationally.
In figure 14 the block-based motion compensation, combined with the VT median, corrects most of the motion between the two frames. However, vertical artifacts are introduced in the windows of the building by the median, and the front parts of the plane are still jagged due to the large range of motion of the airplane falling outside of the block-search range.

Motion compensating algorithms are now quite common in the deinterlacing field,$^{23, 24}$ in the next chapter, a new algorithm is proposed that may also be considered motion compensating.
CHAPTER 4

NEW METHOD

MPEG-4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group) and finalized into an international standard, ‘ISO/IEC 14496’ in early 1999. Although a full consideration of the MPEG-4 standard is well outside the scope of this paper, one feature of the MPEG-4 standard, the “Video Object Plane” deserves some consideration, as it is fundamental to the new method proposed for deinterlacing a video stream.

Video Object Planes

Unlike earlier versions of the MPEG video standard, MPEG-4 allows video streams to be divided into hierarchical layers, with the lowest layer being designated the Video Object Plane (VOP) layer. In the simplest case this plane is rectangular, and contains the entire scene for that frame. This construction simplifies backwards compatibility with previous versions of the MPEG standard.

However, Video Object Planes are not limited to being rectangular, nor is a stream limited to a single video object plane. Rather, a scene can be composited of multiple, arbitrarily shaped video object planes, each of which can change in size or position independently. In these cases, a past and future video object plane motion vector is
provided to allow for bidirectional prediction of object motion in the stream. These bidirectionally predicted planes are commonly referred to as B-video object plane.

Although these vectors are primarily meant for prediction to allow greater compression rates, their availability makes them a logical choice for motion estimation in the deinterlacing task. In comparison to the previously mentioned motion-estimation techniques, video object planes have the following advantages:

- Since they are previously given, there is no computational overhead to using them for the deinterlacing task.
- A video object plane may move an arbitrary distance from one frame to the next, and will still be tracked by the deinterlacing algorithm.
- While block or mesh based motion detection can only detect translational transforms, a video object plane might be used to predict scaling transforms as well (this is outside of the scope of this paper, however.)
- Since the video object planes by definition indicate which portions of the scene are moving from one frame to the next, it simplifies the task of switching between the optimal temporal or spatial forms of interpolation.

The main drawback to using video object plane information is that most video sequences do not contain them, or at least do not contain content-significant video object planes. A variety of methods have been suggested for extracting video object planes dynamically. Herein a simplified method is described that may be insufficient for compression purposes, but is sufficient to create video object planes for deinterlacing purposes.
Difference Image Creation

Consider two successive fields of video taken from an interlaced stream. Let us then define the difference image of these two fields as follows:

\[
F_d \left( \begin{bmatrix} x \\ y \end{bmatrix}, n \right) = \left| F \left( \begin{bmatrix} x \\ y \end{bmatrix}, n \right) - F \left( \begin{bmatrix} x \\ y \end{bmatrix}, n + 1 \right) \right|
\]

Figure 15 - Difference image function

As the range of \( F_d \) is equivalent to \( F \), \( F_d \) itself can be treated as an image, where high values of \( F_d \) represent areas of change between fields \( n \) and \( n+1 \), and low values of \( F_d \) represent areas of little change.

For the purposes of deinterlacing, we will actually consider successive odd fields, or successive even fields, and create the difference image from them, rather than the difference image between adjacent fields. For instance, if we are deinterlacing fields \( n \) and \( n+1 \), the difference image will be created from fields \( n \) and \( n+2 \). This serves two purposes: first, adjacent fields are vertically shifted by 1 pixel relative to each other, which introduces a motion artifact where none actually exists. Secondly, the greater temporal distance (1/30\(^{th}\) of a second instead of 1/60\(^{th}\)) increases the likelihood that an objects motion will be sufficiently large to be detected.
Figure 16 - Even field (n)

Figure 17 - The next even field (n+2)

Figure 18 - Difference image of fields n and n+2 (border added)
Canny Edge Detection

Detecting edges in an image is performed by searching for steep gradients in adjacent pixels; that is, areas with large spatial derivatives. Each pixel will have a derivative in both the horizontal and vertical directions which can be defined as follows:

\[
D_h\left[\begin{bmatrix} x \\ y \end{bmatrix}\right] = \frac{F\left[\begin{bmatrix} x+1 \\ y \end{bmatrix}\right] - F\left[\begin{bmatrix} x-1 \\ y \end{bmatrix}\right]}{2}
\]

Figure 19 - Horizontal derivative function

\[
D_v\left[\begin{bmatrix} x \\ y \end{bmatrix}\right] = \frac{F\left[\begin{bmatrix} x \\ y+1 \end{bmatrix}\right] - F\left[\begin{bmatrix} x \\ y-1 \end{bmatrix}\right]}{2}
\]

Figure 20 - Vertical derivative function

Given the above, the magnitude and direction of the gradient are then given by the following:

\[
\left|I\left[\begin{bmatrix} x \\ y \end{bmatrix}\right]\right| = \sqrt{D_h\left[\begin{bmatrix} x \\ y \end{bmatrix}\right]^2 + D_v\left[\begin{bmatrix} x \\ y \end{bmatrix}\right]^2}
\]

Figure 21 - Gradient magnitude function
\[ \theta = \arctan \left( \frac{x}{y} \right) \]

\[ D_x \begin{bmatrix} x \\ y \end{bmatrix} \]

\[ D_y \begin{bmatrix} x \\ y \end{bmatrix} \]

Figure 22 - Gradient direction function

Figure 23 - Gradients of the difference image (gamma adjusted and border added)

The above process when performed on an image will find the horizontal and vertical edges, but often these edges will be of variable width depending on image content and contrast. For our purposes we are interested only in the maximal point of the edge, and wish to suppress the rest of the detected edge pixels. While this can be calculated using the magnitude and direction above, the costly square-root and arctangent functions can be avoided by using the signs and relationship between the horizontal and vertical components of the gradient. This will not give an exact answer, but will give a sufficient one since the pixels are discrete and not continuous.
\[ |D_y| > |D_x| \land D_x > 0 \land D_y > 0 \rightarrow 45^\circ \leq \theta \leq 0^\circ \]
\[ |D_y| > |D_x| \land D_x > 0 \land D_y > 0 \rightarrow 90^\circ \leq \theta \leq 45^\circ \]
\[ |D_y| > |D_x| \land D_x < 0 \land D_y > 0 \rightarrow 135^\circ \leq \theta \leq 90^\circ \]
\[ |D_y| > |D_x| \land D_x < 0 \land D_y > 0 \rightarrow 180^\circ \leq \theta \leq 135^\circ \]
\[ |D_y| > |D_x| \land D_x < 0 \land D_y < 0 \rightarrow 225^\circ \leq \theta \leq 180^\circ \]
\[ |D_y| > |D_x| \land D_x < 0 \land D_y < 0 \rightarrow 270^\circ \leq \theta \leq 225^\circ \]
\[ |D_y| > |D_x| \land D_x > 0 \land D_y < 0 \rightarrow 315^\circ \leq \theta \leq 270^\circ \]
\[ |D_y| > |D_x| \land D_x > 0 \land D_y < 0 \rightarrow 360^\circ \leq \theta \leq 315^\circ \]

**Figure 24 - Gradient direction estimation**

Once we have this estimation of the direction of the gradient of a given pixel, we can then detect whether it has a local maximum magnitude. In order to do so, let us consider a function \( p \) to estimate a gradient’s magnitude (more cheaply than the square-root method above):

\[ p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = |D_x\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)| + |D_y\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)| \]

**Figure 25 - Gradient magnitude estimation**

Since a pixel has 8 neighbors, the above estimation will predict that each end of the gradient lies between a set of two of these neighbors. Let’s then define values \( P_a \) and \( P_b \) as follows:
\begin{align*}
45^\circ & \geq \theta \geq 0^\circ \lor 225^\circ \geq \theta \geq 180^\circ \rightarrow P_a &= \frac{p(x+1,y) + p(x+1,y-1)}{2}, P_b = \frac{p(x-1,y) + p(x-1,y+1)}{2}, \\
90^\circ & \geq \theta \geq 45^\circ \lor 270^\circ \geq \theta \geq 225^\circ \rightarrow P_a = \frac{p(x,y-1) + p(x-1,y-1)}{2}, P_b = \frac{p(x,y+1) + p(x-1,y+1)}{2}, \\
135^\circ & \geq \theta \geq 90^\circ \lor 315^\circ \geq \theta \geq 270^\circ \rightarrow P_a = \frac{p(x,y) + p(x-1,y)}{2}, P_b = \frac{p(x-1,y+1) + p(x+1,y+1)}{2}, \\
180^\circ & \geq \theta \geq 135^\circ \lor 360^\circ \geq \theta \geq 315^\circ \rightarrow P_a = \frac{p(x-1,y) + p(x-1,y+1)}{2}, P_b = \frac{p(x+1,y) + p(x+1,y+1)}{2}.
\end{align*}

Figure 26 - Calculation of $P_a$ and $P_b$.

Figure 27 - Non-maximal suppressed gradients (gamma adjusted and border added)

If $p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) > P_a \land p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) > P_b$, then that gradient is a local maxima, and the pixel at that location is retained as a candidate to be an edge pixel. If not, it is eliminated from consideration.
While the pixels remaining at this point might simply be accepted as the final set of edge pixels, this would leave an unacceptable amount of noise in the image. A simple method of reducing noise would be to define a threshold value and simply discard all pixels with a $p$-value below the threshold. However, to do so runs the risk of accidentally deleting important pixels should the threshold be set too high, or leaving an unacceptable amount of noise if the value is set too low. To deal with this problem, we apply a process here termed *hysteresis*, which works as follows: define two thresholds, $T_{\text{high}}$ and $T_{\text{low}}$. All pixels with a $p$-value above $T_{\text{high}}$ are immediately retained. All with values below $T_{\text{low}}$ are immediately discarded. For all other pixels, a pixel is retained if it has at least one neighbor which is retained, and discarded if all its neighboring pixels are discarded. Generally speaking, a path starts at a value over $T_{\text{high}}$, and continues as long as there are contiguous pixels above $T_{\text{low}}$. It should be noted that the values of $T_{\text{high}}$ and $T_{\text{low}}$ are variable, and should be chosen with respect to the type and content of the image in question.

![Figure 28 - Gradient image, post-hysteresis (border added)](image)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Once hysteresis has been performed, the remaining edge pixels can be set to black (equivalent to performing a classical thresholding operation with a threshold of one) to make the edges easier to see and more simple to compare (boolean logic instead of integer arithmetic).

Video Object Plane Bounding Box Extraction

So let us now examine the edge extracted image of the difference image. In the case of a static image, it will be completely white. Otherwise, there will generally be a set of parallel lines describing the motion of the objects in the scene. It is these moving objects we wish to concentrate upon, so the following algorithm is proposed for creating a set of “bounding boxes” around the moving parts of the image, given the edge extracted difference image as input.

Let us define the “capture distance” of a given bounding box to be the distance, outside of the box, at which if a white pixel exists the box will grow to “capture” the pixel. This capture distance essentially represents the size of a white “gap” we are willing to cross in order to include another pixel within our bounding box. As a bounding box gets larger, it becomes more likely that nearby pixels should also be considered part of the same image. However, this capture distance should never shrink below 1 (which would prevent further capture) or grow larger than some maximum value M, which would cause the entire scene to be considered part of one object. Here, we define the capture distance as:
\[ C_x = \min(\max(1, \frac{\text{width}}{2}), M_x) \]
\[ C_y = \min(\max(1, \frac{\text{height}}{2}), M_y) \]

Figure 29 - Bounding box capture distance

Therefore a pixel at coordinates \(x,y\) is within the capture area of a bounding box if and only if:

\[ \text{Box}_{\text{left}} - C_x \leq x \leq \text{Box}_{\text{right}} + C_x \land \text{Box}_{\text{top}} - C_y \leq y \leq \text{Box}_{\text{bottom}} + C_y \]

Figure 30 - Bounding box capture area

Given this definition of capture distance, we create a set of bounding boxes on the image as follows:

1. Start with an empty set of bounding boxes
2. For each white pixel in the image:
   a. If it is within the capture area of any of the bounding boxes in the set, that box captures the pixel
   b. Otherwise, a new bounding box is created containing only that pixel, and is added to the set

The above procedure will end with every black pixel within at least one bounding box. However, the set may contain overlapping boxes. Two boxes are considered overlapping if any of the four corner coordinates of one box are within the capture area of another box. Any pair of boxes so overlapped should then be collapsed into one box,
with the new box defined as having the minimum top and left coordinates, and maximum bottom and right coordinates, of the original boxes. This procedure is repeated until there are no overlapping boxes in the set.

Finally, there generally will be some size threshold below which we consider motion not worth compensating. Any bounding boxes within the set whose area is less than this threshold are removed.

Figure 31 - Video object plane bounding boxes (border added)

At the end of this process, we now have a set of boxes defining where motion occurred between the two fields. We will treat these boxes as our video object plane for the purpose of deinterlacing.

Video Object Plane Based Deinterlacing

Now let us turn to the deinterlacing of the two successive fields. First, we note that in still images, there are no artifacts introduced by the temporal difference between
successive fields; therefore in still images a perfect deinterlace is achieved by simple field insertion. As the extracted video object plane data allows us to segment an image into static and non-static regions, we take advantage of this as follows:

\[
F_s(x, y, n) = \begin{cases} 
F_s[x, y, n], (n \mod 2 = y \mod 2) \\
F_s[x, y, n+1], (n + 1 \mod 2 = y \mod 2) \\
F_s[x, y, n] + d[x, y, n], (n + 1 \mod 2 = y \mod 2)
\end{cases}
\]

This function is similar to any motion-compensated deinterlace method, and differs only in how \( d[x, y, n] \) is calculated (using video object plane, shown below), and in how a given pixel is marked as either moving or static (moving if within any of the bounding boxes generated above, static otherwise). Since a static image has a purely white difference image, no video object planes will be found, and the entire image will be created using field insertion, which is ideal for that case.

Calculating \( d[x, y, n] \) is done as follows:

1. For each bounding box created in the previous step, a sub image is extracted from each of the fields. Although the second image is vertically shifted by 1 relative to the first, this difference will be transparently removed by the motion detector (it will appear as an increment to the y component of the motion vector)
2. These sub images are then converted to an edge extracted image, using the same process as was used on the difference image. This results in both sub images being black and white images, with the black pixels making single-pixel lines representing edges in the original image.

Given these two images, one can be subtracted from the other using the same method as shown above for creating difference images (in fact, as both are black and white images, the subtraction can be performed by the more simple XOR operation). Any pixels left black after this subtraction represent lines present in one image but not the other. The count of black pixels remaining after subtraction gives a simple integer approximation of the difference between the two sub image's alignments.

![Figure 33 - Even field edge image (border added)](image)

We now perform a search similar to the block-based search to find the offset of sub image 1 and sub image 2 such that this difference is minimized. As with block-based searching, the size of the search space is variable. For our purposes, since we know that the object represented within the video object plane is fully within the bounding box in
both images, there is no need to search extensively beyond the border of the video object plane to find a best matching. In addition to the size of the video object plane being related to the size of the object, it is related to the size of the motion vector, therefore the search space is linearly related to the size of the video object plane as well.

Let the range of $m_x$ then be $1/16$th the width of the bounding box (video object plane), and $m_y$ be $1/16$th the height of the bounding box. Then $-m_x \leq X_{offset} \leq m_x + w$ and $-m_y \leq Y_{offset} \leq m_y + h$, and the $d(x, y, n)$ is chosen with the minimum SAE, as defined in figure 35. Note that as all pixels are either white or black, there error for each pixel can be moved to $[0,1]$ by dividing by 255 without loss of generality.

$$SAE(d(x_{offset}, y_{offset}, n) = \sum_{i,j \in \{0,1,2,3\}} |f(x+i, y+j, n) - f(x+i + X_{offset}, y+j + Y_{offset}, n-1)|$$

Figure 34 - Odd field edge image (border added)

Figure 35 - Video object plane motion vector
This motion vector is treated as the correct motion vector for all pixels lying within the bounding box of the given video object plane, and is used in the motion-corrected deinterlace step shown above.

\[
F_{\alpha}(\begin{bmatrix} x \\ y \end{bmatrix}, n) = \begin{cases} 
F(\begin{bmatrix} x \\ y \end{bmatrix}, n), & (y \mod 2 = n \mod 2) \\
\text{med}(F(\begin{bmatrix} x \\ y-1 \end{bmatrix}, n), F(\begin{bmatrix} x \\ y+1 \end{bmatrix}, n), F(\begin{bmatrix} x \\ y \end{bmatrix}) + m(\begin{bmatrix} x \\ y \end{bmatrix}, n), n-1), & (\text{otherwise})
\end{cases}
\]

**Figure 36 - Moving-area video object plane deinterlace function**

**Figure 37 - Video object plane deinterlaced frame**
Similar to the block based motion compensation in chapter three, a bad motion vector, or a subsection of a video object plane that did not move in harmony with the rest of the video object plane, can cause a given pixel to be poorly interpolated. Therefore, we here combine the above motion-compensated deinterlace process with a vertical-temporal three-point median filter, which will tend to remove extremely bad pixels. This leads to the final deinterlace function within moving areas, shown in figure 36. The output of this function, as applied to the previous fields, can be seen in figure 37.
CHAPTER 5

COMPARATIVE RESULTS

Comparisons of deinterlacing methods are always difficult due to the subjective nature of the results, and the difficulty in designing a method that is at the same time objective and representative of a typical deinterlacing task. The graphs below represent the RSME even-to-odd interpolation comparison values of the various algorithms for three cases: the first, a high motion scene (the plane scene from the rest of the paper). The second is a low motion scene (a close-up of a woman’s head, talking.) The last is a single, unmoving frame.

Figure 38 - High-motion deinterlacing comparison
Since field insertion does not vary depending on which field is being interpolated, this measure would be valueless for it, and therefore is left out of the comparisons.

![Graph showing comparison of deinterlacing methods]

**Figure 39 - Low-motion deinterlacing comparison**

As this data shows, video object plane deinterlacing compares favorably with the other deinterlacing methods implemented using this measure. Particularly worthy of note is the zero value for the still images, which indicates that video object plane deinterlacing is competitive with field insertion for still images, while not exhibiting field insertion’s tearing effect on high motion sequences.
Figure 40 - Still frame test

Table 1  Summary RSME of compared deinterlacing functions

<table>
<thead>
<tr>
<th></th>
<th>High Motion Average RMSE</th>
<th>Low Motion Average RMSE</th>
<th>Still Image RMSE</th>
<th>Average RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line Averaging</td>
<td>13.05</td>
<td>6.48</td>
<td>34.76</td>
<td>18.10</td>
</tr>
<tr>
<td>VT Interpolation</td>
<td>8.29</td>
<td>4.50</td>
<td>18.87</td>
<td>10.55</td>
</tr>
<tr>
<td>VT 3Tap Median</td>
<td>9.62</td>
<td>5.00</td>
<td>23.88</td>
<td>12.83</td>
</tr>
<tr>
<td>Edge Line Averaging</td>
<td>13.46</td>
<td>6.72</td>
<td>38.96</td>
<td>19.71</td>
</tr>
<tr>
<td>Motion-Compensation</td>
<td>14.60</td>
<td>6.72</td>
<td>32.78</td>
<td>18.04</td>
</tr>
<tr>
<td>Video Object Planes</td>
<td>7.40</td>
<td>4.37</td>
<td>0.00</td>
<td>3.92</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
There is still no single deinterlacing scheme that is a perfect fit for all video streams and situations. However, the use of video object planes as a means of calculating motion vectors within a video stream has the following advantages:

• Unlike block based motion compensation, video object plane compensation can correct for an arbitrary amount of motion.

• If the video object plane can be trusted completely (which can be done when they are given, but not when calculated as above), the motion vector can be calculated perfectly. In addition, the changing shape of the video object plane can be used to perform both a translation and a scaling operation prior to deinterlacing.

• If the video object plane is given beforehand, the processing necessary to do a motion compensated deinterlacing is minimal.

• Since only the moving parts of the image are compensated, any searching necessary to perform the motion compensation is limited to a smaller area than in full block-based or mesh-based searching.

However, the video object plane approach is not without its disadvantages as well:

• Video object plane based motion compensation still will be adversely affected by any rotation of an object.
- If the video object plane is estimated and not given, very large Video Object Planes may lead to computationally expensive searches for the motion vector.
- In scenes where the camera moves, this may appear to the video object plane detector as a whole-scene move, limiting video object plane’s effectiveness.

Clearly, in videos where video object plane’s are not previously included by other means, the effectiveness of video object plane as a predictor of motion is highly dependant on the effectiveness of the video object plane extraction algorithm. However, with the algorithms here presented, video object planes may be effectively used as an aid to clear deinterlacing.
REFERENCES


VITA

Graduate College
University of Nevada, Las Vegas

Christopher Weiss

Home Address:
3216 Weber Court
Las Vegas, Nevada 89121

Degree:
Bachelor of Science, Computer Science, 1997
University of Nevada, Las Vegas

Thesis Title:
An Improved Algorithm for Deinterlacing Video Streams

Thesis Examination Committee:
Chairperson, Dr. Evangelos Yfantis, Ph. D.
Committee Member, Dr. John Minor, Ph. D.
Committee Member, Dr. Ajoy Datta, Ph. D.
Graduate Faculty Representative, Dr. Derrick Dubose, Ph. D.