De-interlacing of Video Images Using a Shortest Path Technique

Changming Sun

CSIRO Mathematical and Information Sciences
Locked Bag 17, North Ryde, NSW 1670, AUSTRALIA
changming.sun@cmis.csiro.au

Abstract—**This paper presents a fast algorithm for de-interlacing of video images using a shortest path technique. The algorithm applies dynamic programming techniques to find a shortest path in a cost matrix. The motion information obtained from this shortest path is used to re-align the fields of a video image. By using the shortest path via dynamic programming, the motion information estimated is more reliable than simply performing a search in a local neighbourhood. A variety of real images have been tested, and good results have been obtained.**

Keywords—**De-interlacing; Shortest path; Dynamic programming; Image matching; Image motion.**

I. Introduction

Television pictures are broadcast in an interlaced format which has the purpose of reducing both bandwidth required and large area flicker. De-interlacing means that given an interlaced image sequence we would like to create a new sequence of progressive scan images that has the same total vertical resolution and a frame rate equal to the field rate of the original sequence [1].

Haavisto *et al* discussed the problems related to scan rate up conversion and motion detection, and presented an algorithm that adapts to the motion in the picture and eliminates most of the artifacts caused by imperfect motion detection [2]. The algorithm is based on a weighted median filter structure and a simple motion detector. Kwon *et al* described a motion-adaptive de-interlacing method that interpolates the missing lines of interlaced fields from pixels of the same field or adjacent fields after motion compensation [3]. van Someren studied the use of neural networks for the de-interlacing video images where the training set was from a source of progressively scanned video [1]. Otte and Nagel defined modified Gaussian filters to estimate high order gray value derivatives directly from interlaced and non-interlaced images for the estimation of optical flows [4]. Haan and Bellers proposed a de-interlacing algorithm which applies motion estimation and compensation techniques to achieve a high performance for moving and stationary image parts [5]. Han *et al* presented a motion adaptive de-interlacing algorithm based on the brightness profile pattern difference [6]. Haan described an integrated circuit (IC) for motion compensated television format conversion. The technique that he presented includes progress in motion estimation, motion compensation, de-interlacing and noise reduction [7]. Sugiyama and Nakamura proposed a method of de-interlacing using motion compensated interpolation [8]. Rzeszewski presented an algorithm for a special case of a video pixel multiplier for improving low-resolution video on higher-resolution displays [9].

In this paper we use shortest path techniques for motion measurement and use these motion information to perform de-interlacing of video images. The shortest path extraction is implemented using dynamic programming techniques. The motion measurements for all the pixels on each horizontal scanline are obtained at the same time rather than separately. The rest of the paper is organised as follows: Section II gives a brief outline of the process for de-interlacing. Section III describes the process of building the cost matrix for motion estimation. Section IV presents our approach of using dynamic programming technique for motion estimation from the cost matrix. Section V summarizes the steps of our algorithms for video image de-interlacing. Section VI shows the experimental results obtained using our method. Section VII gives concluding remarks.

II. Interlace to Progressive

In interlaced to progressive conversion the task is to interpolate the missing lines in every received field. Line averaging is a spatial operation and results in a

Contributed Paper
Manuscript received March 6, 2001

loss of resolution with non-moving scenes. Figure 1 illustrates the scanlines of three fields for a video sequence. We carry out scan rate up conversion by using neighbouring fields of a video sequence. The dotted scanlines in a field need to be filled in by a corresponding scanline in the following field. For instance, we can use the pixel information along "L2" in "Even Field 1" to fill the scanline number 2 in "Odd Field 1". But because there may be object motion for pixels on "L2", the scanline "L2" in "Even Field 1" needs to be resampled using motion information before it can be inserted into "Odd Field 1".

When we carry out the scan rate up conversion for "Odd Field 1", the dotted lines will be filled in using the information from the even lines in "Even Field 1". When we carry out the scan rate up conversion for "Even Field 1", the dotted lines will be filled in using the information from the odd lines in "Odd Field 2".

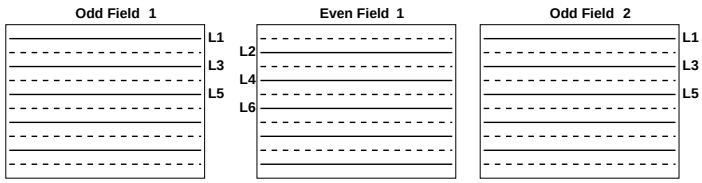


Fig. 1. *Illustration of the interlaced image.*

Figure 2 illustrates the different signal shapes for the neighbouring lines in the neighbouring fields due to object motion. The relative motion for pixels along these two lines needs to be estimated for our de-interlacing purpose.

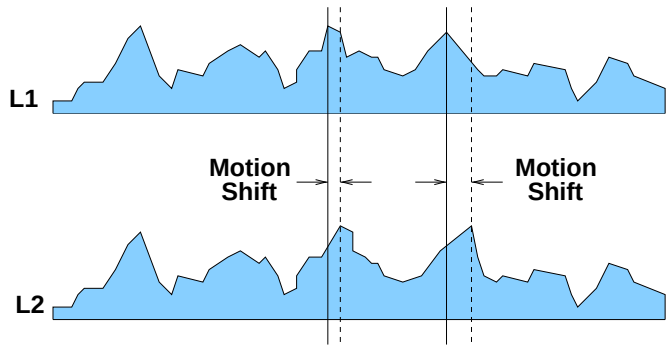


Fig. 2. *Video intensities along two neighbouring lines. The motion shift indicates the cause of the interlacing effect.*

III. Build Cost Matrix

We build a cost matrix from the neighbouring scanlines from two neighbouring fields for the estimation of the motion information; and this motion information will be used for de-interlacing purpose. Assuming we have two lines of pixels "L1" and "L2" as shown at the top of Figure 3. The pixel values for "L1" are indicated as f_i while the pixel values for "L2" are indicated as g_i $(1 \leqslant i \leqslant N)$. The cost matrix that needs to be built has the same length with the length of the scanlines and has a width of R. R is determined by the possible largest motion. If the largest motion in the x-direction is D_{max}, then R can be set to $R = 2D_{max} + 1$.

The following factors need to be considered when one chooses the size of local search. Using a large search range: 1) it is more certain to contain correct match; 2) it has the possibility to include confusing matches; 3) it has a higher computation cost; and 4) it needs a larger memory storage space. On the other hand, a smaller search range: 1) may miss the correct match; 2) has less possibility to include confusing matches; 3) has lower computation cost; and 4) needs a smaller memory storage space.

The illustration in Figure 3 assumes that $D_{max} = 2$ and $R = 5$. For each pixel on line "L1", we calculate some costs for those pixels which are within the search range of R on line "L2". These costs are stored in the cost matrix. For example, for pixel "f3" on "L1", we can calculate the pixel intensity value differences between "f3" and each of "g1", "g2", "g3", "g4" and "g5". The obtained costs "c1", "c2", "c3", "c4" and "c5" for "f3" are stored in the cost matrix in the third column. The formula for calculating the cost is:

$$C(i,j) = |f_{(i)} - g_{(i-j+D_{max})}| \qquad (1)$$

where $1 \leqslant i \leqslant N$ and $1 \leqslant j \leqslant R$. For those pixels that are close to the left or right edges of the scanline, some elements of the cost matrix cannot be calculated because the subscripts $i-j+D_{max}$ of g is out of range. Those elements of the cost matrix are shown as white squares in Figure 3, and they are assigned to a large value.

If we only use one scanline from each field (e.g. "L1" and "L2") for obtaining the cost matrix and from this cost matrix we estimate the motion information, the inserted scanline from the next field ("L2") after motion correction will be aligned to "L1". The result of this will be very much similar to the result obtained by simple line doubling. In our approach, we use two neighbouring lines in one field ("L1" and "L3") and one line ("L2") from the next field for obtaining the cost matrix. We take the sum of the two cost matrixes (one from "L1" and "L2", and one from "L3" and "L2") as our final cost matrix which is then used for motion estimation.

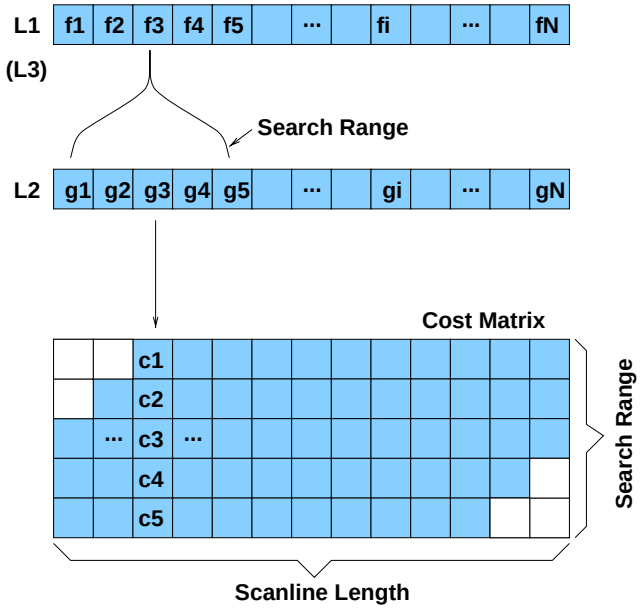


Fig. 3. *Cost matrix calculation from neighbouring scanlines. The top of the figure shows two scanlines from which the cost matrix are obtained.*

Special cares need to be taken for the top and the bottom scanlines of each field. For these lines we can only use one line from one field and one line from the next field for calculating the cost matrix.

Figure 4(a) gives an example of the cost matrix obtained for neighbouring lines. This matrix is used for motion estimation using shortest path extraction technique to be presented in the next section.

IV. Horizontal Image Motion Using Dynamic Programming

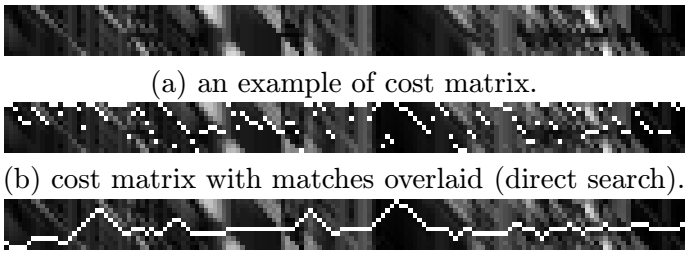
From the obtained 2D cost matrix, a simple method for horizontal motions estimation is using direct search. At each horizontal position of the cost matrix, a position which gives the minimum value for this column is chosen as the motion estimation. The white pixels shown in Figure 4(b) are related to the estimated motion. It can be seen from the positions of these white pixels that neighbouring points may have very different motion estimates.

In this section, we seek to find a shortest path from left to right in the cost matrix for motion estimation so that motions for neighbouring points are consistent with each other. The shortest path is defined as a path from left side of the cost matrix to the right side of the matrix such that the sum of the values on the path is minimised. Most algorithms or applications in the graph framework uses the Dijkstra algorithm for shortest path extraction [10]. We use dynamic programming technique for finding the shortest path because it is computationally efficient on a regular matrix. Table I gives the running times of the Dijkstra-like and dynamic programming algorithms on different sized test images for ordinary shortest path extraction. The tests were run on a 400MHz SUN Enterprise E3500 running Solaris 2.7. It can be seem from the table that the dynamic programming algorithms are generally about five times faster than the Dijkstra-like algorithms if implemented on an image (a regular grid). The Dijkstra-like algorithm used are obtained from the SPLIB which was developed by Boris Cherkassky, Andrew Goldberg, and Tomasz Radzik [10].

TABLE I

Running times of shortest path extraction using Dijkstra-like and Dynamic Programming algorithms on images.

Image Size	Running Time	
	Dijkstra	DP
240×285	0.05 s	0.01 s
512×512	0.26 s	0.05 s



(a) an example of cost matrix.

(b) cost matrix with matches overlaid (direct search).

(c) cost matrix with shortest path overlaid (DP).

Fig. 4. *(a) An example of the cost matrix. (b) the shortest path obtained by using the direct search method applied to (a). (c) the shortest path obtained by using the dynamic programming technique applied to (a).*

Currently only the pixel difference is used. Further work can be carried out by using local windows technique to calculate the difference.

The best path from left to right through the cost matrix can be found by using a dynamic programming technique [11], [12], [13]. The best path gives the minimum sum of the cost values along the path which satisfies certain connectivity and smoothness constraints.

Now we describe the algorithm for the shortest-path extraction in the 2D cost matrix obtained in the previous section using efficient dynamic programming techniques. For $1 \leqslant i \leqslant N$ and $1 \leqslant j \leqslant R$, let $C(i,j)$ be the cost of the (i,j)th value in the 2D matrix of size

NR. The cost of a path P is defined as the sum of the costs along the path. We maintain two arrays for the dynamic programming. Array $Y(i,j)$ contains the accumulated values and $K(i,j)$ has the position which produces the local minimum value. When $i = 1$,

$$Y(1,j) = C(1,j) \tag{2}$$

i.e. the first column of Y is a copy of the first column of C. For the remaining columns (ith column) of the matrix, the Y values at each position is obtained using the following recursion:

$$Y(i,j) = C(i,j) + \min_{t:|t|\leqslant 1} Y(i-1, j+t) \tag{3}$$

The values of t which achieves the minimum in Eq. (3) during each iteration is stored in K.

$$K(i,j) = \text{argmin}_{t:|t|\leqslant 1} Y(i-1, j+t) \tag{4}$$

The values stored in the matrix K are the position information that the path should follows, and this matrix is used to back track along the best path from the minimum value in the last column of Y. After the matrix Y and K have been obtained, we can start the back tracking process to obtain the shortest path. The backtracking process begins from the location of the minimum value in the last column of Y. The possible values of t given in Equations (3) and (4) are -1, 0 and 1. The possible t values are related to the search positions of the shortest path. As shown in Figure 5, if a path arrives at point "A", the possible positions that this path can go through the next column are the three positions "A1", "A2" and "A3". If a path has arrived at a boundary point "B", then only two possible positions ("B1" and "B2") that the path can go through the next column.

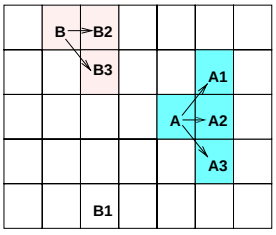


Fig. 5. *The search positions for the shortest path. This constrains how the path should go in the matrix.*

Dynamic programming is used here for motion estimation because it is optimal for the whole horizontal line. The motion estimated for the whole scanline are consistent with each other. This means that neighbouring pixels tend to have similar motions. Dynamic programming technique is also computationally efficient in obtaining optimised results.

A summary of the steps of the dynamic programming algorithm are:
1. Read in the cost matrix C.
2. Allocate two working arrays Y and K for storing the accumulated cost values, and the local current path positions.
3. Carry out the recursions for obtaining Y and K using Equations (3) and (4).
4. Find out the position on the last column of Y which contains the smallest value.
5. Backtrack through the cost matrix from this position to find the whole path.

Figure 4(c) shows the path obtained using the dynamic programming technique described from the cost matrix shown in Figure 4(a). The position of each point on the path relative to the horizontal line through the center of the cost matrix is the motion information of this pixel. Comparing Figure 4(c) and Figure 4(b), we can see that the motion estimates from dynamic programming technique are more consistent within a scanline.

The accuracy of the obtained motion estimation after the dynamic programming step is up to 1 pixel. Sub-pixel accuracy can be obtained by fitting a second degree curve to the values in the neighbourhood of the obtained path position and the extrema of the curve can be obtained analytically. The general form of the second degree curve (parabola) is: $f(x) = a + b \cdot x + c \cdot x^2$. The minimum can be found where the slope is zero in the quadratic equation. The position of this sub-pixel can be found at $x = -b/2c$. If only three points of the values in the cost matrix near the initial position are used, e.g. the points $j-1, j, j+1$, the sub-pixel position of the disparity can be calculated using the following formula [14]:

$$x = j + \frac{1}{2} \times \frac{C(i,j-1) - C(i,j+1)}{C(i,j-1) - 2C(i,j) + C(i,j+1)} \tag{5}$$

where $C(i,j)$ is the value in the cost matrix at position i,j, and x is the sub-pixel disparity position obtained.

After obtaining the motion information among the neighbouring lines from two fields, one of the scanline need to be transformed or resampled using such motion information so that it is aligned with the matching scanlines in the other field.

V. Algorithm Steps

The steps of our de-interlacing algorithm for interlaced images are:

1. Build the cost matrix for neighbouring scanlines from neighbouring fields. Special case for the top and bottom lines.
2. Perform the shortest path extraction on the cost matrix for motion estimation.
3. Use the motion information obtained to align the neighbouring scanlines.

VI. Experiment Results

This section shows some of the results obtained using the method described in this paper. A variety of real images have been tested.

Figure 6 shows an example image illustrating the improvement of the image quality after applying the algorithms described in this paper. The image in Figure 6(a) shows a subregion in the original interlaced input image which is direct merging of the odd and the even fields of a frame. The image in Figure 6(b) gives the result of merging the two fields in a frame after motion estimation and de-interlacing using our algorithm. It is clear from the images that the zig-zag effect shown in Figure 6(a) due to the motion of the camera has greatly been reduced or even removed, especially on the objects such as the light pole and the tree branches.

Note that the technique described in this paper only estimates the horizontal motion along scanlines. The algorithm seems to be effective for de-interlacing purposes in scan rate up conversion and in removing the zig-zag effect for moving objects or cameras. Further research is necessary to investigate the effect of large motions in vertical direction. In this case, complex motion estimation techniques may be needed.

Table II gives the running times of our algorithm on different sized test images. The computer used is the same as mentioned in Section IV for Table I. The typical running time of the algorithm on a 480×320 image is in the order of 0.15 seconds.

TABLE II
Running times of our algorithm on different sized images.

Image size	**Time spent**
128×128	0.01s
480×320	0.15s
720×486	0.35s

VII. Conclusions

We have developed a fast algorithm for de-interlacing of video images using shortest path technique. The motion information is obtained and used for aligning the neighbouring lines in different fields of video images. Because of the use of shortest path in calculating the motion information, the result obtained is consistent throughout the horizontal line. A variety of real image tests has been carried out and the results show that the presented algorithm is effective for de-interlacing of video images.

(a)

(b)

Fig. 6. *An example region of an image before and after de-interlacing operation. (a) A subregion of the input image with the odd and even fields put together; (b) The same subregion of the input image after applying the de-interlacing algorithm described in this paper using motion information obtained via dynamic programming.*

Acknowledgement

The author is grateful to Dr Geoff Poulton of CSIRO Telecommunication and Industrial Physics, Australia for his comments.

References

[1] N. van Someren, *High Quality De-interlacing of Television Images*. PhD thesis, Trinity College, University of Cambridge, September 1994.

[2] P. Haavisto, Y. Neuvo, and J. Juhola, "Motion adaptive scan rate up-conversion," *Multidimensional Systems and Signal Processing*, vol. 3, pp. 113–130, 1992.

[3] S.-K. Kwon, K.-S. Seo, J.-K. Kim, and Y.-G. Kim, "A motion-adaptive de-interlacing method," *IEEE Transactions on Consumer Electronics*, vol. 38, pp. 145–149, August 1992.

[4] M. Otte and H.-H. Nagel, "Estimation of optical flow based on higher-order spatiotemporal derivatives in interlaced and non-interlaced image sequences," *Artificial Intelligence*, vol. 78, pp. 5–43, October 1995.

[5] G. de Haan and E. B. Bellers, "De-interlacing of video data," *IEEE Transactions on Consumer Electronics*, vol. 43, pp. 819–825, August 1997.

[6] D. Han, C.-Y. Shin, S.-J. Choi, and J.-S. Park, "A motion adaptive 3-D de-interlacing algorithm based on the brightness profile pattern difference," *IEEE Transactions on Consumer Electronics*, vol. 45, pp. 690–697, August 1999.

[7] G. de Haan, "IC for motion compensated de-interlacing, noise reduction, and picture-rate conversion," *IEEE Transactions on Consumer Electronics*, vol. 45, pp. 617–624, August 1999.

[8] K. Sugiyama and H. Nakamura, "A method of de-interlacing with motion compensated interpolation," *IEEE Transactions on Consumer Electronics*, vol. 45, pp. 611–616, August 1999.

[9] T. S. Rzeszewski, "Method for improving low-resolution video on higher-resolution displays," *IEEE Transactions on Consumer Electronics*, vol. 45, pp. 1030–1037, November 1999.

[10] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," *The Macmillan Press Ltd*, vol. 73, pp. 129–174, 1996.

[11] M. Buckley and J. Yang, "Regularised shortest-path extraction," *Pattern Recognition Letters*, vol. 18, no. 7, pp. 621–629, 1997.

[12] G. L. Gimel'farb, V. M. Krot, and M. V. Grigorenko, "Experiments with symmetrized intensity-based dynamic programming algorithms for reconstructing digital terrain model," *International Journal of Imaging Systems and Technology*, vol. 4, pp. 7–21, 1992.

[13] S. A. Lloyd, "A dynamic programming algorithm for binocular stereo vision," *GEC Journal of Research*, vol. 3, no. 1, pp. 18–24, 1985.

[14] P. Anandan, "A computational framework and an algorithm for the measurement of visual motion," *International Journal of Computer Vision*, vol. 2, pp. 283–310, 1989.

Dr *Changming Sun* received his PhD in the area of Computer Vision at Imperial College of Science, Technology and Medicine, London in 1992. Then he joined CSIRO Mathematical and Information Sciences, Australia in December 1992 as Research Scientist, both doing research and working on applied projects. His research interests include video processing, robot vision, image analysis, and photogrammetry. Dr Sun is a member of the Australian Computer Society and the Australian Pattern Recognition Society.