REAL-TIME OPTIMAL-MEMORY IMAGE ROTATION FOR EMBEDDED SYSTEMS

Serene Banerjee, Anjaneyulu Kuchibhotla
HP Labs India, Bangalore, India, 560030
{serene.banerjee, anji}@hp.com

ABSTRACT

Skew-corrected document images are necessary for subsequent downstream operations such as archiving, printing or improving OCR performance. Image rotation is a necessary and more expensive step in achieving skew correction of document images. Other applications of rotation include, image registration and orientation correction. Traditional image rotation algorithms [2-4] such as three-shear rotation [1] require three separable shears of the image. The embedded use of such techniques in scanners/printers presents technical challenges, since the memory available is limited and/or the document image is only available progressively in chunks of say 32 or 64 rows (swaths). Traditional image rotation algorithms require the entire image to be available before commencing the rotation operation. This paper presents an approach that allows image rotation using swaths of the image thus minimizing the overall memory requirement. We theoretically prove that the number of image swaths that are to be buffered is independent of the image size and depends only on the rotation angle. This approach enables rotation of any arbitrary sized image on memory constrained devices. The memory savings realized is at least 80%, for an A4-sized document image rotated 15^o. Our progressive approach demonstrates real-time image rotation and hence improves on the state-of-the-art approaches for reduction of rotation complexity [5-10].

Index Terms— image rotation, optimal-memory, real-time performance, embedded systems

1. INTRODUCTION

It is common to observe document skew, in the output of devices that support scanning/photocopying. This arises while scanning thick documents or from incorrect document placement, or from shifting of the object while closing the scanner lid. Embedded skew correction could help alleviate these problems. Skew correction also enables better Optical Character Recognition (OCR). The corrected document is likely to be more useful for subsequent downstream operations such as archiving, modifying, collaborating, communicating, or printing. Image rotation is a necessary step for skew correction and for geometric manipulation of images. Traditional image rotation, such as forward or backward mapping, is time and memory expensive [1]. Previous work [2-4] has reduced complexity of image rotation to O(MxN), where MxN is the image size, by breaking the rotation operation into three separable shears along the horizontal and vertical directions, respectively. This however requires the entire image to be in memory, which may not be possible in some scanning based products due to memory constraints or requirements for real-time operation.

In addition, in automatic document feeder scanners, the input image comes in chunks of 32 or 64 rows called swaths. In normal course, the entire image could be accumulated from all the swaths before running skew detection and correction algorithms. However, for embedded implementations due to memory constraints it may not be practical to use algorithms that use the entire image for rotation. If required, skew detection could be performed continuously on an image in real-time with the availability of each swath [11]. In these cases, image rotation can commence immediately after a satisfactory skew angle is detected. In copiers and printers, the image is again printed swath-by-swath based on the print head size. So, if the output image after skew correction is formed swath-by-swath, the printing can start immediately by printing the first swath. This reduces the user wait time and facilitates memory-constrained embedded implementation.

This work derives the theoretical optimal-memory requirement for swath-based image rotation and realizes image rotation with a circular buffer. This results in realizing rotation with the minimum memory overhead, that is theoretically proven. Our solution realizes rotation in a progressive manner so that real-time rotation is demonstrated on memory constrained devices, without loss of quality. This improves on other complexity reduction approaches as discussed in the next section and enables embedded implementation. The time complexity of the proposed algorithm is comparable to that of the three-shear based approach [2-4]. The fixed point version was tested on an HP All-in-One (AiO).

2. PREVIOUS RESEARCH

To the best of our knowledge, there is no previous work on image rotation algorithms that can work on swaths of images without requiring the entire image for rotation. There have been a couple of approaches for optimization of image rotation and for ensuring good quality of the rotated image. This section will describe the algorithms and present their advantages

and shortcomings. We will then establish a need for a more memory-optimized algorithm. Owen et al. [5] describes image rotation in the transform domain that reduces visible artifacts in the rotated image. However, the run-time complexity of the algorithm is $O(N^2 \log N)$. Yu et al. [6] propose an optimized algorithm for elimination of floating point operations during image rotation. However, the aspect ratio of the rotated image is not similar to that of the original, and could be a potential threat to document image quality. Chandran et al. [7] use Diophantine methods for approximation of the floating point computations during image rotation, and implement image rotation in fixed point. However, the run-time complexity of their algorithm is $O(N^3)$. Shen et al. [8] implement the three-shear rotation in the discrete cosine transform (DCT) compressed domain and achieve a 2X speedup over the traditional spatial domain approach. All these algorithms need the whole image in memory for image rotation that imposes an overhead that may not be acceptable for embedded implementations. Yeshick et al. [9] and Kothandaraman et al. [10] propose algorithms that rotate images for fixed angles (90^o, 180^o, 270^o), without any memory overhead. But, their approach cannot be generalized for any rotation angle.

In this work, we present an approach which works on swaths of images thus substantially reducing memory requirement. We compute the optimal-memory requirement based on the skew angle, and develop an efficient adaptation of the three-shear rotation using a circular buffer that rotates the image optimally.

3. THEORY FOR PROPOSED SWATH-BASED THREE-SHEAR ROTATION

3.1 Three-shear rotation [2-4]

Rotation is one of the most sophisticated affine transforms. The new coordinates (x, y) of the point being rotated are computed as follows, given the original coordinates (u, v) and the rotation angle Θ:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \quad \ldots\ldots\ldots\ldots \quad (1)$$

The same computation can be achieved by decomposing the rotation matrix into several passes, with each pass performing certain amount of horizontal shearing or vertical shearing. The rotation matrix is equivalent to 3 shear matrices as shown:

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} 1 & -\tan\frac{\theta}{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin\theta & 1 \end{bmatrix} \begin{bmatrix} 1 & -\tan\frac{\theta}{2} \\ 0 & 1 \end{bmatrix} \quad (2)$$

Thus, the image is first sheared horizontally by ($\Theta/2$) degrees, then vertically by Θ degrees, and then back along the horizontal direction by ($\Theta/2$) degrees.

3.2 Proposed adaptation for swath-based images

In normal course the entire image could be accumulated from all the swaths before running skew detection and correction algorithms. In our case the skew detection [11] is performed on a downscaled and binarized version of the input image and rotation is carried out based on swaths of the image. Skew detection can also be performed on an image which is available swath-by-swath if necessary [11]. The minute a satisfactory skew angle is detected, image rotation can commence. In printers and copiers the image is again printed swath-by-swath based on the size of print head. So if the output image after skew correction is formed swath-by-swath, the printing can start immediately from the first swath. This reduces the wait time for the user and optimizes the memory requirement thereby enabling an embedded implementation. This work adapts the three-shear rotation for swath-based input/output (Algorithm 1).

Fig. 1 shows the block diagram of the adaptation of the three-shear algorithm for swath-based input/ output. In our approach just enough input swaths need to be buffered, so that the output swaths will be properly generated. Fig. 2 derives the optimal number of swaths that need to be buffered. Our requirement is that the output image dimensions should be equal to the input to be able to print the same sized documents, as the scanned/copied ones. So, it is necessary to crop the image after the last shear so that the cropped image size is equal to the input image size. So, initially *rows_to_crop* number of rows would not be output anyway, as they would be cropped, where *rows_to_crop* = (*shear3_height* – *image_height*)/2. For a proper output swath, *swath_size* rows need to be ready after *rows_to_crop*. Say *x* swaths are to be buffered before the first output swath is ready. Then from Fig. 2, it can be seen that

$x * Ver\_dist >= swath\_size$, and …… (3)

Ver_dist = difference in rows of the top-left corners of two successive swaths after the second (vertical) shear

= number of valid rows that are filled with data after processing each input swath

$= \{image\_width * \text{abs}(\sin\Theta) + (x+1) * swath\_size * \cos\Theta + 1\} - \{image\_width * \text{abs}(\sin\Theta) + x * swath\_size * \cos\Theta + 1\}$

$= swath\_size * \cos\Theta$ ……. (4)

So, interestingly the minimum number of input swaths that are to be buffered for the first output swath is, ceil(1/cos(Θ)) and thus independent of the image size. In order to store these input swaths we use a circular buffer so that we can optimize and reuse memory. The circular buffer size or the minimum memory required to buffer the second vertical shear output is,

$image\_width * \text{abs}(\sin\Theta) + (\text{ceil}(1/\cos\Theta)) * swath\_size * \cos\Theta + 1$ …… (5)

3.3 Proposed realization using a circular buffer

We implement the intermediate memory required as a circular buffer, whose size was computed in the previous section. The idea is that once an output swath is dispatched/printed, the next processed input swath can be overwritten in that memory in a circular fashion. So, the first horizontal shear and the second vertical shear is processed for each input swath, and

the output of the second vertical shear is stored in the circular buffer. After these two operations, the row number is checked to see if the first output swath is ready (refer to the pseudo code in Algorithm 1). If not, the first and the second steps are repeated for the next set of input swaths. Once we know that the first output swath is ready, the third horizontal shear is performed on the output swath, and it is overwritten on the buffer by the next input swath. The above steps are performed in a pipelined manner, until all the input swaths have been rotated properly. The pseudo-code is given in Algorithm 1.

Allocate memory for circular buffer capable of storing minimum number of swaths
Reset *flag_first_output_ready* = 0
for *each input swath* **do**
  Increment input swath number;
  Perform horizontal shear of swath;
  Perform vertical shear of swath & fill circular buffer;
  if *(the minimum number of rows filled in circular buffer >= rows_to_crop + swath_height) && (flag_first_output_ready = 0)* **do**
    /* Indicate that enough rows are available to generate the first output swath */
    flag_first_output_ready = 1;
  end
  if *flag_first_output_ready == 1* **do**
    Perform 3rd horizontal shear and output swath;
    Increment output swath number;
  end
end
for *output swath number <= input swath number* **do**
  Perform 3rd horizontal shear and output swath;
  Increment *output swath number*;
end

Algorithm 1: Pseudo-code for swath-based rotation

Here it is to be noted that the data fill in rate in the circular buffer is *swath_size**cos(Θ), and the rate at which it is taken out is *swath_size*. As cos(Θ)≤1, after sufficient number of swaths, the data fill in rate could fall behind the data take out rate, and rows could be inadvertently overwritten before they are taken out. To avoid this problem, based on the image dimensions, swath size and the rotation angle, it is possible to check how many times this condition could occur, and increase the circular buffer memory *a priori*, as:

$$image\_width*\text{abs}(\sin\Theta)+(\text{ceil}(1/\cos\Theta))*swath\_size*\cos\Theta+1+RowsToAdd \quad \ldots\ldots\ (6)$$

where *RowsToAdd* rows would be overwritten when the data input rate falls behind the data take out rate.

4. RESULTS

Image rotation was evaluated on 4 sets of 60 images at 75, 100, 150 and 300 dpi, respectively with skews of ±15°. All the images were rotated successfully. A few results are shown in Fig. 3. The rotated images with the proposed algorithm were bit-exact with those rotated with the traditional three-shear rotation. Thus there is no loss of image quality with our proposed optimizations. The size of swaths, though 32 or 64 rows for our case, could vary from as low as 1 row, up to image height. Irrespective of the swath-size the rotated images were bit-exact, ensuring image quality.

As computed above the run-time memory requirement of this algorithm depends on the image width, size of swath and the degree of skew. For swath-size of 32 rows, Fig. 5 plots the memory requirement in MB for rotating a 2550 x 3300 image with skews of ±15°. Storing the whole image in memory would require 25 MB. The memory requirement for our algorithm varies from 0.2MB-5MB for rotation angles of 0 to 15°.

Fig. 6 plots the memory for rotating an image of varying dimensions by 15°. It shows that the run-time memory needed for the proposed algorithm is much less than that of the traditional three-shear rotation. The computational overhead was minimal and a fixed-point implementation has been ported to a HP AiO.

4. CONCLUSIONS

This work adapts the three-shear rotation algorithm for swath-based input/output. We have theoretically derived the optimal-memory requirement, and have demonstrated real-time embedded rotation. On detection of the skew angle [11]; swaths are rotated in place, and made available for printing. This enables real-time embedded skew correction. The proposed approach could reduce complexity for other applications needing rotation, such as image registration and orientation correction.

5. REFERENCES

1. R. C. Gonzalez, R. E. Woods, "Digital Image Processing", *Prentice Hall, 2nd ed*., 2002.
2. A. W. Paeth, "A Fast Algo. for General Raster Rotation", *Proc. Graphics Intf.*, pp. 77-81, May 1986.
3. M. Unser, P. Thevenaz and L. Yaroslavsky, "Convolution-based Interpolation for Fast, High-quality Rotation of Images", *IEEE Trans. On Image Proc.*, vol. 10, no. 4, pp. 1371—1381, Oct. 1995.
4. D. Fraser, "Comparison at High Spatial Frequencies of Two-pass and One-pass Geometric Transformation Algorithms", *Computer Vision, Graphics and Image Processing*, vol. 46, pp. 267—283, 1989.
5. C. B. Owen and F. Makedon, "High Quality Alias Free Image Rotation", *Proc. IEEE Asilomar Conf. on Signals, Sys. and Comp.*, Vol. 1, pp. 115—119, 1996.
6. Z. Yu, J. Dong, Z. Wei and J. Shen, "A Fast Image Rotation Algorithm for Optical Character Recognition on Chinese Documents", *Proc. IEEE Int. Conf. on Comm., Circuits and Sys.*, vol. 1, pp. 485—489, 2006.
7. S. Chandran, A. K. Potty, M. Sohoni, "Fast Image Trans. Using Diophantine Methods", *IEEE Trans. On Image Proc.*, vol. 12, no. 6, pp. 678—684, June 2003.
8. B. Shen and I. K. Sethi, "Scanline Algorithm in Compressed Domain", *SPIE Proc. On Digital Video Comp. Algorithms and Tech.*, vol. 2668, Apr. 1996.
9. W. E. Yeshik, "Image Rotation Circuit", *International Computer Ltd*., Patent No. 4.916,746, Apr. 10, 1990.
10. S. Kothandaraman, J. R. Zbiciak, "Using Super-pixels for Efficient In-place Rotation of Images", *TI*, Patent No.: US 2006/0204130 A1, Sep. 2006.
11. S. Banerjee, S. Noushath, P. Parikh, "Edge detection", *HP*, Patent No.: PCT/IN2009/000325, Jul. 2009.

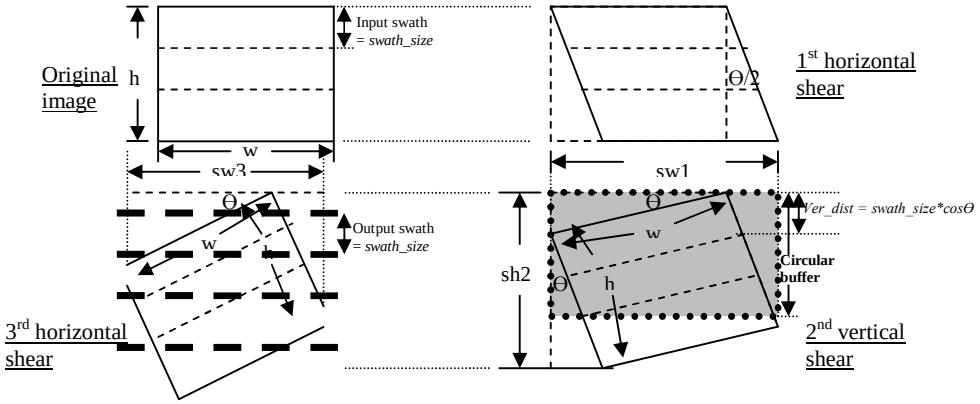

Fig. 1: Adaptation of three-shear based rotation for swath-based input/output

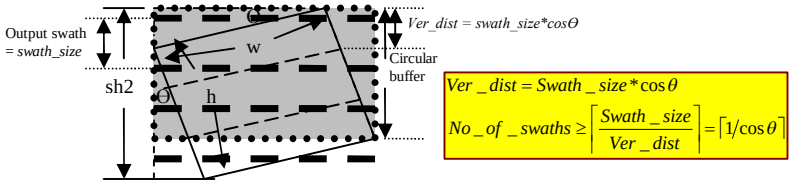

Fig. 2: Computation of the minimum memory requirement for a given image width and degree of skew. The above figure refers to the image after the second (vertical) shear.

Fig. 3: Examples showing skew in scanned documents, and their successful deskewing.

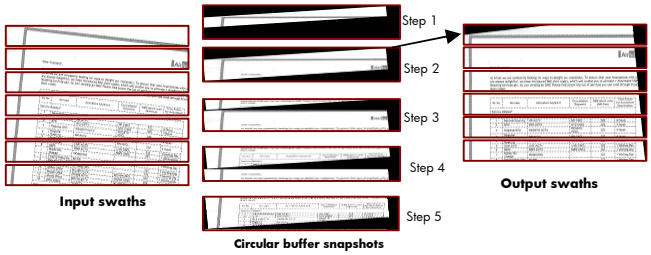

Fig. 4: Example showing buffering of the swaths in the circular buffer during the horizontal shear. Swath size is 64 rows, and skew angle is 6.5 degrees. Two swaths need to be buffered in the circular buffer before the first output is ready. Once the first output is ready, the space can be reused for the subsequent input swaths as shown in Step 3.

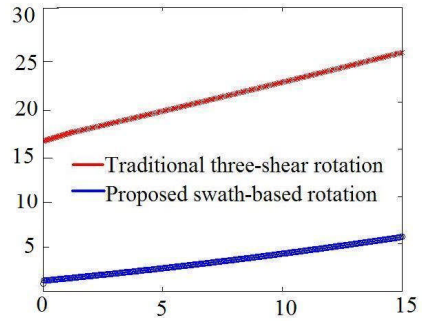

Fig. 5: Savings of run-time memory with varying skew from 0-15^{o}. Image size: 2550x3300 pixels. x-axis: 0-15^{o}, y-axis: memory in MB, assuming integer as 2 bytes

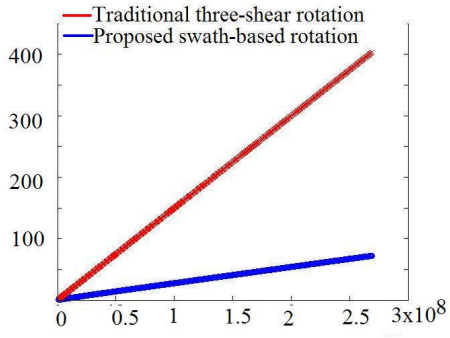

Fig. 6: Savings of run-time memory with varying image width (300-5000 pixels, aspect ratio: 8.5:11) for 15^{0}. x-axis: image size in MB, y-axis: memory in MB